11장 표현 언어(Expression Language)

11.1 표현 언어란?

- JSP에서 사용가능한 스크립트 언어

- EL의 주요 기능
  - JSP의 네 가지 기본 객체가 제공하는 영역의 속성 사용
  - 집합 객체에 대한 접근 방법 제공
  - 수치 연산, 관계 연산, 논리 연산자 제공
  - 자바 클래스 메서드 호출 기능 제공
  - 표현언어만의 기본 객체 제공

- 표현 언어는 표현식보다 간결하고 편리하기 때문에 많이 사용한다.

```
<%-- 표현식 --%>
<%= member.getAddress().getZipcode() %>

<%-- 표현 언어 --%>
${member.address.zipcode}
```

11.1.1 EL의 구성

- ${expr} 형식은 구문을 분석할 때 곧바로 값을 계산한다.

- #{expr} 형식은 실제로 값이 사용될 때 값을 계산한다.
  - JSP 템플릿 텍스트에서는 사용 불가 (잘 사용되지 않음)

```
<%-- 기본형 --%>
${expr}, #{expr}

<%-- EL의 예시--%>
<jsp:include page="/module/${skin.id}/header.jsp" flush="true" />

<b>${sessionScope.mememeber.id}</b>님 환영합니다.

<%
    Member m = new Member();
    m.setName("이름1");
%>

<c:set var="m" value="<%= m %>" />
<c:set var="name" value="${m.name}" /> <%-- 이 시점에는 바로 값 계산 --%>
<% m.setName("이름2"); %>
${name}  <%-- name의 값은 "이름1" --%>

<c:set var="m" value="<%= m %>" />
<c:set var="name" value="#{m.name}" /> <%-- 이 시점에는 값 생성하지 않음 --%>
<% m.setName("이름2"); %>
${name} <%-- 구문을 분석할 때 값 계산, "이름2" 출력 --%>

<%-- JSP의 템플릿 텍스트에서 #{expr}을 사용하면 에러 발생 --%>
#{sessionScope.name.id}님 환영합니다.
```

11.2 EL 기초

11.2.1 EL의 데이터 타입과 리터럴

- EL은 불리언(Boolean) 타입, 정수 타입, 실수 타입, 문자열 타입, 그리고 널 타입의 5가지 타입을 제공한다.

```
${10}은 정수, ${10.1}은 실수
```

11.2.2 EL의 기본 객체

기본 객체	설명
pageContext	JSP의 pageContext 기본 객체와 동일하다.
pageScope	pageContext 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map 객체이다.
requestScope	request 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map 객체이다.
sessionScope	session 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map 객체이다.
applicationScope	application 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map 객체이다.
param	요청 파라미터의 <파라미터 이름, 값> 매핑을 저장한 Map 객체이다. 파라미터 값 타입은 String 이며, request.getParameter(이름)의 결과와 동일하다.
paramValues	요청 파라미터의 <파라미터 이름, 값> 매핑을 저장한 Map 객체이다. 파라미터 값 타입은 String 이며, request.getParameterValues(이름)의 결과와 동일하다.
header	요청 정보의 <헤더 이름, 값> 매핑을 저장한 Map 객체이다. request.getHeader(이름)의 결과와 동일하다.
headerValues	요청 정보의 <헤더 이름, 값 배열> 매핑을 저장한 Map 객체이다. request.getHeaders(이름)의 결과와 동일하다.
cookie	<쿠키 이름, Cookie> 매핑을 저장한 Map 객체이다. request.getCookies()로 구한 Cookie 배열로부터 매핑을 생성한다.
initParam	초기화 파라미터의 <이름, 값> 매팅을 저장한 Map 객체이다.

[chap11/useELObject.jsp]

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%
          request.setAttribute("name", "최범균");
%>
<html>
<head><title>EL Object</title></head>
<body>

요청 URI: ${pageContext.request.requestURI}<br>
request의 name 속성: ${requestScope.name}<br>
code 파라미터: ${param.code}

</body>
</html>
```

11.2.3 객체 접근

- EL 언어는 객체에 저장된 값에 접근할 때 점(.)이나 대괄호([])를 사용한다.

```
${coockie.ID.value}
${coockie.name} == ${cookie['name']}
```

11.2.4 객체 탐색

```
${pageScope.NAME}
${name} <%-- page,request,session,application 영역을 차례대로 검사해서 이름이 name의 속성값 사용 --%>
```

11.2.5 수치 연산자

```
${"10" + 1 } <%-- 숫자 11 --%>
${"일" + 10} <%-- 에러 발생 --%>
${null + 1 } <%-- 1 --%>
```

11.2.6 비교 연산자

```
${someValue == '2004'}
<%-- someValue.compareTo('2004') ==0과 같은 의미로 객체가 같은 값을 갖는지를 비교한다. --%>
```

11.2.7 논리 연산자

■ 자바 언어의 논리 연산자와 완전히 동일하다.

11.2.8 empty 연산자

■ 검사할 객체가 텅 빈 객체인지를 검사하기 위해 사용한다.

11.2.9 비교 선택 연산자

■ <수식>의 결과값이 true이면 <값1>을 리턴하고, false이면 <값2>를 리턴한다.

```
<수식> ? <값1> : <값2>
```

11.2.10 문자열 연결

■ EL 3.0 버전에 문자열 연결을 위한 += 연산자가 추가됐다.

```
${"문자" += "열" += "연결"}  --> "문자열연결"
```

11.2.11 컬렉션

■ EL 3.0 버전부터 EL 식에서 직접 List, Map, Set 타입의 객체를 생성할 수 있게 되었다.

```
# EL에서 List 타입 객체를 생성할 때는 [원소1, 원소2, 원소3] 구문을 사용한다.
<c:set var="vals" values="${[1,2,5,10]}" />
${vals[2]} --> 5

<c:set var="mem" value="${ {'name':'홍길동','age':20} }" />
${mem.name}, ${mem.age}

# Set 타입의 EL 객체를 생성할 때는 {원소1, 원소2, 원소3} 구문을 사용한다.
<c:set var="hangul" value="${ {'가','나','다'} }" />
${hangul}

# Map과 List를 혼합해서 생성할 수도 있다.
<c:set var="codes" value="${ [ {'code':'001','label':'1번'}, {'code':'001','label':'1번'} ] }" />
${codes[0].code / ${codes[0].label}
```

11.2.12 세미콜론 연산자

■ EL 3.0에 추가된 연산자로서, 세미콜론 연사자를 사용하면 다음과 같이 두 개의 식을 붙일 수 있다.

```
${1+1; 10+10} --> 20
${A;B} --> B
```

11.2.13 할당 연산자

```
${var1 = 10 } --> 10
```

11.3 EL에서 객체의 메서드 호출

[chap11/WEB-INF/src/chap11/Thermometer.java]

```
package chap11;

import java.util.HashMap;
import java.util.Map;

public class Thermometer {

          private Map<String, Double> locationCelsiusMap =
                              new HashMap<String, Double>();

          public void setCelsius(String location, Double value) {
                    locationCelsiusMap.put(location, value);
```

```
	}

	public Double getCelsius(String location) {
		return locationCelsiusMap.get(location);
	}

	public Double getFahrenheit(String location) {
		Double celsius = getCelsius(location);
		if (celsius == null) {
			return null;
		}
		return celsius.doubleValue() * 1.8 + 32.0;
	}

	public String getInfo() {
		return "온도계 변환기 1.1";
	}
}
```

[chap11/thermometer.jsp]

```
<%@page import="chap11.Thermometer"%>
<%@ page contentType="text/html; charset=utf-8"%>
<%
	Thermometer thermometer = new Thermometer();
	request.setAttribute("t", thermometer);
%>
<html>
<head>
	<title>온도 변환 예제</title>
</head>
<body>
	${t.setCelsius('서울', 27.3)}
	서울 온도: 섭씨 ${t.getCelsius('서울')}도 / 화씨 ${t.getFahrenheit('서울')}

	<br/>
	정보: ${t.info}
</body>
</html>
```

11.4 EL에서 정적 메서드 호출하기 1

- 클래스의 static 메서드를 EL에서 호출 가능
- EL에서 호출하려면 다음의 작업 필요
  - EL 함수 정의한 TLD(Tag Library Descriptor) 파일 작성
  - web.xml 파일에 TLD 파일 지정
  - JSP 코드에서 TLD에 정의한 함수 실행

11.4.1 예제에서 사용할 클래스 작성

- EL에서 클래스의 메서드를 사용하기 위해서는 클래스의 메서드를 static으로 정의해야 하며, static이 아닌 메서드는 사용할 수 없다.

[chap11/WEB-INF/src/util/FormatUtil.java]

```
package util;

import java.text.DecimalFormat;

public class FormatUtil {

          public static String number(long number, String pattern) {
                    DecimalFormat format = new DecimalFormat(pattern);
                    return format.format(number);
          }

}
```

11.4.2 함수를 정의할 TLD 파일 작성

```
<?xml version="1.0" encoding="euc-kr" ?>
<taglib … version="2.1">
    <description>EL에서 함수실행</description>
    <tlib-version>1.0</tlib-version>
    <short-name>ELfunctions</short-name>
    <function>
        <description>Date 객체 포맷팅</description>
        <name>dateFormat</name> <!-- EL에서 사용될 함수명 -->
        <function-class>kame.chap15.DateUtil</function-class>
        <function-signature>
            java.lang.String format(java.util.Date)
        </function-signature>
    </function>
</taglib>
```

11.4.3 web.xml 파일에 TLD 내용 추가하기

```
<web-app ... version="2.5">
    <jsp-config>
        <taglib>
            <taglib-uri>
                /WEB-INF/tlds/el-functions.tld
            </taglib-uri>
            <taglib-location>
                /WEB-INF/tlds/el-functions.tld
            </taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

11.4.4 EL에서 함수 사용하기

[chap11/viewNumber.jsp]

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ taglib prefix="elfunc" uri="/WEB-INF/tlds/el-functions.tld" %>
<%
          request.setAttribute("price", 12345);
%>
```

```
<html>
<head><title>EL 함수 호출</title></head>
<body>

가격은 <b>${elfunc:formatNumber(price, '#,##0') }</b>원 입니다.

</body>
</html>
```

11.5 EL에서 정적 메서드 호출하기 2

■ EL 3.0 버전은 복잡한 과정이 없이 정적 메서드를 호출할 수 있는 기능을 추가했다.

[chap11/viewNumber2.jsp]

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import="util.FormatUtil" %>
<%
        request.setAttribute("price", 12345L);
%>
<html>
<head><title>EL 함수 호출</title></head>
<body>

가격은 <b>${FormatUtil.number(price, '#,##0') }</b>원 입니다.

</body>
</html>
```

11.6 람다식 사용하기

```
${ greaterThan = (a,b) -> a>b ? true:false; }  <%—람다식을 greaterThan 변수에 할당한다. --%>
${ greaterThan(1,3) }  <%-- false --%>

${ factorial = (n) -> n == 1 ? 1 : n * factorial(n-1); }
${ factorial(5) }  <%-- 5 * 4 * 3 * 2 * 1 = 120 --%>
```

11.7 스트림 API 사용하기

```
<%-- JSTL의 반복문 --%>
<c:set var="list" value="<%= java.util.Arrays.asList(1,2,3,4,5) %>" />
<c:forEach var="val" items="${list}">
   <c:set var="sum" value="${sum + val}" />
</c:forEach>

<%-- JSTL + EL 3.0 버전의 스트림 API -->
<c:set var="list" value="<%= java.util.Arrays.asList(1,2,3,4,5) %>" />
<c:set var="sum" value="${list.stream().sum()}" />

<%-- EL만 사용 -->
${list = [1,2,3,4,5]; sum = list.stream().sum(); "}
```

11.7.1 스트림 API 기본

```
<%-- 기본형 --%>
collection.stream()      <%-- 콜렉션에서 스트림 생성 --%>
          .map(x -> x*x) <%-- 중간 연산(스트림 변환) --%>
          .toList()      <%-- 최종 연산(결과 생성) --%>

<%-- 예시 --%>
${list.stream()
     .filter(x -> x%2 == 0)
     .map(x -> x*x)
     .toList()}
```

11.7.2 stream()을 이용한 스트림 생성

```
${ list = [1,2,3,4,5]; " }
${ list.stream().sum() }
```

- Map에 stream()을 사용하고 싶다면 다음과 같이 Map.entrySet()과 같이 Map에서 컬렉션 타입 객체를 생성한 다음에 그 객체의 stream() 메서드를 사용하면 된다.

```
<%
   java.util.Map<String, String> map = new java.util.HashMap<>();
   map.put("code1", "코드1");
   map.put("code2", "코드2");
   request.setAttribute("map", map);
%>
${ map.entrySet().stream().map(entry -> entry.value).toList() }
```

11.7.3 filter()을 이용한 걸러내기

- filter() 메서드는 스트림의 각 원소에 대해 람다식을 실행하고 그 결과가 true인 원소를 제공하는 새로운 스트림을 생성한다.

```
<%-- 기본형 --%>
collection.stream()             <%-- 콜렉션이 [1,2,3,4,5]일 경우 --%>
          .filter(x -> x%2 == 0) <%-- 람다식이 true인 [2,4] 원소를 갖는 새로운 스트림 생성 --%>
          .toList()             <%-- filter()의 결과 스트림에서 새로운 List 생성 --%>
```

11.7.4 map()을 이용한 변환

- map() 메서드는 스트림의 각 원소에 대해 람다식을 실행하고 람다식의 결과로 구성된 새로운 스트림을 생성한다.

```

```

```
<%-- 기본형 --%>
collection.stream()      <%-- 콜렉션이 [1,2,3,4,5]일 경우 --%>
          .map(x -> x*x) <%-- [1,4,9,16,25]를 원소로 갖는 새로운 스트림 생성 --%>
          .toList()

<%-- 예시 --%>
${ ageList = members.stream().map(mem -> mem.age).toList(); " }
${ members.stream().map(mem -> mem.age).filter(x -> x>=20).average().get() }
${ members.stream().filter(m -> m.age>=20).map(m -> m.age).average().get() }
```

11.7.5 sorted()를 이용한 정렬

- sorted()를 사용하면 스트림을 정렬할 수 있다.

```
${ vals = [20,17,30,2,9,23];
   sortedVals = vals.stream().sorted().toList() }
```

11.7.6 limit()을 이용한 개수 제한

- limit()은 지정한 개수를 갖는 새로운 스트림을 생성한다.

```
${ nums.stream().sorted().limit(3).toList() }
```

11.7.7 toList()와 toArray()를 이용한 결과 생성

- toList()는 자바 리스트 객체를 생성하고, toArray()는 자바 배열 객체를 생성한다.

```
${ list = members.stream().map(m -> m.name).toList(); " }
${ arr = members.stream().map(m -> m.age).toArray(); " }
```

11.7.8 count()를 이용한 개수 확인

- count() 연산은 스트림의 원소 개수를 리턴한다.

```
${ members.stream().count() }
```

11.7.9 Optional 타입

- 결과값이 존재하거나 존재하지 않는 경우가 있을 때 사용하는 타입이 Optional이다.

```
${ [].stream().min().get() } <%-- 결과값이 없는 Optional이므로 ELException 발생 --%>
${ [].stream().min().orElse("없음") } <%-- 값이 없으므로 '없음' 결과 출력 --%>
```

```
${ [].stream().min().orElseGet(() -> -1) } <%-- 값이 없으므로 -1 결과 출력 --%>

${ minValue = '-';" }
${ [1,2,3].stream().min().ifPresent(x -> (minValue=x)) }
<%-- 최소값이 존재하면 해당 최소값을 출력하고, 존재하지 않으면 '-'를 리턴한다. --%>
```

11.7.10 sum()과 average()를 이용한 수치 연산 결과 생성

- 스트림이 숫자로 구성된 경우 sum()을 이용해서 합을 구할 수 있다. average()는 값의 평균을 구한다.

```
${ [1,2,3,4,5].stream().sum() }  <%-- 15 출력 --%>
${ [1,2,3,4,5].stream().average().get() } <%-- 2.5 출력 --%>
```

11.7.11 min()과 max()를 이용한 최소/최대 구하기

```
${ someLongVals.steam().min().get() }
```

11.7.12 anyMatch(), allMatch(), noneMatch()를 이용한 존재 여부 확인

- anyMatch()는 스트림에 조건을 충족하는 요소가 존재하는지 검사할 때 사용한다.

```
${ list = [1,2,3,4,5];" }
${ matchOpt = list.stream().anyMatch(x -> x>4);" }
${ matchOpt.get() } <%-- true --%>

${ list = [1,2,3,4,5];" list.stream().allMatch(x -> x>5).get() } <%-- false --%>
${ list = [1,2,3,4,5];" list.stream().noneMatch(x -> x>5).get() } <%-- true --%>
```

11.8 표현 언어 비활성화 방법

11.8.1 web.xml 파일에 EL 비활성화 옵션 추가하기

- <jsp-property-group> 태그의 <el-ignored> 태그의 값으로 true를 설정하면 JSP는 ${expr}과 #{expr} 형식의 EL을 모두 일반 문자로 처리한다.

[chap11/WEB-INF/web.xml]

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                              http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
                     version="3.1">

          <jsp-config>
                    <taglib>
```

```
                        <taglib-uri>
                                /WEB-INF/tlds/el-functions.tld
                        </taglib-uri>
                        <taglib-location>
                                /WEB-INF/tlds/el-functions.tld
                        </taglib-location>
                </taglib>

        <jsp-property-group>
            <url-pattern>/oldversion/*</url-pattern>
            <el-ignored>true</el-ignored>
        </jsp-property-group>

          </jsp-config>

</web-app>
```

11.8.2 JSP 페이지에서 EL 비활성화시키기

- page 디렉티브에서 설정
  - isELIgnored 속성을 true로 하면 EL 무시
  - deferredSyntaxAllowedAsLiteral 속성을 true로 하면 #{expr}을 문자열로 처리

```
<%-- EL을 비활성화 시키는 경우 --%>
<%@ page isELIgnored="true" %>

<%-- #{expr} 형식의 EL을 비활성화 시키는 경우 --%>
<%@ page deferredSyntaxAllowdAsLiteral="true" %>
```

11.8.3 web.xml 파일의 버전에 따른 EL 처리

- 서브릿 2.3 버전의 web.xml: EL을 지원하지 않는다.
- 서브릿 2.4 버전의 web.xml: #{expr}을 지원하지 않는다.
- 서브릿 2.5/3.x 버전의 web.xml: ${expr}와 #{expr}을 지원한다.