JSP 프로그래밍(3)

JSTL(JSP Standard Tag Library)

남 광 우

JSTL 개요

- JSTL(JSP Standard Tag Library)
  - JSP에서 자바의 각종 기능을 태그 형태로 사용할 수 있도록 표준으로 정의해 놓은 라이브러리
    - 자바나 JSP 언어를 쓰지 않고 HTML 상에서 불러다 쓸수 있음
  - 커스텀 태그(Custom Tag) 란?
    - 자신이 직접 정의할 수 있는 태그
    - 커스텀 태그의 장점
      - 라이브러리 형태로 만들어서 필요할 때마다 불러서 쓸수 있음
      - 자기가 직접 작성한 태그이기 때문에 유지보수 및 가독성 높음
      - 반복적 기능을 쉽게 구현가능
  - JSTL
    - 최대한 Java Code를 줄이기 위해 거의 모든 Java Code를 커스텀 태그 라이브러리로 지원하도록 표준화 한것

Custom Tag 개념

- 간단한 Custom Tag의 예

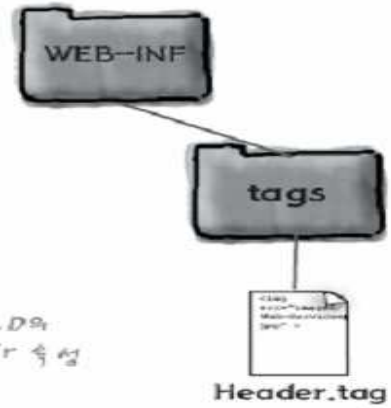
초 간단 태그 파일 사용 방법

① 포함할 파일(Header.jsp)을 복사해서 확장자를 .tag로 바꿉니다.


```
<img src="images/Web-Services.jpg" > <br>
```

이것이 전체 파일 내용입니다. <html>, <body> 태그는 모두 제거했습니다. 왜냐고요? 함에 나와는데... 처음 생성되는 JSP에 중복되면 안되잖아요.

② "WEB-INF" 밑에 "tags" 디렉토리를 새로 만들고 여기에 태그 파일("Header.tag")을 옮깁니다.



③ 태그 파일을 호출할 JSP를 만들고 taglib 지시자(tagdir 속성 포함)를 아래와 같이 작성합니다.

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

<html><body>

<myTags:Header/>

Welcome to our site.
</body></html>
```

태그 라이브러리를 식별하기 위하여 TLD의 uri를 사용했던 것처럼, 여기선 tagdir 속성을 사용합니다.

태그 이름이 바로 태그 파일 이름입니다. (tag는 생략합니다)

아래 코드 대신:

```
<jsp:include page= "Header.jsp" />
```

이제는 초 간단 버전:

```
<myTags:Header/>
```

Custom Tag 개념

- Tag 파일로 정보 전달하기

JSP에서 태그 호출

이전(<jsp:param>을 이용하여 요청 파라미터 설정)

```
<jsp:include page="Header.jsp">
   <jsp:param name="subTitle" value="We take the sting out of SOAP." />
</jsp:include>
```

이후(태그 속성)

```
<myTags:Header subTitle="We take the String out of SOAP" />
```

태그 파일에서 속성 사용하기

이전(요청 파라미터 값을 이용해서)

```
<em><strong>${param.subTitle}</strong></em>
```

이후(태그 파일 속성을 이용하면)

```
<em><strong>${subTitle}</strong></em> <br>
```

실제 태그 파일 코딩이고(포함될 파일 내용이란 말입니다).

Custom Tag 개념

- attribute 속성을 이용한 태그 속성 정의

태그 파일
(Header.tag)

속성이 옵션이 아니란 말이죠.

문자열도 될 수 있고, 표현식도 가능하다는 말.

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

<img src="images/Web-Services.jpg" > <br>
<em><strong>${subTitle}</strong></em> <br>
```

태그를 사용하는 JSP

```
<%@ taglib prefix="myTags" tagdir="/WEB-INF/tags" %>
<html><body>
<myTags:Header subTitle="We take the String out of SOAP" />

<br>
Contact us at: ${initParam.mainEmail}
</body></html>
```

Custom Tag 개념

- doBody 를 이용한 큰 내용 넘기기

태그 파일
(Header.tag)

이제 속성 지시자는 필요 없죠!

```
<img src="images/Web-Services.jpg" > <br>
<em><strong><jsp:doBody/></strong></em> <br>
```

무슨 의미냐 하면, "나를 호출한 태그 몸체에 있는 내용이 무엇이든 그걸 여기에 갖다 주세요"라는 뜻입니다.

태그를 사용하는 JSP

```
<%@ taglib prefix="myTags" tagdir="/WEB-INF/tags" %>
<html><body>

<myTags:Header>
   We take the sting out of SOAP. OK, so it's not Jini,<br>
   but we'll help you get through it with the least<br>
   frustration and hair loss.
</myTags:Header>
```

양이 많은 경우 시작 태그 속성에 기술하지 않고, 이제 몸체에 내용을 기술하면 됩니다.

```
<br>
Contact us at: ${initParam.mainEmail}
</body></html>
```

Custom Tag 개념

- body-content의 사용
  - 파라미터로 넘겨진 컨텐츠를 일반 텍스트로 취급

tag 지시자를 사용한 태그 파일
(Header.tag)

```
<%@ attribute name="fontColor" required="true" %>

<%@ tag body-content="tagdependent" %>

<img src="images/Web Services.jpg" > <br>
<em><strong><font color="${fontColor}"><jsp:doBody/></font></strong></em> <br>
```

이것의 의미는 몸체에 있는 컨텐츠를 일반 텍스트로 취급한다는 말입니다. 즉 EL, 태그, 스크립트를 실행(평가)하지 않고 텍스트로 포함하라는 말이죠. 여기에 들어갈 수 있는 값으로는 "empty", "scriptless"(디폴트)가 있습니다.

태그를 사용하는 JSP

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

<myTags:Header fontColor="#660099">
   We take the sting out of SOAP. OK, so it's not Jini,<br>
   but we'll help you get through it with the least<br>
   frustration and hair loss.
</myTags:Header>

<br>
Contact us at: ${initParam.mainEmail}
</body></html>
```

"fontColor"는 태그 파일에 attribute 지시자로 선언되어 있죠.

tag 지시자에 body-content 속성에 몸체를 사용할 수 있다고 정의했기 때문에 가능합니다.

Custom Tag 개념

- 배포 디렉토리에서 Tag들의 위치

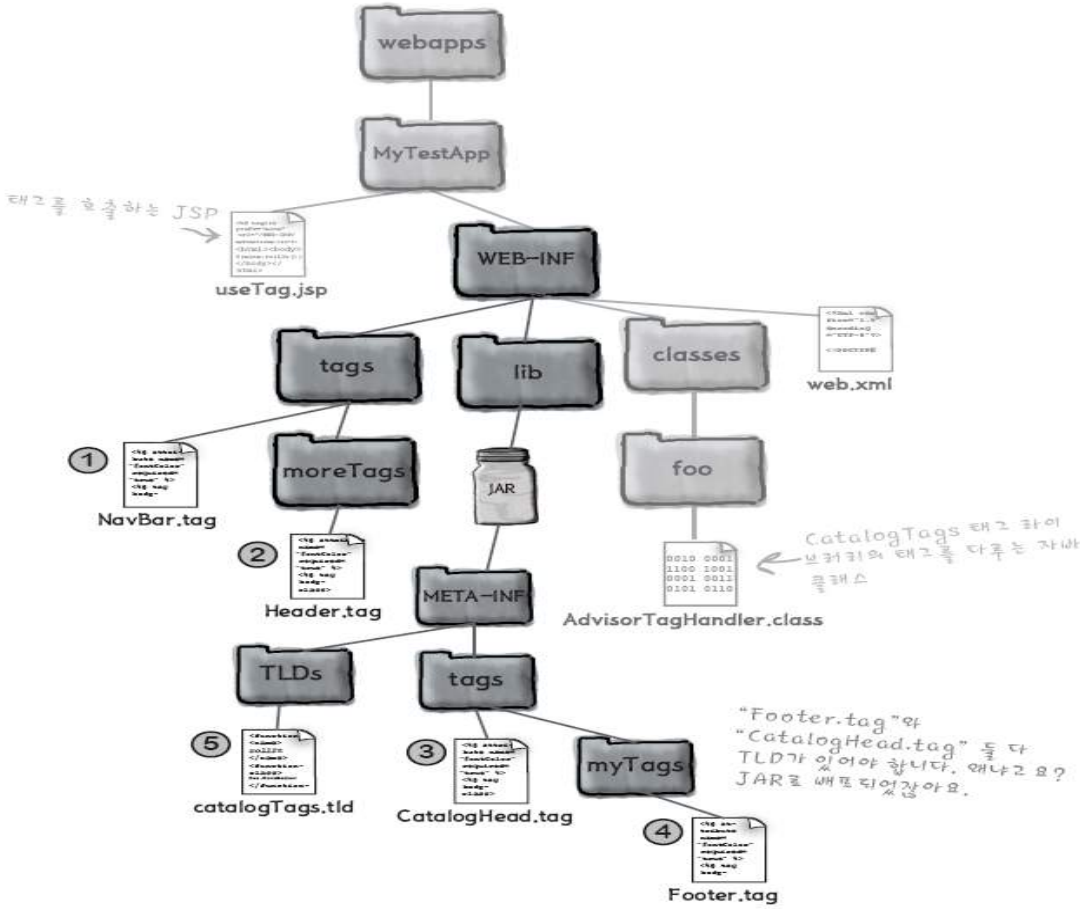
① WEB-INF/tags 바로 밑에

② WEB-INF/tags의 하위 디렉토리에

③ WEB-INF/lib에 JAR 파일로 배포되었다면, JAR 파일 META-INF/tags 밑에

④ WEB-INF/lib에 JAR 파일로 배포되었다면, JAR 파일 META-INF/tags의 하위 디렉토리에

⑤ *태그 파일이 JAR 파일로 배포되었다면, 반드시 TLD 파일이 있어야 합니다.*



Custom Tag 핸들러 개념

- 커스텀 태그 핸들러
  - 태그 실제 작업을 처리하는 간단한 자바 클래스
  - 태그 속성, 태그 몸체, request, response와 생존범위에 설정된 속성까지 pageContext를 통해 접근 가능
  - 형식 : 클래식(JSP 2.0 이전), 심플(JSP 2.0)

Custom Tag 핸들러 개념

커스텀 태그 핸들러의 사용

① SimpleTagSupport를 상속받아 클래스를 작성합니다.

```
package foo;
import javax.servlet.jsp.tagext.SimpleTagSupport;
// 필요한 import 구문을 넣으세요.

public class SimpleTagTest1 extends SimpleTagSupport {
    // 태그 핸들러 코드가 여기 들어갑니다.
}
```

② doTag() 메소드를 구현합니다.

```
public void doTag() throws JspException, IOException {
    getJspContext().getOut().print("This is the lamest use of a custom tag");
}
```

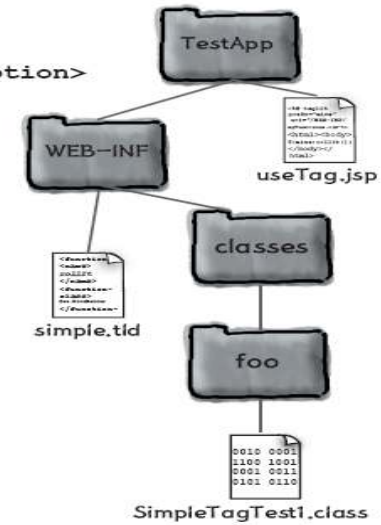
doTag() 메소드는 IOException이 정의되어 있기 때문에 이 문장을 try/catch로 둘러싸지 않아도 됩니다.

③ 태그를 위해서 TLD를 작성합니다.

```
<taglib ...>
  <tlib-version>1.2</tlib-version>
  <uri>simpleTags</uri>
  <tag>
      <description>worst use of a custom tag</description>
      <name>simple1</name>
      <tag-class>foo.SimpleTagTest1</tag-class>
      <body-content>empty</body-content>
  </tag>
</taglib>
```

④ 태그 핸들러와 TLD를 배포합니다.

WEB-INF 디렉토리에 TLD를 배포합니다. 그리고 태그 핸들러는 WEB-INF/classes에 패키지 구조에 맞추어 배포합니다. 태그 핸들러 클래스도 다른 웹 애플리케이션 자바 클래스와 같은 곳에 들어간다는 얘기죠.



⑤ 태그를 사용할 JSP를 작성합니다.

```
<%@ taglib prefix="myTags" uri="simpleTags" %>
<html><body>
<myTags:simple1/>
</body></html>
```

Custom Tag 핸들러 개념

- 커스텀 태그 핸들러의 사용

태그를 사용할 JSP

```
<%@ taglib prefix="myTags" uri="simpleTags" %>
<html><body>
Simple Tag 2:
<myTags:simple2>
    This is the body
</myTags:simple2>
</body></html>
```

이번에는 몸체를 작성해서 태그를 한 번 실행해 보죠.

*역자주: getJspBody()는 JspFragment를 리턴하고, JspFragment.invoke의 원형은 invoke(Writer)입니다. 인자 Writer가 널일 경우 JspContext.getOut() 메소드 리턴 값인 JspWriter로 출력합니다.

태그 핸들러 클래스

```
package foo;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import java.io.IOException;

public class SimpleTagTest2 extends SimpleTagSupport {

    public void doTag() throws JspException, IOException {
        getJspBody().invoke(null);
    }
}
```

코드 말, "태그 몸체를 읽은 다음 응답(Response)에 출력해 주세요" 인자가 널(null)이라는 것은 다른 Writer를 말고 Response로 출력하라는 의미입니다.*

TLD 파일

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd" ver-
sion="2.0">

  <tlib-version>1.2</tlib-version>
  <uri>simpleTags</uri>
  <tag>
     <description>marginally better use of a custom tag</description>
     <name>simple2</name>
     <tag-class>foo.SimpleTagTest2</tag-class>
     <body-content>scriptless</body-content>
  </tag>
</taglib>
```

코드 말, "몸체를 가질 수는 있으나, 스크립팅은 안돼요"라는 뜻. 스크립팅에는 스크립틀릿, 스크립팅 표현식, 선언문 등이 있지요.

Custom Tag 핸들러 개념

- 심플 태그 핸들러 작성
  - 태그에 몸체에 표현식을 사용하는 경우 (1)

JSP에서 태그 호출

```
<table>
  <myTags:simple4>
    <tr><td>${movie}</td></tr>
  </myTags:simple4>
</table>
```

태그가 호출되는 시점까지 "movie" 속성은 존재하지 않다가, 태그 핸들러가 이를 설정합니다. 태그 핸들러는 배열 크기만큼 루핑을 돌며 몸체를 호출합니다.

태그 핸들러 doTag() 메소드

```
String[] movies = {"Monsoon Wedding", "Saved!", "Fahrenheit 9/11"};

public void doTag() throws JspException, IOException {
   for(int i = 0; i < movies.length; i++) {
      getJspContext().setAttribute("movie", movies[i]);
      getJspBody().invoke(null);
   }
}
```

배열에 있는 값을 속성 "movie"로 묶어 둡니다.

몸체를 다시 호출합니다.

JSP

```
<myTags:simple4>
  <tr><td>
    ${movie}
  </td></tr>
</myTags:simple4>
```

태그 핸들러

```
for(int i = 0; i < movies.length; i++) {
   getJspContext().setAttribute("movie", movies[i]);
   getJspBody().invoke(null);
}
```

태그 핸들러가 루핑 돌며 "movie" 속성값을 재설정하고, getJspBody().invoke()를 호출합니다.

Custom Tag 핸들러 개념

- 심플 태그 핸들러 작성
  - 태그에 몸체에 표현식을 사용하는 경우

JSP에서 태그 호출

```
<table>
  <myTags:simple5  movieList="${movieCollection}">
    <tr>
      <td>${movie.name}</td>
      <td>${movie.genre}</td>
    </tr>
  </myTags:simple5>
</table>
```

다른 태그 속성을 설정하듯 똑같이 설정하면 됩니다. 태그 핸들러가 속성값을 읽어 뭔가 작업을 하겠죠.

태그 핸들러 doTag() 메소드

```
public class SimpleTagTest5 extends SimpleTagSupport {

  private List movieList;

    public void setMovieList(List movieList) {
        this.movieList=movieList;
    }

    public void doTag() throws JspException, IOException {
        Iterator i = movieList.iterator();
        while(i.hasNext()) {
            Movie movie = (Movie) i.next();
            getJspContext().setAttribute("movie", movie);
            getJspBody().invoke(null);
        }
    }
}
```

속성을 저장하기 위한 멤버 변수.

빈 스타일 속성 설정자를 작성합니다. 메소드명은 반드시 TLD에 있는 속성 이름과 일치해야 합니다(물론 "set" 빼고, 첫 글자를 소문자로 바꾼 다음에).

TLD 파일

```
<tag>
    <description>takes an attribute and iterates over body</description>
    <name>simple5</name>
    <tag-class>foo.SimpleTagTest5</tag-class>
    <body-content> scriptless </body-content>
    <attribute>
        <name>movieList</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

<tag>와 <attribute> 태그를 일반 속성 정의하듯 하면 됩니다.

Custom Tag 핸들러 개념

- SkipPageException

태그 핸들러 doTag() 메소드

```
public void doTag() throws JspException, IOException {
   getJspContext().getOut().print("Message from within doTag().<br>");
   getJspContext().getOut().print("About to throw a SkipPageException");
   if (thingsDontWork) {
     throw new SkipPageException();
   }
}
```

이 다음부터 나머지 태그 부분, 나머지 페이지 처리가 중지됩니다. 예외가 발생하기 바로 전까지 처리된 내용만 응답으로 내려가게됨.

태그를 호출하는 JSP

```
<%@ taglib prefix="myTags" uri="simpleTags" %>
<html><body>
About to invoke a tag that throws SkipPageException <br>
<myTags:simple6/>
<br>Back in the page after invoking the tag.
</body></html>
```

위에 있는 태그 핸들러 doTag() 메소드가 호출되겠죠.

JSTL

- JSTL의 기본적인 태그들
  - 일반
    - <c:out> 출력을 할때 쓰이는 태그.
    - <c:set> 변수를 지정하는 태그
    - <c:remove> 지정된 변수를 삭제하는 태그
    - <c:catch> 예외를 처리하는 태그
  - 조건
    - <c:if>
    - <c:choose>
    - <c:when>
    - <c:otherwise>
  - URL
    - <c:import>
    - <c:url>
    - <c:redirect>
    - <c:param>
  - 반복
    - <c:forEach>
    - <c:forEachToken>

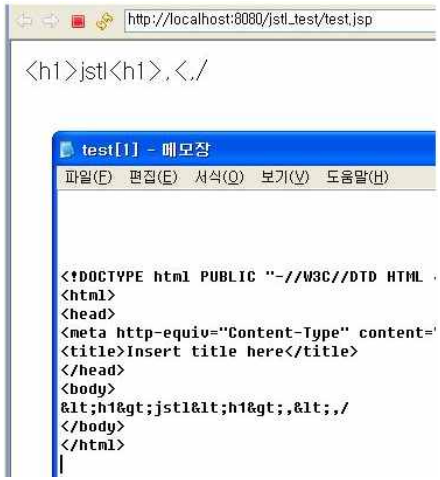
출처 : http://blog.naver.com/pksaladin/30078103589

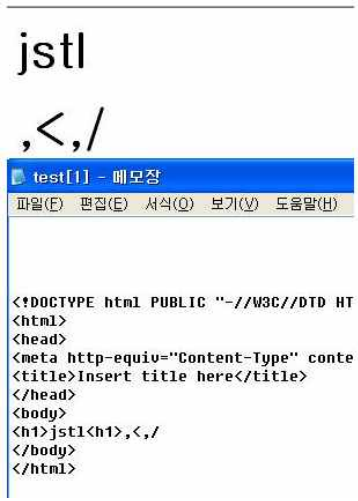
JSTL

- JSTL Core Library의 설정
  - <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

JSTL

- <c:out>
  - 지정된 값을 출력하는 태그
    - <c:out value="출력값" default="기본값" escapeXml="true or false">
    - 사용 예
      - <c:out value='${name}' default="jung">
      - 결과 : 당신의 이름은 jung 입니다
    - escapeXml
      - true = value 안의 html을 텍스트로 인식하여 그대로 출력
      - false = value 안의 값을 html 로 인식하여 출력
    - 사용 예
      - <c:out value="<h1>jstl<h1>,<,/" escapeXml='true'/>
      - <c:out value="<h1>jstl<h1>,<,/" escapeXml='false'/>





JSTL

- <c:set>
  - EL 변수 생성
    - 기존의 액션태그인<jsp:setProperty>와 비슷
  - 기본 형식
    - 몸체가 없는 형태 <c:set var="변수네임" value="값" scope="영역" />
    - 몸체가 있는 형태 <c:set var="변수네임">값</c:set>
      - var : 값을 지정할 EL변수의 이름
      - value : 변수값을 지정. 표현식 ,EL 정적 텍스트를 지정할 수 있음 (ex:"aa",%{aa},<%=aa>)
      - scope : 변수를 저장할 영역을 선택 page,session,application,request

JSTL

<c:set>의 예

```
<%@ page contentType="text/html; charset=euc-kr" %>
<%@ page import="jung.Info" %> <!--jung이라는 패키지안에 Info.java를 포함 -->
<%@ page import="java.util.HashMap" %><!--HashMap기능을 포함 -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <!--코어태그를쓰기 위해! -->
<%
        Info info = new Info(); //프로퍼티를 사용하기 위해 미리 만들어놓은 빈파일을 객체화
        HashMap<String, String> hmap = new HashMap<String, String>(); //해시 맵쓸준비
%>
<html>
<body>
<c:set var="info" value="<%= info %>" />
    <!--c:set으로 info라는 변수명지정 값은 위쪽에 생성한 빈파일 객체 -->
<c:set target="${info}" property="name" value="살라딘" />
    <!--프로퍼티기능으로 info에는 살라딘 라는 값이 등록됨 -->
<c:set var="hmap" value="<%= hmap %>" />
<c:set var="Youage" value="#{hmap.age}" />
    <!--Youage라는 변수지정 값은 해시맵객체의 age값 -->
회원 이름: ${info.name} <br></br>
    <!--빈파일을 통해 등록된 이름을 출력 -->
나이: ${Youage}<Br></Br>
    <!--등록된 나이를 출력(현재는 등록된 값이 없어서 null값임 -->
<c:set target="${hmap}" property="age" value="20" />
    <!-- 나이값 지정 -->
나이 값 지정시 출력: ${Youage} 세
</body>
</html>
```

```
package jung;
public class Info {
        private String name;
        private String age;
        public void setName(String name){
                this.name=name;
        }
        public void setAage(String age){
                this.age=age;
        }
        public String getName(){
                return name;
        }
        public String getAge(){
                return age;
        }
}
```

```
회원이름 : 살라딘
나이 :
나이값 지정시 출력 : 20세
```

JSTL

- <c:remove>
  - 지정된 변수를 삭제하는 태그
    - <c:remove var= "info"/>
    - <c:remove var ="hmap"/>

- <c:catch>
  - 예외를 처리하는 태그
    - 예외처리를 해서 에러메시지를 출력

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
<c:catch var="myerr"> <!-- 변수지정-->
<%int x = 10/0; %> <!-- 10으로 0을 나눴습니다;; -->
</c:catch>
<c:if test="${myerr !=null}">
${ myerr.message} <!-- 메시지 출력-->
</c:if>
</body>
</html>
```

JSTL

- <c:if>
  - 사용방법
    - <c:if test="조건>
      - 조건이 참일시 실행되는 부분
    - </c:if>

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<% int a =10; %> <!--비교할 숫자를 넣어줌 -->
<!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=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<c:set var="age" value="<%=a %>"/> <!--앞서배운 set을이용하여 age변수에 10을 넣어줍니다 -->
${age}
<c:if test="${age}==20"> <!--여기서 값을 비교합니다 . -->
숫자가 맞습니다.
</c:if>
숫자가 틀립니다.
</body>
</html>
```

JSTL

- <c:choose> <c:when> <c:otherwise>
  - choose태그는 자바의 switch 구문과 if - else부분을 혼합한 형

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<% int a =1; %>
<!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=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<c:set var="sex" value="<%=a %>"/>
<c:choose>
<c:when test="${sex==1}">
당신은 남자
</c:when>
<c:when test="${sex==2}">
당신은 여자
</c:when>
<c:otherwise>
그 외? 중성인가용?
</c:otherwise>
</c:choose>
</body>
</html>
```

JSTL

- <c:import>
  - URL결과를 읽어와서 현재위치에 삽입하는 태그
    - <jsp:include>와 비슷한 개념이지만 이 액션태그는 jsp의 동적인 자원을 포함하기 위해 만든것
    - <c:import url ="http://www.naver.com"/>
      - 위와 같이쓰고 태그를 실행하면 네이버의 메인 페이지가 삽입됨

- <c:url>
  - url을 재작성하기 위한 태그
    - <a href=" *<c:url value="test.jsp"/> "> test로가기*

- <c:param>
  - 파라미터 값을 넘기느 태그
    - <c:param name="파라미터 명" value="파라미터 값" />
      - <c:url var=*"nameSend" value="/test.jsp"*>
        - <c:param name=*"name" value="jung" />*
      - </c:url>
      - <a href =<c:out value="${*nameSend*}" />>*url*

JSTL

- <c:forEach>
  - for문 작성을 위해 사용되는 태그

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
String[ ] tList ={"aa","bb","cc"} //배열생성
request.setAttribute("tlist",tList); //배열을쓰기위해 속성지정
%>
<html>
<body>
<table border="1">
<c:forEach var="mo" items="${tlist}"><!-- 반복태그지정-->
        <tr>
        <td> ${mo} </td>
        <td> ${mo} </td>
        </tr>
</c:forEach>
</table>
</body>
</html>
```

결과화면

aa aa
bb bb
cc cc