[JSP 와 Servlet ]

웹 브라우저를 열어보면?

웹 브라우저는 네크워크 회선을 통해 다른 어떤 곳에 있는 컴퓨터로 웹 페이지를 보내달라고 요청을 하고, 그 컴퓨터는 해당 웹 페이지를 찾아서 우리의 컴퓨터로 보내준다.

웹 브라우저의 요청을 받아 해당하는 웹 페이지를 찾아서 보내주는 일을 하는 컴퓨터(일을 하는 소프트웨어)를 웹 서버라고 한다.

웹 브라우저가 웹 서버로 보내는 것은 URL이라는 웹 페이지의 주소이고,

웹 서버가 웹 브라우저로 보내는 것은 HTML 문서라는 텍스트파일이다.

웹 서버가 하는 일을 좀 더 정확히 표현하면, 웹 브라우저로부터 URL을 받아서 그에 해당하는 HTML문서를 찾아서 웹 브라우저로 보내주는 것이다.

HTML문서는 순수하게 텍스트로만 이루어지며, <HTML>, </HTML>, <BODY>, </BODY>, <H1>, </H1>과 같이 꺽쇠괄호로 묶여진 부분을 태그(tag) 또는 마크업(markup)이라고 한다.

태그는 웹 브라우저 상에 그대로 표시되는 것이 아니라 태그에 의해 싸여진 부분이 웹 브라우저 상에 어떻게 표시될지 지시하는 역할을 한다.

웹 페이지에는 텍스트 뿐만 아니라 그림이나 사진도 표시된다.

HTML 문서의 태그 중에 <IMG>라는 것이 있는데, 이 태그 안에는 그림이나 사진의 URL을 쓸 수 있으며 웹 브라우저는 태그를 발견하면 웹 서버로 다시 URL을 보내고, 웹 서버는 그에 해당하는 이미지 파일을 찾아 웹 브라우저로 보낸다.

그리고 똑같은 웹 서버로 똑같은 URL을 보내도 그때마다 다른 내용이 표시되는 경우도 있다.

ex) 포털 사이트에는 신문 기사의 제목들이 표시되어 있는데, 이런 제목은 그 사이트로 접속할 때마다 다른 제목으로 바뀌어서 표시된다.

웹 서버가 단순히 URL을 받아서 그에 해당하는 HTML문서를(또는 이미지 파일을) 찾아서 보내주는 일만 한다면 이런 일은 어떻게 가능한 걸까?

웹 서버는 HTML 문서 파일을 찾아서 보내주는 일 뿐만 아니라 HTML 문서를 생성하는 프로그램을 호출해서 그 프로그램의 실행 결과를 보내주는 일도 한다.

이때 호출 되는 프로그램을 “웹 애플리케이션”이라고 한다.

웹 애플리케이션이 생성하는 HTML 문서를 동적 HTML 문서라고 하고, 웹 서버 쪽에 파일 형태로 저장되어 있는 HTML문서를 정적 HTML 문서라고 한다.

웹 애플리케이션은 웹 서버와 동등하게 실행되는 독립적인 프로그램인가? 아니면 웹 서버에 종속된 모듈일까? 이 두가지 형태 모두 존재한다.

최초의 웹 애플리케이션 기술인 CGI(Common Gateway Interface)에서는 웹 애플리케이션을 웹 서버와 동등한 독립적인 프로그램 형태로 작성하였다.

이런 독립적인 프로그램을 CGI프로그램이라고 했는데, 이 프로그램에는 커다란 문제점이 있었다.

1. 첫째 독립적인 프로그램을 실행하기 위해서는 시스템 자원이 많이 필요했고,
2. 둘째 운영체제에 의해 별개로 취급되는 CGI프로그램의 실행 흐름을 웹 서버가 적절하게 제어할 수 없었다.

그 결과 웹 서버로 아주 많은 URL요청이 한꺼번에 들어오면 너무 많은 CGI프로그램이 동시에 실행되어서 컴퓨터 전체가 다운되는 일이 빈번히 발생하였다.

이런 문제로 인해 CGI 프로그램은 곧 커다란 골칫거리가 되었고, 얼마 지나지 않아 웹 서버의 모듈형태로 웹 애플리케이션을 만드는 새로운 기술이 개발되기 시작하였다.

이때 등장한 기술 중 하나가 바로 Servlet이다.

-------------------------------------------------------------------------------

**[Servlet** **기술]**

Servlet이란 자바를 기반으로 하는 웹 애플리케이션 프로그래밍 기술이다.

이 기술에서는 자바클래스 형태로 웹 애플리케이션을 작성하는데, 그 클래스를 Servlet 클래스라고 한다.

<Servlet 클래스를 작성할 때 지켜야 할 규칙>

규칙은 JCP(Java Community Process: 자바 표준 개발을 전담하는 기관)에서 제정한 Servlet 규격서(Servlet Specification)에 기술되어 있다.

그 중 가장 기본적인 규칙 몇 가지를 알아보자.

1. 첫째, Servlet 클래스는 javax.servlet 패키지에 속하는 Servlet인터페이스를 구현하도록 만들어야 한다. 이 인터페이스를 구현하는 클래스를 작성하는 것은 아주 복잡하고 번거로운 일이다. 그래서 Servlet 규격서에서는 그보다 더 쉬운 방법을 권장하고 있는데, 그것은 Servlet 인터페이스를 구현하는 javax.servlet.http.HttpServlet이라는 클래스의 서브클래스를 작성하는 방법이다. 그렇게 하면 그 서브클래스는 간접적으로 Servlet 인터페이스를 구현하게 되기 때문이다.
2. 둘째, doGet이나 doPost라는 메서드를 선언하고, 그 안에 Servlet 클래스가 호출되었을 때 해야 할 일을 써 넣어야 한다.

이 method를 작성할 때 지켜야 할 규칙이 있는데 그것은 method의 파라미터와 익셉션에 관한 것이다.

이 두 메서드는 각각 두 개의 파라미터를 받아야 하는데,

첫 번째 파라미터는 javax.servlet.http.HttpServletRequest 타입이어야 하고,

두 번째 파라미터는 javax.servlet.http.HttpServletRespons 타입이어야 한다.

그리고 메서드 밖으로 java.io.IOException과 javax.servlet.ServletException을 던질 수 있도록 선언해야 한다.

셋째, 동적 HTML 문서를 생성하기 위해서는 doGet, doPost 메서드의 두 번째 파리미터를 이용해야 한다.

이 파라미터에 대해 getWriter라는 메서드를 호출하면 printWriter 타입의 객체가 리턴되는데, 그 객체에 대해 print, println, printf 등의 메서드를 호출하면 웹 브라워저로 HTML 코드를 출력할 수 있다.

이 밖에도 지켜야 할 규칙이 많이 있지만, 이 세가지가 가장 기본적인 것들이다.

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public class HundredServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletRespons response)

throws ServletException, IOException {

PrintWriter out = response.getWriter();

out.println("<HTML>");

out.println("<HEAD><TITLE>Sum Of 1 To 100</TITLE></HEAD>");

out.println("<BODY>");

int total = 0;

for(int cnt = 1; cnt <= 100; cnt++)

total += cnt;

out.println("1 + 2 + 3 + ... + 100 = " + total);

out.println("</BODY>");

out.println("</HTML>");

}

}

**<Servlet 기술의 장점>**

Servlet 기술이 다른 웹 애플리케이션 기술보다 나은 점은 무엇인가?

Servlet 기술은 자바를 기반으로 하기 때문에 자바의 장점을 그대로 물려받는다.

자바의 플랫폼 독립성은 Servlet 클래스의 실행 코드(컴파일 결과물)를 다양한 컴퓨터와 운영체제에 옮겨서 그대로 사용할 수 있게 한다.

자바의 또 다른 장점은 네트워크 환경에서 보안이 용이하고, 프로그래밍 언어 수준에서 멀티스레드 기능이 지원된다는 것이다.

**<Servlet 기술의 단점>**

프로그래밍 작업의 효율성을 떨어뜨린다.

이것은 HTML 코드가 자바 코드안으로 들어가는 구조로 인한 것이다.

이런 구조는 Servlet 클래스가 출력하는 동적 HTML 문서의 구조를 이해하기 어렵게 만들고, 웹 페이지의 디자인 작업을 위해서도 소스 코드에 손을 대개 만드는 비효율성을 낳는다.

이런 이유로 인해 얼마 뒤 자바의 유용한 장점을 살리면서 Servlet 기술의 단점을 보완하는 새로운 기술이 개발되었는데 그것이 바로 JSP이다.

-------------------------------------------------------------------------------

[UTF-8]

ECLIPSE의 설정에서 아래 항목의 encoding 타입을 utf-8로 변경한다.

GENERAL -> workspace

JASON -> jason files

WEB -> css files, html files, jsp files

XML -> xml files

-------------------------------------------------------------------------------

**[JSP 기술]**

JSP(Java Server Page)도 Servlet과 마찬가지로 자바를 기반으로 하는 웹 애플리케이션 프로그래밍 기술입니다.

JSP 페이지는 Servlet 클래스와 반대로 HTML 문서에 자바 코드가 삽입되는 구조이다.

<% %> 스크립틀릿 (scriptlet)

<% %> 사이에는 자바 명령문(들)을 쓸 수 있다.

이 명령문들은 웹 서버에 의해 일반 자바 프로그램과 똑같은 방법으로 실행된다.

<%= %> 익스프레션 (expression)

<%= %> 사이에는 자바 식을 쓸 수 있다.

이 식은 웹 서버 쪽에서 계산되고 그 결과만 웹 브라우저로 전송된다.

JSP 페이지는 Servlet 클래스와 달리 컴파일 과정이나 등록 과정이 필요하지 않다.

텍스트 에디터로 소스 코드를 작성해서 웹 서버에 속한 directory에 저장해 놓기만 하면 된다.

요즘은 JSP 기술과 Servlet 기술을 혼용한 새로운 프로그래밍 방법이 권장되고 있다.

프로그램의 기능을 구현하는 복잡한 로직은 Servlet 클래스 안에 기술하고, 그 결과를 가져다가 출력하는 일만 JSP 페이지가 담당하도록 만드는 방법이다.

public class HundredServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletRespons response)

throws ServletException, IOException {

int total = 0;

for(int cnt = 1; cnt <= 100; cnt++)

total += cnt;

request.setAttribute("result", total);

RequestDispatcher rd

= request.getRequestDispatcher("Hundred.jsp");

rd.forward(request, response);

}

}

<HTML>

<HEAD><TITLE>Sum Of 1 To 100</TITLE></HEAD>

<BODY>

1부터 100까지의 합 : ${result}

</BODY>

</HTML>

[JSP 사용 준비]

JDK 설치

APACHE TOMCAT 설치

netstat -na => 네트워크 상태를 보여준다(프로토콜, 로컬주소, 포트번호......)

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

[JSP]

JSP의 문법에는 3가지 형태가 있다.

1. 스크립틀릿(scriptlet)

<% %> 스크립틀릿

<%@ %> 지시자 (directive)

<%= %> 익스프레션 (expression)

2. 익스프레션 언어 (expression language)

3. 액션 (action)

[선언부]

<%! %> 변수 선언이나 메서드 선언을 쓸 수 있다.

<%! final static int MAX = 10000; %>

<%! private int add(int n1, int n2) {

...........

return sum; }

%>

**지시자(directive)**는 JSP의 다른 문법들(스크립팅 요소, 익스프레션 언어, 액션)과는 다른 목적으로 사용되는 문법.

웹컨테이너가 JSP페이지를 Servlet 클래스로 변환할 때 필요한 여러 가지 정보들을 기술하기 위해서 사용하는 문법이다.

[JSP 페이지에서 사용할 수 있는 지시자]

-page 지시자

-include 지시자

-taglib 지시자

<%@ 로 시작해서 %> 로 끝나야 한다.

<%@ 바로 다음에는 지시자 이름이 와야한다.

지시자 이름 다음에는 여러가지 attribute가 올 수 있다.

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

contentType

JSP 페이지가 생성하는 문서의 종류와 그 문서를 웹 브라우저로 전송할 때 사용되는 인코딩 타입

language

스크립팅 요소안에서 사용할 프로그래밍 언어.

pageEncoding

JSP 페이지의 인코딩 타입.

import

스크립팅 요소 안에서 사용할 자바 클래스와 인터페이스를 임포트하기 위해 사용.

**[JSP 페이지가 자동으로 임포트하는 패키지]**

JSP 페이지가 서블릿 클래스로 변환될 때 자동으로 임포트되므로 import 어트리뷰트에 쓰지 않아도 된다.

-javax.sevlet

-javax.servlet.jsp

-javax.servlet.http

[include 지시자]

다른 JSP 페이지나 HTML 문서를 불러다가 현재 JSP 페이지의 일부로 만들기 위해서 사용한다.

불러올 대상은 file이라는 어트리뷰트를 이용해서 지정할 수 있으며, 이 어트리뷰트 값은 이 지시자가 속하는 JSP페이지를 기준으로 한 상대적인 URL로 해석된다.

[taglib 지시자]

액션태그를 사용할 때 필요하다.

[주석]

<!-- HTML 주석 -->

/\* JAVA 주석 \*/

// JAVA 주석

<%-- JSP 주석 --%>

C:\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\JspStudy\org\apache\jsp\base

**[JSP 페이지의 내장 변수]**

일반적인 자바 프로그램에서는 변수를 사용하기 전에 반드시 선언을 해야 한다.

JSP 페이지의 내장변수(implict variable)는 선언하지 않고도 사용할 수 있다.

웹 컨테이너가 JSP 페이지를 Servlet 클래스로 변환할 때 자동으로 선언해 주기 때문이다.

[JSP 페이지에서 사용할 수 있는 내장 변수]

request, response, out, application, config,

pageContext, session, page, exception

[request 내장 변수]

서블릿 클래스에 있는 doGet, doPost 메서드의 첫 번째 파라미터와 동일한 역할을 하고,

타입도 동일하게 javax.servlet.http.HttpServletRequest이다.

그래서 그 파라미터와 똑같은 방법으로 사용할 수 있다.

[out 내장 변수]

JSP 페이지에서 출력하는 데이터는 out.print, out.println메서드를 호출할 경우뿐만 아니라 HTML 코드나 익스프레션을 통해 출력할 경우에도 모두 out 내장 변수를 통해 출력된다.

이 내장 변수는 출력하는 데이터를 일단 퍼버에 저장했다가 한꺼번에 전송하는데, 그 버퍼의 크기는 웹 컨테이너의 종류에 따라 달라질 수 있다.

autoFlush 속성

out 내장 변수의 출력 버퍼는 데이터가 가득 차면 버퍼에 있는 모든 내용을 자동으로 웹 브라우저로 전송한다.

이것을 auto flush(오토 플러시)기능이라고 한다.

[response 내장 변수]

response 내장 변수는

서블릿 클래스에 있는 doGet, doPost 메서드의 두 번째

파라미터와 동일한 역할을 하고, 타입도 동일하게

javax.servlet.http.HttpServletResponse이다.

sendRedirect 메서드는 현재 실행 중인 JSP 페이지의 실행을

중단하고 다른 웹 자원이 대신 호출되록 만드는 기능을

한다.

response.sendRedirect("http://www.daum.net");

sendRedirect 메서드의 호출 전과 호출 후에 웹 브라우저로

데이터를 출력하면 않된다.

[application 내장 변수]

application 내장 변수는 웹 어플리케이션에 관련된 여러가지 기능을 제공한다.

여기서 말하는 웹 어플리케이션이란 서블릿 클래스나 JSP 페이지와 같은 웹 프로그램을 말하는 것이 아니라, 특정 웹 어플리케이션 디렉토리에 속하는 모든 웹 자원들의 집합을 의미한다.

[파일 입출력]

JSP 페이지에서 파일을 읽기 위해서는

java.io.FileReader 클래스 나 java.io.FileInputStream

클래스를 이용하면 된다.

파일을 쓰기 위해서는

java.io.FileWriter, java.io.PrintWriter, java.io.FileOutputStream등의 클래스를 이용한다.

C:\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\JspStudy\WEB-INF\BBS

int age = Integer.parseInt(request.getParameter("age"));

[sendRedirect 메서드를 통한 데이터 전달 방법]

response.sendRedirect("BBSPostResult.jsp?result=SUCCESS");

response.sendRedirect("BBSPostResult.jsp?sum=10&count=5");

[forward 메서드 와 sendRedirect 메서드의 차이점]

1.sendRedirect 메서드는 URL을 웹 브라우저로 보내서

간접적으로 웹 자원을 호출하지만,

forward 메서드는 웹 서버 쪽에서 직접 호출한다.

2.sendRedirect 메서드를 이용하면 다른 웹 서버에 있는 웹 자원도 호출할 수 있지만,

forward 메서드를 이용하면 같은 웹 서버, 웹 어플리케이션 디렉토리 내에 있는 웹 자원만 호출할 수 있다.

3.sendRedirect 메서드를 이용하면 텍스트 데이터만 전달할 수 있지만,

forward 메서드를 이용하면 객체 형태의 모든 데이터를 전달할 수 있다.

4.sendRedirect 메서드를 통해 한글 데이터를 전달하기 위해서는 URL인코딩을 해야 하지만,

forward 메서드에서는 그런 작업이 필요치 않다.

RequestDispatcher dispatcher

= request.getRequestDispatcher("Result.jsp");

request.setAttribute("NAME", "향단이");

request.setAttribute("AGE", "18");

request.setAttribute("TEL", "0101234-5678");

dispatcher.forward(request, response);

forward 메서드는 request 내장 변수를 통해 전달하는 것은 attribute뿐만이 아니라,

request 내장 변수 안에 있는 모든 데이터를 함께 전달한다.

예) 웹브라우저에서 입력한 데이터도 포함된다.

[include 메서드]

호출된 JSP 페이지가 끝나고 나면 실행 흐름의 제어가 본래의 JSP 페이지로 돌아온다.

여러 JSP 페이지가 공통을 사용하는 코드를 호출하고자 할 때 사용하는 것이 좋다.

include 메서드를 이용하면 JSP 페이지가 아닌 다른 웹 자원/서블릿 클래스(HTML 문서 등)도 호출할 수 있다.

------------------------------------------------------------------------------------

[쿠키와 세션]

쿠키 기술

전달할 데이터를 웹 브라우저로 보냈다가 웹 서버 쪽으로 되돌려 받는 방법을 사용한다.

첫 번째 웹 브라우저로 HTML 문서를 보낼 때 전달할 데이터를 함께 보낸다.

그러면 웹 브라우저는 그 데이터를 저장해 두었다가 두 번째 웹 컴포넌트를 호출할 때 URL과 함께 웹 서버로 보낸다.

세션 기술

웹 브라우저를 거치지 않고 웹 서버에 있는 데이터 영역을 통해 데이터를 전달하는 방법을 사용한다.

첫 번째 웹 컴포넌트가 웹 서버 쪽에 데이터를 저장해 놓고 그 데이터를 읽기 위해 필요한 세션 아이디만 웹 브라우저로 보낸다.

그러면 웹 브라우저는 그 아이디를 저장해 두었다가 두 번째 웹 컴포넌트를 호출할 때 웹 서버로 보낸다.

그렇게 하면 그 웹 컴포넌트는 세션 아이디를 가지고 해당 데이터를 찾아서 사용할 수 있다.

세션 아이디를 전송하기 위해 사용하는 것도 쿠키기술이다.

[쿠키 기술]

쿠키 기술을 사용하는 목적은 웹 컴포넌트 간의 데이터를 전달하기 위해서.

[새로운 쿠키 데이터를 저장하는 방법 - 입력기능]

쿠키 데이터를 웹 브라우저 쪽에 저장하기 위해서는 두 가지 작업을 해야 한다.

1. Cookie 클래스의 객체를 만들어야 한다.

2. addCookie 메서드를 호출한다.

String 타입으로 만들어야 한다.(이름, 값)

Cookie cookie = new Cookie("AGE", "18");

respone.addCookie(cookie);

#웹 브라우저가 쿠키를 저장할 때 웹 서버의 주소도 함께 저장해 놓는다.

#JSP 페이지에는 출력 버퍼가 있다.

그 버퍼의 내용이 출력되기 전에 addCookie 메서드를 호출하면 쿠키 데이터가 제대로 될 수 있다.

[쿠키 데이터를 읽는 방법 - 조회 기능]

웹 브라우저는 웹 서버가 아무런 요청을 하지 않아도

웹 서버로 URL을 보낼 때마다 그 URL에 포함된 웹 서버의 주소에 해당하는 모든 쿠키를 찾아서

웹 서버로 보낸다.

getCookie 메서드는 JSP 페이지에서는 request 내장 변수에 대해 호출해야 한다.

이 메서드는 웹 브라우저가 보낸 모든 쿠키를 Cookie 배열로 만들어서 리턴하기 때문에 리턴 값을 Cookie 배열 변수에 받아야 한다.

Cookie cookies[] = request.getCookies();

\* getCookie 메서드는 웹 브라우저로부터 온 쿠키 데이터가 하나도 없을 경우에는 null을 리턴한다.

이 메서드가 리턴한 Cookie 배열에서 특정 쿠키를 찾기 위해서는 배열에 있는 Cookie 객체를 하나씩 가져다가 이름을 비교해서 찾을 수 밖에 없다.

String name = cookie.getName();

String value = cookie.getValue();

웹 브라우저가 쿠키와 함께 저장해 놓았던 웹 서버의 주소를 URL에 포함된 웹 서버의 주소와

글자 그대로 비교한다.

그러므로 쿠키 기술을 이용해서 웹 어플리케이션을 구현한 다음에는 거기에 속하는 모든 웹 컴포넌트들을 동일한 웹 서버 주소로 호출해야 한다.

HTTP 요청 메시지

웹 브라우저는 웹 서버로 URL을 보낼 때 쿠키도 함께 보내며, 이 때 그 밖의 데이터들도 함께 전송한다.

이런 데이터의 묶음을 HTTP 요청 메시지라고 한다.

[쿠키 데이터를 수정하는 방법 - 수정 기능]

Cookie 객체를 만들어서 addCookie 메서드에 넘겨주면 된다. 그러면 이 메서드는 파라미터로 받은 쿠키를 웹 브라우저로 보낼 것이고, 웹 브라우저는 자신이 갖고 있던 쿠키 중에 같은 이름의 것이 있으면 그것을 지우고 새로운 쿠키를 저장한다.

[쿠키 데이터를 삭제하는 방법 - 삭제 기능]

쿠키의 남은 수명을 0으로 설정하는 방법을 사용한다.

웹 브라우저는 실행을 끝낼 때 자신이 가지고 있던 쿠키를 모두 지워 버리는 것이 기본 동작이다.

쿠키에 수명을 지정해 놓으면 웹 브라우저를 시작하고 끝내는 동작과 상관없이 그 쿠키가 일정기간 동안 보존된다.

쿠키에 수명을 지정하기 위해서는 addCookie 메서드를 호출하기 전에 Cookie 객체에 대해 setMaxAge라는 메서드를 호출하면 된다.

setMaxAge 메서드에 쿠키의 남은 수명을 초단위로 설정한다.

cookie.setMaxAge(3600);

0 이라는 값을 넘겨주면 웹 브라우저는 쿠키의 수명이 이미 다했다고 판단해서 똑같은 이름의 쿠키를 찾아서 삭제할 것이다.

minus 값을 넘겨주면 이 메서드를 전혀 호출하지 않았을 때와 마찬가지로 웹 브라우저가 끝날 때 쿠키를 제거할 것이다.

웹 서버는 이 값(남은 수명값)을 가지고 쿠키의 만료기한을 계산해서 'Expires=Wdy, DD-Mon-YY HH:MM:SS GMT' 포맷을 만들어서 웹 브라우저로 전송한다.

[쿠키가 특정 경로명을 갖는 URL로만 전송되도록 만드는 방법]

웹 브라우저는 웹 서버로 URL을 보낼 때 그 웹서버에 속하는 모든 쿠키를 함께 보내는 것이 기본적인 동작이지만, 때로는 이런 전송 범위를 좁혀야 할 경우가 있다.

그럴 때는 addCookie 메서드를 호출하기 전에 Cookie 객체에 대해 setPath라는 메서드를 호출한다.

setPath 메서드는 특정 경로명을 갖는 URL로만 쿠키가 전송되도록 만드는 기능을 한다.

파라미터로 경로명을 넣어주면 된다.

URL 경로명은 반드시 /로 시작해야 하고, 마지막도 /로 끝내는 것이 좋다.

cookie.setPath("/cookie/sub/");

----------------------------------------------------------------------------------

**[세션 기술]**

session - 사전적 의미는 여러 명의 사람들이 회의를 시작해서 끝내기까지의 기간을 말한다.

세션을 시작하고 끝내는 기능과 그 기간 동안 웹 컴포넌트들이 주고 받을 수 있는 기능이 있다.

**[세션의 적용 범위]**

cookie 데이터는 같은 웹 서버 내에 있는 모든 웹

컴포넌트들과 자바가 아닌 웹 어플리케이션 프로그램들

까지도 공유할 수 있다.

session 데이터는 같은 웹 어플리케이션 디렉토리에 있는

웹 컴포넌트들끼리만 공유할 수 있다.

세션의 적용 범위도 웹 어플리케이션 디렉토리 내로 국한된다.

세션기술에서는 세션을 이런 식으로 관리하기 위해서

쿠키형태로 전송되는 세션 아이디 뒤에 해당 웹 어플리케이션 디렉토리의 URL 경로명을 붙여서 전송한다.

**[JSP 페이지에서 세션 기술을 사용하는 방법]**

session 내장 변수를 사용하면 세션 데이터 영역에

데이터를 저장할 수도 있고, 그 영역에 있는 데이터를

읽어오거나 삭제할 수 있다.

session.setAttribute("ID", "Hong1234");

session.getAttribute("ID");

session.invalidate(); //세션을 끝내는 메서드

JSP 페이지에서 세션을 사용할 필요가 없으면

page 지시자의 sessiong 어트리뷰트에 'false'라는 값을

지정하면 된다.

**[에러 페이지 처리]**

웹 컴포넌트의 코드 구조를 망가뜨리지 않으면서 익셉션을

처리하는 가장 좋은 방법은 에러를 처리하는 부분만

따로 떼어서 별도의 웹 컴포넌트로 만든 다음에

호출하는 것이다.

이런 웹 컴포넌트를 에러 페이지라고 한다.

회원가입

========

아이디:

비밀번호:

비밀번호확인:

이름:

주소:

전화번호:

회원가입동의:

[확인][취소]

==> MemberInsertResult.jsp

if(session.getAttribute("ID") == null)

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0">

[Servlet]

서블릿 기술에서 웹 어플리케이션을 구현하기 위해 작성

해야하는 코드는 '서블릿 클래스'이다.

웹 컨테이너는 서블릿 클래스를 가지고 서블릿 객체를 만든 다음에

그 객체를 초기화해서 웹 서비스를 할 수 있는 상태로 만드는데

이런 작업을 거친 서블릿 객체만 서블릿이라고 할 수 있다.

멀티-스레드 모델에서 일어날 가능성이 있는 데이터 공유문제는 서블릿 클래스의 인스턴스 변수(instance variable, static이 아닌 클래스의 멤버 변수)로 인한 것인데,

이런 변수를 선언해서 사용하지 않으면 된다.

[서블릿 클래스의 작성을 위한 준비]

서블릿 클래스를 작성할 때 지켜야 할 규칙

1.서블릿 클래스는 javax.servlet.http.HttpServlet 클래스를 상속하도록 만들어야 한다.

2.doGet 또는 doPost 메세드 안에 웹 브라우저로부터 요청이 왔을 때 해야 할 일을 기술해야 한다.

3.HTML 문서는 doGet, doPost 메서드의 두 번째 파라미터를 이용해서 출력해야 한다.

[서블릿 클래스 컴파일하기]

javac HundredServlet.java ==> HundredServlet.class

javac -encoding utf-8 HundredServlet.java

<실행 java HundredServlet>

[서블릿 등록]

<servlet>

<servlet-name>Hundred</servlet-name>

<servlet-class>basic.HundredServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Hundred</servlet-name>

<url-pattern>/hundred</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file-list> 태그는 사용자가 웹 어플리케이션의

루트컨텍스트 경로 즉, 서버의 루트 주소만을 입력하여

접근했을 경우 가장 먼저 띄워줄 경로를 설정하는 태그이다.

서버는 <welcome-file></welcome-fil>을 차례대로 읽어가며

해당 자원이 존재하는 경우 그 자원을 사용자에게 돌려주게

된다.

<pre> 요소 내의 텍스트는 시스템에서 미리 지정된 고정폭 글꼴(fixed-width font)을 사용하여 표현되며, 텍스트에 사용된 여백과 줄바꿈이 모두 그대로 브라우저 화면에 나타납니다.

이러한 <pre> 요소를 사용하면, 독특한 서식의 텍스트나 컴퓨터 코드 등을 HTML 문서에 그대로 표현할 수 있습니다.

[서블릿에서 세션 기술을 사용하는 방법]

HttpSession session = request.getSession();

getSession() <= 세션을 시작하는 메서드

세션 정보를 가져오는 메서드

session.setAttribute("FOOD", food);

session.invalidate(); <= 세션을 종료하는 메서드

[서블릿 클래스에서 에러 페이지 호출하기]

서블릿 클래스는 page 지시자를 사용할 수 없기 때문에

에러 페이지를 호출하려면 forward 메서드를 이용하여야 한다.

} catch (NumberFormatException e) {

RequestDispatcher dispatcher

= request.getRequestDispatcher("DataError.jsp");

dispatcher.forward(request, response);

}

[try문을 사용하지 않고 에러 페이지를 호출하는 방법]

익셉션을 발생하는 JSP 페이지에서 page 지시자에 errorPage 애트리뷰트를 쓰고,

거기에 URL 경로명을 지정해 놓으면 그 JSP 페이지 안에서 익셉션이 발생했을 때 그 URL 경로명 위치에 있는 웹 자원이 호출된다.

<%@page errorPage="DataError.jsp"%>

[좀 더 자세한 정보를 보여주려면?]

그런 일은 에러페이지의 exception 내장 변수를 이용해서 하면 된다.

page 지시자의 errorPage 애트리뷰트를 이용해서 호출한 에러 페이지에는 익셉션 객체가 자동으로 전달되며, 그 익셉션 객체는 에러 페이지의 exception 내장 변수에 저장되기 때문이다.

- exception 내장 변수를 사용하기 위해서 먼저 하여야 할 일

기본적인 JSP 페이지에는 exception 내장 변수가 생기지 않기 때문에 page 지시자에 isErrorPage라는 애트리뷰트를 쓰고 그 값으로 'true'를 지정해야 한다.

<%@page isErrorPage="true"%> <= exception 내장 변수가 생기도록 만드는 애트리뷰트 값

String message = exception.getMessage(); <= 에러 메시지를 가져오는 메서드

### HTTP응답 코드 값이 200이 오면 정상적으로 실행 결과라는 것이다.

코드값 200을 받으면 정상적으로 DataError.jsp를 출력할 수가 있다.

간혹 코드값 500이 넘어 오는 경우가 있는데,

이때는 우리가 만든 프로그램보다 시스템에서 발생하는 에러가 우선이기 때문에 시스템 에러페이지가 나타나게 된다.

[서블릿 클래스에서 에러 페이지 호출하기]

서블릿 클래스에서는 page 지시자를 사용할 수 없기 때문에 에러 페이지를 호출하려면 forward 메서드를 이용할 수 밖에 없다.

[web.xml 파일에 에러 페이지 등록하기]

<익셉션 타입별로 에러 페이지를 등록하는 방법>

web.xml 파일에는 에러 타입별로 에러 페이지를 등록해 놓을 수 있다.

에러 페이지는 해당 웹 어플리케이션 디렉토리 안에서 그에 해당하는 타입의 익셉션이 발생했을 때 자동으로 호출된다.

이 방법을 사용하면 여러 웹 컴포넌트에서 발생한 익셉션을 하나의 에러 페이지가 처리하도록 만들 수 있다.

JSP 페이지에서는 page 지시자의 errorPage 어트리뷰트를 쓰지 않으면 된다.

<web.xml 파일에 등록한 에러 페이지를 작성할 때 지켜야 할 규칙>

exception 내장 변수가 필요하다면 page 지시자의 isError 어트리뷰트에 true 값을 지정하고, HTTP 상태 코드를 200으로 바꿔주면 된다.

page 어트리뷰트의 errorPage 어트리뷰트에 지정한 에러 페이지가

web.xml 파일에 등록한 에러 페이지보다 우선권을 갖는다.

web.xml 파일에는 여러 타입의 익셉션 타입에 대한 에러 페이지를 함께 등록할 수 있다.

web.xml 파일에는 슈퍼클래스-서브클래스 관계에 있는 여러 익셉션 타입에 대한 에러 페이지를 함께 등록해 놓은 수도 있다.

이럴 경우에 익셉션이 발생하면 가장 가까운 타입에 해당하는 에러 페이지가 호출된다.

서블릿 클래스에서 발생하는 익셉션도 web.xml에 등록한 에러페이지를 이용하여 처리할 수 있다.

서블릿 클래스를 작성할 때 doGet, doPost 메서드 안에서 발생한 익셉션을 메서드 밖으로 던지도록 만들어야 한다.

[서블릿 클래스의 doGet, dbPost 메서드 밖으로 익셉션을 던지는 다른 방법]

서블릿 클래스에서 doGet, dbPost 메서드의 throws 절에 기술할 수 있는 익셉션의 종류는 정해져 있기 때문에, 이 메서드 밖으로

던질 수 있는 익셉션도 한정되어 있다.

#컴파일러에서 에러가 발생하는 java.lang.Exception 이나 java.sql.SQLException은 메서드 밖으로 던지려고 하면 컴파일 에러가 발생한다.

#해당 익셉션을 javax.servletg.ServletException 객체로 포장한 다음에 던지면 된다.

public class DBReaderServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

try {

.........

} catch(java.sql.SQLException e) {

throw new ServletException(e); <= SQL Exception을 ServletException으로 조장해서 메서드 밖으로 던진다.

}

}

}

[HTTP 상태 코드별로 에러 페이지를 등록하는 방법]

톰캣은 웹 브라우저로부터 받은 URL에 해당하는 웹 자원이 없으면 [HTTP Status 404]라는 웹 페이지를 만들어서 웹 브라우저로 보낸다.

URL에 해당하는 웹 자원이 없을 때 그에 해당하는 웹 페이지를 만들어 호출되도록 하는 것은 프로그래머가 해야 할 일이다.

<error-page>

<error-code>404</error-code>

<location>/NotFoundError.jsp</location>

</error-page>

NotFoundError.jsp - 등록한 에러 페이지는 에러 페이지에 전달되는 익셉션 객체가 없으므로 page지시자의 isErrorPage 애트리뷰트 값을 'true'로 지정해야 할 필요는 없으나, HTTP 상태 코드를 200으로 바꿔주는 일은 해야 한다.

<% response.setStatus(200); %>

웹 서버는 웹 컴포넌트 안에서 발생하는 모든 익셉션에 대해 500이라는 상태 코드를 발생하기 때문에 500 상태 코드에 대한 에러 페이지는 그런 모든 익셉션에 대해 적용될 수 있다.

그렇다면 web.xml 파일에 이런 에러 페이지와 익셉션 타입별로 등록한 에러 페이지가 모두 있을 때는 어느 것이 우선권을 가지게 될까요?

이럴 때는 익셉션 타입별로 등록된 에러 페이지가 우선권을 갖습니다.

================================================================

[Expression Language]

익스프레션 언어는 JSP 규격서가 처음 만들어질 때부터 있던 문법이 아니라 나중에 새롭게 추가된 문법이다.

이 문법을 이용하면 데이터를 출력하는 코드를 함축적이고 간결하게 구사할 수가 있어서 코드의 가독성과 유지보수 용이성에 큰 도움이 된다.

**[Expression Language = EL식]**

익스프레션 언어는 식을 중심으로 코드를 기술하는 언어.

${ 와 }로 둘러싸서 표현한다.

${cnt+1} <= 익스프레션 언어의 식

cnt => 자바 변수의 이름이 아니다.

EL 식에 포함된 데이터 이름은 기본적으로 어트리뷰트의

이름으로 해석된다.

어트리뷰트란 setAttribute, getAttribute, removeAttribute 메서드를 통해 저장되고 관리되는 데이터를 의미한다.

<%= cnt+1 %> <= 익스프레션의 식 cnt => 자바 프로그램의 변수이름

익스프레션 언어에서 사용할 수 있는 산술연산자

* 산술연산자 : + - \* / %
* 비교연산자 : < > >= <= !=
* 논리연산자 : && ||

자바클래스의 정적메서드를 호출해서 그 결과를 출력할 수도

있다. ${m:sqrt(100)}

Expression 언어의 유일한 목적은

-> 식을 계산해서 그 결과를 출력하는 것이기 때문에 아래와 같이 하나의 문법으로 표현할 수 있다.

이 문법을 EL식이라 한다.

${식}

'식'의 위치에는 들어갈 수 있는 것.

* 데이터 이름 하나로만 구성된 식
* 연산자를 포함하는 식
* 자바의 Static(정적) method를 호출하는 식

Servlet과 JSP 기술에서 사용할 수 있는 setAttribute, getAttribute, removeAttribute 메서드는 어트리뷰트의 종류도 4가지이다.

* page Attribute => pageContext 내장 변수
* request Attribute => request 내장 변수
* session Attribute => session 내장 변수
* application Attribute => application 내장 변수

주의할 점:

EL 식 안에 데이터 이름이 4종류의 Attribute 중

어느 것으로도 해석될 수 있다.

${RESULT}

JSP 페이지 안에 이름이 같고 종류가 다른 둘 이상의

어트리뷰트가 존재한다면?

사용범위가 좁은 어트리뷰트부터 점점 더 사용범위가 넓은

어트리뷰트 순으로 진행된다.

EL 식 안에 있는 데이터 이름이 해석되는 순서

page --> request --> session --> application

attribute attribute attribute attribute

이 순서에 상관없이 특정한 종류의 어트리뷰트를 지정해서

출력하고 싶을 때는?

데이터 이름 앞에 그 데이터 영역에 해당하는

pageScope, requestScope, sessionScope, applicationScope

이라는 단어를 붙여 표시하면 된다.

${pageScope.SUM}

${requestScope.SUM}

${sessionScope.SUM}

${applicationScope.SUM}

**EL 식에 사용된** pageScope, requestScope, sessionScope, applicationScope이름은

익스프레션 언어의 내장 객체 이름이다.

**[익스프레션 언어의 내장 객체]**

EL 식 안에서 사용되는 내장 객체는 익스프레션 언어에 속하는

것이고 스크립팅 요소 안에서는 사용할 수 없다.

pageScope page 어트리뷰트의 집합

requestScope request 어트리뷰트의 집합

sessionScope session 어트리뷰트의 집합

applicationScope application 어트리뷰트의 집합

param 웹 브라우저로부터 입력된 데이터의 집합

paramValues 웹 브라우저로부터 입력된 데이터의 집합

(똑 같은 이름의 데이터가 여럿일 경우 사용)

header HTTP 요청 메시지에 있는 HTTP 헤더의 집합

headerValues HTTP 요청 메시지에 있는 HTTP 헤더의 집합

(똑 같은 이름의 HTTP 헤더가 여럿일 경우 사용)

cookie 웹 브라우저로부터 전송된 쿠키의 집합

initParam 웹 어플리케이션의 초기화 파라미터 집합

pageContext JSP 페이지의 환경 정보의 집합

**[param 내장 객체]**

웹 브라우저에서 <FORM> 엘리먼트를 통해 입력된 데이터를

가져올 때 사용하는 내장 객체이다.

사용방법

1. ${param.NUM}

2. ${param["COLOR"]}

<FORM>엘리먼트를 통해 똑 같은 이름의 데이터가 여러개 입력될

경우도 있다. <select>, <checkbox>, ...

이런 경우는 paramValues를 사용한다.

${paramValues.ANIMAL[0]}

${paramValues.ANIMAL[1]}

**[header 내장 객체]**

HTTP 요청 메시지에 포함된 HTTP 헤더 값을 가져올 때 사용하는

내장 객체이다.

1. ${header.HOST}

2. ${header["User-Agent"]}

1. HTTP 헤더 이름이 자바 식별자 명명규칙을 따르지 않을 때는

사용할 수 없다.

헤더 이름에 영문자, 숫자, $, \_ 가 아닌 특수문자가

포함되어 있을 경우는 사용할 수 없다.

${header.User-Agent} (잘못된 EL식)

${header["User-Agent"]}(올바른 EL식)

HTTP 요청 메시지 안에 똑같은 이름의 HTTP 헤더가 둘 이상있을

때는 headerValues 내장 객체를 사용해야 한다.

${headerValues.Accept[0]}

${headerValues.Accept[1]}

[cookie 내장 객체]

웹 브라우저가 웹 서버로 보낸 쿠키를 가져올 때 사용하는

내장 객체이다.

1. ${cookie.CART}

2. ${cookie["USER\_NAME"]}

주의할 점은

EL 식이 가져오는 것은 쿠키의 값이 아니라 쿠키 객체이다.

그러므로 이런 EL 식을 JSP 페이지 안에 써 놓으면 사용자에게

의미없는 쿠기 객체의 참조 값이 출력될 것이다.

쿠키 값을 출력하기 위해서는 어떻게 해야 할까요?

${cookie.CART.value}

${cookie.CART["value"]}

${cookie["USER\_NAME"].value}

${cookie["USER\_NAME"]["value"]}

쿠키 객체 안에는 쿠키 값 외에도 쿠키가 속하는 도메인 이름,

URL 경로명, 쿠키의 수명 같은 중요한 정보들이 들어 있다.

이런 정보를 출력하기 위해서는 EL 식에서 value 대신에

domain, path, maxAge라는 이름을 써 넣으면 된다.

${cookie.CART.domain}

${cookie.CART["path"]}

${cookie.["CART"]["maxAge"]}

[initParam 내장 객체]

웹 어플리케이션의 초기화 파라미터 값을 가져다가 출력할 때

사용하는 내장 객체

${initParam.DB\_NAME}

${initParam["DB\_NAME"]}

[pageContext 내장 객체]

JSP 페이지의 주변 환경에 대한 정보를 제공하는 객체.

${pageContext.request}

${pageContext["request"]}

${pageContext.request.requestURI}

requestURI 메서드의 리턴 값을 가져온다.

[익스프레션 언어의 연산자]

산술연산자 + - \* / % div(나눗셈) mod(나머지)

비교연산자 < > <= >= == != lt gt le ge eq ne

논리연산자 && || ! and or not

조건연산자 ? :

엠프티연산자 empty

대괄호와 마침표연산자 [] .

괄호 ()

[문자열 비교]

${gender == "female"}

${"CAR" < "CAT"}

${"CAT" > "DOG"}

[EL 식에 쓸 수 있는 상수]

정수 : 0 ~ 9의 숫자로 표시, 부호 포함 가능

부동소수점수 : 0 ~ 9의 숫자와 소수점의 조합으로 표시.

부호포함가능. 10의 거듭제곱으로 표시할 경우에는

가수 뒤에 E나 e를 쓰고, 그 뒤에 지수를 써야한다.

3.14, 5.0, -10.1, 0.5E10, 3e10

문자열 : 큰 따옴표나 작은 따옴표로 묶어서 표시.

불리언 : true, false

객체 : 객체가 없을 경우에는 null,

그 밖의 경우에는 상수로 표시할 수 없다.

[엠프티 연산자

데이터의 존재 여부를 확인하는 단항 연사자이다.

${empty NAME}

[연산자의 우선순위를 바꾸는 괄호 연산자]

${2 + 3 \* 4}

${(2 + 3) \* 4}

[대괄호 연산자와 마침표 연산자]

대괄호 연산자와 마침표 연산자로 가리킬 수 있는

데이터 항목

- 배열 항목

- java.util.List 객체의 데이터 항목

- java.util.Map 객체의 데이터 항목

- JavaBean 프로퍼티

배열과 java.util.List 객체의 데이터 항목은 반드시

대괄호 연산자를 이용해서 가리켜야 한다.

java.util.Map 과 JavaBean 프로퍼티는 대괄호 연사자와

마침표 연산자 중 어느 것을 사용해서 가리킬 수 있다.

**[java.util.List]**

java.util.List는 자바의 표준 라이브러리에 있는

인터페이스 이름이다.

java.util.List 의 구현 클래스인 java.util.ArrayList나

java.util.LinkedList의 객체를 만들어서 사용해야 한다.

**[java.util.Map]**

java.util.Map은 자바의 표준 라이브러리에 있는

인터페이스 이름이다.

이 인터페이스에는 데이터 항목을 <이름, 값>쌍으로

만들어서 저장하고 관리할 수 있는 메서드들이 있다.

**[JavaBean 프로퍼티]**

JavaBean 규격서에 따라 작성된 자바 클래스를 말한다.

규격서에 따르면 이 클래스에는 파라미터가 없는 생성자가

반드시 있어야 하고, 클래스 외부에서 필드(멤버변수)를

필요로 할 때는 직접 접근할 수 없고 반드시 메서드를

통해서 접근해야 한다.

Bean VO (Value Object)

DTO (Data Transfer Object)

---------------------------------------------------------------

[액션 태그]

jsp 규격서에서는 XML 기술을 이용해서 기존의 JSP 문법을

확장하는 메커니즘을 제공하고 있다.

이렇게 확장된 문법은 XML 태그형태를 띠고 있기 때문에

액션태그(action tag)라고 부르며 간단히 액션(action)

이라고 한다.

-----------------------------------------------------------------

표준 액션(standard action)

JSP 페이지에서 바로 사용할 수 있다.

커스텀 액션(custom action)

별도의 라이브러리를 설치해야 사용할 수 있다.

--------------------------------------------------------------

[표준 액션]

표준 액션에는 모든 태그의 이름 앞에 jsp라는 접두어가

붙고, 커스텀 액션에는 그 밖의 접두어가 붙는다.

<jsp:include page="/test/sub.jsp"/>

HTML 태그는 웹 브라우저로 바로 전달되나,

표준액션은 웹 컨테이너 쪽에서 실행되고 그 결과만

출력된다.

---------------------------------------------------------------

<jsp:include> 표준 액션

JSP 페이지에 다른 웹 자원(JSP페이지, HTML문서..)을

포함시키고자 할 때 사용한다.

이 액션에는 포함할 웹 자원의 URL을 지정하는

page 어트리뷰트를 써야 한다.

#액션 태그는 XML 문법을 따르므로

단독 태그인 경우에는 /> 로 끝나도록 만들어야 한다.

<jsp:include page="Now.jsp"/>

<%@ include file="Now.jsp" %>

<jsp:include> 표준 액션은 JSP 페이지가 실행될 때

동적으로 해당 웹 자원을 읽어서 포함하지만

include 지시자는 JSP 페이지가 서블릿 클래스로

변환될 때 해당 웹 자원을 읽어서 서블릿 클래스 코드의

일부로 만든다.

---------------------------------------------------

<jsp:forward page="Next.jsp"/>

JSP 페이지를 실행하다가 다른 JSP페이지로 제어를

넘기고자 할 때 사용한다.

---------------------------------------------------

자바빈의 호출에 사용되는 표준 액션

<jsp:userBean>

<jsp:getProperty>

<jsp:setProperty>

자바빈은 자바빈 규격서에 따라 만들어진 클래스를 말하며,

이 클래스의 내부 데이터는 get-메서드와 set-메서드를

통해서만 접근할 수 있다.

ProductInfo obj = new ProductInfo();

표준 액션에서는 자바 코드를 작성하지 않고,

자바빈 객체를 만들 수 있다.

<jsp:useBean id="obj" class="mall.ProductInfo"/>

obj.setName("텔레비젼");

<jsp:setProperty name="obj"

property="name"

value="오디오"/>

String name = obj.getName();

<jsp:getProperty name="obj" property="name" />

[자바빈 객체의 생존 범위]

forward 메서드를 통해 자바빈 객체를 전달하려면 scope 어트리뷰트의 값을 최소한 "request"로 설정해야 한다.

<jsp:useBean class="mall.PersonalInfo" id="pinfo"

scope="request" />

자바빈 객체의 생존 범위를 지정하는 애트리뷰트

page, request, session, application

※※※※ forward메서드를 통해 호출된 JSP 페이지 안에서 request 영역에 있는 자바빈 객체를 가져다가 사용하기 위해서는 어떻게 할까요?

<jsp:useBean> 표준액션을 사용하면 된다.

<jsp:useBean> 표준액션은 주어진 scope 영역에 id값에 해당하는 자바빈 객체가 없을 경우에만 새로운 객체를 생성하고, 그런 객체가 있으면 그 객체를 가져다가 사용한다.

[다형성 : polymorphism]

하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것.

즉, 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있는 것이 다형성이다.

class Tv {

boolean power; //전원의 상태

int channel; //채널

void power() { power = !power; }

void channelUp { ++channel; }

void channelDn { --channel; }

}

class CaptionTv extends Tv {

String text; //캡션내용을 보여주는 변수

void caption() { ...... }

}

Tv t = new Tv();

CationTv c = new CaptionTv();

------------------------------------------------------

Tv t = new CaptionTv();

CaptionTv c = Tv();

==========================================================================

[JSTL JSP Standard Tag Library]

JSP 페이지 안에서 사용할 수 있는 커스텀 액션과 함수를 제공한다.

[JSTL을 가지고 할 수 있는 일]

-간단한 프로그램 로직의 구사

(자바의 변수 선언, if문, for문 등에 해당하는 로직)

-다른 JSP페이지를 호출( <c:redirect>, <c:import>)

-날짜, 시간, 숫자 포맷

-JSP 페이지 하나를 가지고 여러 가지 언어의 웹 페이지 생성

-데이터베이스로의 입력, 수정, 삭제, 조회

-XML 문서의 처리

-문자열을 처리하는 함수 호출

문자열을 처리하는 함수 호출을 제외한 나머지 기능들은

모두 커스텀 액션 형태로 제공된다.

커스텀 액션이란 XML문법을 따르면서 특정한 동작을 수행하는

태그를 말한다.

<c:forEach begin="1" end="10">

<h2>안녕하세요! 여러분</h2>

</c:forEach>

<% for(int i = 1; i <= 10; i++) { %>

<h2>안녕하세요! 여러분</h2>

<% } %>

http://jakarta.apache.org/

JSTL 라이프러리를 구성하는 2개의 파일

-jstl.jar

-standard.jar

[코어 라이브러리]

일반 프로그래밍 언어에서 제공하는 변수선언, 조건분기,

반복수행등의 구사할 수 있고, 다른 JSP페이지를

호출할 수 있다.

<c:set> 커스텀 액션

변수를 선언하고 나서 그 변수에 초기값을 대입하는

기능의 커스텀 액션.

int num = 100;

<c:set var="num" values="100"/>

초기값이 선택적인 것이아니라 필수적이다.

<c:set> 커스텀 액션은 XML문법을 따라야 하므로

데이터가 수치값이더라도 반드시 따옴표로 묶어서

표시해야 한다.

이렇게 선언한 변수는 익스프레션 언어의 EL식 안에서

사용할 수 있다. 스크립팅 요소 안에서 사용할 수는

없다. <c:set>커스텀 액션을 이용해서 선언한 변수는

자바 변수가 되는 것이 아니라, page 데이터 영역의

어트리뷰트가 되기 때문이다.

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

${num} <= OK

<%=num%> <= Error

<% int num1 = 10, num2 = 20; %>

<c:set var="sum" value="<%=num1+num2%>" /> <= OK

<c:set> 액션을 이용해서 선언한 변수를 스크립팅 요소

안에서 사용하는 것은 불가능하지만,

스크립팅 요소 안에서 선언한 변수를 <c:set> 액션의

value 어트리뷰트에 사용하는 것은 가능하다.

<c:set> 액션을 이용하면 page 데이터 영역뿐만 아니라

request, session, application 데이터 영역에

어트리뷰트를 저장하는 것도 가능하다.

<c:set> 태그에 scope라는 어트리뷰트를 추가하고,

page, request, session, application 중 하나를

어트리뷰트 값으로 지정하면 된다.

<c:set var="PRICE" value="25000" scope="request" />

<c:remove> 커스텀 액션

<c:set> 커스텀 액션을 이용해서 선언한 변수를 삭제.

<c:set var="num" values="100"/>

<c:remove var="num" />

특정 데이터 영역에 있는 변수를 제거하기 위해서는

<c:remove> 커스텀 액션에 scope 어트립뷰트를 추가하고

그 값으로 page, request, response, application 중

하나를 지정하면 된다.

<c:set var="code" values="100" scope="page/>

<c:set var="code" values="100" scope="request/>

<c:set var="code" values="100" scope="response/>

<c:set var="code" values="100" scope="application/>

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

<c:if> 커스텀 액션

if(num1 > num2) {

System.out.println("num1이 더 큽니다.");

}

<c:if test="${num1 > num2}">

num1이 더 큽니다.

</c:if>

<c:if> 커스텀 액션의 조건식에 직접 쓸 수 있는 상수

true, false

<c:if test="true">

이 문장을 보여드립니다.

</c:if>

<c:choose> 커스텀 액션

자바의 switch 문과 유사하다.

switch (num) {

case 0:

System.out.println("프로그램을 종료합니다.");

break;

default:

System.out.println("다시 선택하십시오.");

break;

}

<c:choose>

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

프로그램을 종료합니다.

</c:when>

<c:otherwise>

다시 선택하십시오.

</c:otherwise>

</c:choose>

<c:forEach> 커스텀 액션

for(int cnt = 0; cnt < 10; cnt++) {

System.out.println("안녕하세요?");

}

<c:forEach begin="1" end="10">

안녕하세요?<br>

</c:forEach>

<c:forEach var="cnt" begin="1" end="10">

${cnt} 번 안녕하세요?<br>

</c:forEach>

<c:forEach var="cnt" begin="1" end="10" step="2">

${cnt} 번 안녕하세요?<br>

</c:forEach>

<c:forEach> 커스텀 액션을 이용하면 여러 개의 항목으로

구성된 데이터를 순서대로 출력할 수 있다.

액션 태그에 items 어트리뷰트를 사용하고, 그곳에 배열

이름을 지정하면 된다. 이때 var라는 이름의 어트리뷰트도

함께 써야하는데, 이 어트리뷰트의 값은 배열의 항목을

담는 변수 이름으로 사용된다.

<c:forEach var="str" items="${arr}">

${str}<br>

</c:forEach>

<c:forEach>액션의 items어트리뷰트를 이요해서 처리 할

수 있는 데이터

-배열

-java.util.Collection 객체

-java.util.iterator 객체

-java.util.Enumeration 객체

-java.util.Map 객체

-콤마로 구분된 항목들을 포함한 문자열

<c:forTokens> 커스텀 액션

자바의 for문과 java.util.StringTokenizer클래스의 기능을

합친 것과 같은 기능을 제공한다.

이 액션에는 items, var, delims 라는 3개의 어트리뷰트를

써야 한다.

<c:forTokens var="pet" items="강아지 고양이 타조"

delims=" ">

${pet}<br>

</c:forTokens>

<c:forTokens var="fruit" items="딸기\*키위/수박-참외"

delims="\*/-">

${fruit}<br>

</c:forTokens>

<c:catch> 커스텀 액션

<c:catcg>는 자바 프로그래밍 언어의 try문과 비슷한

역할을 한다.

<c:catcg> 커스텀 액션은 try문의 try블럭에 해당하는

일만 하기 때문에 catch 블럭에 해당하는 일은 별도로

코딩해야 한다.

try {

int result = num1 / num2;

System.out.println(result);

} catch (Exception e) {

Systme.out.println(e.getMessage());

}

<c:catch var="e"> //e : 익셉션 객체를 저장할 변수

<%int result = num1 / num2;%>

나눗셈의 결과는? <%=result%>

</c:catch>

<c:if test=${e != null}">

에러 메싲 : ${e.message}

</c:if>

<c:redirect> 커스텀 액션

<c:redirect url="www.daum.net" />

<c:redirect> 의 url에 변수와 값을 보낼 때

<c:redirect url="www.daum.net">

<c:param name="NUM1" value="10" />

<c:param name="NUM2" value="3" />

</c:redirect>

<c:import> 커스텀 액션

<jsp:include> 표준액션과 비슷한 일을한다.

즉,현재 JSP 페잊에 다른 JSP 페이지의 결과를 포함 시킨다.

한가지 다른 점이 있다.

이 액션을 이용하면 다른 웹 서버에 있는 JSP 페이지를

불러올 수 있고, JSP 페이지가 아닌 다른 종류의 웹

자원도 불러올 수 있다.

<c:import url="Multiply.jsp" />

<c:import url="Multiply.jsp" >

<c:param name="product" value="TV" />

<c:param name="price" value="2000000" />

</c:import>

<c:url> 커스텀 액션

<c:url> 커스텀 액션은 URL을 저장하기 위한 변수의

선언에 사용된다.

<c:url var="myUrl" value="Multiply.js" />

<c:param name="NUM1" value="10" />

<c:param name="NUM2" value="3" />

<c:url var="myUrl" value="Multiply.js" >

</c:url>

<c:out> 커스텀 액션

데이터를 출력할 때 사용.

웹 브라우저에 의해 특수 문자로 해석될 가능성이 있는

<, >, &, 문자를 포함하는 데이터는 이 액션을 사용하는

것이 좋다.

<c:out> 커스텀 액션을 이용해서 데이터를 출력할 때는

value어트리뷰트를 쓰고, 거기에 출력할 데이터를 지정

하면 된다.

< &lt;

> &gt;

& &amp;

' &#039;

" &#034;

[포매팅 라이브러리]

<fmt:>

<fmt:formatDate> 커스텀 액션

날짜와 시각을 포맷하는 커스텀 액션.

이 액션에는 출력할 날짜와 시각을 java.util.Date클래스

타입의 객체로 넘져줘야 하기 때문에, 먼저 이 크래스의

객체를 만들어야 한다.

Date date = new Date();

<fmt:formatDate value="${date}" /> => 2019.05.02

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

<fmt:formatDate value="${date}" type="both"/>

<fmt:formatNumber> 수치를 포맷

<fmt:formatNumber value="10000" />

<fmt:formatNumber value="10000" groupingUsed="true" />

세자리 수마다 쉼표를 하나씩 첨가한다.

<fmt:formatNumber value="3.14158" pattern="#.##" />

주어진 값을 소수점 아래 2자리까지 끊어서 출력한다.

<fmt:formatNumber value="3.1" pattern="#.00" />

<fmt:formatNumber value="0.5" type="percent" />

주어진 수치를 퍼센트 단위로 포맷하여 출력한다.

value 어트리뷰트에 주어진 값에 100을 곱하고 그 뒤에

%표시를 붙인 결과가 출력된다.

<fmt:formatNumber value="12345700" type="currency" />

number는 디폴트 값으로 일반 수치를 의미하고,

currency는 금액을 의미한다.

currency라는 값을 지정하면 수치가 금액에 적합한

포맷으로 만들어서 출력한다.

<fmt:formatNumber value="12345700" type="currency"

currencySymblo="￦" />

심볼이 표시되고 뒤에 value에 있는 값이 출력된다.

<fmt:setLocale> 커스텀 액션

<fmt:setLocale value="en" />

<fmt:setLocale value="us\_en" />

ISO 언어 코드와 ISO 국가 코드

https://docs.oracle.com/javase/8/docs/api/

시간대를 설정하는

<fmt:timeZone> 과 <fmt:setTimeZone> 커스텀 액션

시간대마다 달라지는 날짜와 시각을 자동으로 계산해서

표시하기 위해 필요한 커스텀 액션.

<fmt:timeZone>

value 어트리뷰트에 특정 시간대에 해당하는 지역이름을

쓰면 이 액션의 시작 태그와 끝 태그 사이에서 출력되는

날짜와 시각은 그 시간대에 맞게 표시된다.

이때 날짜와 시각은 <fmt:formatDate> 커스텀 액션을

사용해서 출력하면 된다.

<fmt:timeZone value="America/New\_York">

날짜:<fmt:formatDate value="${date}" type="date" />

시각:<fmt:formatDate value="${date}" type="time" />

</fmt:timeZone>

<fmt:setTimeZone> 은

이 액션이 실행된 다음의 모든 코드에 영향을 미친다.

<fmt:setTimeZone value="Europe/London" />

[다국어를 지원하는 <fmt:setBundle> 과 <fmt:setbundle>]

일반 텍스트도 여러가지 언어로 표현해야 할 경우가 있다.

언어에 따라 서로 다른 파일의 내용을 읽어서 사용하는

방식을 취하므로 그런 파일을 만들어야 된다.

그런 파일은 자바기술에서 사용하는 프로퍼티 파일형태로

만들어야 하는데,

property file이란 데이터를 '키=값' 형태로 표현해서

모아 놓은 텍스트 파일을 말한다.

모든 프로퍼티 파일 이름이 .properties라는 확장자를

갖도록 만들어야 한다.

언어만 다르고 같은 종류의 내용을 포함하는 여러 프로퍼티

파일이 있는 경우에는 모든 파일을 대표하는 이름을

만든다음에, 그 이름 뒤에 밑줄(\_)을 붙이고, 그 뒤에

각각의 파일에 해당하는 ISO언어 코드를 쓰고,

그 다음에 .properties라는 확장자를 붙여서 파일이름을

정해야 한다.

Intro\_ko.properties

Intro\_en.properties

<fmt:setBundle> 사용할 프로퍼티 파일을 지정하는 역할.

<fmt:setBundle basename="Intro" />

<fmt:message key="TITLE" />

[POST 메서드로 전송된 한글 입력 데이터를

받기 위해 필요한 커스텀 액션]

[함수 라이브러리]

JSTL의 함수 라이브러리는 익스프레션 언어의 EL식 안에서

사용할 수 있는 EL 함수들의 라이브러리이다.

이 함수들은 주로 문자열을 처리하는 일을 하며,

자바 JDK 라이브러리의 java.lnag.String 클래스에 속하는

메서드들과 비슷한 기능을 제공한다.

EL 함수들은 String 객체에 대해 호출하는 것이 아니라

처리해야 할 문자열을 파라미터로 넘겨주어야 한다.