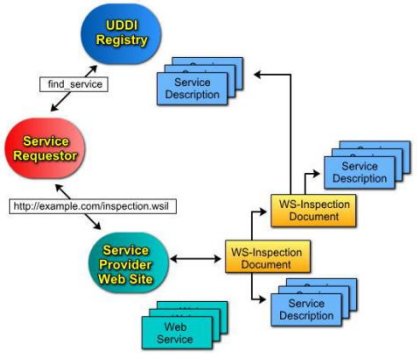
Web Application & Service



Expression language & JSTL

컴퓨터 과학과
김 희 열

표현 언어(Expression Language)

- 표현 언어란
  - 처음 JSTL(JSP Standard Tag Library)이 소개되었을 때 나온 것
  - MVC 패턴에 따라 뷰(view) 역할을 수행하는 JSP를 더욱 효과적으로 만들려는 목적
  - 간단한 방법으로 데이터를 표현하기 위해 고안된 언어인 SPEL(Simplest Possible Expression Language)에 기본을 둠

JSP 표현식 사용

```
<H2>
<jsp:useBean id="test" class="TestBean" />
<%= test.getName() %> 혹은 <jsp:getProperty name="test" property="name" />
</H2>
```

EL 사용

```
<H2>
${test.name}
</H2>
```

표현 언어

- 현재 페이지에서 출력하고자 하는 데이터(or 객체)가 미리 확보 되어 있어야 함
- page, request, application, session 내장 객체중 하나에 사용하고자 하는 객체가 있어야만 표현언어를 이용해 데이터 출력이 가능
- 표현언어의 기본적인 문법

  - 표현 언어는 '$'로 시작한다.
  - 모든 내용은 '{표현식}'과 같이 구성된다.
  - 표현식에는 기본적으로 변수 이름, 혹은 '객체_이름.멤버변수_이름'구조로 이루어진다.
  - 표현식에는 부가적으로 숫자, 문자열, boolean, null과 같은 상수 값도 올 수 있다.
  - 표현식에는 기본적인 연산을 할 수 있다.

표현 언어

- 표현 언어에서 사용할 수 있는 내장 객체
  - 표현언어에서는 객체가 생성되어 이미 전달되었다는 것을 가정
  - 표현언어 에서 사용 시점에 객체를 선언할 필요가 없음
  - 표현언어에서는 다음과 같이 객체에 접근 할 수 있음

    ${member.id} 혹은 ${member["id"]} → member 객체의 getId() 메서드 호출과 동일

    ${row[0]} → row라는 이름의 컬렉션 객체의 첫 번째 값

  - 또한 몇몇 내장객체를 통해 컨테이너가 제공하는 다른 객체에 접근할 수 있는 방법을 제공

표현 언어

[표 10-1] 표현 언어에서 사용할 수 있는 내장객체

내장객체	기능
pageScope	page 범위에 포함된 속성 값에 접근할 수 있는 객체다.
requestScope	request 범위에 포함된 속성 값에 접근할 수 있는 객체다.
sessionScope	session 범위에 포함된 속성 값에 접근할 수 있는 객체다.
applicationScope	application 범위에 포함된 속성 값에 접근할 수 있는 객체다.
param	request.getParameter("xxx")로 얻을 수 있는 값들이다. ${param.xxx}처럼 사용한다.
paramValues	request.getParameterValues("xxx")와 동일한 기능을 수행한다. ${paramValues.xxx}처럼 사용한다.
header	request.getHeader("xxx")와 동일한 기능을 수행한다. ${header.xxx}처럼 사용한다.
headerValues	request.getHeaderValues("xxx")와 동일한 기능을 수행한다. ${headerValues.xxx}처럼 사용한다.
initParam	컨텍스트의 초기화 매개변수 값이다.
cookie	쿠키 정보에 접근할 수 있는 객체다.
pageContext	pageContext 객체다.

표현 언어

- 표현 언어에서 사용할 수 있는 연산자
  - 표현식에서 기본적인 연산을 할 수 있으며 다음과 같은 연산자 사용이 가능함

[표 10-2] **산술 연산자**

연산자	기능	연산자	기능
+	더하기	-	빼기
*	곱하기	/ or div	나누기
% of mod	몫		

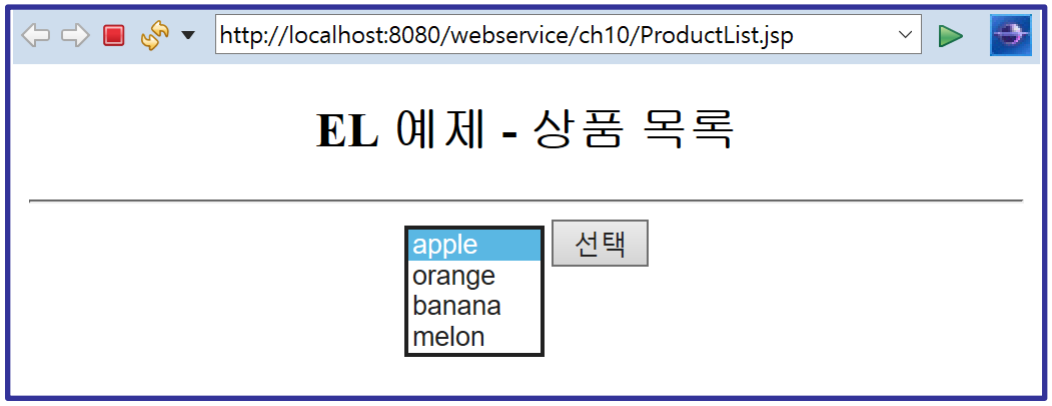
[표 10-3] **비교/조건 연산자**

연산자	기능	연산자	기능
== 혹은 eq	같다.	!= 혹은 ne	같지 않다.
< 혹은 lt	좌변이 우변보다 작다.	> 혹은 gt	좌변이 우변보다 크다.
<= 혹은 le	좌변이 우변보다 같거나 작다.	>= 혹은 ge	좌변이 우변보다 같거나 크다.
a?b :c	a가 참이면 b, 거짓이면 c를 반환한다.		

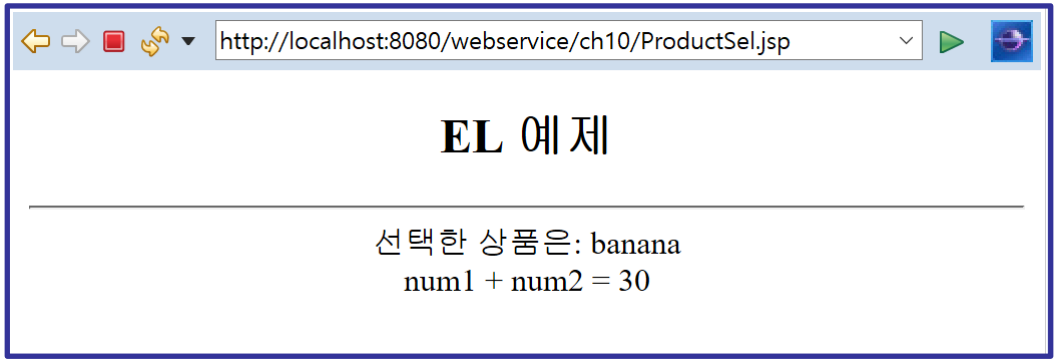
[표 10-4] **관계 연산자**

연산자	기능
&& 혹은 and	AND 연산
|| 혹은 or	OR 연산
! 혹은 not	NOT

실습



ProductList.jsp



ProductSel.jsp

실습

Product.java

```
package myapp.ch10;

public class Product {
    private String[] list = {"apple", "orange", "banana", "melon" };
    private int num1 = 10;
    private int num2 = 20;

    public String[] getList() {
        return list;
    }
    public int getNum1() {
        return num1;
    }
    public int getNum2() {
        return num2;
    }
}
```

실습

ProductList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<jsp:useBean id="product" class="myapp.ch10.Product" scope="session" />
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    body {text-align:center;}
</style>
</head>
<body>
    <h2>EL 예제 - 상품 목록</h2>
    <hr>
    <form name=form1 method=post action=ProductSel.jsp>
        <select name="sel">
            <%
                for (String item:product.getList())
                    out.println("<option>" + item + "</option>");
            %>
        </select>
        <input type=submit value="선택" />
    </form>
</body>
</html>
```

실습

ProductSel.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    body {text-align:center;}
</style>
</head>
<body>
    <h2>EL 예제</h2>
    <hr>
    선택한 상품은: ${param.sel} <br>
    num1 + num2 = ${product.num1 + product.num2}
</body>
</html>
```

커스텀 태그

- 커스텀 태그(Custom Tag)란?
  - 원래 JSP 페이지에서 반복적인 프로그램 로직을 캡슐화하기 위해 고안됨
  - 기본적으로 제공되는 태그 이외 사용자가 확장한 태그라는 의미에서 붙여진 이름
  - HTML 문서는 브라우저에 의해 해석되므로 커스텀 태그를 구현할 수 없지만, JSP는 서버에서 해석되므로 커스텀 태그를 구현할 수 있음

```
<table cellpadding=5 cellspacing=0 border="1">
    <tr bgcolor="#99CCFF"><td>번호</td><td>작성자</td><td>전화
    번호</td><td>작성일</td><td>내용</td></tr>
    <%
        for(GuestBook guestbook : datas) {
    %>
        <tr>
            <td><%=guestbook.getGb_id() %></td>
            <td><%=guestbook.getGb_name() %></td>
            <td><%=guestbook.getGb_tel() %></td>
            <td><%=guestbook.getGb_date() %></td>
        </tr>
        <%
            }
        %>
</table>
```

```
<%@ taglib uri="getListTag" prefix="boardTag" %>
<HTML>
<BODY>
<boardTag:getList />
</BODY>
</HTML>
```

커스텀 태그

- 웹 화면과 프로그램이 복잡해질수록 커스텀 태그를 사용하면 소스를 간결하게 유지할 수 있고 구현된 기능의 재활용이 용이해짐
- 커스텀태그 라이브러리 장점

❶ **비즈니스 로직으로부터 화면 표현을 분리할 수 있다** : 프로그램 개발자는 비즈니스 로직을 개발하고 표현을 위한 커프텀 태그를 제공하며, 화면 개발자는 화면 디자인에 집중할 수 있다.

❷ **비즈니스 로직의 캡슐화할 수 있다** : 재사용할 수 있고 유지보수에 유리하다.

❸ **보다 완벽한 MVC 패턴을 구현할 수 있다** : JSP를 순수하게 뷰 형태로 사용할 수 있으므로 더욱 완벽한 MVC 패턴을 구현할 수 있다.

커스텀 태그

- 커스텀 태그를 구현하는 방법
  - 태그 파일 기반의 개발 방법
  - 태그 핸들러 클래스 기반 개발 방법

- 태그 파일 기반 개발
  - 비교적 쉽게 jsp 기술들을 활용해 커스텀 태그를 만들 수 있음
  - 구조상 소스가 노출되고 라이브러리화 하기 어려움
  - 복잡한 태그나 라이브러리 개발의 경우에는 태그 핸들러 클래스 기반 개발이 좋음

[표 10-6] **유형별 커스텀 태그 개발 방법**

태그 개발 목적		개발 방법
• 많은 화면을 생성해내는 태그 • 단일 사이트에 적용	• 비교적 간단한 기능들로 구성	태그 파일 기반 개발 방법
• 다양한 기능 제공 • 여러 사이트에 적용	• 태그 라이브러리로 개발 • 공개 목적	핸들러 클래스 기반 개발 방법 (SimpleTag, SimpleTagSupport 등 태그)

커스텀 태그

- 태그의 기본 구조
  - 커스텀 태그도 일반적인 태그와 동일한 구조
  - 태그의 구성 요소는 태그, 속성, 태그 바디

```
<H2>custom tag test</H2>
<form name="form1" method="post">
<input type="text" name="item1">
</form>
```

  - 태그
    - 기본적으로 모든 태그는 쌍으로 이루어지며 속성을 포함하고 있다. <H2></H2>, <form></form>, <input>은 모두 태그인데, <input>의 경우 쌍으로 이루어지지 않았다.
  - 속성
    - 속성은 태그 안에서 부가적인 정보를 제공하는 것으로, <form> 태그 내에 있는 name과 method, <input> 태그에 있는 type과 name 등이 속성에 해당한다. HTML 자체에서는 제약이 크지 않으나, 가급적 속성 값들은 " "로 묶어주는 것이 좋다.
  - 태그 바디
    - 시작 태그와 종료 태그 사이에 있는 내용을 말한다.

커스텀 태그

- taglib 지시어
  - 커스텀 태그를 사용하기 위해서는 태그 사용을 원하는 jsp에 taglib 지시어를 기술
  - taglib 지시어는 태그에 대한 정보 수집을 위한 uri 혹은 태그파일 디렉토리와 태그에 붙이기 위한 prefix 정보를 등록한다.

```
<%@ taglib uri="/WEB-INF/tld/MsgTag.tld" prefix="mytag" %>
            ❶                              ❷
```

❶ uri : tld 파일 위치를 지정한다. 대개 태그 라이브러리를 관리하는 단체 uri가 온다. 로컬 디렉터리에 태그 라이브러리 기술자 파일이 위치하는 경우 경로를 넣어주면 된다. 태그 파일로 구현된 커스텀 태그를 사용하려면 uri 대신 tagdir을 쓴다.

❷ prefix : 커스텀 태그를 구분하기 위한 이름으로, 한 페이지에 커스텀 태그를 사용할 경우에는 prefix를 이용하면 혼동을 피할 수 있다. 동일한 이름의 태그가 성격이 다른 여러 태그 라이브러리에 있을 수 있기 때문에 이를 구분하기 위한 목적으로 사용하는 것이다.

  - 본문에서 태그를 사용할때에는 다음과 같이 prefix:태그_이름 형태로 사용한다.

```
<prefix:태그_이름> 태그 바디</prefix:태그_이름>
```

태그파일 기반 커스텀 태그

- 태그 파일 개요
  - 태그 파일을 이용하면 비교적 간단하고 JSP 페이지 개발과 유사한 구조로 태그 파일을 만들 수 있다.
  - tag 지시어를 사용해 태그 파일을 선언하고 JSP 문법과 표현언어, JSTL 등을 자유롭게 사용할 수 있다.
  - 기본적으로 태그 파일은 JSP와 유사하나 다음과 같은 차이가 있다.
    - **태그 파일** : .tag 파일로서, 몇 가지 제약사항을 제외하고는 대부분의 JSP 파일과 구성이 동일하다. [WEB-INF\tag] 폴더에 저장한다. 표현언어와 JSTL 등을 사용할 수 있다.
    - **JSP 파일** : 커스텀 태그를 사용하려면 JSP 파일에 taglib 지시어를 설정한 후 커스텀 태그를 사용한다.

태그파일 기반 커스텀 태그

- 태그 파일의 구조
  - 태그 파일은 실제 태그의 기능을 구현한 파일.
  - 기본적인 JSP 지시어와 추가된 tag 지시어 attribute 와 variable 이라는 태그를 사용할 수 있다.

[표 10-7] **태그 파일에서 사용하는 지시어**

지시어	설명
taglib	JSP에서의 taglib 지시어와 동일하다. 현재 태그 파일에서 다른 커스텀 태그나 JSTL 등을 사용하려면 해당 태그 라이브러리를 사용하기 위한 taglib 지시어를 작성한다.
include	JSP에서의 include 지시어와 동일하다. 그러나 포함되는 파일의 구조가 태그 파일의 구조를 따라야만 한다.
tag	새로운 지시어로, page 지시어와 유사하다. 현재 파일이 태그 파일이라는 것을 알리고 옵션을 설정한다.
attribute	작성하는 태그에 포함될 속성 등을 정의한다.
variable	태그 내용에 필요한 표현 언어 변수를 선언한다.

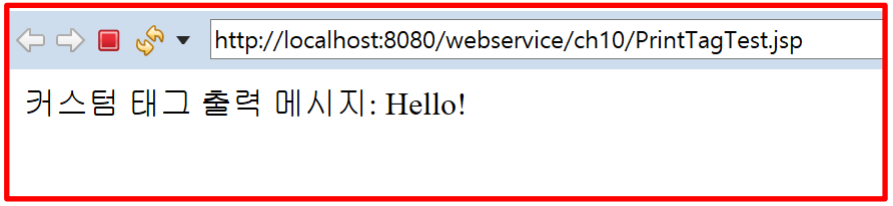
예 제

/WEB-INF/tags/print.tag

```
<%@ tag language="java" pageEncoding="UTF-8" body-content="empty" %>
커스텀 태그 출력 메시지:Hello!
```

PrintTagTest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http
<%@ taglib tagdir="/WEB-INF/tags" prefix="mytag" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <mytag:print />
</body>
</html>
```



실습

상품목록

- apple
- orange
- banana
- melon

ListItem.jsp

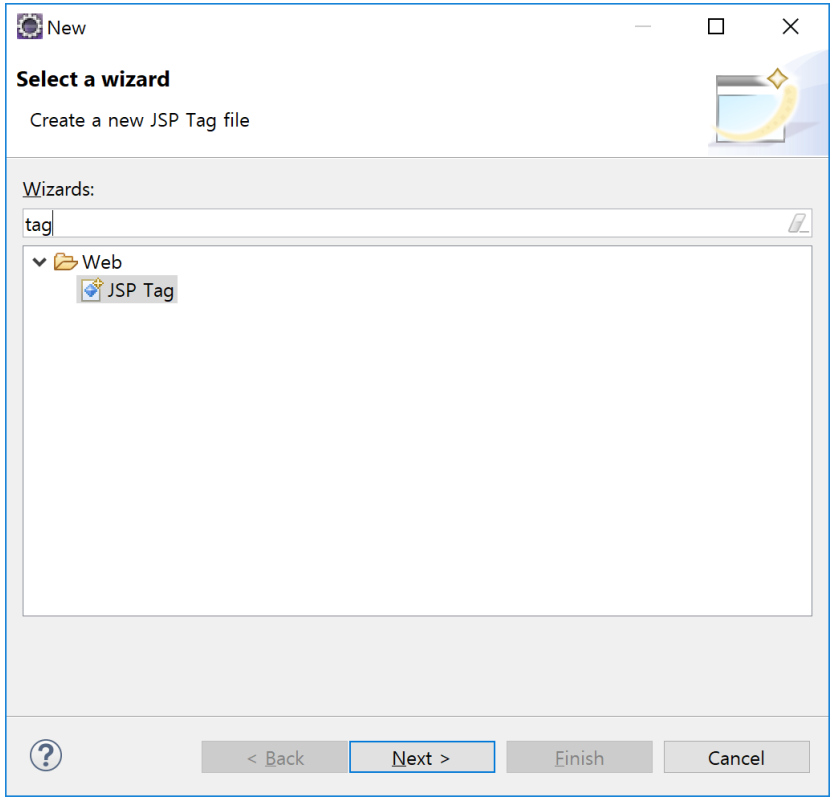
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http
<%@ taglib tagdir="/WEB-INF/tags" prefix="mytag" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <center>
    <mytag:item border="3" bgcolor="yellow">
        상품목록
    </mytag:item>
    </center>
</body>
</html>
```

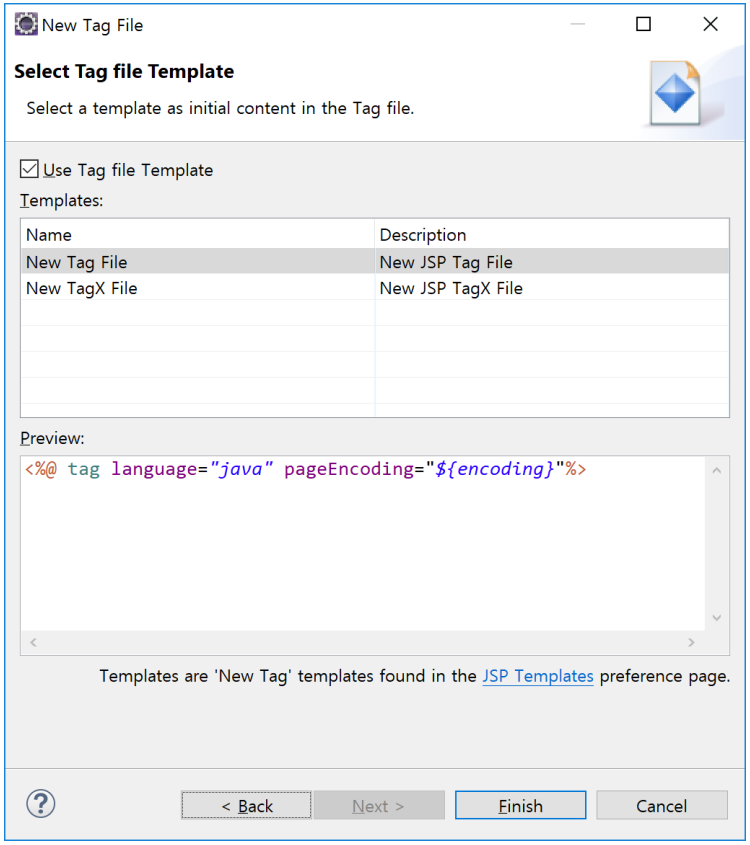
attributes

body

실습

- Tag 파일 생성


New
Select a wizard
Create a new JSP Tag file
Wizards:
tag
Web
JSP Tag
< Back
Next >
Finish
Cancel


New Tag File
Select Tag file Template
Select a template as initial content in the Tag file.
Use Tag file Template
Templates:
Name
Description
New Tag File
New JSP Tag File
New TagX File
New JSP TagX File
Preview:
<%@ tag language="java" pageEncoding="${encoding}"%>
Templates are 'New Tag' templates found in the JSP Templates preference page.
< Back
Next >
Finish
Cancel

실습

item.tag

```
<%@ tag language="java" pageEncoding="UTF-8" body-content="scriptless" %>

<%@ attribute name="bgcolor" %>
<%@ attribute name="border" %>

<jsp:useBean id="product" class="myapp.ch10.Product" />

<h2><jsp:doBody /></h2>

<table border="${border}" bgcolor=${bgcolor} width="150" >
    <%
        for (String item:product.getList())
            out.println("<tr><td>"+item+"</td></tr>");
    %>
</table>
```

태그 핸들러 기반 커스텀 태그

- 태그 핸들러 개요
  - 태그 핸들러란 커스텀 태그를 처리하는 객체를 말한다. 앞에서 배운 태그 파일과 달리 자바 클래스를 이용해 커스텀 태그를 구현하는 방법이다.
  - SimpleTag 인터페이스가 제공되면서 SimpleTagSupport 클래스를 상속받아 이전보다는 비교적 쉽게 구현할 수 있게 되었다. 그러나 태그 파일을 이용한 방식보다는 많이 복잡하고 자바 프로그램의 비중 이 높으므로 구현 난이도는 높은 편이다.
  - 태그 핸들러 기반 커스텀 태그는 세가지 구성요소를 가지며 태그 파일의 경우에는 이러한 부분이 간소화 되어 있다.

[표 10-11] **태그 핸들러 기반 커스텀 태그 구성요소**

구성요소	비고
태그 핸들러 클래스(Tag Handler Class)	태그 파일의 경우에는 별도 핸들러 클래스가 없다.
태그 라이브러리 기술자(Tag Library Descriptor)	태그 파일의 경우 태그 파일이 기술자를 겸한다.
taglib 지시어(taglib Directives)	커스텀 태그를 사용하는 모든 JSP에서 선언해야 한다.

태그 핸들러 기반 커스텀 태그

- **태그 핸들러 클래스**

커스텀 태그를 실제 구현한 자바 클래스다. 태그 라이브러리 기술자에서 설계된 내용을 구현해야 한다. 태그 라이브러리 기술자와 마찬가지로 태그 파일 기반의 커스텀 태그에서는 필요 없다.

- **태그 라이브러리 기술자(Tag Library Descriptor)**

xml 규격으로 커스텀 태그에 대한 구조를 정의하는 파일이다. .tld 파일로 만들어지며 태그 파일 기반의 커스텀 태그에서는 필요하지 않다.

- **taglib 지시어**

jsp 지시어의 한 종류로, JSP 페이지에 공통으로 필요한 정보를 기술하는 부분이다. 커스텀 태그 사용을 위한 태그 파일 혹은 태그 라이브러리 기술자의 위치 등을 설정한다. 따라서 커스텀 태그를 사용하는 모든 JSP 페이지에 taglib 지시어를 사용해야 한다.

태그 핸들러 기반 커스텀 태그

- 태그 핸들러 클래스 구조
  - 기본적으로 커스텀 태그 개발은 SimpleTag 인터페이스를 구현하는 것이지만 보통은 SimpleTag 인터페이스를 구현하고 있는 SimpleTagSupport 클래스를 상속받아 자신만의 커스텀 태그를 만들게 된다.

[표 10-12] SimpleTagSupport 클래스 주요 메서드

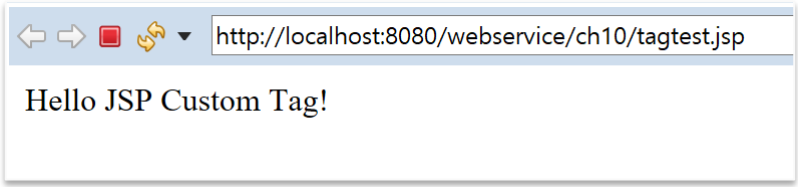
메서드	설명
`void doTag()`	시작 태그를 만나면 호출되는 메서드, 즉 실제 태그 기능을 정의하는 메서드다.
`JspFragment getJspBody()`	태그 바디를 처리하려고 JspFragmet를 얻어오기 위한 메서드다. JspFragment 객체를 이용해 바디를 처리한다.
`JspContext getJspContext()`	JspContext를 얻어오는 메서드로써, out 객체의 참조 등 해당 JSP로부터 다양한 정보를 얻을 수 있다.

  - 실제 개발에 가장 중요한 메서드는 doTag() 메서드로, 태그의 실제 기능을 구현하는 메서드라 할 수 있다.

태그 핸들러 기반 커스텀 태그

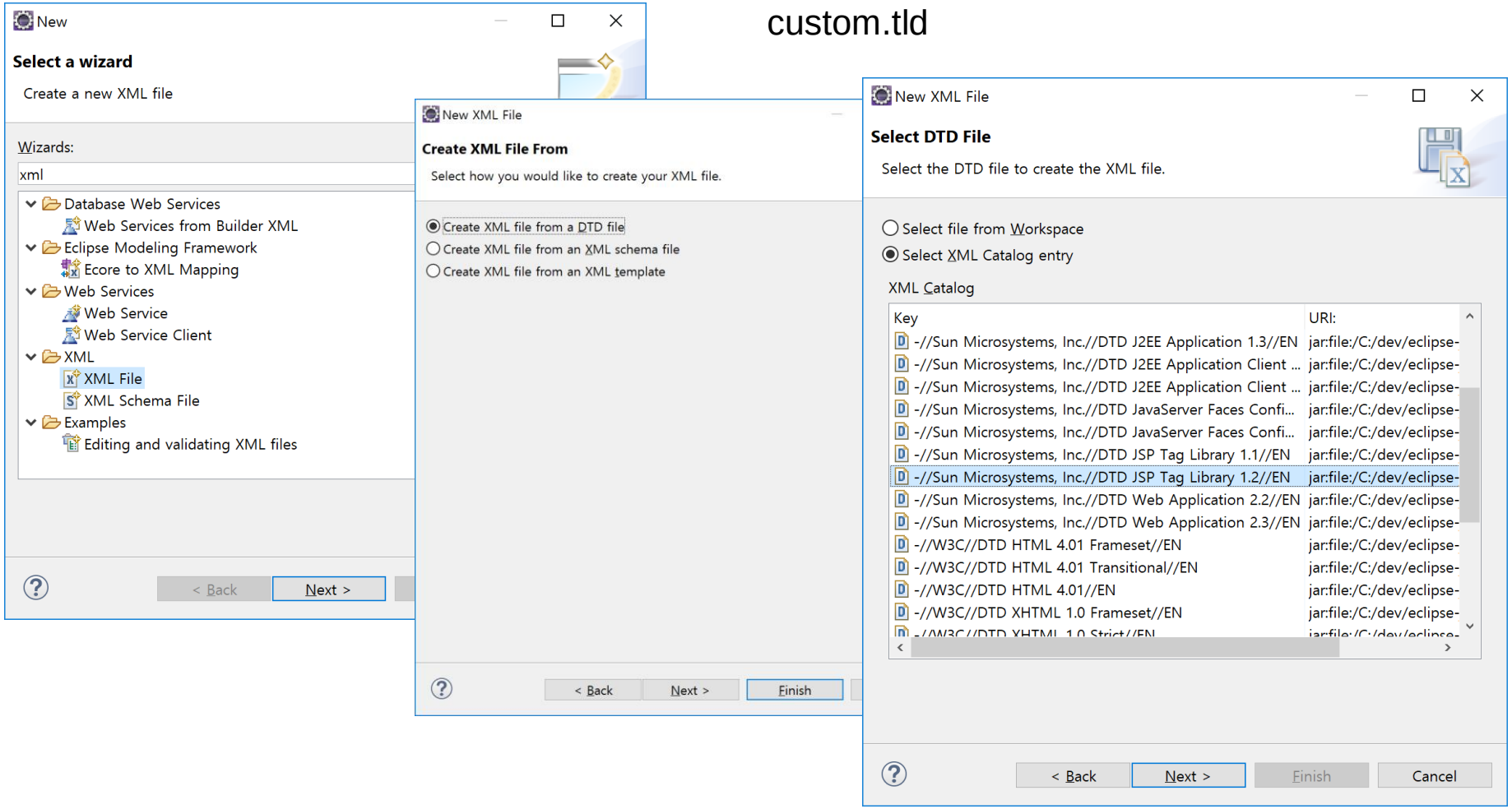
\ch10\tagtest.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="mytag" uri="/WEB-INF/custom.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <mytag:HelloWorld/>
</body>
</html>
```



태그 핸들러 기반 커스텀 태그

- \WEB-INF 아래에 tld 파일 생성



custom.tld

태그 핸들러 기반 커스텀 태그

- \WEB-INF 아래에 tld 파일 생성

custom.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD
<taglib>
  <tlib-version>tlib-version</tlib-version>
  <jsp-version>jsp-version</jsp-version>
  <short-name>my tags</short-name>
  <tag>
    <name>HelloWorld</name>
    <tag-class>myapp.ch10.HelloWorldTag</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

태그 핸들러 기반 커스텀 태그

- Tag handler 구현

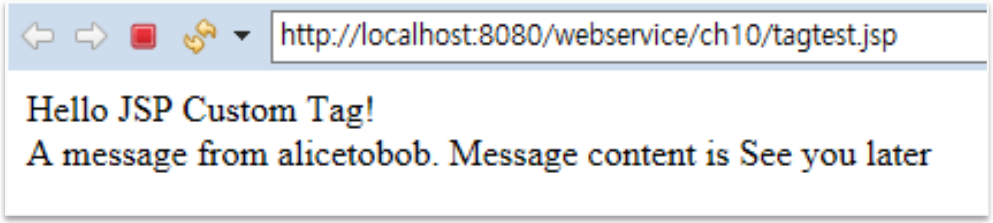
myapp.ch10.HelloWorldTag

```
package myapp.ch10;

import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HelloWorldTag extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {
        final JspWriter out = getJspContext().getOut();
        out.println("Hello JSP Custom Tag!");
    }
}
```

태그 핸들러 기반 커스텀 태그

ch10/tagtest.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="mytag" uri="/WEB-INF/custom.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <mytag:HelloWorld/>
    <br>
    <mytag:SendMessage from="alice" to="bob">See you later</mytag:SendMessage>
</body>
</html>
```



태그 핸들러 기반 커스텀 태그

- 기존 tld 파일에 다음 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>my tags</short-name>
  <tag>
    <name>HelloWorld</name>
    <tag-class>myapp.ch10.HelloWorldTag</tag-class>
    <body-content>empty</body-content>
  </tag>
  <tag>
    <name>SendMessage</name>
    <tag-class>myapp.ch10.SendMessageTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <name>from</name>
        <required>true</required>
    </attribute>
    <attribute>
        <name>to</name>
        <required>true</required>
    </attribute>
  </tag>
</taglib>
```

태그 핸들러 기반 커스텀 태그

- Tag Handler 구현

myapp.ch10.SendMessageTag

```
package myapp.ch10;

import java.io.IOException;
import java.io.StringWriter;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SendMessageTag extends SimpleTagSupport {
    private String  from;
    private String  to;

    public void setFrom(final String from) {
        this.from = from;
    }
    public void setTo(final String to) {
        this.to = to;
    }

    @Override
    public void doTag() throws JspException, IOException {
        final StringWriter sw = new StringWriter();
        getJspBody().invoke(sw);
        getJspContext().getOut().print("A message from " + from + "to" + to
                + ". Message content is " + sw.toString());
    }
}
```

JSTL의 개념과 구성

- JSTL이란?
  - JSTL(JSP Standard Tag Library)은 커스텀 태그 라이브러리 기술을 이용해서 일반적으로 필요한 기능들을 표준화한 것으로 크게 핵심(CORE), xml, I18N(국제화), 데이터베이스(SQL), 함수(functions) 라이브러리로 구성된다.
  - 커스텀 태그 기반이므로 JSTL을 사용하는 방법은 일반적인 커스텀 태그와 같다.
  - 다운로드 : http://tomcat.apache.org/taglibs/standard/
  - taglib 지시어 사용법(core 라이브러리의 경우)

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

[표 11-1] JSTL 라이브러리별 URI 및 prefix

라이브러리	URI	prefix
핵심	http://java.sun.com/jsp/jstl/core	c
XML	http://java.sun.com/jsp/jstl/xml	x
I18N	http://java.sun.com/jsp/jstl/fmt	fmt
데이터베이스	http://java.sun.com/jsp/jstl/sql	sql
함수	http://java.sun.com/jsp/jstl/functions	fn

JSTL의 개념과 구성

- JSTL 구성
  - JSTL은 태그가 제공하는 기능의 성격에 따라 5가지 주요 라이브러리로 구분된다.

[표 11-2] JSTL 라이브러리

라이브러리	기능	태그	접두어
핵심	General Purpose Actions	• catc • out • remove • set	c
	Conditional Actions	• choose • when • otherwise • if	
	Iterator Actions	• forEach • forTokens	
	URL Related Actions	• import • redirect • url • param	
XML	Core	• out • parse • set	x
	Flow Control	• choose • when • otherwise • forEach • if	
	Transformation	• transform • param	

JSTL의 개념과 구성

- I18N 은 다국어 처리와 관련된 기능을 제공하고 sql(데이터베이스)는 간단하게 데이터베이스 관련 작업을 지원하며 functions(함수)는 여러 부가기능들을 제공한다.

라이브러리	기능	태그	접두어
I18N	Locale	• setLocale	fmt
	Message Formatting	• bundle • message • param • setBundle • requestEncoding	
	Number and DateFormatting	• formatNumber • formatDate • parseDate • parseNumber • setTimeZone • timeZone	
데이터베이스	Database Access	• setDataSource • query • dateParam • param • transaction • update • dateParam • param	sql
함수		• ontains • containsIgnoreCase • endsWith • escapeXml • indexOf • join • length • replace • split • startsWithsubstring • substringAfter • substringBefore • toLowerCase • toUpperCase • trim	fn

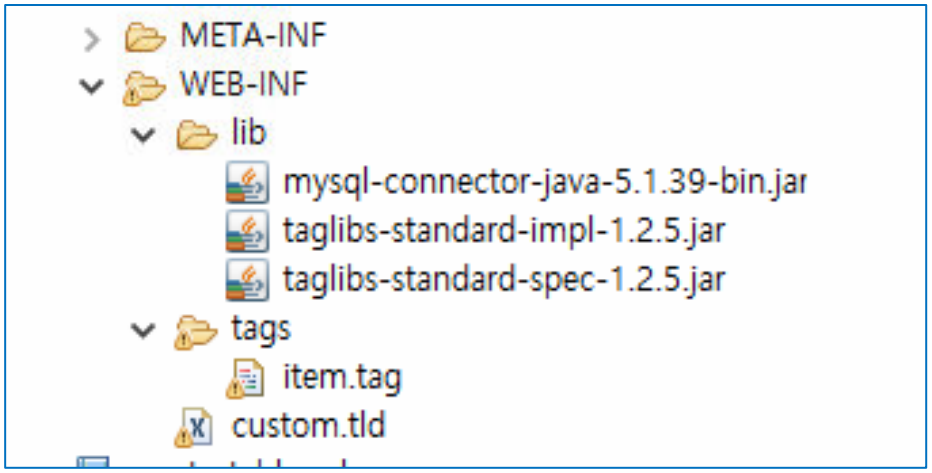
JSTL의 개념과 구성

- JSTL 설치

  tomcat.apache.org/taglibs/standard/에서 1.2 버전을 다운로드

  taglibs-standard-impl-1.2.5.jar , taglibs-standard-spec-1.2.5.jar 를

  \WEB-INF\lib 에 복사



핵심 라이브러리의 주요 태그

- <c:out> 태그
  - 초기에는 jsp 표현식(<%= %>)을 대체하기 위해 개발 되었으나 표현언어가 JSP에 기본으로 제공 되면서 사용 빈도는 줄었으나 몇몇 옵션은 유용하게 사용할 수 있다.
  - **사용법**
    - 태그 바디가 없는 경우

```
<c:out value="value" [escapeXml="{true|false}"] [default=defaultValue"]/>
```

    - 태그 바디가 있는 경우

```
<c:out value="value" [escapeXml="{true|false}"]>
default value (value에 내용이 없을 때 출력될 기본 값)
</c:out>
```

[표 11-3] **<c:out> 태그 속성 값**

속성	필수	기본 값	설명
value	Y	없음	출력될 내용 또는 표현식이다.
default	N	태그 바디에 있는 내용	value 값에 내용이 없는 경우에 출력할 내용으로, 태그 바디 혹은 속성 값 형태로 올 수 있다.
excapeXml	N	true	출력될 내용에 <, >, &, ', " 등의 문자를 일반 문자로 변환할 것인지 결정한다. 예를 들어 출력될 내용에 HTML 태그가 포함되어 있다면 이 값을 false로 해야 태그가 반영된 내용이 화면에 보인다. 만일 true로 할 경우 태그가 그대로 화면에 보이게 된다.

핵심 라이브러리의 주요 태그

- 14행의 <c:out value="${member.name}"/> 은 ${member.name}로 대체할 수 있음.

```
11 <table border="1" cellpadding=5 cellspacing=0>
12    <c:forEach var="member" items="${members}">
13       <tr>
14          <td><c:out value="${member.name}"/></td>
15          <td><c:out value="${member.email}" escapeXml="false">
16             <font color=red>email 없음</font>
17             </c:out>
18          </td>
19       </tr>
20    </c:forEach>
21 </table>
```

핵심 라이브러리의 주요 태그

- <c:set> 태그
  - <c:set> 태그는 변수 값을 설정하거나 객체의 멤버변수 값을 설정할 때 사용한다.
  - 사용법
    - 해당 범위에 속성 값을 추가하는 경우(바디가 올 수도 있다)

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

    - 특정 target 객체에 새로운 속성 값을 설정하는 경우(바디가 올 수도 있다)

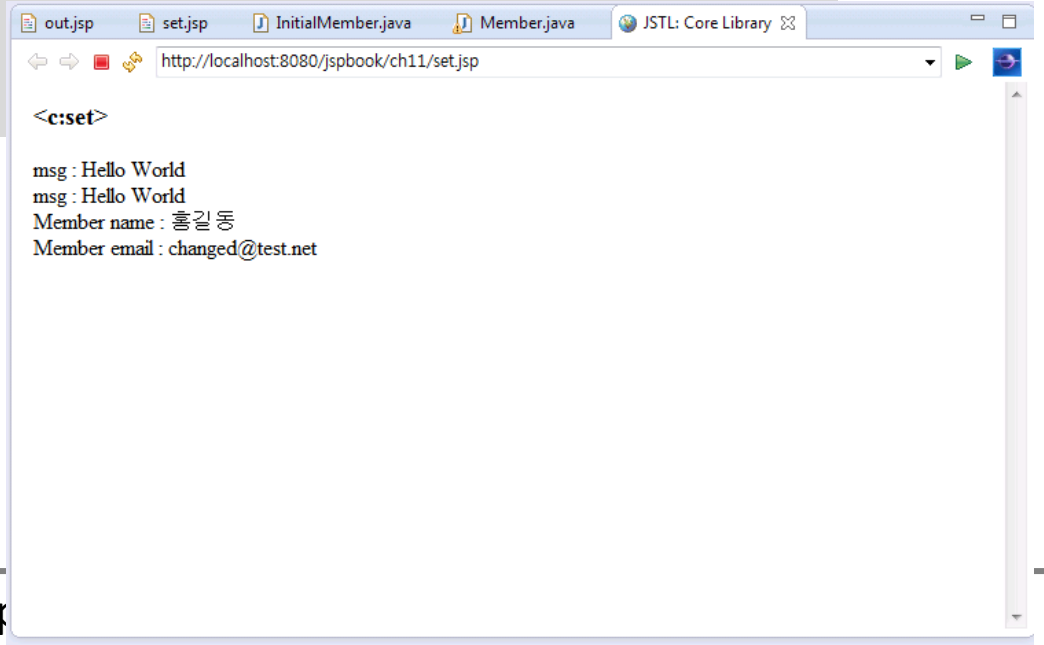
```
<c:set value="value" target="target" property="propertyName"/>
```

[표 11-4] <c:set> 태그 속성 값

속성	필수	기본 값	설명
value	N	없음	저장할 변수 값
target	N	없음	값이 저장될 객체 이름
property	N	없음	target 객체의 멤버변수 이름
var	N	없음	값이 저장될 변수 이름
scope	N	page	값이 저장될 범위(page, session, request, application)

핵심 라이브러리의 주요 태그

```
09 <h3>&lt;c:set&gt;</h3>
10 <c:set value="Hello World" var="msg"/>
11 msg : ${msg} <BR>
12 msg : <%=pageContext.getAttribute("msg") %><BR>
13
14 <c:set target="${member}" property="email" value="changed@test.net" />
15 Member name : ${member.name} <BR>
16 Member email : ${member.email}
```



핵심 라이브러리의 주요 태그

- <c:remove> 태그
  - <c:remove> 태그는 해당 scope 에 설정된 객체를 제거 한다.
  - **사용법**

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

[표 11-5] **<c:remove> 태그 속성 값**

속성	필수	기본 값	설명
var	Y	없음	삭제할 변수 이름
scope	N	모든 범위	삭제할 범위

핵심 라이브러리의 주요 태그

- <c:catch> 태그
  - <c:catch> 태그는 바디에서 실행되는 코드의 예외를 처리한다.
  - JSP를 뷰 역할에 충실하게 프로그래밍 한다면 크게 사용할 일은 많지 않다.
  - **사용법**

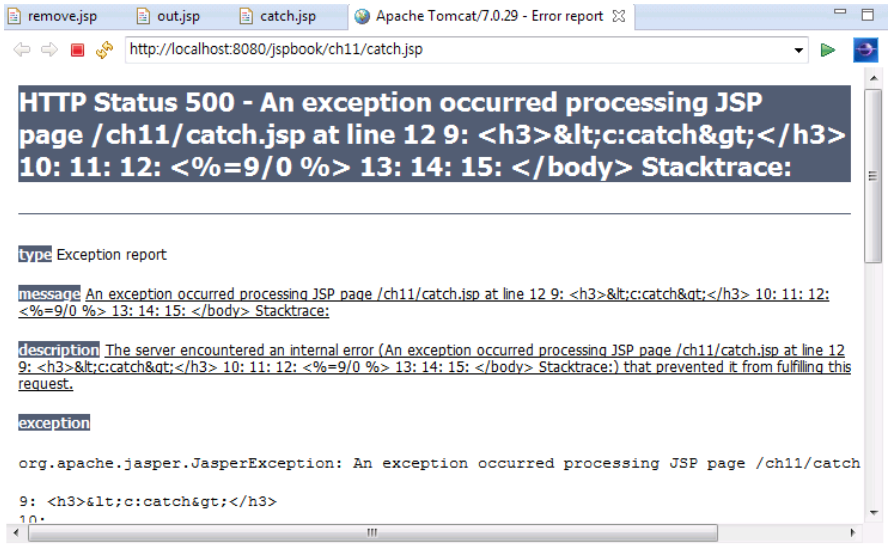
```
<c:catch [var="varName"]>
nested actions
</c:catch>
```

[표 11-6] **<c:catch> 태그 속성 값**

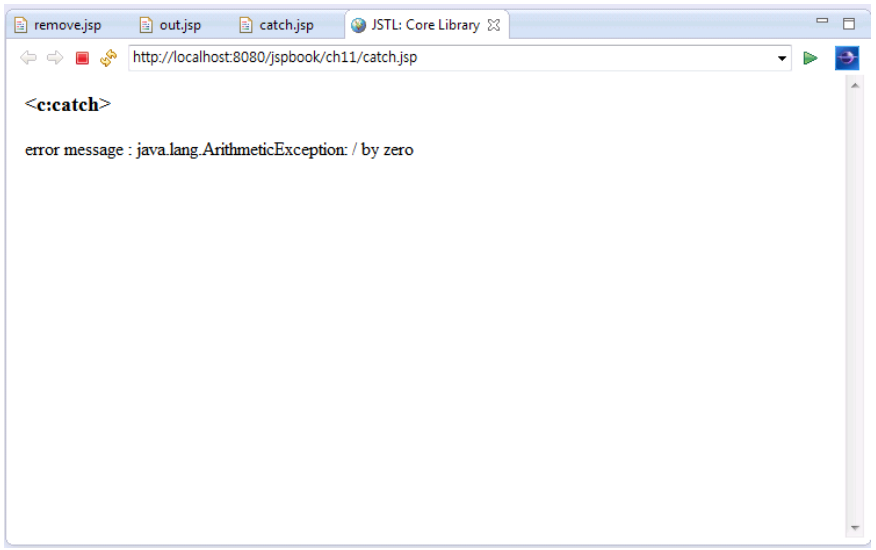
속성	필수	기본 값	설명
var	Y	없음	오류 메시지를 저장할 변수 이름

핵심 라이브러리의 주요 태그

```
09 <c:catch var="errMsg">
10 <%=9/0 %>
11 </c:catch>
```



[그림 11-5] 일반적인 예외 처리 화면



[그림 11-6] <c:catch> 태그를 사용한 경우

핵심 라이브러리의 주요 태그

- <c:forEach> 태그
  - 반복문과 관련된 태그로 자바의 for 문과 유사하다. 가장 중요하고 널리 쓰이는 JSTL 태그 중 하나임.
  - **사용법**
    - 컬렉션 객체의 크기만큼 반복

```
<c:forEach[var="varName"] items="collection" [varStatus="varStatusName"]
[begin="begin"] [end="end"] [step="step"]>
body content
</c:forEach>
```

    - 지정된 횟수 반복

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

핵심 라이브러리의 주요 태그

- varStatus는 반복 과정에서 프로그램적으로 필요한 여러 정보를 제공함.
- 자바 언어에서의 for와 같은 자유도는 없기 때문에 모든 반복 처리에 <c:forEach>를 사용하기는 어렵고 데이터 생성시 <c:forEach>에서 처리되기 편리한 형태로 가공해주는 것이 필요하다.

[표 11-9] <c:forEach> 속성 값

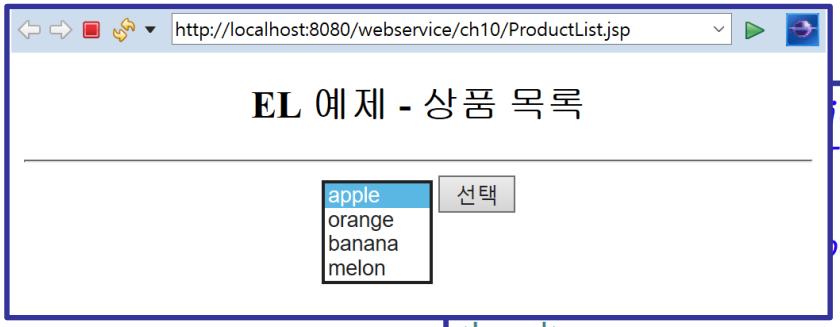
속성	필수	기본 값	설명
items	N	없음	반복을 위한 데이터를 가진 아이템의 컬렉션
begin	N	0	반복 시작 번호
end	N	컬렉션의 마지막 값	반복 끝 번호
step	N	1	반복의 증가분
var	N	없음	현재 아이템이 있는 변수
varStatus	N	없음	반복 상태 값이 있는 변수

핵심 라이브러리의 주요 태그

```
13 <jsp:useBean id="product" class="jspbook.ch10.Product" scope="session"/>
14 <select name="sel">
15 <%
16     for(String item : product.getProductList()) {
17         out.println("<option>"+item+"</option>");
18     }
19 %>
20 </select>
```

```
03 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
....
13 <jsp:useBean id="product" class="jspbook.ch10.Product" scope="session"/>
14 <select name="sel">
15     <c:forEach items="${product.productList}" var="item">
16         <option>${item}</option>
17     </c:forEach>
18 </select>
```

실습



ProductList.jsp

```
ava" contentType="text/html; charset=UTF-8"
F-8"%>

oduct" class="myapp.ch10.Product" scope="session" />

<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    body {text-align:center;}
</style>
</head>
<body>
    <h2>EL 예제 - 상품 목록</h2>
    <hr>
    <form name=form1 method=post action=ProductSel.jsp>
        <select name="sel">
            <%
                for (String item:product.getList())
                    out.println("<option>" + item + "</option>");
            %>
        </select>
        <input type=submit value="선택" />
    </form>
</body>
</html>
```

실습

forEach를 사용하는 ProductList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<jsp:useBean id="product" class="myapp.ch10.Product" scope="session" />
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    body {text-align:center;}
</style>
</head>
<body>
    <h2>EL 예제 - 상품 목록</h2>
    <hr>
    <form name=form1 method=post action=ProductSel.jsp>
        <select name="sel">
            <c:forEach items="${product.list}" var="item" >
                <option>${item}</option>
            </c:forEach>
        </select>
        <input type=submit value="선택" />
    </form>
</body>
</html>
```

핵심 라이브러리의 주요

- <c:choose> 태그
  - Java switch 문과 유사
  - <c:when>, <c:otherwise>와 함께 사용

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
    <c:when test="${income <= 1000}">
       Income is not good.
    </c:when>
    <c:when test="${income > 10000}">
        Income is very good.
    </c:when>
    <c:otherwise>
       Income is undetermined...
    </c:otherwise>
</c:choose>
</body>
</html>
```

```
Your income is : 16000
Income is very good.
```

핵심 라이브러리의 주요 태그

- <c:url> 태그
  - 적합한 URL 생성
- <c:param> 태그
  - URL에 parameter 추가

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:url value="/index1.jsp" var="completeURL"/>
 <c:param name="trackingId" value="786"/>
 <c:param name="user" value="Nakul"/>
</c:url>
${completeURL}
</body>
</html>
```

Output:

```
/JSP/index1.jsp?trackingId=786&user=Nakul
```