

JSP

웹 프로그래밍이란?

- 웹 프로그래밍이란?

=> 인터넷 쇼핑 -> 브라우저에 주소 입력 -> 브라우저에 요청함

➔요청을 받아 웹 페이지 찾아 주는 것 : 웹 서버

➔요청된 페이지를 받아보는 브라우저 : 클라이언트

1. 사이트 주소 입력
2. 웹 페이지 요청
3. 웹 서버는 해당 웹 페이지를 찾는다.
4. 찾은 웹 페이지를 보낸다.

웹 프로그래밍이란?

- Html만으로는 시시각각 변경되는 새로운 정보를 제공해주지 못함
왜? 정적인 페이지이기 때문

인터넷은 바로바로 새로운 내용을 제공해주어야 함 -> html만 가지고는 웹 프로그래밍하는데 한계가 있음 => 그래서 동적인 페이지가 등장함

동적인 페이지에서 새로운 정보를 제공해주기 위해서는 방대한 정보를 관리할 데이터베이스가 필요!!!!

Ex) 게시판 -> 디비에 저장되었다가 보여주는 것임

다양한 정보를 데이터베이스에서 얻거나 저장하기 위해 등장한 언어
=> php, asp, 서블릿/jsp

웹 프로그래밍이란?

서버는

1. 사용자의 요청이 들어오면 이에 대한 처리를 한 결과 페이지를 전송하는 웹 서버(Web Server)
2. 실질적으로 요청한 페이지의 로직이나 데이터베이스와의 연동을 처리할 수 있는 비즈니스 로직이 구현되어야 하는 웹 애플리케이션 서버 (Web Application Server : WAS) 로 이루어져 있다.

대표적인 WAS -> Tomcat, Jaus, WebLogic...

Tomcat은 웹 서버 기능이 내장되어 있어 별도로 웹 서버 설치안해도 WAS역할을 한다.

웹 프로그래밍이란?

1. 회원 가입 페이지에서 회원정보 입력한 후 submit버튼 클릭
2. 웹 서버에 입력된 회원 정보 전송됨
3. 톰캣으로 입력된 회원정보를 읽어와 데이터베이스에 저장
4. 회원가입 성공 실패여부를 결과값으로 얻어옴
5. 톰캣으로 회원가입 처리 결과 전송(웹 서버에)
6. 웹서버는 입력된 정보를 페이지에서 확인하기 위해 출력해준다.

SERVLET

- Servlet: Server + Applet 의 합성어
 - 서버에서 실행되는 Applet으로 자바를 이용하여 웹에서 실행되는 프로그램을 작성 (자바 클래스 형태의 웹 애플리케이션)
 - 서블릿은 JVM에서 동작해야 하므로 클래스 파일이 생성되어야 한다
->클래스 형태로 작성
- (HttpServlet 클래스를 상속받아 구현해야함)

HTTPSERVLET

- 톰캣을 설치하면 제공되는 클래스
- 웹 애플리케이션으로 동작하는 기본동작

<클라이언트가 서버에 요청하는 두 가지 방식>

Get : 주소창을 타고 넘어가서 서버로 보내는 데이터가 주소창에 노출되기 때문에 보안에 취약함 / 255자 이하의 적은 양의 데이터를 전송한다

Post : header를 타고 넘어가 보안에 강하며 255자 이상의 대용량의 데이터를 전송한다

```

package unit01;

import java.io.IOException; //doGet 메서드의 throws절에서 IOException 사용했기 때문 (입출력시 예외처리)
import java.io.PrintWriter; //클라이언트에 결과를 출력하기 위한 out객체를 PrintWriter클래스로 선언했기 때문

import javax.servlet.ServletException; //서블릿에서 발생하는 예외 처리를 위한 클래스 doGet에서 ServletException사용
import javax.servlet.annotation.WebServlet; //@WebServlet사용
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Add
 */
@WebServlet("/Add") //직접클래스를 요청하는 것이 아니고 url로 요청한다
public class Add extends HttpServlet { //HttpServlet에는 요청에 대한 응답이 가능하도록 하는 내용이 있기때문에 상속받아야함
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Add() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    //서블릿이 요청받으면 이벤트 처리 방식으로 자동으로 호출되는 메서드 -> 어떤 처리를 해야하는지 명시(오버라이딩)
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int n1=20;
        int n2=10;
        int sum=n1+n2;
        PrintWriter out=response.getWriter();
        out.println("<html><head><title>Add</title></head></html>");
        out.println("<body>");
        out.println(n1+"+"+n2+"="+sum);
        out.println("</body>");
        out.println("</html>");
    }
}

```



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

```
<!-- 사용하는 언어가 자바이며, 이 페이지는 html문서이며 한글인코딩을 utf-8로 처리하겠다 -->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
<%
```

```
    int n1=20;
```

```
    int n2=10;
```



```
    int sum=n1+n2;
```

```
%>
```

```
<%=n1 %>+<%=n2 %>=<%=sum %>
```

```
</body>
```

```
</html>
```

-   localhost:8080/Test/hello

JSP(JAVA SERVER PAGE)

- 자바로 서버 페이지를 작성하기 위한 언어
 - html과 jsp태그로 구성되어 화면을 작성하는데 유리함
 - 웹 서버에서 실행되는 페이지로 요청에 필요한 페이지를 위한 로직, 데이터베이스와의 연동을 위해 필요한 것들을 포함함
-
- 웹 프로그래밍 언어
 - jsp/서블릿 : 자바 언어를 기반으로 웹에서 동작하는 프로그래밍 기술
 - 서블릿이 먼저 나온 기술
 - 서블릿으로는 웹 문서 만들기가 어려워서 jsp는 나중에 쉽게 웹문서를 만들도록 만들어진 기술
 - 동적인 HTML을 생성
 - 사용자 요구에 따른 HTML을 프로그램에 의해서 생성
 - 동적인 HTML 가능 종류 : jsp, asp, php, python(flask, django...)

JSP(JAVA SERVER PAGE)

-서블릿과 jsp차이

디자이너가 자바를 이해하지 못한 채 html코드를 자바 형태인 서블릿에서 작성한다는 것은 불가능

이러한 문제점 때문에 jsp등장함

jsp는 html문서에 자바 코드가 들어가는 구조 -> 서블릿 보다 jsp개발이 쉽고 간단하다

서블릿 컨테이너는 jsp를 서블릿으로 변환한 후에 동작시킨다.

mvc패턴으로 -> 컨트롤러로 서블릿이 사용된다

스크립트 요소의 이해

- 스크립트 요소란?
 - JSP 프로그래밍에서 사용되는 문법의 표현 형태
- 스크립트 요소
 - 선언문(Declaration) <%! %>
 - 스크립트릿 (Scriptlet) <% %>
 - 표현식 (Expression)
 - 주석 (Comment)

멤버 변수의 선언

- ❖ 선언문에서 선언된 변수는 JSP 페이지가 서블릿 코드로 변환이 되면 서블릿 클래스의 멤버변수로 변환됨

```
<%!  
    String name = "JSPStudy";  
    int year = 2013;  
%>
```

메소드 선언

- 선언문에서 선언된 메소드는 JSP페이지 내에서는 일반적인 형태의 메소드로 선언됨.

```
<%!  
    String name = "Korea";  
    public String getName(){  
        return name;  
    }  
%>
```

- getName() 메소드는 멤버 변수 name값을 리턴 시켜주는 메소드로 선언.
- name변수가 선언문에서 선언이 되었기 때문에 멤버변수의 역할이 되면서 접근이 가능한 것.

스크립트릿

- 스크립트릿이란? <% %>
- JSP 페이지가 서블릿으로 변환되고 요청될 때 _jspService(Tomcat 기준으로 설명) 메소드 안에 선언이 되는 요소.
- 스크립트릿은 선언문과 달리 선언된 변수는 지역 변수로 선언이 되고 메소드 선언은 할 수 없음.
- 만약 선언을 하게 되면 메소드 안에 메소드를 선언한 것이기 때문에 만들 수가 없음.

```
<%
```

```
    이곳에 필요한 자바코드를 삽입합니다.(지역 변수 선언, for, while, if 등...)
```

```
%>
```


표현식

- 동적인 JSP 페이지를 브라우저로 표현을 하기 위한 요소.
- 변수를 출력하거나 메소드의 결과값을 브라우저에 출력 할 수 있음.
- 스크립트릿 코드 내에서 out이라는 내장객체를 통해 브라우저에 출력 가능.
- 스크립트릿과 달리 변수나 메소드를 출력하고 할 때 세미콜론(;)은 표기하지 않음.

(서블릿 코드로 변환될 때 자동적으로 세미콜론은 붙여짐)

⟨%=변수 혹은 메소드%⟩

주석

- 주석이란?
 - 프로그램에 직접적인 영향을 미치지 않지만 개발자들이 소스 분석 내용 및 파일 설명 처리를 위해서 없어서는 안 될 꼭 필요한 요소

■ HTML 형식의 주석

```
<!-- Fighting <%=name%> -->
```

■ JSP 형식의 주석

```
<%-- Fighting <%=name%> --%>
```

■ 스크립트 요소의 주석

```
<% /*주석.....여러 줄 주석 및 부분 주석)*/ %>
```

```
<% //주석....(한줄 주석)%>
```

기본 제어문(FOR & WHILE)

- for문이란?
 - 반복문은 모두 스크립트 요소에서 사용하여 JSP 페이지에서 반복적인 내용을 출력할 수 있음
 - Database의 질의 결과를 순서대로 출력할 때 매우 유용하게 사용
 - for문은 크기가 고정되어 있을 때 사용이 많이 됨
- **while문이란?**
 - 조건을 검사해서 조건이 참(true)이면 실행문을 반복적으로 실행하고 그렇지 않으면 while문을 빠져 나오는 동작을 하는 반복문.
 - while문 안에 조건이 항상(true)인 경우는 while문이 무한반복되는 경우도 있음.

지시자의 종류 3가지

page

include

taglib

지시자
(Directive)

PAGE 지시자 속성 종류

속성	값	기본값	예제
info	텍스트	없음	<code>info="Copyright 2013 by JspStudy.co.kr"</code>
language	스크립팅 언어	<code>"java"</code>	<code>language="java"</code>
contentType	MIME 타입, 문자집합	<code>contentType="text/html; charset=ISO-8859-1"</code>	<code>contentType="text/html; charset=EUC-KR"</code>
extends	클래스 이름	없음	<code>extends="kr.co.jspstudy.board.JspPage"</code>
import	클래스/패키지 이름	없음	<code>import="java.util.Vector"</code> <code>import="java.sql.*,java.net.*"</code>
session	boolean 값	<code>"true"</code>	<code>session="true"</code>
buffer	buffer값 or "none"	<code>"8kb"</code>	<code>buffer="12kb"</code> <code>buffer="false"</code>
autoFlush	boolean 값	<code>"true"</code>	<code>autoFlush="false"</code>
isThreadSafe	boolean 값	<code>"true"</code>	<code>isThreadSafe="true"</code>
trimDirectiveWhitespaces	boolean 값	<code>"false"</code>	<code>trimDirectiveWhitespaces="false"</code>
errorPage	로컬 URL	없음	<code>errorPage="error/fail.jsp"</code>
isErrorPage	boolean 값	<code>"false"</code>	<code>isErrorPage="false"</code>
pageEncoding	페이지의 캐릭터 인코딩값	<code>"ISO-8859-1"</code>	<code>pageEncoding="EUC-KR"</code>

PAGE 지시자 속성-1

info 속성

- `<%@page info="naver.com"%>`

language 속성

- `<%@page language="java"%>`

contentType 속성

- `<%@page contentType="text/html"%>`
- `<%@page contentType="text/html"; charset="EUC-KR"%>`

PAGE 지시자 속성-2

extends 속성

- `<%@page extends="com.jspstudy.Diretive"%>`

import 속성

- `<%@page import="java.util.*,java.sql.*" import="java.io.*" %>`

session 속성

- `<%@page session="false"%>`

buffer 속성

- `<%@page buffer="16kb"%>`, `<%@page buffer="none"%>`

autoFlush 속성

- `<%@page autoFlush="false"%>`

isThreadSafe 속성

- `<%@page isThreadSafe="false"%>`

내부객체란?

- JSP 페이지를 작성할 때 특별한 기능을 제공하는 JSP 컨테이너가 제공하는 특별한 객체
- JSP에서 선언하지 않고 사용할 수 있는 객체
- 사용되는 범주에 따라 4가지 형태로 분류
 - JSP 페이지 입출력 관련 내부 객체
 - JSP 페이지 외부 환경 정보 제공 내부 객체
 - JSP 페이지 서블릿 관련 내부 객체
 - JSP 페이지 예외 관련 기본객체

내부 객체란?

<%@include file="로컬URL"%>

내부 객체	Type	설명
request	javax.servlet.http.HttpServletRequest	파라미터를 포함한 요청을 담고 있는 객체
response	javax.servlet.http.HttpServletResponse	요청에 대한 응답을 담고 있는 객체
out	javax.servlet.jsp.JspWriter	페이지 내용을 담고 있는 출력 스트림 객체
session	javax.servlet.http.HttpSession	세션 정보를 담고 있는 객체
application	javax.servlet.ServletContext	어플리케이션 Context의 모든 페이지가 공유할 데이터를 담고 있는 객체
pageContext	javax.servlet.jsp.PageContext	페이지 실행에 필요한 Context 정보를 담고 있는 객체
page	javax.servlet.jsp.HttpJspPage	JSP 페이지의 서블릿 객체
config	javax.servlet.ServletConfig	JSP 페이지의 서블릿 설정 데이터 초기화 정보 객체
exception	java.lang.Throwable	JSP 페이지의 서블릿 실행 시 처리하지 못한 예외 객체

```
<body>
  <%
    out.print("JSP");
  %>
</body>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 사용하는 언어가 자바이며, 이 페이지는 html 문서이며 한글인코딩을 utf-8로 처리하겠다 -->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <%= request.getContextPath() %><br>
  <%= request.getMethod() %><br>
  <%= request.getRequestURL() %><br>
  <%= request.getRequestURI() %><br>
  <%= request.getServerName() %><br>
  <%= request.getProtocol() %><br>

</body>
</html>
```

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 사용하는 언어가 자바이며, 이 페이지는 html 문서이며 한글인코딩을 utf-8로 처리하겠다 -->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
        response.sendRedirect("http://www.naver.com");
    %>

</body>
</html>

```

```

<%
    response.sendRedirect("main.jsp?age=" + 20);
    //                파라미터이름    파라미터값
%>

```

내부객체란?

- request, session, application, pageContext 내부객체는 임의의 속성값을 저장하고 읽을 수 있는 메소드를 제공

메소드	설명
setAttribute(key, value)	주어진 key(이름 등)에 속성값을 연결합니다.
getAttributeNames()	모든 속성의 이름을 얻어냅니다.
getAttribute(key)	주어진 key에 연결된 속성값을 얻어냅니다.
removeAttribute(key)	주어진 key에 연결된 속성값을 제거합니다.

내부객체란?

- request 내부객체는 브라우저에서 **JSP** 페이지로 전달되는 데이터의 묶음으로 **HTTP** 헤더와 **HTTP** 바디로 구성

메소드	설명
String getParameter(name)	name에 할당된 값을 반환하며 지정된 파라미터 값이 없으면 null 값을 반환합니다.
String[] getParameterValues(name)	name의 모든 값을 String 배열로 반환합니다.
Enumeration getParameterNames()	요청에 사용된 모든 파라미터 이름을 java.util Enumeration 타입으로 반환합니다.

메소드	설명
String getMethod()	요청에 사용된 요청 방식(GET, POST, PUT)을 반환합니다.
String getRequestURI()	요청에 사용된 URL로부터 URI를 반환합니다.
String getQueryString()	요청에 사용된 Query 문장을 반환합니다.
String getRemoteHost()	클라이언트의 호스트 이름을 반환합니다.
String getRemoteAddr()	클라이언트의 주소를 반환합니다.
String getProtocol()	사용 중인 프로토콜을 반환합니다.
String getServerName()	서버의 도메인 이름을 반환합니다.
int getServerPort()	서버의 주소를 반환합니다.
String getHeader(name)	HTTP 요청 헤더에 지정된 name의 값을 반환합니다.

RESPONSE 내부객체

- response 내부객체는 요청을 시도한 클라이언트로 전송할 응답을 나타내는 데이터의 묶음.

메소드	설명
<code>void setHeader(name, value)</code>	응답에 포함될 Header를 설정합니다.
<code>void setContentType(type)</code>	출력되는 페이지의 contentType을 설정합니다.
<code>String getCharacterEncoding()</code>	응답 페이지의 문자 인코딩 Type을 반환합니다.
<code>void sendRedirect(url)</code>	지정된 URL로 요청을 재전송합니다.

OUT 내부객체

- out 내부객체는 JSP페이지의 결과를 클라이언트에 전송해 주는 출력 스트림을 나타내며 JSP페이지가 클라이언트에게 보내는 모든 정보는 out 객체를 통해서 전달됩니다.

메소드	설명
boolean isAutoFlush()	출력 버퍼가 다 채워져 자동으로 flush 했을 경우는 true 반환, 그렇지 않은 경우는 false를 반환합니다.
int getBufferSize()	출력 버퍼의 전체 크기를 바이트 단위로 반환합니다.
int getRemaining()	출력 버퍼의 남은 양을 바이트 단위로 반환합니다.
void clearBuffer()	현재 출력 버퍼에 저장된 내용을 취소합니다.
String println(string)	string을 브라우저에 출력합니다.
void flush()	현재 출력 버퍼의 내용을 flush하여 클라이언트로 전송합니다.
void close()	출력 버퍼의 내용을 flush하고 스트림을 닫습니다.

SESSION 내부객체

- session 내부객체는 클라이언트 요청에 대한 context 정보의 세션과 관련된 정보(데이터)를 저장하고 관리하는 객체

메소드	설명
String getId()	해당 세션의 세션 ID를 반환합니다.
long getCreationTime()	세션의 생성된 시간을 반환합니다.
long getLastAccessedTime()	클라이언트 요청이 마지막으로 시도된 시간을 반환합니다.
void setMaxInactiveInterval(time)	세션을 유지할 시간을 초단위로 설정합니다.
int getMaxInactiveInterval()	setMaxInactiveInterval(time)로 지정된 값을 반환합니다. 기본값은 30분으로 지정됩니다.
boolean isNew()	클라이언트 세션 ID를 할당하지 않은 경우 true 값을 반환합니다.
void invalidate()	해당 세션을 종료시킵니다.

APPLICATION 내부 객체

- application 내부객체는 서블릿 또는 어플리케이션 외부 환경 정보(Context)를 나타내는 객체로 서버의 정보와 자원 그리고 이벤트 로그 같은 정보를 제공합니다.

메소드	설명
String getServerInfo()	서블릿 컨테이너의 이름과 버전을 반환합니다.
String getMimeType(fileName)	지정한 파일의 MIME 타입을 반환합니다.
String getRealPath(url)	URL를 로컬 파일 시스템으로 변경하여 반환합니다.
void log(message)	로그 파일에 message를 기록합니다.

PAGECONTEXT 내부객체

- pageContext 내부객체는 현재 JSP 페이지의 Context를 나타내면 pageContext 객체를 통해서 다른 내부 객체에 접근할 수가 있

메소드	설명
<code>ServletRequest getRequest()</code>	페이지 요청 정보를 담고 있는 객체를 반환합니다.
<code>ServletResponse getResponse()</code>	페이지 요청에 대한 응답 객체를 반환합니다.
<code>String getRealPath(url)</code>	URL를 로컬 파일 시스템으로 변경하여 반환합니다.
<code>JspWriter getOut()</code>	페이지 요청에 대한 응답 출력 스트림을 반환합니다.
<code>HttpSession getSession()</code>	요청한 클라이언트의 세션 정보를 담고 있는 객체를 반환합니다.
<code>ServletContext getServletContext()</code>	페이지에 대한 서블릿 실행 환경 정보를 담고 있는 객체를 반환합니다.
<code>Object getPage()</code>	페이지의 서블릿 객체를 반환합니다.
<code>ServletConfig getServletConfig()</code>	페이지의 서블릿 초기 정보의 설정 정보를 담고 있는 객체를 반환합니다.
<code>Exception getException()</code>	페이지 실행 중에 발생하는 에러 페이지에 대한 예외 객체를 반환합니다.

CONFIG 내부객체

- config 내부객체에는 javax.servlet.ServletConfig 클래스 타입이고 Servlet이 초기화 될 때 참조 해야 할 다른 여러 정보를 가지고 있습니다.

메소드	설명
Enumeration getInitParameterNames()	서블릿 설정 파일에 지정된 초기 파라미터 이름을 반환합니다.
String getInitParameter(name)	지정한 name의 초기 파라미터 이름을 반환합니다.
String getServletName()	서블릿의 이름을 반환합니다.
ServletContext getServletContext()	실행하는 서블릿 ServletContext를 반환합니다.

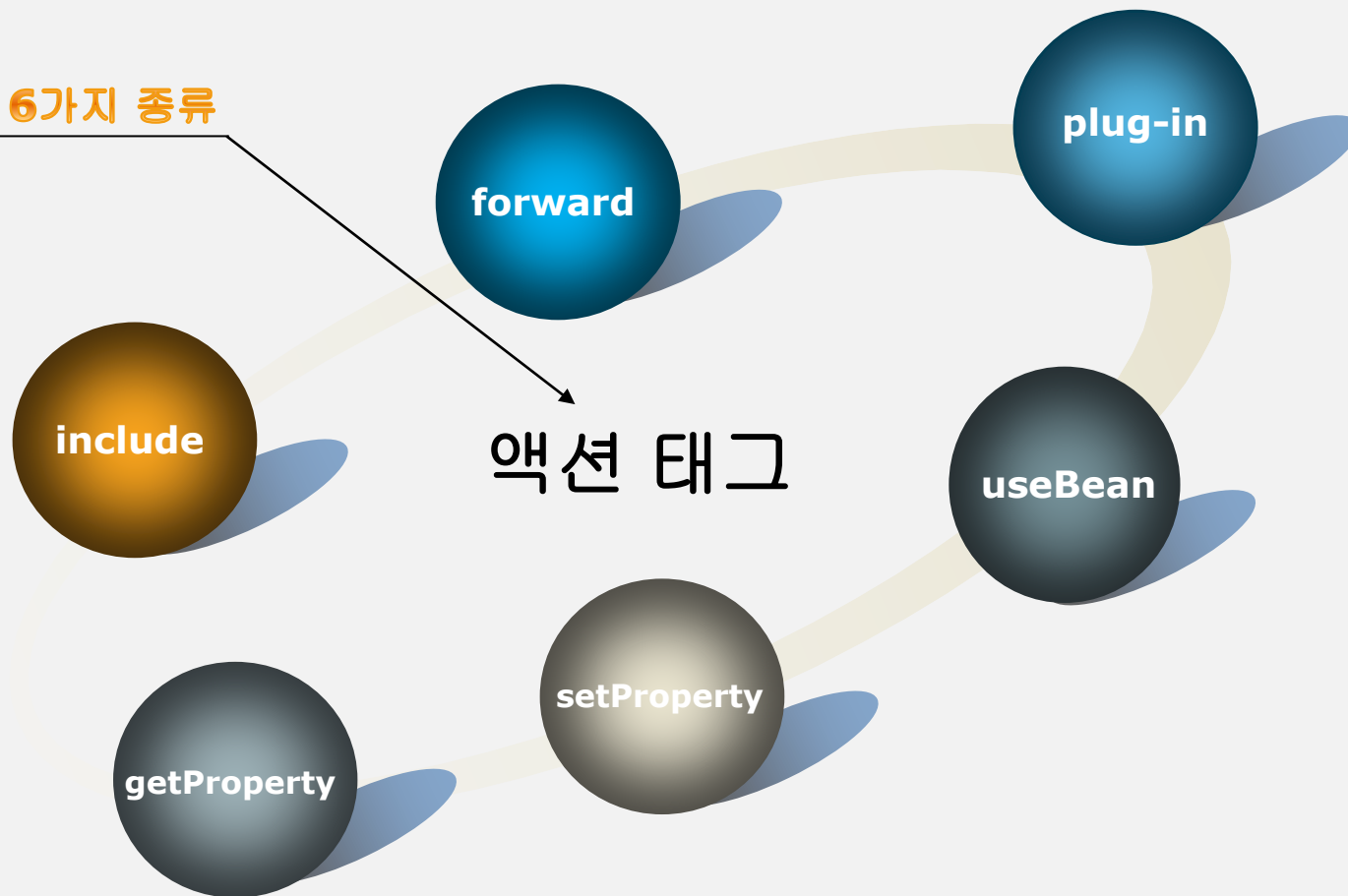
EXCEPTION 내부객체

- exception 내부객체는 개발자가 JSP 페이지에서 발생한 예외를 처리하는 페이지를 지정한 경우 에러 페이지에 전달되는 예외객체입니다.
- page 지시자의 isErrorPage 속성을 true로 지정한 JSP 페이지만 사용 가능한 객체이고 예외처리를 설정한 JSP 페이지에는 errorPage 속성에 예외처리 페이지를 설정해야 합니다.

메소드	설명
String getMessage()	에러 메시지를 반환합니다.
String toString()	에러 실체의 클래스명과 에러 메시지를 반환합니다.

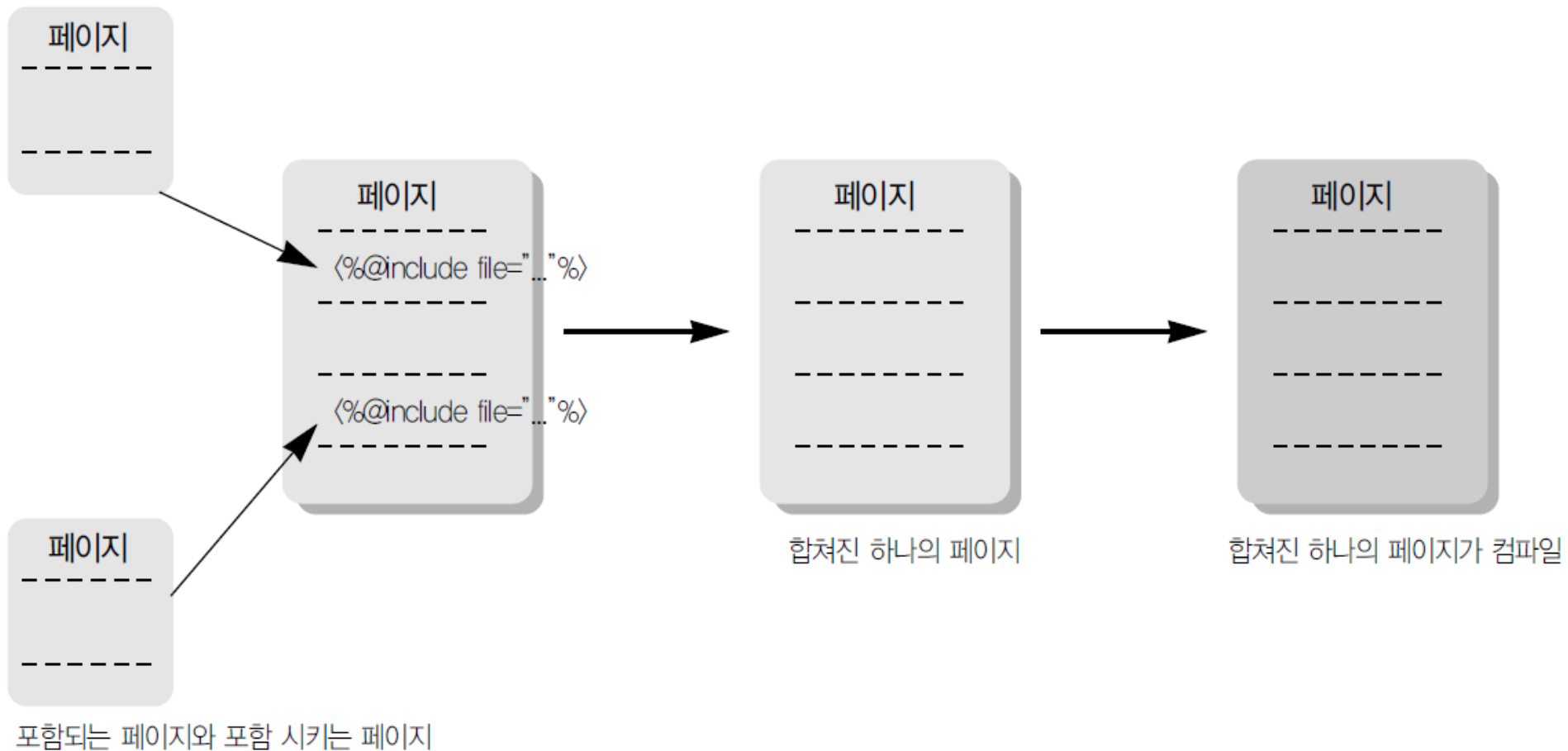
액션 태그

액션 태그 6가지 종류



INCLUDE 지시자

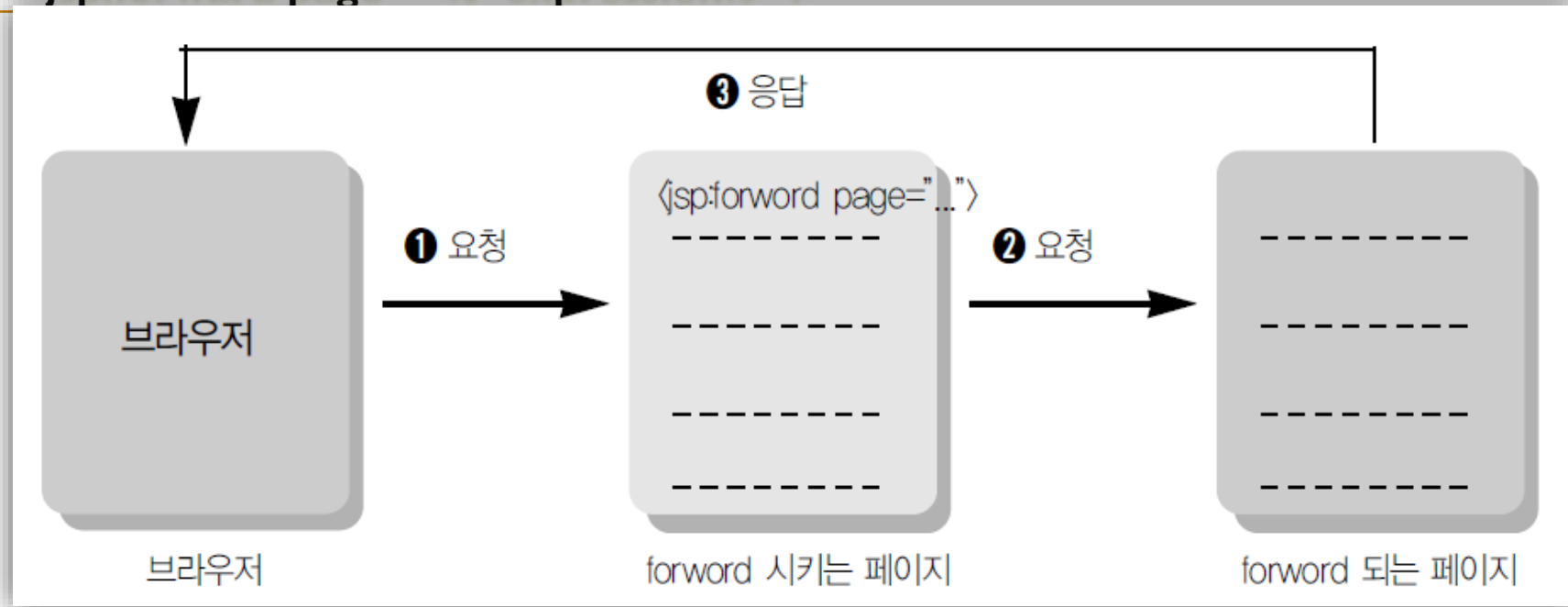
`<%@include file="로컬URL"%>`



FORWARD 액션 태그

forward 사용 예

- `<jsp:forward page="로컬URL"/>`
- `<jsp:forward page="로컬URL"></jsp:forward>`
- `<jsp:forward page='<%=expression%>' />`



스크립트 요소를 대체하는 액션 태그

스크립트 요소 대체 사용 예

- `<jsp:declaration> 코드 </jsp:declaration>`
- `<jsp:scriptlet> 코드 </jsp:scriptlet>`
- `<jsp:expression> 코드 </jsp:expression>`
- `<jsp:directive.page contentType="text/html; charset=EUC-KR" />`
- `<jsp:directive.include file="xxx.jsp" />`

기타 액션 태그(PLUG-IN, USEBEAN)

plug-in 속성

- <jsp:plugin> 액션은 자바 플러그인(Java Plug-in)을 사용하여

useBean 속성

- 릿을 JSP 페이지에서 실행할 때 사용하는 액션 태
- ① <jsp:useBean id=".." class=".." scope=".."/>
 - ② <jsp:setProperty name=".." property=".." value=".."/>
 - ③ <jsp:getProperty name=".." property=".."/>

자바빈즈

- 하나의 묶음으로 전송하는 것 (데이터 저장소)

: 프로그램에서 사용되는 정보가 여러 개라면 변수에 저장하고 필요할 때마다 개별적으로 접근해서 사용하기 보다는 필요한 정보를 객체를 구성하는 멤버로 설정해 두고 한꺼번에 데이터에 접근해서 사용할 수 있다.

: 자바 은닉 개념을 사용해 데이터 손상을 막는다(private)

: 공개된 메서드를 통해서만 접근(public)

=> 일종의 템플릿 개념

자바빈 - 액션태그

종류	설명
<jsp:useBean>	자바 빈을 생성한다 (jsp와 자바 빈을 연결하기 위한 자바 빈 객체 생성)
<jsp:getProperty>	자바 빈에서 정보 얻어 온다
<jsp:setProperty>	자바 빈에서 정보 저장한다

자바빈 - 액션태그

- 자바 빈 객체 생성
- `<jsp:useBean class="클래스 풀 네임" id="빈이름" [scope="범위"]/>`

(scope)범위는 생략 가능

class : 자바 빈 클래스의 풀네임을 써야함,

id : 자바 빈 객체 이름

scope: 자바 빈 객체가 사용되는 유효범위(page, request, session, application)

자바빈 - 액션태그

Scope값	설명
page	자바 빈은 생성된 페이지 내에서만 접근되어 사용할 수 있음 - 해당 페이지 내에서만
request	자바 빈이 생성된 요청을 수행하는 페이지들에서 사용할 수 있음 - 요청 페이지까지
session	자바 빈이 생성된 세션에서 요청을 처리하는 페이지들에서 사용할 수 있음 - 브라우저 닫히기 전까지
application	자바 빈이 생성된 응용프로그램에 포함된 페이지들에서 사용할 수 있음 - 톰캣을 restart시켜 서버를 재시작하기 전까지

자바빈 - 액션태그

- `<jsp:useBean class="클래스 풀 네임" id="빈이름" [scope="범위"]/>`

:id값은 반드시 set, getProperty의 name과 일치해야한다!!!!

=> 자바빈 객체의 정보를 얻어올 수 있음

`< jsp:getProperty name="id값" property="">`

데이터베이스 커넥션 풀

- 웹 페이지에 접속자의 수가 많게 되면 커넥션을 그만큼 걸어주어야 하기 때문에 서버에 부하가 발생!! -> 서버가 다운되는 현상 발생!!

⇒ 커넥션 풀 필요

- DBCP(DataBase Connection Pool) : 접속 인원이 많은 웹 페이지에서 데이터베이스의 효율성과 속도를 높이기 위해 사용됨

=> DBCP 매니저가 어느 정도의 연결을 확보해 놓고 있다가 클라이언트의 요청이 들어오면 연결해주고, 클라이언트의 작업이 다 끝나면 연결을 다시 DBCP 매니저에게 반환한다

