UNIT : 7

Reference :-Marty Hall, Larry Brown, "Core Servlets and JavaServer Pages Volume – 1", Pearson Education, 2nd ed.(2004)
Chapter :- 16 : Simplifying Access to Java Code : The JSP 2.0 Expression Language

MVC Approach

- servlet responds to the request;
- invokes the appropriate business logic or data access code;
- places the resultant data in beans;
- stores the beans in the request, session, or servlet context;
- forwards the request to a JSP page to present the result.

Inconvenience in MVC Approach

- Final step → presenting the results in the JSP page.
- jsp:useBean and jsp:getProperty, but these elements are a bit verbose and clumsy.
- jsp:getProperty only supports access to simple bean properties; if the property is a collection or another bean, accessing the "subproperties" requires you to use complex Java syntax

JSP 2.0 EL

JSP 2.0 EL

- simplify the presentation layer by replacing hard-to-maintain Java scripting elements or clumsy jsp:useBean and jsp:getProperty elements **with short and readable entries of the following form.**
- ${expression}

Expression Language Capabilities

- Concise access to stored objects → To output a "Scoped variable"
- E.g. (object stored with setAttribute in the PageContext, HttpServletRequest, HttpSession, or ServletContext) named saleItem use

${saleItem}

Expression Language Capabilities

- Shorthand notation for bean properties→ To output a "Scoped variable"
- E.g. To output the companyName property (i.e., result of the getCompanyName method) of a scoped variable named company, you use

${company.companyName}

- To access the firstName property of the president property of a scoped variable named company, use

${company.president.firstName}

Expression Language Capabilities

- Simple access to collection elements → To output a "Scoped variable"
- E.g. To access an element of an array, List, or Map, you use
  ${variable[indexOrKey]}
- Provided that the index or key is in a form that is legal for Java variable names, the dot notation for beans is interchangeable with the bracket notation for collections.

Expression Language Capabilities

- Succinct access to request parameters, cookies, and other request data → To access the standard types of request data, you can use one of several predefined implicit objects.
- A small but useful set of simple operators → To manipulate objects within EL expressions, you can use any of several arithmetic, relational, logical, or empty-testing operators
- Conditional output → To choose among output options, you do not have to resort to Java scripting elements. Instead, you can use ${test ? option1 : option2}.

Expression Language Capabilities

- Automatic type conversion → The expression language removes the need for most typecasts and for much of the code that parses strings as numbers.
- Empty values instead of error messages → In most cases, missing values or NullPointerExceptions result in empty strings, not thrown exceptions.

Invoking Expression Language

${expression}

- These EL elements can appear in ordinary text or in JSP tag attributes, provided that those attributes permit regular JSP expressions.
- <UL> <LI>Name: ${expression1} <LI>Address: ${expression2} </UL> <jsp:include page="${expression3}" />
- <jsp:include page="${expr1}test${expr2}" />

Preventing EL Evaluation

- In JSP 1.2 and earlier, strings of the form ${...} had no special meaning.
- So, it is possible that the characters ${ appear within a previously created page that is now being used on a server that supports JSP 2.0.
- To deactivate the expression language in that page.
  Four options for doing so :

Preventing EL Evaluation - 1

- Deactivating the expression language in an entire Web application → Use a web.xml file that refers to servlets 2.3 (JSP 1.2) or earlier.
- The JSP 2.0 expression language is automatically deactivated in Web applications whose deployment descriptor (i.e., WEB-INF/web.xml file) refers to servlet specification version 2.3 or earlier (i.e., JSP 1.2 or earlier).
- Figure→Compatible with JSP 1.2 EL should be deactivated by default.
- Figure → Compatible with JSP 2.0 EL should be activated by default.

Preventing EL Evaluation - 2

- Deactivating the expression language in multiple JSP pages → Use the jsp-property-group web.xml element to designate the appropriate pages.
- use the el-ignored subelement of the jsp-property-group web.xml element to designate the pages in which the expression language should be ignored. Here is an example that deactivates the expression language for all JSP pages in the legacy directory.
- Figure → Web application whose deployment descriptor specifies servlets 2.4 (JSP 2.0).

Preventing EL Evaluation - 3

- Deactivating the expression language in individual JSP pages → Use the isELEnabled attribute of the page directive.
- To disable EL evaluation in an individual page, supply false as the value of the isELEnabled attribute of the page directive, as follows.
  <%@ page isELEnabled="false" %>
- Note that the isELEnabled attribute is new in JSP 2.0 and it is an error to use it in a server that supports only JSP 1.2 or earlier.
- So, you cannot use this technique to allow the same JSP page to run in either old or new servers without modification.
- Consequently, the jsp-property-group element is usually a better choice than the isELEnabled attribute.

Preventing EL Evaluation - 4

- Deactivating individual expression language statements→ In JSP 1.2 pages that need to be ported unmodified across multiple JSP versions (with no web.xml changes), replace $ with $, the HTML character entity for $.
- In JSP 2.0 pages that contain both expression language statements and literal ${ strings, you can use \${ when you want ${ in the output.

Preventing EL Evaluation - 4

- Suppose you have a JSP 1.2 page containing ${ that you want to use in multiple places. In particular, you want to use it in both JSP 1.2 Web applications and in Web applications that contain expression language pages,
- to be able to drop the page in any Web application without making any changes either to it or to the web.xml file. Although this is an unlikely scenario, it could happen, and none of the previously discussed constructs will serve the purpose. In such a case, you simply replace the $ with the HTML character entity corresponding to the ISO 8859-1 value of $ (36). So, you replace ${ with ${ throughout the page. For example,
- ${blah} will portably display ${blah} to the user.

Preventing EL Evaluation - 4

- Note, however, that the character entity is translated to $ by the browser, not by the server, so this technique will only work when you are outputting HTML to a Web browser.
- Finally, suppose you have a JSP 2.0 page that contains both expression language statements and literal ${ strings. In such a case, simply put a backslash in front of the dollar sign. So, for example,
- \${1+1} is ${1+1}. will output
- ${1+1} is 2.

Preventing Use of Standard Scripting Elements

- Expression Language capability eliminates much of the need for the explicit Java scripting elements.
- to use a no-classic-scripting-elements approach throughout projects use the scripting-invalid subelement of jsp-property-group to enforce this restriction.
- Figure

Accessing Scoped Variable

- To permit the JSP page to access the data, the servlet needs to use setAttribute to store the data in one of the standard locations:
  - the HttpServletRequest,
  - the HttpSession, or
  - the ServletContext.
- Objects in these locations are known as "scoped variables.
- Note : scoped variables stored in the PageContext object, but this is much less useful because the servlet and the JSP page do not share PageContext objects.

Accessing Scoped Variable

- page-scoped variables apply only to objects stored earlier in the same JSP page, not to objects stored by a servlet.
- ${name}
- <jsp:useBean id="name" type="somePackage.SomeClass" scope="..."> → you have to know which scope the servlet used, and you have to know the fully qualified class name of the attribute.

Accessing Scoped Variable

- **Choosing Attribute Names**
- To use the JSP expression language to access scoped variables, you must choose attribute names that would be legal as Java variable names. So, avoid dots, spaces, dashes, and other characters that are permitted in strings but forbidden in variable names.
- it is technically possible for attribute names to be repeated, so you should be aware that the expression language searches the PageContext, HttpServletRequest, HttpSession, and ServletContext in that order.

Accessing Collection

- The JSP 2.0 expression language lets you access different types of collections in the same way: using array notation. For instance, if attributeName is a scoped variable referring to an array, List, or Map, you access an entry in the collection with the following:
- ${attributeName[entryName]} If the scoped variable is an array, the entry name is the index and the value is obtained with theArray[index]. For example, if customerNames refers to an array of strings,
- ${customerNames[0]} would output the first entry in the array.

Accessing Collection

- If the scoped variable is an object that implements the List interface, the entry name is the index and the value is obtained with theList.get(index). For example, if supplierNames refers to an ArrayList,
- ${supplierNames[0]} would output the first entry in the ArrayList.
- If the scoped variable is an object that implements the Map interface, the entry name is the key and the value is obtained with theMap.get(key). For example, if stateCapitals refers to a HashMap whose keys are U.S. state names and whose values are city names,
- ${stateCapitals["maryland"]}

Accessing Collection

- would return "annapolis". If the Map key is of a form that would be legal as a Java variable name, you can replace the array notation with dot notation. So, the previous example could also be written as:
- ${stateCapitals.maryland} However, note that the array notation lets you choose the key at request time, whereas the dot notation requires you to know the key in advance.

Accessing Collection

- would return "annapolis". If the Map key is of a form that would be legal as a Java variable name, you can replace the array notation with dot notation. So, the previous example could also be written as:
- ${stateCapitals.maryland} However, note that the array notation lets you choose the key at request time, whereas the dot notation requires you to know the key in advance.

Referencing Implicit Objects

- The expression language is not restricted to use in the MVC approach.
- pageContext

  The pageContext object refers to the PageContext of the current page. The PageContext class, in turn, has request, response, session, out, and servletContext properties (i.e., getRequest, getResponse, getSession, getOut, and getServletContext methods). So, for example, the following outputs the current session ID.

  ${pageContext.session.id}

Referencing Implicit Objects

param and paramValues

- These objects let you access the primary request parameter value (param) or the array of request parameter values (paramValues).
- So, for example, the following outputs the value of the custID request parameter (with an empty string, not null, returned if the parameter does not exist in the current request).
- ${param.custID}

Referencing Implicit Objects

header and headerValues

- These objects access the primary and complete HTTP request header values, respectively. Remember that dot notation cannot be used when the value after the dot would be an illegal property name. So, for example, to access the Accept header, you could use either
- ${header.Accept} or
- ${header["Accept"]} But, to access the Accept-Encoding header, you must use
- ${header["Accept-Encoding"]}

Referencing Implicit Objects

cookie

- The cookie object lets you quickly reference incoming cookies. However, the return value is the Cookie object, not the cookie value. To access the value, use the standard value property (i.e., the getValue method) of the Cookie class.
- So, for example, either of the following outputs the value of the cookie named userCookie (or an empty string if no such cookie is found).
- ${cookie.userCookie.value} ${cookie["userCookie"].value}

Referencing Implicit Objects

initParam

- The initParam object lets you easily access context initialization parameters. For example, the following outputs the value of the init param named defaultColor.
- ${initParam.defaultColor}

Referencing Implicit Objects

- pageScope, requestScope, sessionScope, and applicationScope
- These objects let you restrict where the system looks for scoped variables. For example, with
- ${name} the system searches for name in the PageContext, the HttpServletRequest, the HttpSession, and the ServletContext, returning the first match it finds. On the other hand, with
- ${requestScope.name} the system only looks in the HttpServletRequest.

EL Operators

- The JSP 2.0 expression language defines a number of arithmetic, relational, logical, and missing-value-testing operators.

  Note
- Use the expression language operators for simple tasks oriented toward presentation logic (deciding how to present the data).
- Avoid using the operators for business logic (creating and processing the data). Instead, put business logic in regular Java classes and invoke the code from the servlet that starts the MVC process.

EL Arithmetic Operators

- + and –
- These are the normal addition and subtraction operators, with two exceptions. First, if either of the operands is a string, the string is automatically parsed to a number (however, the system does not automatically catch NumberFormatException). Second, if either of the operands is of type BigInteger or BigDecimal, the system uses the corresponding add and subtract methods.

EL Arithmetic Operators

- *, /, and div
- These are the normal multiplication and division operators, with a few exceptions. First, types are converted automatically as with the + and – operators. Second, the normal arithmetic operator precedence applies, so, for instance,
- ${ 1 + 2 * 3} returns 7, not 9. You can use parentheses to change the operator precedence. Third, the / and div operators are equivalent; both are provided for the sake of compatibility with both XPath and JavaScript (ECMAScript).

EL Arithmetic Operators

- % and mod
- The % (or equivalently, mod) operator computes the modulus (remainder), just as with % in the Java programming language.

EL Relational Operators

- == and eq
- These two equivalent operators check whether the arguments are equal. However, they operate more like the Java equals method than the Java == operator. If the two operands are the same object, true is returned. If the two operands are numbers, they are compared with Java ==. If either operand is null, false is returned. If either operand is a BigInteger or BigDecimal, the operands are compared with compareTo. Otherwise, the operands are compared with equals.

EL Relational Operators

- != and ne
- These two equivalent operators check whether the arguments are different. Again, however, they operate more like the negation of the Java equals method than the Java != operator. If the two operands are the same object, false is returned. If the two operands are numbers, they are compared with Java !=. If either operand is null, true is returned. If either operand is a BigInteger or BigDecimal, the operands are compared with compareTo. Otherwise, the operands are compared with equals and the opposite result is returned.

EL Relational Operators

- < and lt, > and gt, <= and le, >= and ge
- These are the standard arithmetic operators with two exceptions. First, type conversions are performed as with == and !=. Second, if the arguments are strings, they are compared lexically.

EL Logical Operators

- &&, and, ||, or, !, not
- These are the standard logical AND, OR, and NOT operators. They operate by coercing their arguments to Boolean, and they use the normal Java "short circuit" evaluation in which the testing is stopped as soon as the result can be determined. && and and are equivalent, || and or are equivalent, and ! and not are equivalent.

empty Operator

- This operator returns true if its argument is null, an empty string, an empty array, an empty Map, or an empty collection. Otherwise it returns false.