CHAPTER 01. 웹 프로그래밍 개요

## 웹 프로그래밍이란 무엇인가?

* 월드 와이드 웹 기반에서 동작되는 프로그래밍 방식.
* 웹은 메뉴 방식으로 서비스되던 기존의 인터넷 서비스 대신 하이퍼텍스트를 기반으로 이루어진 것이다. 문서 활용에 엄청난 편리성을 제공한 것이다.

## 웹 프로그래밍 언어의 종류

* CGI (Common Gateway Interface) : 웹 페이지에 동적으로 변화하는 데이터를 처리하고 표시하기 위해서 개발되었다. 웹 서버와 외부 프로그램 사이에서 정보를 주고 받는 방법이나 규약들을 말한다.

Perl은 CGI의 단점을 개선한 것으로 스크립트 언어를 사용해서 페이지를 생성한다. 이것은 C/C++과 문법이 같으며 간단하고 편하게 사용할 수 있다. 하지만 PERL 또한 C/C++ 지식이 없는 사용자가 사용하기에는 쉽지 않다.

그러나 CGI는 UNIX 플랫폼에서 문자열 처리가 간단한 Perl, Windows 플랫폼에서는 비주얼 베이직 등이 사용되는 경우가 많다.

* ASP(Active Server Page) : 비주얼 베이직 언어를 기반으로 사용한다. 스크립트 방식으로 동적인 웹페이지를 작성할 수 있도록 지원하는 기술로, 서버에서 실행되는 스크립트 언어라 할 수 있다.

닷넷 기반의 ASP.NET이 있으나 우리나라에선 그다지 사용하지않고 국제표준또한 Ajax를 사용해서 처리하는 것이라 잘 쓰이지 않는 언어이다.

* PHP: C언어 기반으로 만들어진 PHP는 서버에서 실행되는 스크립트 언어로, 기존의 C언어에 익숙한 개발자들이 보다 쉽게 접근할 수 있다는 장점이 있다. PHP는 배우기 쉽고 개발 속도가 빠르다는 장점을 가지고 있어 한때 대대적인 환영을 받았다.

그러나 컴포넌트를 사용할 수 없을 뿐만 아니라 보안에 취약해 PHP 기반으로 만들어진 웹 사이트들은 해킹의 대상이 되고 있다는 단점이 있다.

* Servlet과 JSP: 서블릿은 자바를 통해 동적 웹 페이지를 작성할 수 있도록 지원한다. 서블릿은 멀티 쓰레딩에 의해 사용자 요구를 처리하고 가공하여 이에 대한 결과를 사용자에게 응답한다. Servlet 객체는 쓰레드가 여러 개 돌아가면서 사용하므로 서블릿의 메소드들은 반드시 멀티쓰레드에 대한 고려를 해야 한다.

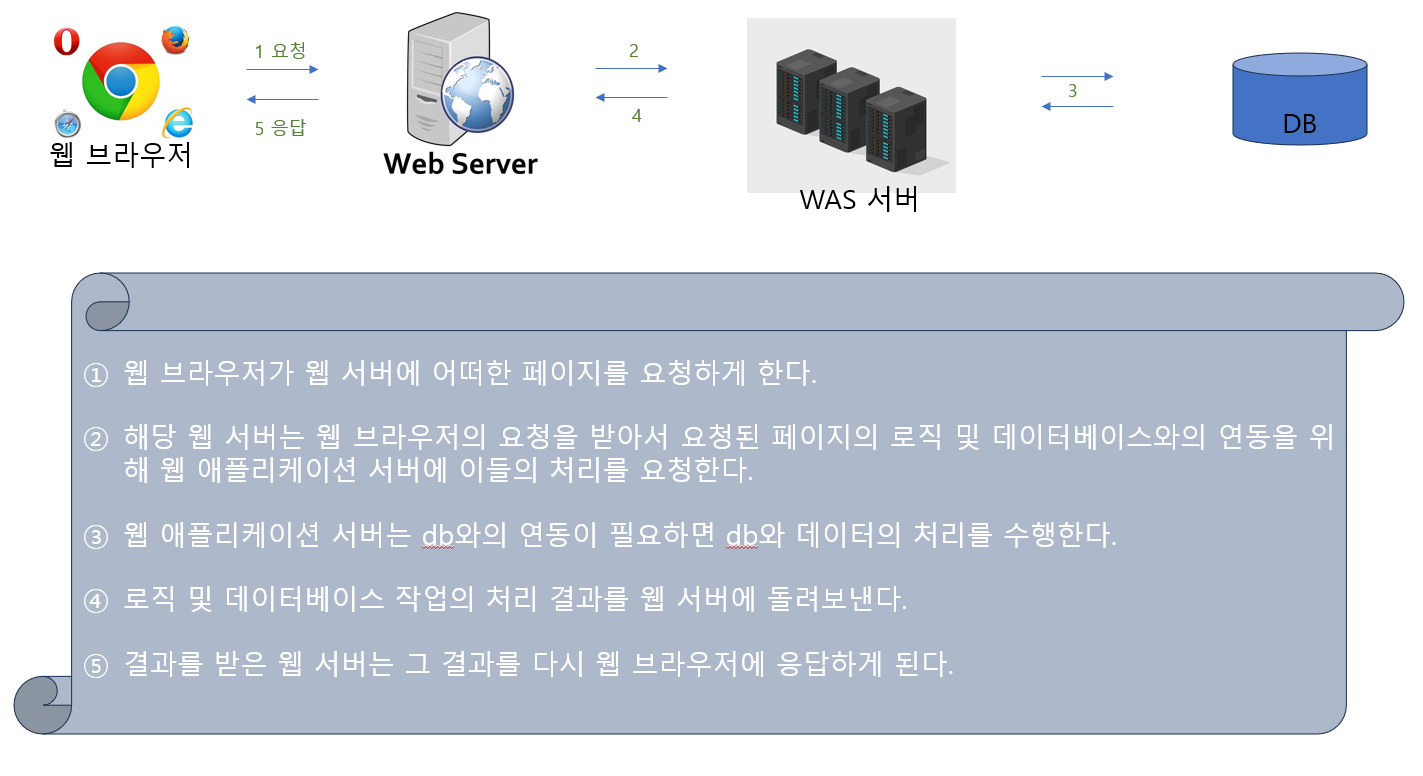
JSP는 서블릿과 마찬가지로 자바 기반의 웹 프로그래밍 언어이다. JSP는 서블릿에 비해 자바 코드에 덜 의존적이라 프로그래밍하기가 보다 쉽고 편하다.

JSP는 페이지의 요청이 있을 때 최초에 한 번 자바 코드로 변환된 후 Servlet 클래스로 컴파일된다. 결론적으로 JSP는 실행시 Servlet으로 변환된다. 단 한 번만 서블릿으로 변경되면 코드를 수정하기 전까지 재변환 작업이 일어나지 않으므로 수행 속도에서는 JSP나 서블릿 간에 별 차이가 없다.

ASP나 PHP와 같이 서버에서 실행되는 스크립트 언어 방식으로 동적인 웹 페이지를 작성하고 Servlet의 구조와 똑같고 그것과 마찬가지로 java 언어를 기반으로 하여 서블릿의 장점이 더해져 작성이 쉽다는 장점이 더해졌다.

JSTL의 완전히 지원되어 프로그램의 작성과 유지보수가 쉽다.

## 웹 프로그래밍과 웹 어플리케이션



## 웹 어플리케이션 처리 방식

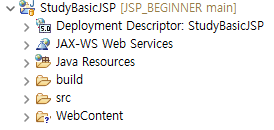
* CGI 방식: 웹 서버가 애플리케이션 프로그램을 직접 호출하는 구조를 가지고 있다. 이때 애플리케이션 프로그램으 ㅣ처리 방식은 프로세스를 생성하여 처리한다. 하나의 요청에 대해 한 개의 프로세스가 생성되어 그 요청을 처리한 뒤 종료한다.
* WAS 서버 방식: 웹 서버가 직접 애플리케이션 프로그램을 처리하는 것이 아니라, 웹 애플리케이션 서버에 처리를 넘겨주고 애플리케이션 서버가 애플리케이션 프로그램을 처리한다.

CHAPTER 02. JSP 개발 환경 설정

1. JDK 다운로드
2. 자바 환경변수 설정
3. 톰캣 다운로드
4. 톰캣 환경변수 설정
5. 동적 웹 프로젝트 작성: File – New – Project 메뉴에서 Web하위의 Dynamic Web Project 화면으로 진행 후 프로젝트 이름에 원하는 거 넣고 기본값 그대로 한 후 Next버튼을 클릭한다.

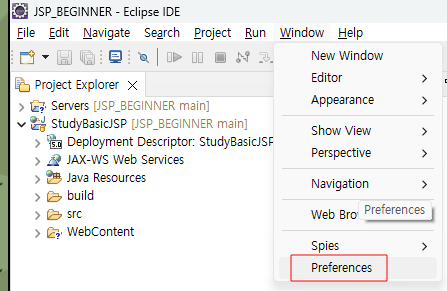
기본값 [build\classes]를 그대로 사용한 후 next를 누른다. 여기선 기본 소스 .java가 저장된 폴더 src에 대응되는 .class 파일이 저장되는 폴더를 지정한다.

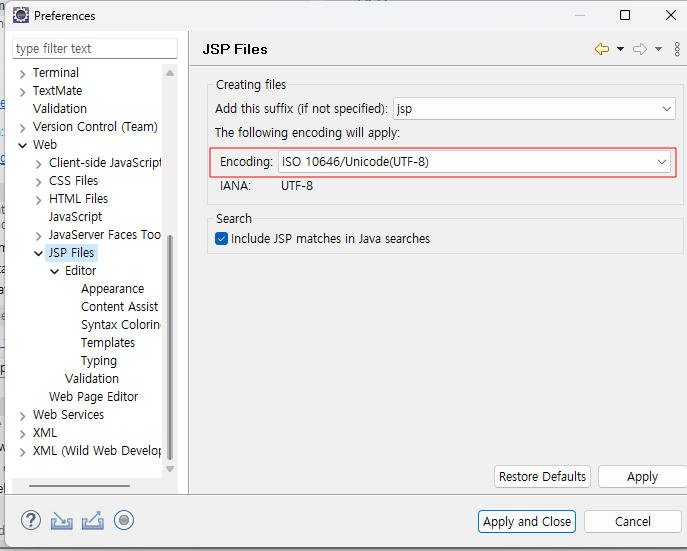
Context root에는 프로젝트 이름을 /\* 일반적으로 프로젝트의 디렉토리가 선택된다. \*/ Context directory로는 /\* 일반적으로 WebContent가 설정되어 있음 \*/ 기본값을 하거나 임의 지정한다.

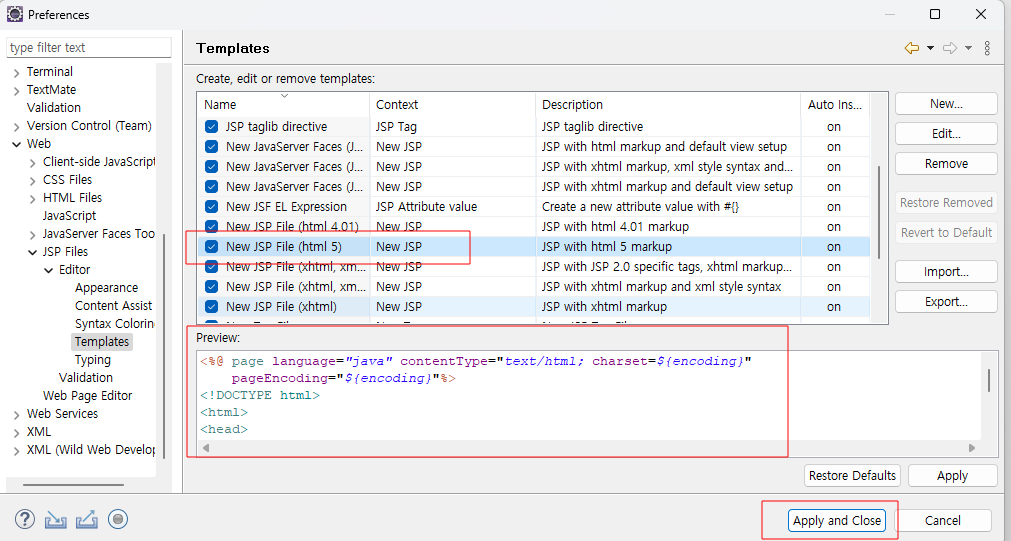


위 사진처럼 프로젝트가 생성된 것을 확인할 수 있다.

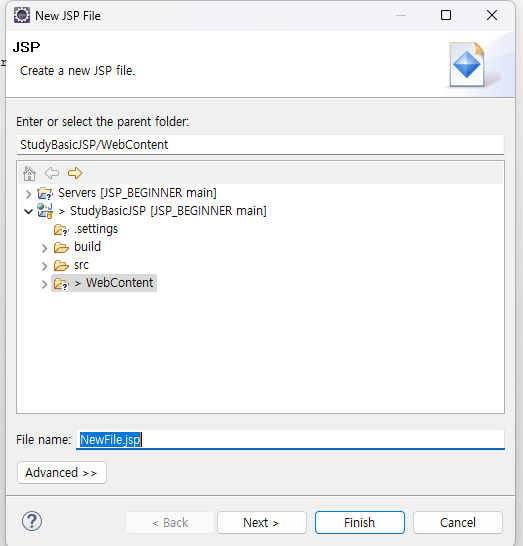
1. 동적 웹 프로젝트를 서버에 추가
2. JSP 페이지 세팅



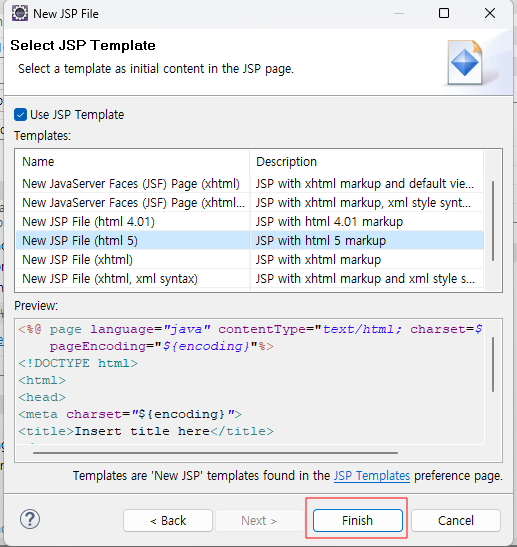




1. JSP 페이지 작성



index.jsp 파일 생성을 위해 WebContent 폴더 우클한 후 New – JSP File 에서 이름 “index.jsp”지정해준다. 그리고 Next에서 New JSP File(html5)를 클릭 후 끝낸다.



CHAPTER 03. JSP 프로그래밍 개요

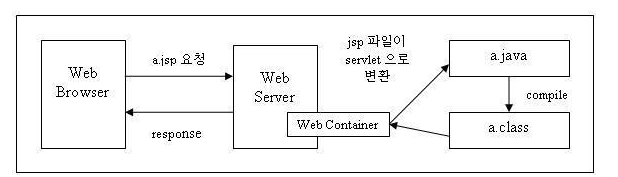
## JSP 페이지 개요

* JSP에서는 로직 관련 코드는 스크립트인 <% %>에 주로 기술한다.
* HTML은 웹 브라우저에 기술했던 내용이 출력되지만 서버사이드 스크립트인 JSP에선 화면에 표시되지 않고 실행 결과만 표시된다.

## JSP 페이지의 처리 과정

* 웹 브라우저에서 JSP 페이지를 웹 서버로 요청하면 웹 서버는 JSP에 대한 요청을 웹 컨테이너로 넘기게 된다. 이러한 요청을 받은 웹 컨테이너는 해당 JSP 페이지를 찾아서 서블릿(.JAVA 파일 생성)으로 변환하는 파싱의 과정을 거친 후 컴파일(.CLASS 파일 생성)한다. 컴파일된 서블릿(.CLASS)은 최종적으로 웹 브라우저에 응답하여 사용자는 응답 결과를 보게 된다.

이러한 과정은 해당 JSP 페이지가 최초로 요청되었을 때 단 한번만 실행되고 이후 같은 페이지에 대한 요청이 있으면 변환된 서블릿 파일로 서비스를 처리한다.



1. **a.jsp 요청**

: 사용자의 웹 브라우저에서 <http://serverURL/xxx.jsp>와 같은 형태로 해당 페이지를 요청한다.

1. **웹 서버 -> 웹 컨테이너**

: 웹 서버는 요청한 해당 페이지를 처리하기 위해서 JSP 컨테이너에 페이지의 처리를 넘긴다.

1. **Jsp 파일이 servlet으로 변환**

: 해당 JSP 페이지가 처음 요청된 것이면 jsp 페이지를 서블릿으로 파싱(변환)한다. 이전에 요청되었던 페이지일 경우 다시 파싱할 필요가 없으므로 바로 (a.class -> 웹 컨테이너) 단계로 넘어간다.

1. **a.java -> a.class**

서블릿 파일은 자바에서 실행가능한 상태인 클래스 파일로 컴파일된다.

1. **a.class -> 웹 컨테이너**

클래스 파일은 메모리에 적제되어 실행한다.

1. **웹 컨테이너 -> 웹 서버**

이 실행결과는 다시 웹 서버에 넘겨진다.

1. **웹 서버 -> 웹 브라우저**

: 웹 서버는 웹 브라우저가 인식할 수 있는 html 형태로 결과를 브라우저에 응답한다. 웹 서버로부터 응답 받은 결과물인 html 페이지를 웹 브라우저에서 실행시켜 해당 페이지가 웹 브라우저에 표시된다.

JSP 파일은 실행을 위해 서블릿으로 파싱되고 클래스 파일로 컴파일되는데 이러한 과정은 JSP 파일이 처음으로 호출되었을 때 거치게 된다. 만일 이전에 어떤 JSP 파일이 호출된 적이 있었다면, 이후로 들어오는 해당 JSP 파일의 요청에 대해서 3단계, 4단계의 과정을 거치지 않는다. 그러나 이미 요청되었던 JSP 페이지의 내용이 변경된 경우에는 다시 서블릿으로 변환되고 컴파일되는 과정을 다시 거쳐서 웹 브라우저의 요청에 응답한다,.

웹 브라우저는 HTML 태그로 구성된 페이지를 실행시켜주는 프로그램으로 웹 서버에서 HTML 페이지가 실행되는 것이 아니라 웹 브라우ㅈ에서 HTML 태그들이 실행되어 보여지는 것이다.

JSP 페이지는 서블릿으로 변환되어 웹 브라우저의 요청에 대한 응답을 HTML 문서로 생사한다.

CHAPTER 04. JSP 페이지의 디렉티브

## JSP Page 디렉티브(Directive) - <%@ page%>

JSP 페이지에 대한 정보는 page 디렉티브의 속성들을 사용해서 정의한다. 즉, 생성되는 문서의 타입, 스크립팅 언어, import할 클래스, 세션 및 버퍼의 사용 여부, 버퍼의 크기 등 JSP 페이지에서 필요한 설정 정보를 지정한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 속성 | 속성의 기본값 | 사용법 | 속성 설명 |
| **info** |  | info="설명..." | 페이지를 설명해주는 문자열을 지정하는 속성 |
| **language** | "java" | language="java" | JSP 페이지의 스크립트 요소에서 사용할 언어를 지정하는 속성 |
| **contentType** | "text/html;  charset=ISO-8859-1" | contentType="text/html;charset=utf-8" | JSP 페이지가 생성할 문서의 타입을 지정하는 속성 |
| **extentd** |  | extends="system.MasterClass" | 자신이 상속받을 클래스를 지정할 때 사용하는 속성 |
| **import** |  | import="java.util.Vector" import="java.util." | 다른 패키지에 있는 클래스를 가져다 쓸 때 사용하는 속성 |
| **session** | "true" | session="true" | HttpSession을 사용할지의 여부를 지정하는 속성 |
| **buffer** | "8kb" | buffer="10kb" buffer="none" | JSP 페이지 출력 버퍼의 크기를 지정하는 속성 |
| **autoFlush** | "true" | autoFlush="false" | 출력 버퍼가 다 찰 경우에 저장되어 있는 내용의 처리를 설정하는 속성 |
| **isThreadSafe** | "true" | autoFlush="false" isThreadSafe="true" | 현재 페이지에 다중 쓰레드를 허용할지의 여부를 설정하는 속성 |
| **isErrorPage** | "false" | isErrorPage="false" | 해당 페이지를 에러 페이지로 지정하는 속성 |
| **pageEncoding** | "ISO-8859-1" | pageEncoding="utf-8" | 해당 페이지의 문자 인코딩을 지정하는 속성 |
| **isELIgnored** | jsp 버전 및 설정에 따라 다름 | isELIgnored="true" | 표현 언어(EL)에 대한 지원 여부를 설정하는 속성 |

### info 속성

해당 JSP에 대한 설명을 기술하는 속성으로 속성값의 내용이나 문자열의 길이 제한은 없다. 이 속성은 설정하지 않아도 해당 JSP 페이지의 처리 내용에는 아무런 영향을 미치지 않는다.

Project Explorer 
Servers 
v Study3asicJSP 
Deployment Descriptor: Study3asicJSP 
JAX-WS web services 
Java Resources 
build 
v src 
v main 
java 
v webapp 
v chou 
pageDirectivelnfoJsp 
META-INF 
WEB-INF 
index.iso 
index_jsp pageDirectivelnfoJsp 
1 page 
3 page 
4 html) 
6' (head) 
7 (meta 
8 'titlepage 
g q' head) 
Ice <bodp 
page 
info 
title) 
info 
e./h2> 
14 
get-Servletlnfo page info 210101Zf. 
q' body) 
q' html 

C G) localhost8080/StudyBasicJSP/ch04/pageDirectivelnfo.jsp 
4 Old-I Google 
PREQUEL- VULNERAE 
YouTube 
- info 
page 
starter of backdoohyun Sl developer 

### langeage 속성

JSP 페이지의 스크립트에서 사용할 프로그램언어를 지정하는 속성, 이 속성은 생략해도 되며 생략 시 기본값으로 JAVA가 지정됨.



### contentType 속성

JSP 페이지의 내용이 어떤 타입의 문서로 생성되는지를 지정하는 속성이다. 사용자 요청에 대한 응답 결과가 어떤 형태로 웹 브라우저에 출력될지를 MIME 타입으로 지정하는 속성이다.

지정할 속성값으로는 text/html, text/plain, text/xml 등 여러 가지 형태의 문서를 생성할 수 있으며 기본값은 text/html이다.

응답 결과를 보여줄 때 사용할 문자의 인코딩을 지정할 수 있는데 이를 charset을 사용한다.



♨ <@ page contentType="text/html;charset=utf-8"%>

웹 브라우저에 표시되는 한글이 깨지지 않게 하려면 UTF-8로 하는 것이 좋다.

### extends 속성

해당 JSP 페이지가 상속받을 클래스를 지정하는 속성이다.

JSP 페이지가 서블릿으로 변환(파싱)되는 과정에서 상속받을 클래스를 지정할 때 사용된다.

*사실 이 작업은 필요 없는 작업이다.*

*이유는 JSP 컨테이너가 알아서 적절한 클래스들을 상속시켜 변환해 주기 때문이다.*

♨ <%@ page extends="com.samyangm.ClassDef"%>

♨ <%--com.samyangm.ClassDef 클래스를 상속받겠다는 의미이다. --%>

### import 속성

다른 패키지에 있는 클래스를 가져다 쓸 때 사용되는 속성으로 자바의 import 문과 같다.

import 속성은 page 디렉티브 중에 유일하게 한 페이지 내에 여러 번 기술이 가능한 속성이다.

♨ <%@ page import="java.util.\*, java.sql.\*"%>

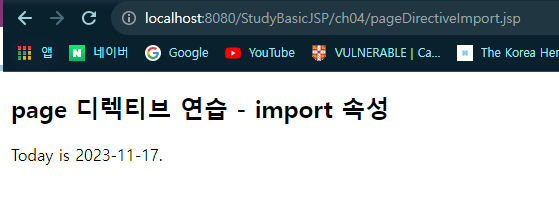
♨ <%@ page import="java.io.\*" %>

♨ <%-- 은 JSP 주석이다. --%>

♨ <%-- 여러 개의 패키지를 쉼표로 구분해서 사용할 수 있다. --%>

♨ <%-- page 디렉티브의 속성 중에서 여러 번 기술해서 사용할 수 있다. --%>





### Session 속성

▶ 해당 JSP 페이지가 HttpSession을 사용할 지 여부를 지정하느 속성이다.

▶ 속성의 값이 "true"일 경우 현재의 JSP 페이지가 세션을 사용하는 것으로 세션을 유지한다.

▶ 속성의 값이 "false"일 경우 세션을 사용하지 못한다.

▶ 기본값은 "true"로 session 속성의 속성값을 false로 지정할 수 있다.

♨ <%@ page session=”false” %>

### Buffer 속성

▶ JSP 페이지의 출력 버퍼의 크기를 지정하는 속성이다.

▶ 기본 값은 "8KB"이다.

▶ buffer 속성의 값을 "none"으로 지정한 경우 출력 버퍼를 사용하지 않는다는 의미로 이때는

JSP 페이지가 출력 버퍼를 거치지 않고 바로 웹 브라우저에 출력된다.

▶ 기본값으로 "8KB"를 사용하는 이유는 프로그래머들의 오랜 경험 끝에 JSP 페이지에서 가장

타당한 크기로 인식되었기 때문이다. 거의 대부분 8KB로 충분하다.

♨ <%@ page buffer=”10kb”%>

♨ <%@ page buffer=”none”%>

### autoFlush 속성

▶ JSP 페이지의 내용들이 웹 브라우저에 출력되기 전에 출력 버퍼가 다 찰 경우 저장되어 있는

내용들을 어떻게 처리할지를 지정하는 속성이다.

▶ autoFlush 속성의 속성값을 "true"로 설정하면 버퍼가 다 찼을 경우 자동적으로 버퍼의 내용이

웹 브라우저에 출력되고 출력 버퍼는 비워진다.

▶ autoFlush 속성의 기본값은 "true"이다. buffer 속성의 값을 "none"으로 지정한 경우 autoFlush

속성값은 "false"로 지정할 수 없다. 버퍼를 사용하지 않으므로 출력할 내용을 바로바로 화면

에 표시해야 하기 때문이다.

♨ <%@ page autoflush=”false” %>

### isThreadSafe 속성

▶ JSP 페이지에서 다중 쓰레드를 사용할 수 있는지를 지정하는 속성이다.

▶ 기본 값은 "true"이다.

▶ JSP 페이지에서는 하나의 요청을 하나의 쓰레드로 처리하기 때문에 한 페이지에서 여러

사용자 요청을 동시에 받아들 수 있게 되어 있다.

▶ isThreadSafe 속성이 "true"인 경우 해당 JSP 페이지가 여러 사용자 요청을 동시에 받아들

일 수 있다는 뜻이다.

▶ isThreadSafe 속성이 "false"인 경우 다수 사용자의 요청을 동시에 처리하지 않고 요청한

순서대로 처리하므로 웹 브라우저의 요청을 처리하는 데 많은 시간이 걸린다.

이 속성은 "쓰레드의 프로세스화"라고 불리는 것으로 JSP의 장점인 다중 쓰레드를 사용

할 수 없어서 처리속도가 떨어지므로 가급적이면 사용 안 하는 것이 좋다.

♨ <%@ page isThreadSafe = “false”%>

### errorPage 속성

▶ JSP 페이지를 처리하는 도중에 해당 페이지에서 예외가 발생할 경우 예외를 처리할 페이지를

지정하는 속성이다.

▶ 해당 페이지에서 예외를 처리하지않고 errorPage 속성값으로 지정한 다른 페이지에서 예외를

처리한다.

♨ <%@ page errorPage=”errorPage1.jsp” %>

: 현재 페이지에서 예외가 발생하면 에러를 처리하는 페이지인 errorPage1에서 처리

♨ JSP 2.0부터는 위와 같은 방법으로 처리하지 않는다. 왜냐하면 Tomcat 5.5.9 이상이면

위 속성을 인식하지 못한다.

♨ JSP 2.0 버전에서의 예외처리 권고안은 web.xml 파일에서 <error-page> 태그를 사용

하여 처리하도록 되어있다.

### isErrorPage 속성

▶ 현재의 JSP 페이지가 일반적인 페이지인지 예외를 처리하는 페이지인지를 지정할 때 사용되는

속성이다.

▶ 요청된 현재 페이지가 발생한 예외를 처리하는 페이지이면 isErrorPage 속성의 속성값을 "true"로 지정한다.

▶ 기본적으로 일반적인 JSP 페이지는 예외를 처리하는 페이지가 아니므로 isErrorPage 속성의

기본값은 "false"이다.

♨ <%@ page isErrorPage=”true” %>

♨ errorPage 속성에서 예외를 처리할 것으로 명시된 페이지에서 기술하는 속성으로

현재는 별로 사용되지 않는다.

### pageEncoding 속성

▶ JSP 페이지에서 사용하는 문자의 인코딩을 지정할 때 사용된다.

▶ 생략 시 기본값으로 ISO-8859-1을 사용하고 한글을 처리할 때는 UTF-8을 사용한다.

♨ <%@ page pageEncoding=”utf-8”%>

♨ <%@ page contentType=”text/html” pageEncoding=”utf-8” %>

: pageEncoding 속성값은 contentType 속성값의 charset과 같다.

: <%@ page contentType=”text/html; charset=utf-8”%>

: 위는 두 번째 예시와 같이 contentType 속성의 값을 지정해 준 것과 같은 결과를

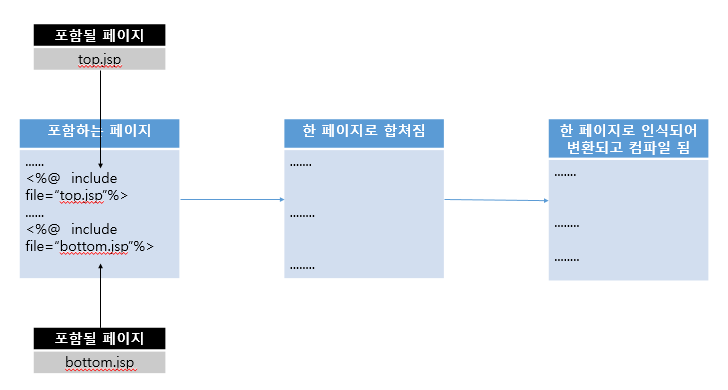
갖는다.

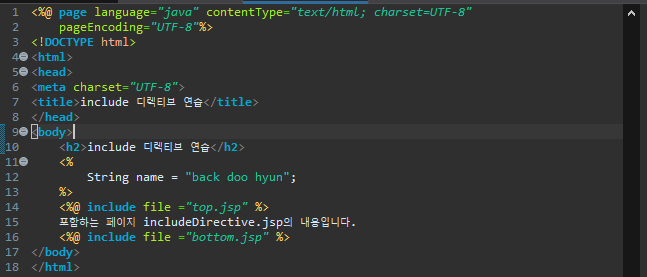
## JSP include 디렉티브 - <%@ include %>

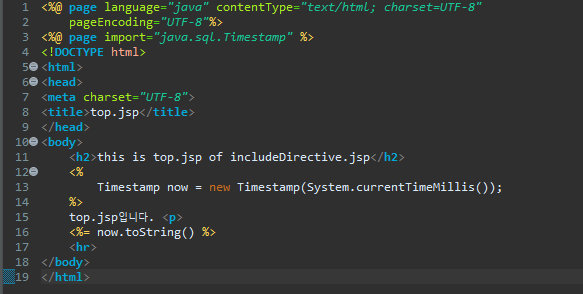
JSP 페이지에서는 여러 JSP 페이지에서 공통적으로 사용되는 내용이 있을 때 이러한 내용을 별도의 파일로 저장해두었다가 필요한 JSP 페이지 내에 삽입할 수 있는 기능을 제공한다. 이때 공통적으로 포함될 내용을 가진 파일을 해당 JSP 페이지 내에 삽입하는 기능을 제공하는 것이 include 디렉티브이다.

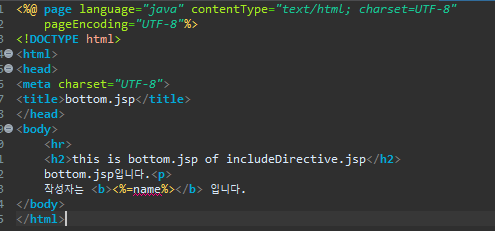
Include 디렉티브는 단순히 포함될 파일의 내용을 복사해서 붙여넣기하는 방식으로 해당 페이지에 가져오므로 처리방식이 정적이라고 할 수 있다. 따라서 include 디렉티브를 사용한 JSP 페이지가 컴파일 되는 과정에서 include되는 JSP 페이지의 소스 내용을 그대로 포함하여 컴파일하게 된다. 즉, 복사 & 붙여넣기 방식으로 두 개의 파일이 합쳐진 후 하나의 파일로서 변환되고 컴파일된다.

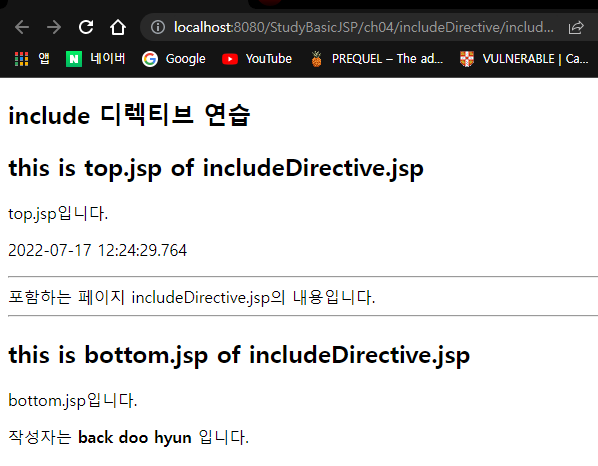
Include 디렉티브느 주로 조각코드를 삽입할 때 사용된다.











## JSP tablib 디렉티브 <%@ taglib %>

표현 언어(EL), JSTL, 커스텀 태그를 JSP 페이지 내에 사용할 때 마다 필요하다.

사용방법은 아래와 같이 두 개의 속성인 prefix 속성과 uri 속성의 값을 지정해 주어야 한다.

사용자가 정의한 어떤 태그의 설정정보는 uri 속성의 값이 가지고 있고 이것을 해당 페이지 내에서 사용할 때 uri 속성의 값이 복잡하므로 prefix 속성의 값이 별명과 같은 역할을 한다. 즉, prefix 속성의 값을 사용하면 uri 속성의 값을 사용하는 것과 같다.

♨ <%@ taglib prefix=”c” uri=<http://.....> %>

♨ ---------- 중 략 -------------

♨ <c:set=”aInt” value=”123” %>

## JSP 페이지의 스크립트 요소

* 선언문 (Declaration) -- <%! %>

전역 변수 선언 및 메소드 선언에 사용

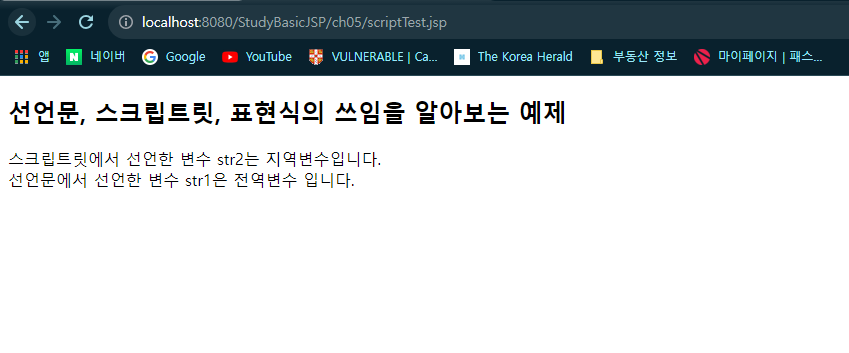
* 스크립트릿 (Scriptlet) -- <% %>

프로그래밍 코드 기술에 사용

* 표현식 (Expression) -- <%= %>

화면에 출력할 내용 기술에 사용





### 선언문 <%! %>

JSP 페이지 내에서 필요한 멤버 변수나 메소드가 필요할 때 선언해서 쓰기위한 요소

* 선언문과 스크립트릿의 차이점

: 선언문에서 선언된 변수는 전역변수 역할을 하는 멤버 변수가 된다. 또한 선언문에서 선언된 메소드는 일반 메소드들과 똑같은 역할을 한다.

: 스크립트릿에서 선언된 변수는 지역변수이다.

### 스크립트릿 <% %>

JSP 페이지에서 가장 일반적으로 쓰이는 스크립트 요소로 주로 프로그래밍 로직을 기술할 때 많이 쓰인다. 스크립트릿은 JSP 페이지가 서블릿으로 변환되고 이 페이지가 호출될 때 \_jspService 메소드 안에 선언이 된다. 스크립트릿에서 선언된 변수는 지역 변수로 선언된다.

### 표현식

JSP 페이지에서 웹 브라우저에 출력할 부분을 표현하기 위한 것이다.

표현식은 웹 브라우저에 출력을 목적으로 하는 변수의 값 및 메소드의 결과 값도 출력할 수 있다. 다만 스크립트릿 안에 표현식을 쓸 수 없다.

### HTML 주석 <!-- 🡪

### JSP 주석 <@-- 🡪

## JSP 페이지 조건문

Prgect Explorer x 
v StudyBasicJSP 
Deployment Descriptor. Study8asicJSP 
JAX-WS web services 
v Java Resources 
src/main/java 
> Libraries 
build 
main 
java 
webapp 
v chD6 
ifTestForm.j 
ifTestProJsp 
META-INF 
WEEnNF 
ifTestFormjsp x ifTestPro.jsp 
I page language— "java" contentType— "text/html; 
page Encodi ng— "LITF -8 
3 html> 
50 
6 <meta charset= 
8 < / head > 
ge < body > 
charset-UTF-8" 
120 
16 
(form method: "post" 
<input name: 
<input value:" 
< / form> 
< 'body > 
< 'html > 

Proect Explorer x 
Servers 
studyBasicJSP 
Deployment Descriptor StudyBasicJSP 
JAX-WS web services 
v Java Resources 
src/main/java 
Libraries 
build 
src 
main 
java 
v webapp 
ch06 
ifTestForm.jsp 
ifTestPro.j+ 
META-INF 
WEB-INF 
ifTestForm.jsp 
ifTestProJsp x 
1 page language= "java" 
page Encoding= "WF •8 "X) 
3 html> 
4. < html > 
50 (head> 
6 (meta charset= 
7 (title 
8 a/head> 
go < body > 
lezzy 
charset=UTF-8" 
12. 
22 
23 
String strNumber = 
request. getParameter( "number " ) ; 
int number — Integer. parseInt(strNumber) ; 
if(number lø) { 
"—number 
< / body > 
< html > 

## 톰캣 기반에서의 한글 처리

#### 서버에서 웹 브라우저에 응답되는 페이지의 화면 출력 시 한글 처리

<%@ page contentType=”text/html;charset=utf-8” %>

#### 웹 브라우저에서 서버로 넘어오는 파라미터 값에 한글이 있는 경우 (Post 방식)

<% request.setCharacterEncoding(“utf-8”); %>

#### 웹 브라우저에서 서버로 넘어오는 파라미터 값에 한글이 있는 경우 (Get 방식)

이 경우에 한글을 깨지지 않게 하려면 두 곳에 위치한 server.xml 파일의 한글 인코딩을 지정해야 한다. 하나는 실제로 서비스하는 환경인 톰캣홈/conf 폴더에 있는 server.xml 파일이고 다른 하나는 이클립스의 경우 [Project Explorer] 뷰의 [Servers] – [Tomcat v?.? Server ~~] 하위에 있는 server.xml파일이다.

해당 server.xml의 Design과 Source 탭에서 Source탭을 선택한다.

<Connector connectionTimeout=”???” port=”???” protocol=”???” redirectPort=”??” **URIEncoding=”EUC-KR”**> 로 수정하여 톰캣을 다시 지정한다.

실제 서비스하는 환경인 톰캣홈\conf의 server.xml에서도 똑같이 URIEncoding을 지정해주면 된다.