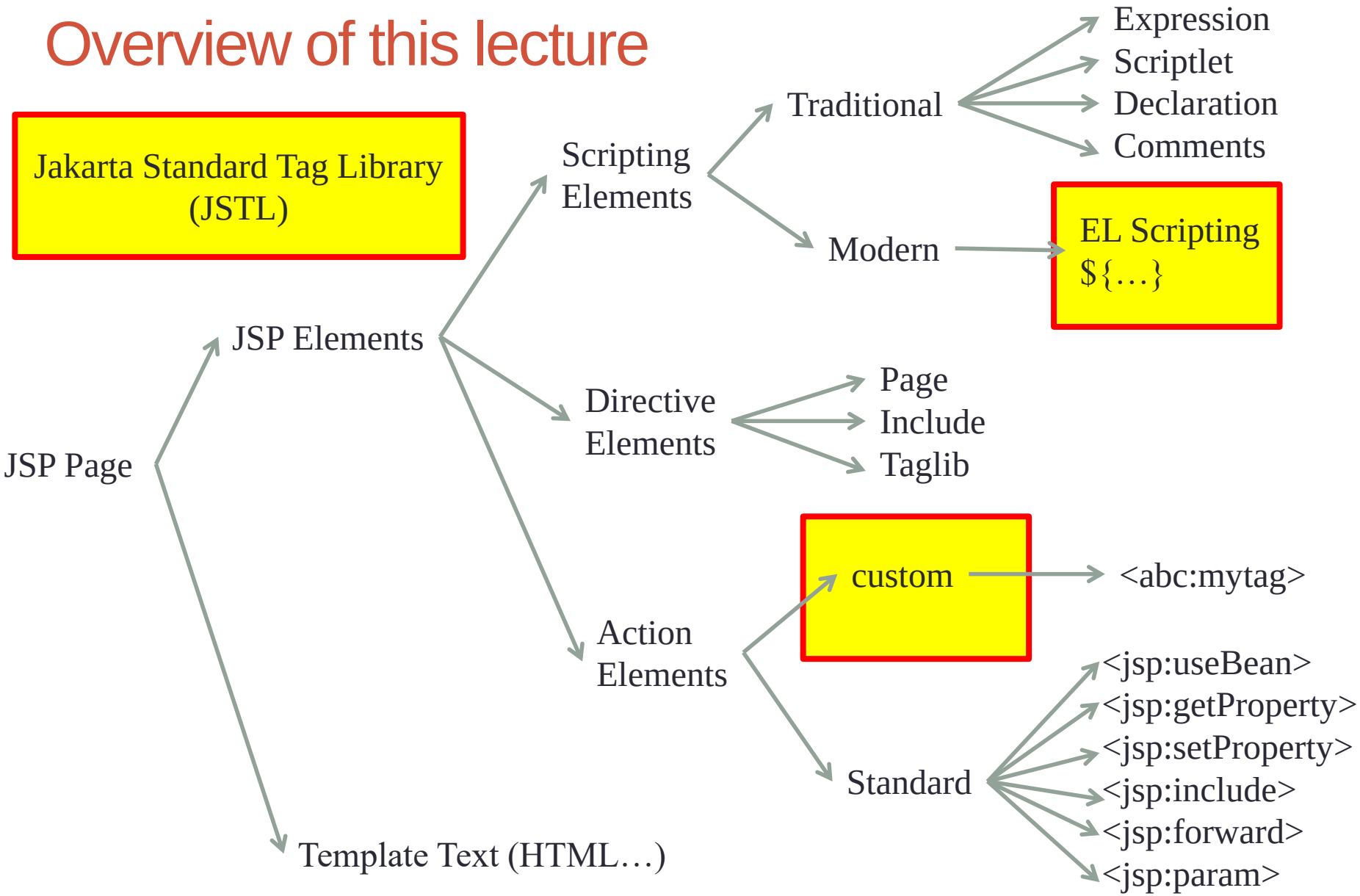
COMP S380F Lecture 5: EL, JSTL, Custom tag

Dr. Keith Lee
School of Science and Technology
Hong Kong Metropolitan University

Overview of this lecture



Overview of this lecture

- EL (Expression Language)
  - EL expression
  - EL implicit objects
  - Accessing scoped variables
  - dot operator & bracket [ ] operator
- JSTL (Jakarta Standard Tag Library)
  - **Core**: <c:forEach>, <c:if>, <c:choose>, <c:when>, <c:otherwise>, <c:set>, <c:url>, <c:param>
  - **Fn** (*but **Fmt**, **SQL**, **XML** are not covered*)
- JSP custom tags

Motivating example for Expression Language (EL)

Problem of using <jsp:useBean>, <jsp:get/setProperty> in JSP

- XML syntax is verbose and clumsy (e.g., easy to miss a closing tag)
- Accessing sub-properties of a JavaBean is not supported

```
protected void doGet(...) {
   Dog dog = new Dog();
   dog.setName("Lucky");
   Person p = new Person();
   p.setName("Keith");
   p.setDog(dog);
   request.setAttribute("person", p);
   RequestDispatcher view =
       request.getRequestDispatcher("result.jsp");
   view.forward(request, response);
}
```

Person

- name: String
- dog: Dog

Dog

- name: String

result.jsp

```
<jsp:useBean id="person" class="package.Person" scope="request" />
My dog is: <%= person.getDog().getName() %>
```

Expression Language (EL)

- JSP 2.3 lets you simplify the way of accessing Java code by using **Jakarta Unified Expression Language (EL)**.
- EL can replace scripting elements, useBean & get/setProperty actions.
  - Towards scriptless JSP
- Syntax: **${ expression }**
- E.g., the previous code in result.jsp:

```
<jsp:useBean id="person" class="package.Person" scope="request" />
My dog is: <%= person.getDog().getName() %>
```

```
My dog is: ${person.dog.name}
```

EL Expression

- EL syntax: **${ expression }**
- The expression should always evaluate to some value
  e.g., ${105.509}
- If the expression starts with a named variable, this variable must be either an attribute or an EL implicit object (which is different to JSP implicit object).

```
${firstThing.secondThing}
```

EL implicit object

```
pageScope
requestScope
sessionScope
applicationScope

param
paramValues
```

```
header
headerValues

cookie

initParam
```

Attribute

```
attribute objects
in page scope
in request scope
in session scope
in application scope
```

EL Properties

- Can be used anywhere ***except*** in JSP directives, and JSP declarations, scriptlets and expressions (as they expect Java code)
- It should **always evaluate to some value**.
- EL implicit objects (besides the JSP implicit objects)
- Easy access to
  - Stored attributes

    e.g., `session.setAttribute("subscription","Monthly")`

    Your subscription choice is: ${subscription}
  - Bean properties

    e.g., ${person.name}, ${person.dog.name}
  - Collections: accessing array or List or Map

    e.g., ${**orderItems[0]**}

EL Properties (cont')

- Automatic type conversion
  - ${105} → int
  - ${-132147483648} → long
  - ${13922337203685477580 7} → BigInteger
  - ${105.509} → float
  - ${340000000000000000000000000000001.0} → double
  - ${1.7976931348623157 0e+309} → BigDecimal
  - String to number
- Null handling (empty value instead of an error message)
- Strings (quoted in double or single quotes) are concatenate with **+=**:
  - ${'The user will see ' += expr1 += "text and will " += expr2 += '.'}

Accessing Scoped Variables

- Scoped Variables (Attributes): name-value pair

Scope	Location of storage
page	PageContext object (which is not shared)
request	HttpServletRequest object
session	HttpSession object
application	ServletContext object

- In each of the location, you could use setAttribute() / getAttribute() to store / retrieve the name-value attribute pairs.

```
request.setAttribute("customerLocation","Hong Kong"); // a String
request.setAttribute("ShoppingCart", cart); // an object
```

- EL gives you an easy way to access these scoped variables:

```
<p>You are shopping from : ${customerLocation}</p>
<p>Total price is: ${ShoppingCart.total} </p>
```

Accessing Scoped Variables (cont')

- We do not need to specify the scope of the variable in the expression.
- The attribute name in the expression ${attribute_name} is searched in all four locations, in the search order:

  1. PageContext
  2. HttpServletRequest
  3. HttpSession
  4. ServletContext

from specific to general

- Choose unique names for the attribute if you don't want the container to be confused, or return an unexpected answer.
- In case, you need to access an attribute of a particular scope. You can use the EL implicit object.
  E.g., ${**requestScope**.attribute_name}, ${**sessionScope**.attribute_name}

dot operator

If the expression has a variable followed by a dot (.):

```
${person.name}
```

left-hand variable

- java.util.Map (something with a key), or
- A bean (something with properties)

right-hand variable

- a Map key, or
- a bean property

Must be a Java name, e.g.,

- ✓ Cannot start with a number
- ✓ Cannot be a Java keyword
- ✓ No spaces

dot operator (cont')

Lecture05 web app: branch lecture05 of https://github.com/cskeith/380_2024.git

Consider the following request attribute "dinnerMap":

Servlet

```
Map<String, String> dinnerMap = new ConcurrentHashMap<>();
dinnerMap.put("Chinese", "Green River");
dinnerMap.put("Japanese", "Sushi Express");
dinnerMap.put("Hongkong", "Australia Dairy");
request.setAttribute("dinnerMap", dinnerMap);
```

key	value
"Chinese"	"Green River"
"Japanese"	"Sushi Express"
"Hongkong"	"Australia Dairy"

- ${dinnerMap.Chinese} prints `Green River`

- For beans and Maps, using dot (.) operator is good enough.
- For a bean "person" with property "name", the following ELs are the same:
  - ${person.name}
  - ${person["name"]}

bracket [ ] operator

If EL has a variable followed by a bracket [ ]:

```
${person[name]}
```

- java.util.Map, a bean, a List, or an array
- a Map key, a bean property, a List index, or array index.

About the value inside [ ]:

- The value can be a string literal (i.e., "value")
  - E.g., ${dinnerMap["Chinese"]}
- If there are no quotes, the value is treated as an attribute and evaluated
  - E.g., ${dinnerMap[Chinese]}
  - It finds an attribute named **`Chinese`**. Then, use the value of that attribute as the key into the Map, or return null.
  - Since there is no attribute called **`Chinese`** in any of the 4 scopes, this EL does not work and returns null.

bracket [ ] operator (cont')

Consider the two request attributes "dinnerMap" and "DinnerType":

Servlet

```
Map<String, String> dinnerMap = new ConcurrentHashMap<>();
dinnerMap.put("Chinese", "Green River");
dinnerMap.put("Japanese", "Sushi Express");
dinnerMap.put("Hongkong", "Australia Dairy");
request.setAttribute("dinnerMap", dinnerMap);

String[] dinnerTypes = {"Chinese", "Japanese", "Hongkong"};
request.setAttribute("DinnerType", dinnerTypes);
```

dinnerMap:

key	value
"Chinese"	"Green River"
"Japanese"	"Sushi Express"
"Hongkong"	"Australia Dairy"

DinnerType:

index	value
0	"Chinese"
1	"Japanese"
2	"Hongkong"

JSP page:

```
${dinnerMap[DinnerType[0]]}
        → ${dinnerMap["Chinese"]}
                → Green River
```

URL:
<Lecture05 base URL>/operator

bracket [ ] operator: Access collections

- Bracket [ ] operator gives you a uniform way of accessing collections.
  - The dot operator requires you to know the key/property in advance.
  - But the [ ] operator can accept a variable that will be evaluated at run time (because the ***unquoted*** values inside [ ] are evaluated).

<Lecture05 base URL>/display_names.jsp

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.util.*, java.util.concurrent.*" %>
<%
    String[] firstNames = {"Elon", "Bill", "Mark"};
    List<String> lastNames = new CopyOnWriteArrayList<>();
    lastNames.add("Musk");
    lastNames.add("Gates");
    lastNames.add("Zuckerberg");
    Map<String, String> companyNames = new ConcurrentHashMap<>();
    companyNames.put("Gates", "Microsoft");
    companyNames.put("Musk", "Tesla");
    companyNames.put("Zuckerberg", "Meta");
    request.setAttribute("first", firstNames);
    request.setAttribute("last", lastNames);
    request.setAttribute("company", companyNames);
%>
```

Using [ ] operator: Access collections (cont')

<Lecture05 base URL>/display_names.jsp (cont')

```
<!DOCTYPE html>
<html>
    <head>
        <title>Accessing Collections with EL</title>
    </head>
    <body>
        <h1>Accessing Collections with EL</h1>
        <ul>
            <li>${first[0]} ${last[0]} (${company[last[0]]})</li>
            <li>${first[1]} ${last[1]} (${company[last[1]]})</li>
            <li>${first[2]} ${last[2]} (${company[last[2]]})</li>
        </ul>
    </body>
</html>
```

Accessing Collections with EL

- Elon Musk (Tesla)
- Bill Gates (Microsoft)
- Mark Zuckerberg (Meta)

EL Implicit Objects

EL provides its own implicit objects.

- Most of them are Maps.
- Not to be confused with *JSP implicit objects*.

param & paramValues:

- Let you access the primary request parameter (param), or
- The array of request parameter values (paramValues).

header & headerValues:

- Let you access the primary and complete HTTP request header values.
- Some of the value names must be enclosed with [ ], e.g., "User-Agent".

initParam:

- Let you access context initialization parameters.

EL Implicit Objects (cont')

cookie:

- Let you refer to incoming cookies.
- The return value is a **`Cookie`** object. This means to access its value, you have to get the **`value`** property (using the getValue() method).

pageScope, requestScope, sessionScope, applicationScope:

- These objects let you restrict where the system looks for scoped variable.
  - E.g., ${requestScope.name} only looks into HttpServletRequest

pageContext:

- Object that represents current page.
- This object has request, response, session, and servletContext properties.
  - E.g., ${pageContext.session.id}

EL Implicit Object: Example

<Lecture05 base URL>/el_implicit.jsp

```
<h1>Using EL implicit objects</h1>
<ul>
    <li>Request parameter called <code>test</code>: ${param.test}</li>
    <li>User-agent info in request header: ${header["User-Agent"]}</li>
    <li>Cookie (JSESSIONID) value: ${cookie.JSESSIONID.value}</li>
    <li>Server info: ${pageContext.servletContext.serverInfo}</li>
    <li>Request method: ${pageContext.request.method}</li>
</ul>
```

<Lecture05 base URL>/el_implicit.jsp?test=Hello+World

Using EL implicit objects

- Request parameter called `test`: Hello World
- User-agent info in request header: Mozilla/5.0 (Windows NT 10.0; Win64;
- Cookie (JSESSIONID) value: 5DF3EC365A04241F770C4F5B01F44D54
- Server info: Apache Tomcat/10.1.5
- Request method: GET

EL Operators

Category	Operators
Arithmetic	+, -, *, / (div), % (mod)
Relational	== (eq), != (ne), < (lt), > (gt),<= (le), >= (ge)
Logical	&& (and), || (or), ! (not)
Validation	empty

Examples:

```
${item.price * (1 + taxRate[user.address.zipcode])}

${empty param.userName}
```

Jakarta Standard Tag Library (JSTL)

- JSP actions and EL allows you to remove bulk of JSP scripting elements.
- However, you may need more functionality, something beyond what you can get with JSP actions and EL.
  - E.g., EL has no control-flow statements.
- Imagine that your JSP is expecting a **`userName`** parameter.
- If it does not exist, you want to call another JSP that will ask for **`userName`** from the user. That is, you want to check for some conditions…

We are sorry ... you need to log in again.
Name: [ ] Submit

- You may have to resort to scripting again…

JSTL Motivating Example

<Lecture05 base URL>/hello_user1.jsp

```
<body>
    Welcome to your page!
    <% if (request.getParameter("userName") == null) { %>
        <jsp:forward page="CollectName.jsp" />
    <% } %>
    Hello ${param.userName}.
</body>
```

<Lecture05 base URL>/CollectName.jsp

```
<body>
    We are sorry ... you need to log in again.
    <form action="hello_user1.jsp" method="get">
        Name: <input name="userName" type="text" />
        <input name="Submit" type="submit" />
    </form>
</body>
```

We are sorry ... you need to log in again.
Name: Submit

JSTL Motivating Example (cont')

- JSP provides a set of tag library for performing tasks that are common in programming.
- With JSTL and EL, you can do just about everything without using JSP scripting elements.
  - JSTL is not part of the JSP specification.
- Here is how you might handle the conditional forward with JSTL: (You need to update CollectName.jsp to go to hello_user2.jsp)

<Lecture05 base URL>/hello_user2.jsp

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
...
<body>
    Welcome to your page!
    <c:if test="${empty param.userName}" >
        <jsp:forward page="CollectName.jsp" />
    </c:if>
    Hello ${param.userName}.
</body>
```

JSTL Major Libraries

The JSTL consists of the following major libraries.

- *Core*: (programming-like) actions.
  <%@ taglib uri="jakarta.tags.core" prefix="c" **%>**
  - You may also use "http://java.sun.com/jsp/jstl/core" as the uri.
- *Fmt*: Formatting and internationalization.
  <%@ taglib uri="jakarta.tags.fmt" prefix="fmt" **%>**
- *SQL*: SQL database actions.
  <%@ taglib uri="jakarta.tags.sql" prefix="sql" **%>**
- *XML*: XML processing actions.
  <%@ taglib uri="jakarta.tags.xml" prefix="x" **%>**
- *Fn*: Functions.
  <%@ taglib uri="jakarta.tags.functions" prefix="fn" **%>**

More details: https://jakarta.ee/specifications/tags/3.0/tagdocs/

Looping Collections: JSP scripting

- Suppose you want to print the array content in an HTML table.

```
String[] companyList = {"Tesla", "Microsoft", "Meta"};
request.setAttribute("companyList", companyList);
```

- Using JSP scripting:

<Lecture05 base URL>/c_forEach.jsp

```
<h2>JSP scripting</h2>
<ul>
<%
    String[] companies = (String[]) request.getAttribute("companyList");
    for (String company : companies) {
%>
    <li><%= company %></li>
<%  } %>
</ul>
```

Looping Collections: <c:forEach>

- Suppose you want to print the array content in an HTML table.

```
String[] companyList = {"Tesla", "Microsoft", "Meta"};
request.setAttribute("companyList", companyList);
```

- The JSTL tag **<c:forEach>** allows you to **loop through** the entire companyList array (i.e., companyList attribute) and print each element.

<Lecture05 base URL>/c_forEach.jsp

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
...
<h2>JSTL</h2>
<ul>
    <c:forEach var="company" items="${companyList}">
        <li>${company}</li>
    </c:forEach>
</ul>
```

Looping Collections: <c:forEach> (cont')

The variable that holds
each element in the collection.
It's value changes with each iteration.

Thing to loop over
(Array, Collection, Map)

```
<c:forEach var="company" items="${companyList}">
    <li>${company}</li>
</c:forEach>
```

```
String[] companies = (String[]) request.getAttribute("companyList");
for (String company : companies) {
    out.println(company);
}
```

Conditional output: <c:if>

- The JSTL tag <c:if> allows you to have conditional output, e.g., only display a comment form for members with full access right.
- Suppose we have the attributes `commentList` and `memberType`.

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<html><body>
<ul>
  <c:forEach var="comment" items="${commentList}">
    <li>${comment}</li>
  </c:forEach>
</ul>
<hr>
<c:if test="${memberType eq 'full_access'}">
  Add your comments
  <form action="postcomment.jsp" method="post">
    <textarea name="input" cols="40" rows="10"></textarea>
    <input type="submit" value="Add comment" />
  </form>
</c:if>
```

Conditional output: <c:choose>, <c:when>, <c:otherwise>

- There is no “else” construct in JSTL.
- To **check for multiple tests**, we use **<c:choose>, <c:when>, <c:otherwise>**.

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
...
<c:choose>
  <c:when test="${pageContext.request.scheme eq 'http'}">
    This is an insecure Web session.
  </c:when>
  <c:when test="${pageContext.request.scheme eq 'https'}">
    This is a secure Web session.
  </c:when>
  <c:otherwise>
    You are using an unrecognized Web protocol. How did this happen?!
  </c:otherwise>
</c:choose>
```

Setting attribute / map / bean: <c:set>

<c:set> tag comes in two flavours: **var** and **target**

1. This code sets an attribute in a scope (i.e., like setAttribute() method).

```
<c:set var="userLevel" scope="session" value="full_access" />

<c:set var="Lucky" scope="request" value="${person.dog}" />
```

2. This code sets a Map value and a property of a bean.

```
<c:set target="${person}" property="name" value="Keith" />

<c:set target="${PetMap}" property="dogName" value="Lucky" />
```

- person is a bean and its name property is set to “Keith”
- PetMap is a Map and the value of the key “dogName” is set to “Lucky”.

Setting attribute / map / bean: <c:set> (cont')

```
<c:set var="Lucky" scope="request" value="${person.dog}" />
```

- **scope** is optional and is used only for **var**; the default is **page** scope.
- If the value evaluates to null, the attribute Lucky will be removed from the scope.
- If the attribute Lucky does not exist, it will be created (only if the value is not null).

```
<c:set target="${person}" property="name" value="Keith" />
```

- The target must be an expression which evaluates to the Object (not the String "id" name of the bean or Map).
- The container throws an exception, if
  - the target expression is null, or
  - the target expression is not a Map or a bean.

Working with URL using <c:url>

- Suppose your web app has a base URL http://www.example.org/forums/.
- If you place the following hyperlink in your JSP page

  Product Forum

  the user will be taken to http://www.example.org/forum.jsp
- This URL is relative to the server URL, not the web app's base URL.
- If you actually wants http://www.example.org/forums/forum.jsp , use:

```
<a href="<c:url value="/forum.jsp" />">Product Forum</a>
```

- **<c:url>** properly encodes URLs, and rewrites them if necessary to add the session ID, and can also output URLs in your JSP.
- It saves the trouble of worrying about what base URL your application is deployed to.

Working with URL using <c:url> & <c:param>

```
<c:set var="first" value="Tai Man" />
<c:set var="last" value="Chan" />
...
<c:url value="/nextpage.jsp?first=${first}&last=${last}" />
```

- The URL created by <c:url> is invalid due to the space in “Tai Man”.
- **<c:param>** can be used to encode the query parameters in <c:url> properly:

```
<c:url value="/nextpage.jsp" var="nextURL">
  <c:param name="first" value="${first}" />
  <c:param name="last" value="${last}" />
</c:url>
```

- Note the use of **var** in <c:url>, which allows us to reuse the URL, e.g.,

```
<a href="${nextURL}">Next page</a>
```

JSTL: Functions

${fn:contains(String, String)}

- test whether the first string contains one or more instances of the second string and returns **`true`** if it does.

${fn:escapeXml(String)}

- If a string you are outputting could contain special characters, you can use this function to escape those special characters.
  - E.g., **${fn:escapeXml("<p>")}** becomes **<p>**
  - Especially useful for **preventing cross-site scripting (XSS) attacks**.

${fn:join(String[], String)}

- Join an array of strings together using the specified string as a delimiter.
  - E.g., ${fn:join(emailArray, ",")}

JSTL: Functions (cont')

${fn:length(Object)}

- If Object is a string, it returns the result of calling the length method on the specified string.
- If Object is a Collection, Map, or array, it returns the size of that Collection, Map, or array.
- No other object types are supported.

${fn:toLowerCase(String)}, ${fn:toUpperCase(String)}

- Change the case of a string to all lowercase or all uppercase.

${fn:trim(String)}

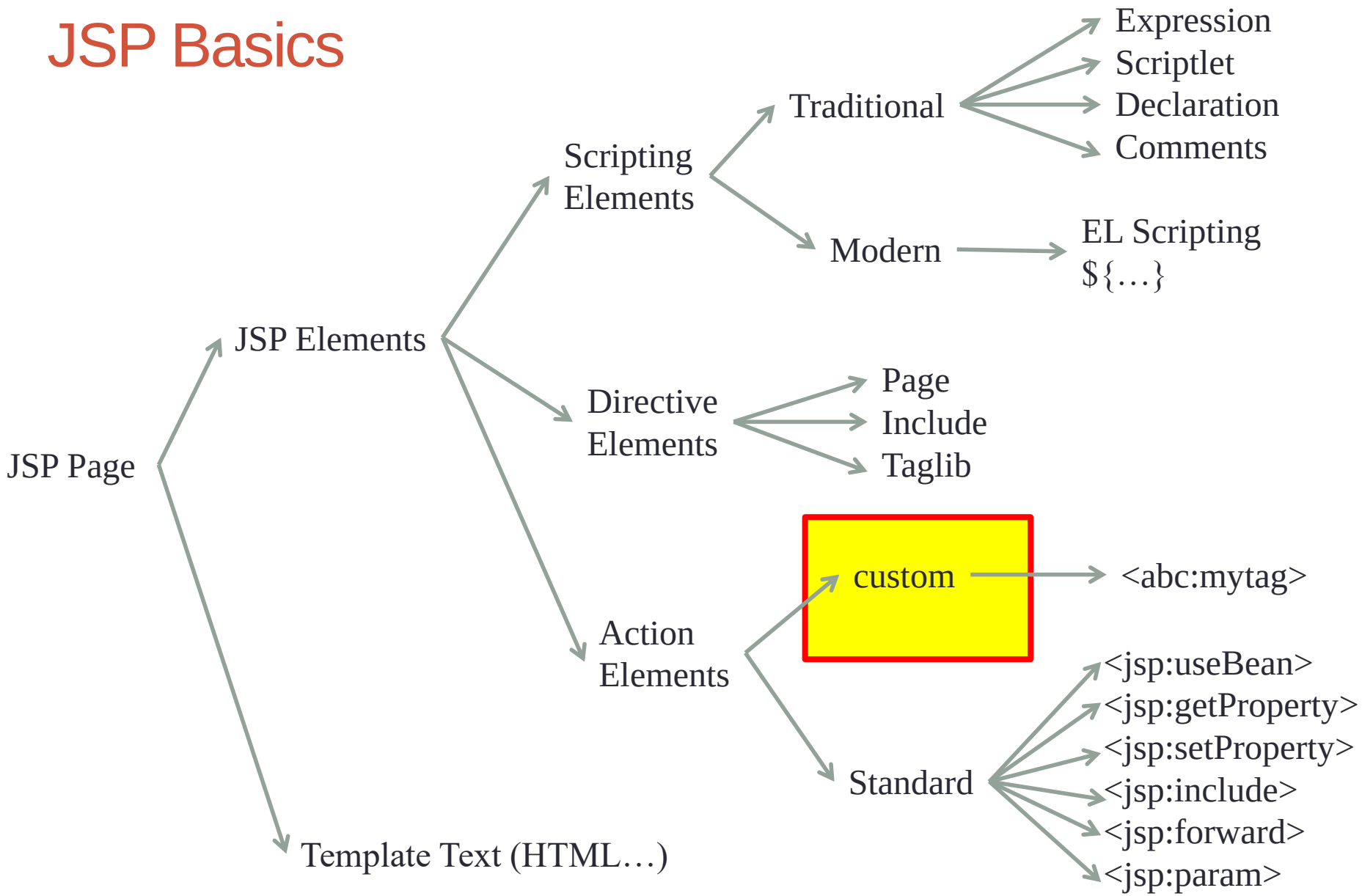
- Trim all white space from both ends of the specified string.

Other things available in JSTL

- More tags are available in JSTL. You can learn it online, e.g., https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0.html

Core	Formatting tools	XML tools
<c:out>	<fmt:formatNumber>	<x:parse>
<c:catch>	<fmt:parseDate>	<x:forEach>
<c:redirect>	<fmt:setTimeZone>	<x:transform>
<c:import>	<fmt:parseNumber>	<x:param>
<c:remove>	<fmt:setLocale>	<x:if>
<c:forTokens>	<fmt:message>	<x:out>
…	…	…

JSP Basics


JSP Page
JSP Elements
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
Template Text (HTML…)

JSP Custom Tags

- Tags are understood by a program that “reads” them; i.e., the program knows what to do with tags.
  - E.g., HTML tags are understood by browsers:

    My homepage

    <h1>My page title</h1>
- General Syntax of tags:

<tagName attributeName="attValue">Body Text</tagName>

- JSP allows you to create your own tags.
  - The behaviour of a custom tag is implemented by a Java class, called TagHandler.
  - When the JSP engine encounters a custom tag, it executes the Java code that implements the behaviour.

JSP Custom Tags: Example

TagHandler

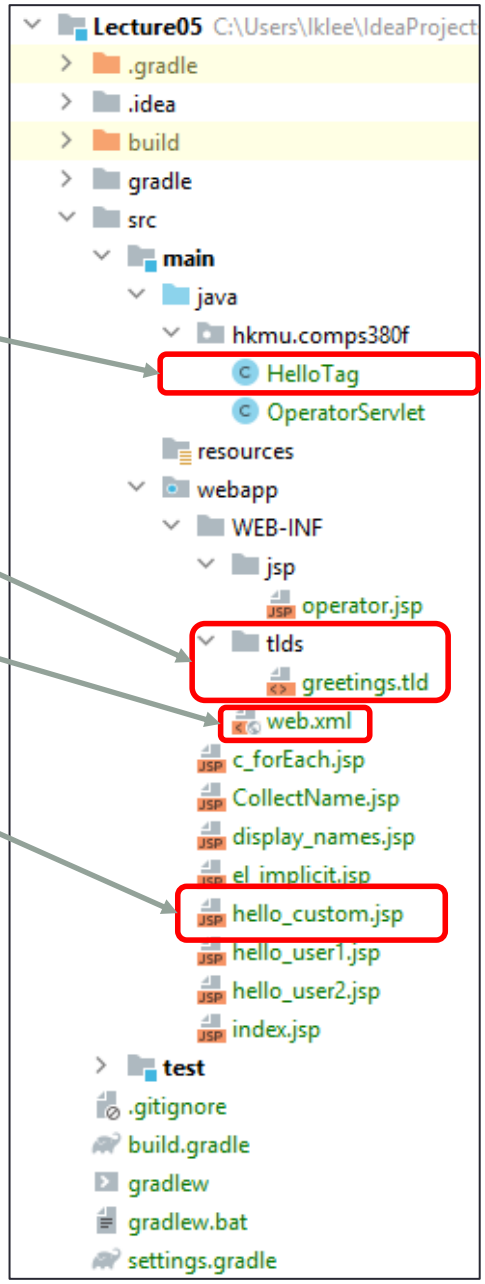
Tag Library Descriptor

Deployment descriptor

JSP page that uses the custom tag

Greeting!

Hi Tommy
My dearest Elon
Now look here Mark
Now look here Keith
My dearest Bill



JSP Custom Tags: JSP page

<Lecture05 base URL>/hello_custom.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="greet" uri="comps380f.hkmu.edu/greetings/HelloTag" %>
<!DOCTYPE html>
<html>
    <head>
        <title>Greetings!</title>
    </head>
    <body>
        <h1>Greeting!</h1>
        <greet:hello form="brief" />Tommy<br />
        <greet:hello form="effusive" />Elon<br />
        <greet:hello form="serious" />Mark<br />
        <greet:hello form="serious" />Keith<br />
        <greet:hello form="effusive" />Bill<br />
    </body>
</html>
```

Greeting!

Hi Tommy
My dearest Elon
Now look here Mark
Now look here Keith
My dearest Bill

JSP Custom Tags: Tag Library Descriptor

- The custom tag is declared in a Tag Library Descriptor: /WEB-INF/tlds/greetings.tld.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="3.1" …>
  <tlib-version>1.0</tlib-version>
  <short-name>greet</short-name>
  <!-- Ways of saying Hello. -->
  <uri>comps380f.hkmu.edu/greetings/HelloTag</uri>
  <tag>
    <name>hello</name>
    <tag-class>hkmu.comps380f.HelloTag</tag-class>
    <body-content>empty</body-content>
    <!-- Say hello. -->
    <attribute>
      <name>form</name>
      <required>true</required>
    </attribute>
  </tag>
</taglib>
```

JSP Custom Tags: TagHandler

- Actions associated with the tags are implemented: HelloTag.java

```
public class HelloTag extends SimpleTagSupport {
    private String form;

    @Override
    public void doTag() throws JspException {
        JspWriter out = getJspContext().getOut();

        try {
            String greeting = null;
            if (form.equals("brief"))  greeting = "Hi";
            if (form.equals("effusive"))  greeting = "My dearest";
            if (form.equals("serious"))  greeting = "Now look here";
            out.print(greeting + " ");
        } catch (java.io.IOException ex) {
            throw new JspException("Error in HelloTag tag", ex);
        }
    }

    public void setForm(String form) {
        this.form = form;
    }
}
```

JSP Custom Tags: Deployment Descriptor

- In the deployment descriptor (web.xml):

```
<jsp-config>
   <taglib>
      <taglib-uri>comps380f.hkmu.edu/greetings/HelloTag</taglib-uri>
      <taglib-location>/WEB-INF/tlds/greetings.tld</taglib-location>
   </taglib>
</jsp-config>
```

JSP Custom Tags: Summary

JSP that uses the tag

```
<html><body>
<% taglib prefix="mine" uri="randomThings" %>

Advisor Page<br>

<mine:advice user="${userName}/>
</body></html>
```

AdvisorTagHandler class

```
void doTag() {
  // tag implementation
}

void setUser(String user) {
  this.user = user;
}
```

TLD file

```
<taglib ...>
<uri>randomThings</uri>
<tag>
  <name>advice</name>
  <tag-class>foo.AdvisorTagandler</tag-class>
  <body-content>empty</body-conent>
  <attribute>
    <name>user</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
```