

표현 언어와 커스텀 태그

524730-1
2019년 봄학기
5/20/2019
박경신

표현 언어

- 표현 언어 (Expression Language)
 - JSP에서 사용 가능한 새로운 스크립트 언어
 - JSP의 PAGE, REQUEST, SESSION, APPLICATION 영역에 저장된 속성에 사용
 - 수치 연산, 관계 연산, 논리 연산자 제공
 - 자바 클래스 메서드 호출 기능 제공
 - 쿠키, 내장객체의 속성 등 JSP를 위한 표현언어의 내장 객체 제공
 - 람다식을 이용한 함수 정의와 실행 (EL3.0)
 - 스트림 API를 통한 컬렉션 처리 (EL3.0)
 - 정적 메소드 실행(EL3.0)
- 간단한 구문 때문에 표현식 `<%= %>` 대신 사용
 - `<!-- 표현식 -->`
 - `<%= member.getAddress().getZipCode() %>`
 - `<!-- 표현언어 -->`
 - `$(member.address.zipCode)`

표현 언어 구문

- 표현언어는 JSP 스크립트 요소(스크립트릿, 표현식, 선언부)를 제외한 나머지 부분에서 사용
- 기본 문법
 - `#{expr}`은 표현식이 실행되는 시점에 바로 값 계산
 - `<jsp:include page="/module/${skin.id}/header.jsp" flush="true" />`
 - `${sessionScope.member.id}`님 환영합니다.
 - `#{expr}`은 값이 실제로 필요한 시점에 값 계산
 - JSP 템플릿 텍스트에서는 사용 불가

EL 사용 예시

- `<jsp:setProperty>` 태그에 EL 사용
 - `<jsp:setProperty name="box" property="perimeter" value="${2*box.width + 2*box.height}"/>`
- `<jsp:text>` 태그에 EL사용
 - `<jsp:text>`
 - Box 둘레: `$(2*box.width + 2*box.height)`
 - `</jsp:text>`

EL 자료형

- Boolean 타입
 - true 와 false
- 정수 타입
 - 0~9로 이루어진 정수 값
- 실수 타입
 - 0~9로 이루어져 있으며, 소수점('.')을 사용할 수 있고, 3.24e3과 같이 지수형으로 표현 가능
- 문자열 타입
 - 따옴표(' 또는 ")로 둘러싼 문자열.
 - 작은 따옴표 사용시, 값에 포함된 작은 따옴표는 \'로 입력
 - \ 기호 자체는 \\로 표시
- 널 타입
 - null

EL 연산자

- 수치 연산자
 - +, -, *, / (또는 div), % (또는 mod)
- 비교 연산자
 - == (또는 eq), != (또는 ne), < (또는 lt), <= (또는 le), > (또는 gt), >= (또는 ge)
- 논리 연산자
 - && (또는 and), || (또는 or), ! (또는 not)
- empty 연산자
 - empty <값>
 - 값이 null이면, true
 - 값이 빈 문자열("")이면, true
 - 값의 길이가 0인 배열이나 컬렉션이면 true
 - 이 외의 경우에는 false
- 비교 선택 연산자
 - <수식> ? <값1> : <값2>

EL 내장 객체

| EL 내장 객체 | 설명 |
|------------------|--|
| pageContext | JSP의 page 기본 객체와 동일. servletContext, session, request, response 등의 여러 객체를 참조 가능 |
| pageScope | pageContext 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체. \${pageScope.속성}으로 값을 참조 |
| requestScope | request 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체. \${requestScope.속성}으로 값을 참조 |
| sessionScope | session 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체. \${sessionScope.속성}으로 값을 참조 |
| applicationScope | application 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체. \${applicationScope.속성}으로 값을 참조 |
| param | 요청 파라미터의 <파라미터 이름, 값>을 저장한 Map 객체. \${param.이름}은 request.getParameter(이름)을 대체 |
| paramValues | 요청 파라미터의 <파라미터 이름, 값 배열>을 저장한 Map 객체. request.getParameterValues() 처리와 동일 |
| header | 요청 정보의 <헤더 이름, 값>을 저장한 Map 객체. \${header["이름"]}은 request.getHeader(헤더 이름)과 같음 |
| headerValues | 요청 정보의 <헤더 이름, 값 배열>을 저장한 Map 객체. request.getHeaders()의 처리와 동일 |
| cookie | <쿠키 이름, Cookie>을 저장한 Map 객체. request.getCookies()의 Cookie 배열의 이름과 값으로 Map을 생성 |
| initParam | 초기화 파라미터의 <이름, 값>을 저장한 Map 객체. \${initParam.이름}은 application.getInitParameter(이름) 대체 |

EL에서 객체에 접근

- **`\${<표현1>.<표현2>}`** 또는 **`\${<표현1>[<표현2>]}`** 형식 사용
- 처리 과정
 1. <표현1>을 <값1>로 변환
 2. <값1>이 null이면 null을 반환
 3. <값1>이 null이 아닐 경우 <표현2>를 <값2>로 변환
 1. <값2>가 null이면 null을 반환
 2. <값1>이 Map, List, 배열인 경우
 1. <값1>이 Map이면
 1. <값1>.containsKey(<값2>)가 false이면 null을 반환
 2. 그렇지 않으면 <값1>.get(<값2>)를 반환
 2. <값1>이 List나 배열이면
 1. <값2>가 정수 값인지 검사 (정수 값이 아닐 경우 에러 발생)
 2. <값1>.get(<값2>) 또는 Array.get(<값1>, <값2>)를 반환
 3. 위 코드가 예외를 발생하면 에러를 발생
 5. <값1>이 다른 객체이면
 1. <값2>를 문자열로 변환
 2. <값1>이 이름이 <값2>이고 읽기 가능한 프로퍼티를 포함하고 있다면 프로퍼티의 값을 반환
 3. 그렇지 않을 경우 에러를 발생

EL에서 클래스 메서드 호출하기

- 클래스에 정의한 메소드를 표현언어로 호출하려면
 - `${ prefixname:functionname() }`
 - 먼저 접두어 **prefixname**으로 태그를 선언
- EL에서 함수를 이용하려면 다음 3 가지 작업을 수행
 1. 클래스 작성
 2. EL 함수를 정의한 TLD(Tag Library Descriptor) 파일 작성 & web.xml 파일에 TLD 파일 지정
 3. JSP 코드에서 TLD에 정의한 함수 실행

| 작업 | 파일이 저장되는 폴더 | 파일 이름 |
|-----------|------------------------------|-----------------|
| 클래스 작성 | [Java Resources: src]/[패키지] | ELDateUtil.java |
| TLD 파일 작성 | [WebContent]/[WEB-INF]/[tld] | ELfunction.tld |
| JSP 파일 작성 | [WebContent] | function.jsp |

EL에서 클래스 메서드 호출하기

- 클래스의 static 메서드를 EL에서 호출 가능

```
public class ELDateUtil {  
    public static String format(Date date) {  
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");  
        return formatter.format(date);  
    }  
}
```

EL 함수를 정의한 TLD 파일

```
<?xml version="1.0" encoding="euc-kr" ?> <!-- XML선언문 -->  
<taglib ... version="2.1" > <!-- tag 정보 -->  
    <description>EL에서 함수실행</description>  
    <tlib-version>1.0</tlib-version>  
    <short-name>ELfunction</short-name>  
    <function>  
        <description>Date 객체 포매팅</description>  
        <name>dateFormat</name> <!-- EL에서 사용될 함수명 -->  
        <function-class>el.ELDateUtil</function-class>  
        <function-signature>  
            java.lang.String format(java.util.Date)  
        </function-signature>  
    </function>  
</taglib>
```

TLD 파일

- TLD (Tag Library Descriptor)
 - 사용자 정의 태그에 대한 설명을 담고 있는 파일
 - TLD 구성
 - XML 선언문
 - DTD 지정
 - 태그 정보 - 사용자정의 태그 기본 정보 기술
 - 사용자 정의 태그 - JSP에서 사용할 사용자 정의 태그항목 기술

TLD 파일

□ 태그 정보

- 사용자정의 태그 기본 정보 기술

| 태그 | 설명 |
|----------------|-------------------------------|
| <tlib-version> | 태그 라이브러리 버전으로 개발자에 의해 설정되는 부분 |
| <jsp-version> | 적용되는 JSP 버전 |
| <short-name> | 현재 태그 라이브러리의 간단한 이름 |
| <uri> | taglib을 고유한 것으로 만들어주는 URI 정보 |
| <display-name> | 출력에 사용되는 taglib 이름 |
| <tag> | 사용자 정의 태그가 위치하는 부분 |
| <description> | 태그 라이브러리에 대한 설명이 들어가는 부분 |

TLD 파일

□ 사용자 정의 태그

- JSP에서 사용할 사용자 정의 태그항목 기술

| 태그 | 설명 |
|----------------|--|
| <name> | 태그 이름으로 사용자 정의 태그 핸들러와 연결 |
| <tag-class> | 사용자 정의 태그 핸들러 클래스 |
| <body-content> | 태그 내에 다른 태그를 가질 것인지, JSP 문장이 올 수 있는지 지정. empty : body가 없는 태그로 <xxx /> 형태로 사용. JSP : body 내용에 JSP 코드가 올 수 있음. <xxx> JSP 코드 </xxx> tag-dependent : 태그 핸들러에 의해 처리. 또 다른 사용자 정의 태그 올 수 있음. |

web.xml 파일에 TLD 파일 지정

```
<web-app ... version="2.5">
  <jsp-config>
    <taglib>
      <taglib-uri>
        /WEB-INF/tlds/ELfunction.tld JSP에서 사용될 URI
      </taglib-uri>
      <taglib-location>
        /WEB-INF/tlds/ELfunction.tld TLD 파일 경로
      </taglib-location>
    </taglib>
  </jsp-config>
</web-app>
```

JSP에서 EL 함수 호출

```
<%@ taglib prefix="elfunc" uri="/WEB-INF/tlds/ELfunction.tld" %>
<%
  java.util.Date today = new java.util.Date();
  request.setAttribute("today", today);
%>
<html>
<head> <title>EL 함수 호출</title> </head>
<body>
  오늘은 <b>${elfunc:dateFormat(today)}</b> 입니다.
```

↓
prefix:TLD에정의된함수명()

EL 비활성화

□ web.xml 파일에 비활성화 옵션 설정

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>/oldversion/*</url-pattern>
    <el-ignored>true</el-ignored>
  </jsp-property-group>
</jsp-config> * <deferred-syntax-allowed-as-literal>를 이용한 #{expr}을 문자열로 처리
```

□ page 디렉티브에서 설정

- isElIgnored 속성을 true로 하면 EL 무시
- deferredSyntaxAllowedAsLiteral 속성을 true로 하면 #{expr}을 문자열로 처리

□ web.xml 파일 버전에 따라 자동 비활성화

- 서블릿 2.3 버전 : EL 지원하지 않음
- 서블릿 2.4 버전 : \${expr} 형식의 EL 지원

EL 용법

- Request 나 session 속성으로 전달한 값을 출력
- 액션 태그나 커스텀 태그의 속성 값
 - <jsp:include page="/lo/\${layout.module}.jsp" flush="true"
- 함수의 호출
 - 코드의 간결함 및 가독성 향상

커스텀 태그(Custom Tag)

```
<%
while(iter.hasNext()) {
  HashMap data = (HashMap)iter.next();
  %>
  <form name=<%=data.get("gb_id") %> action=guestbook_edit_form.jsp method=post>
  <input type=hidden name="gb_id" value="<%=data.get("gb_id") %>"
  <table cellpadding=5 cellspacing=0 border="1" width=400>
    <tr>
      <td bgcolor="#99CCFF" height="23" >작성 자</td>
      <td <%= data.get("gb_name") %></td>
      <td bgcolor="#99CCFF" height="23" >작성 일</td>
      <td <%= data.get("gb_date") %></td>
    </tr>
    <tr>
      <td bgcolor="#99CCFF" height="23">email</td>
      <td colspan=3><%= data.get("gb_email") %></td>
    </tr>
    <tr>
      <td colspan=4><%= data.get("gb_contents") %></td>
    </tr>
    <tr>
      <td colspan=4>비밀번호 <input type=password name=gb_passwd size=6>
      <input type=submit value="수정/취소" ></td>
    </tr>
  </table>
  </form>
  <p>
  <%
  } // end while
```

사용자 정의 태그 사용 전

사용자 정의 태그 사용 후 /WEB-INF/tags/getList.tag

```
<%taglib prefix="tf" tagdir="/WEB-INF/tags" %>
<html>
<body>
  <tf:getList />
</body>
</html>
```

커스텀 태그(Custom Tag)

- JSP 액션 태그와 같은 태그를 추가할 수 있는 기능
 - JSTL도 커스텀 태그 라이브러리의 한 종류
- 커스텀 태그의 장점
 - 재사용성 향상 - 한번 작성한 커스텀 태그는 JSP 컨테이너에서 사용 가능
 - 쉽고 단순한 JSP 제작 - 커스텀 태그를 사용하여 쉽게 JSP 작성
 - 코드 가독성 향상 - 스크립트 코드를 줄일 수 있기때문
- 커스텀 태그의 종류
 - JSP 1.2 스타일 커스텀 태그
 - JSP 2.0 이상의 **SimpleTag**를 사용한 커스텀 태그
 - JSP 2.0 이상의 **태그 파일**을 사용한 커스텀 태그 (구현이 쉬움)

태그 파일(tag file)

- JSP 문법을 사용해서 커스텀 태그로 동작할 수 있도록 만들어진 소스 코드
- 서블릿보다 JSP가 작성하기 쉽듯이, 클래식 태그나 SimpleTag보다 태그 파일이 작성하기 쉬움
- JSP와 동일한 방식으로 태그 파일 작성
- JSP 소스코드를 서블릿 클래스로 변환 하듯이, 태그 파일을 커스텀 태그 클래스로 변환
- 커스텀 태그의 표준 인터페이스를 구현하거나 클래스를 상속받아서 구현
- 소스코드 상태 그대로 설치해야 하므로 소스코드가 공개
- taglib 지시어를 사용해서 태그 파일에서 다른 커스텀 태그를 사용할 수 있음

태그 파일의 위치 및 참조

- 위치 및 확장자
 - WEB-INF/tags 디렉터리 및 하위 디렉터리에 위치
 - 확장자 .tag 또는 .tagx를 갖는 파일만 태그 파일로 인식
 - 태그 파일의 이름은 커스텀 태그의 이름이 됨
 - uri 속성 대신 tagdir 속성을 사용. 태그 파일이 위치한 디렉터리의 경로를 입력하고 해당 디렉터리에 있는 태그 파일의 이름은 각각 하나의 커스텀 태그 이름이 됨
- JSP에서 태그 파일 참조


```
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags/util" %>
...
<tf:removeHtml ...>...</tf:removeHtml>
```

 - /WEB-INF/tags/util 디렉터리의 **removeHtml.tag** 파일을 **<tf:removeHtml>** 태그의 구현 코드로 사용

태그 파일에서 사용할 수 있는 지시어

| 디렉티브 | 설명 |
|------------------|--|
| tag | JSP 페이지의 page 지시어와 동일. tag 지시어는 태그 파일의 정보를 명시 |
| taglib | JSP 페이지와 마찬가지로 태그 파일에서 사용할 태그 라이브러리를 명시할 때 사용 |
| include | JSP 페이지와 마찬가지로 태그 파일에 특정한 파일을 포함시킬 때 사용. 태그 파일에 포함되는 파일은 태그 파일에 알맞은 문법으로 작성되어야 함 |
| attribute | 태그 파일이 커스텀 태그로 사용될 때 입력 받을 속성 을 명시 |
| variable | EL 변수로 사용될 변수 에 대한 정보를 지정 |

tag 디렉티브의 주요 속성

| 속성 | 설명 |
|-------------------------------------|---|
| display-name | 태그 파일을 도구에서 보여줄 때 사용될 이름 지정 |
| body-content | 몸체 내용의 종류를 지정. empty, tagdependent, scriptless 중 한가지 사용. 기본값은 scriptless |
| dynamic-attributes | 동적 속성을 사용할 때, 속성의 <이름,값>이 저장될 Map 객체 를 page 범위의 속성에 저장할 때 사용할 이름을 명시. EL에서 변수이름으로 사용가능 |
| import | page 지시어의 import 속성과 동일 (선택) |
| pageEncoding | page 지시어의 pageEncoding 속성과 동일 (선택) |
| isELIgnored | page 지시어의 isELIgnored 속성과 동일 (선택) |
| deferredSyntaxAllowedAsLiteral(2.1) | page 지시어의 deferredSyntaxAllowedAsLiteral 속성과 동일 (선택) #{ 문자가 문자열 값으로 사용되는 것을 허용 여부를 지정 |
| trimDirectiveWhitespaces(2.1) | page 지시어의 trimDirectiveWhitespaces 속성과 동일 (선택) 출력 결과에서 템플릿 텍스트의 공백 문자를 제거할지의 여부를 지정 |

태그 파일의 내장 객체

- jspContext
 - pageContext가 제공하는 setAttribute(), getAttribute() 메서드를 그대로 제공하며, 각 속성과 관련된 작업을 처리
- request
 - JSP 페이지의 request 기본 객체와 동일
- response
 - JSP 페이지의 response 기본 객체와 동일
- session
 - JSP 페이지의 session 기본 객체와 동일
- application
 - JSP 페이지의 application 기본 객체와 동일
- out
 - JSP 페이지의 out 기본 객체와 동일

attribute 지시어의 기본 사용법

- attribute 지시어의 기본 사용법
 - name : 속성 이름. 태그 파일에서 변수명으로 사용가능
 - required : 속성의 필수 여부를 지정 (기본값은 false)
 - type : 속성 값의 타입을 지정 (기본값은 java.lang.String)

```
<!-- header.tag -->
<%@ tag ... %>
<%@ attribute name="title" required="true" %>
<%@ attribute name="level" type="java.lang.Integer" %>
... // level에 따른 startTag=" <h1>" & endTag=" </h1>" 셋팅
<%=startTag %>${title}<%= endTag%>
```
- JSP에서의 사용법

```
<tf:header title="<%= article.getTitle() %>" level="1" />
<tf:header title="${article.title}" />
<tf:header title="이 글의 제목" />
```

<jsp:attribute> 액션 태그 이용 속성 값 전달

- 태그 파일의 attribute 지시어의 fragment 속성이 true 경우
 - JSP에서 속성에 값을 전달할 때 <jsp:attribute> 사용해야함
 - 태그 파일에서는 <jsp:invoke>를 이용해서 설정한 속성 값 사용

```
<!-- header.tag -->
<%@ attribute name="title" fragment="true" %>

// 첫 번째, <jsp:attribute>의 몸체 내용을 그대로 처리하여 출력
<jsp:invoke fragment="title" />
// 두 번째, <jsp:attribute>의 몸체 내용을 처리한 결과를
// 지정한 영역의 속성에 저장
<jsp:invoke fragment="title" var="rs" scope="page" />
${pageScope.rs}
```
- <jsp:attribute> 액션 태그로 값 지정

```
<tf:header>
  <jsp:attribute name="title">${article.title}</jsp:attribute>
</tf:header>
```

동적 속성

- attribute 디렉티브로 지정하지 않은 속성을 Map에 저장
 - 상황에 따라서 동적으로 속성을 추가할 때 사용
 - tag 디렉티브의 dynamic-attributes 속성에 동적 속성을 저장할 변수명 지정 (변수 타입은 Map)
 - 동적 속성들은 java.util.Map 타입의 EL 변수에 저장되므로 dynamic-attributes 속성에서 명시한 Map 이름으로 Map EL 변수를 통해서 접근 가능
- ```
<%@ tag dynamic-attributes="dynamicMap" %>
...
${dynamicMap.attrName} .. <!-- attrName 속성값 -->
```

## 태그 파일 이용 커스텀 태그에 몸체 내용 전달하기

### □ 방법1 - 태그 몸체

```
<tf:someTagFile attr1="속성값">
 여기에 몸체 내용을 입력한다.
</tf:someTagFile>
```

### □ 방법2 - <jsp:body> 액션 태그 사용

- 태그 파일 이용 커스텀 태그의 몸체에는 표현식 사용 불가

```
<tf:someTagFile attr1="속성값">
 <jsp:attribute name="attr2">value</jsp:attribute>
 <jsp:body>
 여기에 몸체 내용을 입력한다.
 </jsp:body>
</tf:someTagFile>
```

## 태그 파일에서 몸체 내용 사용하기

### □ EL 및 태그가 처리된 몸체 내용 사용하기

```
<%@ tag body-content="scriptless" %>
<!-- 몸체로 전달 받은 내용을 사용 -->
<jsp:doBody />
```

### □ 몸체 내용을 데이터로 사용하기

```
<%@ tag body-content="tagdependent" pageEncoding="euc-kr" %>
...
<jsp:doBody var="bodyText" />
```

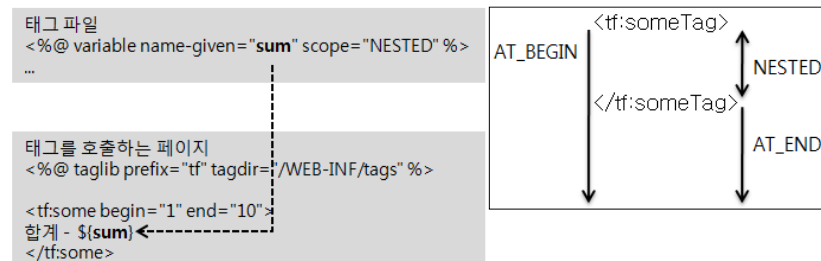
## variable 지시어와 name-given을 이용한 변수 추가

### □ variable 지시어를 사용해서 태그 파일을 사용하는 JSP/태그 파일에서 사용할 EL 변수 추가

### □ variable 지시어의 기본 문법

```
<%@ variable name-given="EL변수이름"
 variable-class="변수타입"
 scope="변수범위" />
```

- name-given 속성 및 scope 속성에 따른 EL 변수 특징



## name-from-attribute 속성을 이용한 변수 생성

### □ 특정 속성의 값을 생성할 변수명으로 사용

### □ name-from-attribute 속성의 사용법

```
<%@ attribute name="var" rtexprvalue="false"
 required="true" type="java.lang.String" %>
<%@ variable alias="localName" name-from-attribute="var"
 scope="영역" %>
...
<c:set var="localName" value="..." />
```

- name-from-attribute 속성에 명시한 속성의 제약 조건
  - rtexprvalue 속성의 값이 false
  - required 속성이 true
  - type 속성의 값은 java.lang.String
- alias 속성 : 태그 파일에서 생성할 변수에 접근할 때 사용할 EL 변수명



## 변수 동기화 처리

---

|                      | AT_BEGIN  | NESTED    | AT_END    |
|----------------------|-----------|-----------|-----------|
| 태그 시작                | 아무것도 안 함  | EL 변수 저장  | 아무것도 안 함  |
| <jsp.doBody><br>실행 전 | 태그→페이지 복사 | 태그→페이지 복사 | 아무것도 안 함  |
| 태그 끝                 | 태그→페이지 복사 | EL 변수 복구  | 태그→페이지 복사 |