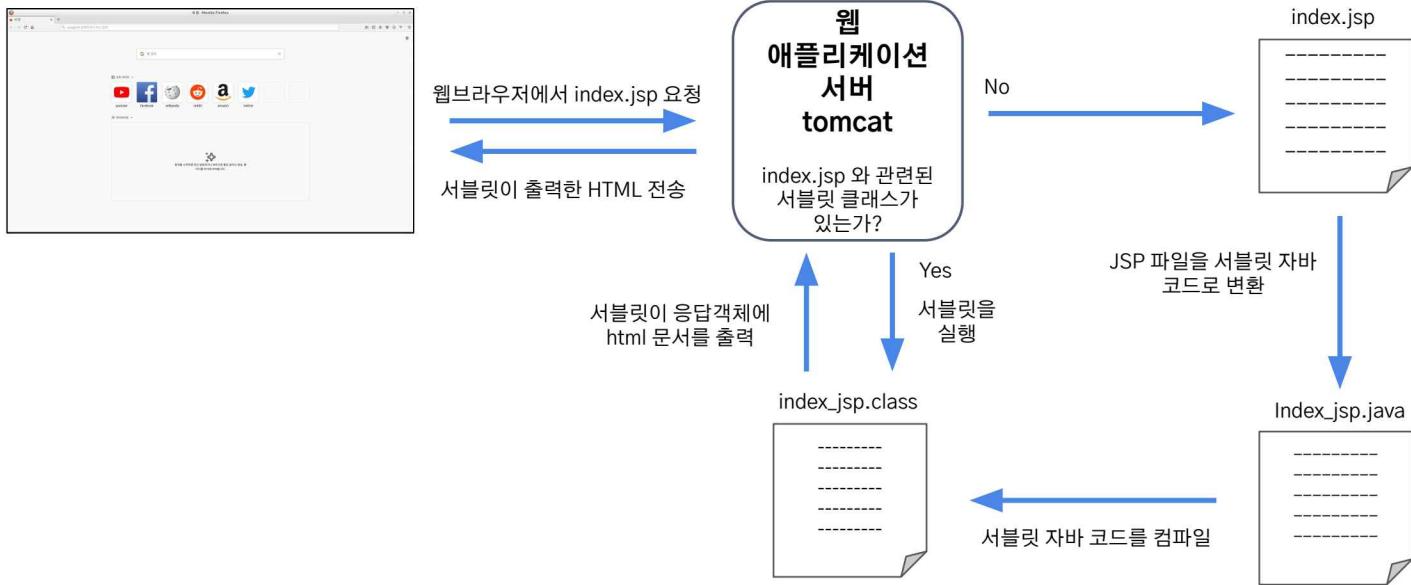


# # JSP 처리 과정



# # JSP는 HTML의 탈을 쓴 Servlet

Hello.jsp	Hello_jsp.java	Hello.java
<pre> &lt;%@ page contentType="text/html;charset=utf-8" %&gt; &lt;%@ page import="java.util.Date" %&gt; &lt;%     private String s = new String("s"); %&gt; &lt;html&gt; &lt;head&gt;&lt;title&gt;예제&lt;/title&gt;&lt;/head&gt; &lt;body&gt;  &lt;%     Date now = new Date(); %&gt; 현재 시간은 &lt;%= now %&gt; &lt;/body&gt; &lt;/html&gt; </pre>	<pre> import java.util.Date; public class Hello_jsp extends HttpJspBase {     public void _jspInit() { }     public void _jspDestroy() { }     private String s = new String("s");     public void _jspService( HttpServletRequest request, HttpServletResponse response ) {         response.setContentType("text/html; charset=UTF-8");         JspWriter out = pageContext.getOut();         out.write("&lt;html&gt;\n");         out.write("&lt;head&gt;...&lt;/head&gt;");         out.write("&lt;body&gt;\n");         Date now = new Date();         out.write("현재 시간은 ");         out.print( now );         out.write("&lt;/body&gt;\n");         out.write("&lt;/html&gt;\n");     } } </pre>	<pre> import java.util.Date; public class Hello extends HttpServlet {     public void init() { }     public void destroy() { }     private String s = new String("s");     protected void service( HttpServletRequest request, HttpServletResponse response ) {         response.setContentType("text/html; charset=UTF-8");         PrintWriter out = response.getWriter();         out.write("&lt;html&gt;\n");         out.write("&lt;head&gt;...&lt;/head&gt;");         out.write("&lt;body&gt;\n");         Date now = new Date();         out.write("현재 시간은 ");         out.print( now );         out.write("&lt;/body&gt;\n");         out.write("&lt;/html&gt;\n");     } } </pre>

# # JSP 페이지 작성 방법

- JSP 파일에 대한 기본 설정을 <%@ page %> 디렉티브로 기술
  - <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
- HTML과 텍스트 내용은 그대로 작성 (일반적인 HTML 파일처럼)
- 실행할 자바 코드는 <% %> 내부에 작성
  - 별도의 변수 선언 없이 사용 가능한 기본 객체들 사용 가능
    - pageContext, request, response, session, application, out, config, exception, page
  - 실행 결과를 화면에 출력하고 싶은 자바 코드는 <%= %> 내부에 작성
  - 처음 한번만 실행하고, 요청이 올 때마다 실행하지 않을 자바 코드는 <%! %> 내부에 작성
    - 서블릿의 service() 내부에 위치할 코드는 <% %>를, 외부에 위치할 코드는 <%! %>를 사용
- 자바 코드에서 사용하는 클래스는 <%@ page import %> 로 임포트
  - <%@ page import="java.util.Date" %>

## 스크립트 요소

- 스크립트릿(Scriptlet)
- 표현식(Expression)
- 선언부(Declaration)

# # JSP 페이지 작성 방법

- JSP 기본객체에 저장된 데이터(속성값)을 사용하는 코드는 \${ }로 대체
  - <%=pageContext.getAttribute("xxx") %> → \${pageScope.xxx} 또는 \${xxx}
  - <%=request.getAttribute("xxx") %> → \${requestScope.xxx} 또는 \${xxx}
  - <%=session.getAttribute("xxx") %> → \${sessionScope.xxx} 또는 \${xxx}
  - <%=application.getAttribute("xxx") %> → \${applicationScope.xxx} 또는 \${xxx}
- JSP 기본객체는 \${pageContext.기본객체명}으로 사용
- if, switch 문은 <c:if> 또는 <c:choose><c:when> 태그로 대체
- for, while 문은 <c:foreach> 태그로 대체
- 다른 JSP 파일의 내용을 포함시키고 싶을 때
  - <%@ include %>, <jsp:include >, <c:import > 태그 중 선택하여 사용
- 날짜/시간 데이터를 원하는 형태로 출력하고 싶을 때 <fmt:formatDate />

# # page 디렉티브

## • <%@ page 속성명="속성값" 속성명="속성값" ... %>

- import – 임포트할 클래스
- contentType("text/html;charset=ISO-8859-1") – JSP가 생성(출력)할 문서의 MIME 타입
- pageEncoding("ISO-8859-1") – 현재 파일의 문자 인코딩을 지정
- language("java") – 스크립트 요소에서 사용할 언어를 지정
- errorPage – 현재 JSP에서 에러 발생 시, 발생한 에러를 처리할 페이지를 지정
- isErrorPage("false") – 해당 페이지가 에러를 처리를 위한 에러 페이지라는 것을 지정
- trimDirectiveWhitespace("false") – 출력 결과에서 공백 문자를 제거할지의 여부를 지정
- buffer("8Kb") – JSP가 사용할 출력 버퍼의 크기
- info – 현재 페이지에 대한 설명 (getServletInfo() 메서드 호출시 반환될 문자열 값)
- extends – JSP가 생성하는 서블릿 클래스가 상속할 부모 클래스
- isELIgnored("false") – EL 무시 여부 (표현 언어(EL)에 대한 지원 여부를 설정)
- isThreadSafe("true") – false 설정시, 웹 컨테이너는 이 JSP로의 요청들을 직렬화해서 하나씩 처리
- autoFlush("true") – 출력 버퍼가 가득 찰 경우, 자동으로 버퍼 내용을 전송하고 비울 것인지 여부
- session("true") – 세션 객체를 생성(사용)할 것인지 여부 지정
- deferredSyntaxAllowedAsLiteral("false") – true면 #{expr} 형식의 EL을 단순 문자열로 처리
  - 서블릿 2.3부터 EL을 지원, 서블릿 2.4부터 \${expr} 지원 / #{expr} 미지원, 서블릿 2.5부터 \${expr} / #{expr} 지원

# # JSP 기본 객체

## • JSP 스크립트 요소 내에서 별도의 변수 선언 없이 사용 가능한 자바 객체

기본 객체	실제 타입	서블릿
pageContext	javax.servlet.jsp.PageContext	현재 JSP 페이지에서 사용하는 데이터를 속성으로 저장 및 읽기 pageContext를 통해서 다른 JSP 기본 객체들에 접근 가능
request	javax.servlet.http.HttpServletRequest 또는 javax.servlet.ServletRequest	service(HttpServletRequest request, HttpServletResponse response)
response	javax.servlet.http.HttpServletResponse 또는 javax.servlet.ServletResponse	service(HttpServletRequest request, HttpServletResponse response)
session	javax.servlet.http.HttpSession	HttpSession session = request.getSession();
application	javax.servlet.ServletContext	ServletContext application = request.getServletContext();
out	javax.servlet.jsp.JspWriter	PrintWriter out = response.getWriter(); 와 유사
config	javax.servlet.ServletConfig	ServletConfig config = this.getServletConfig();
page	java.lang.Object	JSP 페이지를 구현한 자바 클래스 인스턴스 (this)
exception	java.lang.Throwable	발생한 예외 객체 (에러 페이지에서만 사용)

# # JSP 기본 객체

## • 속성(Attribute)을 사용하여 데이터 저장이 가능한 기본 객체들

- void setAttribute(String name, Object value) : 이름이 name인 속성의 값으로 value를 저장
- Object getAttribute(String name) : 이름이 name인 속성의 값 조회 (없을 경우 null 반환)
- void removeAttribute(String name) : 이름이 name인 속성을 삭제
- Enumeration.getAttributeNames() : 속성의 이름 목록 조회 (pageContext는 사용 불가)

기본 객체	스코프	사용
pageContext	page	(하나의 요청을 처리하는) 하나의 JSP 페이지 내에서 공유할 데이터 저장 커스텀 태그에서 새로운 변수를 추가, 다른 기본 객체 사용, 에러 데이터 등
request	request	하나의 요청을 처리할 때 사용되는 모든 JSP 페이지들에서 공유할 데이터 저장
session	session	하나의 사용자(브라우저)와 관련된 데이터를 다수의 요청 및 JSP들에서 공유 사용자의 로그인 정보 저장
application	application	모든 사용자들의 모든 요청에서 함께 공유할 데이터를 저장 임시 디렉토리 경로와 같은 웹 애플리케이션의 설정 정보 저장

# JSP Action Tags

많이 사용하는 자바 코드를 HTML 태그와 유사한 형태인 <jsp:xxx>로 교체

# # Standard Actions

- 객체 생성 및 사용
  - <jsp:useBean>
    - <jsp:setProperty>
    - <jsp:getProperty>
- 페이지 이동
  - <jsp:include>
    - <jsp:param>
  - <jsp:forward>
    - <jsp:param>
- 커스텀 태그 작성
  - <jsp:invoke>
  - <jsp:doBody>
- 자바 객체(애플릿)을 포함시키는 HTML 생성
  - <jsp:plugin>
    - <jsp:params>
      - <jsp:param>
    - <jsp:fallback>
- XML 생성
  - <jsp:output>
  - <jsp:root>
  - <jsp:element>
    - <jsp:attribute>
    - <jsp:body>
- JSP 기본 내용
  - <jsp:declaration> → <%! %>
  - <jsp:scriptlet> → <% %>
  - <jsp:expression> → <%= %>
  - <jsp:text> → no needed
    - EL 표현식을 포함한 내용을 출력

# # 자바 객체 생성 및 사용

- 자바빈(JavaBean)은 데이터를 표현하는 것을 목적으로 하는 자바 클래스
  - 자바빈은 프로퍼티, 지속성, 이벤트 등 다양한 특징을 갖는데, JSP에서는 프로퍼티를 많이 사용
    - 인자 없는 생성자
    - Serializable
    - 속성값을 위한 getter/setter 메서드
- 스코프에 저장되어 있는 자바 객체를 가져와서 사용하거나, 자바 객체를 생성하여 스코프에 저장

```
<jsp:useBean id="instanceName"
    scope= "page | request | session | application"
    class= "packageName.ClassName" type= "packageName.ClassName"
    beanName="packageName.ClassName | <%= expression %>">
</jsp:useBean>
```

```
Type id = scope.getAttribute("id");
if (id==null) {
    id = new Class();
    scope.setAttribute("id", id);
}
```

- scope 생략시 page(pageContext)가 기본
- type 생략시, class를 대신 사용
- class 생략시, 스코프에 지정한 이름의 속성값이 존재하지 않으면, 객체를 새로 생성하지 않고 오류 발생
- 생성할 객체의 클래스와 해당 객체를 저장할 변수의 타입이 다른 경우, class 속성과 type 속성을 모두 지정
- beanName: java.beans.Beans.instantiate() 메서드를 사용하여 인스턴스를 생성

# # 자바 객체의 속성 값 저장 및 읽기

## • <jsp:setProperty> 액션 태그로 자바빈 객체의 프로퍼티 값을 설정

- <jsp:setProperty name="xxx" property="yyy" value="zzz" /> → xxx.setYyy("zzz");
  - name : 자바빈 객체의 이름 (<jsp:useBean> 태그의 id 속성에서 지정한 값을 사용)
  - property : 값을 지정할 프로퍼티의 이름
  - value : 프로퍼티의 값 (표현식 사용 가능)
  - param : value 속성 대신에 사용하여, 파라미터 값을 프로퍼티의 값으로 지정
    - property 속성 값을 "\*"로 지정하면 각 프로퍼티의 값을 같은 이름을 갖는 파라미터의 값으로 설정
  - <jsp:useBean> 태그 안에 <jsp:setProperty>를 사용하면, 빈을 새로 생성한 경우에만 값을 설정하고, 이미 빈이 있는 경우에는 설정하지 않는다.

## • <jsp:getProperty> 액션 태그로 자바빈 객체의 프로퍼티 값을 출력

- <jsp:getProperty name="xxx" property="yyy" /> → xxx.getYyy();
  - name : 자바빈 객체의 이름 (<jsp:useBean> 태그의 id 속성에서 지정한 값을 사용)
  - property : 값을 지정할 프로퍼티의 이름

```
<jsp:useBean id="member" class="kr.next.member.MemberBean" scope="request" %></jsp:useBean>
<jsp:setProperty name="member" property="name" value="이산" />
<jsp:setProperty name="member" property="id" param="memberId" />
<jsp:getProperty name="member" property="name" />
```

# # 페이지 이동 및 포함

## • <jsp:forward> 액션 태그를 이용한 JSP 페이지 이동

- request.getRequestDispatcher("주소").forward(request, response);
- <jsp:forward page="주소" />

## • <jsp:include> 액션 태그를 이용한 JSP 페이지 삽입

- request.getRequestDispatcher("주소").include(request, response);
- <jsp:include page="주소" flush="true" />
  - page : 이동 또는 삽입하고자 하는 페이지의 URL 주소 (표현식 사용 가능)
  - flush : 페이지 이동 전에 응답 버퍼의 내용을 브라우저로 전송 (버전 1.1부터 값은 항상 "true"로 지정)

## • <jsp:include>, <jsp:forward> 내부에 <jsp:param>로 파라미터 추가 가능

- <jsp:param name="파라미터명" value="파라미터값" />
- value 값에 표현식을 사용할 수 있지만, 결국 파라미터는 문자열 형태로만 값을 전달 할 수 있는 제약이 존재하므로, 기본 객체(보통 request)의 속성을 이용해서 데이터를 전달하는 것이 편리

# EL (Expression Language)

<%= %>를 더 단순하고 간편한 표현인 \${ }로 교체

## # EL (Expression Language)

---

- JSP에서 값을 표현하기 위하여 사용하는 (표현식 보다 간결한) 언어
  - <%= expr %> ➔ \${ expr } (JSTL1.0에서 소개, JSP2.0에 포함, JSP2.1부터 EL2.1로 독립)
  - deferred expression : #{expr} (JSP2.1, Servlet2.5부터)
- JSP의 스크립트 요소를 제외한 부분에 사용 가능
  - 액션 태그 또는 커스텀 태그의 속성값으로 사용 가능
- 값이 null인 경우에는 JAVA와 달리 오류가 발생하지 않고, null이 아닌데 오류가 발생하는 경우는 JAVA와 동일하게 오류 발생
- \문자를 사용하여 EL 비활성화
  - \\${xxx} 또는 \#{xxx} → “\${xxx}”과 “#{xxx}”을 문자 그대로 출력

# # EL의 기능

- \${} 내부에서 별도의 변수 선언 없이 사용 가능한 EL만의 기본 객체 제공
  - pageContext, pageScope, requestScope, sessionScope, applicationScope, param, paramValues, header, headerValues, cookie, initParam
- 연산자
  - . : 맵/빈의 속성 참조, [ ] : 맵/빈/배열/리스트의 속성 참조
  - 수치(산술) 연산, 관계(비교) 연산, 논리 연산
  - 대입 연산자 (EL 3.0)
- 데이터 리터럴
  - 정수(Long), 실수(Double), 문자열(String), 논리(Boolean), null 타입
  - List, Set, Map (EL 3.0)
- 메서드 호출
  - TLD 작성률 통해서 클래스의 정적 메서드 실행
  - TLD 작성 없이 <%@ import %>를 통해서 클래스의 정적 메서드 실행 (EL3.0)
  - 인스턴스의 메서드 실행 가능 (EL2.2) (반환타입이 void인 메서드도 가능)
  - 람다식을 이용한 함수 정의와 실행 (EL3.0)
  - 스트림API를 이용한 컬렉션 처리 (EL3.0)

# # EL (Expression Language)

## • JSP(<% %>)의 기본객체와 EL(\${})의 기본객체는 다르다

- JSP 기본 객체가 필요하다면 "pageContext.기본객체명"을 통해서 사용
- "xxxScope"를 통해서 요청, 세션, 서블릿컨텍스트에 저장된 속성값들을 편리하게 사용

리터럴	연산자	기본객체	기본객체의 속성
true, false	. (맵or빈의 속성 참조)	pageContext	xxx
숫자 (1.23e4도 가능)	[ ] (맵or빈or배열or리스트의 속성 참조)	pageScope	pageScope.xxx
'문자열', "문자열"	+, -, *, /, div, %, mod	requestScope	requestScope.xxx
null	and, &&, or,   , not, !	sessionScope	sessionScope.xxx
(EL3.0)	==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge	applicationScope	applicationScope.xxx
List : [ 1, 2, 3 ]	? :	param	
Map : {"x": "1", "y": 2}	empty (null, "", 빈 배열/Collection/Map)	paramValues	<% %>에서
Set : { 1, 2, 3 }	(EL3.0) += ; =	header headerValues cookie initParam	pageContext.getAttribute("xxx") request.getAttribute("xxx") session.getAttribute("xxx") application.getAttribute("xxx")

# JSTL

JSP Standard Tag Library

## # JSTL (JSP 표준 태그 라이브러리)

- **JSTL(JSP Standard Tag Library)**

- 커스텀 태그 중에서 많이 사용되는 것들을 모아서 만든 것
  - 커스텀 태그 : 원하는 기능을 수행하는 태그를 개발자가 직접 구현하여 사용

- **JSP에서 커스텀 태그를 사용하고 싶다면,**

- 프로젝트에 JSTL 라이브러리 jar 파일을 추가
  - Apache Tomcat > Download > Spec, Impl jar 파일 다운로드
- JSP 페이지에 <%@ taglib %> 디렉티브로 사용할 커스텀 태그 라이브러리 지정
  - <%@ taglib prefix="접두어" uri="사용고싶은JSTL라이브러리의URI" %>
    - URI는 커스텀 태그를 구분하는 식별자(일종의 고유 아이디)
    - 아래 표의 접두어는 일반적으로 사용되는 것으로, 다른 것을 사용해도 무방
- JSP 페이지의 내용에 <접두어:태그이름> 형태로 사용

- **JSTL이 제공하는 태그의 종류**

라이브러리	하위 기능	접두어	URI
코어	변수 지원, 흐름 제어, URL 처리	c	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>
국제화	지역/언어별 메시지, 숫자, 날짜 형식 변환	fmt	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>
함수	컬렉션 처리, String 처리	fn	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>
데이터베이스	SQL 실행	sql	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>
XML	XML 코어, 흐름 제어, XML 변환	x	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>

# # JSTL (JSP 표준 태그 라이브러리)

- if, switch 문은 <c:if>, <c:choose><c:when> 태그로 대체 가능

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

Java	JSTL
if (score > 60) { out.print("통과"); }	<c:if test="\${score > 60 }"> 통과 </c:if>
if (score > 90) { out.print("A"); } else if (score > 80) { out.print("B"); } else { out.print("C"); }	<c:choose> <c:when test="\${score > 90}"> A </c:when> <c:when test="\${score > 80}"> B </c:when> <c:otherwise> C </c:otherwise> </c:choose>

# # JSTL (JSP 표준 태그 라이브러리)

- for, while 문은 <c:foreach> 태그로 대체 가능

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

Java	JSTL
for (int i=0; i<=4; i++){ out.print(i); }	<c:forEach begin="0" end="4" step="1" var="i"> \${i} </c:forEach>
for (int item : list){ out.print(item); }	<c:forEach items="\${list}" var="item"> \${item} </c:forEach>
for (int i=0; i<list.length; i++){ int item = list[i]; out.print(i + " : " + item); }	<c:forEach items="\${list}" var="item" varStatus="i"> \${i.index} : \${item} </c:forEach>

# # JSTL (JSP 표준 태그 라이브러리)

- <c:out> 태그를 사용하여 출력하면 보안 상 유리

- <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
- \${str} → <c:out value="\${str}"></c:out>
- 출력내용에 HTML 태그가 포함된 경우, 태그로 해석되지 않고 문자 그대로 화면에 출력

- <c:url> 태그를 사용하여 주소를 출력하면 상대적으로 편리

- <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
- value에 지정한 주소가 "/"로 시작하는 경우, 컨텍스트 경로가 자동으로 앞에 추가

Java	out.print( request.getContextPath() + "/m/del.do?id=" + a + "&pw=" + b );
EL	\${pageContext.request.contextPath }/m/del.do?id=\${a}&pw=\${b}
JSTL	<c:url value="/m/del.do"> <c:param name="id" value="\${a}" /> <c:param name="pw" value="\${b}" /> </c:url>

# # JSTL (JSP 표준 태그 라이브러리)

- <c:import>로 외부 URL의 내용을 읽어와서 현재 위치에 삽입(출력)

- <c:import url="/m/list.jsp"></c:import>
  - request.getRequestDispatcher("/m/list.jsp").include(request, response);
  - <jsp:include page="/m/list.jsp"></jsp:include>
  - <%@ include file="/m/list.jsp" %>
- <c:import>는 동일한 웹 애플리케이션 뿐만 아니라 외부의 다른 자원을 읽어와 포함 가능
  - <jsp:include>는 동일한 웹 애플리케이션 내에 위치한 자원만 포함 가능
  - <%@ include %>는 동일한 웹 애플리케이션 내에 위치한 정적인 자원만 포함 가능
  - <c:param> 태그로 파라미터 전달 가능 (파라미터의 인코딩은 현재 파일의 contentType에 따라 자동 수행)

- sendRedirect() 문은 <c:redirect>로 대체 가능

- response.sendRedirect( request.getContextPath() + "/m/list.do");
- <c:redirect url="/m/list.do"></c:redirect>
- url에 지정한 주소가 "/"로 시작하는 경우, 컨텍스트 경로가 자동으로 앞에 추가

# # JSTL (JSP 표준 태그 라이브러리)

- 날짜/시간 객체 <-> 날짜/시간 문자열 상호 변환

- 숫자 <-> 숫자 문자열 상호 변환

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

일시 → 문자열	Java	Date d = new Date(); out.print( new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(d) );
	JSTL	<fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="\${d}" />
문자열 → 일시	Java	Date d = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").parse("2014/04/16 12:34:56");
	JSTL	<fmt:parseDate pattern="yyyy/MM/dd HH:mm:ss" value="2014/04/16 12:34:56" var="d" />
숫자 → 문자열	Java	double v = 1234.56; out.print( new DecimalFormat("000,000.000").format(v) );
	JSTL	<fmt:formatNumber pattern="000,000.000" value="\${v}" />
문자열 → 숫자	Java	Number v = new DecimalFormat("###,###.###").parse("1234.56");
	JSTL	<fmt:parseNumber pattern="###,###.###" value="1,234.56" var="v" />

# # JSTL (JSP 표준 태그 라이브러리)

- EL에서 \${fn:함수명() } 과 같은 형태로 사용할 수 있는 함수를 제공

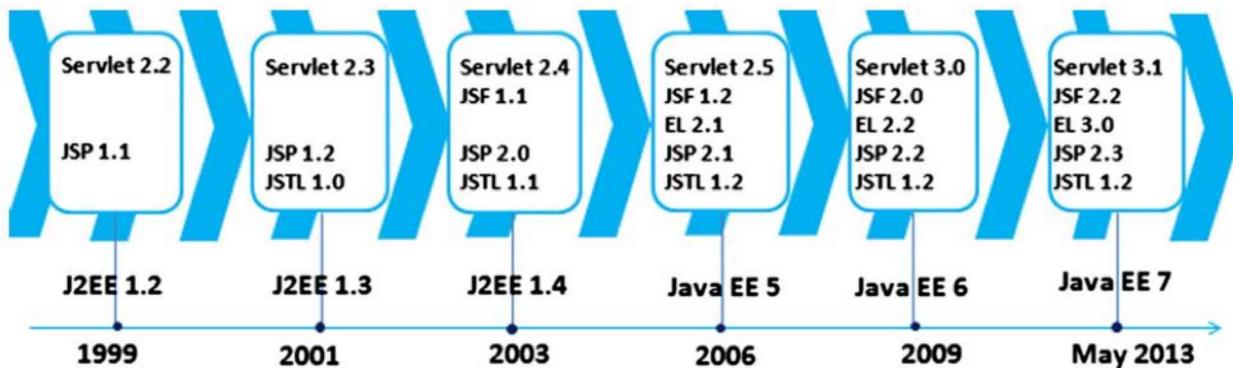
- <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

JSTL	JAVA	JSTL	JAVA
\${fn:contains("str1", "str2")}	"str1".contains("str2")	\${fn:startsWith("str1", "str2")}	"str1".startsWith("str2")
\${fn:containsIgnoreCase("str1", "str2")}	"str1".toLowerCase().contains("str2".toLowerCase())	\${fn:endsWith("str1", "str2")}	"str1".endsWith("str2")
\${fn:escapeXml("xmlContainingText")}	XML 태그 특수문자처리 (< → &lt; , > → &gt; )	\${fn:replace("str1", "str2", "str3")}	"str1".replaceAll("str2", "str3")
\${fn:length(문자열또는컬렉션)}	array.length   string.length()   collection.size()	\${fn:indexOf("str1", "str2")}	"str1".indexOf("str2")
\${fn:join(배열, "str1")}	배열의 요소를 "str1"을 끼워넣고 이어붙여서 하나의 문자열로	\${fn:split("str1", "str2")}	"str1".split("str2")
\${fn:substring("str1",index1,index2)}	"str1".substring(index1,index2);	\${fn:toLowerCase("str1")}	"str1".toLowerCase()
\${fn:substringAfter("str1", "str2")}	"str1".substring("str1".indexOf("str2")+"str2".length());	\${fn:toUpperCase("str1")}	"str1".toUpperCase()
\${fn:substringBefore("str1", "str2")}	"str1".substring(0, "str1".indexOf("str2"));	\${fn:trim("str1")}	"str1".trim()

# # Version History

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	JASPI Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.20	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.41	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.94	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later

<https://tomcat.apache.org/whichversion.html>



Learn Java for Web Development: Modern Java Web Development (Vishal Layka, Apress)

## References

- Oracle JavaServer Pages Developer's Guide and Reference 8.1.7
  - [https://docs.oracle.com/cd/A97336\\_01/buslog.102/a83726/genlovw3.htm](https://docs.oracle.com/cd/A97336_01/buslog.102/a83726/genlovw3.htm)
- JavaServer Pages Standard Tag Library 1.1 Tag Reference
  - <https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/index.html>
- Java EE 8 JSTL API Reference
  - <https://javaee.github.io/javaee-spec/javadocs/index.html?javax/servlet/jsp/jstl/core/package-summary.html>
- w3processing
  - <http://www.w3processing.com/index.php?subMenuLoad=JSP/JSTL/CoreJSTL.php&environmentPath=NB/GF>
- JSP Tutorial
  - <https://www.javatpoint.com/jsp-action-tags-forward-action>