10. EL과 JSTL

10.1 EL (Expression Language)

10.1.1 개요

표현 언어(EL)는 JSTL 1.0 스펙에서 소개되었던 것으로 JSP 2.0 스펙부터 공식적으로 포함되었으며,
JSP 페이지에 사용되는 자바코드를 대신해서
액션태그 엘리먼트(예:<jsp:setProperty>) 의 속성에 값을 지정하는 역할을 한다.

예를 들어 <someTags:aTag> 액션태그 엘리먼트의 속성은 attribute 이고 속성 값은
<%= pageContext.getAttribute("aName") %> 의 경우

```
<someTags:aTag attribute="<%= pageContext.getAttribute("aName") %>">
```

이라고 표현하지만 표현 언어에서는 다음과 같이 나타낸다.

```
<someTags:aTag attribute="${aName }">
```

aCustomer가 자바 빈 객체이고 getAddress()는 자바 빈 객체가 가지고 있는 메서드인 경우

지금까지 사용한 방법 : <%= aCustomer.getAddress() %>
표현 언어에서 사용 : ${aCustomer.address}

표현 언어(EL) 는 표현식(<%= %>) 을 대신하는 효과를 가지며
null 값을 가지는 변수에 대해 좀 더 관대하고, 데이터 형 변환을 자동적으로 해 준다.

JSP 페이지에서 null 값을 가지는 변수인 경우 NullPointerException 예외가 발생하며
객체가 저장되는 곳에 따라 본래의 객체가 Object 형으로 추출되는 경우 사용하기 위해서는
다시 본래의 객체 형으로 형 변환을 해야 하지만 EL은 이들을 자동으로 처리 한다.

■ 표현 언어(EL)의 표현식의 기능
- 변수와 연산자를 포함한다.
- JSP 의 영역(page, request, session, application)에 저장된 속성 및 자바 빈도 표현 언어(EL)의 변수로서 사용될 수 있다.
- 내장 객체도 지원한다.

■ 표현 언어(EL) 작성방법
- 표현 언어(EL) 표현식은 숫자, 문자열, boolean 값과 null 같은 상수 값(리터럴) 들도 포함할 수 있다.
- 표현 언어(EL) 는 $ 와 표현식 그리고 { } 를 사용해서 표현 한다.
- 표현 언어(EL)는 항상 『 ${ 』 로 시작해서 『 } 』로 끝난다.
  - ${num}
  - 표현식은 JSP 스크립트 내부에서 쓸 수 없다. 즉, <% %>, <%! %>, <%= %> 안에는 사용할 수 없다.
- 표현 언어(EL) 표현식 안에 연산식도 작성이 가능 하다.
  - ${num + 1}
  - ${article.num + 1}
    프로퍼티 접근 연산자 닷(dot(.)) 은 표현 언어(EL) 에게 자바 빈이나 컬렉션 객체에서 다음에 오는 이름과 같은 프로퍼티를 찾게 한다.

- ${article['num'] + 1} or ${article["num"] + 1}
  표현 언어(EL) 표현식에는 브라켓 연산자(bracket([ ]) operator) 를 사용해도 된다.
  닷(dot(.)) 과 같다. 즉, 브라켓 연산자의 배열 형태로 객체의 프로퍼트 혹은 변수에 접근 할 수 있다.

10.1.2 표현 언어(EL)에서 사용되는 연산자

■ 연산자
- . → 빈의 프로퍼티나 맵(Map)의 엔트리 접근
- [] → 배열이나 리스트(List) 엘리먼트 접근
- () → 괄호. 표현식의 연산 순서를 바꿔서 연산하게 할 때.
- a?b : c → 조건 테스트 - 조건(a) ? true 일 때 리턴 값(b) : false일 때 리턴 값 (c)
- + → 더하기
- - → 빼기
- * → 곱하기
- / 또는 div → 나누기
- % 또는 mod → 나머지
- == 또는 = → 같다
- != 또는 =! → 다르다
- < 또는 lt → 보다 작다
- > 또는 gt → 보다 크다
- <= 또는 le → 작거나 같다
- >= 또는 ge → 크거나 같다
- && 또는 and → 논리 AND
- || ('|'가 연속으로 두개) 또는 or → 논리 OR
- ! 또는 not → 단항 not (true를 false로 false를 true로)
- empty → 빈 변수 값 체크. null, 빈 문자열, 빈 배열, 엔트리가 없는 맵(Map)이나 컬렉션(Collection) 인가 테스트
- func(args) → 함수 호출. func는 함수 이름이고 args는 인자로 없을 수도 있고, 한개 혹은 쉼표(,)로 분리된 여러 개의 함수 인자를 가질 수 있음

10.1.3 표현 언어(EL)에서 제공되는 내장객체

■ 내장객체
- pageScope : 모든 page 영역 객체들에 대한 컬렉션
- requestScope : 모든 request 영역 객체들에 대한 컬렉션
- sessionScope : 모든 session 영역 객체들에 대한 컬렉션
- applicationScope : 모든 application 영역 객체들에 대한 컬렉션
- param : 모든 request 파라미터들을 문자열로 가진 컬렉션(request.getParameter() 메서드)
- paramValues : 모든 request 파라미터들을 파라미터 당 문자열 배열로 가진 컬렉션
- header : HTTP 요청 헤더를 문자열로 가진 컬렉션
- headerValues : HTTP 요청 헤더들을 헤더 당 문자열 배열로 가진 컬렉션
- cookie : 모든 쿠키의 컬렉션
- initParam : 모든 어플리케이션의 초기화 파라미터의 이름의 컬렉션
- pageContext : 현재 페이지를 위한 javax.servlet.jsp.PageContext

■ 사용 예
- 세션 값 가져오기 : ${sessionScope.userID} → userID 라는 이름으로 설정한 값
- HTTP 파리미터 값 가져오기 : ${param.userID} → userID 라는 이름으로 설정한 값

10.1.4 자바 클래스 메서드에 접근

■ 표현 언어(EL)에서 자바 클래스의 메서드에 접근하는 방법

① 자바의 클래스 파일을 작성한다.
② 클래스 파일을 작성하고 나면 태그라이브러리에 대한 설정정보를 담고 있는 TLD 파일을 작성한다.
③ web.xml 파일에 TLD 파일을 사용할 수 있는 설정 정보를 추가한다.
④ 자바클래스의 메서드에 접근하는 JSP 페이지를 작성한다.

■ 표현 언어(EL)에서 자바 클래스의 메서드에 접근하는 방법
자바 클래스 파일 ↔ TLD파일 ↔ web.xml ↔ JSP 페이지

■ 사용 예
1) 클래스 작성

```
package com.jsp.test;

public class Compute {
   public static int add(String x, String y) {
      int a = 0;
      int b = 0;
      try {
        a = Integer.parseInt(x);
        b = Integer.parseInt(y);
      }catch(Exception e) {}
      return a + b;
   }
}
```

2) "elFunctions.tld" 파일을 작성하여 "웹 루트\WEB-INF\tlds" 폴더에 저장한다.

```
<?xml version="1.0" encoding="euc-kr" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <description>EL에서 함수실행</description>
    <tlib-version>1.0</tlib-version>
    <short-name>ELfunctions</short-name>
    <uri>/ELFunctions</uri>

    <function>
        <description> x 와 y의 합</description>
        <name>add</name>
        <function-class>com.jsp.test.Compute</function-class>
        <function-signature>
             int add(java.lang.String, java.lang.String)
        </function-signature>
   </function>
</taglib>
```

3) "웹 루트\WEB-INF\web.xml" 파일 수정

```
<?xml version="1.0" encoding="euc-kr"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
             http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
   :
  <taglib>
        <taglib-uri>
            /WEB-INF/tlds/elFunctions.tld
        </taglib-uri>
        <taglib-location>
            /WEB-INF/tlds/elFunctions.tld
        </taglib-location>
    </taglib>
</web-app>
```

4) JSP 페이지 작성 - elFunctionTest.jsp

```
<%@ page contentType="text/html;charset=euc-kr"%>
<%@ taglib prefix="test" uri="/WEB-INF/tlds/elFunctions.tld" %>

<% request.setCharacterEncoding("euc-kr");%>

<head>
<title>표현 언어에서 함수사용하기</title>
</head>

<body>

<h3>표현 언어에서 함수사용하기 -두 숫자의 덧셈</h3>
<p/>
<form action="elFunctionTest.jsp" method="post">
   X : <input type="text" name="x" value="${param['x']}"/>
   <br>/
   Y : <input type="text" name="y" value="${param['y']}"/>
   <input type="submit" value="덧셈"/>
</form>

<p/>
합은 : ${test:add(param["x"],param["y"])} 입니다.

</body>
</html>
```

10.2 JSTL(JSP Standard Tag Library)

10.2.1 개요

JSP 에는 XML 처럼 사용자가 태그를 정의해서 사용하는 것이 가능하며
이런 사용자 정의 태그를 커스텀 태그라고 하는데
이들 중 자주 사용하는 것을 표준으로 만들어 놓은 것이 JSTL 이다.
JSP Standard Tag Library(JSTL)는 일반적인 웹 애플리케이션 기능인
반복(iteration)과 조건, 데이터 관리 포맷, XML 조작, 데이터베이스 액세스를 구현하는
커스텀 태그 라이브러리 모음이다.

JSP의 스크립트와 HTML 코드를 같이 혼용하여 사용하면 개발의 편리성을 제공하긴 하지만,
코드의 복잡성을 증대시키는 문제점이 있다.
이러한 문제를 해결하기 위해 로직 부분의 JSP 코드를 태그로 대치시켜서
HTML과 같은 태그로 이루어진 코딩 방법이 제시되었다.

JSTL 은 JSP 페이지의 로직을 담당하는 부분인 if, for, while, 데이터베이스 처리 등과 관련된
표준 커스텀 태그를 제공함으로써 코드를 깔끔하게 하고 가독성을 좋게 하며
데이터 포맷, 반복 콘텐트 또는 조건 콘텐트 같은 전형적인 표현 레이어를 위한 표준 구현을 제공하기 때문에,
JSP 작성자들이 애플리케이션 개발에 집중하는데 도움이 된다.

JSTL 작성 시 한 가지 주의할 사항은 액션 태그도 그렇지만
JSTL과 커스텀 태그도 XML 기반에서 작성이 되었기 때문에
모든 태그는 시작 태그와 종료 태그의 쌍으로 이루어져야 한다.

기존의 컨텍스트에서 JSTL을 사용하기 위해서는
웹 어플리케이션의 WEB-INF/lib 디렉토리에 필요한 라이브러리를 복사하면 된다.
JSTL의 주된 라이브러리 파일은 jstl.jar, standard.jar 이고,
xml에서 지원되는 기능을 사용하기 위해서 jaxen-full.jar, saxpath.jar, jaxp-api.jar 파일 등이 필요하다.
이 파일들을 웹 어플리케이션의 WEB-INF/lib 에 복사하고, 컨텍스트(context)를 리 로드 한다.

■ 환경 설정
JSTL을 사용하기 위해서는 jstl.jar, standard.jar 두 개의 jar 파일이 필요하다.

『 http://jakarta.apache.org/site/downloads/ 』 사이트 다운로드 목록 중에서
"Taglibs" - "Standard 1.1 Taglib" - 1.1.2.zip 파일을 다운로드 받는다.

압축을 해제 하여 두개의 파일을 "톰캣_루트\lib" 폴더(톰캣 5.x 인 경우 "톰캣_루트\commons\lib" 폴더)와 작업 폴더가 study인 경우 "톰캣_루트\webapps\study\WEB-INF\lib" 폴더 안에 복사 한다.

■ JSTL 레퍼런스 사이트
『 http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html 』

10.2.2 JSTL 사용

JSTL은 태생이 커스텀태그이기 때문에 jsp와 밀접하게 관계가 있다. application, session, request, response, pageContext 등의 내장객체에 쉽게 접근하며, 그 외에도 파라미터, 헤더, 쿠키 등을 복잡한 코드를 사용하지 않고, 쉽게 직관적으로 사용할 수 있다. 또한 기본적인 연산이나 객체의 비교 등을 .equals() 메서드 등을 이용하는 대신 == 와 같이 쉽게 구현했으며, 조건, 반복, 이동에 대한 태그를 지원하기 때문에 태그만으로도 반복 기능을 구현할 수 있다.

JSTL 의 처리영역은 크게 4가지로 나누어진다. core, format, xml, sql 로 기능이 구분되고, 각각의 기능은 이름이 말해 주듯, 기본기능, 형식화, xml 처리, sql 처리를 담당한다.

[1] 태그의 종류

■ 코어(Core)
- 기능 : 변수지원, 흐름제어, URL처리
- 접두어(Prefix) : c
- URI : http://java.sun.com/jsp/jstl/core

■ XML
- 기능 : XML 코어, 흐름 제어, XML 변환
- 접두어(Prefix) : x
- URI : http://java.sun.com/jsp/jstl/xml

■ 국제화
- 기능 : 지역, 메시지 형식, 숫자 및 날짜형식
- 접두어(Prefix) : fmt
- URI : http://java.sun.com/jsp/jstl/fmt

■ 데이터베이스
- 기능 : SQL
- 접두어(Prefix) : sql
- URI : http://java.sun.com/jsp/jstl/sql

■ 함수(Functions)
- 기능 : 컬렉션 처리, String 처리
- 접두어(Prefix) : fn
- URI : http://java.sun.com/jsp/jstl/functions

[2] JSP에 taglib 추가

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
```

10.2.3 Core

core(코어) 는 변수 선언, 삭제 등 변수와 관련된 작업과, if, for 문등과 같은 제어문, 그리고 URL 처리 등에 사용 되며 core(코어) 태그 라이브러리를 사용하려면 JSP 페이지에 다음과 같이 <%@ taglib ...> 디렉티브를 작성해야 한다.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

taglib 디렉티브의 prefix 속성은 uri 속성에 명시된 값 대신에 해당페이지에서 prefix 속성 값으로 명시된 값을 사용하겠다는 의미이다. 즉, 태그의 시작에 c 를 사용하겠다는 것이다. c 는 core(코어) 태그를 의미한다.

[1] code 태그

■ <c:out/>
JSP 의 표현식을 대체하는 것으로 가장 많이 사용된다.

```
Syntax 1: body 없는 경우
  <c:out value="value" [escapeXml="{true|false}"] [default="기본_값"] />

Syntax 2: body 있는 경우
  <c:out value="value" [escapeXml="{true|false}"] />
      기본_값
   </c:out>
```

escapeXml 애트리뷰트 또한 선택사항이다. "<", ">", "&" 같은 캐릭터가 <c:out> 태그에 의해 아웃풋 될 때 종료되는지의 여부를 제어한다. escapeXml이 true로 설정되어 있다면 이 캐릭터들은 상응하는 XML 인터티(<, >, &)로 바뀐다.(escapeXml 기본값 true)

· 사용 예

```
  Hello <c:out value="${user.username}" default="Guest"/>!
```

```
  <c:out value="${user.company}" escapeXml="false"/>
```

```
  <c:set var="timezone" scope=="session">
     <c:out value="${cookie['tzPref'].value}" default="CST"/>
  </c:set>
```

■ <c:set/> / <c:remove/>

set는 JSP의 setAttribute()와 같은 역할을 한다. (page|request|session|application) 범위의 변수(속성)를 설정한다.

remove는 JSP의 removeAttribute()와 같은 역할을 한다. (page|request|session|application) 범위의 변수(속성)를 제거한다. 형식은 <c:remove> 태그로 사용한다.

scope 속성이 생략될 경우 기본 값은 page 이다.

```
Syntax 1: scope 에 해당하는 변수에 속성 값을 정한다.
  <c:set value="value" var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: scope 에 해당하는 변수에 body 값을 정한다.
  <c:set var="varName" [scope="{page|request|session|application}"]>
      body content
  </c:set>

Syntax 3: 속성 값으로 target 객체의 프로퍼티 값을 정한다.
  <c:set value="value" target="target" property="propertyName"/>

Syntax 4: body 값으로 target 객체의 프로퍼티 값을 정한다.
  <c:set target="target" property="propertyName">
     body content
  </c:set>
```

■ <c:catch/>

body 위치에서 실행되는 코드의 예외를 잡아내는 역할을 담당한다.

```
Syntax
  <c:catch var="name">
    body content
  </c:catch>
```

・사용 예

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<% response.setContentType("text/html"); %>

<h3>코어 </h3>
<h4>&lt;c:out></h4>
<pre>
${1+2} <c:out value="${1+2}"/>
${1>3} <c:out value="${1>3}"/>
${1 gt 3} <c:out value="${1 gt 3}"/>

\$표시 <c:out value="${'${'}test}"/>

escapeXml 속성 기본값은 false
false: <c:out value="<b>bold</b> <,>,&,',\" " escapeXml="false"/>
true:  <c:out value="<b>bold</b> <,>,&,',\" " escapeXml="true"/>

" 큰따옴표 사용주의 ' 작은따옴표로 대치
&lt;c:out value='&lt;font color="blue">파랑&lt;/font>'/>
<c:out value='<font color="blue">파랑</font>' escapeXml="false"/>

<hr><h4>&lt;c:set></h4>
set session scope var "name": <c:set var="name" value="하늘" scope="session"/>
c:out name: <c:out value="${name}"/>
expression name: <%= session.getAttribute("name")%>
set page scope var "name": <c:set var="name">
  hello
</c:set>
c:out name: <c:out value="${pageScope.name}"/>
c:out sessionScope.name: <c:out value="${sessionScope.name}"/>
expression name: <%= session.getAttribute("name")%>

<hr><h4>&lt;c:remove></h4>
remove session scope var "name": <c:remove var="name" scope="session"/>
expression name: <%= session.getAttribute("name")%>
c:out sessionScope.name: <c:out value="${sessionScope.name}"/>

<hr><h4>&lt;c:catch></h4>
<c:catch var="errmsg">
line1
<%=1/0 %>
line2
```

```
</c:catch>
<c:out value="${errmsg}"/>
</pre>
```

■ <c:if/>
조건문을 사용할 때 쓴다.

```
Syntax 1: Body 없는 경우
  <c:if test="testCondition" var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: Body 있는 경우
  <c:if test="testCondition" [var="varName"] [scope="{page|request|session|application}"]>
     body content
  </c:if>
```

<c:if/> 에서 나온 결과를 varName 변수에 넣고, 나중에 활용이 가능하다. 변수의 scope는 임의로 지정할 수 있고, 생략될 경우 기본 값은 page 이며 else 문은 존재 하지 않는다.

■ <c:choose/>, <c:when/>, <c:otherwise/>
<c:choose/> 태그는 java 의 switch 문과 같지만, 조건에 문자열 비교도 가능하고 쓰임의 범위가 넓다. 또한 <c:if/> 태그에 else가 없기 때문에 이의 대체 기능도 수행한다.

```
Syntax
  <c:choose>
    body content
    (하나 이상의 <when>과 하나 이하의 <otherwise> 서브태그)
      <c:when test="조건_1">
          body content
      </c:when>
      <c:when test="조건_2">
          body content
      </c:when>
         :
      <c:otherwise>
          conditional block
      </c:otherwise>
  </c:choose>
```

・조건을 판단하는 간단한 예

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<% response.setContentType("text/html"); %>

<h3>조건</h3>
파라미터 없음:<c:out value="${empty param.name}" />
<h4>&lt;c:if test=""></h4>
<c:if test="${empty param.name}">
<form>
이름을 적어주세요.<br/>
```

```
    <input type="text" name="name"/>
    <input type="submit" value="확인"/>
</form>
</c:if>
<c:if test="${!empty param.name}">
    안녕하세요. <c:out value="${param.name}"/>님.
</c:if>

<h4>&lt;c:choose> &ltc:when test=""> &ltc:otherwise></h4>
<c:choose>
    <c:when test="${empty param.name}">
        <form>
        이름을 적어주세요.<br/>
            <input type="text" name="name"/>
            <input type="submit" value="확인"/>
        </form>
    </c:when>
    <c:when test="${param.name=='admin'}">
        안녕하세요. 관리자님.
    </c:when>
    <c:otherwise>
        안녕하세요. <c:out value="${param.name}"/>님.
    </c:otherwise>
</c:choose>
```

■ <c:forEach/>
<c:forEach/> 는 강력한 반복실행 태그이다.

```
Syntax 1: 객체 전체에 걸쳐서 반복
  <c:forEach [var="varName"] items="collection" [varStatus="varStatusName"]
           [begin="begin"] [end="end"] [step="step"]>
      body content
  </c:forEach>

Syntax 2: 지정한 횟수만큼 반복
  <c:forEach [var="varName"] [varStatus="varStatusName"]
           begin="begin" end="end" [step="step"]>
      body content
  </c:forEach>
```

<c:forEach/> 태그는 여러 가지로 활용이 가능하다. 원하는 구간만큼 반복할 수도 있고, 객체를 받아와서 그 객체의 길이만큼 반복할 수도 있다. begin , end 속성은 시작번호와 끝 번호를 지정하고, step 속성을 이용해서 증가 구간을 정할 수 있다. var 속성에서 정한 변수로 반복되는 내부 구간에서 사용할 수 있다.

컬렉션의 멤버들 사이를 반복할 때 <c:forEach> 태그의 추가 애트리뷰트인 items 애트리뷰트가 사용된다.

· 사용 예

```
<table>
    <tr><td>Value</td>
      <td>Square</td></tr>
```

```
  <c:forEach var="x" begin="0" end="10" step="2">
    <tr><td><c:out value="${x}"/></td>
        <td><c:out value="${x * x}"/></td></tr>
  </c:forEach>
</table>
```

■ <c:forTokens/>
<c:forTokens/> 는 java.util.StringTokenizer 를 이용한 것이다

```
Syntax
  <c:forTokens items="stringOfTokens" delims="delimiters" [var="varName"]
       [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>
    body content
  </c:forEach>
```

・forEach/forTokens 예

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<% response.setContentType("text/html"); %>

<h3>반복</h3>
<h4>&lt;c:forEach></h4>

<c:forEach var="one" begin="1" end="10">
    <c:out value="${one}"/>
</c:forEach>
<p><b>header</b></p>
<c:forEach var="h" items="${header}">
    <c:out value="${h.key}:${h.value}"/><br/>
</c:forEach>

<h4>&lt;c:forTokens></h4>
<c:forTokens var="one"
            items="서울¦인천,대전,대구,부산,광주,평양"
            delims="," varStatus="sts">
    <c:out value="${sts.count}:${one}"/>&middot;
</c:forTokens>
<hr/>
<c:forTokens var="one"
            items="서울¦인천,대전,대구,부산,광주,평양"
            delims=",¦" varStatus="sts">
    <c:out value="${sts.count}:${one}"/>&#149;
</c:forTokens>
```

■ <c:import/>
<c:import/> 는 아주 강력한 도구로 웹 어플리케이션 내부의 자원 접근은 물론이고, http, ftp 같은 외부에 있는 자원도 가져와서 페이지 내에 귀속시킨다. 자유롭게 가공할 수도 있고, 편집도 가능하다.

```
Syntax 1: 해당 주소를 바로 출력하거나 String 에 담아놓는다.
```

```
<c:import url="url" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]
        [charEncoding="charEncoding"]>
   <c:param> 서브 태그 위치
</c:import>

Syntax 2: 해당 주소의 컨텐츠를 Reader 객체로
<c:import url="url" [context="context"]
        varReader="varReaderName"
        [charEncoding="charEncoding"]>
    varReader 를 사용하는 액션
</c:import>
```

・사용 예 - 3개의 주소 불러 오기

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<% response.setContentType("text/html"); %>

<c:set var="url" value="http://www.daum.net/"/>
<c:import url="${url}" var="u"/>
<c:out value="${url}"/> 가져옵니다.
<hr/>
-- 홈페이지 결과 출력<br/>
 <c:out value="${u}" escapeXml="false"/>
<hr/>

<c:set var="url" value="http://www.naver.com"/>
<c:import url="${url}" var="u"/>
<c:out value="${url}"/> 가져옵니다.
<hr/>
-- 소스 출력<br/>
<pre><c:out value="${u}"/></pre>
<hr/>

<c:set var="url" value="gugu.jsp"/>
<c:import url="${url}" var="u">
    <c:param name="su" value="5"/>
</c:import>
<c:out value="${url}"/> 가져옵니다.
<hr/>
<c:out value="${u}" escapeXml="false"/>
<hr/>
```

■ <c:url/>

<c:url/> 태그는 컨텍스트를 자동으로 추가해서 주소를 자동으로 생성해준다. context 속성이 지정되었을 경우 value 와 context 의 값은 『 / 』로 시작을 해야 된다. context 속성이 생략되면 당연히 현재의 컨텍스트가 적용된다.

```
Syntax 1: Body 없는 경우
```

```
  <c:url value="value" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]/>

Syntax 2: Body 있는 경우 쿼리 스트링 파라미터 지정
  <c:url value="value" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]>
    <c:param> 서브태그
  </c:url>
```

・사용 예

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:url value="/images/tomcat.gif"/>
<img src='<c:url value="/images/tomcat.gif"/>' />
```

■ <c:redirect/>
response.sendRedirect() 를 대체하는 태그이다. 컨텍스트를 지정해서 다른 컨텍스트로 이동이 가능하다.

```
Syntax 1: Body 없는 경우
  <c:redirect url="value" [context="context"]/>

Syntax 2: Body 있는 경우 쿼리 스트링 파라미터 지정
  <c:redirect url="value" [context="context"]/>
    <c:param> 서브태그
  </c:redirect>
```

・사용 예

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<c:redirect url="/test.jsp"/>
```

■ <c:param/>
<import>태그의 URL뒤에 파라미터로 붙여서 사용할 수 있다. 형식은 <c:param> 태그로 사용되며 <c:param/>은 다음과 같이 url 에 바로 붙여서 쓸 수도 있다.

```
<c:import url="/exec/doIt">
  <c:param name="action" value="register"/>
</c:import>

이 방법은 아래 태그와 같은 효과가 있다.
<c:import url="/exec/doIt?action=register"/>
```

[2] 기능별 분류

■ 표현 언어(EL) 지원 기능
<c:catch>, <c:out>, <c:remove>, <c:set>

■ 흐름 제어 기능
<c:choose>(<c:when>, <c:otherwise>), <c:forEach>, <c:forTokens>, <c:if>

■ URL 관리 기능
<c:import>(<c:param>), <c:redirect> (<c:param>), <c:url> (<c:param>)

10.2.4 I18N (Internationalization) - fmt 태그

JSTL 국제화 지역화 태그로 다국어 문서를 처리할 때 유용하고, 날짜와 숫자 형식을 다룰 때 사용된다. fmt 태그 라이브러리를 사용하려면 JSP 페이지에 다음과 같이 <%@ taglib> 디렉티브를 작성해야 한다.

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

taglib 디렉티브의 prefix 속성은 uri 속성에 명시된 값 대신에 해당페이지에서 prefix 속성 값으로 명시된 값을 사용하겠다는 것으로 여기서는 태그의 시작에 fmt 를 사용하겠다는 것이다. fmt 는 JSTL fmt 태그를 의미 한다.

■ <fmt:setLocale/>
다국어를 지원하는 페이지를 만들 경우 사용할 경우 ResourceBundle 로 불러오는 *.properties 파일들과 연계되어서 사용한다.

```
Syntax
  <fmt:setLocale value="locale" [variant="variant"]
        [scope="{page|request|session|application}"]/>
```

value 속성에 들어가는 locale 값은 http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt 언어코드와 http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html 국가코드로 이루어진다. 생략될 경우 톰캣 서버의 기본값으로 설정이 되고, 둘 중에 하나만 사용할 수도 있다.

・사용 예

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.comjsp/jsp/jstl/fmt" prefix="fmt"%>

<pre>
  default locale : <%= response.getLocale() %>
  set locale : ko <fmt:setLocale value="ko" />
  now: <%= response.getLocale() %>
  set locale : ja <fmt:setLocale value="ja" />
  now: <%= response.getLocale() %>
  set locale : en <fmt:setLocale value="en" />
  now: <%= response.getLocale() %>
</pre>
```

■ <fmt:requestEncoding/>
request.setCharacterEncoding() 역할을 한다.

```
Syntax
  <fmt:requestEncoding [value="charsetName"]/>
```

・사용 예

```
<%@ page contentType="text/html;charset=euc-kr" %>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<fmt:requestEncoding value="euc-kr"/>

파라미터:<c:out value="${param.id}"/>
<form method="post">
    <input type="text" name="id"/>
    <input type="submit"/>
</form>
```

■ <fmt:bundle/>
properties 확장자를 사용하는 자원 파일을 읽어오는 역할을 한다.

```
Syntax
  <fmt:bundle basename="basename" [prefix="prefix"]>
    body content
  </fmt:bundle>
```

basename 속성에 지정된 properties 파일을 찾아서 locale 에 따라 읽어들인다.
properties 파일은 보통 WEB-INF/classes 아래에 위치하며 디렉토리의 깊이에 따라서 패키지형식의 이름을 취한다. TestBundle.properties 파일이com/itexpert/test/msg 디렉토리에 있다면 basename="com.itexpert.test.msg.TestBundle" 이라고 지정하면 된다.
locale 이 ko 라면 TestBundle_ko.properties 파일을 읽어오게 되며, locale 이 맞지 않는 경우에는 TestBundle.properties 처럼 코드가 붙지 않은 파일을 읽어온다.
prefix 속성은 key 명칭이 공통적인 부분을 지정해서 body 에서 표현되는 key 를 단축시킨다. import 에서 패키지 명을 지정하면 클래스명만 쓸 수 있는 것과 같이 생각할 수 있다.
properties 파일의 경우 j2sdk의 /bin/native2ascii.exe 를 이용해서 유니코드로 변환될 필요가 있으나, 파일 수가 많을 경우 ant 의 <native2ascii> 태스크를 이용해서 쉽게 처리할 수 있다.

■ <fmt:message/>
번들 태그에서 정한 값들을 가져온다.

```
Syntax 1: body 없는 경우
  <fmt:message key="messageKey" [bundle="resourceBundle"] [var="varName"]
        [scope="{page|request|session|application}"]/>

Syntax 2: 메시지 파라미터를 지정하는 body가 있는 경우
  <fmt:message key="messageKey" [bundle="resourceBundle"] [var="varName"]
        [scope="{page|request|session|application}"]>
    <fmt:param> 서브태그
  </fmt:message>

Syntax 3: 키와 선택적 메시지 파라미터를 지정하는 body가 있는 경우
  <fmt:message [bundle="resourceBundle"] [var="varName"]
        [scope="{page|request|session|application}"]>
    key
    선택적 <fmt:param> 서브태그
  </fmt:message>
```

번들에 있는 key 값을 불러온다. bundle 속성으로 번들을 직접 설정할 수도 있고, <fmt:bundle/> 태그 사이에 중첩되어서 키 값만 받아서 출력할 수 있다.

• 번들 예제

1) 웹 루트\WEB-INF\classes\bundle 경로에 다음의 properties 파일 두 개를 작성 한다.

- testBundle_ko.properties

```
name=\ud558\ud558\ud558.
message=JSTL \uc7ac\ubbf8\uc788\ub2e4.
```

- testBundle.properties

```
name=Han.
message=JSTL is fun.
```

[참고] properties 파일은 한글은 입력할 수 없기 때문에 유니코드로 입력해야 하며 j2sdk의 /bin/native2ascii.exe 파일을 이용하여 한글을 유니코드 값을 확인 할 수 있다. 먼저 한글을 입력하고 엔터를 누르면 유니코드로 변환되어 출력한다.

properties 파일을 두 개 작성하는 이유는 로케일에 따라 영어 또는 한글로 출력할 수 있다. 영어로 출력하기 위해서는 익스플로러 의 "인터넷 옵션- 일반- 언어"에서 한국어를 지우고 영어를 선택하면 영어로 출력 된다.

2) JSP 파일 작성

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib prefix="c" uri="http://java.sun.comjsp/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"  %>

<html>
<head><title>JSTL fmt 예제 - bundle , message</title></head>
<body>

<fmt:bundle basename="bundle.testBundle">
   <fmt:message key="name"/>
   <p/>

   <fmt:message key="message" var="msg"/>
   <c:out value="${msg}"/>
</fmt:bundle>

</body>
</html>
```

■ <fmt:setBundle/>

페이지 전체에서 사용할 수 있는 번들을 지정할 수 있는데, 이에 대한 지정은 <fmt:setBundle/> 태그가 담당한다. var 속성에서 정한 변수를 이후에 나오는 <fmt:message/> 태그에서 basename 속성에 변수 명으로 대체할 수 있다.

```
Syntax
  <fmt:setBundle basename="basename"
        [var="varName"] [scope="{page|request|session|application}"]/>
```

■ <fmt:formatNumber/>
숫자 형식을 표현할 때 사용된다.

```
Syntax 1: body 없는 경우
  <fmt:formatNumber value="numericValue"
        [type="{number|currency|percent}"] [pattern="customPattern"]
        [currencyCode="currencyCode"] [currencySymbol="currencySymbol"]
        [groupingUsed="{true|false}"] [maxIntegerDigits="maxIntegerDigits"]
        [minIntegerDigits="minIntegerDigits"] [maxFractionDigits="maxFractionDigits"]
        [minFractionDigits="minFractionDigits"] [var="varName"]
        [scope="{page|request|session|application}"]/>

Syntax 2: 형식에 맞출 수치가 body에 있는 경우
  <fmt:formatNumber [type="{number|currency|percent}"]
        [pattern="customPattern"] [currencyCode="currencyCode"]
        [currencySymbol="currencySymbol"] [groupingUsed="{true|false}"]
        [maxIntegerDigits="maxIntegerDigits"] [minIntegerDigits="minIntegerDigits"]
        [maxFractionDigits="maxFractionDigits"] [minFractionDigits="minFractionDigits"]
        [var="varName"] [scope="{page|request|session|application}"]>
    형식화될 수치
  </fmt:formatNumber>
```

・속성

속성	동적	Type	설명
value	true	String 또는 Number	형식화될 수치
type	true	String	숫자, 통화, 퍼센트 중 어느 것으로 표시할 지 지정 {number|currency|percent}
pattern	true	String	사용자가 지정한 형식 패턴.
currencyCode	true	String	ISO 4217 통화 코드. 통화 형식일 때만 적용 (type="currency")
currencySymbol	true	String	통화 기호. 통화 형식일 때만 적용 (type="currency")
groupingUsed	true	boolean	형식 출력에 그룹 분리기호를 포함할지 여부
maxIntegerDigits	true	int	형식 출력에서 integer 최대 자리 수
minIntegerDigits	true	int	형식 출력에서 integer 최소 자리 수
maxFractionDigits	true	int	형식 출력에서 소수점 이하 최대 자리 수.
minFractionDigits	true	int	형식 출력에서 소수점 이하 최소 자리 수.
var	false	String	형식 출력 결과 문자열을 담는scope 에 해당하는 변수 명
scope	false	String	var 의 scope

■ <fmt:parseNumber/>
반대로 정해진 패턴을 문자열에서 수치를 파싱해내는 태그 이다.

```
Syntax 1: body가 없는 경우
  <fmt:parseNumber value="numericValue"
        [type="{number|currency|percent}"] [pattern="customPattern"]
        [parseLocale="parseLocale"] [integerOnly="{true|false}"]
```

```
        [var="varName"] [scope="{page|request|session|application}"]/>

Syntax 2: 파싱할 수치를 body 에 갖고 있는 경우
  <fmt:parseNumber [type="{number|currency|percent}"]
        [pattern="customPattern"] [parseLocale="parseLocale"]
        [integerOnly="{true|false}"] [var="varName"]
        [scope="{page|request|session|application}"]>
    파싱할 수치
</fmt:parseNumber>
```

・속성

속성	동적	Type	설명
value	true	String 또는 Number	파싱할 수치
type	true	String	숫자, 통화, 퍼센트 중 어느 것으로 표시할 지 지정 {number|currency|percent}
pattern	true	String	사용자가 지정한 형식 패턴.
parseLocale	true	String 또는 java.util.Locale	파싱 작업의 기본 형식 패턴(숫자, 통화, 퍼센트 각각)을 제공하는 Locale
integerOnly	true	boolean	주어진 값에서 integer 부분만 파싱할지 여부를 지정
var	false	String	파싱 결과(java.lang.Number 타입)를 담는 scope에 해당하는 변수 명
scope	false	String	var 의scope

■ <fmt:formatDate/>
날짜 형식을 표현하는 태그

```
Syntax
  <fmt:formatDate value="date"
        [type="{time|date|both}"] [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"] [pattern="customPattern"]
        [timeZone="timeZone"] [var="varName"]
        [scope="{page|request|session|application}"]/>
```

・속성

속성	동적	Type	설명
value	true	java.util.Date	형식화될 Date 와 time
type	true	String	형식화할 데이터가 시간, 날짜, 모두 인지 셋 중 하나를 지정한다.
dateStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="date", type="body", type속성이 생략된 경우 사용.
timeStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="time", type="body" 의 경우 사용.
pattern	true	String	사용자 지정 형식 스타일
timeZone	true	String 또는 java.util.TimeZone	형식화 시간에 나타날 타임 존
var	false	String	형식 출력 결과 문자열을 담는 scope에 해당하는 변수 명
scope	false	String	var 의 scope

■ <fmt:parseDate/>
정해진 패턴의 문자열에서 날짜를 파싱해내는 태그

```
Syntax 1: body 없는 경우
  <fmt:parseDate value="dateString"
        [type="{time|date|both}"] [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"] [pattern="customPattern"]
        [timeZone="timeZone"] [parseLocale="parseLocale"]
        [var="varName"] [scope="{page|request|session|application}"]/>

Syntax 2: 파싱한 값이 body 에 있는 경우
  <fmt:parseDate [type="{time|date|both}"]
        [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"] [pattern="customPattern"]
        [timeZone="timeZone"] [parseLocale="parseLocale"] [var="varName"]
        [scope="{page|request|session|application}"]>
    파싱할 Date 와 time
  </fmt:parseDate>
```

・속성

속성	동적	Type	설명
value	true	java.util.Date	파싱할 Date 와 time
type	true	String	파싱할 데이터가 시간, 날짜, 모두 인지 셋 중 하나를 지정한다.
dateStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="date", type="body", type속성이 생략된 경우 사용.
timeStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="time", type="body" 의 경우 사용.
pattern	true	String	사용자 지정 형식 스타일
timeZone	true	String 또는 java.util.TimeZone	형식화 시간에 나타날 타임 존
parseLocale	true	String 또는 java.util.Locale	파싱하는 동안 적용될 미리 정의된 형식 스타일의 Locale
var	false	String	파싱 결과(java.util.Date)를 담는 scope에 해당하는 변수 명
scope	false	String	var 의 scope

・숫자, 날짜 태그 예제

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ page pageEncoding="MS949" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<pre><fmt:setLocale value="ko_KR"/>
number  : <fmt:formatNumber value="9876543.61" type="number"/>
currency: <fmt:formatNumber value="9876543.61" type="currency"/>
percent : <fmt:formatNumber type="percent">9876543.61</fmt:formatNumber>

pattern=".000"    :<fmt:formatNumber value="9876543.61" pattern=".000" />
pattern="#,#00.0#":<fmt:formatNumber value="9876543.612345" pattern="#,#00.0#"/>

<jsp:useBean id="now" class="java.util.Date"/>
<c:out value="${now}"/>
date: <fmt:formatDate value="${now}" type="date"/>
time: <fmt:formatDate value="${now}" type="time"/>
both: <fmt:formatDate value="${now}" type="both"/>

default:<fmt:formatDate value="${now}"
            type="both" dateStyle="default" timeStyle="default"/>
short  :<fmt:formatDate value="${now}"
            type="both" dateStyle="short"   timeStyle="short"  />
medium :<fmt:formatDate value="${now}"
            type="both" dateStyle="medium"  timeStyle="medium" />
long   :<fmt:formatDate value="${now}"
            type="both" dateStyle="long"    timeStyle="long"   />
full   :<fmt:formatDate value="${now}"
            type="both" dateStyle="full"    timeStyle="full"   />
```

```
pattern="yyyy년MM월dd일 HH시mm분ss초"
         <fmt:formatDate value="${now}" type="both"
            pattern="yyyy년MM월dd일 HH시mm분ss초"/>
</pre>
```

■ <fmt:setTimeZone/>, <fmt:timeZone/>
특정 스코프의 타임 존을 설정하는 <fmt:setTimeZone/> 태그의 형식은 다음과 같다.

```
Syntax
  <fmt:setTimeZone value="timeZone"
        [var="varName"] [scope="{page|request|session|application}"]/>
```

타임 존을 부분 적용하는 <fmt:timeZone/> 태그는 다음과 같은 형식이다.

```
Syntax
  <fmt:timeZone value="timeZone">
    body content
  </fmt:timeZone>
```

· 타임 존을 이용한 시간 표시

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<pre><fmt:setLocale value="ko_KR"/>
<jsp:useBean id="now" class="java.util.Date"/>

default : <c:out value="${now}"/>
Korea   KST  : <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
<fmt:timeZone value="GMT">
Swiss   GMT  : <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
</fmt:timeZone>
<fmt:timeZone value="GMT-8">
NewYork GMT-8 : <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
</fmt:timeZone>
</pre>
```

10.2.5 JSTL sql

DataSource 를 이용해서 SQL 을 처리하는 작업등에 사용되며 sql 태그 라이브러리를 사용하려면 JSP 페이지에 다음과 같이 <%@ taglib> 디렉티브를 작성해야 한다.

```
  <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

taglib 디렉티브의 prefix 속성은 uri 속성에 명시된 값 대신에 해당페이지에서 prefix 속성 값으로 명시된 값을 사용하겠다는 것으로 여기서는 태그의 시작에 sql 를 사용하겠다는 것이다. sql 는 JSTL sql 태그를 의미 한다.

■ <sql:setDataSource/>
DataSource 를 지정하는 태그 이다.

```
Syntax
  <sql:setDataSource
        {dataSource="dataSource" ¦
            url="jdbcUrl"
            [driver="driverClassName"]
            [user="userName"]
            [password="password"]}
        [var="varName"]
  [scope="{page¦request¦session¦application}"]/>
```

오라클에서 사용을 한다면 다음과 같이 DataSource 를 설정할 수 있다.

```
<sql:setDataSource
            url="jdbc:oracle:thin:@localhost:1521:ora"
            driver="oracle.jdbc.driver.OracleDriver"
            user="scott"
            password="tiger"
        var="jspDS"
        scope="application" />
```

이미 컨텍스트에 JNDI 설정이 되어있다면 다음과 같이 바로 불러서 사용하거나 <sql:query/> 에서 바로 사용할 수 있다.

기존의 dataSource 를 불러와 사용하는 경우(JNDI 가 jdbc/myoracle 인 경우)

```
<sql:setDataSource
        dataSource="jdbc/myoracle"
        var="jspDS"
        scope="application" />
```

<sql:query/> 에서 바로 사용하는 경우

```
<sql:query var="emp"
        dataSource="jdbc/myoracle">
```

■ <sql:query/>
java 와는 달리 sql 문장을 문자열로 연결하지 않아도 가독성을 높여서 작성할 수 있다.

```
Syntax 1: body 없는 경우
  <sql:query sql="sqlQuery"
        var="varName" [scope="{page¦request¦session¦application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]/>

Syntax 2: body 에 쿼리의 파라미터가 있는 경우
  <sql:query sql="sqlQuery"
```

```
        var="varName" [scope="{page|request|session|application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]>
    <sql:param> 액션들
  </sql:query>

Syntax 3: 쿼리와 파라미터들이 body 에 있는 경우
  <sql:query var="varName"
        [scope="{page|request|session|application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]>
    sqlQuery
    선택적 <sql:param> 액션들
  </sql:query>
```

■ <sql:dateParam/>, <sql:param/>
파라미터에는 두 가지가 있는데, 날짜 형식의 <sql:dateParam/> 와 일반적인 <sql:param/>태그가 있으며 형식은 다음과 같다.

```
Syntax
  <sql:dateParam value="value" type="[timestamp|time|date]"/>
```

```
Syntax 1: value 속성에 파라미터 값이 지정된 경우
  <sql:param value="value"/>

Syntax 2: body 내용에 파라미터 값이 지정된 경우
  <sql:param>
    parameter value
  </sql:param>
```

<sql:dateParam/>은 java.sql.PreparedStatement.setTimestamp() 역할을 하고,
<sql:param/> 은 java.sql.PreparedStatement.setString() 의 역할을 한다. 바인드 변수의 순서에 따라서 써주면 된다.

■ <sql:update/>
java.sql.Statement.executeUpdate() 메서드에 해당하는 <sql:update/> 태그의 형식은 다음과 같다.

```
Syntax 1: body 없는 경우
  <sql:update sql="sqlUpdate"
        [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]/>

Syntax 2: update 파라미터가 body에 있는 경우
  <sql:update sql="sqlUpdate"
        [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]>
    <sql:param> 액션들
  </sql:update>
```

```
Syntax 3: update 문과 선택적 update 파라미터가 body에 있는 경우
  <sql:update [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]>
    sqlUpdate
    선택적 <sql:param> 액션들
  </sql:update>
```

형식과 동작은 <sql:query/> 태그와 동일하다. 다른 점은 executeUpdate() 메서드를 수행하기 때문에 DB에 변경을 가할 수 있다는 것이다.

■ <sql:transaction/>
트랜잭션을 구현하는 <sql:transaction/> 태그의 형식은 다음과 같다.

```
Syntax
  <sql:transaction [dataSource="dataSource"]
        [isolation=isolationLevel]>
    <sql:query> 과 <sql:update> 문들
  </sql:transaction>

  isolationLevel ::= "read_committed" | "read_uncommitted" | "repeatable_read" | "serializable"
```

격리 수준(isolationLevel)은 java.sql.Connection 의 setTransactionIsolation() 메서드를 사용한다.

10.2.6 JSTL functions

JSTL functions 은 JSTL 에서 제공하는 각종 함수를 사용해서 문자열이나, 컬렉션들을 처리한다.

JSTL functions 태그 라이브러리를 사용하려면 JSP 페이지에 다음과 같이 <%@ taglib> 디렉티브를 작성해야 한다.

```
  <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

taglib 디렉티브의 prefix 속성은 uri 속성에 명시된 값 대신에 해당페이지에서 prefix 속성 값으로 명시된 값을 사용하겠다는 것이다. 여기서는 태그의 시작에 fn 를 사용하겠다는 것이다. fn 는 functions 태그를 의미 한다.

■ 태그 요약
- boolean contains(java.lang.String string, java.lang.String substring)
  string이 substring을 포함하면 true값을 리턴 한다.
- boolean containsIgnoreCase(java.lang.String string, java.lang.String substring)
  대소문자에 관계없이, string이 substring을 포함하면 true값을 리턴 한다.
- boolean endsWith(java.lang.String string, java.lang.String substring)
  string이 suffix로 끝나면 true값을 리턴 한다.
- java.lang.String escapeXml(java.lang.String string)
  string에 XML과 HTML에서 < >& ' " 문자들을 각각 < > & ' "로 바꿔준 뒤 문자열을 리턴 한다.
- int indexOf(java.lang.String string, java.lang.String substring)
  string에서 substring이 처음으로 나타나는 인덱스를 리턴 한다.
- java.lang.String join(java.lang.String[], java.lang.String separator)
  배열 요소들을 separator를 구분자로 하여 모두 연결해서 리턴 한다.

- int length(java.lang.Object item)
  item이 배열이나 컬렉션이면 요소의 개수를, 문자열이면 문자의 개수를 리턴 한다.
- java.lang.String replace(java.lang.String string, java.lang.String before, java.lang.String after)
  string 내에 있는 before 문자열을 after 문자열로 모두 바꿔서 리턴 한다.
- java.lang.String[] split(java.lang.String string, java.lang.String separator)
  string 내의 문자열을 separator에 따라 잘라내서 잘려진 문자열들을 배열로 구성해서 리턴 한다.
- boolean startsWith(java.lang.String string, java.lang.String prefix)
  string이 prefix로 시작하면 true값을 리턴 한다.
- java.lang.String substring(java.lang.String string, int begin, int end)
  string에서 begin 인덱스에서 시작해서 end 인덱스에 끝나는 부분의 문자열을 리턴 한다.
- java.lang.String substringAfter(java.lang.String string, java.lang.String substring)
  string에서 substring이 나타나는 이후의 부분에 있는 문자열을 리턴 한다.
- java.lang.String substringBefore(java.lang.String string, java.lang.String substring)
  string에서 substring이 나타나기 이전의 부분에 있는 문자열을 리턴 한다.
- java.lang.String toLowerCase(java.lang.String string)
  string을 모두 소문자로 바꿔 리턴한다.
- java.lang.String toUpperCase(java.lang.String string)
  string을 모두 대문자로 바꿔 리턴한다.
- java.lang.String trim(java.lang.String string)
  string 앞뒤의 공백을 모두 제거하여 리턴 한다.

10.2.7 JSTL XML

[1] 개요

XML 은 XML 을 처리해주기 위한 것으로 XML 출력, 흐름제어, XML 변환 등에 작업에 사용된다.

JSTL XML 을 사용하려면 기본적으로 XML 에 대해 알고 있어야 가능하다.

XML 태그 라이브러리를 사용하려면 JSP 페이지에 다음과 같이 <%@ taglib> 디렉티브를 작성해야 한다.

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

taglib 디렉티브의 prefix 속성은 uri 속성에 명시된 값 대신에 해당페이지에서 prefix 속성 값으로 명시된 값을 사용하겠다는 것이다. 여기서는 태그의 시작에 x 를 사용하겠다는 것이다. x 는 XML 태그를 의미 한다.

■ xml 태그와 XPath

xml 태그를 사용하기 위해서는 XPath 를 먼저 이해할 필요가 있다. xml 소스 트리의 정확한 위치를 지정해 주기 위한 경로지정 문법이며 XSLT와 XPointer 를 위해서 만들어진 것이다. xml 엘리먼트들을 노드(node) 로 접근한다. 파일 시스템과 유사하며 다음과 같은 특성이 있다.

- / 로 시작하면 절대경로처럼 root node 에서 시작된다.
- //로 시작할 경우는 모든 영역에서 해당 엘리먼트를 선택하게 된다.
- 표시는 이전 엘리먼트 아래의 모든 자식 엘리먼트를 나타낸다.
- 동일한 엘리먼트들이 있을 경우, [] 안에 포함된 숫자는 엘리먼트의 순번이다. 조건식이 올 경우 해당하는 것이 선택된다. last() 일 경우는 맨 마지막 엘리먼트를 표시한다.
- 속성은 @ 로 시작된다.
- normalize-space() 함수는 앞뒤 공백을 제거하는 trim() 역할을 한다.

■ JSTL에서 XPath를 통해서 내장객체에 쉽게 접근할 수 있다.

표현	매핑
$foo	pageContext.findAttribute("foo")
$param:foo	request.getParameter("foo")
$header:foo	request.getHeader("foo")
$cookie:foo	maps to the cookie's value for name foo
$initParam:foo	application.getInitParameter("foo")
$pageScope:foo	pageContext.getAttribute("foo", PageContext.PAGE_SCOPE)
$requestScope:foo	pageContext.getAttribute("foo", PageContext.REQUEST_SCOPE)
$sessionScope:foo	pageContext.getAttribute("foo", PageContext.SESSION_SCOPE)
$applicationScope:foo	pageContext.getAttribute("foo", PageContext.APPLICATION_SCOPE)

예를 들어서 다음 문장은 parameter 로 받은 "name"의 값이 bar 엘리먼트의 x속성의 값과 같은 것들을 선택하게 된다.

```
/foo/bar[@x=$param:name]
```

[2] XML 태그

■ <x:out/>
XPath에 지정한 패턴에 따라 xml 내용을 출력하는 <x:out/> 태그의 형식은 다음과 같다.

```
Syntax
  <x:out select="XPathExpression" [escapeXml="{true|false}"]/>
```

■ <x:parse/>
xml 문서를 읽어서 파싱하는 <x:parse/> 태그는 다음과 같은 형식이다.

```
Syntax 1: String 또는 Reader 객체로 지정된 XML 문서
  <x:parse xml="XMLDocument"
        {var="var" [scope="scopeName"]|varDom="var" [scopeDom="scopeName"]}
        [systemId="systemId"] [filter="filter"]/>

Syntax 2: body 내용으로 지정된 XML 문서
  <x:parse {var="var" [scope="scopeName"]|varDom="var" [scopeDom="scopeName"]}
        [systemId="systemId"] [filter="filter"]>
    파싱할 XML 문서
  </x:parse>
  scopeName 은 {page|request|session|application} 중의 하나
```

■ <x:set/>
XPath에 따라 선택된 내용을 변수에 저장하는 <x:set/> 태그의 형식은 다음과 같다.

```
Syntax
  <x:set select="XPathExpression" var="varName"
```

```
        [scope="{page|request|session|application}"]/>
```

■ <x:if/>
<c:if/> 태그와 마찬가지로 xml 태그에도 <x:if/> 가 있고 형식은 <c:if/> 태그와 유사하다. <x:if/>의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
  <x:if select="XPathExpression"
        var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: Body 있는 경우
  <x:if select="XPathExpression"
        [var="varName"] [scope="{page|request|session|application}"]>
    body content
  </x:if>
```

test 속성 대신에 select 속성으로 진위를 따지는데, 다음 3가지 기준을 유념할 필요가 있다. 이 세 가지 기준은 <x:choose/>의 <x:when/> 과 <x:forEach/> 에서도 동일하게 사용된다.

- number 가 true 인 때는 + 또는 - 0 도 아니고, NaN(Not A Number) 도 아닐 경우
- node-set 이 true 인 때는 empty 가 아닐 경우
- string 이 true 인 때는 길이가 0 이 아닐 경우

■ <x:choose/>, <x:when/>, <x:otherwise/>
<c:choose/> 태그와 마찬가지로 xml태그에도 <x:choose/> 가 있고 형식은 <c:choose/> 태그와 유사하다.

```
Syntax
  <x:choose>
    body content (<x:when> and <x:otherwise> 서브태그)
  </x:choose>
```

```
Syntax
  <x:when select="XPathExpression">
    body content
  </x:when>
```

```
Syntax
  <x:otherwise>
    conditional block
  </x:otherwise>
```

■ <x:forEach/>
<x:forEach/> 태그는 XPath에 따라서 해당하는 엘리먼트 수만큼 반복하게 된다.

```
Syntax
<x:forEach [var="varName"] select="XPathExpression">
    body content
</x:forEach>
```

■ <x:transform/>, <x:param/>
xml과 xslt 파일을 결합해서 새로운 형식의 문서를 생성해 내는 <x:transform/>의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
  <x:transform
        xml="XMLDocument" xslt="XSLTStylesheet"
        [xmlSystemId="XMLSystemId"] [xsltSystemId="XSLTSystemId"]
        [{var="varName" [scope="scopeName"]¦result="resultObject"}]>

Syntax 2: 변환 파라미터를 body에서 지정하는 경우
  <x:transform
        xml="XMLDocument" xslt="XSLTStylesheet"
        [xmlSystemId="XMLSystemId"] [xsltSystemId="XSLTSystemId"]
        [{var="varName" [scope="scopeName"]¦result="resultObject"}]>
    <x:param> 액션들
  </x:transform>

Syntax 3: XML문서와 선택적 변환 파라미터들이 body에 지정된 경우
  <x:transform
        xslt="XSLTStylesheet"
        xmlSystemId="XMLSystemId" xsltSystemId="XSLTSystemId"
        [{var="varName" [scope="scopeName"]¦result="resultObject"}]>
    XML Document
    optional <x:param> actions
  </x:parse>
  scopeName 은 {page¦request¦session¦application} 중에 하나
```

var 속성에 지정된 결과와 result 속성에 지정된 결과의 차이점은 var 속성은 scope 지정해서 다른 곳에서도 사용할 수 있고, result 는 현재 페이지에서만 사용할 수 있다는 것이다.
또한 타입도 차이가 나는데, var는 org.w3c.dom.Document 로, result는 javax.xml.transform.Result 객체로 저장이 된다.

xml의 파라미터를 지정하는 <x:param/> 태그의 형식은 다음과 같다.

```
Syntax 1: value 속성에 파라미터 값이 지정된 경우
<x:param name="name" value="value"/>

Syntax 2: body 내용에 파라미터 값이 지정된 경우
<x:param name="name">
    parameter value
</x:param>
```

<x:transform/> 태그를 사용할 때 jdk 내에 있는xalan과 jstl이 충돌을 일으킨다.
이런 경우, 톰캣 실행을 중지하고, jstl의WEB-INF/lib디렉토리에 있는 xalan.jar 파일과 xercesImpl.jar 파일 두 개를 <CATALINA_HOME>/common/endorsed 디렉토리에 복사한다. 이전 버전의 xercesImpl.jar 파일을 덮어씌운다. 만일 이 과정이 생략되면, 다음과 같은 예외를 만나게 된다.

```
org.apache.xml.utils.WrappedRuntimeException: The output format must have a
'{http://xml.apache.org/xslt}content-handler' property!
```

[3] XML 태그 예제

■ 예제

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

<% response.setContentType("text/html;"); %>

<%-- 파라미터 받아서 출력 --%>
<c:if test="${!empty param.name}">
param: <x:out select="$param:name"/>
</c:if>
<form>
name: <input type="text" name="name"/>
<input type="submit"/>
</form>
<hr>

<%-- xml 데이터를 xdata 변수에 할당 --%>
<x:parse var="xdata">
<namecard>
    <person>
        <name>이순신</name>
        <id>남자</id>
        <email>aaa@abc.com</email>
        <phone>111-2222-3333</phone>
    </person>
    <person>
        <name>홍길동</name>
        <id>남자</id>
        <email>bbb@abc.com</email>
        <phone>222-3333-4444</phone>
    </person>
    <person>
        <name>허허허</name>
        <id>남자</id>
        <email>ccc@abc.com</email>
        <phone>333-4444-5555</phone>
    </person>
</namecard>
</x:parse>
<%-- XPath 를 이용해서 xdata에서 추출 --%>
<x:out select="$xdata//person[1]/name"/>
<x:out select="$xdata//person[last()]/name"/>
<hr/>

<%-- person 으로 반복해서 email과 phone 출력 --%>
<table border="1">
<x:forEach select="$xdata//person">
<tr><td><x:out select="email" /></td>
```

```
<td><x:out select="phone" /></td></tr>
</x:forEach>
</table>
```

파라미터 name 의 접근은 EL에서는 ${param.name} 으로 사용하지만 XPath 에서는 $param:name 을 사용한다는 차이가 있다.
파라미터를 받아서 <x:out/> 으로 출력하는 부분이 제일 상단이고, 그 다음은 xml 데이터를 파싱하는 부분이다.
DTD 까지 쓰지 않아도 형식이 잘 갖춰지기만 하면 (well-formed) xml 데이터로 인식을 한다. 이것을 xdata라는 변수에 할당한 다음에 이후에 처리하게 된다.

xml 데이터는 파일로 따로 만든 후에 접근할 수 있다. 이때는 <c:import> 를 같이 사용하는데, 다음과 같이 쓸 수 있다.

■ namecard.xml

```
<?xml version="1.0" encoding="euc-kr" ?>
<namecard>
    <person>
        <name>이순신</name>
        <id>남자</id>
        <email>aaa@abc.com</email>
        <phone>111-2222-3333</phone>
    </person>
    <person>
        <name>홍길동</name>
        <id>남자</id>
        <email>bbb@abc.com</email>
        <phone>222-3333-4444</phone>
    </person>
    <person>
        <name>허허허</name>
        <id>남자</id>
        <email>ccc@abc.com</email>
        <phone>333-4444-5555</phone>
    </person>
</namecard>
```

■ jstlxml02.jsp

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>

<% response.setContentType("text/html;"); %>

<%-- namecard.xml 파일을 불러와 xdata 변수에 할당 --%>
<c:import url="namecard.xml" var="xmldata" />
<x:parse xml="${xmldata}" var="xdata"/>

<%-- XPath 를 이용해서 xdata에서 추출 --%>
<x:out select="$xdata//person[1]/name"/>
```

```
… 이하 생략 …
```

이 xdata 변수에 들어간 xml 에서 추출하는 방식은 XPath 를 사용한다고 했다.
$select="$xdata//person[1]/name" 에서 person[1] 은 person 으로 사용하는 것과 같다. [] 안에는 순서가 들어가고 생략될 경우 첫 엘리먼트를 찾기 때문이다.
〈x:forEache/〉 태그로 〈person〉 엘리먼트 수만큼 반복하게 했다. 이때 자동으로 기준은 〈person〉 이 되기 때문에 그 이후에 〈email〉 이나 〈phone〉 엘리먼트들은 /person/email 과 /person/phone 을 바로 참조하게 된다.