JSP

Java Server Pages

Java의 특성을 물려받는다. ⇒ 플랫폼 독립적 + 보안↑ + 멀티스레드 (Servlet과 반대로) **HTML 문서에 Java 코드가 삽입되는 구조** → HTML 코드는 웹브라우저로 그대로 전송 → JSP 코드는 웹 컨테이너 쪽에서 실행되고 결과만 웹 브라우저로 전송 (Servlet에 비해 밑줄 친 단계가 추가된 것.)

- **JSP 페이지** –(변환)→ **Servlet 클래스의 소스 코드** –(컴파일)→ **Servlet 클래스 파일** –(인스턴스화)→ **Servlet 객체** –(초기화)→ **Servlet**

Java, Web

Servlet vs JSP

	장점	단점
Servlet	- HTML 문서에 소스코드가 공개되지 않는다.	- Java 코드안에 HTML이 작성된다. ⇒디자인시 수정에도 코드 변경 필요! - 설치과정이 복잡하다.
JSP	- HTML 중심의 코드 구조로 수정이 쉽다. - 설치과정이 쉽다.	- HTML 문서에서 소스 코드가 공개된다.

그러나 **복잡한 로직**을 필요로 하는 경우 Servlet을 사용해서, JSP 페이지에 결과만 출력하는 구조가 적합하다!

Servlet

Java를 기반으로 하는 웹 어플리케이션 프로그래밍 기술

기본 규격 (작성시 지켜야 할 3가지 규칙)

1. (**javax.servlet.http.HttpServlet** 를 상속받아) **javax.servlet.Servlet** 인터페이스를 구현해야 한다.
2. **doGet()**, **doPost()** 메소드의 선언 및 기능을 정의해야 한다.
   - Argument : (javax.servlet.http.httpServletRequest, javax.servlet.httpServletResponse)
   - Exception : java.io.IOException & javax.servlet.ServletException
3. 동적 HTML 문서를 생성하기 위해서는 doGet(), doPost()의 **두 번째 인자**를 이용해야 한다! 두번째 인자형 값에 getWriter() 메소드를 이용하면 PrinterWriter 객체가 반환되는 데, 이를 이용하여 print(), println(), printf()등의 메소드로 웹브라우저를 통해 HTML 문서에서 출력할 수 있다.

알아두어야 할 점

웹 컨테이너가 Servlet을 운영하는 방법으로

- 웹 서버가 여러 웹 브라우저(User)로부터 **동시 접속**을 받을 수 있고, 여러 Servlet이 **동시 실행** 될 수 있다.

그렇기 때문에 모든 웹 서버는 **Multi-Thread**로 작동한다!

- (동시 요청이 왔을 때) 웹 컨테이너가 만드는 Servlet의 개수
  - Multi-Thread Model : Thread 여러개로 하나의 Servlet으로 제어
  - ~~Single-Thread Model~~ : Thread 하나로 여러개의 Servlet 제어 → 자원 낭비! 대규모 요청시 심각! **~~2.4부터 폐용화.~~**
    - 구현 방법 : javax.servlet.SingleThreadModel 구현.

web.xml 설정 방법

- <url-class>에서 따로 package를 import 했으면 같이 적어줘야 한다! (패키지.클래스)
- <url-pattern>에서 첫시작에서 "/"를 빼먹지 말자!
  - “URL패턴”은 <form> 태그의 "action" 속성에 지정되고, Servlet을 대표하는 URL로 자유롭게 지정할 수 있다. (나머지 URL은 웹 애플리케이션 디렉토리로써 정확히 적어줘야 한다.)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
    <servlet>
        <servlet-name>NAME</servlet-name>                    <!-- 4, 8번줄 같아야 한다. -->
        <servlet-class>[PACKAGE.]SERVLET_NAME</servlet-class> <!-- 생성한 Servlet 이름 (Package 포함!) -->
    </servlet>
    <servlet-mapping>
        <servlet-name>NAME</servlet-name>                <!-- 4, 8번줄 같아야 한다. -->
        <url-pattern>/WEBAPPDIR/MyServlet</url-pattern> <!-- /웹 애플리케이션 디렉토리/"URL패턴" -->
    </servlet-mapping>
</web-app>
```

기초 지식

문법의 3종류

- JSP에서 변환된 명령문들은 doGet(), doPost() 역할을 동시에 하는 **_jspService()** 안에 들어간다!

<% %>

- Scripting Elements를 아예 쓰지 않고, Expression Language와 Ac ti on만 사용하는 것을 권장. (Web Designer를 위해)

스크립팅 요소 (Scripting Elements)

Scriptlet	Expression	선언부(Declaration)
<% 명령문 %>	<%= 식(호출된 결과값) %>	<%! 변수 및 메소드 선언 %>
<% int sum = 0; for(int i=0 ; i<10 ; i++) sum += i; %>	<%= Math.sqrt(sum) %> out.println(Math.sqrt(sum));	<%! final static int LIMIT = 99999 %> ※ 인스턴스 변수 선언 주의! 실행 중 변경안되도록 "final static" 붙이자!

```
<%@ page language="java" contentType="text/html" pageEncoding="UTF-8"%>
<!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>

<%
    int a = 3;
    int b = 4;
    int total = add(a, b);
%>

<%!
    private int add(int a, int b) {
        return a+b;
    }
%>

<%=a %> + <%=b %> = <%=total %>

</body>
</html>
```

지시자 (Directive) : <%@ %>

- 다른 문법들(Scripting Elements, Expression Language, Action)의 목적과 달리,
  웹 브라우저로부터의 요청을 처리하기 위해서가 아니라 **웹 컨테이너가 JSP 페이지를 Servlet 클래스로 변환할 때 필요한 여러 정보들을 기술**하기 위해 사용되는 문법

page

- JSP 페이지 전체에 적용되는 정보를 기술

Attribute	용도	예시
contentType	JSP가 생성하는 문서 종류 및 인코딩 타입	`<%@page contentType="text/html; charset=euc-kr" %>`
import	Scripting Elements 안에서 사용할 Java 클래스&인터페이스의 import	`<%@page import="java.util.*, java.io.*" %>`
buffer	출력 Buffer 크기	`<%@page buffer="4kb" /* none */ %>`
autoFlush	출력 버퍼가 모두 찼을 때의 동작	`<%@page autoFlush="false" %>`
isThreadSafe	Single-Thread로 동작 변경시 필요	`<%@page isThreadSafe="false" /* Single-Thread 동작*/ %>`
session	세션 참여 여부	`<%@page session="false" %>`
errorPage	Error 처리할 JSP 페이지 URL	Exception 처리
isErrorPage	Error 처리하는 JSP 페이지 여부	
isELIgnored	Expression Language 무시/처리 여부	

include

- 다른 JSP/HTML 문서를 **현재 JSP 문서의 일부로 만들기 위해** 사용.
- **file** 속성으로 **상대적 URL**를 이용한다.

```
...
<body>
  <%@include file="example.jsp" %>
</body>
...
```

taglib

- Action을 사용할 때, **Library가 필요할 때** 사용.

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

주석

Servlet 클래스로 컴파일 과정에서

JSP 주석(<%-- --%>)	HTML 주석() Java 주석(// /**/)
제거된다.	그대로 기록된다.

Expression Language : ${ }

- Expression Language

Action : XML Tag 형태

- Standard Action
- JSTL
- Custom Action

내장 변수 (Implicit variable)

- 선언하지 않아도 사용할 수 있는 변수
  ☞ **_jspServlet()**에 속하는 Parameter 변수와 Local 변수

변수명	Type	용도	예시
request	javax.servlet.http.HttpServletRequest	doGet(), doPost() 첫 번째 Parameter	`<%= request.setCharacterEncodeing("euc-kr"); %>` `<%= request.getParameter("NAME"); %>`
response	javax.servlet.http.HttpServletResponse	doGet(), doPost() 두 번째 Parameter	`<% response.sendRedirect("ex.jsp?RESULT="+msg+"&NUM=7") %>`
out	javax.servlet.jsp.JspWriter (PrintWriter 아님!)	웹 브라우저로 HTML 코드 출력	`<% out.println(" "); %>` `<% out.flush(); /* buffer 수동 비우기 */ %>`
session	javax.servlet.http.HttpSession	세션 관련 기능	Session 기술의 사용 방법
application	javax.servlet.ServletContext	JSP 문서가 속하는 Web App의 관련 기능	`<% String relPath = application.getContextPath(); String absPath = application.getRealpath("/RelativePath"); %>`
config	javax.servlet.ServletConfig	JSP 문서의 구성 정보 가져오는 기능	
pageContext	javax.servlet.jsp PageContext	JSP 문서 범위 내에서 사용할 수 있는 데이터저장 기능등	
page	java.lang.Object	JSP 문서로부터 생성된 Servlet	
exception	java.lang.Throwble	Exception 객체	Exception 처리

다른 JSP 문서 호출하기

- 가독성과 유지보수를 위해 JSP 문서를 여러 개로 나눴을 때 필요.

forward()

- 다른 JSP 문서를 호출할 때 사용.
- 실행 흐름의 제어를 되돌려주지 않음.
  ☞ 어떤 JSP 문서가 **할 일을 모두 마친** 후 다른 Web Resource(JSP, Servlet, HTML, ...)를 호출할 때 사용.
- 호출하는 문서에 **HTML 코드 출력하면 Exception 발생!**

```
<%
    RequestDispatcher dispatcher = request.getRequestDispatcher("ex.jsp");
//상대경로만 사용가능! 절대경로 불가능!
    //request.setAttribute("NAME", new Integer(7)); //String, Object
    dispatcher.forward(request, response);
%>
```

forward() vs sendRedirect()

- **핵심** : 웹 브라우저를 거치는가? (= 같은 Servlet 객체를 공유하는가? 아니면 새로 생성하는가?)

	forward()	**sendRedirect()**
호출 방식	웹 서버 쪽에서 **직접** ⇒자원 절약+응답 시간↑	URL을 웹 브라우저로 보내서 **간접적**
호출 경로	상대 경로**만**! = 같은 웹서버&디렉토리**만** 호출 가능. ⇒ 회선을 거치지 않고 **직접** 데이터를 전달하기 때문에 **보안↑**	상대 경로, 절대 경로 = 다른 웹 서버의 웹 자원 호출 가능. ⇒ 새 URL 유도에 사용 ⇒ "새로 고침"시 중복 동작 방지에 사용.
Data 전달 방식	request 내장 변수를 통해	URL 뒤에 붙여서
전송 가능한 Data Type	Object	String
한글 Data URL 인코딩 여부	불필요	필요

Data 송수신 예	//송신
	`request.setAttribute("Age", new Integer(11));`
	`RequestDispatcher dispatcher = request.getRequestDispatcher("Result.jsp");`
	`dispatcher.forward(request, response);`
	
	//수신
	`Integer age = (Integer)request.getAttribute("Age");`
		//송신
		`String str = URLEncoder.encode("안녕?", "EUC-KR");`
		`response.sendRedirect("Test.jsp?STR=" + str + "&NUM=11")`
		
		//수신
		`String str = reqeust.getParameter("STR");`
		`String num = reqeust.getParameter("NUM");`

include()

- 다른 웹 자원(JSP, Servlet, HTML, ...)를 호출하는 기능
- 호출된 웹 자원이 끝나고 나면 **실행 흐름의 제어가 본래의 웹 자원으로 돌아온다.**
  ☞ 여러 웹 자원이 **공통으로 사용하는 코드**를 호출 할 때 사용.

```
<%
    out.flush(); //출력 버퍼에서 JSP 코드 이전의 HTML 코드 제거.
    RequestDispatcher dispatcher = request.getRequestDispatcher("ex.jsp");
//상대경로만 사용가능! 절대경로 불가능!
    //request.setAttribute("NAME", new Integer(7)); //String, Object
    dispatcher.include(request, response);
%>
```

include() vs include 지시자

포함된 Web component가 별도의 Servlet class를 생성하는가?

include()	include 지시자
O	X[1)]

get vs post

입력된 데이터가 URL에

get	post
공개	비공개

Cookie vs Session

- 3개 이상의 화면으로 구성된 애플리케이션을 작성할 경우, 서로간의 **데이터 송수신** 처리를 위해 필요.

	Cookie	Session
저장 위치	Client	Server
저장 Type	String	Object
용량 제한	4KB, 한 Client(Domain) 당 20개	서버가 허용하는 한 용량, 개수 제한 없음
단일 기록량	1	多 (☞ Hashtable 사용)

적절한 사용 시점?

- **Cookie**
  - 다른 언어나 Platform끼리 연동하여 Data 송수신을 할 경우
- **Session**
  - Data 크기가 큰 경우

        - ☞ 용량이 큰 Cookie를 내보내면 Network 부하를 증가 시키게 되지만 Session은 Servlet이 수행되는 Server의 Memory 공간에 Data가 저장되므로 Client에서 Network 부하는 단지 Session ID 정도만 사용하게 됨.
    - 보안이 필요한 Data인 경우
        - ☞ Server에 저장되므로

정의

Web Component(JSP 페이지와 Servlet 클래스를 통칭)간의 **데이터를 송수신**이 가능케하는 기술.

- **Cookie** : 전달할 Data를 **Web Browser로 보냈다가** 웹 서버 쪽으로 되돌려 받는 방법. (Client)
    - Data 공유 범위 : 같은 웹 서버 내에 있는 모든 웹 컴포넌트들과 자바가 아닌 웹 애플리케이션 프로그램들 전체.
- **Session** : 전달할 실제 Data를 **Web Browser를 거치지 않고 Session ID만 보내어 웹 서버에 있는 데이터 영역**을 통해 데이터를 전달하는 방법. (Server)
  ☞ Web Browser(Client)로 Session ID를 전송하는 기술은 **Cookie!** (웹 브라우저로 세션아이디를 보낼 때 **JSSESSIONID**란 이름의 쿠키 형태로 만들어 전송한다.)
  ☞ Cookie에 비해 보안성 높음.
    - 여러 웹 컴포넌트들이 협력 작업을 시작해서 끝내기까지의 기간

- 보안상 중요한 데이터는 Session, 그렇지 않으면 Cookie 이용.
- Session Data를 전송하기 위해 사용하는 기술도 Cookie! (☞ 둘 다 알아야 한다!)
- Data 공유 범위 : 같은 웹 애플리케이션 디렉터리에 있는 웹 컴포넌트들

탄생 배경 - 왜 두 기술로 나뉘었는가?

WWW 등장 초창기에는 인프라 구축이 덜 되어 대규모의 동시 사용자를 위한 웹앱의 중간 데이터를 서버 쪽에 저장해 두는 것이 어려웠기 때문에 **Cookie** 기술이 먼저 개발되었다. (= Client측에서 처리)

Cookie 기술의 사용 방법

입력 & 수정

```
<%@page contentType="text/html; charset=euc-kr" %>
//HTML 코드 명령문보다 앞에 오는 것이 바람직. (Buffer 때문)
<%
  //두 Argument는 "String" Type! 수치값인 경우 String Data로 만들어야 한다!
  //쿠키명 같으면 "수정(Set)".
     response.addCookie(new Cookie("AGE", "20"));
%>

<html>
...
</html>
```

조회

```
<%@page contentType="text/html; charset=EUC-KR" %>
<%
    Cookie[] cookies = request.getCookies(); //배열!
%>
<html>
<head><title></title></head>
<body>
    Age : <%= getCookieValue(cookies, "AGE") %>
</body>
</html>

<%!
    private String getCookieValue(Cookie[] cookies, String name) {
        if(cookies == null)
            return null;
        for(Cookie cookie : cookies) {
            if(cookie.getName().equals(name)) {
                return cookie.getValue();
            }
        }
        return null;
    }
%>
```

삭제

```
<%@page contentType="text/html; charset=euc-kr" %>
<%
  //초단위. 수명을 0로 하면 삭제된다.
  //음수면 아래 메서드를 호출하지 않았을 때와 마찬가지로 "웹 브라우저가 끝날 때" 쿠키 제거
    Cookie cookie = new Cookie("Key", ""); //삭제할 쿠키이므로 Value는 아무렇게나 상관없음.
    cookie.setMaxAge(0);
    response.addCookie(cookie)'
%>

<html>
...
</html>
```

전송 범위 조정

쿠키가 특정 경로명을 갖는 URL로만 전송되도록 만드는 방법 (전송 범위 좁히기)

- 웹 브라우저는 웹 서버로 URL을 보낼 때, 그 웹 서버에 속하는 모든 쿠키를 함께 보내는 것이 기본 동작이지만, 이런 전송 범위를 좁혀야 할 경우가 있을 때 사용.

```
<%@page contentType="text/html; charset=euc-kr" %>
<%
    Cookie cookie = new Cookie("Age", "20");
    cookie.setPath("/local/"); //URL 경로명은 "반드시 /로 시작"해야하고, 마지막도 /로 끝내는 것을 권장.
    response.addCookie(cookie);
%>
<html>
...
</html>
```

쿠키가 여러 웹 서버로 전송되도록 만드는 방법 (전송 범위 넓히기)

- 웹 브라우저가 저장할 수 있는 쿠키의 수가 웹 서버별로 한정되어 있다. 남용시 데이터 손실되니 주의!

```
<%@page contentType="text/html; charset=euc-kr" %>
<%
    Cookie cookie = new Cookie("Age", "20");
    cookie.setDomain(".site.co.kr"); //대표 도메인 설정. "."로 시작해야 한다!
    response.addCookie(cookie);
%>
<html>
...
</html>
```

Session 기술의 사용 방법

- Session의 적용 범위는 **같은** 웹 애플리케이션 **디렉토리** 내에 있는 **웹 컴포넌트(JSP, Servlet)**로 국한된다. (= 디렉토리별로 독립적, 같은 디렉토리 내 Java 웹 애플리케이션끼리만 공유 가능)

Servlet 클래스에서 세션 기술을 사용하는 방법

```
HttpSession session = request.getSession(); //Session 시작/가져오기
```

```
session.setAttribute(String, Object); //데이터 저장
Object att = session.getAttribute(String); // 데이터 가져오기. 다른 타입은 Casting 필요.
session.removeAttribute(String); // 데이터 삭제
session.invalidate(); //세션 끝내기

int timeout = session.getMaxInactiveInterval(); //타임아웃 기간 반환
session.setMaxInactiveInterval(-1); //타임아웃 기간 설정.(초단위), 음수값이면 무한대
```

JSP 페이지에서 세션 기술을 사용하는 방법

- Servlet 클래스로 변환될 때, **HttpSession session = request.getSession();**이 자동으로 추가되므로 session 변수를 그냥 이용하면 된다.
- Session을 사용하지 않으려면 **<%@page session="false" %>**를 지정하면 된다. 그러나 Session을 끝내는 기능까지 하는 것이 아니므로 **반드시** invalidate()를 호출해야 한다.

URL 재작성 메커니즘의 사용 방법

- **Cookie**를 사용할 수 없는 웹 환경에서 사용. (예를 들어 정책에 따른 차단)
- **"URL;jsessionid=SESSION_ID"** 형태를 이용하는 데, SESSION_ID를 얻기 위해서는 HttpSession의 getID()를 이용할 수 있지만 더 쉽게 doGet, doPost 메서드의 두 번째 Parameter **"response"**를 이용한다.
- "response.encodeURL()"은 지능적으로 작동.
  - Parameter로 넘겨준 URL이 현재 웹 애플리케이션 디렉토리에 속하지 않을 경우, URL 재작성하지 않고 본래 URL 반환
  - 웹 브라우저로부터 쿠키가 하나라도 있으면 쿠키를 사용할 수 있는 웹 환경이라고 판단하여 본래 URL 반환
- 상대적인 URL 경로 사용 가능. ("myHome/main.jsp")
- 웹 서버 내 URL 경로 사용 가능. ("/root/myHome/main.jsp")

```
<%
  /* 반환 결과는 URL;jsessionid=SESSION_ID */
    String url1 =
response.encodeURL("http://localhost:8080/myHome/main.jsp");
    String url2 =
response.encodeRedirectURL("http://localhost:8080/myHome/main.jsp");
//sendRedirect()의 URL 재작성 기법. Redirect시 세션 유지
%>
```

Exception 처리

일반적인 Java 프로그램의 try~catch를 이용하게 되면 HTML 코드와 섞여 난잡해진다. 이를 해결하기 위해 Exception 처리를 위한 웹 컴포넌트를 별도로 만들어 호출하면 된다.

여러 페이지 만들어서 호출

- try~catch + forward() ☞ Servlet에서는 page 지시자를 이용할 수 없기 때문에 이를 이용.
- page 지시자의 **errorPage/isErrorPage** Attribute **(☞ try~catch를 쓰지 않고 간결하게 처리 가능! (JSP만 가능))**

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

```
<%@page contentType-"text/html; charset=euc-kr" isErrorPage="true" %>
<%@ response.setStatus(200); %> //200이면 정상적인 실행 결과, 500이면 에러 발생 결과. 항상 의도한대로 작동시키기 위해 200 설정.
<html>
<body>
    <%= exception.getMessage() %> //isErrorPage 지시자를 이용하면 세부적인 Exception 메시지를 확인할 수 있다.
</body>
</html>
```

web.xml 파일에 여러 페이지 등록하기

웹 컴포넌트의 규모가 커질 경우 여러 페이지의 공통 Exception을 처리할 수 있으므로 효율적.

- page 지시자의 errorPage 사용하지 않아야 한다.

Exception Type별

- "HTTP 상태 코드별" 처리보다 높은 **우선권**을 갖는다.

```
<web-app ... >
    ...
    <error-page>
        <exception-type>java.lang.NumberFormatException</exception-type> <!- Type -->
        <location>/Error.jsp</location> <!-- Exception 처리 페이지 경로명 -->
    </error-page>
    ...
</web-app>
```

```
//□ doGet(), doPost() throws 선언
//□ throws 할 수 있는 Exception이 한정되어 있기 때문에 SevletException으로 포장한다.
```

```
/* 던질 수 있는 Exception 종류
 * java.io.IOException, java.servlet.ServletException,
java.lang.ArithmeticException, java.lang.NumberFormatException
 */
try {~}
catch (java.sql.SQLExcption e) {
    throw new ServletException(e);
}
```

HTTP 상태 코드별

- 웹 컴포넌트의 실행과 상관없이, 웹 브라우저로부터 받은 **URL**에 해당하는 웹 자원이 **웹 서버에 없는 경우** 사용.

```
<web-app ... >
    ...
    <error-page>
        <error-code>404</error-code> <!-- HTTP 상태 코드. 404는 웹 자원이 없을 때, 500
은 웹 컴포넌트 안의 Exception 발생 상태 코드 -->
        <location>/Error.jsp</location> <!-- Exception 처리 페이지 경로명 -->
    </error-page>
    ...
</web-app>
```

Servlet의 Life Cycle

똑같은 데이터, 로직, 결과를 갖는 코드를 한번만 실행하기

Servlet 클래스의 init(), destory()

- Init()

Finabocci 수열 출력

```
import java.io.IOException;
import java.io.PrintWriter;
import java.math.BigInteger;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class Finabocci extends HttpServlet {
    private BigInteger[] arr;
    @Override
    public void init() throws ServletException {
        arr = new BigInteger[100];
        arr[0] = new BigInteger("1");
        arr[1] = new BigInteger("1");
        for (int i = 2; i < arr.length ; i++) {
            arr[i] = arr[i - 2].add(arr[i - 1]);
        }
    }
    @Override
    /*public void init(ServletConfig config) throws ServletException {
   // 원리적으로
        // Web container에서 Servlet을 초기화할 때 이 Method가 불리어진다.
   // 그런데 내부적으로 Parameter가 없는 init이 불러오기 때문에
        // Parameter가 없는 init으로 초기화가 가능하다.
   // 주의점으로 이 메소드를 이용할 때는 반드시 부모 Class의 init을 호출해야 한다.
   // 혹시나 부모 클래스를 재정의한 경우가 있을 수 있으므로...
        super.init(config);
    }*/
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String str = request.getParameter("NUM");
        int num = Integer.parseInt(str);
        if(num > 100)
            num = 100;
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Fibonacci</title></head");
        out.println("<body>");
        for(int i = 0; i < num ; i++) {
            out.println(arr[i] + " ");
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
  <servlet>
    <servlet-name>finabocciServlet</servlet-name>
    <servlet-class>Finabocci</servlet-class>

  <!-- 매번 Servlet을 실행할 때마다 대기시간이 길어질 수 있다. 해결방법은?? -->
    <!-- Web Container가 시작될 때 초기화한다. -->
  <!-- 내부 정수값은 우선순위. 작을수록 높음. -->
```

```
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>finabocciServlet</servlet-name>
    <url-pattern>/test</url-pattern>
  </servlet-mapping>
</web-app>
```

- destroy()

Log Writer

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Log Writer</title>
</head>
<body>
    <form action="logwriter">
        <input type="text" name="MESSAGE">
        <input type="submit" value="확인">
    </form>
</body>
</html>
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.GregorianCalendar;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LogWriter extends HttpServlet {

    private PrintWriter logFile;
    @Override
    public void init() throws ServletException {
        /* web.xml에 기재된 초기화 Parameter 이용*/
        String fileName = getInitParameter("FILE_NAME");
        try {
            logFile = new PrintWriter(new FileWriter(fileName, true));
        } catch (IOException e) {
            throw new ServletException(e);
        }
```

```
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String message = request.getParameter("MESSAGE");
        if(message == null)
            throw new NullPointerException("Message가 NULL이다!");
        if(logFile != null) {
            GregorianCalendar now = new GregorianCalendar();
            logFile.printf("%TF %TT - %s %n", now, now, message);
        }
        response.setContentType("text/html ; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>test</title></head>");
        out.println("<body>");
        out.println(message + "이 기록되었습니다.");
        out.println("</body>");
        out.println("</html>");
    }
    @Override
    public void destroy() {
        if(logFile != null)
            logFile.close();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
  <servlet>
    <servlet-name>logWriter-servlet</servlet-name>
    <servlet-class>LogWriter</servlet-class>
  <!-- 소스 코드를 건들지 않고 Path 변경 -->
  <!-- 초기화 Parameter -->
    <init-param>
        <param-name>FILE_NAME</param-name>
        <param-value>C:\\log.txt</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>logWriter-servlet</servlet-name>
    <url-pattern>/logwriter</url-pattern>
  </servlet-mapping>
</web-app>
```

JSP 페이지의 jspInit(), jspDestroy()

※ Servlet과의 차이점으로 선언부(<%! %>)에서는 **throws** 절을 사용할 수 없다.

logWriter

```
<%@page import="java.util.GregorianCalendar"%>
<%@page import="java.io.*"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%!
        private PrintWriter logFile;
        public void jspInit() {
            /* web.xml에 기재된 초기화 Parameter 이용*/
            String fileName = getInitParameter("FILE_NAME");
            try {
                logFile = new PrintWriter(new FileWriter(fileName, true));
            } catch (IOException e) {
                // throw 절 사용 불가!
                System.err.printf("%TT - %s 파일을 열 수 없습니다. %n", new
GregorianCalendar(), fileName);
            }
        }
    %>

<!DOCTYPE html>
<html>
<head>
<title>logWriter</title>
</head>
<body>
    <%
        GregorianCalendar now = new GregorianCalendar();
        String date = String.format("현재 날짜: %TY년 %Tm월 %Te일", now, now,
now);
        String time= String.format("현재 시각: %TI시 %TM분 %TS초", now, now,
now);
        if(logFile != null) {
            logFile.println(date + " " + time);
            out.println(date + "<br>");
            out.println(time + "<br>");
        }
        else
            out.print("초기화 실패");
    %>
</body>
</html>

<%!
    public void jspDestroy() {
```

```
        if(logFile != null)
            logFile.close();
    }
%>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
  <servlet>
      <servlet-name>logWriter-jsp</servlet-name>
      <!-- Servlet 설정과 비교했을 때 아래 부분만 다름 -->
      <jsp-file>/logWriter.jsp</jsp-file>
  <!-- 소스 코드를 건들지 않고 Path 변경 -->
  <!-- 초기화 Parameter -->
    <init-param>
        <param-name>FILE_NAME</param-name>
        <param-value>C:\\log2.txt</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>logWriter-jsp</servlet-name>
      <url-pattern>/logwriter</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet의 환경을 표현하는 ServletContext

Servlet의 환경 정보를 가져오는 방법

- Servlet

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServerInfoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
```

```
response) throws ServletException, IOException {
        ServletContext context = getServletContext();
        String serverInfo = context.getServerInfo();
        int majorVersion = context.getMajorVersion();
        int minorVersion = context.getMinorVersion();
        response.setContentType("text/html ; charset=euc-kr");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>ServerInfo</title></head>");
        out.println("<body>");
        /* tomcat 버전과 Servlet 버전 출력*/
        out.println(serverInfo + " // " + majorVersion + "." +
minorVersion);
        out.println("</body>");
        out.println("</html>");
    }
}
```

- JSP
  - **applicaion** 내장 변수를 이용하여 더 간단히 작성 가능!

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
    <%= application.getServerInfo() %> //
    <%= application.getMajorVersion() %> . <%= application.getMinorVersion()
%>
</body>
</html>
```

초기화 Parameter 가져오는 getInitParameter()

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
  <!-- 결과적으로 Hello, world! 출력 -->
```

```
    <%= application.getInitParameter("PARAM_TEST") %>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
    <!-- Web application 전체 속하는 초기화 Parameter -->
    <context-param>
        <param-name>PARAM_TEST</param-name>
        <param-value>Hello, world!</param-value>
    </context-param>
</web-app>
```

Log 기록하는 Log()

- 실행되는 동안 [Tomcat]/logs 내에 **localhost.yyyy-mm-dd.log**에 기록됨.

같은 Web application에 속하는 Web component끼리 Data를 주고 받는 방법

application.setAttribute(String arg0, Object arg1)
application.getAttribute(String arg0)
application.removeAttribute(String arg0)

setAttribute, getAttribute, removeAttribute 4Set 구분

- 사용 범위가 좁은 Attribute 부터 순서 시작.

호출할 때 사용하는 내장 변수	Method 소속	Data 전송 범위	용도	다루는 장
pageContext	javax.servlet.jsp.JspContext 클래스	현재 JSP 페이지 내	같은 JSP 페이지 안에 있는 Scripting 요소와 Expression Language 간의 Data 교환	7장
request	javax.servlet.ServletRequest 인터페이스	현재 웹 컴포넌트 및 forward(), include()를 통해 호출되는 또 다른 Web component	forward(), include()를 통해 호출되는 Web component로의 Data 전달	3장
session	javax.servlet.http.HttpSession	같은 세션에 속하는 모든 Web component	여러 화면으로 구성되는 Application에서 Web component간의 Data 전달	4장
application	javax.servlet.ServletContext 인터페이스	같은 Web application에 속하는 모든 Web component	같은 Web application의 Directory에 속하는 Web component간의 Data 공유	6장

Expression Language

- **간략한 표현**으로 식을 계산해서 그 **데이터 처리 결과를 출력**하는 목적으로 사용.
  - Logic 구현이 목적이 아니다! 오직 **출력**!

${EXPRESSION}	EXPRESSION은 기본적으로 **"Attribute 이름"**이다. 이외 산술연산자, 논리연산자 사용 및 정적 Method를 호출할 수 있다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%
    int sum=0;
    for(int i=1 ; i<=10 ; i++)
        sum += i;

    request.setAttribute("SUM", sum);
%>

<!DOCTYPE html>
<html>
<head>
<title>EL</title>
</head>
<body>
    <!-- <%= request.getAttribute("SUM") %> 동일.-->
    ${SUM} <br><br>
  <!-- 산술연산자 및 논리연산자 사용이 가능 -->
    ${SUM * 2} <br>
    ${SUM == 55}
  <!-- 정적 Method 호출은 하단 참조 -->
</body>
</html>
```

Expression 언어의 기초 문법

Attribute 이름 하나로만 구성된 EL 식

```
${ATTRIBUTE_NAME}
```

- 만약 Attribute 이름이 같고, Method가 소속된 Package이 다른 경우???
  - 사용 범위가 좁은 Attribute부터 해석된다!
    - page → request → session → application
  - 특정 소속의 Attribute를 선택하고 싶다면??
    - **pageScope**, **requestScope**, **sessionScope**, **applicationScope** 내장 객체를 이용한다.

```
${pageScope.SUM}
${requestScope.SUM}
${sessionScope.SUM}
${applicationScope.SUM}
```

Expression Language의 내장 객체

내장 객체 이름	표현하는 데이터	Type
pageScope	page Attribute 집합	Map
requestScope	request Attribute 집합	Map
sessionScope	session Attribute 집합	Map
applicationScope	application Attribute 집합	Map
param	Web browser로 입력된(<form> element를 통해 입력된) Data 집합 (paramValues는 같은 이름의 Data가 여러개일 때 사용. (checkbox or select))	Map
paramValues		
header	HTTP 요청 Message에 있는 HTTP Header의 집합 (headerValues는 같은 이름의 HTTP Header가 여러개일 때 사용)	Map
headerValues		
Cookie	Web browser로부터 전송된 Cookie 집합	Map
initParam	Web application의 초기화 Parameter 집합	Map
pageContext	JSP Page의 환경 정보 집합 (javax.servlet.jsp.PageContext)	PageContext

pageContext를 제외하고 모두 **Map** Type이기 때문에

1. 단수 Data인 경우
    1. 내장객체.ATTRIBUTE_NAME
    2. 내장객체.[“ATTRIBUTE_NAME”]
2. 복수 Data인 경우
    1. 내장객체.ATTRIBUTE_NAME[INDEX]
    2. 내장객체.[“ATTRIBUTE_NAME”][INDEX]

두 가지 방법으로 사용 가능하다. (b 방법에서 큰 따옴표(“) 대신 작은 따옴표(')로 사용 가능하다.)
단, ATTRIBUTE_NAME이 Java 식별자 명명 규칙을 따르지 않는 경우(첫 글자가 숫자이거나 영문자, 숫자, $/_ 이외 특수문자가 포함된 경우) b 방법을 사용해야 한다.

예제

- param, paramValues

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>input</title>
</head>
<body>
```

```
    <form action="result.jsp">
   성 <input type="text" name="lastName"> <br>
   이름 <input type="text" name="firstName"> <br><br>
   좋아하는 음식 <br>
        <input type="checkbox" name="food" value="초콜릿"> 초콜릿
        <input type="checkbox" name="food" value="국수"> 국수
        <input type="checkbox" name="food" value="탕수육"> 탕수육
        <input type="checkbox" name="food" value="스테이크"> 스테이크 <br><br>
   성별
        <select name="sex">
            <option>남</option>
            <option>여</option>
        </select>
        <br><br>
        <input type="reset" value="취소">
        <input type="submit" value="확인">
    </form>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>result</title>
</head>
<body>
  <!-- 하나일 경우 아래 두 가지 방식으로 표현 가능. (" 대신 ' 사용 가능.) -->
  성 : ${param.lastName} <br>
  이름 : ${param["firstName"]} <br>
  <!-- 여러개일 경우 아래 두 가지 방식으로 표현 가능. (" 대신 ' 사용 가능.) -->
  좋아하는 음식 : ${paramValues.food[0]}
                  ${paramValues.food[1]}
                  ${paramValues.food[2]}
                  ${paramValues.food[3]}
    <br>
  성별 : ${paramValues["sex"][0]}
          ${paramValues["sex"][1]}
</body>
</html>
```

- cookie

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
```

```
    Cookie c = new Cookie("job", "Developer");
    response.addCookie(c);
%>

<!DOCTYPE html>
<html>
<head>
<title>result</title>
</head>
<body>
    value : ${cookie.job.value} <br>
    domain : ${cookie.job.domain} <br>
    path : ${cookie.job.path} <br>
    maxAge : ${cookie.job.maxAge} <br>
</body>
</html>
```

- header, headerValues

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
    <!-- Firefox일 경우 -->
    <!-- Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like
Gecko -->
    ${header["User-Agent"]}<br>
    <!-- application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*  -->
    ${headerValues.Accept[0]}
</body>
</html>
```

- initParam

초기화 Parameter 가져오는 getInitParameter() 그대로 인용.

```
...
<!-- 결과적으로 Hello, world! 출력 -->
${initParam.PARAM_TEST}
...
```

- pageContext
    - API에서 **Parameter**가 없고, **get-**으로 시작하는 8개의 Method 중에서

get을 제거하고 첫자를 소문자로 변경하여 사용.

- errorData
- exception
- page
- request
- response
- servletConfig
- servletContext
- session

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    HttpSession s = request.getSession();
    session.setAttribute("mySession", "Hello, world!");
%>

<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
    <!-- mySession -->
    ${pageContext.session.valueNames[0]}<br>

    <!-- /Example/result.jsp -->
    ${pageContext.request.requestURI}
</body>
</html>
```

Expression Language의 연산자

산술, 비교, 논리, 조건 연산자

- IDE나 Editor에서 "<", ">", "%" 기호 판단을 제대로 못할 경우 사용

div	/
mod	%
lt (less than)	<
gt (greater than)	>
le (less or equal to)	<=
ge (greater or equal to)	>=

eq (equal to)	==
ne (not equal to)	!=

- 문자열에 사용할 경우 유니코드에 따른 사전식 비교가 된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
    <!-- true -->
    ${"apple" lt "banana"}
</body>
</html>
```

Empty 연산자

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>ServerInfo</title>
</head>
<body>
    <!-- Data의 존재 여부 확인 -->
    ${empty param.ID ? "guest" : param.ID}
</body>
</html>
```

[], . 연산자

아래 4가지 항목을 가리킨다.

- 배열의 Data
- java.util.List의 Data
- java.util.Map의 Data
- JavaBean Property[2)]
    - 컴파일하여 [Web application]/WEB-INF/classes/[Package] 에 넣어야 한다.
    - **주의점!! JavaBean 작성시 default package는 비허용되므로 참조가 불가능하다!**

```
<%@page import="java.util.HashMap"%>
<%@page import="java.util.ArrayList"%>
<%@page import="bean.PersonInfo"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    int[] arr = {11, 22, 33};
    request.setAttribute("ARR", arr);
    ArrayList<String> list = new ArrayList<>();
    list.add("a");
    list.add("b");
    list.add("c");
    request.setAttribute("LIST", list);
    HashMap<String, Integer> map = new HashMap<>();
    map.put("Jake", 21);
    map.put("Chris", 55);
    request.setAttribute("MAP", map);
    PersonInfo personInfo = new PersonInfo();
    personInfo.setName("Luke");
    personInfo.setAge(28);
    request.setAttribute("BEAN", personInfo);
%>

<!DOCTYPE html>
<html>
<head>
<title>test</title>
</head>
<body>
  <!-- 배열의 Data 출력 -->
    ${ARR[0]} ${ARR[1]} ${ARR[2]}<br>
    <!-- List의 Data 출력 -->
    ${LIST[0]} ${LIST[1]} ${LIST[2]}<br>
    <!-- Map의 Data 출력 (두 가지 방식 사용 가능) -->
    ${MAP["Jake"]} ${MAP.Chris}<br>
    <!-- JavaBeans Property의 Data 출력 (두 가지 방식 사용 가능) -->
  <!-- 원론적으로는 해당 class를 컴파일하여 [Web application]/WEB-
INF/classes/[Package] 에 넣어야 한다. -->
    <!-- default package는 불가!!! -->
  <!-- 그러나 Eclipse를 사용할 경우 Java Resources/src에 class 생성하면 자동으로 컴파일이
되어 바로 참조 가능. (이미 설정되어 있음.) -->
    ${BEAN["name"]} ${BEAN.age}<br>
</body>
</html>
```

```
package bean; //JSP에서는 default package는 참조 불가
import java.io.Serializable;
```

```
public class PersonInfo implements Serializable {
    private String name;
    private int age;

    public PersonInfo() {
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Expression Language로 정적 Method 호출

1. tld(tag library discriptor) 파일 생성 및 function 기술
2. web.xml에 tld 파일 등록
3. 호출하려는 jsp 파일에서 **taglib** 지시자 기술

```
package util; // default package는 불가!

import java.text.SimpleDateFormat;
import java.util.Date;

public class MyMath {
    public static int sum(int start, int end) {
        int sum = 0;
        for(int i=start ; i<=end ; i++) {
            sum+=i;
        }
        return sum;
    }
    public static String now() {
        SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd hh:mm:ss");
        return sdf.format(new Date()).toString();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
 <!-- 아래 두 element는 반드시 작성! -->
  <tlib-version>tlib-version</tlib-version>
  <short-name>short-name</short-name>
  <function>
      <name>sqrt</name>
      <function-class>java.lang.Math</function-class>
      <function-signature>double sqrt(double)</function-signature>
  </function>
  <function>
      <name>parseInt</name>
      <function-class>java.lang.Integer</function-class>
      <!-- Full package 명을 기입해야 한다! -->
      <function-signature>int parseInt(java.lang.String)</function-
signature>
  </function>
 <!-- 사용자 정의 함수 -->
  <function>
      <name>total</name>
      <function-class>util.MyMath</function-class>
      <function-signature>int sum(int, int)</function-signature>
  </function>
  <function>
      <name>now</name>
      <function-class>util.MyMath</function-class>
      <function-signature>java.util.String now()</function-signature>
  </function>
</taglib>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib uri="../../math-func.tld" prefix="m" %>
<!DOCTYPE html>
<html>
<head>
<title>test</title>
</head>
<body>
  제곱근 : ${m:sqrt(25)} <br>
  문자→숫자 : ${m:parseInt("123")} <br>
  사용자 정의 합 : ${m:total(1, 10)} <br>
  사용자 정의 현재 시간 : ${m:now()} <br>
</body>
</html>
```

Standard action

JSP 페이지의 모듈화에 사용되는 Standard action

<jsp:include>

- include() Method와 결과가 동일하다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
    <h1>오늘의 메뉴</h1>
    <ul>
        <li>짜장면</li>
        <li>짬뽕</li>
        <li>볶음밥</li>
    </ul>
    <br>
  <!-- 별도의 class 파일을 생성하지 않음. 현재 JSP 파일의 Servlet class에서 코드가 포함됨. -
->
    <%-- <%@include file="today.jsp" %> --%>
  <!-- 별도의 class 파일을 생성함. -->
    <%-- <%
        out.flush();
        RequestDispatcher dispatcher =
request.getRequestDispatcher("today.jsp");
        dispatcher.include(request, response);
    %> --%>
  <!-- 별도의 class 파일을 생성함. -->
  <!-- 즉, include() 메소드와 동일. -->
    <jsp:include page="today.jsp" />
</body>
</html>
```

```
<%@page import="java.util.GregorianCalendar"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%
    GregorianCalendar now = new GregorianCalendar();
```

```
%>
<%= String.format("%TF %TT", now, now) %>
```

<jsp:forward>

- forward()와 Method와 결과가 동일하다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
    <%
        int sum = 0;
        for(int i=1 ; i<=100 ; i++)
            sum+=i;
        request.setAttribute("RESULT", sum);
    %>
    <jsp:forward page="result.jsp" />
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>result</title>
</head>
<body>
  합 : ${RESULT}
</body>
</html>
```

JavaBean 호출에 사용되는 Standard action

기초 사용 방법

JavaBean은 [], . 연산자 사용.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
    <!-- id : 변수이름 -->
    <!-- JavaBean 객체의 기본 생존 범위는 JSP "page" 안 -->
    <!-- forward를 이용해 Data를 공유하려면 최소 "request" 설정 -->
    <!-- page < request < session < application -->
    <jsp:useBean id="info" class="bean.PersonInfo" scope="request" />
    <jsp:setProperty property="name" value="Luke" name="info" />
    <jsp:setProperty property="age" value="28" name="info" />
  <!-- 위와 같이 연달아 action이 나오는 경우 아래 방법으로도 사용 가능. -->
    <%-- <jsp:useBean id="info" class="bean.PersonInfo" scope="request">
        <jsp:setProperty property="name" value="Luke" name="info" />
        <jsp:setProperty property="age" value="28" name="info" />
    </jsp:useBean> --%>
    <jsp:forward page="result.jsp" />
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>result</title>
</head>
<body>
    <jsp:useBean id="info" class="bean.PersonInfo" scope="request" />
  이름 : <jsp:getProperty property="name" name="info"/> <br>
  나이 : <jsp:getProperty property="age" name="info"/>
</body>
</html>
```

Web browser로부터 입력된 Data를 JavaBean Property로 설정하는 방법

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>input</title>
</head>
<body>
    <form action="result.jsp">
   이름 : <input type="text" name="name"> <br>
   나이 : <input type="text" name="age"> <br>
        <input type="submit" value="확인">
    </form>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>result</title>
</head>
<body>
    <jsp:useBean id="info" class="bean.PersonInfo" />

    <!-- value="${PARAM.NAME}"과 param="NAME" 결과 동일. -->
  <!-- 당연한 얘기지만 value와 param 혼용 불가. -->
    <%-- <jsp:setProperty property="name" name="info" param="name" />
    <jsp:setProperty property="age" name="info" param="age" /> --%>

    <!-- Property 이름과 입력 데이터의 이름이 동일(대소문자 구분)하다면 아래 방식으로 사용 가
능. -->
    <jsp:setProperty property="*" name="info" />

  이름 : <jsp:getProperty property="name" name="info"/> <br>
  나이 : <jsp:getProperty property="age" name="info"/>
</body>
</html>
```

JavaBean의 다형성을 활용하는 방법

- 상속받은 요소가 Interface 혹은 Abstract class인 경우 **type** attribute를 사용한다!

```
package bean; //JSP에서는 default package는 참조 불가
import java.io.Serializable;

abstract public class PersonInfo implements Serializable {
    private String name;
```

```
    private int age;

    public PersonInfo() {
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

```
package bean;

public class EmployeeInfo extends PersonInfo {
    private long departmentNo;

    public long getDepartmentNo() {
        return departmentNo;
    }

    public void setDepartmentNo(long departmentNo) {
        this.departmentNo = departmentNo;
    }
}
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
    <jsp:useBean id="info" class="bean.EmployeeInfo" scope="request">
        <jsp:setProperty property="name" name="info" value="Luke" />
        <jsp:setProperty property="age" name="info" value="28" />
```

```
    </jsp:useBean>
    <jsp:forward page="result.jsp" />
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>result</title>
</head>
<body>
  <!-- 만약, 상속받은 요소가 추상 클래스거나 인터페이스인 경우 "type" attribute를 사용한다! -->
    <jsp:useBean id="info" type="bean.PersonInfo" scope="request" />
  이름 : <jsp:getProperty property="name" name="info"/> <br>
  나이 : <jsp:getProperty property="age" name="info"/>
</body>
</html>
```

그 밖에 유용한 Standard action

Script 요소를 대신하는 Standard action

문법의 3종류 대신 사용 가능.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
    <%-- <% %> --%>
    <jsp:scriptlet>
        int num1 = 1;
        int num2 = 2;
    </jsp:scriptlet>
    <%-- <%= %> --%>
    <jsp:expression>add(num1, num2)</jsp:expression>
    <%-- <%! %> --%>
    <jsp:declaration>
```

```
        private int add(int a, int b) {
            return a + b;
        }
    </jsp:declaration>
    <br>
    <%-- <%@ %> --%>
    <jsp:directive.include file="today.jsp" />
</body>
</html>
```

today.jsp

JSTL

JSP **S**tandard **T**ag **L**ibrary

- API docs
  - http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/
  - http://docs.oracle.com/javaee/5/tutorial/doc/bnake.html
- Download
  - http://tomcat.apache.org/taglibs/standard/
    - 1.1 : http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/
  - WEB-INF/lib 에 넣는다. (이클립스는 프로젝트 내, 아니면 tomcat 내)
- 무엇을 할 수 있는가?
  - 간단한 프로그램 Logic 구사 (변수 선언, if/for 등의 간단한 Logic)
  - 다른 JSP Page 호출
  - 날짜, 시간, 숫자 Format
  - JSP Page 하나를 가지고 여러 가지 언어의 Web page 생성
  - Database의 입력, 수정, 삭제, 조회
  - XML 문서의 처리
  - 문자열 처리 함수 호출

Library	기능	URI 식별자	접두어
core	변수 선언, 실행 흐름 제어, 다른 JSP page 제어	http://java.sun.com/jsp/jstl/core	c
formatting	숫자, 날짜, 시간 formatting 및 국제화, 다국어 지원	http://java.sun.com/jsp/jstl/fmt	fmt
database	입력/수정/삭제/조회	http://java.sun.com/jsp/jstl/sql	sql
xml	xml 처리	http://java.sun.com/jsp/jstl/xml	x
functions	문자열 처리 함수 제공	http://java.sun.com/jsp/jstl/functions	fn

Core library

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

<c:set>

변수 선언 및 초기화.

- 초기화는 선택이 아닌 **필수**
- 전달되는 Data는 **Attribute**이므로 Expression 사용 불가.

```
<!-- 전달되는 Data의 Attribute 이름이 "num"일 때, -->

<!-- 사용가능 -->
${num}
<%= request.getAttribute("num") %>

<!-- 사용불가 -->
<%= num %>
<%
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="7" scope="request" />
<c:set var="num2" value="9" scope="request" />
<c:set var="result" value="${num1*num2}" scope="request" />
<jsp:forward page="result.jsp" />
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    ${num1} * ${num2} = ${result}
    <!-- attribute 형태이므로 Expression 사용 불가 -->
</body>
</html>
```

<c:remove>

특정 Attribute 삭제.

```

```

```
<!-- scope를 지정하지 않으면 -->
<!-- "num"이라는 이름의 Attribute를 -->
<!-- 모든 영역(page, request, session, application)에서 삭제. -->
<c:remove var="num" scope="request" />
```

<c:if>

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="1" scope="request" />
<c:set var="num2" value="9" scope="request" />
<jsp:forward page="result.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>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- "test" attribute 빼먹지 말자! -->
    <!-- "test" 안에는 EL식! -->

    <c:if test="${num1 >= num2}" var="result">
        num1이 크거나 같다! <br>
        <br>
        ${result}
    </c:if>
    <c:if test="${num1 < num2}" var="result">
        num2가 크다!
        <br>
        ${result}
    </c:if>
</body>
</html>
```

<c:choose>

switch와 같은 역할.

```
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="99" scope="request" />
<jsp:forward page="result.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>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <c:choose>
        <c:when test="${num == 0}">
            <!-- switch의 case -->
    0이다!
        </c:when>
        <c:when test="${num == 1}">
    1이다!
        </c:when>
        <c:otherwise>
            <!-- switch의 default -->
    0도 1도 아니다!
        </c:otherwise>
    </c:choose>
</body>
</html>
```

<c:forEach>

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
        String[] arr = {"돈까스","짜장면","콩국수"};
        request.setAttribute("MENU", arr);
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
```

```
</head>
<body>
  <!-- 1부터 10까지, 2간격식. -->
    <c:forEach var="cnt" begin="1" end="10" step="2">
        <font size=${cnt}>야호</font> <br>
    </c:forEach>
    <!-- Iterator -->
    <ul>
        <c:forEach var="dish" items="${MENU}">
            <li>${dish}</li>
        </c:forEach>
    </ul>
</body>
</html>
```

<c:forTokens>

for + StringTokenizer

```
<%@ 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>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <c:set var="str" value="동해물과 백두산이/마르고^닳도록" />
    <c:forTokens var="lyric" items="${str}" delims=" /^">
        ${lyric} <br>
    </c:forTokens>
</body>
</html>
```

<c:catch>

try 문.

```
<!-- http://localhost:8080/Example2/result.jsp?NUM1=10&NUM2=0 -->

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

```
<%
    int num1 = Integer.parseInt(request.getParameter("NUM1"));
    int num2 = Integer.parseInt(request.getParameter("NUM2"));
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- "catch" block에 해당하는 code는 별도 작성 필요. -->
    <c:catch var="e">
        <% int result = num1 / num2; %>
        <%= result %>
    </c:catch>
    <c:if test="${e != null}">
        ${e.message}
    </c:if>
</body>
</html>
```

<c:redirect>

sendRedirect()와 동일.
그렇기 때문에 JSP Page가 아닌 Web resource와 다른 Web server에 있는 Web resource 호출 가능.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:redirect url="result.jsp">
    <c:param name="num1" value="7" />
    <c:param name="num2" value="11" />
</c:redirect>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- attribute가 아닌 parameter! -->
    ${param.num1} * ${param.num2} = ${param.num1 * param.num2}
```

```
</body>
</html>
```

<c:import>

현재 JSP 페이지에 다른 Web resource를 포함시킴.

- <jsp:include>처럼 같은 Web server에 있는 JSP Page 포함 가능.
- 다른 Web server에 있는 JSP Page 포함 가능.
- JSP Page가 아닌 다른 종류의 Web resource 포함 가능.

사용법은 <c:redirect>와 비슷. Data를 넘겨주는 형태도 동일.

<c:url>

구조적인 url을 위한 변수 선언.

```
<%@ 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>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- /Example2/result.jsp?NUM1=999&NUM2=111   -->
    <c:url var="myUrl" value="/result.jsp">
        <c:param name="NUM1" value="999" />
        <c:param name="NUM2" value="111" />
    </c:url>

    ${myUrl}
</body>
</html>
</html>
```

<c:out>

Data 출력. 특이점으로 Tag로 해석될 가능성이 있는 특수문자(<, >, &, ', ")를 **Escape Sequence로 바꿔주는 기능**이 있어 기본적으로 그대로 출력됨.

```
<%@ 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>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- Tag를 해석하지 않고 그대로 출력 -->
    <c:out value="<h1>Test</h1>" /> <br>
    <!-- Tag를 해석한다. -->
    <c:out value="<h1>Test</h1>" escapeXml="false" />
  <!-- 출력 Data가 없는 경우, default 출력 -->
    <c:out value="${param.NAME}" default="Hello!" />
</body>
</html>
```

Formatting library

<fmt:formatDate>

날짜와 시간 format.
주의사항으로 **Date 객체**를 EL식으로 넘겨줘야 한다!

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<c:set var="date" value="<%= new Date() %>" />
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <!-- 2016. 6. 29 -->
    <fmt:formatDate value="${date}" /> <br>
  <!-- 오후 11:49:09 -->
    <fmt:formatDate value="${date}" type="time" /> <br>
  <!-- 2016. 6. 29 오후 11:49:09 -->
    <fmt:formatDate value="${date}" type="both" />
    <br><br>
  <!-- 16. 6. 30 오전 12:00 -->
```

```
    <fmt:formatDate value="${date}" type="both" dateStyle="short"
timeStyle="short" /> <br>
  <!-- 2016. 6. 30 오전 12:00:28 -->
    <fmt:formatDate value="${date}" type="both" dateStyle="medium"
timeStyle="medium" /> <br>
  <!-- 2016년 6월 30일 (목) 오전 12시 00분 28초 -->
    <fmt:formatDate value="${date}" type="both" dateStyle="long"
timeStyle="long" /> <br>
  <!-- 2016년 6월 29일 수요일 오후 3:00:28 -->
    <fmt:formatDate value="${date}" type="both" dateStyle="full"
timeZone="full" /> <br>
    <br><br>
  <!-- 2016/06/30 (목) -->
    <fmt:formatDate value="${date}" type="date" pattern="yyyy/MM/dd (E)" />
<br>
  <!-- (오전) 12:09:23 -->
    <fmt:formatDate value="${date}" type="time" pattern="(a) hh:mm:ss" />
<br>
</body>
</html>
```

<fmt:formatNumber>

수치 format.

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>

    <!-- 1,234,500 -->
    <fmt:formatNumber value="1234500" groupingUsed="true" /> <br>
    <!-- 3.14  -->
    <fmt:formatNumber value="<%= Math.PI %>" pattern="#.##" /> <br>
    <!-- 10.50 -->
    <fmt:formatNumber value="10.5" pattern="#.00#" />
    <br><br>
    <!-- 120% -->
    <fmt:formatNumber value="1.2" type="percent" /> <br>
    <!-- □123,000  -->
    <fmt:formatNumber value="123000" type="currency" /> <br>
```

```
    <!-- $123,000  -->
    <fmt:formatNumber value="123000" type="currency" currencySymbol="$" />
</body>
</html>
```

<fmt:setLocale>

지역 설정.

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>

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

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="date" value="<%=new Date()%>" />

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
    <fmt:setLocale value="ko_kr"/>
    <!--
        ￦1,000,000
   2016년 6월 30일 목요일 오후 12시 28분 45초 KST
    -->
    <fmt:formatNumber value="1000000" type="currency" /> <br>
    <fmt:formatDate value="${date}" type="both" dateStyle="full"
timeStyle="full" />
    <br><br>

    <fmt:setLocale value="en_us"/>
    <!--
        $1,000,000.00
        Thursday, June 30, 2016 12:28:45 PM KST
    -->
    <fmt:formatNumber value="1000000" type="currency" /> <br>
    <fmt:formatDate value="${date}" type="both" dateStyle="full"
timeStyle="full" />
    <br><br>

    <fmt:setLocale value="ja_jp"/>
    <!--
        ￦1,000,000
```

```
       2016年6月30日 12時28分45秒 KST
    -->
    <fmt:formatNumber value="1000000" type="currency" /> <br>
    <fmt:formatDate value="${date}" type="both" dateStyle="full"
timeStyle="full" />
</body>
</html>
```

<fmt:timeZone>, <fmt:setTimeZone>

지역별 시간대 설정.
TimeZone.getAvailableIDs()으로 사용 가능한 **value** Attribute를 얻는다.

- <fmt:timeZone>
  - Tag 안에만 영향을 미침.
- <fmt:setTimeZone>
  - 이 Action 다음의 모든 코드에 영향을 미침.

```
<%@page import="java.util.TimeZone"%>
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>

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

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="date" value="<%=new Date()%>" />

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result</title>
</head>
<body>
  <!-- 2016년 6월 30일 목요일 오전 9:57:48 -->
    <fmt:setTimeZone value="Europe/Berlin"/>
    <fmt:formatDate value="${date}" type="both" dateStyle="full" />
    <br>
  <!-- 2016년 6월 30일 목요일 오전 2:57:48 -->
    <fmt:timeZone value="America/Lima">
        <fmt:formatDate value="${date}" type="both" dateStyle="full" />
    </fmt:timeZone>
</body>
</html>
```

<fmt:setBundle>, <fmt:bundle>

다국어 지원.

- <fmt:setBundle>은 전체 영향, <fmt:bundle>는 Tag 안에만 영향.
- properties 파일은 ASCII Code만 인식되므로, 특히 **한글** 포함일 경우 변환 과정 필요.
    - native2ascii [원본명] [출력파일명]
    - Eclipse 사용시 Properties Editor 사용.
        - **WEB-INF/classes** 에 넣는다.
- Web browser의 언어 설정으로 영향을 받는다.

- <fmt:setBundle>

```
TITLE=About Us
GREETING=Hi, {0}. You have visited this site {1} times!
BODY=We are a dedicated software devlopment company.
COMPANY_NAME=Duke Software Inc.
```

```
<!--
TITLE=회사 소개
GREETING={0}님 안녕하세요. {1}번째 방문이시군요!
BODY=당사는 소프트웨어 개발을 주업무로 하는 회사입니다.
COMPANY_NAME=(주) 듀크 소프트웨어
-->
TITLE=\ud68c\uc0ac \uc18c\uac1c
GREETING={0}\ub2d8 \uc548\ub155\ud558\uc138\uc694. {1}\ubc88\uc9f8
\ubc29\ubb38\uc774\uc2dc\uad70\uc694!
BODY=\ub2f9\uc0ac\ub294 \uc18c\ud504\ud2b8\uc6e8\uc5b4 \uac1c\ubc1c\uc744
\uc8fc\uc5c5\ubb34\ub85c \ud558\ub294 \ud68c\uc0ac\uc785\ub2c8\ub2e4.
COMPANY_NAME=(\uc8fc) \ub4c0\ud06c \uc18c\ud504\ud2b8\uc6e8\uc5b4
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="ID" value="Jake" scope="request" />
<c:set var="COUNT" value="3" scope="request" />
<jsp:forward page="result.jsp" />
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:setBundle basename="intro"/>
<!-- 변수로 저장하여 사용할 수도 있다! -->
```

```
<fmt:message var="title" key="TITLE" />

<!-- Parameter를 받아 처리할 수도 있다. -->
<fmt:message var="greeting" key="GREETING" >
    <fmt:param>${ID}</fmt:param>
    <fmt:param>${COUNT}</fmt:param>
</fmt:message>

<!--
회사 소개
당사는 소프트웨어 개발을 주업무로 하는 회사입니다.
Jake님 안녕하세요. 3번째 방문이시군요!

(주) 듀크 소프트웨어
-->

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>${title}</title>
</head>
<body>
    <h1>${title}</h1>
    <fmt:message key="BODY" /> <br>
    <!-- parameter 받아 출력 -->
    ${greeting} <br>
    <br>
    <fmt:message key="COMPANY_NAME" />
</body>
</html>
```

<fmt:requestEncoding>

POST Method로 전송된 한글 입력 Data를 받기 위해 필요.

Expression Language과 Ac ti on을 이용하면 코드 작업량을 줄일 수 있고, 가독성에서도 유리하므로 Web designer에게도 편리하기 때문에 Scripting Elements를 사용하지 않도록 권장된다.

그런데 이 때 발생하는 문제가 한글 인코딩(<% request.setCharacterEncoding("euc-kr") %>)을 사용해야 하는 경우와 충돌되는 데, 이런 경우에 사용한다.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Log Writer</title>
</head>
```

```
<body>
    <form action="result.jsp" method="POST" >
   한글이름 <input type="text" name="NAME"> <br>
        <input type="submit" value="확인">
    </form>
</body>
</html>
```

```

```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:requestEncoding value="UTF-8" />
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
  한글이름 : ${param.NAME}
</body>
</html>
```

Functions library

주로 문자열 처리. String Class의 Method와 비슷한 기능 제공.

- API

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

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="str" value="<h1>hello, world!</h1>" />

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
```

```
    <!-- <h1> Tag가 적용된 "HELLO, WORLD!" -->
    ${fn:toUpperCase(str)} <br>
    <!-- world! -->
    ${fn:split(str, " ")[1]} <br>
    <!-- <h1>hello, world!</h1> -->
    ${fn:escapeXml(str)}
</body>
</html>
```

Custom action

Tag file과 Tag class의 선택 기준

- 소스 은닉 필요?
- 복잡도?

Tag File을 이용해서 만드는 방법

1. [FILENAME].tag 작성 ([FILENAME]은 Custom action의 Tag 이름)
2. WEB-INF/[SUBDIRECORY] 에 넣는다. (예를 들어 /tags)
3. taglib 지시자 작성.

지시자 이름	역할	비고
include	다른 Tag file 포함	
taglib	다른 Custom action의 Tag library 정보 기술	
tag	Web container가 Tag file을 compile 및 실행하기 위한 정보 기술	tag file에서만 사용 가능.
attribute	Custom action의 attribute 기술	
variable	Custom action의 변수 정보 기술	

본체 내용을 갖지 않는 경우

Attribute를 갖는 경우

동적 Attribute를 갖는 경우

```
<!-- 아래 지시자와 attribute를 지정하면 body(본체)를 가질 수 없도록 제한함. -->
<%@tag body-content="empty" %>
-----------------------------------<br>
```

```
<%@tag body-content="empty" %>
<%@attribute name="color" %>
<!-- 기본적으로 attribute value는 String이므로 변환 필요 -->
```

```
<!-- required="true"는 attribute 기술 강제 요구 -->
<%@attribute name="size" type="java.lang.Integer" required="true" %>
<font color=${color} >
<%
    for(int i=0 ; i<size ; i++)
        out.print("-");
%>
</font><br>
```

```
<%@tag import="java.util.Map"%>
<%@tag body-content="empty" %>
<%@tag dynamic-attributes="attrs" %>
<font color=${attrs.color} >
<%
    Map<String, String> attrs = (Map<String,
String>)jspContext.getAttribute("attrs");
    int size = Integer.parseInt(attrs.get("size"));

    for(int i=0 ; i<size ; i++)
        out.print("-");
%>
</font><br>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="u" tagdir="/WEB-INF/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- ---------------------------------- -->
    <u:line />
    <%-- <u:line>Error 발생!</u:line> --%>
    <!-- attribute value를 갖는 Custom tag -->
  <!-- 적색의 "-" 25개 -->
    <u:line_att color="red" size="25" />
    <!-- dynamic attribute value를 갖는 Custom tag -->
    <!-- attribute가 존재하는 지, 값이 유효한 지 확인하지 않음. 문법적 유연성 제공. -->
    <u:line_datt color="blue" size="100" style="bold" />
</body>
</html>
```

본체 내용을 갖는 경우

- body-content="scriptless" : scripting 요소 금지. (EL이나 Action은 인식됨.)
- body-content="tagdependent" : Text 그대로 취급. (EL이나 Action은 인식되지 않음.)

```
<%@tag body-content="scriptless" %>
<table border="1" cellpadding="20">
    <tr>
        <td>
     <!-- 본체 내용 출력. (tag file에서만 사용할 수 있다!) -->
            <jsp:doBody />
        </td>
    </tr>
</table>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="u" tagdir="/WEB-INF/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- <table> tag가 적용된 text 출력 -->
    <u:box>hello, world!</u:box>
</body>
</html>
```

변수를 지원하는 **Custom action**

```
<%@tag body-content="empty" pageEncoding="UTF-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@attribute name="num1" type="java.lang.Integer" %>
<%@attribute name="num2" type="java.lang.Integer" %>

<!-- rtexprvalue="false" : Attribute value로 Scripting Elements나 EL 사용 금지 -
->
<%@attribute name="var" required="true" rtexprvalue="false" %>

<!--
```

```
    scope="NESTED" : default. 본체 내부에서만 변수 사용 가능.
    scope="AT_BEGIN" : default. 시작 Tag 다음 위치부터 변수 사용 가능.
    scope="NESTED" : default. 끝 Tag 다음 위치부터 변수 사용 가능.
-->
<%-- <%@variable name-given="result" variable-class="java.lang.Integer"
scope="AT_END" %> --%>
<!-- 결과 변수명 유동적으로 사용 -->
<%@variable name-from-attribute="var" alias="result" variable-
class="java.lang.Integer" scope="AT_END" %>

<%
    int sum = num1 + num2;
%>
<c:set var="result" value="<%=sum %>" />
```

```
<%@tag pageEncoding="UTF-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@attribute name="start" type="java.lang.Integer" %>
<%@attribute name="end" type="java.lang.Integer" %>

<!-- rtexprvalue="false" : Attribute value로 Scripting Elements나 EL 사용 금지 -
->
<%@attribute name="var" required="true" rtexprvalue="false" %>

<!--
    scope="NESTED" : default. 본체 내부에서만 변수 사용 가능.
    scope="AT_BEGIN" : default. 시작 Tag 다음 위치부터 변수 사용 가능.
    scope="NESTED" : default. 끝 Tag 다음 위치부터 변수 사용 가능.
-->
<%-- <%@variable name-given="result" variable-class="java.lang.Integer"
scope="AT_END" %> --%>
<!-- 결과 변수명 유동적으로 사용 -->
<%@variable name-from-attribute="var" alias="result" variable-
class="java.lang.Integer" scope="NESTED" %>

<%   for(int i=start ; i<=end ; i++) { %>
<c:set var="result" value="<%= i %>" />
<jsp:doBody/>
<% } %>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="u" tagdir="/WEB-INF/tags" %>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- Custom action의 body(본체) 밖에서 변수를 사용할 경우 -->
  <!-- 어떤 attribute를 넘겨주면 return값을 출력할 경우. -->
    <h1>a + b</h1>

    <!-- name-given을 사용할 경우 -->
    <%-- <u:sum num1="1" num2="10" />
  결과 : ${result} --%>

    <u:sum var="RESULT" num1="1" num2="10" />
  결과 : ${RESULT}
    <!-- Custom action의 body(본체) 안에서 변수를 사용할 경우 -->
    <h1>제곱</h1>
    <u:square var="num" start="5" end="10">
        ${num}의 제곱은? ${num * num} <br>
    </u:square>
</body>
</html>
```

Tag Class를 이용해서 만드는 방법

- JSP 1.2 : IterationTag, BodyTag Interface 구현.
  - Custom action 안에 Script Elements를 사용할 수 있다.
- JSP 2.0 : SimpleTag Interface 구현.
  - Custom action 안에 Script Elements를 사용할 수 없다.
  - 매번 Method 내부를 작성하는 것이 비효율적. 모든 Method를 구현해놓은 Class ⇒ **SimpleTagSupport**

본체 내용을 갖지 않는 경우

```
package tool;
import java.io.IOException;

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

public class StarLineTag extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
```

```
        out.println("**********************<br>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" 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>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
    <jsp-config>
        <taglib>
            <taglib-uri>tools.tld</taglib-uri>
            <taglib-location>/WEB-INF/tlds/tools.tld</taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <tool:starLine />
</body>
</html>
```

Attribute를 갖는 경우

1. setter를 생성한다.
2. tag 파일에 attribute를 기술한다.

```
package tool;

import java.io.IOException;

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

public class NewLineTag extends SimpleTagSupport {
    private int size;
    private String color;

    public void setSize(int size) {
        this.size = size;
    }

    public void setColor(String color) {
        this.color = color;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        out.println("<font color=" + color + ">");
        for(int i=0 ; i<size ; i++)
            out.print("*");
        out.println("</font><br>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>newLineTag</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>
```

web.xml은 Tag Class를 이용해서 만드는 방법#기초와 동일.

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <tool:newLineTag color="red" size="10" />
</body>
</html>
```

동적 Attribute를 갖는 경우

```
package tool;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

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

/* DynamicAttributes 구현 */
public class NewerLineTag extends SimpleTagSupport implements
DynamicAttributes {
    private Map<String, Object> attrs = new HashMap<>();

    @Override
    public void setDynamicAttribute(String uri, String localName, Object
value) throws JspException {
        attrs.put(localName, value);
    }
```

```
    @Override
    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 i=0 ; i<size ; i++)
            out.print("*");
        out.println("</font><br>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>newerLine</name>
        <tag-class>tool.NewerLineTag</tag-class>
        <body-content>empty</body-content>
        <dynamic-attributes>true</dynamic-attributes>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- background, height 는 존재하지 않지만 Exception 발생하지 않음.-->
    <tool:newerLine color="red" size="10" background="black" height="10" />
</body>
</html>
```

본체 내용을 갖는 경우

body-content에 대한 이해는 Custom action#본체 내용을 갖는 경우 참조.

```
package tool;

import java.io.IOException;

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

public class BoxTag extends SimpleTagSupport {
    @Override
    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>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>box</name>
        <tag-class>tool.BoxTag</tag-class>
        <body-content>scriptless</body-content>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- table로 둘러 쌓인 hello, world! 출력 -->
    <tool:box>hello, world!</tool:box>
</body>
</html>
```

본체 내용 조작

문자열 두 개를 attribute로 받아 치환하는 Scenario.

1. 두 개의 setter를 추가한다.
2. StringWriter를 이용하여 출력

```
package tool;

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

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

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;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        JspFragment body = getJspBody();
        /* StringWriter : String 객체를 가상의 하드웨어 장치처럼 출력 */
        StringWriter writer = new StringWriter();
        body.invoke(writer);
        String newStr = writer.toString().replaceAll(oldWord, newWord);
        out.print(newStr);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
```

```
    <tag>
        <name>replace</name>
        <tag-class>tool.ReplaceTag</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>oldWord</name>
            <type>java.lang.String</type>
        </attribute>
        <attribute>
            <name>newWord</name>
            <type>java.lang.String</type>
        </attribute>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
  <!-- 철수가 민기로 치환 -->
    <tool:replace oldWord="철수" newWord="민기">
   철수는 영희에게 인사를 하였습니다.<br>
   하지만 영희는 철수를 보지 못하고 지나쳤습니다.<br>
    </tool:replace>
</body>
</html>
```

변수를 지원하는 Custom action

결과를 출력할 변수 이름이 고정된 경우

```
package tool;

import java.io.IOException;

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

```
public class SumTag extends SimpleTagSupport {
    private int num1, num2;
    public void setNum1(int num1) {
        this.num1 = num1;
    }

    public void setNum2(int num2) {
        this.num2 = num2;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        context.setAttribute("result", num1 + num2);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>sum</name>
        <tag-class>tool.SumTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>num1</name>
            <type>java.lang.Integer</type>
            <!-- RTimeEXPressionVALUE : Expression이나 EL식 포함 가능 여부 -->
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>num2</name>
            <type>java.lang.Integer</type>
            <!-- RTimeEXPressionVALUE : Expression이나 EL식 포함 가능 여부 -->
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <variable>
            <name-given>result</name-given>
            <variable-class>java.lang.Integer</variable-class>
            <scope>AT_END</scope>
        </variable>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- http://localhost:8080/Example2/result.jsp?NUM1=17&NUM2=33 -->
    <tool:sum num1="${param.NUM1}" num2="${param.NUM2}" />
  합 : ${result}
</body>
</html>
```

결과를 출력할 변수 이름을 유동적으로 직접 지정할 경우

```
package tool;

import java.io.IOException;

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

public class SumTag extends SimpleTagSupport {
    private int num1, num2;
    private String var;
    public void setNum1(int num1) {
        this.num1 = num1;
    }

    public void setNum2(int num2) {
        this.num2 = num2;
    }
    public void setVar(String var) {
        this.var = var;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        context.setAttribute(var, num1 + num2);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
```

```
    <short-name>tool</short-name>
    <tag>
        <name>sum</name>
        <tag-class>tool.SumTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>num1</name>
            <type>java.lang.Integer</type>
            <!-- RTimeEXPressionVALUE : Expression이나 EL식 포함 가능 여부 -->
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>num2</name>
            <type>java.lang.Integer</type>
            <!-- RTimeEXPressionVALUE : Expression이나 EL식 포함 가능 여부 -->
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>var</name>
            <type>java.lang.String</type>
     <!-- 필수 기재 -->
            <required>true</required>
            <!-- RTimeEXPressionVALUE : Expression이나 EL식 포함 가능 여부 -->
            <rtexprvalue>false</rtexprvalue>
        </attribute>
        <variable>
            <name-from-attribute>var</name-from-attribute>
            <variable-class>java.lang.Integer</variable-class>
            <scope>AT_END</scope>
        </variable>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
    <!-- http://localhost:8080/Example2/result.jsp?NUM1=17&NUM2=33 -->
    <!-- var를 xxx라는 이름의 Attribute로 지정 -->
    <tool:sum var="xxx" num1="${param.NUM1}" num2="${param.NUM2}" />
  합 : ${xxx}
</body>
</html>
```

Child Custom Action

Custom action 안에 또 다른 Custom action을 포함하는 경우.

기본

```
package tool;

import java.io.IOException;

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

public class ListTag extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspFragment body = getJspBody();
        // null을 넘겨주면 JSP 페이지와 동일한 출력 스트림을 통해 본체의 내용 출력.
   // 현재 의도는 포함된 자식 Custon action의 Tag Class를 알아서 호출
        body.invoke(null);
    }
}
```

```
package tool;

import java.io.IOException;

import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class ItemTag extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspTag parent = getParent();
        if(parent == null || !(parent instanceof ListTag))
            throw new JspException("The Parent is not ListTag");
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
```

```
        JspFragment body = getJspBody();
   /*~[본체 내용]<br> */
        out.print("~ ");
        body.invoke(null);
        out.println("<br>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>list</name>
        <tag-class>tool.ListTag</tag-class>
        <body-content>scriptless</body-content>
    </tag>
    <tag>
        <name>item</name>
        <tag-class>tool.ItemTag</tag-class>
        <body-content>scriptless</body-content>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
  <!-- ~[본체 내용]<br> -->
    <tool:list>
        <tool:item>사과</tool:item>
        <tool:item>배</tool:item>
        <tool:item>딸기</tool:item>
    </tool:list>
</body>
</html>
```

부모와 자식간의 정보를 주고 받는 경우

```
package tool;

import java.io.IOException;

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

public class ListTag extends SimpleTagSupport {
    private char bullet;
    public char getBullet() {
        return bullet;
    }

    public void setBullet(char bullet) {
        this.bullet = bullet;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspFragment body = getJspBody();
        // null을 넘겨주면 JSP 페이지와 동일한 출력 스트림을 통해 본체의 내용 출력.
   // 현재 의도는 포함된 자식 Custon action의 Tag Class를 알아서 호출
        body.invoke(null);
    }
}
```

```
package tool;

import java.io.IOException;

import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class ItemTag extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspTag parent = getParent();
        if(parent == null || !(parent instanceof ListTag))
            throw new JspException("The Parent is not ListTag");
        JspContext context = getJspContext();
        JspWriter out = context.getOut();
        JspFragment body = getJspBody();
        /* [bullet] [본체 내용]<br> */
        char bullet = ((ListTag) parent).getBullet();
        out.print(bullet + " ");
```

```
        body.invoke(null);
        out.println("<br>");
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <tag>
        <name>list</name>
        <tag-class>tool.ListTag</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>bullet</name>
            <type>java.lang.Charater</type>
            <required>true</required>
        </attribute>
    </tag>
    <tag>
        <name>item</name>
        <tag-class>tool.ItemTag</tag-class>
        <body-content>scriptless</body-content>
    </tag>
</taglib>
```

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib prefix="tool" uri="/WEB-INF/tlds/tools.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
  <!-- ★[본체 내용]<br> -->
    <tool:list bullet="⋆">
        <tool:item>사과</tool:item>
        <tool:item>배</tool:item>
        <tool:item>딸기</tool:item>
    </tool:list>
</body>
</html>
```

Tag Library를 만드는 방법

Tag Class 이용

1. Directory 계층 구조 생성
    1. [ROOT]/META-INF[/...] : *class 제외 나머지 파일. (*.tld)
    2. [ROOT]/[PACKAGE] : *.class
2. TLD 파일 수정.
    1. 프로젝트에서 참조할 uri 추가.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <uri>/taglibs/tools.tld</uri>
    <tag>
        ...
    </tag>
</taglib>
```

3. JAR 생성
    1. jar cvf0 [FILENAME].jar *

Tag file 이용

1. Directory 계층 구조 생성
    1. [ROOT]/META-INF/tags[/...] : *class 제외 나머지 파일. (*.tld)
2. TLD 파일 생성.
    1. 프로젝트에서 참조할 uri를 작성하고, tag-file을 기술한다. 주의점으로 path는 최상위 Directory 기준 작성이므로 /로 시작할 것!

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>tool</short-name>
    <uri>/taglibs/util.tld</uri>
    <tag-file>
        <name>line</name>
        <path>/META-INF/tags/line.tag</path>
    </tag-file>
</taglib>
```

    2. [ROOT]/META-INF[/...] 에 저장. (단, tags 제외)
3. JAR 생성
    1. jar cvf0 [FILENAME].jar *

Filter & Wrapper

- Filter : Web component에 대해 **똑같은 사전작업이나 사후작업을 공통적으로 처리** (예. 먼저 로그인 여부 검사)
- Wrapper : Web browser와 Web component간에 오가는 **데이터 변형** (예. 데이터 암호화, 일부 데이터 전달 방지)

Filter Class의 작성, 설치, 등록

1. Filter class를 컴파일하고 Web container에 설치.
    1. javax.servlet.Filter의 Method를 구현한다.
    2. Eclipse를 사용하지 않을 경우 **WEB-INF/classes/[PACKAGE]**

new PoolableConnectionFactory(connFactory, objPool, null, null, false, true);

```
// Web Container에 등록
PoolingDriver driver = new PoolingDriver();
/* XML을 이용해서 생성하는 DBCP은 이름 앞에 자동으로 "/"가 붙기 때문에 일관성을 위해 "/"로 시작하는 것이 좋다!*/
driver.registerPool(DB_POOL_NAME, objPool);
```

%>

<!DOCTYPE html> <html> <head> <meta charset=“UTF-8”> <title>DBCP 생성</title> </head> <body>

```
DBCP 생성 및 등록 성공!<br>
Pool name : <%=DB_POOL_NAME %>
```

</body> </html> </sxh>

연결 및 해제

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" errorPage="DBError.jsp"%>
<%@page import="java.sql.*"%>
<%! private final static String DB_POOL_NAME = "/webdb_pool"; %>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Test DBCP</title>
</head>
<body>
    <%
```

```
        Class.forName("org.apache.commons.dbcp.PoolingDriver");
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn =
DriverManager.getConnection("jdbc:apache:commons:dbcp:" + DB_POOL_NAME);
        if(conn == null) {
            out.println("연결 실패");
            return;
        }
        out.println("연결 취득 완료<br>");
        conn.close();
        out.println("연결 반환 완료<br>");
    %>
</body>
</html>
```

조회

1. 자동 DBCP 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="org.apache.commons.dbcp.*"%>
<%@page import="org.apache.commons.pool.impl.GenericObjectPool"%>
<%!
    private final static String DB_URL =
"jdbc:mysql://localhost:3306/webdb";
    private final static String DB_USER = "root";
    private final static String DB_PASSWORD = "root";
    private final static String DB_POOL_NAME = "/webdb_pool";
    public void jspInit() {
        GenericObjectPool objPool =  new GenericObjectPool();
        DriverManagerConnectionFactory connFactory = new
DriverManagerConnectionFactory(DB_URL, DB_USER, DB_PASSWORD);
        new PoolableConnectionFactory(connFactory, objPool, null, null,
false, true);
        PoolingDriver driver = new PoolingDriver();
        driver.registerPool(DB_POOL_NAME, objPool);
    }
%>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
    <servlet>
```

```
        <servlet-name>auto-create-db-pool-jsp</servlet-name>
        <jsp-file>/AutoCreateDBPool.jsp</jsp-file>
        <load-on-startup>1</load-on-startup>
    </servlet>
</web-app>
```

2. 연결 및 조회

```
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" errorPage="DBError.jsp"%>
<%@page import="java.sql.*"%>
<%@page import="java.io.UnsupportedEncodingException"%>

<%! private final static String DB_POOL_NAME = "/webdb_pool"; %>

<%
    String code = request.getParameter("code");
    Connection conn = null;
    Statement stmt = null;

    try {
        Class.forName("org.apache.commons.dbcp.PoolingDriver");
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection("jdbc:apache:commons:dbcp:" +
DB_POOL_NAME);
        if(conn == null)
            throw new Exception("연결 불가!");
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM goodsinfo WHERE code
= '" + code + "';");
        if(rs.next()) {
            String title = rs.getString("title");
            String writer = rs.getString("writer");
            String price = rs.getString("price");
            request.setAttribute("CODE", code);
            /* request.setAttribute("TITLE", toUnicode(title));
            request.setAttribute("WRITER", toUnicode(writer)); */
            request.setAttribute("TITLE", title);
            request.setAttribute("WRITER", writer);
            request.setAttribute("PRICE", new Integer(price));
        }
    }
    finally {
        try {
            stmt.close();
        }
        catch (Exception e) {
        }
        try {
            conn.close();
```

```
        }
        catch (Exception e) {
        }
    }
    RequestDispatcher dispatcher =
request.getRequestDispatcher("GoodsInfoViewer.jsp");
    dispatcher.forward(request, response);
%>

<%!
    private String toUnicode(String str) {
        try {
            byte[] b = str.getBytes("ISO-8859-1");
            return new String(b);
        } catch (UnsupportedEncodingException e) {
            System.err.println(e.getMessage());
            return null;
        }
    }
%>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>상품 정보</title>
</head>
<body>
    ${CODE} <br>
    ${TITLE} <br>
    ${WRITER} <br>
    ${PRICE}
</body>
</html>
```

JOCL File을 이용한 Database Connection Pool 생성 방법

Java **O**bject **C**onfiguration **L**anguage

DBCP 생성에 필요한 객체 정보를 “[FILENAME].jocl”이란 XML 문서에 기술하여 사용. **이 때, DBCP명은 자동으로 [FILENAME]이 된다!**

1. jocl 작성

```
<object class="org.apache.commons.dbcp.PoolableConnectionFactory"
```

```
xmlns="http://apache.org/xml/xmlns/jakarta/commons/jocl">
    <object class="org.apache.commons.dbcp.DriverManagerConnectionFactory">
        <string value="jdbc:mysql://localhost:3306/webdb" />
        <string value="root" />
        <string value="root" />
    </object>
    <object class="org.apache.commons.pool.impl.GenericObjectPool" />
    <object class="org.apache.commons.pool.KeyedObjectPoolFactory"
null="true" />
    <string null="true" />
    <boolean vlaue="false" />
    <boolean value="true" />
</object>
```

2. Library 설치

JDBC, DBCP, POOL, Collections를 WEB-INF/lib 에 복사.

3. 프로그램 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" errorPage="DBError.jsp"%>
<%@page import="java.sql.*"%>
<%@page import="org.apache.commons.dbcp.PoolingDriver"%>
<%! private final static String DB_POOL_NAME = "/wdbpool"; %>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Test DBCP</title>
</head>
<body>
    <%
        Class.forName("org.apache.commons.dbcp.PoolingDriver");
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn =
DriverManager.getConnection("jdbc:apache:commons:dbcp:" + DB_POOL_NAME);
        if(conn == null) {
            out.println("연결 실패");
            return;
        }
        out.println("연결 취득 완료<br><br>");
        out.println("현재 작동중인 DBCP<br>");
        PoolingDriver driver = (PoolingDriver)
DriverManager.getDriver("jdbc:apache:commons:dbcp:");
        String[] names = driver.getPoolNames();
        for(String name : names)
            out.println(name + "<br>");
        out.println("<br>");
```

```
        conn.close();
        out.println("연결 반환 완료<br>");
    %>
</body>
</html>
```

1)

해당 코드를 호출하는 JSP 페이지의 Servlet class의 일부로 만든다.

2)

getter/setter Method를 통해 관리되는 data

From:
https://wiki.ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://wiki.ledyx.xyz/doku.php?id=back-end:jsp

Last update: **2021/02/07 05:51**