

09장. JSTL

JSTL 개요

1. JSTL은 JSP 표준 태그 라이브러리(JSP Standard Tag Library)의 약어이다.
2. 라이브러리란 여러 프로그램이 공통으로 사용하는 코드를 모아놓은 코드의 집합이다.
3. JSTL을 가지고 할 수 있는 일
 - 간단한 프로그램 로직의 구사
(자바의 변수 선언, if 문, for 문 등에 해당하는 로직)
 - 다른 JSP 페이지 호출(<c:redirect>, <c:import>)
 - 날짜, 시간, 숫자의 포맷
 - JSP 페이지 하나를 가지고 여러 가지 언어의 웹 페이지 생성
 - 데이터베이스로의 입력, 수정, 삭제, 조회
 - XML 문서의 처리
 - 문자열을 처리하는 함수 호출
4. 문자열을 처리하는 함수 호출을 제외한 나머지 기능들은 모두 커스텀 액션 형태로 제공된다.

JSTL 개요 (계속)

5. JSTL에 있는 `<c:forEach>` 커스텀 액션은 자바의 for 문과 비슷한 기능을 한다.

```
<c:forEach begin="1" end="10">
  <H5>안녕하세요, 여러분!</H5>
</c:forEach>
```

시작 태그와 끝 태그 사이에 있는 코드를
10번 반복해서 출력합니다.

```
<c:forEach items="${리스트가 받아올 배열이름}" var="for문 내부에서 사용할 변수"
  varStatus="상태용 변수">
```

```
  // 반복해서 표시할 내용 혹은 반복할 구문
</c:forEach>
```

이 때, 상태용 변수를 status라고 지정했다면 아래와 같이 활용할 수 있다.

`${status.current}` 현재 for문의 해당하는 번호
`${status.index}` 0부터의 순서
`${status.count}` 1부터의 순서
`${status.first}` 첫 번째인지 여부
`${status.last}` 마지막인지 여부
`${status.begin}` for문의 시작 번호
`${status.end}` for문의 끝 번호
`${status.step}` for문의 증가값

활용 예시

```
<c:forEach items="${list}" var="list" varStatus="status">
  <c:out value="${status.index}" /> / <c:out value="${status.end}" />
</c:forEach>
```

JSTL 개요 (계속)

6. JSTL에 있는 <fmt:formatNumber> 커스텀 액션은 수치 값을 포맷하는 기능

```
<fmt:formatNumber value= "3.14159 " pattern= "#.00 " />
```

주어진 수치를 소수점 이하 2자리까지 끊어서 출력합니다

7. JSTL에는 커스텀 액션만 있는 게 아니라 익스프레션 언어에서 사용할 수 있는 EL 함수도 있다.

```
$fn:toUpperCase( "Hello ")
```

이 함수는 'HELLO' 라는 문자열을 리턴합니다

JSTL을 구성하는 작은 라이브러리들

라이브러리	기능	URI 식별자	접두어
코어	일반 프로그래밍 언어에서 제공하는 것과 유사한 변수 선언, 실행 흐름의 제어 기능을 제공하고, 다른 JSP 페이지로 제어를 이동하는 기능도 제공합니다.	http://java.sun.com/jsp/jstl/core	c
포매팅	숫자, 날짜, 시간을 포매팅하는 기능과 국제화, 다국어 지원 기능을 제공합니다.	http://java.sun.com/jsp/jstl/fmt	fmt
데이터베이스	데이터베이스의 데이터를 입력/수정/삭제/조회하는 기능을 제공합니다.	http://java.sun.com/jsp/jstl/sql	sql
XML 처리	XML 문서를 처리할 때 필요한 기능을 제공합니다.	http://java.sun.com/jsp/jstl/xml	x
함수	문자열을 처리하는 함수를 제공합니다.	http://java.sun.com/jsp/jstl/functions	fn

JSTL 지시자 사용 방법

1. JSP 페이지에서 앞 페이지의 접두어를 사용하기 위해서는 taglib 지시자를 이용해서 라이브러리의 URI 식별자와 접두어를 연결해야 한다.
2. taglib 지시자는 다른 지시자와 마찬가지로 <%@으로 시작해서 %>로 끝난다.
3. taglib 지시자에는 uri와 prefix라는 두 개의 애트리뷰트를 써야 하고, 이 두 애트리뷰트에 각각 URI 식별자와 접두어를 값으로 주어야 한다.

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

↑
접두어

↑
라이브러리를 식별하는 URI

JSTL 다운로드 받기 |

<http://jstl.java.net/>

[Aquarium](#) [Core](#) [Webtier](#) [WS/XML](#) [Tools](#)

[Login](#) | [Register](#) | [Join](#) | [Help](#)



GlassFish » JSP Standard Tag Library

JSP

EL

JSTL

Downloads

Mailing list

Membership

Developing JSTL

IssueTracker

JSP

Jersey

Phobos

WADL

LRWP in Java

Governance Document

A community project

- **Licenses:** Common Development and Distribution License (CDDL) version 1.0 + GNU General Public License (GPL) version 2
- **Governance:** Same as Project GlassFish

This project provides an implementation of the Standard Tag Library for JavaServer Pages (JSTL) 1.2. The main goals are

- Improves current implementation: bug fixes and performance improvements
- Provides API for use by other tools, such as Netbeans



Get Started

Find out what JSTL is and get started.



Download JSTL

Get the latest version of JSTL.



Get Involved

Learn how to build/use JSTL.

[Blog](#) [Releases](#)

[Terms of Use](#) | [Privacy Policy](#) | Copyright ©2008-2011 (revision 20111104.8bf0245)

JSTL 다운로드 받기2

[Aquarium](#) [Core](#) [Webtier](#) [WS/XML](#) [Tools](#)

[Login](#) | [Register](#) | [Join](#) | [Help](#)

 **GlassFish » JSP Standard Tag Library**

JSP

EL

JSTL

JSF

Jersey

Phobos

WADL

LRWP in Java

Governance Document

Downloads

Mailing lists

Membership

Developing JSTL

IssueTracker

A [java.net](#) project

The current JSTL can be downloaded from the maven repositories:

- [JSTL API](#)
- [JSTL Implementation](#)

[Terms of Use](#); [Privacy Policy](#); Copyright ©2008-2011 (revision 20111104.8bf0245)

JSTL 설치

- ▲ ch09
 - ▷ Deployment Descriptor: ch09
 - ▷ JAX-WS Web Services
 - ▷ Java Resources
 - ▷ JavaScript Resources
 - ▷ build
 - ▲ WebContent
 - ▷ cstl
 - ▷ jatl
 - ▷ jstl
 - ▷ META-INF
 - ▲ WEB-INF
 - ▷ classes
 - ▲ lib
 - javax.servlet.jsp.jstl-1.2.1.jar
 - javax.servlet.jsp.jstl-api-1.2.1.jar
 - ▲ taglibs
 - tools.tld
 - ▷ tags

Libraries 설치

WEB-INF의 lib에
붙임

코어 라이브러리 사용하기

1. <c:set> 커스텀 액션의 사용 방법

<c:set>은 변수를 선언하고 초기값을 대입하는 커스텀 액션이다.

자바 프로그램에서 변수를 선언할 때는 기본적으로 변수의 타입과 이름을 기술하고, 선택적으로 초기값을 기술한다.

```
int num=100;
```

2. <c:set> 커스텀 액션을 이용해서 변수를 선언할 때는 변수의 타입을 쓰지 않는다.

```
<c:set var= "num " value= "100 " />
```

3. value 애트리뷰트 값 위치에 EL 식을 쓸 수도 있다.

```
<c:set var= "sum " value= "${num1+num2} " />
```

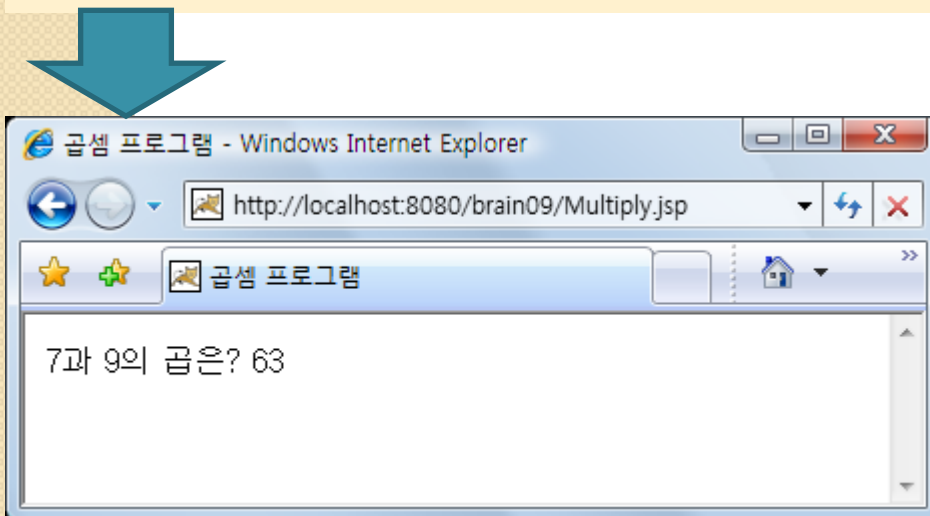
value 애트리뷰트 값으로 EL 식을 쓸 수도 있습니다

<c:set> 커스텀 액션의 사용 방법

Multiply.jsp

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@taglib prefix= "c " uri= "http://java.sun.com/jsp/jstl/core " %>
<c:set var= "num1 " value= "7 " />
<c:set var= "num2 " value= "9 " />
<c:set var= "result " value= "${num1*num2} " />
<HTML>
<HEAD><TITLE>곱셈 프로그램</TITLE></HEAD>
<BODY>
${num1}과 ${num2}의 곱은? ${result}
</BODY>
</HTML>
```

실행결과



코어 라이브러리 사용하기(<c:set> 커스텀 액션)

1. <c:set> 커스텀 액션의 사용 방법

<c:set> 액션을 이용해서 선언한 변수는 page 데이터 영역의 애트리뷰트가 된다.

<c:set> 태그에 scope 애트리뷰트를 추가하고 page, request, session, application 중 한 값을 지정하면 선언된 변수가 page, request, session, application 데이터 영역의 애트리뷰트가 되도록 지정하는 것도 가능하다.

```
<c:set var= "PRICE " value= "15000 " scope= "request " />
```

변수가 저장될 데이터 영역

2. scope 애트리뷰트에 request라는 값을 지정하고 나서 forward 메서드를 통해 다른 JSP 페이지를 호출하면 그 JSP 페이지 안에서도 선언된 변수를 사용할 수 있다

코어 라이브러리 사용하기(<c:set> 커스텀 액션) 예시

ProductInfo.jsp

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core " %>
<c:set var= "CODE" value= "80012 " scope= "request " />
<c:set var= "NAME" value= "온습도계 " scope= "request " />
<c:set var= "PRICE" value= "15000 " scope= "request " />
<jsp:forward page= "ProductInfoView.jsp " />
```

request 데이터
영역에 데이터를
저장

호출

ProductInfoView.jsp

```
<%@page contentType= "text/html; charset=utf-8"%>
<HTML>
  <HEAD> <TITLE>상품 정보</TITLE> </HEAD>
  <BODY>
    <H3>상품 정보</H3>
    상품코드: ${CODE} <BR>
    상품명: ${NAME} <BR>
    단가: ${PRICE}원 <BR>
  </BODY>
</HTML>
```

코어 라이브러리 사용하기 <c:remove>와 <c:if>

1. <c:remove> 커스텀 액션은 이런 애트리뷰트를 삭제하는 기능을 한다.

```
<c:remove var= "num " />
```

2. 위 코드는 page, request, session, application 데이터 영역에 저장되어 있는 num이라는 이름의 애트리뷰트를 모두 찾아서 제거한다. 특정 영역의 애트리뷰트만 제거하려면 scope 애트리뷰트를 사용하면 된다.

// request 데이터 영역에 있는 변수를 제거합니다

```
<c:remove var= "code " scope= "request " />
```

1 <c:if> 커스텀 액션의 사용 방법

① <c:if> 커스텀 액션은 자바 프로그램의 if 문과 비슷 한 역할을 한다.

② 자바 프로그램에서 if 문을 작성하는 방법은 다음과 같다

// 조건식의 결과가 true일 때만 실행되는 명령문

```
if (num1 > num2) {  
    System.out.println( "num1이 더 큽니다. ");  
}
```

③ <c:if> 커스텀 액션에서는 조건식을 괄호 안에 쓰는 것은 아니라, test라는 이름의 애트리뷰트 값으로 지정해야 한다

// 조건식의 결과가 true일 때만 실행되는 명령문

```
<c:if test= "${num1 > num2} ">  
    num1이 더 큽니다.  
</c:if>
```

코어 라이브러리 사용하기 예시1

NumberInput.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<% request.setCharacterEncoding("UTF-8");%>
<%
    int NUM1= 7, NUM2= 9;
%>
포워딩하는 페이지 Maxmum.jsp입니다.<br>
<jsp:forward page="Maxmum.jsp">
<jsp:param name="NUM1" value="<%=NUM1%>" />
    <jsp:param name="NUM2" value="<%=NUM2%>" />
</jsp:forward>
```



호출

Maximum.jsp

```
<%@page contentType= "text/html; charset=UTF-8"%>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core" %>
<HTML>
    <HEAD> <TITLE> 최대값 구하기 </TITLE> </HEAD>
    <BODY>
        최대값:
        <c:if test= "${param.NUM1 - param.NUM2 >= 0} ">
            ${param.NUM1}
        </c:if>
        <c:if test= "${param.NUM1 - param.NUM2 < 0} ">
            ${param.NUM2}
        </c:if>
    </BODY>
</HTML>
```

코어 라이브러리 <c:choose> 커스텀 액션

1. <c:choose> 커스텀 액션은 자바 프로그램의 switch 문과 비슷한 역할을 한다.
2. <c:when>, <c:otherwise>라는 커스텀 액션과 함께 사용되며, 두 커스텀 액션은 각각 switch 문의 case, default 절과 비슷한 역할을 한다.
3. 자바 프로그램의 switch 문의 문법은 다음과 같다.

적용

```
switch (num) {  
    case 0 :  
        System.out.println( "처음 뵙겠습니다. ");  
        break;  
    case 1 :  
        System.out.println( "반갑습니다. ");  
        break;  
    default :  
        System.out.println( "안녕하세요. ");  
        break;  
}
```

첫번째 조건 만족 실행 명령문

두번째 조건 만족 실행 명령문

아무 조건 만족하지 않을때
실행 명령문

코어 라이브러리 <c:choose> 커스텀 액션

1. c:choose> 커스텀 액션의 사용 방법

<c:choose> 커스텀 액션의 전체적인 구조는 switch 문과 비슷하지만, 변수의 이름이 아니라 조건식을 <c:when> 커스텀 액션을 test 애트리뷰트에 EL 식 형태로 지정

적용

<c:choose>

<c:when test= "\${num == 0} ">

처음 뵙겠습니다.

</c:when>

<c:when test= "\${num == 1} ">

반갑습니다.

</c:when>

<c:otherwise>

안녕하세요.

</c:otherwise>

</c:choose>

첫번째 조건 만족 실행 명령문

두번째 조건 만족 실행 명령문

아무 조건 만족 않을때
실행 명령문

코어 라이브러리<c:forEach> 커스텀 액션

<c:forEach> 커스텀 액션은 자바 프로그램의 for 문에 해당하는 기능을 제공한다.
이것을 이용하면 특정 HTML 코드를 지정된 횟수만큼 반복해서 출력

초기
값

반복
종료
기준
값

카운
터
증가
값

```
for (int cnt = 0; cnt < 10 ; cnt++) {  
    System.out.println("야호 " );  
}
```

반복 실행 명령문

<c:forEach> 액션을 사용할 때는 begin과 end라는 이름의 애트리뷰트를 쓰고, 거기에 각각 카운터 변수의 시작 값과 끝 값을 지정

적용

```
<c:forEach var= "cnt " begin= "1 " end= "10 " step= "2 ">  
    ${cnt} <BR>  
</c:forEach>
```

반복 실행 명령문

코어 라이브러리 <c:forTokens> 커스텀 액션

- 1.<c:forTokens> 커스텀 액션은 자바의 for 문과 java.util.StringTokenizer 클래스의 기능을 합친것 같은 기능을 제공한다.
- 2.이 액션에는 items, delims, var라는 3개의 애트리뷰트를 써야 한다. items 애트리뷰트에는 토큰을 포함하는 문자열을, delims 애트리뷰트에는 토큰 분리에 사용할 구획 문자를, var 애트리뷰트에는 분리된 토큰을 대입할 변수의 이름을 써야 함



```
<c:forTokens var= "pet " items= "햄스터 이구아나 소라게" delims= " " >  
    ${pet} <BR>  
</c:forTokens>
```

토큰의 구획 문자로 한 종류 이상의 문자를 지정할 수도 있다.

코어 라이브러리 <c:catch> 커스텀 액션

- 1.<c:catch> 커스텀 액션은 자바 프로그래밍 언어의 try문과 비슷한 기능을 한다.
- 2.<c:catch> 커스텀 액션의 시작 태그와 끝 태그 사이에서 에러가 발생하면 실행의 흐름이 곧바로 <c:catch> 액션 다음에 있는 코드로 넘어간다.

익셉션 개체를
저장할 변수

```
<c:catch var="e">
```

```
    <% int result = num1 / num2 %>
```

```
    나눗셈의 결과는? <%= result %>
```

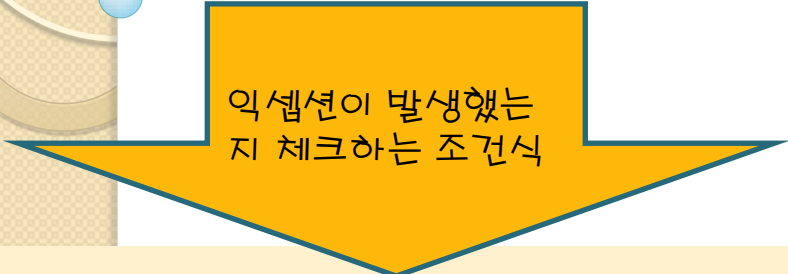
```
</c:catch>
```

Error가 발생할수 있는 부분

<c:catch> 커스텀 액션은 자바의 try 블록에 해당하는 일만 하기 때문에 catch 블록에 해당하는 일은 별도로 코딩


코어 라이브러리 <c:catch> 커스텀 액션 사용방법

- var 애트리뷰트에 지정된 변수(익셉션 객체가 저장되는 변수)는 <c:catch> 액션의 범위 밖에서도 EL 식을 통해 사용할 수 있으므로, 이를 이용해서 에러 처리를 하면 된다



익셉션이 발생했는지
체크하는 조건식

```
<c:if test= "${e != null} " >  
    에러 메시지: ${e.message}  
</c:if>
```



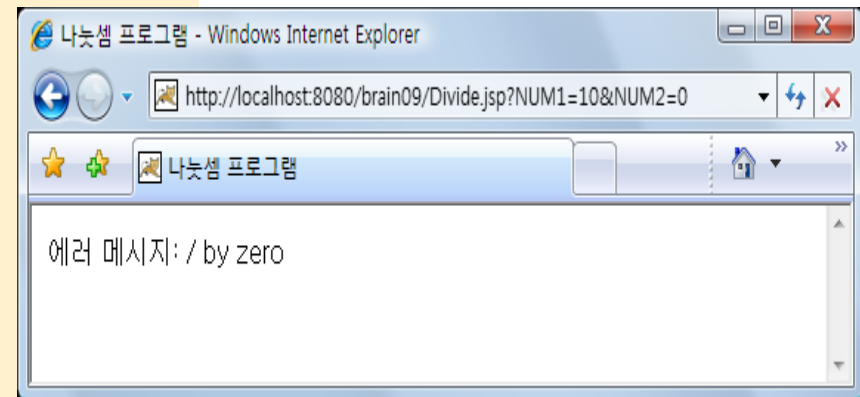
Error 출력 코드

`${e.message}`라는 EL 식은 익셉션 객체 `e`에 대해 `getMessage` 메서드를 호출하는 일을 한다.

코어 라이브러리 <c:catch> 커스텀 액션 사용방법

Divide.jsp

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core " %>
<%
    String str1 = request.getParameter( "NUM1 ");
    String str2 = request.getParameter( "NUM2 ");
    int num1 = Integer.parseInt(str1);
    int num2 = Integer.parseInt(str2);
%>
<HTML>
<HEAD> <TITLE>나눗셈 프로그램</TITLE> </HEAD>
<BODY>
    <c:catch var= "e ">
        <% int result = num1 / num2; %>
        나눗셈의 결과는? <%= result %>
    </c:catch>
    <c:if test= "${e != null} " >
        에러 메시지: ${e.message}
    </c:if>
</BODY>
</HTML>
```



코어 라이브러리 <c:redirect> 커스텀 액션 사용방법

1. <c:redirect> 커스텀 액션은 sendRedirect 메서드를 통해 다른 웹 자원을 호출하는 일을 한다.
호출할 웹 자원의 URL은 url 애트리뷰트를 이용해서 지정

```
<c:redirect url= "http://cafe.naver.com/fordeveloper" />
```

Redirect.jsp

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

코어 라이브러리 <c:import> 커스텀 액션 사용방법

1. <c:import> 커스텀 액션은 <jsp:include> 표준 액션과 비슷하지만 다른 JSP 페이지 뿐만 아니라 다른 종류의 웹 자원도 호출할 수 있다는 점이 다릅니다.
2. 호출할 웹 자원의 URL은 url 애트리뷰트를 이용해서 지정하면 됩니다.
3. <c:import url="http://www.daum.net" /> // 호출할 웹 자원의 URL
호출할 웹 자원에 데이터를 넘겨주려면 <c:import> 커스텀 액션의 시작 태그와 끝 태그 사이에 <c:param> 커스텀 액션을 쓰면 됩니다.

```
<c:import url="http://www.naver.com/">  
    <c:param name="product" value="TV" />  
    <c:param name="ad_index" value="007"/>  
</c:import>
```

↑ ↑
데이터 이름 데이터 값

코어 라이브러리 <c:url> 커스텀 액션 사용방법

1. <c:url> 커스텀 액션은 <c:set> 커스텀 액션과 마찬가지로 변수의 선언에 사용되지만, URL을 쉽게 다룰 수 있는 방법을 제공한다는 점이 다릅니다.
2. <c:url>의 기본적인 사용방법은 <c:set>과 동일 합니다. var 애트리뷰트에 변수 이름을 지정하고, value 애트리뷰트에 변수의 초기값을 지정하면 됩니다.
3. <c:url var= "myUrl " value= "http://localhost:8181/Add.jsp " >

↑ ↑
변수이름 변수 값

```
<html>
<body>
  <c:url var="myUrl" value="color.jsp"> </c:url>
  <c:redirect url="{myUrl }"> </c:redirect>
</body>
</html>
```

4. <c:url>의 시작 태그와 끝 태그 사이에 <c:param> 커스텀 액션을 쓰면, URL 뒤에 쿼리 스트링 형태로 덧붙는 데이터를 지정할 수 있습니다.

```
<c:url var= "myUrl " value= "http://localhost:8080/Add.jsp " >
  <c:param name= "NUM1 " value= "999 " />
  <c:param name= "NUM2 " value= "1 " />
</c:url>
```

```
<!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>Insert title here</title> </head>
<body>
  <c:url var="gugu" value="gugu.jsp">
    <c:param name="num" value="7"> </c:param>
  </c:url>
  <c:redirect url="{gugu }"> </c:redirect>
</body> </html>
```

코어 라이브러리 <c:out> 커스텀 액션 사용방법

1. <c:out> 커스텀 액션은 데이터를 출력할 때 사용하는 커스텀 액션인데, 웹 브라우저에 의해 특수한 문자로 해석되는 <, >, &, ', "를 포함하는 데이터를 출력할 때 편리.
2. 출력할 데이터는 value 애트리뷰트에 지정하면 됩니다.

<c:out value= "<INPUT>은 <FORM>의 서브엘리먼트입니다. " />



이 두 태그는 HTML 태그로 해석되지 않고, 웹 브라우저 상에 그대로 나타남

3. default 애트리뷰트를 이용하면 출력할 데이터의 디폴트 값을 지정할 수 있다. 이 방법은 EL 식의 결과를 출력할 때 유용.

<c:out value= "\${str}" default= "No Data " />



이 값이 없으면 이 값을 대신 출력합니다

out.jsp

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

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

```
<HTML>
```

```
<HEAD> <TITLE>HTML 문법 설명</TITLE> </HEAD>
```

```
<BODY>
```

```
    <H3>FONT 태그에 대하여</H3>
```

```
    <c:out value="<FONT size=7>커다란 글씨</FONT>는 다음과 같은 출력을 합니다." />
```

```
    <BR> <BR> <c:out value="<FONT size=7>커다란 글씨</FONT>" escapeXml="false" />
```

```
    <BR> <BR> 안녕하세요 <c:out value="${param.ID}" default="guest" /> 님
```

```
</BODY>
```

```
</HTML>
```

포매팅 라이브러리 사용하기

1. JSTL의 국제화/형식화 액션 : fmt - JSTL fmt란? . 국제화/형식화의 기능을 제공해 주는 JSTL 라이브러리 . 국제화는 다국어 내용을 처리, 형식화는 날짜와 숫자 형식등을 처리 .

JSTL의 국제화/형식화 액션 : fmt

- JSTL fmt란?

- . 국제화/형식화의 기능을 제공해 주는 JSTL 라이브러리
- . 국제화는 다국어 내용을 처리, 형식화는 날짜와 숫자 형식등을 처리

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

Fmt라이브러리 태그 알아보기

- 인코딩 관련 태그 : <fmt:requestEncoding>
- 국제화 관련 태그 : <fmt:setLocale>, <fmt:timeZone>, <fmt:setTimeZone>, <fmt:bundle>, <fmt:setBundle>, <fmt:message>, <fmt:param>
- 형식화 관련 태그 : <fmt:formatNumber>, <fmt:parseNumber>, <fmt:formatDate>, <fmt:parseDate>

포매팅 라이브러리 사용하기

1. <fmt:requestEncoding>

- Request 객체로 부터 전달 받은값들을 인코딩할 때 사용
- 보통 한글 값이 넘어올 경우 <%request.setCharacterEncoding("UTF-8");%> 코드로 Request 객체로 받는 값을 한글에 맞게 인코딩

```
<fmt:requestEncoding value="인코딩값">
```

2. <fmt:setLocale>

- 다국어 페이지를 사용할 때 언어를 지정하는 태그
- value 속성은 어떤 언어를 사용할지 지정하는 속성이며 언어코드를 입력
- variant 속성은 브라우저의 스펙을 지정할 때 사용

```
<fmt:setLocale value="값" variant="" scope="범위">
```

3. <fmt:timeZone>

- 지정한 지역값으로 시간대를 맞추어 사용할 때 사용
- <fmt:setTimeZone>는 특정 페이지 전체에 적용이 되지만, <fmt:timeZone> 태그는 첫태그와 끝태그 사이의 Body 부분의 내용에 적용

```
<fmt:timeZone value="값">
```

포매팅 라이브러리 사용하기

1. <fmt:setTimeZone>

- 지정한 지역값으로 시간대를 맞추는 기능(페이지 전체에 작용)
- var속성의 변수에 지정한 시간대 값이 저장
- scope로 범위를 지정

```
<fmt:setTimeZone value="값" var="변수명" scope="범위">
```

2. <fmt:bundle>

- properties 확장명을 가진 파일의 리소스를 불러 올때 사용
- basename에는 properties 확장명을 가진 파일을 지정
- prefix는 properties 내의 키값에 쉽게 접근할 수 있도록 간략한 접근어를 사용할 수 있게 해 준다.
- 불러온 리소스는 이태그의 Body내에서만 사용할 수 있다.

```
<fmt:bundle basename="basename" prefix="prefix">
```

3. <fmt:setBundle>

- <fmt:bundle> 태그와 같은 기능을 하는 태그
- <fmt:setBundle> 태그는 사용범위가 페이지 전체에 적용
- var에는 설정한 리소스 내용을 가지고 있을 변수명을 입력
- scope로 범위를 지정

```
<fmt:setBundle basename="basename" var="변수명" scope="범위">
```

포매팅 라이브러리 사용하기

4. <fmt:message>

- 설정한 properties 파일의 리소스 내용을 읽을 때 사용
- properties 파일에는 여러키 값들로 내용이 구분되어 있으며, <fmt:message> 태그의 key 속성으로 키값으로 설정된 내용을 가져올 수 있다.
- bundle 속성에는 <fmt:setBundle> 태그에서 var속성에 지정했던 변수를 입력

```
<fmt:message key="키값" bundle="bundle변수" var="변수명" scope="범위">
```

5. <fmt:param>

- <fmt:message> 태그로 읽어온 리소스 내용에 파라미터를 전달하는 태그

6. <fmt:formatNumber>

- 태그는 숫자형식의 패턴을 설정할 때 사용
- value 속성에는 패턴을 적용 시킬 원래의 값을 입력
- type은 숫자의 타입을 의미. 숫자, 통화, 퍼센트중 원하는 타입으로 설정 가능
- pattern 속성은 지정한값을 어떤 패턴으로 변화 시킬지를 정 할수 있다.
- currencyCode는 통화코드를 의미하며, 숫자타입이 통화일 경우만 유효하다.
- currencySymbol도 숫자타입이 통화일 유효하며, 통화기호를 지정할 수 있다.
- groupingUsed는 그룹기호의 포함 여부를 나타낸다.
- maxIntegerDigits는 최대정수 길이
- minIntegerDigits는 최소 정수 길이
- maxFractionDigits은 최대 소수점 자릿수
- minFractionDigits는 최소 소수점 자릿수를 의미

포매팅 라이브러리 사용하기

```
<fmt:formatNumber value="값" type="타입" pattern="패턴" currencyCode="값" currencySymbol="값"
groupingUsed="True 또는 False" maxIntegerDigits="값" minIntegerDigits="값" maxFractionDigits="
값" minFractionDigits="값" var="변수명" scope="범위">
```

<fmt:parseNumber>

- 문자열을 숫자, 통화 또는 퍼센트의 형태로 변환할 때 사용하는 태그
- value 속성에 문자열 값을 입력
- type에는 숫자, 통화, 퍼센트중 변환할 타입을 설정
- pattern 속성은 value 속성의 값을 설정된 타입으로 변환하면서 어떤 패턴을 갖게 할 것인지를 지정할 수 있다.
- parseLocale은 파싱할 기본 패턴을 지정
- integerOnly는 지정된 값중 정수부분만 해석할 지의 여부를 지정하는 속성

```
<fmt:parseNumber value="값" type="타입" pattern="패턴" parseLocale="값" integerOnly="True
또는 False" var="변수명" scope="범위">
```

포매팅 라이브러리 사용하기

5. <fmt:formatDate>

- 날짜 형식의 패턴을 설정할 때 사용되는 태그
- value 속성에는 날짜 또는 시간을 입력 - type 속성으로 날짜인지 시간인지 또는 날짜와 시간 둘다 포함한 타입인지를 지정
- dateStyle은 날짜의 스타일을 지정. type 속성이 date 또는 both일 때만 적용
- timeStyle은 시간의 스타일을 지정. type 속성이 time 또는 both일 때만 적용
- timeZone속성은 날짜와 시간이 표시될 시간대를 지정

```
<fmt:formatDate value="값" type="타입" dateStyle="값" timeStyle="값" pattern="패턴" timeZone="값" var="변수명" scope="범위">
```

6. <fmt:parseDate>

- 문자열을 날짜와 시간의 형태로 변환하는 태그
- value 속성에 입력된 문자열값을 type 속성에 지정된 타입으로 날짜와 시간의 형태로 변환
- 나머지속성은 앞에서 설명한 <fmt:formatDate>의 속성과 기능 동일
- <fmt:parseDate>는 문자열값을 파싱하여 날짜 , 시간형태로 변환

```
<fmt:parseDate value="값" type="타입" dateStyle="값" timeStyle="값" pattern="패턴" timeZone="값" parseLocale="값" var="변수명" scope="범위">
```


포매팅 라이브러리 사용하기

7. 날짜와 시각을 포맷하는 <fmt:formatDate> 커스텀 액션

- 1) <fmt:formatDate>는 날짜와 시각을 포맷하는 커스텀 액션이다.
- 2) 이 액션에는 출력할 날짜와 시각을 java.util.Date 클래스 타입의 객체로 넘겨줘야 하므로 먼저 이 클래스의 객체를 만들어야 한다.

Date date = new Date(); // 현재의 날짜와 시각을 포함한 Date 객체를 생성

- 3) <fmt:formatDate> 커스텀 액션의 value 애트리뷰트에 Date 객체를 지정하면 그 객체가 포함하고 있는 날짜가 YYYY. MM. DD 포맷으로 출력된다.

<fmt:formatDate value= "\${date}" /> // Date 객체

- 1) dateStyle 애트리뷰트 full, long, medium, short 중 한 값을 넘겨주면 날짜를 다른 포맷으로 출력할 수 있다.

// 날짜를 '2009년 5월 5일 (화)' 포맷으로 출력하도록 지시

<fmt:formatDate type= "date" value= "\${date}" dateStyle= "long" />

- 2) timeStyle 애트리뷰트에 full, long, medium, short 중 한 값을 넘겨주면 시각도 다른 포맷으로 출력.

// 시각을 '오후 1시 31분 42초 KST' 포맷으로 출력하도록 지시

<fmt:formatDate type= "time" value= "\${date}" timeStyle= "full" />

- 3) type 애트리뷰트에 both 값을 지정해서 날짜와 시각을 한꺼번에 출력할 때는 dateStyle과 timeStyle 애트리뷰트를 함께 씀

// 날짜를 '2009년 5월 5일 (화)' 포맷으로, 시각을 '오후 2:50' 포맷으로 출력하도록 지시

<fmt:formatDate type= "date" value= "\${date}" dateStyle= "long" />

포매팅 라이브러리 - 수치 포맷

1. 출력할 수치 값은 `<fmt:formatNumber>`의 `value` 애트리뷰트에 지정하면 된다.
`<fmt:formatNumber value= "10000 " />` // 출력할 수치 Data
2. 세 자리마다 쉼표를 찍은 포맷으로 출력하려면 `groupingUsed`라는 애트리뷰트를 추가하고, 그 값으로 `true`를 지정
// 주어진 값을 '1,234,500' 포맷으로 출력하도록 지시
`<fmt:formatNumber value= "1234500" groupingUsed= "true" />`
3. `pattern` 애트리뷰트를 사용하면 소수점 아래의 숫자를 원하는 만큼 끊거나 늘려서 표시.
// 주어진 값을 소수점 아래 2자리까지 끊어서 출력하도록 지시
`<fmt:formatNumber value= "3.14158 " pattern= "#.## " />`
4. `pattern` 애트리뷰트의 값에서 0이라고 쓴 위치는 표시할 유효숫자가 없으면 0으로 채워진다.
// 주어진 값을 소수점 아래 2자리까지 끊어서 출력하도록 지시
`<fmt:formatNumber value= "10.5 " pattern= "#.00 " />`
5. `type` 애트리뷰트에 `percent`라는 값을 지정하면 주어진 수치를 퍼센트 단위로 표시.
// 주어진 수치를 퍼센트 단위로 포맷하여 출력하도록 지시
`<fmt:formatNumber value= "0.5" type= "percent" />`
6. `type` 애트리뷰트에 `currency`라는 값을 지정하면 주어진 수치가 금액에 적합한 포맷으로 만들어져서 출력.
// 주어진 수치를 금액으로 표시하여 출력하도록 지시
`<fmt:formatNumber value= "2500000" type= "currency" />`

포매팅 라이브러리 - 지역 설정

1. `<fmt:setLocale>` 커스텀 액션은 출력할 데이터의 포맷을 특정 지역에 맞게 설정하고자 할 때 사용하는 액션이다.
2. `<fmt:setLocale>` 커스텀 액션을 이용해서 특정 지역을 설정하기 위해서는 `value` 애트리뷰트에 언어 코드 또는 국가코드_언어코드를 지정하면 된다
`<fmt:setLocale value= "en" />` // 언어코드
`<fmt:setLocale value= "us_en" />` // 국가 코드 및 언어 코드
3. 위의 액션이 실행되고 나면 날짜와 시각이 영어권에 맞게 포맷되고, `<fmt:formatNumber>` 액션을 이용해서 출력되는 모든 금액 앞에는 달러를 의미하는 \$기호가 자동으로 붙어서 표시
4. 시간대를 설정하는 `<fmt:timeZone>`과 `<fmt:setTimeZone>` 커스텀 액션
`<fmt:timeZone>`과 `<fmt:setTimeZone>`은 시간대마다 다른 날짜와 시각을 자동으로 계산해서 표시하기 위해 필요한 커스텀 액션.
`<fmt:timeZone>` 커스텀 액션의 시작 태그에 `value` 애트리뷰트를 쓰고 거기에 특정 시간대에 해당하는 지역 이름을 지정하면, 이 액션의 시작 태그와 끝 태그 사이에서 출력되는 날짜와 시각은 그 시간대에 맞게 표시

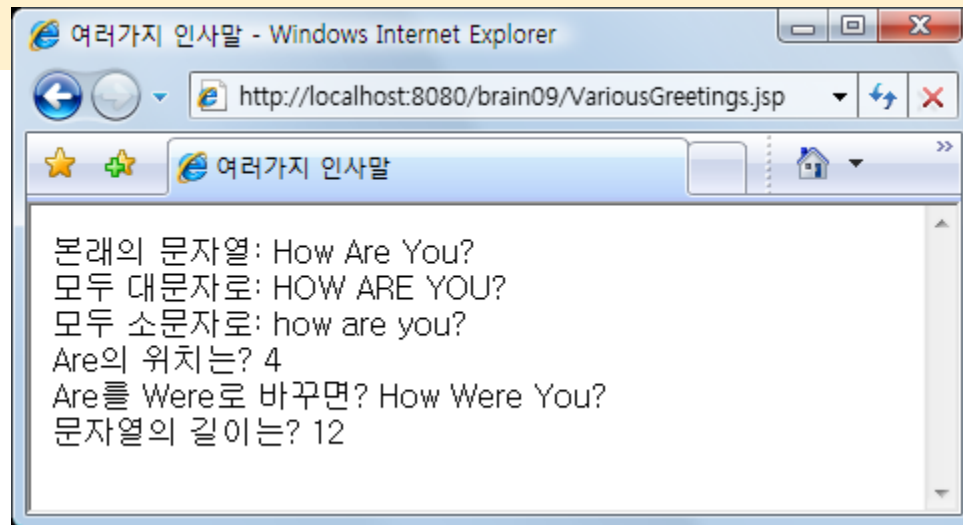
JSTL의 라이브러리에 있는 함수

함수	제공하는 기능
substring(str, index1, index2)	str의 index1부터 index2 - 1까지의 부분자열을 리턴
substringAfter(str1, str2)	str1에서 str2를 찾아서 그 후의 부분자열을 리턴
substringBefore(str1, str2)	str1에서 str2를 찾아서 그 전의 부분자열을 리턴
toUpperCase(str)	모든 소문자를 대문자로 치환한 값을 리턴
toLowerCase(str)	모든 대문자를 소문자로 치환한 값을 리턴
trim(str)	문자열에서 앞뒤 공백 문자를 제거한 결과를 리턴
replace(str, src, dest)	str 문자열에 포함된 src를 모두 dest로 치환한 결과를 리턴
indexOf(str1, str2)	str1에 포함된 str2의 시작 인덱스를 리턴
startsWith(str1, str2)	str1이 str2로 시작하면 true, 그렇지 않으면 false를 리턴
endsWith(str1, str2)	str1이 str2로 끝나면 true, 그렇지 않으면 false를 리턴
contains(str1, str2)	str1이 str2를 포함하면 true, 그렇지 않으면 false를 리턴
containsIgnoreCase(str1, str2)	str1이 str2를 포함하면 true, 그렇지 않으면 false를 리턴. contains 함수와는 달리 대소문자를 구별하지 않고 비교함
split(str1, str2)	str1을 str2를 기준으로 분리해서 만든 부분자열들의 배열을 리턴
join(arr, str2)	arr 배열의 모든 항목을 합쳐서 리턴. 항목 사이에는 str2가 들어감
escapeXml(str)	HTML 문법에 의해 특수 문자로 취급되는 모든 문자(표 9-2 참조)를 이스케이프 시퀀스로 치환한 결과를 리턴
length(obj)	obj가 문자열이면 문자열의 길이, List나 Collection이면 항목의 수를 리턴

JSTL의 라이브러리에 있는 함수 예시

VariousGreetings.jsp

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@page import= "java.util.* "%>
<%@taglib prefix= "c" uri= "http://java.sun.com/jsp/jstl/core " %>
<%@taglib prefix= "fn" uri= "http://java.sun.com/jsp/jstl/functions " %>
<c:set var= "greeting " value= "How Are You? " />
<HTML>
  <HEAD> <TITLE>여러가지 인사말 </TITLE> </HEAD>
  <BODY>
    본래의 문자열: ${greeting} <BR>
    모두 대문자로: ${fn:toUpperCase(greeting)} <BR>
    모두 소문자로: ${fn:toLowerCase(greeting)} <BR>
    Are의 위치는? ${fn:indexOf(greeting, "Are ")} <BR>
    Are를 Were로 바꾸면? ${fn:replace(greeting, "Are ", "Were ")} <BR>
    문자열의 길이는? ${fn:length(greeting)} <BR>
  </BODY>
</HTML>
```



커스텀 액션을 만드는 방법

1. 커스텀 액션을 만드는 방법

1) 커스텀 액션을 만드는 방법은 크게 두 가지로 나뉜다.

- 태그 파일을 작성해서 만드는 방법
- 태그 클래스를 작성해서 만드는 방법

2) 태그 파일(**tag file**)이란 **JSP 페이지와 비슷한 문법으로 작성하는 텍스트 파일**

1. 태그 파일의 예시

```
<%@tag body-content= "scriptless " %>
<TABLE border=1 cellpadding=20>
  <TR>
    <TD>
      <jsp:doBody/>
    </TD>
  </TR>
</TABLE>
```

JSP 페이지와
유사한 문법

커스텀 액션을 만드는 방법

2. 태그 파일도 JSP 페이지처럼 웹 컨테이너의 특정 디렉터리에 저장해 놓기만 하면 바로 사용할 수 있고, 이렇게 간단한 설치 방법은 태그 파일의 장점

```
<%@tag body-content="tagdependent" %>
<TABLE border=1 cellpadding=20>
  <TR>
    <TD>
      <jsp:doBody/>
    </TD>
  </TR>
</TABLE>
</BODY>
```

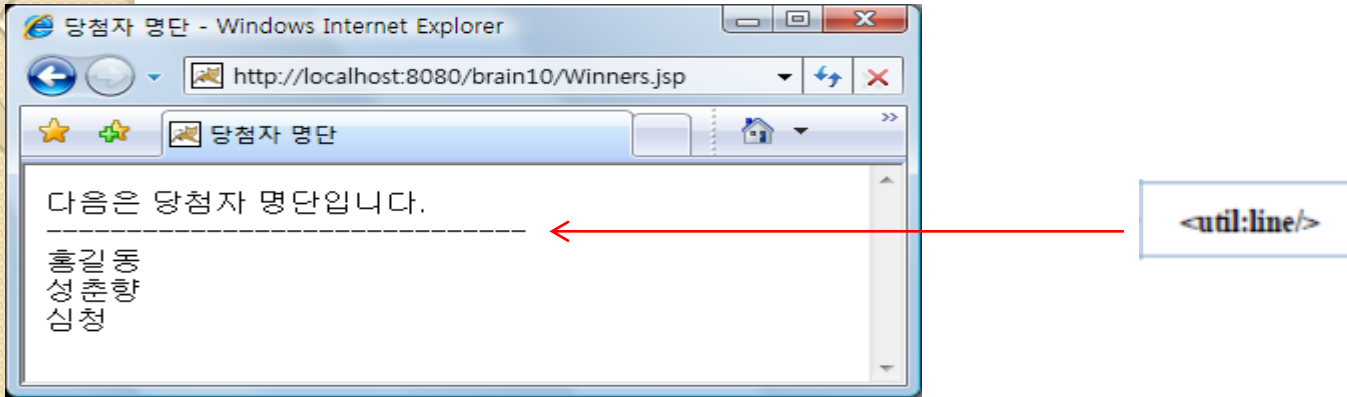
① 태그 파일을 작성합니다.

② 태그 파일을 웹 컨테이너의 디렉터리에 저장합니다.



커스텀 액션을 만드는 방법

1. 태그 파일을 이용해서 커스텀 액션을 만드는 방법
2. 다음과 같이 웹 페이지에 선을 출력하는 태그 파일을 만들어보자



3. 태그 파일을 이용해서 만든 커스텀 액션은 기본적으로 본체를 가질 수 있다.
커스텀 액션의 본체란 시작 태그와 끝 태그 사이에 오는 부분
`<util:box>야유회가 취소되었습니다.</util:box>` // 커스텀 액션의 본체(body)
4. 커스텀 액션이 본체를 갖기 않도록 만들기 위해서는 태그 파일에 다음과 같은 **tag** 지시자를 써야 한다.
`<%@tag body-content= "empty" %>` // 커스텀 액션이 본체를 가질 수 없음을 표시
5. 위와 같은 **tag** 지시자가 있으면 커스텀 액션을 사용할 때 다음과 같이 처리
`<util:line>절취선</util:line>` // 문법 에러가 발생
`<util:line/>` // 정상 처리

커스텀 액션을 만드는 방법(아주 간단한 태그 파일)

1. 태그 파일에는 몇 가지 지시자를 더 사용

이름	역할
tag 지시자	웹 컨테이너가 태그 파일을 처리할 때 필요한 정보를 기술
include 지시자	다른 태그 파일을 포함
taglib 지시자	태그 파일에서 사용할 다른 커스텀 액션의 태그 라이브러리(tag library)에 대한 정보를 기술
attribute 지시자	커스텀 액션의 애트리뷰트에 대한 정보를 기술
variable 지시자	커스텀 액션의 변수에 대한 정보를 기술

2. include 지시자와 taglib 지시자는 JSP 페이지에서와 똑같은 역할을 하고 나머지 세 지시자는 태그 파일에서만 사용될 수 있는 것들이다.

3. JSP 페이지에서 커스텀 액션을 사용하려면 **taglib** 지시자를 써야 함

- 태그 파일을 이용해서 만든 커스텀 액션일 경우에는 **taglib** 지시자에 앞 장에서 배웠던 **uri** 애트리뷰트 대신 **tagdir** 애트리뷰트를 써야 한다.

```
<%@taglib prefix="util" tagdir="/WEB-INF/tags" %>
```

↑ ↑
접두어 태그 파일이 있는 디렉터리의 경로명

4. tablib 지시자를 쓴 다음에는 접두어와 커스텀 액션의 이름을 이용해서 커스텀 액션을 사용
커스텀 액션의 이름은 태그 파일의 이름에서 .tag를 제외한 나머지 부분.

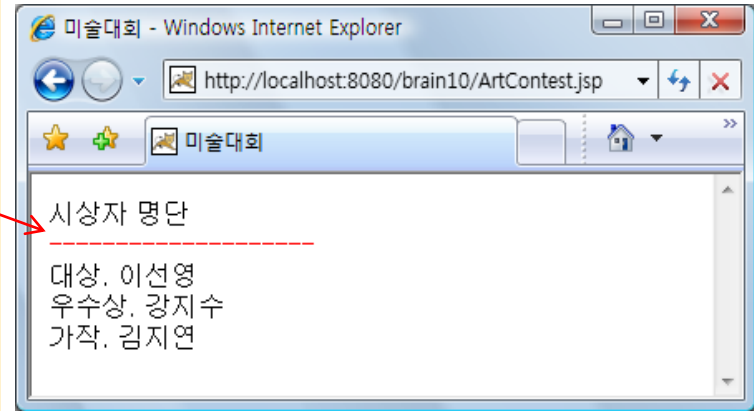
```
<util:line/>              // 커스텀 액션의 이름 (line)
```

커스텀 액션을 만드는 방법(애트리뷰트를 지원하는 태그 파일)

1. 다음과 같은 애트리뷰트가 있는 커스텀 액션 만들어보자
`<util:newLine color="red" size="20" />`

// 20개의 마이너스 기호로 이루어진 빨간색 선을 출력

애트리뷰트가 있는 커스텀 액션을 만들기 위해서는 태그 파일에 각각의 애트리뷰트를 위한 **attribute** 지시자를 써야 함



2. 애트리뷰트를 지원하는 태그 파일

- attribute 지시자의 name 애트리뷰트는 커스텀 액션의 애트리뷰트 이름을 지정하는 역할을 한다.

`<%@attribute name="color" %>` // **애트리뷰트의 이름**

3. 커스텀 액션의 애트리뷰트 값은 태그 파일은 태그 파일로 전달되며, 그 값을 사용하는 방법은 두 가지.
하나는 스크립팅 요소 안에서 자바 변수처럼 사용하는 방법이고, 다른 하나는 익스프레션 언어의 EL 식 안에서 데이터 이름으로 사용하는 방법.

1) `<%= color %>` // 애트리뷰트의 이름

2) `${color}` // 애트리뷰트의 이름

커스텀 액션을 만드는 방법(애트리뷰트를 지원하는 태그 파일)

1. Tag File의 문제점

- <util:newLine> 커스텀 액션은 color와 size 애트리뷰트를 모두 썼을 때는 정상적으로 작동하지만, size 애트리뷰트를 쓰지 않으면 선이 전혀 출력되지 않는다.

2. 이런 문제는 size 애트리뷰트를 필수 애트리뷰트로 만들어서 해결할 수 있다. 방법은 attribute 지시자에 required 애트리뷰트를 추가하고 그 값으로 true를 지정

// 필수 애트리뷰트임을 표시

```
<%@attribute name="size" type="java.lang.Integer" required="true" %>
```

2. 동적 애트리뷰트를 지원하는 태그 파일

- 애트리뷰트 각각을 위해 attribute 지시자를 쓰는 대신 모든 애트리뷰트를 한꺼번에 선언하려면 동적 애트리뷰트를 선언하면 된다.
- 동적 애트리뷰트를 선언하려면 태그 파일의 tag 지시자에 dynamic-attributes 애트리뷰트를 쓰고, 그 값으로 커스텀 액션의 모든 애트리뷰트를 대표할 이름을 쓰면 된다.

```
<%@tag dynamic-attributes="attrs"%> // 동적 애트리뷰트의 대표 이름
```

```
    ${attrs.color} // 동적 애트리뷰트의 대표 이름 & 실제로 사용된 애트리뷰트의 이름
```

커스텀 액션을 만드는 방법(동적 애트리뷰트를 지원하는 태그 파일)

1. 커스텀 액션의 애트리뷰트를 담고 있는 Map 객체는 page 데이터 영역을 통해 태그 파일에 전달되는데, 그 객체는 jspContext 내장 변수에 대해 getAttribute라는 메서드를 호출해서 가져올 수 있다.

```
Map attrs = (Map) jspContext.getAttribute( "attrs "); // 동적 애트리뷰트의 대표 이름
```

2. 동적 애트리뷰트의 경우에는 모든 애트리뷰트의 값이 문자열 형태로 저장되므로, Map 객체에 대해 get 메서드를 호출할 때 리턴값을 String 타입으로 변환해서 받아야 한다.

```
String str = (String) attrs.get( "size "); // 개별적인 애트리뷰트 이름
```

3. 문자열 형태의 애트리뷰트 값은 다음과 같은 방법을 이용해서 다른 데이터 타입으로 변환해서 사용할 수도 있다

```
int size = Integer.parseInt(str); // 필요하다면 가져온 애트리뷰트 값의 타입을 변환
```

2. 동적 애트리뷰트를 지원하는 태그 파일

```
<%@tag body-content= "empty " %>
```

```
<%@tag dynamic-attributes= "attrs "%>
```

```
<FONT color=${attrs.color} >
```

```
<%
```

```
    java.util.Map attrs = (java.util.Map) jspContext.getAttribute( "attrs ");
```

```
    String str = (String) attrs.get( "size ");
```

```
    int size = Integer.parseInt(str);
```

```
    for (int cnt = 0; cnt < size; cnt++) {
```

```
        out.print( "= ");
```

```
    }
```

```
%>
```

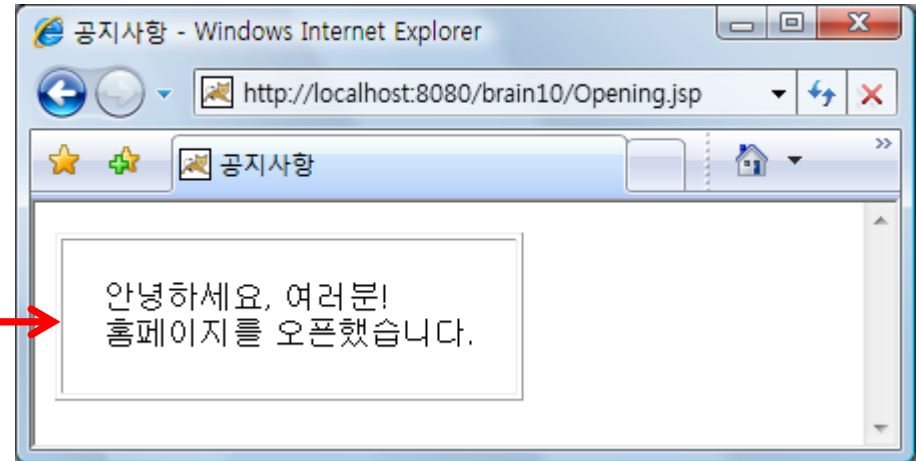
```
</FONT> <BR>
```

// 위의 태그 파일을 톰캣의 WEB-INF/tags/util 디렉터리에 doubleLine.tag라는 이름으로 저장

커스텀 액션의 본체를 처리하는 태그 파일

1. 커스텀 액션의 시작 태그와 끝 태그 사이에 오는 내용을 커스텀 액션의 본체(body)라고 함

```
<util:box>  
안녕하세요, 여러분! <BR>  
홈페이지를 오픈했습니다.  
</util:box>
```



2. 본체가 있는 커스텀 액션을 만들기 위해서는 태그 파일의 tag 지시자에 있는 body-content 애트리뷰트에 empty라는 값 대신 scriptless나 tagdependent라는 값을 써야 한다.

// 커스텀 액션의 본체에 스크립틀릿을 쓸 수 없음을 표시

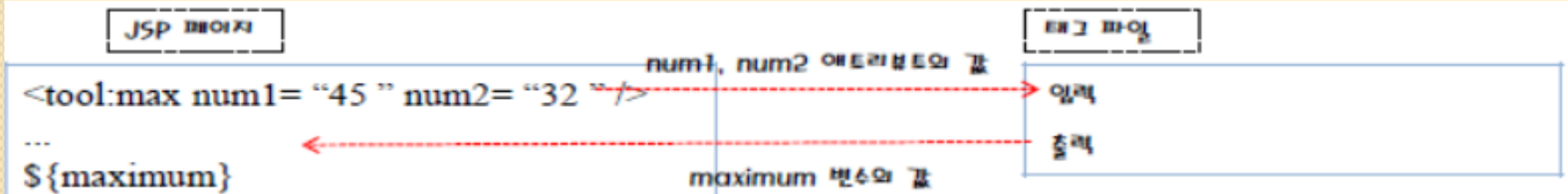
```
<%@tag body-content="scriptless" %>
```

만드는 커스텀 액션을 사용하는 JSP 페이지

```
<%@page contentType="text/html; charset=utf-8"%>  
<%@taglib prefix="util" tagdir="/WEB-INF/tags/util" %>  
<HTML>  
<HEAD><TITLE>공지사항</TITLE></HEAD>  
<BODY>  
    구내 식당에서 알려드립니다.  
    <util:box>  
        공사 관계로 급식을 일시 중단합니다. <BR>  
    </util:box>  
</BODY>  
</HTML>
```

변수를 지원하는 커스텀 액션

1. 변수를 지원하는 커스텀 액션을 만들려면 태그 파위에 **variable** 지시자를 써야 한다.
variable 지시자를 기술하는 방법은 다음과 같다
`<%@variable name=given= "result" %> // 변수의 이름`
2. String 타입이 아닌 다른 타입의 변수를 선언하려면 variable 지시자에 variable-class라는 애트리뷰트를 추가해서 타입을 지정하면 된다.
`<%@variable name=given= "result " variable-class= "java.lang.Integer" %> // 변수의 타입`
3. 주의할 점 - variable-class 애트리뷰트에는 프리미티브 타입을 지정할 수 없으므로, 래퍼 클래스 타입을 대신 사용해야 한다.



-위 그림처럼 작동하는 커스텀 액션을 만들기 위해서는 커스텀 액션이 끝난 다음에 maximum이라는 변수를 사용할 수 있도록 만들어야 한다. 그런 일은 variable 지시자에 scope라는 애트리뷰트를 추가해서 할 수 있다.

-scope 애트리뷰트는 변수의 사용 범위를 지정하는 역할을 하며, NESTED, AT_BEGIN, AT_END 중 한 값을 지정할 수 있다.

```
<%@variable name=given= "result " variable-class= "java.lang.Integer " scope= "AT_END " %>
```

단독으로 사용되는 태그 다음에 변수를 사용할 수 있도록 만들려면 AT_END 값을 지정하는 것이 가장 적합하다

변수를 지원하는 커스텀 액션

1. variable 지시자를 이용해서 선언한 변수에 값을 대입하려면 앞 장에서 배웠던 <c:set> 커스텀 액션을 사용해야 한다.

```
<c:set var= "result " value= "100 " /> // 변수의 이름 & 변수에 대입할 값
```

2. 태그 파일 안에서 변수에 저장한 값은 JSP 페이지로 전달되고, JSP 페이지에서 그 값을 가져다가 사용할 수 있다

3. 커스텀 액션의 결과를 리턴하는 maximum 변수의 이름이 태그 파일 안에 고정되어 있다. 이 문제를 해결하기 위해 다음과 같이 애트리뷰트를 이용해서 변수의 이름을 지정하도록 만들어보자.

```
<util:max var= "maximum " num1= "37 " num2= "42 " /> // 변수이름
```

4. 변수 이름을 저장할 애트리뷰트를 선언할 때는 지켜야 하는 규칙

- 첫째, 필수 애트리뷰트로 만들어야 한다.
- 둘째, 애트리뷰트 값에 스크립팅 요소나 익스프레션 언어를 사용할 수 없도록 만들어야 한다

```
<%@attribute name= "var " required= "true " rtexprvalue= "false "%>
```

↑
애트리뷰트
이름

↑
필수 애트리뷰트임을
표시

↑
애트리뷰트 값으로 스크립팅 요소나
익스프레션 언어를 사용할 수 없음을 표시

변수를 지원하는 커스텀 액션

1. 애트리뷰트를 이용해서 변수 이름을 지정할 때는 variable 지시자도 다르게 기술해야 한다.
첫째, name-given 애트리뷰트를 이용해서 변수의 이름을 지정하는 것이 아니라, name-from-attribute 애트리뷰트를 이용해서 변수의 이름을 지정할 애트리뷰트의 이름을 지정해야 한다.
둘째, 태그 파일 안에서 사용할 변수 이름을 따로 선언해야 한다

```
2. <%@variable name-from-attribute= "var " alias= "maximum "
variable-class= "java.lang.Integer " scope= "AT_END "%>
```

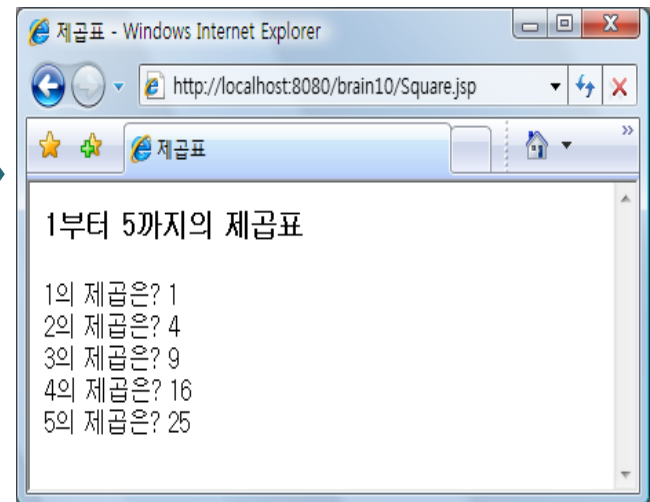
↑
변수 이름을 지정하기 위해
사용할 애트리뷰트 이름

↑
태그 파일 안에서 사용할
변수 이름

3. 이번에는 다음과 같이 커스텀 액션의 본체 안에서 변수를 사용하는 예제를 작성해보자

```
<util:compute var= "num " start= "1 " end= "5 ">
${num}의 제곱은? ${num * num} <BR>
</util:compute>
```

이 커스텀 액션은 **start** 애트리뷰트부터 **end** 애트리뷰트까지 변화하는 값을 본체 안에서 사용할 수 있도록 만들어야 한다.




변수를 지원하는 커스텀 액션

1. 이 경우에는 variable 지시자의 scope 애트리뷰트 값을 **NESTED** 로 지정해야 한다

변수를 지원하는 태그 파일 (3)

```
<%@tag pageEncoding= "utf-8"%>
<%@tag body-content= "scriptless " %>
<%@taglib prefix= "c " uri= "http://java.sun.com/jsp/jstl/core " %>
<%@attribute name= "var " required= "true " rtexprvalue= "false "%>
<%@attribute name= "start " type= "java.lang.Integer "%>
<%@attribute name= "end " type= "java.lang.Integer "%>
<%@variable name-from-attribute= "var " alias= "number "
variable-class= "java.lang.Integer " scope= "NESTED "%>
<% for (int cnt=start; cnt <= end; cnt++) { %>
    <c:set var= "number " value= "<%= cnt %> " />
    <jsp:doBody/>
<% } %>
```



* 몸체를 전달하는 방법은 <jsp:body> 태그를 이용하는 방법 과
몸체 내용을 직접 삽입하는 방법

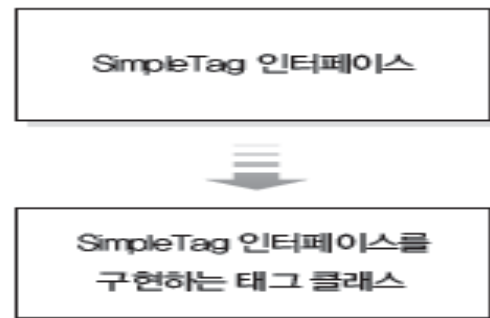
태그 클래스를 이용해서 커스텀 액션을 만드는 방법

1. 태그 클래스를 만드는 방법

• JSP 1.2의 태그 클래스 작성 방법 •



• JSP 2.0의 태그 클래스 작성 방법 •



2. javax.servlet.jsp.tagext패키지에서제공하는인터페이스

인터페이스	설명
BodyTag	Body가 있는 태그를 구현할 때 사용한다.
IterationTag	반복적인 작업을 구현할 때 사용한다.
SimpleTag	Tag와 IterationTag의 내용을 동시에 구현할 수 있다.
Tag	일반적인(Body가 없는) 태그를 구현할 때 사용한다.

3. TagSupport 클래스가 제공하는 주요메소드

인터페이스 메소드	설명
doStartTag()	시작 태그를 만날 때 실행된다.
doEndTag()	끝 태그를 만날 때 실행된다.
setPageContext(PageContext arg0)	전달받은 PageContext 객체를 저장할 때 쓰인다.
getPageContext()	저장해 놓은 PageContext 객체를 얻을 때 쓰인다.

커스텀 액션을 만드는 방법

1. BodyTagSupport클래스가제공하는주요메소드

인터페이스 메소드	설명
doAfterBody()	doStartTag()에서 EVAL_BODY_INCLUDE가 리턴될 경우 실행된다.
doStartTag()	시작 태그를 만날 때 실행된다.
doEndTag()	끝 태그를 만날 때 실행된다.
doInitBody()	Body 내용을 확인하기 전에 실행된다.

2. SimpleTagSupport클래스가제공하는주요메소드

인터페이스 메소드	설명
doTag()	시작 태그 또는 끝 태그를 만날 때 실행된다.
getJspBody()	Body를 처리하기 위한 JspFragment 객체를 얻는다.
getJspContext()	저장되어 있는 JspContext 객체를 얻는다.

커스텀 액션을 만드는 방법

1. BodyTagSupport클래스가제공하는주요메소드

인터페이스 메소드	설명
doAfterBody()	doStartTag()에서 EVAL_BODY_INCLUDE가 리턴될 경우 실행된다.
doStartTag()	시작 태그를 만날 때 실행된다.
doEndTag()	끝 태그를 만날 때 실행된다.
doInitBody()	Body 내용을 확인하기 전에 실행된다.

2. SimpleTagSupport클래스가제공하는주요메소드

인터페이스 메소드	설명
doTag()	시작 태그 또는 끝 태그를 만날 때 실행된다.
getJspBody()	Body를 처리하기 위한 JspFragment 객체를 얻는다.
getJspContext()	저장되어 있는 JspContext 객체를 얻는다.

커스텀 액션을 만드는 방법

1. SimpleTagSupport 클래스를 이용해서 태그 클래스를 작성하는 방법

- SimpleTagSupport 클래스를 이용해서 태그 클래스를 만들 때는 반드시 다음과 같은 형식의 doTag 메서드를 작성해야 한다.

```
public void doTag() throws JspException, IOException {  
      
}  
}
```

이 위치에 커스텀 액션이
해야 할 일을 기술해야 합니다.

2. 웹 브라우저로 HTML 코드를 출력하기 위해서는 우선 다음과 같은 방법으로 JspContext 객체를 구해야 한다.

```
JspContext context = getJspContext();  
JspWriter out = context.getOut();
```

JSP 페이지에 관한 여러 가지 정보가 들어 있는
JspContext 객체를 리턴하는 메서드

JspWriter 객체를 리턴하는 메서드

3. JspContext 객체에 대해 print나 println 메서드를 호출하면 웹 브라우저로 HTML을 출력할 수 있다.

```
out.println( "-----  
<BR> ");
```

웹 브라우저로 선을 출력합니다

커스텀 액션을 만드는 방법

1. **SimpleTagSupport** 클래스를 이용해서 태그 클래스를 작성하는 방법
다음은 선을 출력하는 커스텀 액션을 구현하는 태그 클래스이다

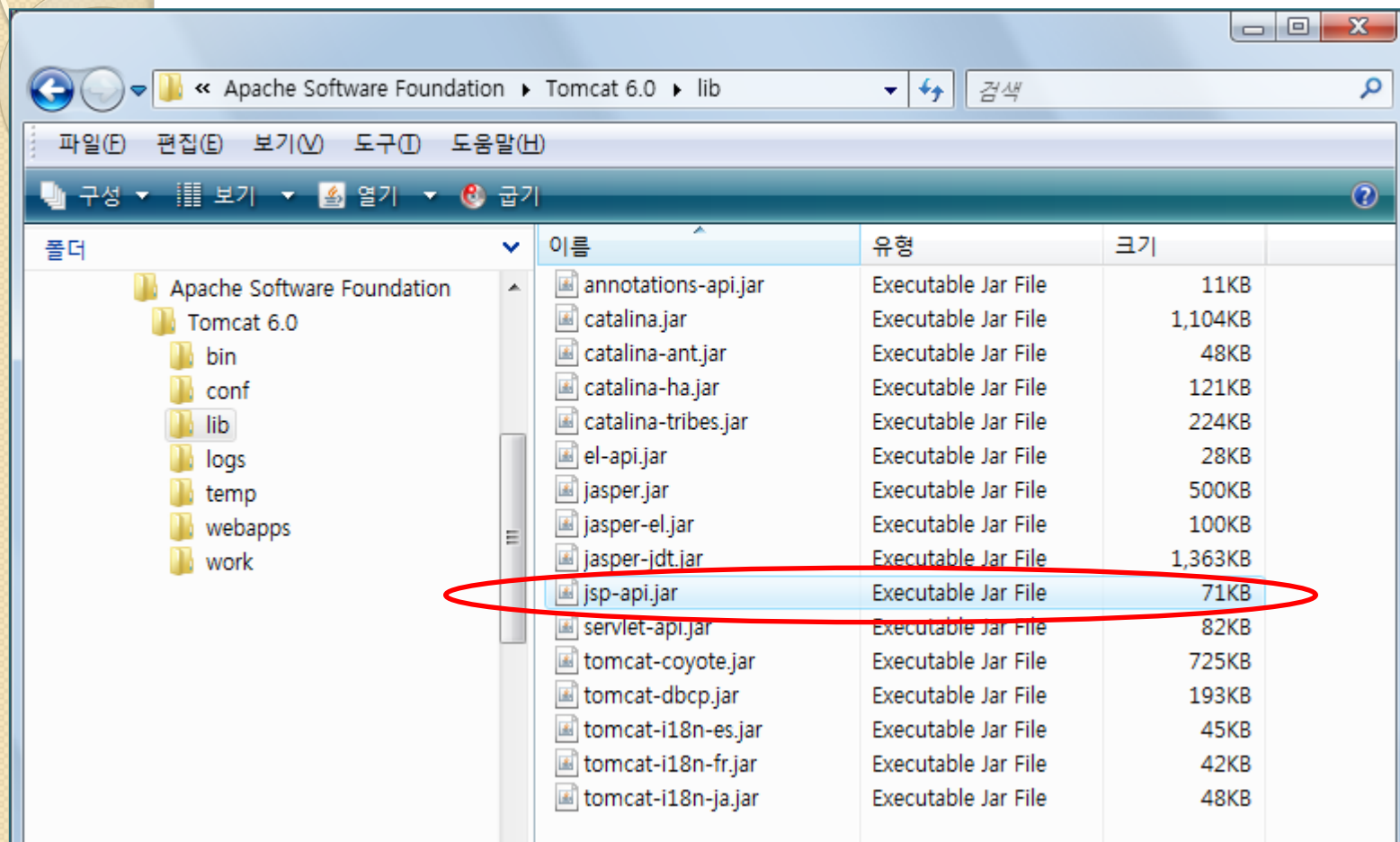
아주 간단한 태그 클래스

```
package tool;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class StarLineTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        out.println( "*****<BR> ");
        return;
    }
}
```

커스텀 액션을 만드는 방법

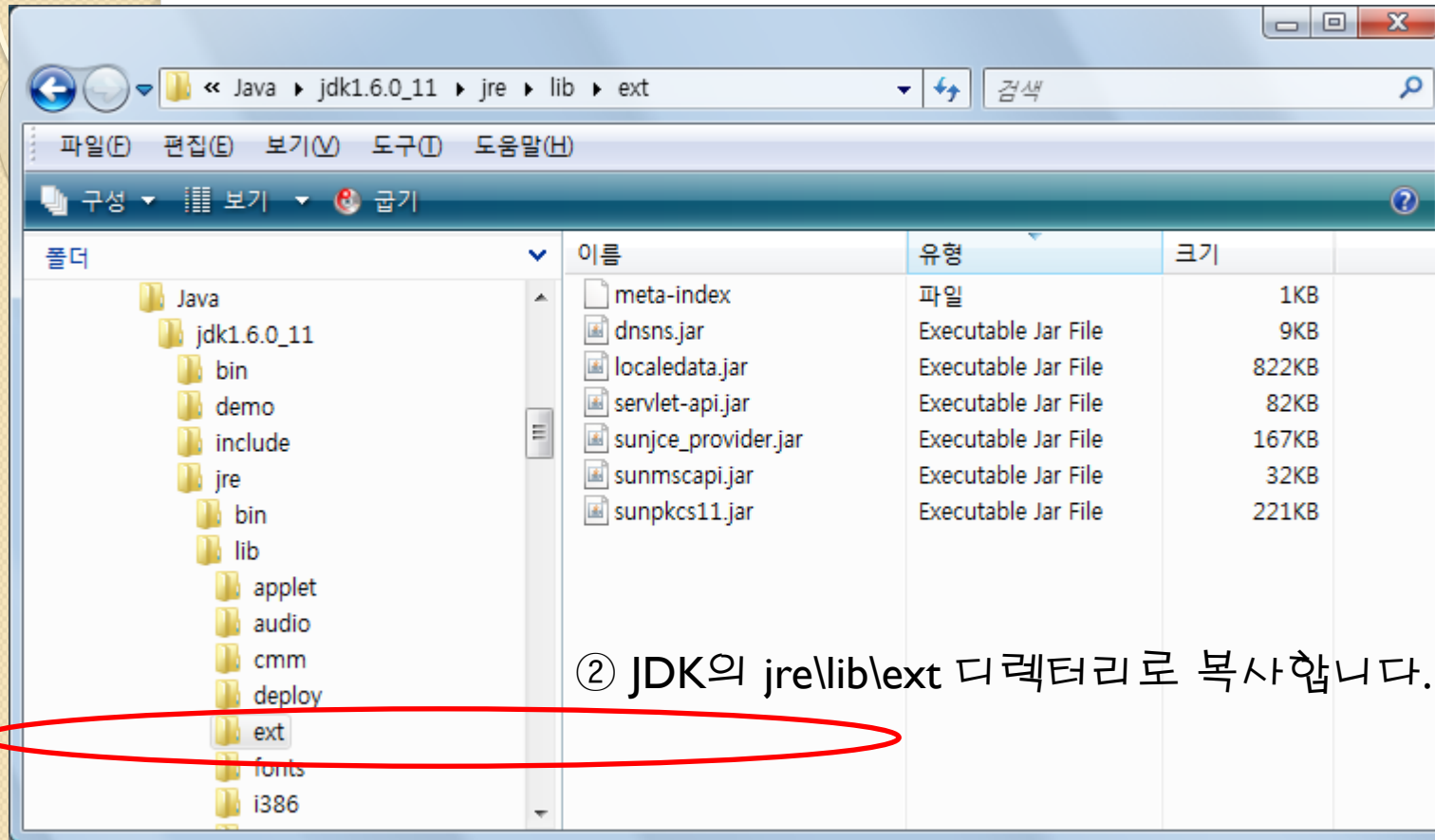
SimpleTagSupport 클래스를 이용해서 태그 클래스를 작성하는 방법

- jsp-api.jar 파일을 JDK의 확장 라이브러리 디렉터리에 설치하기



커스텀 액션을 만드는 방법

SimpleTagSupport 클래스를 이용해서 태그 클래스를 작성하는 방법



SimpleTagSupport 클래스를 이용해서 태그 클래스를 작성하는 방법

1. 태그 클래스를 설치한 다음에는 등록을 해야 함
2. TLD 파일에 태그 클래스를 등록하기 위해서는 다음과 같은 형식의 <tag> 엘리먼트를 추가해야 한다

```
<tag>  
  <name>starLine</name>           <- 커스텀 액션의 이름  
  <tag-class>tool.StarLineTag</tag-class> <- 태그 클래스의 이름  
  <body-content>empty</body-content> <- 본체의 형태  
</tag>
```

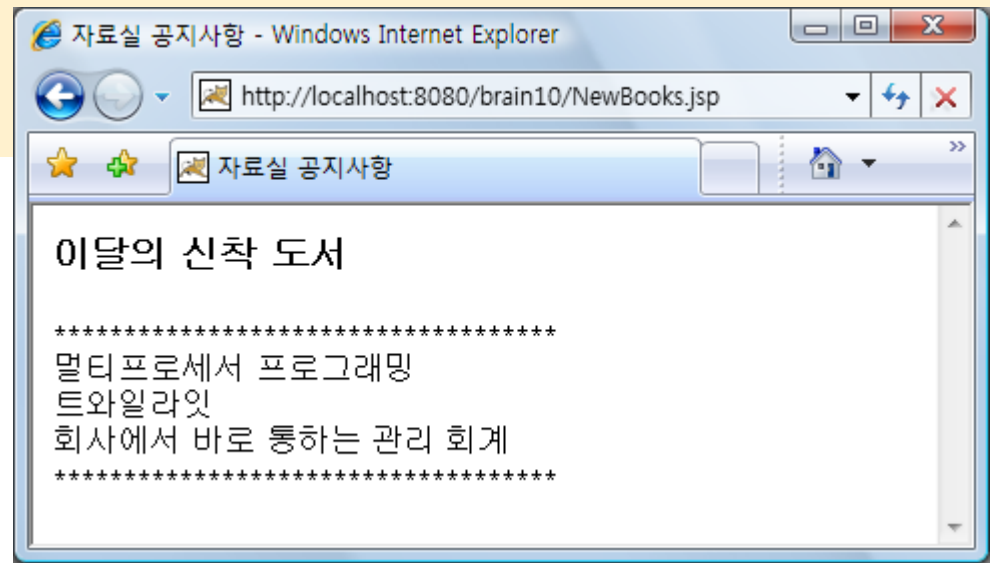
1. <tag> 엘리먼트는 TLD 파일의 루트 엘리먼트인 <taglib> 엘리먼트의 바로 아래에 위치해야 한다

```
<taglib xmlns= "http://java.sun.com/xml/ns/javaee " version= "2.1 ">  
<tlib-version>1.0</tlib-version>  
<short-name>tool</short-name>  
<tag>  
  <name>starLine</name>  
  <tag-class>tool.StarLineTag</tag-class>  
  <body-content>empty</body-content>  
</tag>  
</taglib>
```

SimpleTagSupport 클래스를 이용해서 태그 클래스를 작성하는 방법

태그 클래스가 만드는 커스텀 액션을 사용하는 JSP 페이지

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@taglib prefix= "tool " uri= "/taglibs/tools.tld " %>
<HTML>
<HEAD> <TITLE>자료실 공지사항</TITLE> </HEAD>
<BODY>
    <H3>이달의 신착 도서</H3>
    <tool:starLine/>
        멀티프로세서 프로그래밍<BR>
        트와일라잇<BR>
        회사에서 바로 통하는 관리 회계<BR>
    <tool:starLine/>
</BODY>
</HTML>
```




애트리뷰트가 있는 커스텀 액션을 만드는 태그 클래스

1. 태그 클래스를 이용해서 애트리뷰트가 있는 커스텀 액션을 구현하는 방법

- 1) 애트리뷰트 값을 받는 **public** 메서드를 선언해야 한다.
- 2) 이 메서드의 이름은 **set**으로 시작해야 하고, 그 다음에 애트리뷰트의 본래 이름에서 첫 글자를 대문자로 바꾼 이름을 붙여서 만들어야 한다.

color 애트리뷰트 값을
받는 메서드의 이름

이 파라미터 변수를 통해
애트리뷰트 값이 전달



`public void setColor(String color) {
}`

1.set-메서드가 받은 애트리뷰트 값은 필드(field, 클래스의 멤버 변수)에 저장.

```
public class NewLineTag extends SimpleTagSupport {  
    private String color;  
    ...  
    public void setColor(String color) {  
        this.color = color; // 애트리뷰트 값을 필드에 저장  
    }  
}
```

저장된 애트리뷰트 값은 나중에 doTag 메서드 안에서 사용되는 것이 보통이다

애트리뷰트가 있는 커스텀 액션을 만드는 태그 클래스

애트리뷰트를 지원하는 태그 클래스

```
package tool; import java.io.*; import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class NewLineTag extends SimpleTagSupport {
    private int size;
    private String color;
    public void setSize(Integer size) { // size 애트리뷰트를 받는 메서드
        this.size = size;
    }
    public void setColor(String color) { // color 애트리뷰트를 받는 메서드
        this.color = color;
    }
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        out.println( "<FONT color= " + color + "> ");
        for (int cnt = 0; cnt < size; cnt++) {
            out.print( "* ");
        }
        out.println( "</FONT><BR> ");
        return;
    }
}
```

애트리뷰트가 있는 커스텀 액션을 만드는 태그 클래스

1. 애트리뷰트를 지원하는 태그 클래스를 TLD 파킷에 등록할 때는 <tag> 엘리먼트 안에 <attribute>라는 서브 엘리먼트를 써야 한다

```
<tag>
```

```
  <name>newLine</name>
```

```
  <tag-class>tool.NewLineTag</tag-class>
```

```
  <body-content>empty</body-content>
```



이 위치에 <attribute> 엘리먼트를 추가해야 합니다.

```
</tag>
```

2. <attribute> 엘리먼트 안에는 <name>와 <type>라는 두 개의 서브엘리먼트를 써야 하고, 그 안에 각각 애트리뷰트의 이름과 타입을 표시

```
<attribute>
```

```
  <name>size</name>
```



애트리뷰트의 이름

```
  <type>java.lang.Integer</type>
```



애트리뷰트의 타입

```
</attribute>
```

```
<attribute>
```

```
  <name>color</name>
```



애트리뷰트의 이름

```
  <type>java.lang.String</type>
```

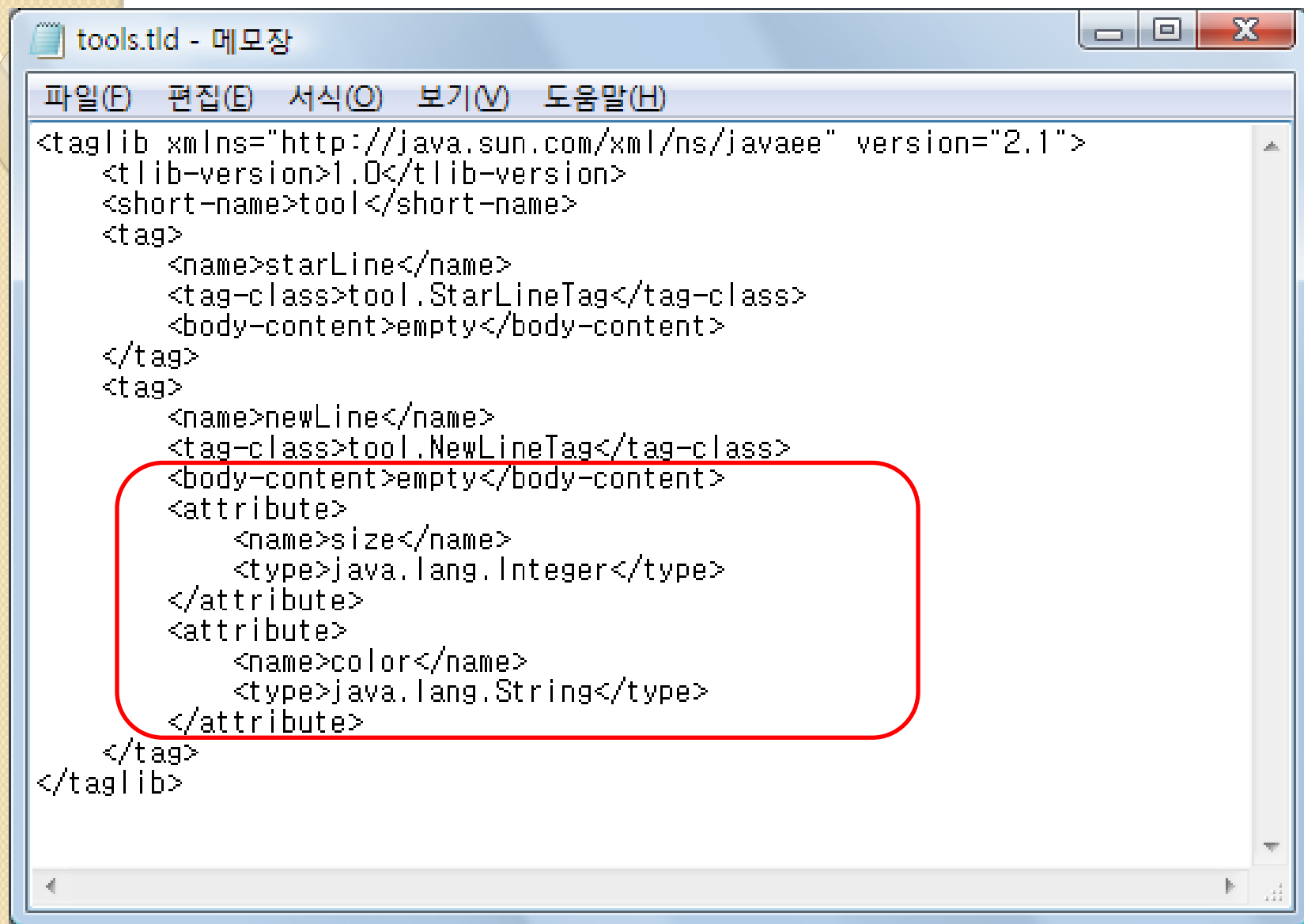


애트리뷰트의 타입

```
</attribute>
```

애트리뷰트가 있는 커스텀 액션을 만드는 태그 클래스

태그 클래스를 TLD 파일에 등록하는 방법

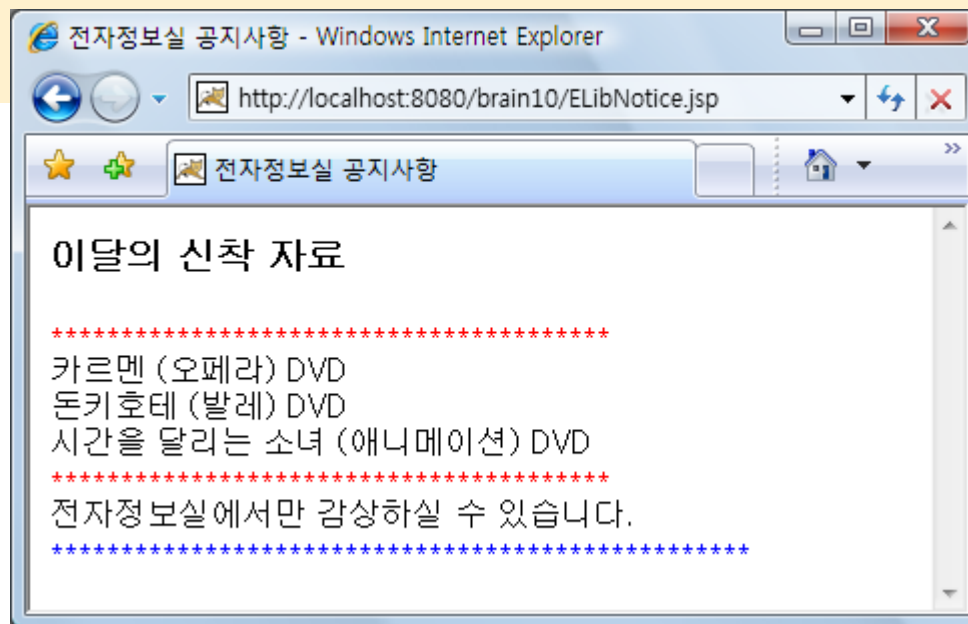


```
tools.tld - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
<taglib xmlns="http://java.sun.com/xml/ns/javaee" version="2.1">
  <tlib-version>1.0</tlib-version>
  <short-name>tool</short-name>
  <tag>
    <name>starLine</name>
    <tag-class>tool.StarLineTag</tag-class>
    <body-content>empty</body-content>
  </tag>
  <tag>
    <name>newLine</name>
    <tag-class>tool.NewLineTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>size</name>
      <type>java.lang.Integer</type>
    </attribute>
    <attribute>
      <name>color</name>
      <type>java.lang.String</type>
    </attribute>
  </tag>
</taglib>
```

애트리뷰트가 있는 커스텀 액션을 만드는 태그 클래스

만드는 커스텀 액션을 사용하는 JSP 페이지

```
<%@page contentType= "text/html; charset=utf-8"%>
<%@taglib prefix= "tool " uri= "/taglibs/tools.tld " %>
<HTML>
  <HEAD> <TITLE> 전자정보실 공지사항 </TITLE> </HEAD>
  <BODY>
    <H3> 이달의 신작 자료 </H3>
    <tool:newLine color= "red " size= "40 " />
    카르멘 (오페라) DVD<BR>
    돈키호테 (발레) DVD<BR>
    시각을 달리는 소녀 (애니메이션) DVD<BR>
    <tool:newLine color= "red " size= "40 " />
    전자정보실에서만 감상하실 수 있습니다.<BR>
    <tool:newLine color= "blue " size= "50 " />
  </BODY>
</HTML>
```



동적 애트리뷰트를 지원하는 태그 클래스

1. 동적 애트리뷰트를 지원하는 태그 클래스를 만들기 위해서는 **setDynamicAttribute**라는 이름의 메서드 하나만 선언하면 된다. 이 메서드는 다음과 같은 3개의 파라미터를 받는다.

```
public void setDynamicAttribute(String uri,    String localName,    Object value) throws JspException {  
}
```

↑
애트리뷰트의 이름이
속하는 네임스페이스의 URI

↑
애트리뷰트의
이름

↑
애트리뷰트의
값

2. 네임스페이스(namespace)란 똑같은 이름의 충돌을 방지하기 위해서 만들어 놓은 이름 공간이다.
3. 특정 네임스페이스에 속하지 않는 애트리뷰트일 경우에 이 메서드의 첫 번째 파라미터 값은 **null**.

동적 애트리뷰트를 지원하는 태그 클래스

애트리뷰트의 네임스페이스는 잘 사용되지 않으므로, 대개의 경우 `setDynamicAttribute` 메서드 안에서는 첫 번째 파라미터 값을 무시하고 두 번째와 세 번째 파라미터 값만 저장해 놓아도 충분.

// 애트리뷰트의 이름과 값을 Map 객체에 저장

```
public class NewerLineTag extends SimpleTagSupport implements DynamicAttributes {  
    private Map<String,Object> attrs = new HashMap<String,Object>();  
    ...  
    public void setDynamicAttribute(String uri, String localName, Object value) throws JspException {  
        attrs.put(localName, value);  
    }  
}
```

```
public class NewerLineTag extends SimpleTagSupport implements  
DynamicAttributes {  
    private Map<String,Object> attrs = new HashMap<String,Object>();  
    ...  
    public void doTag() throws JspException, IOException {  
        ...  
        // Map 객체에 저장되어 있는 애트리뷰트 값을 가져옵니다  
        String color = (String) attrs.get( "color " );  
        ...  
    }  
}
```

동적 애트리뷰트를 지원하는 태그 클래스

동적 애트리뷰트를 지원하는 태그 클래스

```
package tool; import java.io.*; import java.util.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class NewerLineTag extends SimpleTagSupport implements DynamicAttributes {
    private Map<String,Object> attrs = new HashMap<String,Object>();
    public void setDynamicAttribute(String uri, String localName, Object value) throws
JspException {
        attrs.put(localName, value);
    }
    public void doTag() throws JspException, IOException {
        String color = (String) attrs.get( "color " );
        int size = Integer.parseInt((String) attrs.get( "size " ));
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        out.println( "<FONT color= " + color + "> ");
        for (int cnt = 0; cnt < size; cnt++) {
            out.print( "* ");
        }
        out.println( "</FONT> <BR> ");
        return;
    }
}
```

본체가 있는 커스텀 액션을 만드는 태그 클래스

1. 태그 클래스를 이용해서 본체가 있는 커스텀 액션을 만들기 위해서는 다음과 같은 두 단계의 작업을 해야 한다.

첫째, `getJspBody` 메서드를 이용해서 커스텀 액션의 본체 내용을 가져온다.

// 커스텀 액션의 본체를 가져오는 메서드

JspFragment body = getJspBody();

둘째, `JspFragment` 객체를 이용해서 본체의 내용을 출력한다.

// JspFragment 객체의 내용을 출력하는 메서드

body.invoke(out);

2. `invoke` 메서드를 호출할 때는 파라미터로 `null` 값을 넘겨줄 수도 있다.

// `null`을 넘겨주면 JSP 페이지와 동일한 출력 스트림을 통해 본체의 내용이 출력

body.invoke(null);

본체가 있는 커스텀 액션을 만드는 태그 클래스

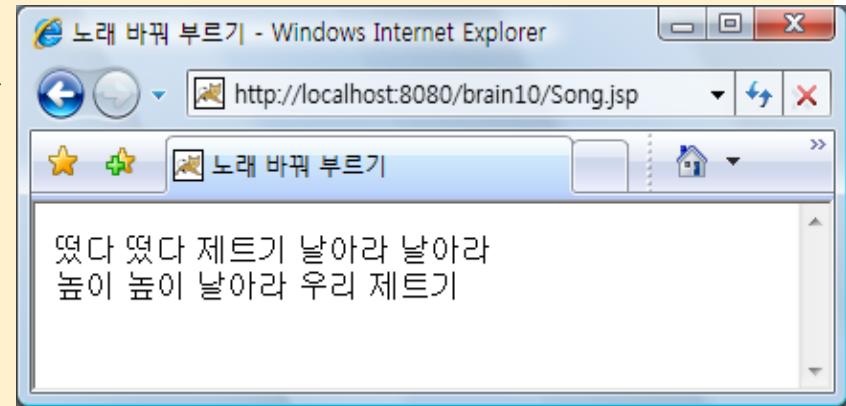
// 본체를 지원하는 태그 클래스

```
package tool;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class BoxTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        JspFragment body = getJspBody();
        out.println( "<TABLE border=1 cellpadding=20> <TR> <TD> ");
        body.invoke(out);
        out.println( "</TD> </TR> </TABLE> ");
        return;
    }
}
```

커스텀 액션의 본체 내용을 조작하는 태그 클래스

// 이번에는 다음 그림처럼 커스텀 액션의 본체 내용을 조작해서 출력하는 커스텀 액션을 만들어보자

```
<util:replace oldWord= "비행기 " newWord= "제트기 ">  
    떴다 떴다 비행기 날아라 날아라 <BR>  
    높이 높이 날아라 우리 비행기<BR>  
</util:replace >
```



이런 커스텀 액션을 만들기 위해서는 태그 클래스 안에서 커스텀 액션의 본체 내용을 가져다가 문자열로 만든 다음, 그 문자열을 조작해서 출력하면 된다. 이런 일은 자바 표준 라이브러리의 **java.io.StringWriter** 클래스를 이용해서 할 수 있음.

커스텀 액션의 본체 내용을 조작하는 태그 클래스

첫째, `getJspBody` 메서드를 호출해서 커스텀 액션의 본체 내용을 가져온다.

// 커스텀 액션의 본체를 가져옵니다.

```
JspFragment body = getJspBody();
```

둘째, `JspFragment` 객체에 대해 `invoke` 메서드를 호출할 때 파라미터로 `StringWriter` 객체를 넘겨준다.

```
StringWriter writer = new StringWriter(); // StringWriter 객체를 생성
```

```
body.invoke(writer); // 본체의 내용을 StringWriter 객체 안으로 출력
```

셋째, `toString` 메서드를 이용해서 `StringWriter` 객체 안의 내용을 문자열로 만든다.

// StringWriter 객체 안에 있는 내용을 문자열로 만듭니다

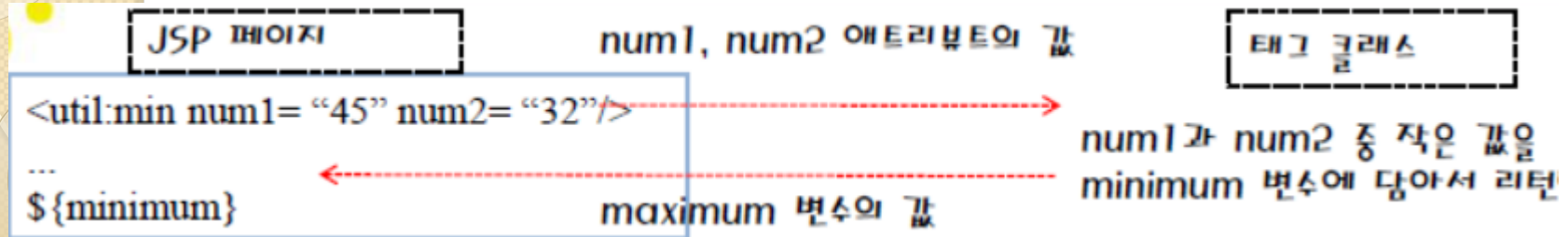
```
String str = writer.toString();
```

커스텀 액션의 본체 내용을 조작하는 태그 클래스

// 커스텀 액션의 본체 내용을 조작하는 태그 클래스

```
package tool; import java.io.*; import javax.servlet.jsp.*; import javax.servlet.jsp.tagext.*;
public class ReplaceTag extends SimpleTagSupport {
    private String oldWord, newWord;
    public void setOldWord(String oldWord) {
        this.oldWord = oldWord;
    }
    public void setNewWord(String newWord) {
        this.newWord = newWord;
    }
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        JspFragment body = getJspBody();
        StringWriter writer = new StringWriter();
        body.invoke(writer);
        String str = writer.toString();
        String newStr = str.replaceAll(oldWord, newWord);
        out.print(newStr);
        return;
    }
}
```

변수를 지원하는 커스텀 액션을 만드는 태그 클래스



1. 태그 클래스가 변수를 지원하도록 만들려면 해당 변수의 이름과 값을 page 데이터 영역에 저장.
2. page 데이터 영역에 데이터를 저장하기 위해서는 우선 `getJspContext` 메서드를 호출해서 `JspContext` 객체를 구해야 한다.

```
JspContext context = getJspContext(); // page 데이터 영역에 데이터를 저장할 수 있는 객체
```

```
context.setAttribute("minimum", num1) // 변수이름 & 변수 값
```


변수를 지원하는 커스텀 액션을 만드는 태그 클래스

// 변수를 지원하는 태그 클래스 (1)

```
package tool;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class MinimumTag extends SimpleTagSupport {
    private int num1, num2;
    public void setNum1(Integer num1) {
        this.num1 = num1;
    }
    public void setNum2(Integer num2) {
        this.num2 = num2;
    }
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        if (num1 < num2)
            context.setAttribute( "minimum ", num1);
        Else
            context.setAttribute( "minimum ", num2);
        return;
    }
}
```

변수를 지원하는 커스텀 액션을 만드는 태그 클래스

1. TLD 파일에 변수를 지원하는 태그 클래스를 등록할 때는 <tag> 엘리먼트 안에 <variable>이라는 서브 엘리먼트를 쓰고, 다시 그 안에 변수에 대한 정보를 기술하는 몇 가지 서브 엘리먼트를 다음과 같은 형식으로 써야 한다

```
<tag>
```

```
...
```

```
<variable>
```

```
<name-given>minimum</name-given>
```

// 변수 이름

```
<variable-class>java.lang.Integer</variable-class>
```

// 변수 타입

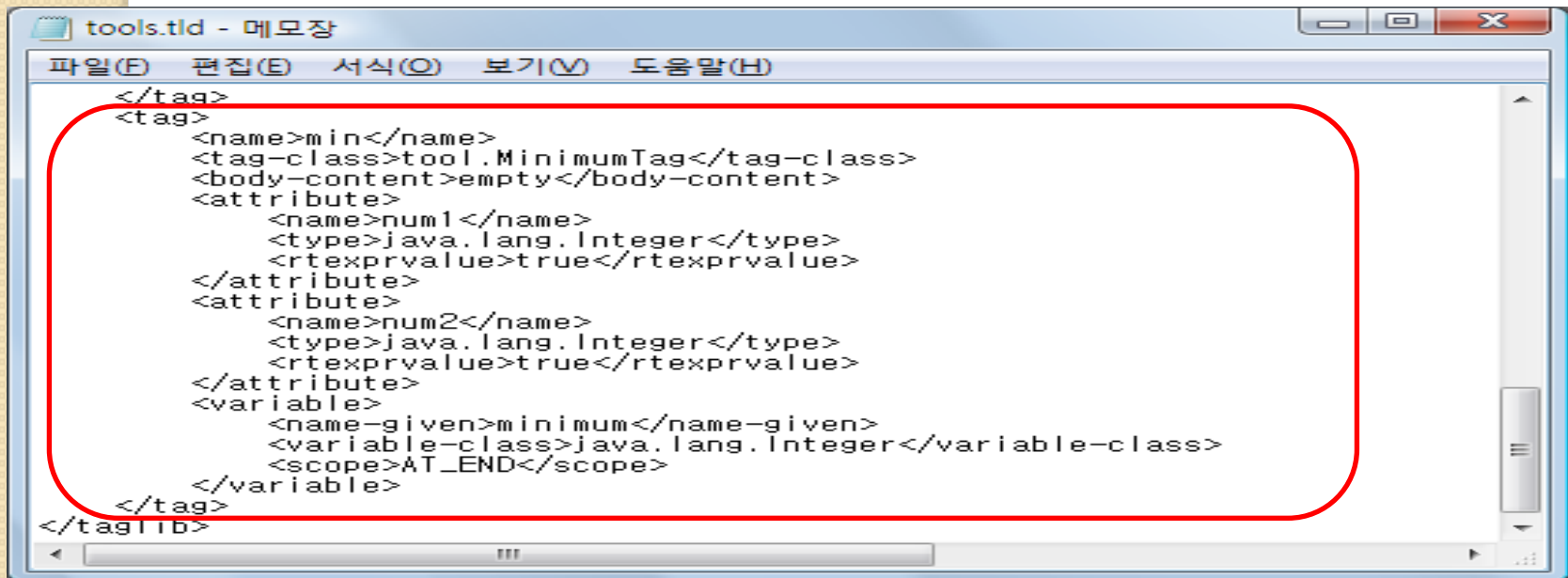
```
<scope>AT_END</scope>
```

// 변수 사용범위

```
</variable>
```

```
...
```

```
</tag>
```



변수를 지원하는 커스텀 액션을 만드는 태그 클래스

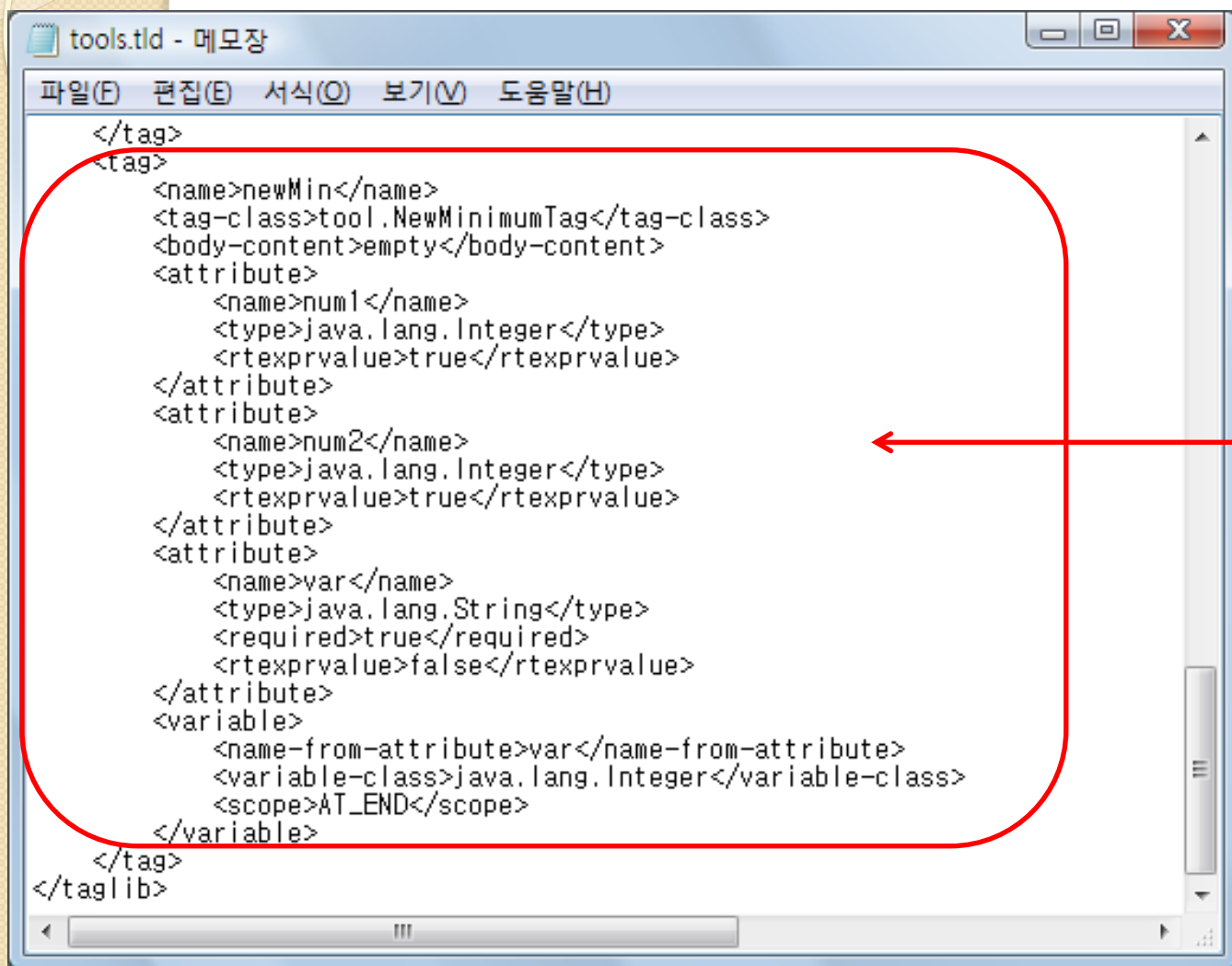
애트리뷰트를 통해 변수의 이름을 지정하는 커스텀 액션을 TLD 파일을 등록할 때는 <variable> 엘리먼트 안에 <name-given> 서브엘리먼트 대신 <name-from-attribute>라는 서브엘리먼트를 쓰고, 그 안에 변수의 이름 지정에 사용되는 애트리뷰트의 이름을 써야 한다.

```
<tag>
...
<variable>
// 변수 이름의 지정에 사용할 애트리뷰트의 이름
<name-from-attribute>var</name-from-attribute>
<variable-class>java.lang.Integer</variable-class> // 변수 타입
<scope>AT_END</scope> // 변수 사용 범위
</variable>
...
</tag>
```

TLD 파일에 변수의 이름 지정에 사용되는 애트리뷰트를 추가할 때는 필수 애트리뷰트로 만들어야 하고, 애트리뷰트 값에 익스프레션이나 EL 식을 사용할 수 없도록 만들어야 한다.

```
<tag>
...
<attribute>
  <name>var</name> // 애트리뷰트의 이름
  <type>java.lang.String</type> // 변수 타입
  <required>true</required> // 필수 애트리뷰트
  <rtexprvalue>>false</rtexprvalue> // 익스프레션이나 EL 식을 애트리뷰트 값으로 사용할 수 없도록 만드는 표시
</attribute>
...
</tag>
```

변수를 지원하는 커스텀 액션을 만드는 태그 클래스



```
</tag>
<tag>
  <name>newMin</name>
  <tag-class>tool.NewMinimumTag</tag-class>
  <body-content>empty</body-content>
  <attribute>
    <name>num1</name>
    <type>java.lang.Integer</type>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>num2</name>
    <type>java.lang.Integer</type>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>var</name>
    <type>java.lang.String</type>
    <required>true</required>
    <rteprvalue>>false</rteprvalue>
  </attribute>
  <variable>
    <name-from-attribute>var</name-from-attribute>
    <variable-class>java.lang.Integer</variable-class>
    <scope>AT_END</scope>
  </variable>
</tag>
</taglib>
```

태그 클래스를
등록하는 <tag>
엘리먼트

변수를 지원하는 커스텀 액션을 만드는 태그 클래스

// 이 만드는 커스텀 액션을 사용하는 JSP 페이지

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

```
<%@taglib prefix= "tool " uri= "/taglibs/tools.tld " %>
```

```
<HTML>
```

```
<HEAD> <TITLE>최소값 구하기 (New)</TITLE> </HEAD>
```

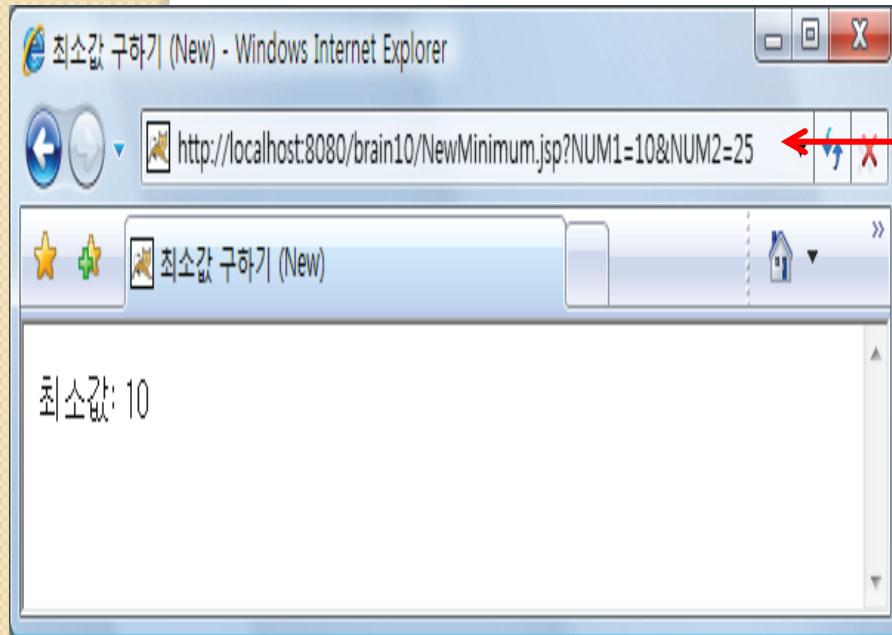
```
<BODY>
```

```
<tool:newMin var= "MIN " num1= "${param.NUM1}" num2= "${param.NUM2}" />
```

```
최소값: ${MIN}
```

```
</BODY>
```

```
</HTML>
```



URL 뒤에 이런 식으로
입력 데이터 값을
직접 쓰세요