

13

## JSP 커스텀 태그와 JSTL



# 이 장에서 다룰 내용

1

커스텀 태그

2

JSTL(JSP Standard Tag Library)



# 커스텀 태그

## □ 커스텀 태그의 개요

- 커스텀 태그란?
  - 개발자가 직접 정의할 수 있는 태그를 의미
  - 현재 존재하는 태그 외에도 새로운 태그를 정의하여 자신만이 원하는 기능을 구현

## □ 커스텀 태그 사용의 장점

- 일반 태그로 구현하지 못했던 한계점 극복
- 작성한 커스텀 태그를 라이브러리로 작성해 두면 다음 개발 시에 정의되어 있는 기능을 가져다 쓰기만 하면 되므로 프로그램의 재사용성 향상
- jsp 스크립트 대신 태그 형태를 사용하므로 가독성 향상
- 유지보수 편리
- 반복적인 기능을 쉽게 구현 가능



# 커스텀 태그

## □ 커스텀 태그의 구조 및 동작 원리

- 커스텀 태그가 동작하기 위해서는 실제로 커스텀 태그의 기능을 하는 클래스 파일 필요
- 클래스 파일을 태그 핸들러라고 부르며 커스텀 태그를 만들 때 가장 중요한 부분
- 태그 핸들러에 TLD파일 필요
  - TLD 파일
    - 태그 핸들러의 기능을 JSP 페이지에서 태그로 사용할 수 있게 연결해주는 기능 수행
- 작성한 커스텀 태그를 JSP 페이지에서 태그 라이브러리로 등록하여 사용



[그림 13-1] 커스텀 태그 사용시의 흐름



# 커스텀 태그

## □ TLD (Tag Library Descriptor) 파일

- 커스텀 태그 정보를 갖고 있는 라이브러리 파일
- 파일의 내용은 XML 형태로 구성

```
1  <?xml version="1.0" encoding="euc-kr"?>
2  <!DOCTYPE taglib
3      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
4      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
5
6      <taglib>
7          <tlibversion>1.0</tlibversion>
8          <jspversion>1.2</jspversion>
9          <shortname>helloTag</shortname>
10         <info>hello CustomTag</info>
11
12         <tag>
13             <name>hello</name>
14             <tagclass>hello.HelloTag</tagclass>
15             <teiclass>hello.HelloTEI</teiclass>
16             <bodycontent>JSP</bodycontent>
17
18             <attribute>
19                 <name></name>
20                 <required></required>
21                 <rtexprvalue></rtexprvalue>
22                 </attribute>
23             </tag>
24         </taglib>
```

# 커스텀 태그

## □ 태그 핸들러

- 커스텀 태그의 실제 기능을 정의한 클래스를 의
- 태그 핸들러 클래스를 만들기 위해서는 **TagSupport** 또는 **BodyTagSupport** 클래스를 상속받아야 한다.
- 이 두 개의 클래스는 커스텀 태그에 관련된 중요 패키지인 **javax.servlet.jsp.tagext**에서 제공하는 인터페이스들로 만들어진 것이다.
- 먼저 **javax.servlet.jsp.tagext** 패키지에서 제공하는 인터페이스

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.



# 커스텀 태그

## ■ TagSupport 클래스가 제공하는 주요 메소드

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.

## ■ BodyTagSupport 클래스가 제공하는 주요 메소드

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.



# 커스텀 태그

## ■ SimpleTagSupport 클래스가 제공하는 주요 메소드

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.

## ■ 예제

```
1 package hello;
2 import javax.servlet.jsp.tagext.TagSupport;
3
4 public class HelloTag extends TagSupport{
5     public int doStartTag() {
6         try{
7             pageContext.getOut().println("Hello Tag!");
8         }catch(Exception e){
9             e.printStackTrace();
10        }
11        return SKIP_BODY;
12    }
13 }
```





# JSTL(JSP Standard Tag Library)

## □ JSTL의 개요

### ■ JSTL

- JSTL은 자카르타에서 제공하는 자주 사용되는 필요한 기능들을 모아 놓은 커스텀 태그
- 커스텀 태그는 코드를 간결하게 해주고 가독성을 높이는 장점이 있기 때문에, JSTL을 잘 사용한다면 효율적인 코딩을 하는 데 많은 도움을 준다.
- JSTL은 용도에 따라 사용하는 기능이 달라지는데 크게 4가지로 나누어진다.
  - core : 기본적인 기능들을 제공
  - fmt : format의 약자로 형식화에 관한 기능들을 제공
  - xml : XML 처리에 좀 더 편한 기능을 제공
  - sql : SQL 처리에 편한 기능을 제공

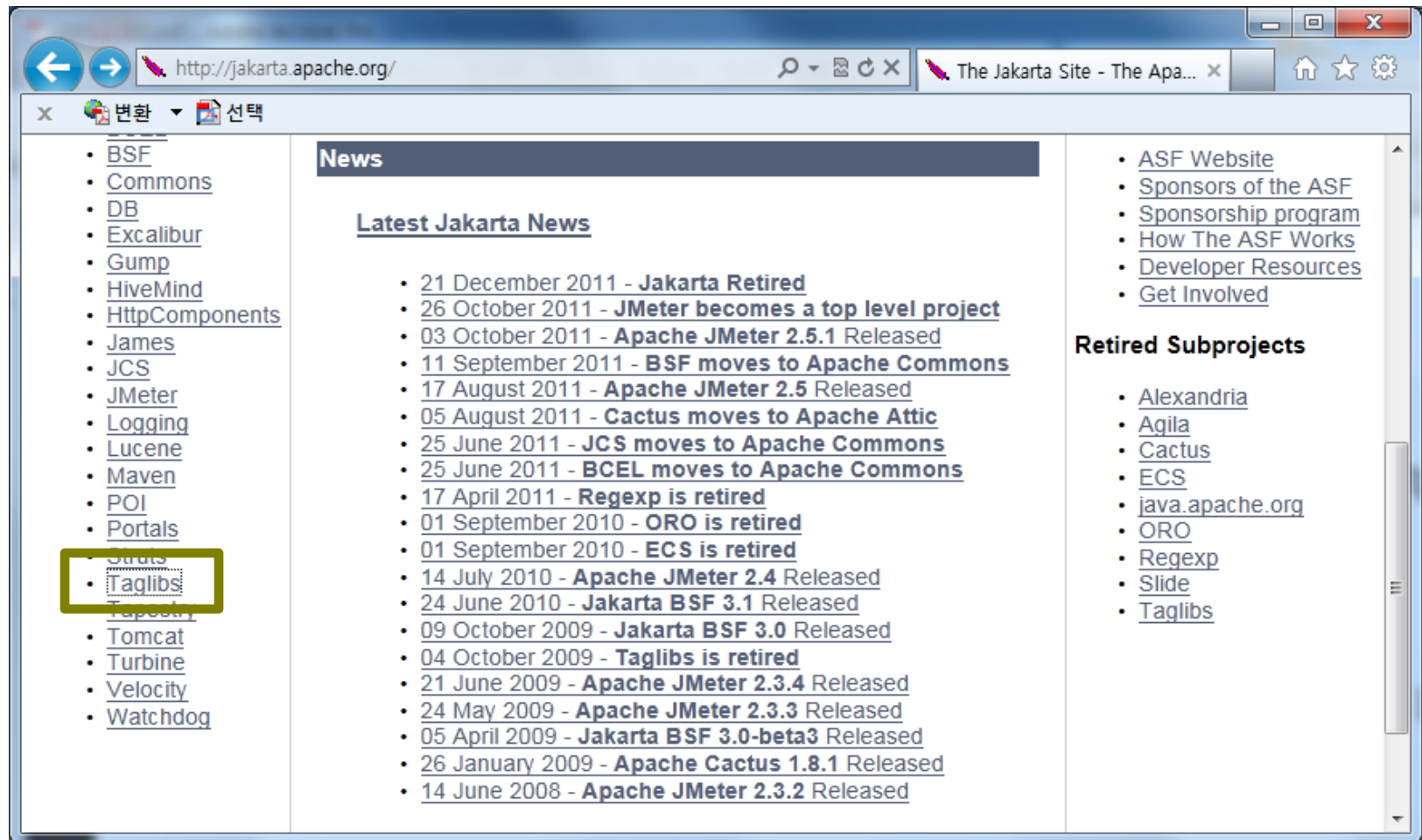
## □ JSTL의 다운로드 및 설치

- <http://jakarta.apache.org/> 사이트에서 다운로드



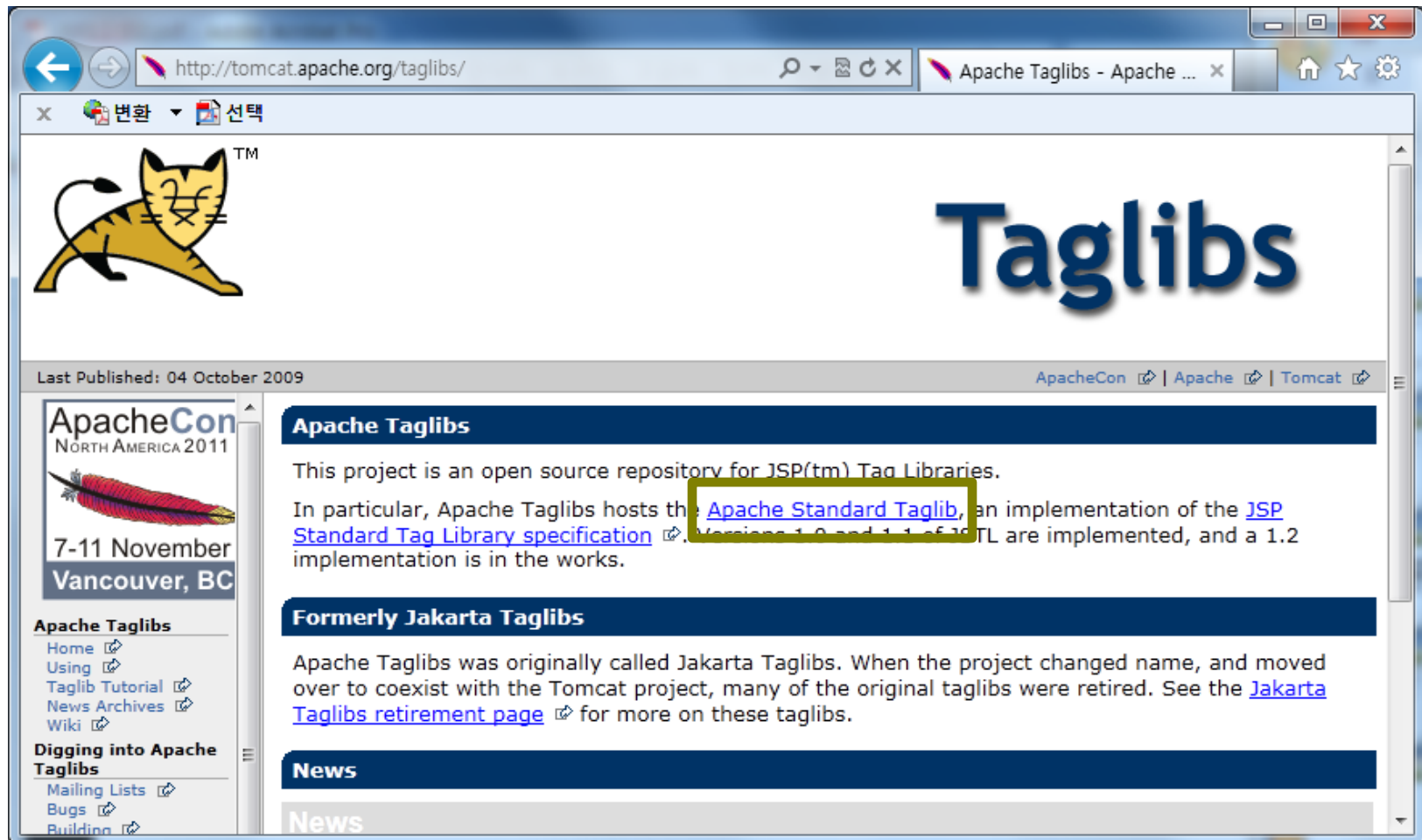
# JSTL(JSP Standard Tag Library)

- 01. <http://jakarta.apache.org>에 접속하여 왼쪽에 있는 Taglibs를 클릭한다.



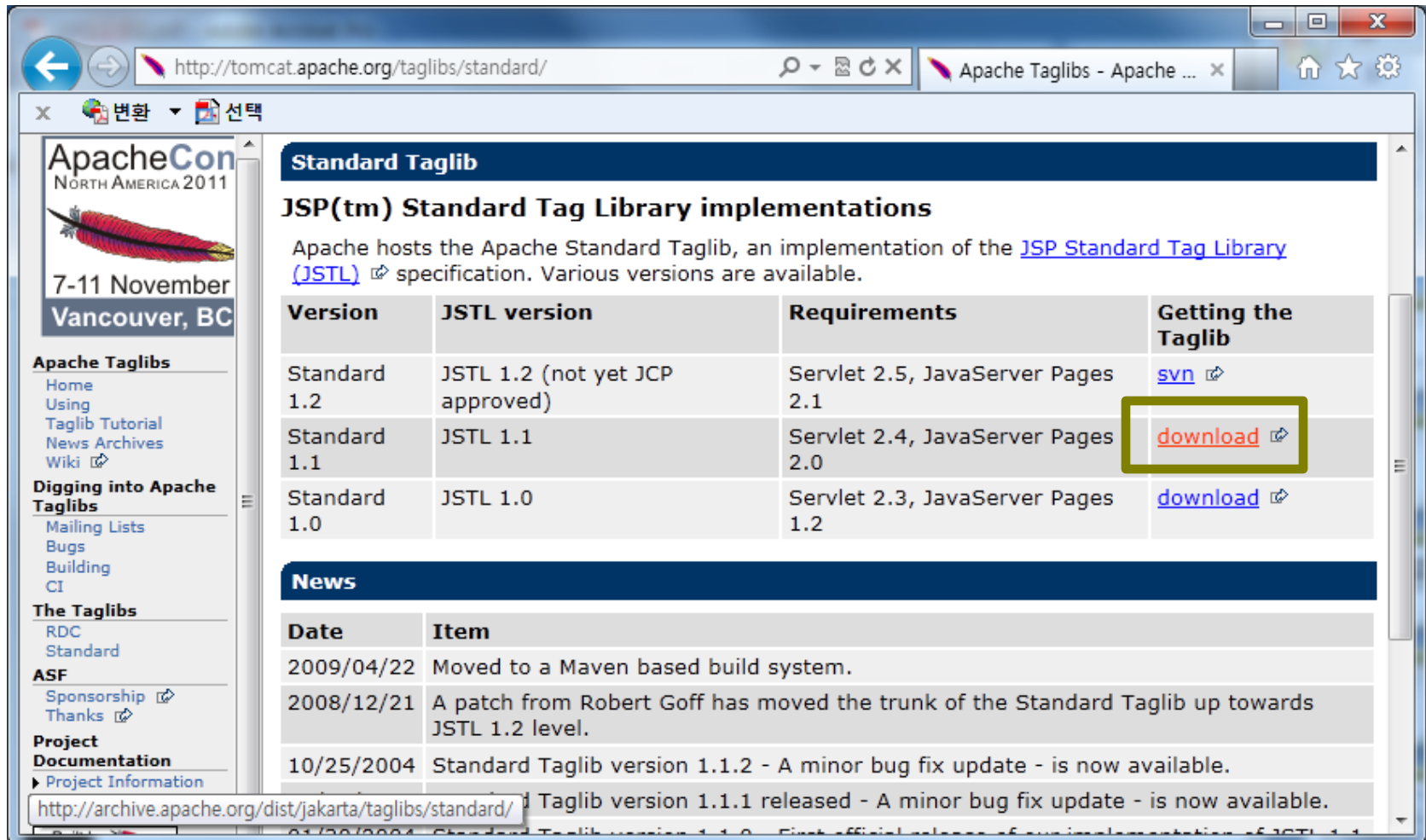
# JSTL(JSP Standard Tag Library)

## ■ 02. Apache Standard Taglib를 클릭한다.



# JSTL(JSP Standard Tag Library)

## ■ 03. JSTL 1.1 의 download를 클릭 한다.



The screenshot shows a web browser window with the URL <http://tomcat.apache.org/taglibs/standard/>. The page title is "Standard Taglib". Below the title, it says "JSP(tm) Standard Tag Library implementations". A paragraph states: "Apache hosts the Apache Standard Taglib, an implementation of the [JSP Standard Tag Library \(JSTL\)](#) specification. Various versions are available."

Version	JSTL version	Requirements	Getting the Taglib
Standard 1.2	JSTL 1.2 (not yet JCP approved)	Servlet 2.5, JavaServer Pages 2.1	<a href="#">svn</a>
Standard 1.1	JSTL 1.1	Servlet 2.4, JavaServer Pages 2.0	<a href="#">download</a>
Standard 1.0	JSTL 1.0	Servlet 2.3, JavaServer Pages 1.2	<a href="#">download</a>

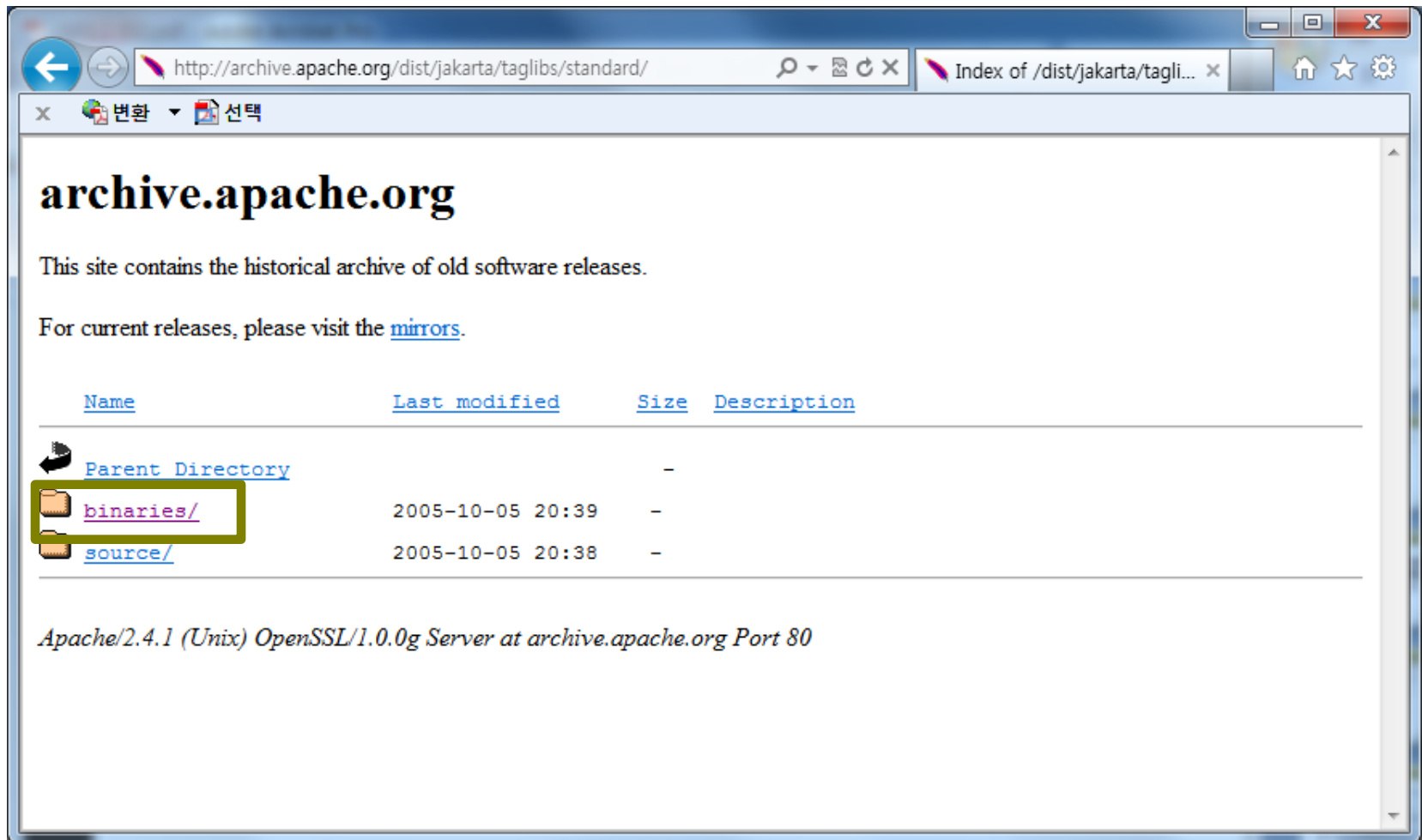
The "download" link for JSTL 1.1 is highlighted with a yellow box. Below the table, there is a "News" section with a table of updates:

Date	Item
2009/04/22	Moved to a Maven based build system.
2008/12/21	A patch from Robert Goff has moved the trunk of the Standard Taglib up towards JSTL 1.2 level.
10/25/2004	Standard Taglib version 1.1.2 - A minor bug fix update - is now available.
01/20/2004	Standard Taglib version 1.1.1 released - A minor bug fix update - is now available.

On the left side of the browser window, there is a sidebar with links for "Apache Taglibs", "Digging into Apache Taglibs", "The Taglibs", and "ASF". At the bottom of the sidebar, there is a link to "Project Information" and a URL: <http://archive.apache.org/dist/jakarta/taglibs/standard/>.

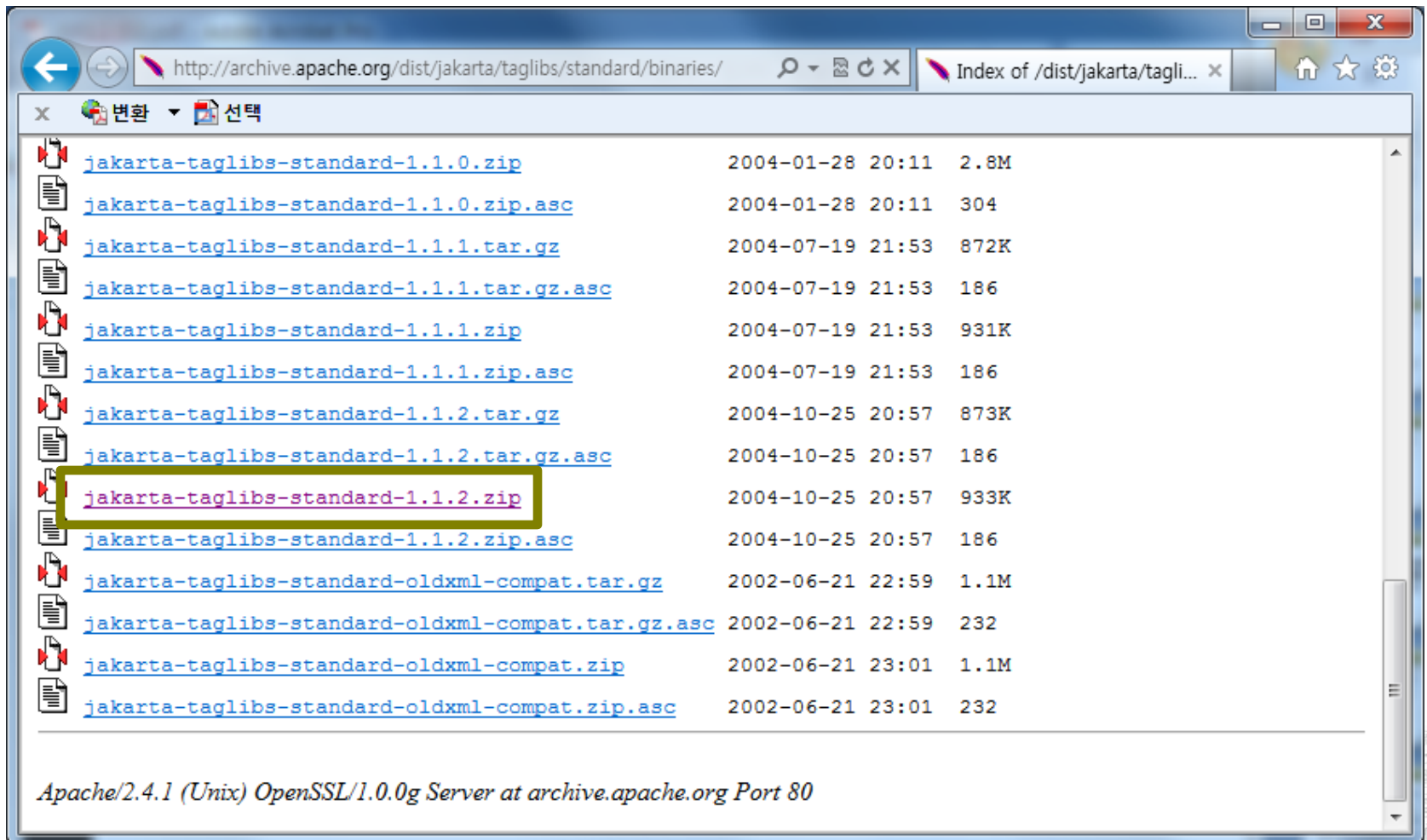
# JSTL(JSP Standard Tag Library)

## ■ 04. binaries를 클릭한다.



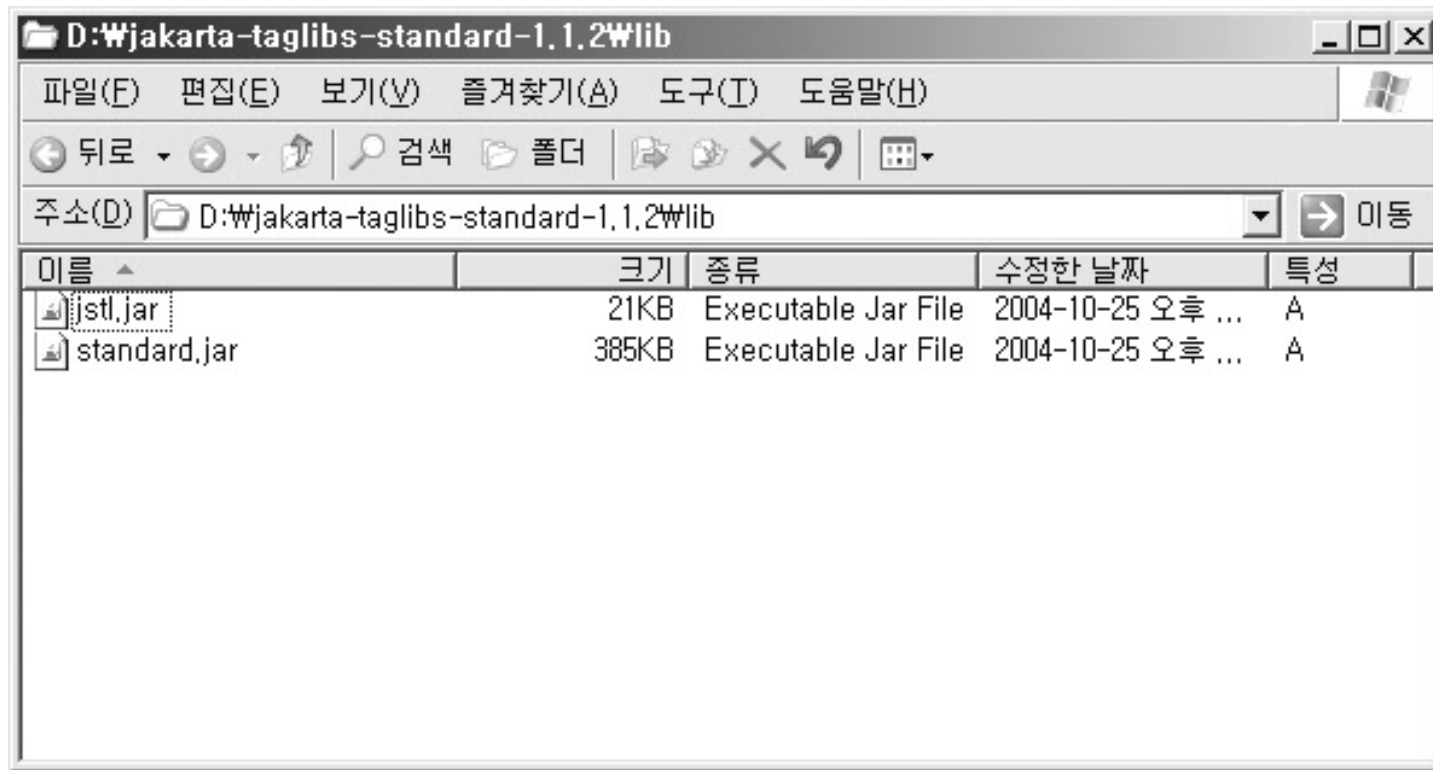
# JSTL(JSP Standard Tag Library)

- 05. jakarta-taglibs-standard-1.1.2.zip를 클릭하여 다운로드 한다.



# JSTL(JSP Standard Tag Library)

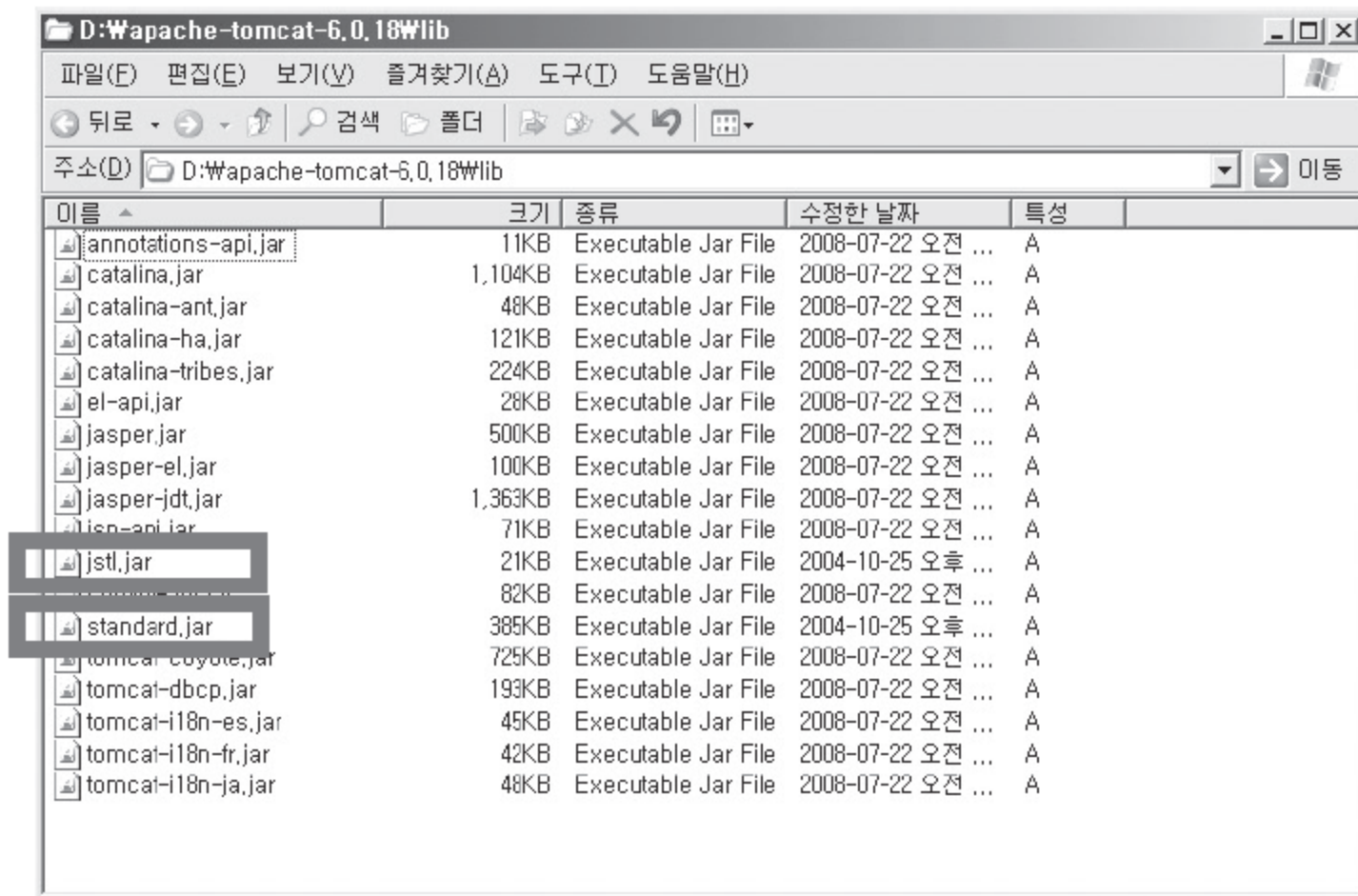
- 06. 다운로드 받은 파일을 압축을 푼 후에 lib 폴더를 보면 jstl.jar 파일과 standard.jar 파일이 존재하는 것을 확인할 수 있다. 이 두 파일이 JSTL의 기능을 제공해주는 jar 파일들이다.





# JSTL(JSP Standard Tag Library)

- 07. JSTL을 사용할 때 이 파일들을 톰캣의 라이브러리 폴더와 사용할 이클립스 프로젝트의 라이브러리 폴더에 넣어주어야 한다. 먼저 톰캣의 라이브러리 폴더에 복사하도록 하자.





# JSTL(JSP Standard Tag Library)

## □ EL(Expression Language)의 개요

### ■ EL(Expression Language)

- 표현 언어를 의미
- jsp 스크립트를 대신하여 속성 값들을 좀더 편리하게 출력하기 위해 제공되는 언어
- 예를 들어 `<%=hello%>`라는 코드를 EL로 표현할 때는 `${hello}`로 표현된다.

### ■ 표현식

test 변수를 표현할 때 → `${test}`

### ■ hello 객체의 test 변수를 표현하면 다음과 같다.

`${hello.test}` 나 `${hello['test']}`



# JSTL(JSP Standard Tag Library)

## □ EL의 내장 객체

내장 객체	설명
pageScope	Page 영역에 존재하는 객체를 참조할 때 사용한다.
requestScope	Request 영역에 존재하는 객체를 참조할 때 사용한다.
sessionScope	Session 영역에 존재하는 객체를 참조할 때 사용한다.
applicationScope	Application 영역에 존재하는 객체를 참조할 때 사용한다.
param	파라미터 값을 얻어올 때 사용한다.
paramValues	파라미터 값을 배열로 얻어올 때 사용한다.
header	Header 정보를 얻어올 때 사용한다.
headerValues	Header 정보를 배열로 얻어올 때 사용한다.
cookie	쿠키 객체를 참조할 때 사용한다.
initParam	컨텍스트의 초기화 파라미터를 의미한다.
pageContext	PageContext 객체를 참조할 때 사용한다.



# JSTL(JSP Standard Tag Library)

## ■ 예제

### • el\_test.jsp (교재 335p 참조)

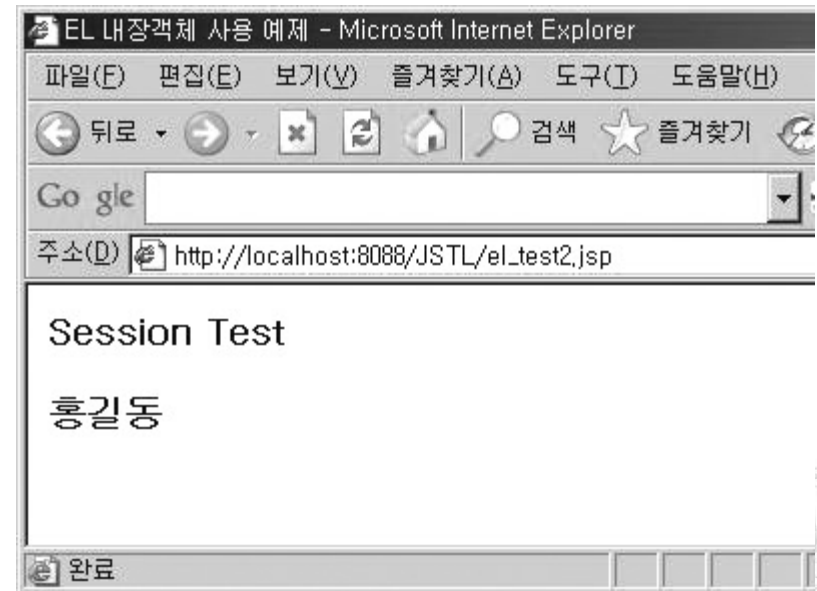
```
1      <%@ page language="java" contentType="text/html; charset=EUC-KR"%>
2      <%
3          session.setAttribute("test","Session Test");
4      %>
5      <html>
6      <head>
7          <title>EL 내장객체 사용 예제</title>
8      </head>
9      <body>
10         <form action="el_test2.jsp" method="post">
11             <table border=1>
12                 <tr><td>이름 : </td><td><input type="text" name="name" value="홍길동"></td>
13                 <tr><td colspan=2 align=center><input type="submit" value="입력"></td></tr>
14             </table>
15         </form>
16     </body>
17 </html>
```



# JSTL(JSP Standard Tag Library)

- **el\_test2.jsp** (교재 335p 참조)

```
1      <%@ page language="java" contentType="text/html; charset=EUC-KR"%>
2      <%
3          request.setCharacterEncoding("euc-kr");
4      %>
5      <html>
6      <head>
7          <title>EL 내장객체 사용 예제</title>
8      </head>
9      <body>
10         <h3>${sessionScope.test}</h3>
11         <h3>${param.name }</h3>
12     </body>
13 </html>
```



# JSTL(JSP Standard Tag Library)

## □ EL의 연산자

### ■ EL 연산자의 종류와 쓰임

#### • EL 연산자

연산자	설명
.	빈 또는 맵에 접근하기 위한 연산자이다.
[]	배열 또는 리스트에 접근하기 위한 연산자이다.
()	연산할 때 우선 순위를 주려고 할 때 사용한다.
$x ? a : b$	$x$ 의 조건이 만족하면 $a$ 를 리턴하고, 만족하지 않으면 $b$ 를 리턴한다.
Empty	값이 NULL일 경우 true를 반환한다.



# JSTL(JSP Standard Tag Library)

## • 산술 연산자

산술 연산자	설명
+	더하기 연산자
-	빼기 연산자
*	곱하기 연산자
/ 또는 div	나누기 연산자
% 또는 mod	나머지 연산자

## • 논리 연산자

논리 연산자	설명
&& 또는 and	두 항의 내용을 모두 만족할 경우 true, 그렇지 않으면 false를 반환한다.
또는 or	두 항의 내용 중 하나라도 만족하면 true, 그렇지 않으면 false를 반환한다.
! 또는 not	값이 만족하지 않으면 true, 만족하면 false를 반환한다. 즉, true는 false 로 false 는 true 로 변경해주는 연산자이다.



# JSTL(JSP Standard Tag Library)

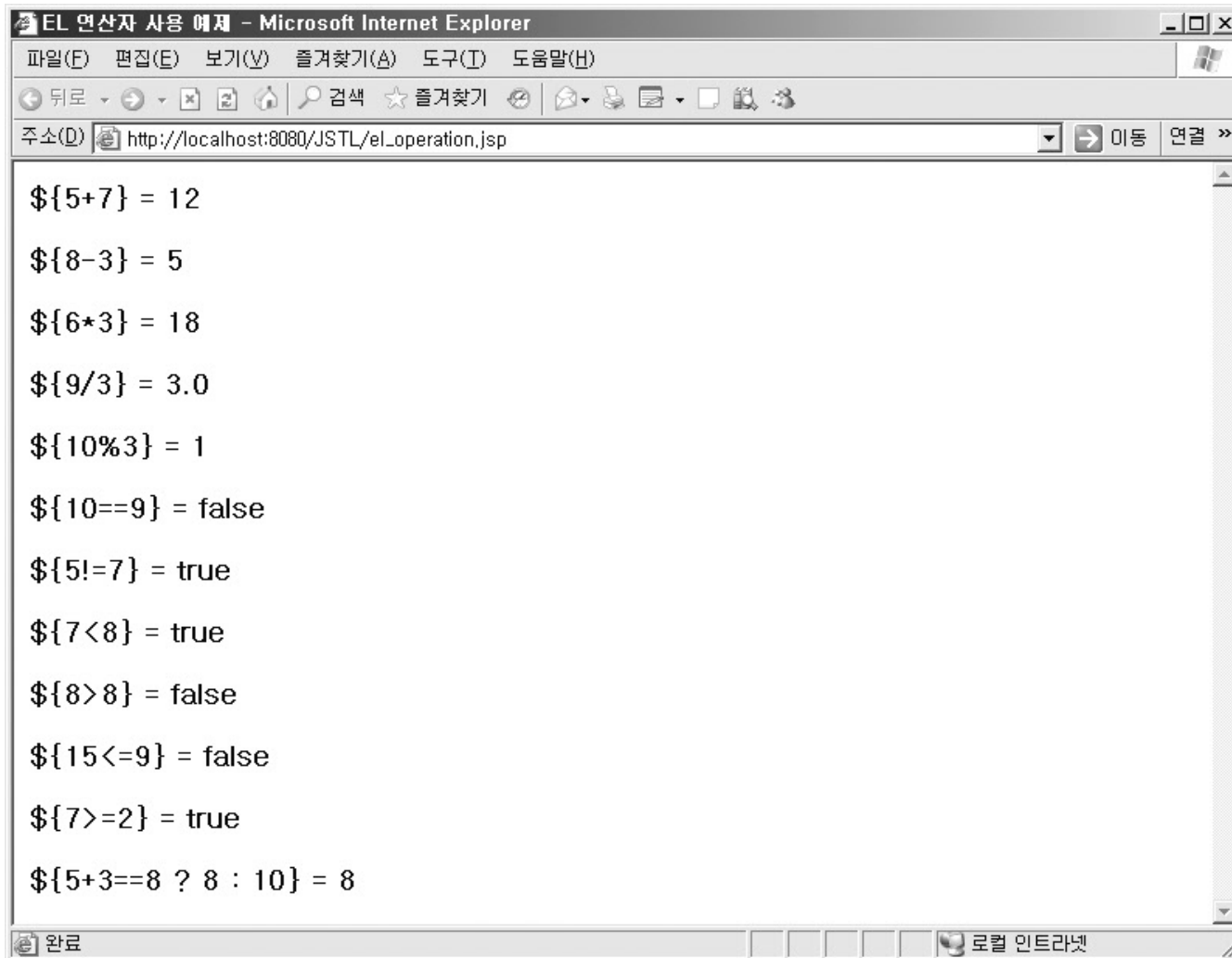
## • 비교 연산자

비교 연산자	설명
== 또는 eq	두 항의 값이 같으면 true, 그렇지 않으면 false를 반환한다.
!= 또는 ne	두 항의 값이 다르면 true, 그렇지 않으면 false를 반환한다.
< 또는 lt	'보다 작다'라는 의미를 갖고, 왼쪽 항이 오른쪽 항보다 작으면 true를 반환한다.
> 또는 gt	'보다 크다'라는 의미를 갖고, 왼쪽 항이 오른쪽 항보다 크면 true를 반환한다.
<= 또는 le	'같거나 작다'라는 의미를 갖고, 왼쪽 항이 오른쪽 항보다 같거나 작으면 true를 반환한다.
>= 또는 ge	'같거나 크다'라는 의미를 갖고, 왼쪽 항이 오른쪽 항보다 같거나 크면 true를 반환한다.



# JSTL(JSP Standard Tag Library)

- EL 연산자의 사용 방법 예제로 알아보기
  - el\_operation.jsp (교재 338p 참조)





# JSTL(JSP Standard Tag Library)

## □ JSTL의 기본 액션 - JSTL core

### ■ JSTL core란?

- JSTL에서 기본적인 기능(즉, 컨트롤에 관계된 기능)들을 구현해 놓은 라이브러리
- 예를 들어 문자열을 출력하거나, 반복문, 조건문과 같은 내용이 core 라이브러리에 포함되어 있는 것이다.

### ■ 등록하는 방법

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

### ■ JSTL core 라이브러리 태그 알아보기

- 출력 태그 : <c:out>
- 변수 설정 및 삭제 태그 : <c:set>, <c:remove>
- 예외 처리 태그 : <c:catch>
- 조건 처리 태그 : <c:if>, <c:choose>, <c:when>, <c:otherwise>
- 반복 처리 태그 : <c:forEach>, <c:forTokens>



# JSTL(JSP Standard Tag Library)

- **<c:out>**

- 지정된 값을 출력시키는 태그
- value 속성을 이용해서 변수의 내용을 출력할 수 있다.
- default 속성은 기본 값을 의미하는데 value 값이 null일 경우 이 기본 값을 출력하게 된다.
- escapeXml은 false가 기본 값으로 지정되어 있으며 <, > 등의 특수 기호를 처리할 때 사용된다. True일 경우 < 값은 &lt;로 표현되고 > 값은 &gt;로 표현된다.

사용법

```
<c:out value="출력값" default="기본값" escapeXml="true 또는 false">
```

- **<c:set>**

- 지정된 변수에 값을 설정하는 태그
- var는 값을 저장할 변수 이름을 의미
- value는 저장할 값을 의미
- target은 값을 설정할 프로퍼티에 대한 객체를 의미
- property는 값을 설정할 객체의 프로퍼티를 의미
- scope는 변수의 유효 범위를 의미하며 설정하지 않을 경우 기본 값으로 page를 갖는다.

사용법

```
<c:set var="변수명" value="설정값" target="객체" property="값" scope="범위">
```



# JSTL(JSP Standard Tag Library)

- **<c:remove>**

- 설정된 속성을 제거하는 태그
- var은 설정된 속성이 저장되어 있는 변수를 의미
- scope는 지정된 범위에서 저장된 속성을 제거하도록 한다.

사용법

```
<c:remove var="변수명" scope="범위">
```

- **<c:catch>**

- 예외 처리를 위한 태그
- 예외가 발생하면 var에 지정된 예외 객체가 할당된다.

사용법

```
<c:catch var="변수명">
```

- **<c:if>**

- 조건 처리를 할 때 사용되는 태그
- test 속성에는 조건을 지정
- var 속성의 변수에는 조건 처리한 결과를 저장한다.
- scope는 var 속성에 지정한 변수의 범위를 의미한다.

사용법

```
<c:if test="조건" var="변수명" scope="범위">
```



# JSTL(JSP Standard Tag Library)

- **<c:choose>**

- 조건 처리를 할 때 쓰이는 태그
- <c:when> 태그에서 test 속성으로 조건을 확인하며 조건에 만족하면 <c:when>과 </c:when> 사이에 있는 내용을 처리한다.
- <c:otherwise> 태그는 <c:when>태그의 조건에 모두 만족하지 않을 경우 실행된다.

사용법

```
<c:choose>
<c:when test="조건"></c:when>
<c:otherwise></c:otherwise>
</c:choose>
```

- **<c:forEach>**

- 자바의 for문과 유사
- items 속성에 컬렉션이나 배열 형태의 객체를 지정하여 객체의 인덱스만큼 반복
- begin과 end 속성으로 원하는 범위만큼 반복문을 수행할 수도 있다.
- Step은 증감식을 설정하여 1, 3, 5, 7 ... 또는 2, 4, 6, 8 ... 등의 반복도 가능하도록 한다.
- var은 반복 중일때 현재 반복하고 있는 값을 기억하는 변수이다.
- varStatus는 반복의 상태를 갖게 되는 변수이다.

사용법

```
<c:forEach items="객체명" begin="시작 인덱스" end="끝 인덱스" step="증감식" var="변수명" varStatus="상태변수">
```

# JSTL(JSP Standard Tag Library)

- **<c:forTokens>**

- 자바의 for문과 StringTokenizer 객체를 결합한 형태
- Items에 지정한 값을 delims 속성의 구분자로 나눈 후 나뉜 만큼 반복을 수행한다.
- items, delims, var 이 3가지의 속성만으로도 <c:forTokens> 태그를 수행할 수 있게 된다.
- begin과 end의 경우는 delims 구분자로 나뉜 값을 기준으로 시작 값과 끝 값을 정해주기 때문이다.

사용법

```
<c:forTokens items="객체명" delims="구분자" begin="시작 인덱스" end="끝 인덱스"
step="증감식" var="변수명" varStatus="상태변수">
```

- **<c:import>**

- 지정된 URL을 태그가 사용된 JSP 페이지에 출력시키는 기능
- url 속성에는 HTTP뿐만 아니라 FTP 외부 리소스도 올 수 있다.
- var는 리소스가 저장될 변수명
- scope는 var 속성의 변수의 범위를 의미
- varReader는 리소스가 저장될 변수를 의미(단, 이 변수는 Reader 객체여야 한다.)
- context 속성은 URL에 접근할 때의 컨텍스트 이름을 의미
- charEncoding속성은 지정된 URL의 리소스를 어떻게 인코딩할 것인지를 의미한다.

사용법

```
<c:import url="URL값" var="변수명" scope="범위" varReader="입력스트림명"
context="contextName" charEncoding="인코딩값">
```

# JSTL(JSP Standard Tag Library)

- **<c:redirect>**

- 지정된 URL로 페이지를 이동시키는 기능
- url 속성에 이동할 URL을 입력
- context 속성에는 컨텍스트 이름을 입력

사용법

```
<c:redirect url="URL값" context="contextName">
```

- **<c:url>**

- value 속성에 지정된 값으로 URL을 생성하는 기능
- scope는 var 속성에 입력되어 있는 변수의 범위를 의미

사용법

```
<c:url var="변수명" scope="범위" value="값" context="contextName">
```

- **<c:param>**

- <c:import> 태그에 파라미터를 전달하기 위한 태그
- name 속성에 파라미터명을 입력
- value에 값을 입력하여 전달

사용법

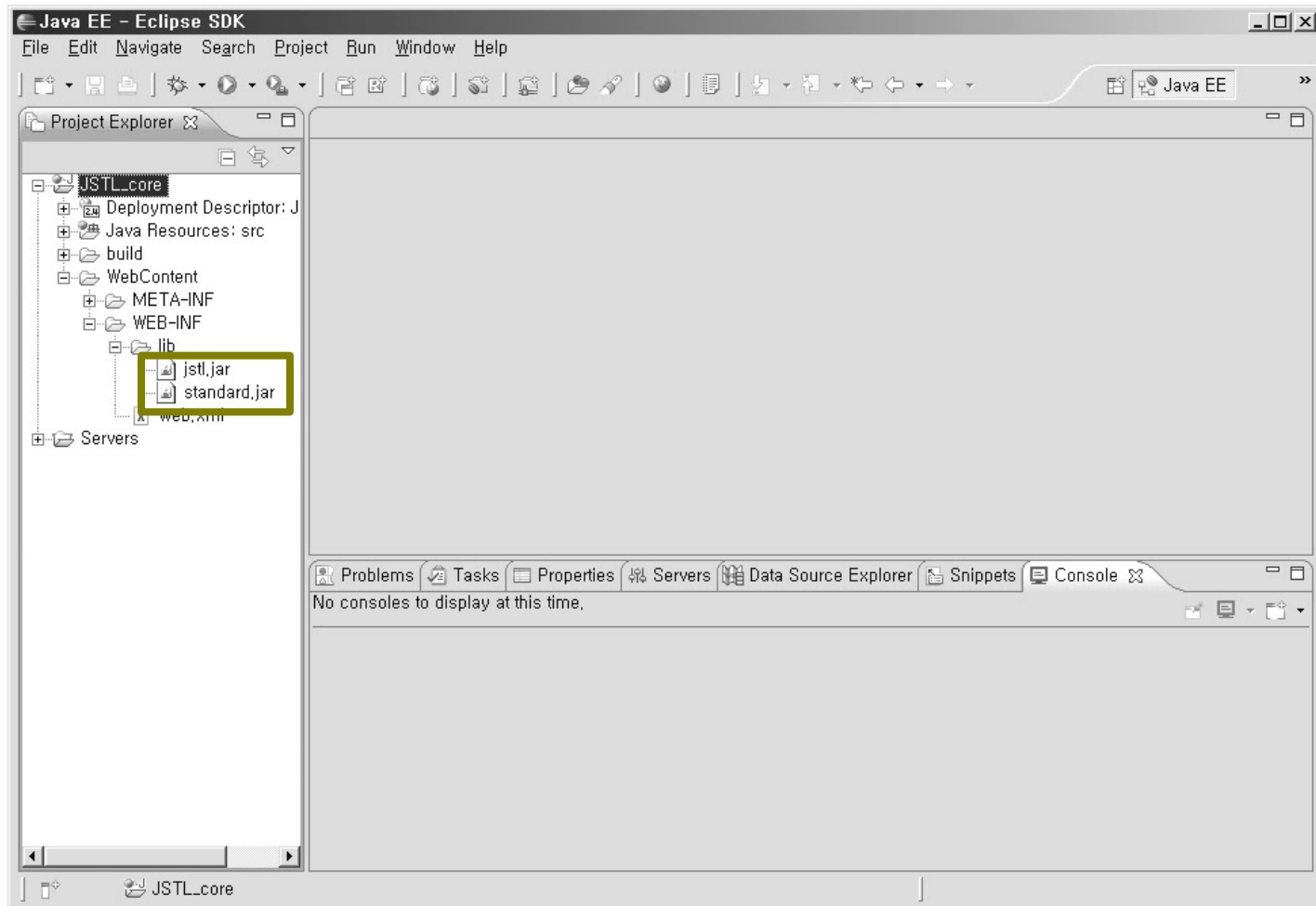
```
<c:param name="파라미터명" value="값">
```



# JSTL(JSP Standard Tag Library)

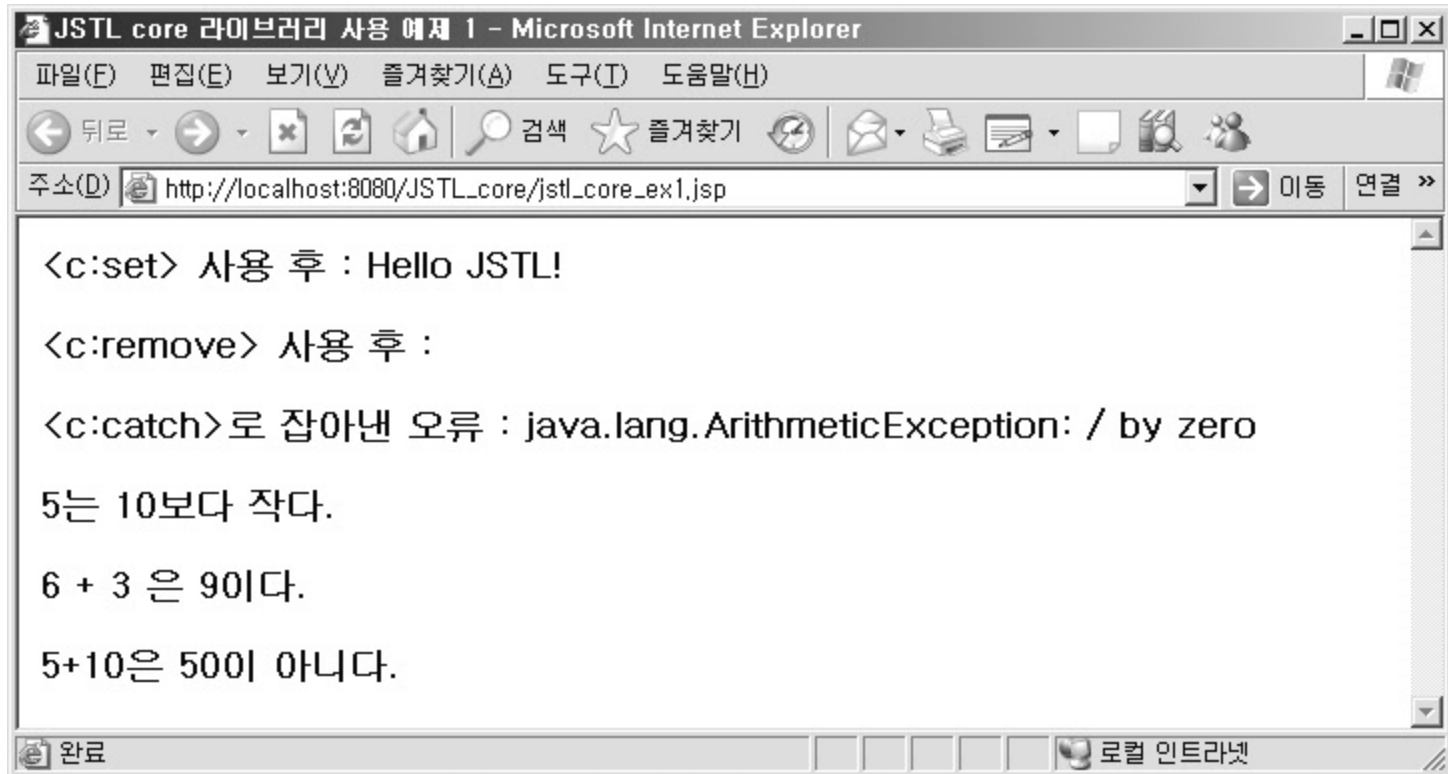
## ■ 예제 작성하기

- JSTL 라이브러리 파일인 `jstl.jar` 파일과 `standard.jar` 파일을 이클립스 프로젝트 폴더의 `WebContent\WEB-INF\lib` 폴더에 복사



# JSTL(JSP Standard Tag Library)

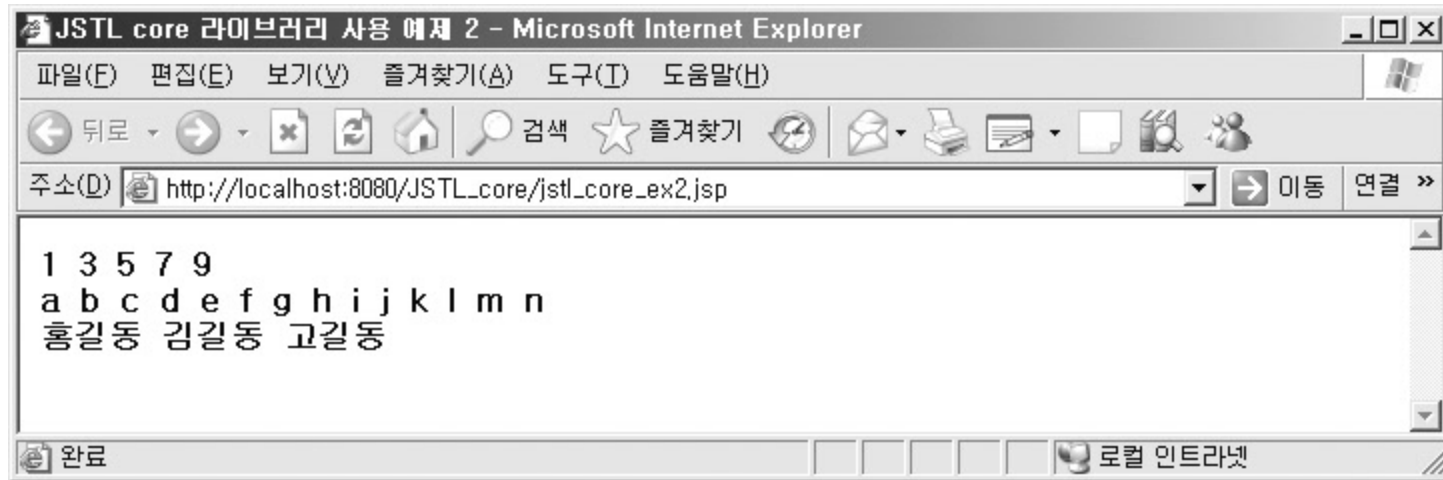
- `jstl_core_ex1.jsp` (교재 344p 참조)





# JSTL(JSP Standard Tag Library)

- `jstl_core_ex2.jsp` (교재 346p 참조)



# JSTL(JSP Standard Tag Library)

## □ JSTL의 국제화/형식화 액션 - JSTL 톳

### ■ JSTL fmt란?

- 국제화/형식화의 기능을 제공해주는 JSTL 라이브러리
- 국제화는 다국어 내용을 처리, 형식화는 날짜와 숫자 형식 등을 처리하는 것을 말한다

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

### ■ fmt 라이브러리 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>



# JSTL(JSP Standard Tag Library)

- **<fmt:requestEncoding>**

- Request 객체로부터 전달 받은 값을 인코딩할 때 사용
- 보통 한글 값이 넘어올 경우 <%request.setCharacterEncoding( "euc-kr" );%> 코드로 Request 객체로 받는 값을 한글에 맞게 인코딩 해준다.

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

- **<fmt:setLocale>**

- 다국어 페이지를 사용할 때 언어를 지정하는 태그
- value 속성은 어떤 언어를 사용할지 지정하는 속성이며 언어 코드를 입력할 수 있다.
- variant 속성은 브라우저의 스펙을 지정할 때 사용

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

- **<fmt:timeZone>**

- 지정한 지역 값으로 시간대를 맞출 때 사용
- <fmt:setTimeZone>는 특정 페이지 전체에 적용이 되지만, <fmt:timeZone> 태그는 첫 태그와 끝 태그 사이의 Body 부분의 내용에만 적용

```
<fmt:timeZone value="값">
```

# JSTL(JSP Standard Tag Library)

- **<fmt:setTimeZone>**

- 지정한 지역 값으로 시간대를 맞추는 기능(페이지 전체에 작용)
- **var** 속성의 변수에 지정한 시간대 값이 저장
- **scope**로 범위를 지정

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

- **<fmt:bundle>**

- **properties** 확장명을 가진 파일의 리소스를 불러올 때 사용
- **basename**에는 **properties** 확장명을 가진 파일을 지정
- **prefix**는 **properties** 내의 키 값에 쉽게 접근할 수 있도록 간략한 접근어를 사용할 수 있게 해준다. 그리고 불러온 리소스는 이 태그의 Body내에서만 사용할 수 있다.

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

- **<fmt:setBundle>**

- **<fmt:bundle>** 태그와 같은 기능을 하는 태그
- **<fmt:setBundle>**태그는 사용범위가 페이지 전체에 적용
- **var**에는 설정한 리소스 내용을 가지고 있을 변수명을 입력
- **scope**로 범위를 지정

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



# JSTL(JSP Standard Tag Library)

- **<fmt:message>**

- 설정한 **properties** 파일의 리소스 내용을 읽을 때 사용
- **properties** 파일에는 여러 키 값들로 내용이 구분되어 있으며, **<fmt:message>** 태그의 **key** 속성으로 키 값으로 설정된 내용을 가져올 수 있다.
- **bundle** 속성에는 **<fmt:setBundle>** 태그에서 **var** 속성에 지정했던 변수를 입력

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

- **<fmt:param>**

- **<fmt:message>** 태그로 읽어 온 리소스 내용에 파라미터를 전달하는 태그

- **<fmt:formatNumber>**

- 태그는 숫자 형식의 패턴을 설정할 때 사용
- **value** 속성에는 패턴을 적용시킬 원래의 값을 입력
- **type**은 숫자의 타입을 의미. 타입은 숫자, 통화, 퍼센트 중 원하는 타입으로 설정가능
- **pattern** 속성은 지정한 값을 어떤 패턴으로 변환시킬지를 정할 수 있다.
- **currencyCode**는 통화 코드를 의미하며, 숫자 타입이 통화일 경우만 유효하다.
- **currencySymbol**도 숫자 타입이 통화일 경우만 유효하며, 통화 기호를 지정할 수 있다.
- **groupingUsed**는 그룹 기호의 포함 여부를 나타낸다.
- **maxIntegerDigits**는 최대 정수 길이
- **minIntegerDigits**는 최소 정수 길이
- **maxFractionDigits**은 최대소수점 자릿수
- **minFractionDigits**는 최소 소수점 자릿수를 의미



# JSTL(JSP Standard Tag Library)

```
<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, 퍼센트 중 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 값 중 정수 부분만 해석할지의 여부를 지정하는 속성

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



# JSTL(JSP Standard Tag Library)

- **<fmt:formatDate>**

- 날짜 형식의 패턴을 설정할 때 사용되는 태그
- value 속성에는 날짜 또는 시간을 입력
- type 속성으로 날짜인지 시간인지 또는 날짜와 시간 둘 다 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
- timeZone 속성은 날짜와 시간이 표시될 시간대를 지정

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

- **<fmt:parseDate>**

- 문자열을 날짜와 시간의 형태로 변환하는 태그
- value 속성에 입력된 문자열 값을 type 속성에 지정된 타입으로 날짜와 시간의 형태로 변환
- 나머지 속성은 앞에서 설명한 <fmt:formatDate>의 속성과 기능 동일
- <fmt:parseDate>는 문자열 값을 파싱하여 날짜, 시간 형태로 변환

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



# JSTL(JSP Standard Tag Library)

- 예제 작성하기
  - 교재 352p ~ 357p 참조





# JSTL(JSP Standard Tag Library)

## □ JSTL의 XML 액션 - JSTL xml

### ■ JSTL xml

- XML 문서에서 자주 사용되는 기능들을 태그 라이브러리로 모아 놓은 것

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

### ■ xml 라이브러리 태그 알아보기

- 출력, 변수 설정 태그 : <x:out>, <x:set>
- 조건 처리 태그 : <x:if>, <x:choose>, <x:when>, <x:otherwise>
- 반복 처리 태그 : <x:forEach>
- 기타 XML 관련 태그 : <x:parse>, <x:transform>, <x:param>



# JSTL(JSP Standard Tag Library)

- **<x:out>**

- 지정된 XPath의 내용을 출력하는 태그
- escapeXml은 기본 값으로 false로 지정되어있으며 <, >등의 특수 기호의 출력 형태를 설정할 때 쓰인다. True일 경우 < 값은 &lt;로 표현되고 > 값은 &gt;로 표현

```
<x:out select="XPath expression" escapeXml="True 또는 False">
```

- **<x:set>**

- 지정된 변수에 지정한 XPath 내의 XML 내용을 저장하는 태그
- var 속성이 값을 저장할 변수를 의미
- select 속성에 XPath 표현식을 입력
- scope는 변수의 범위를 의미

```
<x:set var="변수명" select="XPath expression" scope="범위">
```

- **<x:if>**

- core 라이브러리에 존재하는 <c:if> 태그와 유사한 기능
- select 속성에 지정된 XPath 표현식으로 조건을 판별
- var 속성의 변수에는 조건 처리의 결과를 저장

```
<x:if select="XPath expression" var="변수명" scope="범위">
```



# JSTL(JSP Standard Tag Library)

- **<x:choose>**

- core 라이브러리의 <c:choose>와 유사한 기능. 조건문의 시작을 알리는 태그로 사용된다.
- <x:when>은 조건을 지정할 때 사용
- select 속성에 XPath 표현식을 입력하여 조건을 지정
- <x:otherwise> 태그 사이의 내용은 <x:when> 태그의 조건에도 성립하지 않을 경우 처리

```
<x:choose>  
<x:when select="XPath expression"></c:when>  
<x:otherwise></x:otherwise>  
</x:choose>
```

- **<x:forEach>**

- XML에서 반복 처리가 필요할 때 사용
- <x:forEach>에서 자주 사용되는 속성은 select 속성이며, 여기에 XPath 표현식을 입력하여 반복 처리를 수행

```
<x:forEach var="변수명" select="XPath expression" begin="시작 인덱스" end="끝 인덱스" step="증감식">
```



# JSTL(JSP Standard Tag Library)

- **<x:parse>**

- XML 문서를 파싱할 때 사용
- 보통은 var 속성을 사용하여 파싱할 XML 문서를 var 속성에 입력된 변수에 저장
- systemId는 파싱되고 있는 문서의 URI를 나타낸다.
- filter는 파싱하기 전에 지정된 필터의 내용을 걸러낼 때 사용한다.
- doc속성은 직접 XML 문서의 위치를 지정할 때 사용

```
<x:parse var="변수명" varDom="변수명" scope="범위" scopeDom="범위" doc="source"
systemId="URI" filter="필터">
```

- **<x:transform>**

- XML 문서를 XSL 스타일시트를 이용하여 새로운 문서로 변형시키는 역할 수행
- doc 속성에는 XML문서를 입력
- xslt 속성에는 XSL 스타일시트를 입력
- var 속성에는 생성된 결과를 저장
- result 속성도 생성된 결과를 지정

```
<x:transform var="변수명" scope="범위" result="변수명" doc="source" xslt="XSLTStyleSheet">
```



# JSTL(JSP Standard Tag Library)

- **<x:param>**

- <x:transform> 태그 사이에 파라미터 값을 전달하기 위해 사용
- name 속성에는 전달할 파라미터의 이름 입력
- value 속성에는 전달할 파라미터 값을 입력

```
<x:param name="이름" value="값">
```

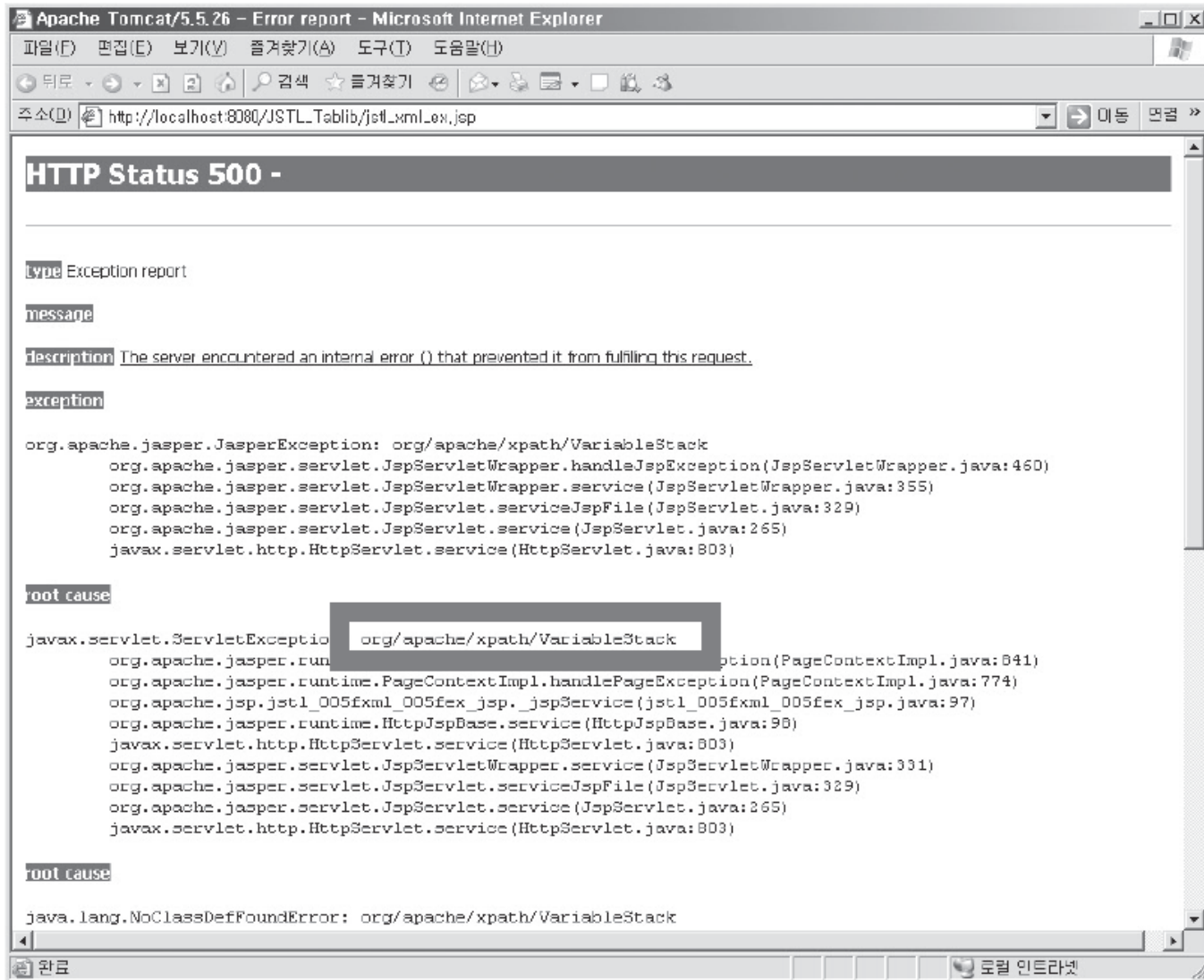
- **예제 작성하기**

- jstl\_xml\_ex.jsp (교재 360p 참조)



# JSTL(JSP Standard Tag Library)

- 실행 결과



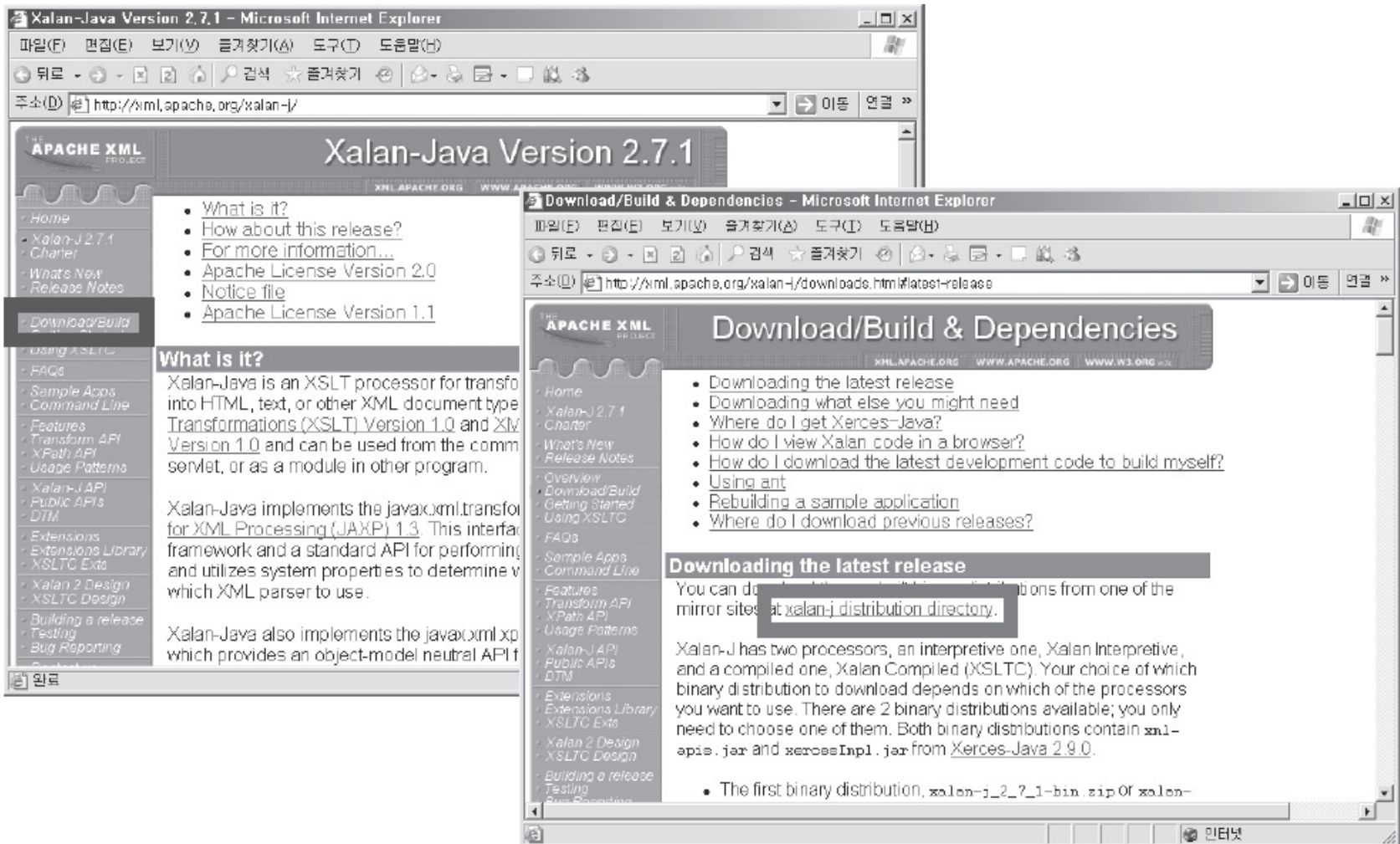
# JSTL(JSP Standard Tag Library)

- 실행결과 에러 발생. 위에 표시되어 있는 `org/apache/xpath/VariableStack` 클래스를 찾을 수 없기 때문이다. xml 라이브러리는 JSTL에 포함되어 있지 않다. `xalan.jar` 파일을 라이브러리 폴더에 등록하면 사용 가능하다.  
<http://xml.apache.org/xalan-j>에서 다운로드 받을 수 있다.



# JSTL(JSP Standard Tag Library)

- xalan.jar 파일을 라이브러리 폴더에 등록하기
  - 01. <http://xml.apache.org/xalan-j>에 접속





# JSTL(JSP Standard Tag Library)

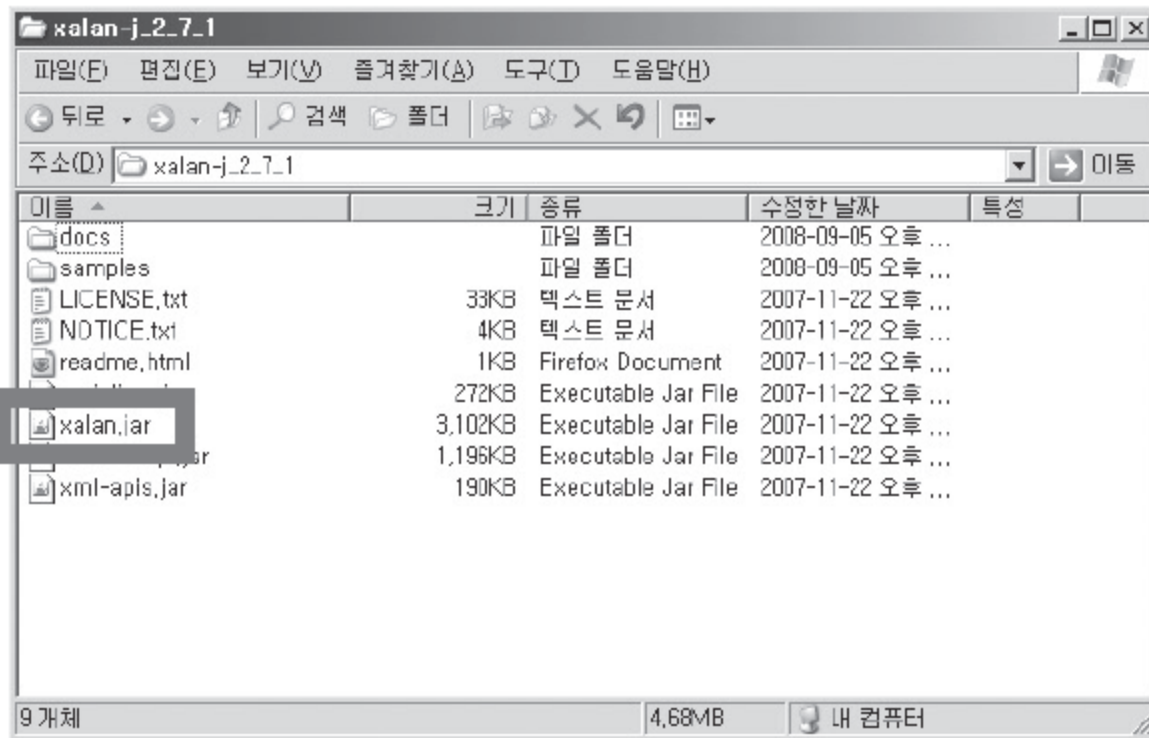
- 02. 원하는 링크를 선택하고 'xalan-j\_2\_7\_1-bin.zip' 파일을 다운로드 한다.

The left screenshot shows the 'Apache Download Mirrors' page. The URL bar displays 'http://www.apache.org/dyn/closer.cgi/xml/xalan-j'. The page content includes a list of mirror sites under the 'HTTP' section, with 'http://mirror.apache-kr.org/xml/xalan-j' highlighted. Below this, there is an 'FTP' section and a 'Backup Sites' section. The right screenshot shows the 'Index of /pub/Apache/xml/xalan-j/' directory listing. The table lists various files and directories, with 'xalan-j\_2\_7\_1-bin.zip' highlighted. The table has columns for Name, Last Modified, Size, and Type.

Name	Last Modified	Size	Type
Parent Directory/	-	-	Directory
binaries/	2007-Nov-28 15:12:33	-	Directory
source/	2007-Nov-28 05:43:49	-	Directory
KEYS	2007-Nov-28 06:48:00	23.2K	text/plain
xalan-j_2_7_1-bin-2jars.tar.gz	2007-Nov-28 04:20:34	12.4K	text/plain
xalan-j_2_7_1-bin-2jars.tar.gz.asc	2007-Nov-28 04:20:34	0.1K	text/plain
xalan-j_2_7_1-bin-2jars.zip	2007-Nov-28 04:25:31	15.7M	application/zip
xalan-j_2_7_1-bin-2jars.zip.asc	2007-Nov-28 04:25:32	0.1K	text/plain
xalan-j_2_7_1-bin.zip	2007-Nov-28 04:30:45	16.5M	application/zip
xalan-j_2_7_1-bin.zip.asc	2007-Nov-28 04:30:46	0.1K	text/plain
xalan-j_2_7_1-src-2jars.tar.gz	2007-Nov-28 04:38:57	5.9M	text/plain
xalan-j_2_7_1-src-2jars.tar.gz.asc	2007-Nov-28 04:38:58	0.1K	text/plain
xalan-j_2_7_1-src-2jars.zip	2007-Nov-28 04:41:15	7.2M	application/zip
xalan-j_2_7_1-src-2jars.zip.asc	2007-Nov-28 04:41:15	0.1K	text/plain
xalan-j_2_7_1-src.tar.gz	2007-Nov-28 04:43:04	5.9M	text/plain
xalan-j_2_7_1-src.tar.gz.asc	2007-Nov-28 04:43:04	0.1K	text/plain
xalan-j_2_7_1-src.sip	2007-Nov-28 04:45:22	7.2M	application/zip
xalan-j_2_7_1-src.zip.asc	2007-Nov-28 04:45:22	0.1K	text/plain

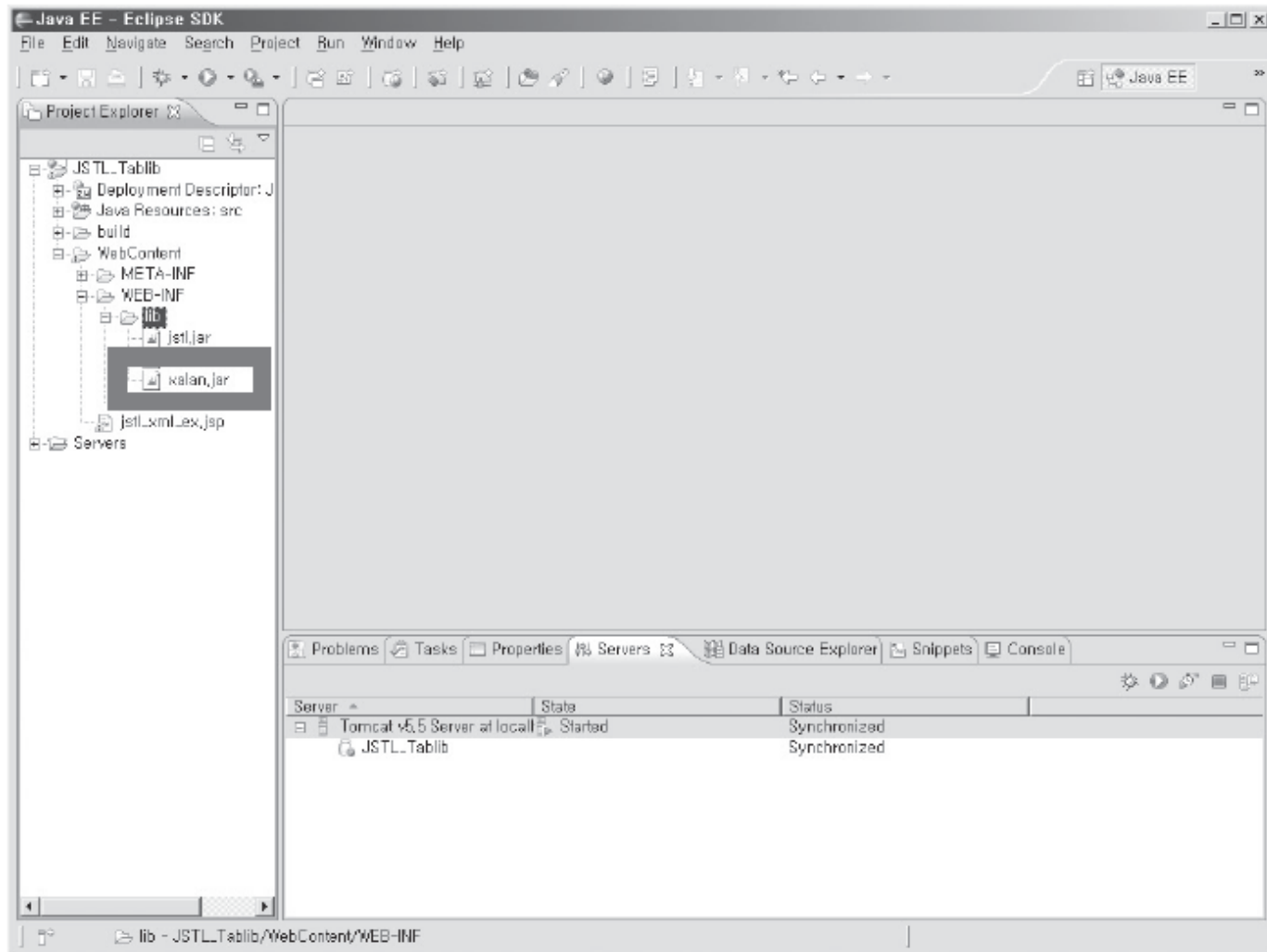
# JSTL(JSP Standard Tag Library)

- 03. 압축을 풀면 xalan.jar 파일이 존재하는 것을 확인할 수 있다.



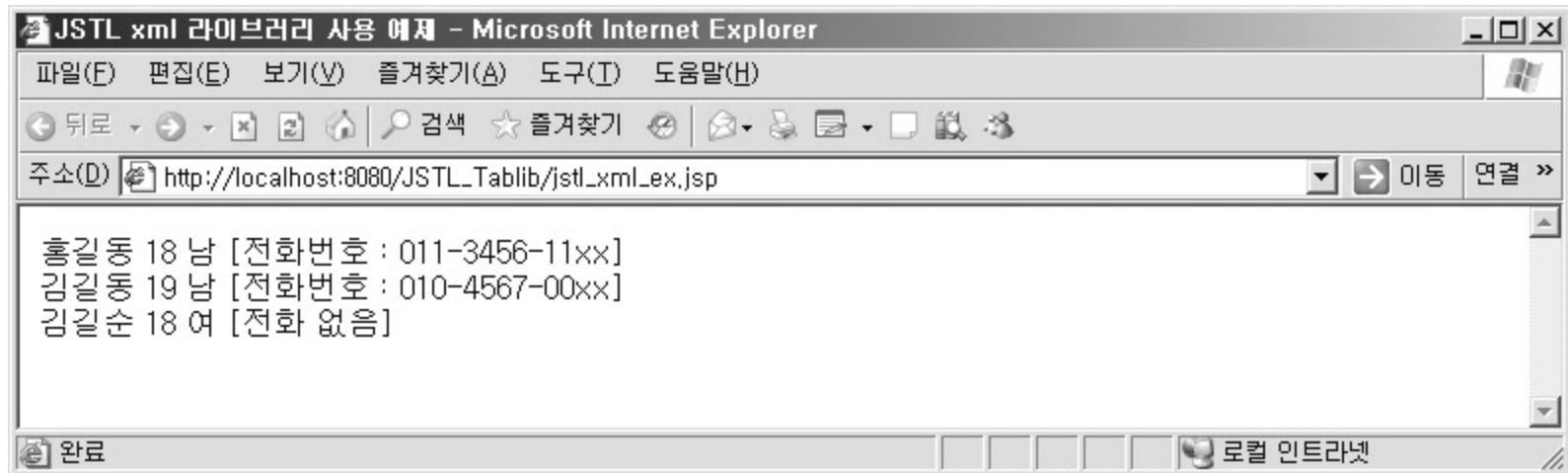
# JSTL(JSP Standard Tag Library)

- 04. 이제 xalan.jar를 이클립스 프로젝트의 라이브러리 폴더에 복사한다. xalan.jar 파일을 라이브러리에 추가한 후 톰캣을 재시작하여 작성한 코드를 다시 실행한다.



# JSTL(JSP Standard Tag Library)

- 05. 이제 어떤 오류도 발생하지 않게 된다. XPath를 사용할 때는 반드시 xalan.jar 파일을 라이브러리 폴더에 추가하는 것을 잊지 않도록 한다.



# JSTL(JSP Standard Tag Library)

## □ JSTL의 SQL 액션 - JSTL sql

### ■ JSTL sql

- SQL 관련 기능을 제공하는 JSTL 라이브러리
- sql 라이브러리를 이용해서 데이터베이스 서버에 접근할 수 있으며, 쿼리를 전송할 수도 있다. 또 트랜잭션 처리도 가능하다.

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

### ■ sql 라이브러리 태그 알아보기

- 데이터베이스 연결 태그 : <sql:setDataSource>
- 쿼리 전송 관련 태그 : <sql:query>, <sql:update>, <sql:param>, <sql:dateParam>
- 트랜잭션 태그 : <sql:transaction>



# JSTL(JSP Standard Tag Library)

- **<sql:setDataSource>**

- 데이터베이스 서버에 접근하기 위해 존재
- 접근 방법 두 가지 방법이 있다. dataSource속성을 이용하거나 driver, url, user, password 속성을 이용하는 방법
- dataSource 속성을 사용할 때는 미리 작성해 둔 JNDI 리소스가 존재해야 한다. 리소스를 그대로 가져다 사용할 경우 dataSource 속성만 이용하면 편리하게 데이터베이스 서버에 접근할 수 있다.
- JNDI 리소스가 존재하지 않을 경우에는 직접 드라이버 이름과, URL 및 아이디, 비밀번호를 설정하여 DB 서버에 접속할 수 있다.

```
<sql:setDataSource var="변수명" scope="범위" dataSource="dataSource" driver="driver" url="url"
user="user" password="password">
```

- **<sql:query>**

- 데이터베이스 서버에 쿼리를 전송할 때 사용
- dataSource 속성에는 JNDI 리소스나 <sql:setDataSource>로 데이터베이스 서버에 접속한 연결 정보를 얻어온 변수를 설정
- sql속성에는 실행할 SQL문을 지정
- var 속성의 변수에는 SQL 쿼리가 실행된 결과를 저장
- startRow는 얻어온 쿼리결과인 시작 행 값을 의미. 1번째 레코드 값부터 시작하려면 0으로 설정한다.
- maxRows는 얻어 온 쿼리 결과의 레코드 최대 수를 의미



# JSTL(JSP Standard Tag Library)

```
<sql:query var="변수명" scope="범위" sql="sql" dataSource="dataSource" startRow="startRow"
maxRows="maxRows">
```

- **<sql:update>**

- 주로 레코드의 추가, 수정, 삭제 기능을 사용
- var 속성에 입력한 변수에는 레코드를 업데이트한 결과가 저장
- sql 속성에는 레코드를 업데이트할 SQL문을 지정

```
<sql:update var="변수명" scope="범위" sql="sql" dataSource="dataSource">
```

- **<sql:param>**

- <sql:query> 태그나 <sql:update> 태그로 SQL문장을 전송할 때 파라미터를 전달해주는 태그
- 예를 들어 'SELECT \* FROM STUDENT WHERE NAME=?' 쿼리를 전송할 때 ? 부분의 값을 <sql:param> 태그를 사용하여 지정할 수 있다.

```
<sql:param value="value">
```



# JSTL(JSP Standard Tag Library)

- **<sql:dateParam>**

- <sql:param>과 같이 파라미터를 전달하는 기능을 제공
- 차이점은 <sql:dateParam>태그는 날짜 정보의 파라미터를 전달할 때 사용
- type 속성에는 date, time, timestamp 중 하나의 값이 올 수 있다.

```
<sql:param value="value">
```

- **<sql:transaction>**

- 데이터베이스 작업 시 트랜잭션을 사용하도록 해준다.
- dataSource 속성에는 JNDI 리소스 이름을 입력하거나 <sql:setDataSource>로 데이터베이스에 연결한 정보를 얻은 변수를 입력한다.
- isolation속성을 이용하여 격리 수준을 지정할 수 있다. 지정할 수 있는 속성 값으로는 read\_committed, read\_uncommitted, repeatable\_read, serializable을 입력할 수 있다.

```
<sql:transaction dataSource="dataSource" isolation="isolation">
```



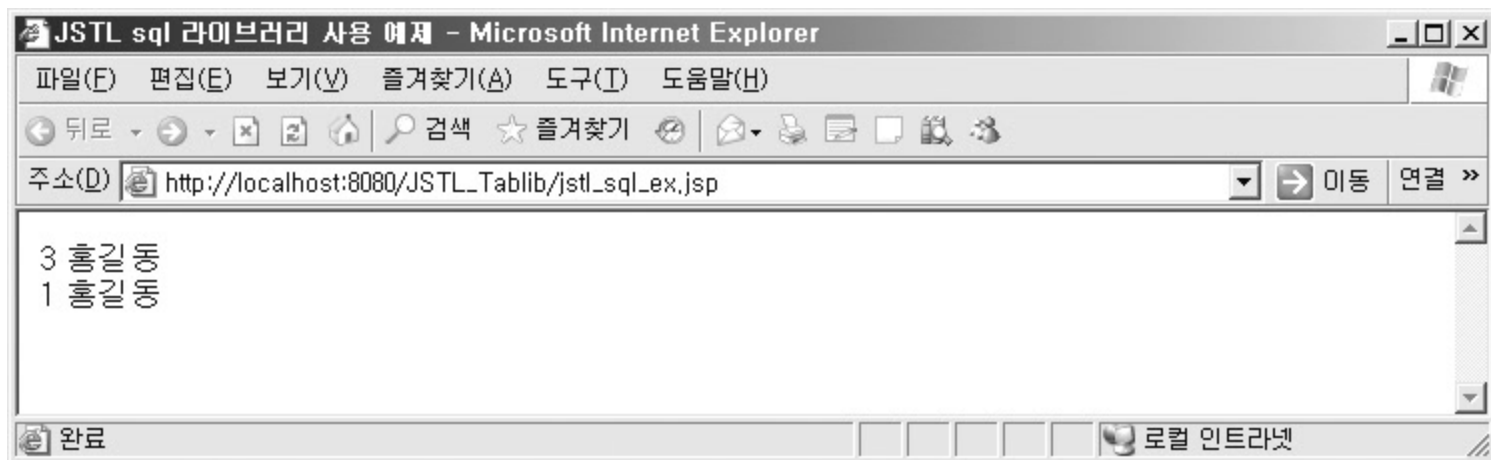


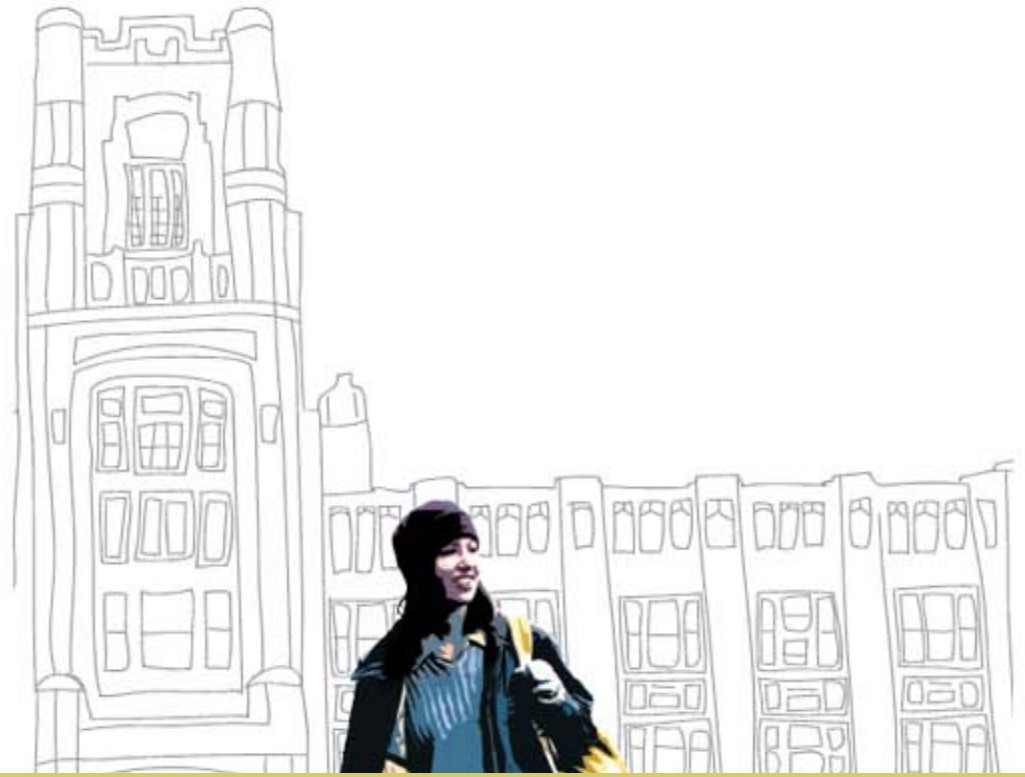
# JSTL(JSP Standard Tag Library)

- sql 라이브러리를 사용한 예제 작성하기
  - 01. sqlplus로 scott 계정에 접속하여 다음과 같은 테이블을 만들도록 한다.

```
CREATE TABLE test(  
  num NUMBER,  
  name varchar2(10),  
  primary key(num)  
);
```

- 02. jstl\_sql\_ex.jsp 소스를 작성 (교재 368p 참조)





**Thank You**