5강 자료

5강

표현 언어 Expression Language; EL

- 표현 언어가 제공하는 기능
  - JSP의 네 가지 기본 객체가 제공하는 영역의 속성 사용
  - 수치 연산, 관계 연산, 논리 연산 가능
  - 자바 클래스의 메서드 호출 가능
  - 쿠키, 기본 객체의 속성 등 JSP를 위한 표현 언어의 기본 객체 제공
  - 람다식을 이용한 함수 정의와 실행
  - 스트림 API를 통한 컬렉션 객체 처리 가능
  - 정적 메서드 실행 가능

표현 언어 Expression Language; EL

- EL의 구성
  - ${ }를 사용하여 값을 표현
  - 액션 태그에서 EL 사용
    - <jsp:include page="/module/${skin.id}/headjer.jsp" flush="true" />
  - 비스크립트 영역에서 EL 사용
    - ${sessionScope.member.id}님 환영합니다.
  - JSP의 스크립트 영역을 제외한 나머지 부분에서 사용 가능

```
<%
   String sessionId = session.getId();
   session.setAttribute("id", sessionId);
   Member m = new Member();
   m.setName("홍길동");    m.setAge(20);
   request.setAttribute("m", m);
%>
세션 아이디 확인1 ${pageScope.sessionId } <br />
세션 아이디 확인2 ${sessionScope. id } <br />
요청 URL은 ${pageContext.request.requestURL }<br />
객체의 값 확인 ${m.getName() } , ${m.getAge() }<br />
```

EL의 기본 객체

기본 객체	설명
pageContext	JSP의 page 기본 객체와 동일
pageScope	pageContext 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
requestScope	request 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
sessionScope	session 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
applicationScope	application 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
param	요청 파라미터의 <이름, 값> 매핑을 저장한 Map 객체 request.getParameter(이름);의 결과와 동일
paramValues	요청 파라미티의 <이름, 값> 매핑을 저장한 Map 객체 request.getParameterValues(이름);의 결과와 동일
header	요청 정보의 <헤더이름, 값> 매핑을 저장한 Map 객체 request.getHeader(이름);의 결과와 동일
headerValues	요청 정보의 <헤더이름, 값> 매핑을 저장한 Map 객체 Request.getHeaders(이름);의 결과와 동일

EL의 기본 객체

기본 객체	설명
cokiee	<쿠키 이름, 값> 매핑을 저장한 Map 객체 request.getCookies();로 얻은 Cookie 배열로부터 매핑을 생성
initParam	초기화 파라미터의 <이름, 값> 매핑을 저장한 Map 객체 application.getInitParameter(이름);의 결과와 동일

EL의 객체 접근과 탐색

- [list1100-1.jsp] 파일 참고

EL의 연산

- 수치 연산
  - +, -, *, /, % 또는 mod, -(단항)
  - 연산하는 숫자 데이터가 null이면 0으로 처리
  - [list1100-2.jsp] 참고

- 비교 연산
  - ==, eq
  - !=, ne
  - <, lt
  - >, gt
  - <=, le
  - >=, ge

EL의 연산

- 논리 연산
  - &&, and
  - ||, or
  - !, not

- empty 연산
  - 형식: empty ${값}
  - 주어진 값이 null, “”, 0, 빈 Map, 빈 Collection이면 true,
  - 아니면 false

- 비교 선택 연산자
  - 형식: ${식 ? 값1 : 값2}
  - 식의 결과가 true이면 값1, 아니면 값2를 출력

EL의 연산

- 문자열 연결
  - ${“문자” + “문자” + “문자”}
  - ${“문자” + 변수 }
  - Tomcat7에서는 문자열 연결 불가능(EL 3.0부터 가능)
- 세미콜론 연산자
  - ${ 1 + 1; 10 + 10}
  - 세미콜론을 사용하여 식을 나열하면 마지막 식의 결과만 출력
- 할당 연산자
  - <% request.setAttribute(“var”, 10); %>
  - <c:set var=“var” value=“${10} />
  - ${var = 10} <%-- 변수에 값을 할당하고 값을 출력한다. --%>
  - ${var = 10} ${var} <%-- 출력 결과: 10 10 --%>

EL에서 연산자 우선순위

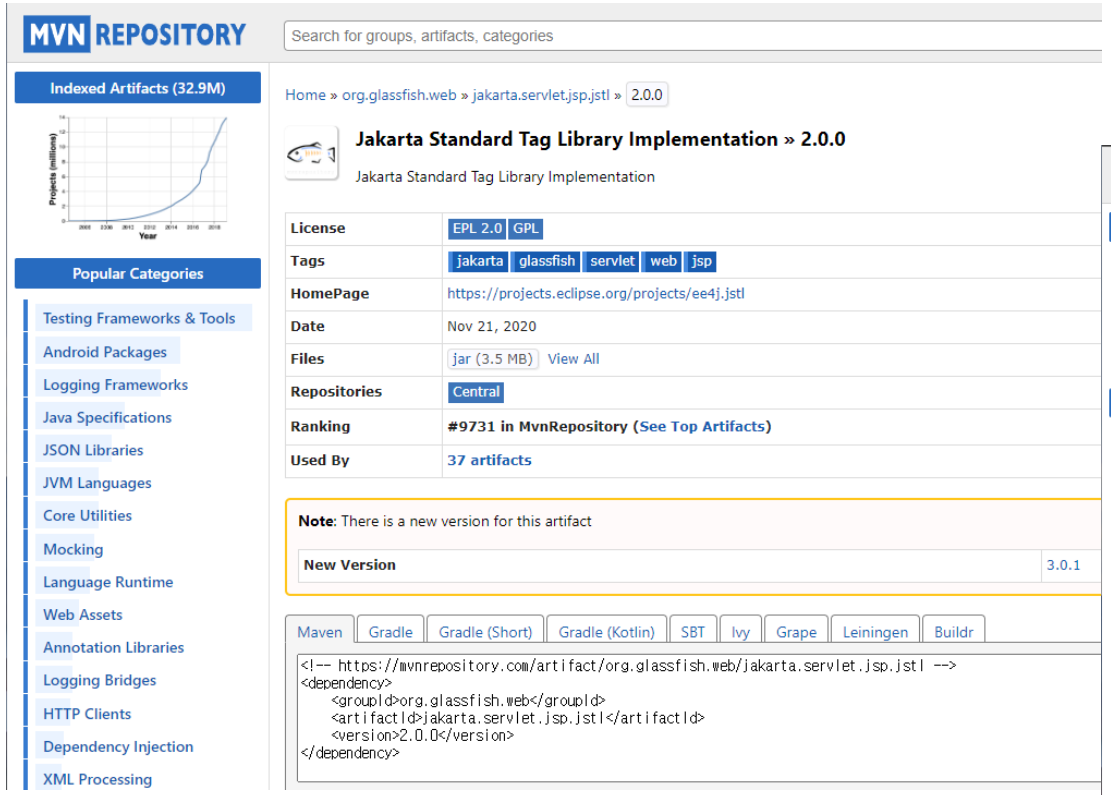
- [] .
- ()
- - (단항) not ! empty
- * / div % mod
- + -
- +=
- < > <= >= lt gt le ge
- == != eq ne
- && and
- || or
- ?:
- ->
- =
- ;

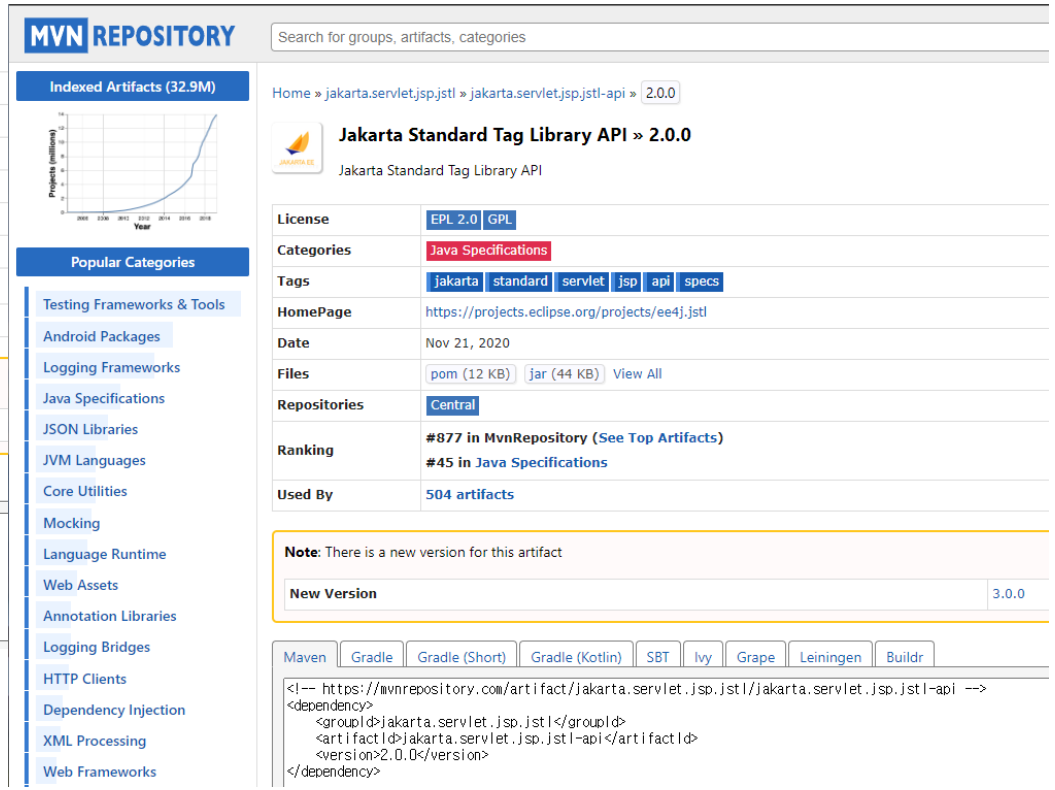
EL에서 특수 문자 처리

- 예
  - 표현식 기본문법: \${ 10 + 10} 또는 \#{20 + 20}

EL에서 특수 문자 처리

<%-- JSP 3.1에서는 jstl-1.2 사용 불가능 --%>
<%-- JSTL-2.0.0은 JSP 3.0이상 필요 --%>

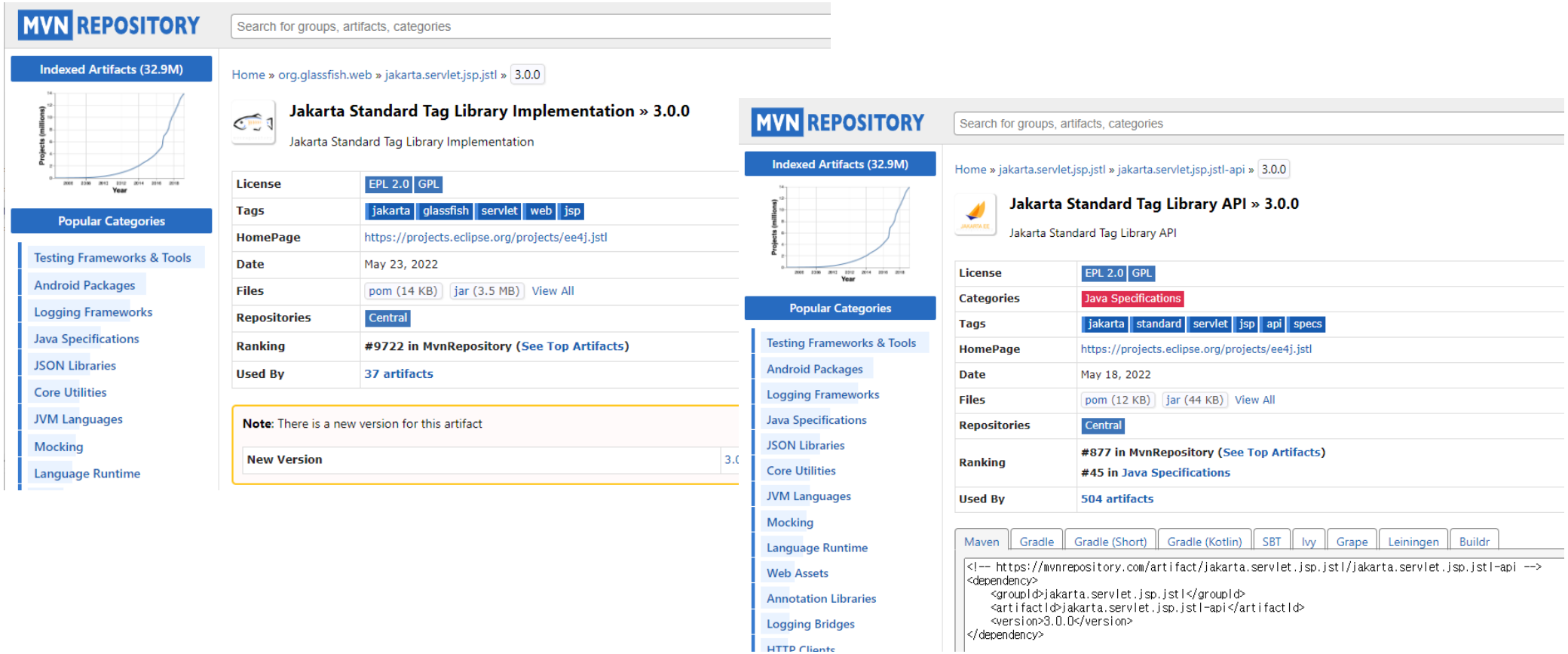




<%-- jar 파일 다운로드 위치 --%>
<%--
https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api/2.0.0
https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl/2.0.0
--%>

EL에서 특수 문자 처리

<%-- JSTL-3.0.0은 아직 지원하지 않는 개발 도구가 있음 --%>



<%-- jar 파일 다운로드 위치 --%>
<%--
https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl/3.0.0
https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api/3.0.0
--%>

EL에서 특수 문자 처리

<%-- JSTL-3.0.0은 아직 지원하지 않는 개발 도구가 있음 --%>
<%-- JSTL-3.0.0에서는 URI 변경됨 --%>

Jakarta Standard Tag Library Tag Libraries

Functional Area	URI	Prefix
core	*jakarta.tags.core*	*c*
XML processing	*jakarta.tags.xml*	*x*
I18N capable formatting	*jakarta.tags.fmt*	*fmt*
relational db access (SQL)	*jakarta.tags.sql*	*sql*
Functions	*jakarta.tags.functions*	fn

EL에서 객체 메서드 호출

- JSP 2.1에서는 객체의 메서드 호출 불가
- JSP 2.2부터는 객체의 메서드 호출 가능
- EL에서 정적 메서드 호출
  - 방법 1:
    - 정적 메서드를 포함하는 클래스 작성
    - TLD 파일을 사용하여 클래스 등록
    - TLD 파일을 web.xml에 등록
    - JSP에서 <@% taglib prefix="이름" uri="클래스 경로" /> 디렉티브 사용
      - ${prefix이름:메서드이름(인수) } 형식으로 호출
  - 방법 2:
    - ${클래스이름.메서드이름(인수) } 형식으로 JSP에서 직접 사용
    - EL 3.0부터 가능

EL에서 람다식 사용

- EL에서 람다식 사용 예
  - ${ greaterThan = (a, b) -> b ? true : false }
  - ${ greaterThan = (a, b) -> b ? true : false; } ‘’ <%-- 단일 따옴표 두 개 --%>

스트림 API 사용

- EL 3.0부터 스트림 API 제공
- EL 3.0 이전에는

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

- EL 3.0부터는

```
<c:set var="lst" value="<% java.util.Array.asList( 1, 2, 3, 4, 5) %>" />
<c:set var="sum" value="${lst.stream().sum() } />
${sum }
```

스트림 API 기본

- 스트림 API 기본 형식

```
Collection.stream()              // collection에서 스트림 생성
          .map(x -> x * x)       // 중간 연산으로 스트림 변환
          .toList()              // 최종 연산으로 결과 생성
```

- 스트림 API 사용 예 1

```
스트림을 사용한 연산 예 <br />
<c:set var="lst"
 value="<%= java.util.Arrays.asList(1, 2, 3, 4, 5) %>" />
  ${lst.stream()
        .map(x -> x * x)
        .toList()
}
<br />
}
```

```
${lst.stream()
      .filter(x -> x % 2 == 0)
      .map(x -> x * x)
      .toList()
}
<br />
${lst.stream()
      .filter(x -> x % 2 == 0)
      .sum()
}
```

스트림 API 기본

- 스트림 API 사용 예를 위한 City 클래스

```
public class City implements Comparable<Object> {
    private String name;
    private double area;
    private int population;
    private String areaCode;
    public City() { } // 인수가 없는 생성자
    public City(String name, double area, int     population, String areaCode) {
        this.name = name;
        this.area = area;
        this.population = population;
        this.areaCode = areaCode;
}
// 이하 Getter와 Setter를 생성한다. [Source]-[Generator Getters/Setters]
```

스트림 API 기본

- 스트림 API 사용 예 2(City.class를 생성한 뒤에)

```
<%
List<City> cities = new ArrayList<>(Arrays.asList(
                        new City("Seoul", 605.2, 9720846, "02"),
                        new City("Ulsan", 1062, 1142190, "052"),
                        new City("Incheon", 1063.3, 2947217, "032"),
                        new City("Daegu", 883.5, 2427954, "053"),
                        new City("Gwangju", 501.1, 1455048, "062"),
                        new City("Busan", 770.1, 3404423, "051")
));
request.setAttribute("cities", cities);
%>
```

```
데이터 선별<br />
${res = cities.stream()
        .filter(city -> city.getArea() > 800)
        .distinct()
        .toList(); ''
}
${res }<br />
<c:forEach var="item" items="${res}">
    ${item.name } ${item.area }
    ${item.population }
    ${item.areaCode }<br />
</c:forEach>}
```

스트림 API 기본

- 스트림 API 사용 예 2(City.class를 생성한 뒤에)

```
데이터 선별<br />
${res = cities.stream()
                .filter(city -> city.getArea() > 800)
                .distinct()
                .toList(); ''
}
${res }<br />
<c:forEach var="item" items="${res}">
    ${item.name } ${item.area } ${item.population } ${item.areaCode }<br />
</c:forEach>}
```

스트림 API 기본

- 스트림 API 사용 예 3

```
sort 테스트<br />
<c:set var="dt" value="<%= java.util.Arrays.asList(1.2, 3.4, 2.7, 4.6) %>"
/>
역순 출력:
${ dt.stream()
        .sorted((i, j) -> j - i)
        .toList()}
<br />
순차 출력:
${dt.stream()
       .sorted((i, j) -> i - j)
       .toList() }
<hr />
```

스트림 API 기본

- 스트림 API 사용 예 4

```
데이터 개수 제한<br />
${res = cities.stream()
.filter(city -> city.getArea() > 800)
.sorted((m1, m2) -> m1.area.compareTo(m2.area))
.limit(2)
.toList(); ''
}
===${res }<br />
<c:forEach var="item" items="${res}">
${item.name } ${item.area } ${item.population } ${item.areaCode }<br />
</c:forEach>}
<hr />
```

스트림 API 기본

- 스트림 API 사용 예 5

```
데이터 개수 확인<br />
${res = cities.stream()
.filter(city -> city.getArea() > 800)
.distinct()
.count(); ''
}
${res }<br />
```

스트림 API 기본

- 스트림 API 사용 예 6

```
데이터 Array로 출력<br />
${res = cities.stream()
                .filter(city -> city.getArea() > 800)
                .distinct()
                .toArray() ; ''
}
<c:forEach var="item" items="${res }" >
   ${item.name } ${item.area } ${item.population } ${item.areaCode }<br />
</c:forEach>
```

스트림 API 기본

- 스트림 API와 Optional 사용 예 1

```
Optional 예 확인: anyMatch<br />
${ res = cities.stream()
.anyMatch( c -> c.area > 800); ''
}
${res.get()} <%-- Optional에서 조건을 만족하는 데이터가 있는지만 확인 --%>
<hr />
Optional 예 확인(값이 존재하지 않을 때: anyMatch<br />
${ res = [].stream()
.anyMatch( v -> v > 4); ''
}
<%--${res.get() } --%>
${ res.orElse("값이 없다.") }
```

스트림 API 기본

- 스트림 API와 Optional 사용 예 2

```
Optional 예 : max<br />
${ res = cities.stream()
.max((c1, c2) -> c1.area.compareTo(c2.area)); ''
}
${res.get()}
${res.get().name } : ${res.get().area } : ${res.get().population } :
${res.get().areaCode }
<hr />
Optional 예 : min<br />
${ res = cities.stream()
.min((c1, c2) -> c1.population.compareTo(c2.population)); ''
}
${res.get()}
${res.get().name } : ${res.get().area } : ${res.get().population } :
${res.get().areaCode }
```

${변수} 와 #{변수} 차이점

```
\${변수}와 \#{변수 } 차이 구분<br />
<%
    City ct = new City();
    ct.setName("홍길동");
%>
<c:set var="city" value="<%= ct %>" />
<c:set var="name1" value="${city.name }" />
name1의 값은 ${name1 } <br />
<c:set var="name2" value="#{city.name}" />
name2의 값은 ${name2 } <br />
<% ct.setName("장길산"); %>
name1의 값은 ${name1 } <br />
name2의 값은 ${name2 } <br />
<% ct.setName("임꺽정"); %>
name1의 값은 ${name1 } <br />
name2의 값은 ${name2 } <br />
```

표준 태그 라이브러리(JSTL)

- JSTL Java Standard Tag Library
  - 스크립틀릿으로 사용하는 자바 코드를 HTML 태그 형식으로 사용할 수 있도록 한다.
  - JSP에서 제공
  - 개발자가 만들어 사용할 수 있다. - Custom Tag
    - Class 작성 - TLD 등록 - web.xml에 TLD 등록 - 사용
- JSTL 사용을 위해서 라이브러리(.jar) 파일 필요
  - JSTL을 위한 .jar 파일을 WEB-INF/lib 폴더에 붙여넣기
  - maven에서는 pom.xml 파일에 정보 등록
  - gradle에서는 bundle.gradle 파일에 정보 등록

JSTL 태그 종류

라이브러리	주요 기능	접두어	URI
코어	변수지원 흐름제어 URL 처리	c	uri="*http://java.sun.com/jsp/jstl/core*"
국제화/형식	지역 메시지 형식 숫자 및 날짜 형식	fmt	uri="http://java.sun.com/jsp/jst/fmt"
데이터베이스	SQL	sql	uri="http://java.sun.com/jsp/jstl/sql"
함수	컬렉션 처리 String 처리	fn	uri="http://java.sun.com/jsp/jstl/functions"
XML	XML 코어 흐름 제어 XML 변환	x	uri="http://java.sun.com/jsp/jstl/xml"

JSTL – Core 태그

기능 분류	태그	설명
변수 지원	set	JSP에서 사용할 변수를 설정
	remove	설정한 변수를 제거
흐름 제어	if	조건에 따라 내부 코드를 실행
	choose	다중 조건을 처리할 때 사용
	forEach	Collection 또는 Map을 처리할 때 사용(반복 기능)
	forTokens	구분자로 구분된 각각의 토큰을 처리할 때 사용
데이터베이스	import	URL을 사용하여 다른 자원을 삽입
	redirect	지정한 경로로 리다이렉트
	url	URL을 재작성
기타 태그	catch	Exception 처리
	out	JspWriter에 내용 출력

JSTL - <c:set ... >

- 변수 설정 기본형식

```
<c:set  var="변수명" value="값"  scope="사용영역" />
<c:set  var="변수명" scope="사용영역" >값</c:set>
```

- 객체의 프로퍼티 설정 기본형식

```
<c:set  target="대상" property="프로퍼티이름"  value="값" />
<c:set  target="대상" property="프로퍼티이름" >값</c:set>
```

속성	표현식(EL)	타입	설명
var	사용 불가	String	EL 변수 이름
value	사용 가능	Object	변수에 할당할 값
scope	사용 불가	String	변수를 생성할 영역으로 기본값은 page
target	사용 가능	Object	프로퍼티 값을 설정할 객체 지정
property	사용 가능	String	프로퍼티 이름

JSTL - <c:remove ... >

- 변수 삭제 기본형식

```
<c:remove  var="변수명" scope="사용영역" />
```

- <c:remove ...> 사용할 때 주의할 점

```
<c:set  var="name" value="홍길동" scope="request" />
<c:set  var="name" value="임꺽정" scope="session" />
<c:remove  var="name" /> <%-- 두 개의 영역에서 모두 삭제, 반드시 scope 지정 --%>
```

속성	표현식(EL)	타입	설명
var	사용 불가	String	삭제할 EL 변수 이름
scope	사용 불가	String	삭제할 변수가 포함된 영역

JSTL - <c:if ... >

- <c:if ...>의 기본형식

```
<c:if test="조건" var="변수이름"/>  <%-- 조건식의 결과를 변수에 저장 --%>
```

- <c:if ...> 사용 예

```
<c:if test="true"> … 항상 실행 … </c:if>
<c:if test="false"> … 항상 실행 않음 … </c:if>
<c:if test="${expr}" … ${expr}의 결과가 참이면 실행 … </c:if>
<c:if test="<%= expr %>" … <%= expr %>의 결과가 참이면 실행 … </c:if>
```

속성	표현식(EL)	타입	설명
test	사용 가능	boolean	검사 조건
var	사용 불가	String	검사 조건의 결과값을 저장할 EL 변수

JSTL - <c:if ... >

- <c:if ...> 사용 예: [리스트 12.1] 참고

```
<c:if test="${param.name = '홍길동'}>
    name 파라미터의 값은 ${param.name} 입니다. </br>
</c:if>
```

JSTL - <c:choose ... ><c:when ...><c:otherwise ...>

- <c:choose ...>의 기본형식

```
<c:choose
   <c:when test="조건1">
      조건1이 참일 때 실행할 내용 …
   </c:when>
   <c:when test= "조건2">
      조건2가 참일 때 실행할 내용 …
   </c:when>
   <c :otherwise>
      조건1과 2가 모두 참이 아닐 때 실행할 내용
   </c/c:otherwise>
</c:choose>
```

JSTL - <c:choose ...>

- <c:choose ...> 사용 예: [리스트 12.2] 참고

```
<c:choose>
   <c:when test= "${param.name == '홍길동' and param.age ge 18 } ">
      전달된 데이터는 '${param.name}'으로 ${age} 이상으로 성인입니다. <br />
   </c:when>
   <c:otherwise>
      전달된 데이터는 '${param.name}'도 아니고, ${param.age} 이상이 아닌
      미성년입니다. <br />
   </c:otherwise>
</c:choose>
```

JSTL - <c:forEacch ... >

- <c:forEach ...>의 기본형식

```
<c:forEach var="변수" items="아이템">
   … 반복할 내용 …
</c:forEach>
```

```
<c:forEach var="변수" begin="값" end="값" step="값" >
   <%-- begin부터 end까지 step 만큼 증감하면서 반복 --%>
   … 반복할 내용 …
</c:forEach>
```

- <c:forEach ...> 사용 예 1

```
<c:forEach var="i" end="10" step="2" > <%-- step을 생략할 때와 비교 필요 --%>
   ${i} 사용<br />
</c:forEach>
```

JSTL - <c:forEach ... >

- <c:forEach ...> 사용 예 2: 배열 형식을 사용할 때 주의

```
<% Array<int> arr = [1, 2 , 3, 4, 5, 6] %>
<%-- 배열 사용할 때, forEach에서 begin은 시작 첨자(순번)가 된다.  --%>
<c:forEach var="ix" items="${arr}" begin="1" end="${length }" step="1"
                varStatus="status">
첨자는 ${status.index } : 값은 ${arr[status.index] } :
current ${status.current } : index ${status.index } : count ${status.count } :
last ${status.last} : begin ${status.begin} : end ${status.end} : step ${status.step }<br />
</c:forEach>
```

JSTL - <c:forEach ... >

- <c:forEach ...> 사용 예 3: 배열 형식을 사용할 때 주의

```
<%-- name과 age, 두 개의 속성을 갖는 Member Class 생성한 뒤에 --%>
<%
    ArrayList<Member> memberList = new ArrayList<Member>();
    memberList.add( new Member("홍길동", 20));
    memberList.add( new Member("장길산", 30));
    memberList.add( new Member("임꺽정", 40));
    session.setAttribute("memberList", memberList);
%>

memberList 내용 확인<br />
<c:forEach var="memList" items="${memberList }" varStatus="status">
<%-- forEach에서 begin 사용할 때와 비교 필요 --%>
이름 ${memList .name} / 나이 ${memList.age} :: index ${status.index } <br />
</c:forEach>
```

JSTL - <c:forEach ... >

속성	표현식(EL)	타입	설명
var	사용 불가	String	forEach 몸체에서 사용할 EL 변수 이름
items	사용 가능	Collection Iterator Enumeration Map 배열	반복 처리할 데이터
varStatus	사용 불가	String	반복 상태를 저장할 EL 변수 이름
begin	사용 가능	int	시작 인덱스 값
end	사용 가능	int	끝 인덱스 값
step	사용 가능	int	인덱스 증분 값(음수 사용 불가능)

JSTL - <c:forTokens ... >

- <c:forTokens ...>의 기본형식

```
<c:forTokens var="변수" items="아이템" delims="구분문자">
   … 반복할 내용 …
</c:forTokens>
```

- <c:forEach ...> 사용 예

```
<c:forTokens var="color" items="red, green. blue" delims=",." >
   <p style="${color};"> 색상 처리</p>
</c:forTokens>
```

JSTL - <c:url ... >

- <c:url ...>의 기본형식

```
<c:url var="URL" var="변수이름" scope="영역">
   <c:param name= " 이름" value="값" />
</c:url>
```

- URL
  - 절대 경로: http://www.dongyang.ac.kr/test.sjp 형식의 완전한 URL
  - 상대 경로:
    - 웹 어플리케이션 내의 절대 경로: "/view/list.jsp"로 사용하면 앞에 contextPath가 붙는다.
    - 현재 JSP 파일에 대한 상대 경로: "../view/list.jsp"로 사용하면 현재 사용하는 JSP 파일의 위치에서 상대 경로 추적한다.

```
<c:url value="/view/list.jsp" /> <%-- ContextPath/view/list.jsp --%>
```

JSTL - <c:url ... >

- <c:url ...>의 사용 예 1

```
<c:url value="/view/list.jsp" /> <%-- 결과는  ContextPath/view/list.jsp --%>
```

- <c:url ...>의 사용 예 2: [리스트 12.5] 참고

```
<c:url value=http://www.dongyang.ac.kr/search var="searchURL" >
    <c:param name="w" value="blog" />
    <c:param name="q" value="공원" />
    <%-- c:param은 URL에 쿼리스트링(GET 방식의 데이터 전달)로 추가 됨 --%>
</c:url>
…
<ul>
    <li>${searchURL}</li>
    <li><c:url value="/use_absoluthPath.jsp" /></li> <%-- 문자열로 표현 --%>
    <li><c:url value="./use_relativePath.jsp" /></li>
</ul>
```

JSTL - <c:redirect ... >

- <c:redirect ...>의 기본형식

```
<c:redirect url="URL" contexte="컨텍스트 경로">
   <c:param name= " 이름" value="값" />
</c:redirect>
```

- <c:redirect ... > 사용 예

```
<c:redirect url="redirectedPage.jsp" /> <%-- url의 페이지로 페이지 이동 --%>
<c:redirect url="redirectedPage.jsp" context="/ch13" />
<c:redirect url="redirectedPage.jsp">
   <c:param name="w" value="blog" />
   <c:param name="q" value="공원" />
</c:redirect>
```

JSTL - <c:redirect ... >

- <c:out ...>의 기본형식

```
<c:out  value="값" escapeXml="true|false" default="기본값">
<c:out  value="값" escapeXml="true|false"> "기본값"</c:out>
```

- value 속성의 값을 출력한다.
  - value 에 Reader 객체가 지정되면 해당 Reader 객체에서 값을 읽어 출력
- default는 value 속성의 값이 없을 때 출력되는 기본값이다.
- escapeXml은 출력 내용에 포함된 기호를 변환하여 출력한다. 기본은 "true"
  - < : <, > : >, & : &, ' : ', " : "
  - 웹 브라우저에서 자동 처리되어 기호 그대로 표시된다.

JSTL - <c:redirect ... >

- <c:out ... > 사용 예: [리스트 12.6] 참고

```
<%-- file 준비 --%>
<%
   FileReader reader = null;
   try {
      String path = request.getParameter("path");
      reader = new FileReader(getServletContext().gtRealPath(path)); // encoding 불가능
%>
<pre>
파일은 <%= path %><br />
<c:out value="<%= reader %>" escapeXml="true" />
</pre>
<%
   } catch(IOException ex) {
%>
      파일 <%= path %> 읽기에서 에러 <%= ex.getMessage() %>
<%
   } finally {
      if(reader != null) {
         try { reader.close();
      } catch { IOException ex) {}
   }
%>
```

** 참고: 파일을 읽을 때 encoding **

- FileReader 객체를 사용할 때 주의 : encoding 불가능
- 앞의 예에서 reader 객체를 생성하는 방법을 변경

```
reader = new FileReader(getServletContext().gtRealPath(path));

-->

reader = new InputStreamReader(
                    new FileInputStream(getServletContext().getRealPath(path)), “utf8”);
```

JSTL - <c:catch ... >

- <c:catch ...>는 발생한 Exception을 EL 변수에 저장
- <c:catch ...>의 기본 형식

```
〈c:catch  var=“exName”〉
    … 익셉션이 발생할 수 있는 코드
〈/c:catch〉
…
${exName} 사용
```

JSTL - <c:catch ... >

- <c:catch ...> 사용 예

```
<c:catch  var="ex">
name 파라미터 값은 <%= request.getParameter("name") %><br
/>
<%
   if(request.getParameter("name").equals("test")) {
%>
      ${param.name}은 ${param.value("name") 입니다.
<%
   }
%>
</c:catch>
<c:if test="${ex != null}">
   처리 중 Exception이 발생했습니다. <br />
</c:if>
```

JSTL - <fmt: ...>

- <fmt: ...> JSTL의 기능

기능분류	태그타입	설명
로케일 지정	setLocale	Locale을 지정한다.
	requestEncoding	요청 파라미터의 문자 Encoding을 지정한다.
메시지 처리	bundle	사용할 번들을 지정
	message	지역에 알맞은 메시지 출력
	setBundle	리소스 번들을 읽어와 특정 변수에 저장
숫자 및 날짜 표현 형식	formatNumber	숫자 표현 형식 설정
	formatDate	날짜 표현 형식 설정
	parseDate	문자열로 표현된 날짜를 Date 객체로 변환
	parseNumber	문자열로 표현된 숫자 데이터를 숫자로 변환
	setTimeZone	timeZone 설정
	timeZone	timeZone 확인

JSTL – 리소스 번들 작성

- 리소스 번들 파일
  - 외부 라이브러리 jar 파일을 두는 위치에 넣어야 한다.
  - 리소스 번들 파일 이름
    - message.properties
    - Message_ko.properties
  - [리스트 12.8] [리스트 12.9] 참고

JSTL - <fmt: ...>

- 리소스 사용 관련 <fmt: setLocale...> / <fmt: bundle ... prefix ... > / <fmt: message ...> / <fmt: param ...> 사용 예 1

```
<fmt:setLocale value="en" scope="session" />
<%-- setLocal 설정이 없으면, 브라우저 기본 Locale 사용 --%>
<fmt:bundle basename="resource.message"  prefix="prefix">
   <fmt:message key="prefixTITLE" var="title" />
...
<html>
   <head>
      <title>${title}</title>
   </head>
   <body>
      <fmt:message key="GREETING" />
      <fmt:message key="BODY" >
         <fmt:param value="${param.id}" />
      </fmt:message>
   </body>
</html>
</fmt:bundle>
```

JSTL - <fmt: ...>

- 리소스 사용 관련 <fmt:setBundle ...> 사용 예 2

```
<fmt:setLocale value="en" scope="session" /> <%-- en -> ko --%>
<%-- setLocal 설정이 없으면, 브라우저 기본 Locale 사용 --%>
<fmt:setBundle var="bundle"  basename="resource.message" >
<fmt:message bundle="${bundle}" key="TITLE" var="title" />
...
<html>
   <head>
      <title>${title}</title>
   </head>
   <body>
      <fmt:message bundle="${bundle} key="GREETING" />
      <fmt:message bundle="${bundle} key="BODY" >
         <fmt:param value="${param.id}" />
      </fmt:message>
   </body>
</html>
```

JSTL - <fmt: ...>

- 숫자 및 날짜 표현 형식 <fmt:formatNumber ...>

속성	표현식/EL	타입	설명
value	사용 가능	String 또는 Number	양식에 맞추어 출력할 숫자를 지정
type	사용 가능	String	어떤 양식으로 출력할지를 지정 number이면 숫자형식(기본값), percent이면 %형식 currency이면 통화형식으로 출력
pattern	사용 가능	String	숫자가 출력되는 양식을 설정 java.text.DemimalFormat에 정의된 양식 사용
currencyCode	사용 가능	String	ISO 4217에 정의된 통화 코드 설정 예) 한국은 'KRW', type 속성이 currency일 때만 유효
currencySymbol	사용 가능	String	통화를 표현할 때 사용할 기호 설정 type 속성이 currency일 때만 유효
groupingUsed	사용 가능	boolean	콤마 문자와 같이 단위를 구분할 때 사용되는 기호 사용 여부를 설정 (기본값은 true) true일 때의 예는 12,000/false이면 12000
var	사용 불가	String	형식이 설정된 결과를 저장할 변수 이름 저장된 결과는 String이 된다.
scope	사용 불가	String	변수를 저장할 영역 설정

JSTL - <fmt: ...>

- 숫자 및 날짜 표현 형식 **<fmt:formatNumber ...>** 사용 예

```
<c:set var="price" value="12000" />
<fmt:formatNumber  value="${price }" type="number" var="numberType" />
<br />
통화 <fmt:formatNumber value="${price}" type="currency" currencySymbol="원" />
<br />
백분율 <fmt:formatNumber value="${price}" type="percent" groupingUsed="false" /><br />
백분율 <fmt:formatNumber value="${price}" type="percent" groupingUsed="true" /><br />
숫자 ${numberType }<br />
패턴 <fmt:formatNumber value="${price }" pattern="00000000.00" /><br />
패턴 <fmt:formatNumber value="${price }" pattern="#######0.00" />
```

JSTL - <fmt: ...>

- 숫자 및 날짜 표현 형식 <fmt:parseNumber ...>

속성	표현식/EL	타입	설명
value	사용 가능	String	파싱할 문자열 지정
type	사용 가능	String	value 속성의 문자열에 대한 타입을 지정 Number(기본값), percent, currency 사용 가능
pattern	사용 가능	String	직접 파싱할 양식을 지정
parseLocale	사용 가능	String 또는 java.util.Locale	파싱할 때 사용할 locale 지정
integerOnly	사용 가능	String	정수 부분만 파싱할 지를 설정 기본값은 false
var	사용 불가	String	파싱한 결과를 저장할 변수
scope	사용 불가	String	변수를 저장할 영역 설정

JSTL - <fmt: ...>

- 숫자 및 날짜 표현 형식 <fmt:formatDate ...>

속성	표현식/EL	타입	설명
value	사용 가능	java.util.Date	포맷팅할 날짜/시간 데이터를 지정
type	사용 가능	String	포맷팅할 대상으로 날짜, 시간, 날짜와 시간 지정 time, date(기본값), both
dateStyle	사용 가능	String	날짜에 대해 포맷팅할 스타일 지정 default(기본값), short, medium, long, full 중 선택
timeStyle	사용 가능	String	시간에 대해 포맷팅할 스타일 지정 default(기본값), short, medium, long, full 중 선택
pattern	사용 가능	String	직접 표현할 양식을 지정 java.text.DateFormat에 있는 양식을 사용
timeZone	사용 가능	String 또는 java.util.TimeZone	시간대를 변경할 때 사용 <fmt:setTImeZone>에서 생성한 TImeZone 사용
var	사용 불가	String	형식이 설정된 결과를 저장할 변수 이름 저장된 결과는 String이 된다.
scope	사용 불가	String	변수를 저장할 영역 설정

JSTL - <fmt: ...>

- 숫자 및 날짜 표현 형식 <fmt:formatDate ...> 사용 예

```
<fmt:setLocale value="en"/>
<fmt:setTimeZone value="GMT" />
full date <fmt:formatDate value="${now}" type="date" dateStyle="full" timeZone="GMT" /> <br />
medium date <fmt:formatDate value="${now}" type="date" dateStyle="long" /> <br />
long date <fmt:formatDate value="${now}" type="date" dateStyle="medium" /> <br />
short date <fmt:formatDate value="${now}" type="date" dateStyle="short" /> <br />
full time <fmt:formatDate value="${now}" type="time" timeStyle="full" /> <br />
long time <fmt:formatDate value="${now}" type="time" timeStyle="long" /> <br />
medium time <fmt:formatDate value="${now}" type="time" timeStyle="medium" /> <br />
short time <fmt:formatDate value="${now}" type="time" timeStyle="short" /> <br />
type='both' <fmt:formatDate value="${now}" type="both" dateStyle="full" timeStyle="full" /> <br />
```

End of Document

Thank you