CH-07 - Actions – XML - EL

1. What are Standard actions?
    a. Standard actions are used for the same purpose as Java language-based scripting: Most things we can achieve with standard actions are achievable with other scripting elements. However they are written using entirely conventional XML syntax. They can be used by the nonprogrammer to introduce dynamic behavior that would otherwise entail Java language knowledge.
    b. Both standard and custom actions are similar in appearance: XML elements that encapsulate functionality on a JSP page. The difference is that you can rely on any J2EE-compliant JSP container to provide support for all the standard actions defined in the JSP spec. Not so custom actions, because they are customized for your web applications.
2. Is it necessary for the java objects that need to be used by standard actions to follow JavaBeans specification?
    a. Yes. Java objects come in many shapes and sizes, so standard actions can work only if those objects obey at least some minimal conventions. The idea of JavaBeans is that you can have Java components (typically classes) that can be interrogated by interested software, using the reflection techniques. By interrogating the methods available on a JavaBean, a standard action can obtain information about the properties that the bean supports—in other words, the data that it stores.
3. Give the general syntax of actions?
    a. The general syntax of actions, whether standard or custom:

        ```
        <prefix:tagname firstAttribute="value" secondAttribute="value"> ...
        </prefix:tagname>
        ```

        Each standard action element consists of a start tag, <prefix:tagname>, and an end tag of the same name, </prefix:tagname>. The start tag may contain named attributes, separated from their corresponding value by equal signs. The value is typically surrounded by double quotes or by single quotes. A standard action may have a body, but it often has no body at all.

4. The prefix for all standard actions is ………
    a. Jsp
5. List down all the standard actions with the jsp prefix?
    a. The standard actions are:
        i. jsp:attribute
        ii. jsp:body
        iii. jsp:element
        iv. jsp:fallback
        v. jsp:forward
        vi. jsp:getProperty
        vii. jsp:include
        viii. jsp:output
        ix. jsp:param
        x. jsp:params
        xi. jsp:plugin
        xii. jsp:setProperty
        xiii. jsp:useBean
6. The ……… standard action declares a JavaBean instance and associates this with a variable name.
    a. <jsp:useBean>
7. Describe the use of <jsp:useBean> standard action?
    a. The <jsp:useBean> standard action declares a JavaBean instance and associates this with a variable name. The instance is then available for use elsewhere in your JSP page: either in Expression Language, other standard actions, or even in Java language scripting.
    b. Eg. <jsp:useBean id="theDog" class="animals.Dog" />
        i. The class attribute must specify the fully qualified name of a class.
        ii. animals.Dog must obey JavaBean conventions.
        iii. animals.Dog must be visible somewhere in the web application—mostly this means it will exist as a class in WEB-INF/classes or within a JAR file in WEB-INF/ lib.

        iv. All ids for beans on a page must be unique.

8. Give two standard actions to write and read properties on our bean set up using the <jsp:useBean> standard action?
    a. <jsp:setProperty> and <jsp:getProperty>
9. How will the jave code in _jspService() reference the object created by <jsp:useBean> tag ?
    a. In two ways:
        i. As a local variable in the method, whose name comes from value of the id attribute.
        ii. As an attribute in some scope or other—page, request, session, or application. If we don’t specify the scope, it will be page scope.
    b. Eg. <jsp:useBean id="theDog" class="animals.Dog" />
        i. theDog becomes a local variable.
        ii. In this case, theDog is the name of the attribute.
10. What will be the scope of the attribute created by <jsp:useBean> ? Can we specify the attribute scope?
    a. If we don’t specify the scope, it will be page scope. We can specify the scope as below:
        i. <jsp:useBean id="theDog" class="animals.Dog" scope="session" />
            1. The valid values for scope are page, request, session, and application.
    b. Specifying the scope attribute creates or uses an attribute in the specified session and make it available as a local variable in the page scope with that name.
11. When we use <jsp:useBean> to create an attribute in one scope and another attribute with same name is already available in that scope, what will happen?
    a. The jsp:useBean> recycles the existing bean; it doesn’t create a new one.
12. An attribute of <jsp:useBean> that offers possibility of using a serialized bean from your file system?
    a. beanName
13. Describe the basic use of <jsp:setProperty>?

   a. The purpose of <jsp:setProperty> is to set the values of one or more properties on a bean previously declared with <jsp:useBean>. The property attribute specifies a property on the bean.
   b. Attributes of <jsp:setProperty> standard action are name, property, param, value. The name is the name of the bean; property is the property for which we need to assign the value. If we need to set the value of a request parameter, then we can use param attribute instead of value and give a valid request parameter as the value of the param attribute.
   c. **Eg:** <jsp:setProperty name="theDog" property="weight" value="6.4" />
      i. Because the property here is “weight,” then the underlying code will assume the existence of a setWeight() method on the theDog bean.
      ii. The value attribute supplies the data for the property — or in code terms, the parameter that is passed into the setter method. Instead of supplying a literal value, you can even substitute an expression. But this feature is not available to any other attributes.
         1. <% float w = 6.4f; %>
         2. <jsp:setProperty name="theDog" property="weight" value="<%=w%>" />

14. Can we use <jsp:setProperty> and <jsp:getProperty> without a previous <jsp:useBean> in a page?
    a. We can use <jsp:setProperty> and <jsp:getProperty> without a previous <jsp:useBean> if an attribute of the right name exists in the PageContext.
    b. However, it’s good practice to include <jsp:useBean> before these actions in the same JSP page. After all, it won’t replace beans of the same name that you have set up by other means, and it will create beans of the right name that don’t exist already.
15. Why is it a good practice to include <jsp:useBean> before <jsp:setProperty> and <jsp:getProperty> even if it will work otherwise if an attribute of the right name exists in the PageContext?

   a. Because <jsp:useBean> won't replace beans of the same name that you have set up by other means, and it will create beans of the right name that don't exist already.

16. If your <jsp:setProperty> and <jsp:getProperty> standard actions try to access an attribute that doesn't exist, what will happen?
   a. If your <jsp:setProperty> and <jsp: getProperty> standard actions try to access an attribute that doesn't exist, they will fail with HTTP 500 errors returned to the requester.
17. Describe : <jsp:setProperty name="theDog" property="weight" param="dogWeight" />.
   a. This is shorthand for saying take the request parameter called "dogWeight," and use the value for this to set the property called "weight" on the bean called "theDog." This is equivalent to:
      i. <jsp:setProperty name="theDog" property="weight" value="<%= request.getParameter("dogWeight") %>" />
18. What will be the value of the property weight in : <jsp:setProperty name="theDog" property="weight" /> ?
   a. The underlying code will look for a parameter (from the ServletRequest or HttpServletRequest) called "weight" and use this to set the property value for "weight" on "theDog."
19. Describe : <jsp:setProperty name="theDog" property="*" /> ?
   a. Any property whose name matches a request parameter name will have its value preloaded from that request parameter.
20. Describe <jsp:getProperty> ?
   a. You use it to output the value of a bean's property to the response. There are two attributes to supply: name (the name of the bean) and property (the name of the property). They're both mandatory.
      i. <jsp:getProperty name="theDog" property="weight" />
21. The standard action ………. can be used to include the response from another file within your JSP page output.
   a. <jsp:include>

22. Can we use <jsp:include> to include files that don't exist at the point where you deploy your including pages?
    a. You might use <jsp:include> to include files that don't exist at the point where you deploy your including pages. There's no check on the existence of the page specified in <jsp:include> during the translation phase.
    b. However you'll get a run-time error if you let your users access JSPs that try to include a page that doesn't exist. So we should introduce controls that prevent access to the including JSP until the files needed for inclusion are actually present in your web application directory structure.
23. Can we include an external file outside the web application using the <jsp:include> standard action?
    a. No.
24. Can we include non-jsp files using the <jsp:include> standard action?
    a. Yes. You can include any file of any MIME type.
25. Is this legal: <jsp:include page='<%= request.getParameter("thePage") %>' />. Explain.
    a. Yes. This is possible because <jsp:include> runs at request time.
26. What is the importance of the flush attribute of the <jsp:include> standard action?
    a. JSP page output is buffered as a rule—not immediately committed to the response. If you set the flush value to "true," this has the effect of fl ushing the buffer in the including page (i.e., committing the response so far) before anything is done about including the target page.
27. Can we have the below line in an included JSP page:

    <% response.setHeader("Date", utcFormatDate); %>

    a. It will be simply ignored in the included JSP page. Included pages can't do anything to the response header similar to the way servlets can't if anything has been written to the response. The assumption is that somewhere along the chain to the included page, some part of the response has been written. Even if you set the flush attribute to "false," and both the including and included page have unfilled, unflushed buffers, there are still restrictions on included pages. A servlet that has been included from another servlet by a RequestDispatcher object is treated in the same way.

28. Differentiate between <jsp:include> Standard Action **and** <%@ include %> Directive?
    a. <jsp:include> Standard Action
        i. Attributes: page (and flush)
        ii. Response from target page included at request time.
        iii. Target page to include can be soft-coded as an expression.
        iv. Can execute conditionally in the middle of page logic.
        v. Target page doesn't have to exist until request time.
        vi. Always includes the latest version of the target page.
    b. <%@ include %> Directive
        i. Attribute: file
        ii. Target file included during translation phase.
        iii. Target file must be a hard-coded literal value.
        iv. Will be processed unconditionally—can't be embedded in page logic.
        v. Target file must exist at translation time.
        vi. Does not necessarily include the latest version of the target file: depends on your container (not mandated by the JSP specification).
29. Describe the use of <jsp:forward> standard action?
    a. The purpose of this standard action is to forward processing to another resource within the web application. There is only one mandatory attribute, which is page="URL." The JSP specification says that a "<jsp:forward> effectively terminates the current page,". Therefore any line after "<jsp:forward> won't be executed.
30. List down the cases where you might get IllegalStateException when using <jsp:forward> ?
    a. The IllegalStateException will be thrown if any part of the response has already been committed and we try to use <jsp:forward>. Below are few specific cases:
        i. There is no buffer, and even one character has been written to the response.
        ii. The buffer has been explicitly flushed (response.flushBuffer()).
        iii. The buffer has been automatically flushed on filling up (in a JSP, this will happen by default).

31. While including or forwarding, can we add in additional parameters to the request?
    a. Yes. For this, we can use the <jsp:param> standard action and include it in the body of a <jsp:include> or <jsp:forward>.
32. Give a reason for including a body in <jsp:include> or <jsp:forward>?
    a. We can use the <jsp:param> standard action to add additional parameters to the request and include it in the body of a <jsp:include> or <jsp:forward>.
33. While using the <jsp:param> standard action to add additional parameters to the request using along with < jsp:include> or <jsp:forward> what will be the lifetime of those variables?
    a. They only last for the duration of the include or forward. Once you're back in the including or forwarding JSP page, the parameters disappear.
34. While using the <jsp:param> standard action to add additional parameters to the request using along with < jsp:include> or <jsp:forward> what will happen if there are variables with the same name?
    a. They don't replace existing parameters of the same name—they merely augment the list of values. (Parameters—unlike attributes—can have multiple values for the same name.) When they do augment the list of values, their values come at the front of the list.
35. Describe the use of <jsp:attribute> standard action?
    a. The jsp:attribute element allows you to define the value of a tag attribute in the body of an XML element instead of in the value of an XML attribute.
        i. <jsp:attribute name="attributeName" [trim= "true | false"] />
    b. All JSP standard actions and custom actions can contain a jsp:attribute standard element as a substitute for any of its attributes. One use case in which jsp:attribute is particularly helpful is where the value of an attribute is the result of a multi-line expression, which would not fit in the value of an attribute in the start tag of the action.
    c. If an action contains any jsp:attribute elements and the action also has a body, it must use the jsp:body tag to represent the body.
36. What is a JSP document?

   a. It’s JSP page source that’s written in XML. Quite often, you use a JSP document to produce XML as well.

37. List few reasons for moving to XML-style page source for JSPs?
    a. If you have an XML fi le you want to produce, it can immediately become the template text for a piece of JSP Page Source—all that remains is to mark it up with some more XML for the dynamic parts.
    b. You can check that your page source is valid in XML terms, using proper XML validators. If you use XML-authoring tools (such as XML Spy), then those same tools can handle the production and validation of your JSP page source as well as other XML fi les you write.
    c. Arguably, XML-style source is easier to write and read than a mishmash of template text and Java language.
    d. Many newer, trendier tool developments use the new xml based syntax.
38. Give the XML based syntax for <% ... %>?
    a. <jsp:scriptlet>... </jsp:scriptlet>
39. Give the XML based syntax for <%= ... %>?
    a. <jsp:expression>... </jsp:expression>
40. Give the XML based syntax for <%! ... %>?
    a. <jsp:declaration>... </jsp:declaration>
41. Give the XML based syntax for scriptlets?
    a. <jsp:scriptlet>... </jsp:scriptlet>
42. Give the XML based syntax for expressions?
    a. <jsp:expression>... </jsp:expression>
43. Give the XML based syntax for declarations?
    a. <jsp:declaration>... </jsp:declaration>
44. Give the XML based syntax for <%@ page attr=“value” %>?
    a. <jsp:directive.page attr=“value” />
45. Give the XML based syntax for <%@ include file=“abc.txt” %>?
    a. <jsp:directive.include file=“abc.txt” />
46. Give the XML based syntax for <%@ taglib prefix= “abc” uri="...” %>?
    a. xmlns:abc="...”

47. Give the XML based syntax for page directive?
    a. <jsp:directive.page attr="value" />
48. Give the XML based syntax for include directive?
    a. <jsp:directive.include file="abc.txt" />
49. Give the XML based syntax for taglib directive?
    a. xmlns:abc="..."
50. Give the JSP based and XML based syntax for comments?
    a. Exclude from translation (and output)
        i. JSP Based old syntax
            1. <%-- ... --%>
        ii. XML Syntax
            1. <!-- ... -->
    b. Include HTML comment in HTML output
        i. JSP Based old syntax
            1. <!-- ... -->
        ii. XML Syntax
            1. &lt;!-- ... --&gt;
51. How can you handle special characters like '<' ans '>' inside <jsp:scriptlet> element?
    a. You have two options to deal with this.
        i. The first is to escape the source code, using as an xml "entity"—beginning with an ampersand (&) and ending in a semicolon (;).
            1. < sign is replaced with the entity &lt;.
            2. > sign is replaced with the entity &gt;
    b. The second option is to mark up the offensive part as XML character data.
        i. <jsp:scriptlet>for (int i = 20; i <![CDATA[<]]> 30; i ++) {</jsp:scriptlet> or,
        ii. <jsp:scriptlet><![CDATA[for (int i = 10; i < 20; i ++) {]]></jsp:scriptlet>
52. If you have some content that needs to be placed in the "no-man's land" of a bodiless tag, then wrap it up with ………
    a. <jsp:text> ... </jsp:text>.

53. For using a standard action in a JSP document do we need to specify the namespace for the JSP prefix?
    a. Yes. The <jsp:directive.page> element can contain an additional attribute like:
        i. <jsp:directive.page xmlns:jsp=http://java.sun.com/JSP/Page contentType="text/html" />
    b. In our example above, the namespace is associated only with <jsp:directive.page>. Here is the same example again with the namespace transferred to <html>, the root element for XHTML documents:
        i. <html xmlns:jsp="http://java.sun.com/JSP/Page" >
        ii. <head><title>Namespaces</title></head>
        iii. <jsp:directive.page contentType="text/html" />

    Now any standard action can be used anywhere in the document without repeating the namespace, for it's available throughout—the prefi x jsp: is suffi cient.
54. Can we have XML based syntax in a normal JSP page?
    a. You can include as much or as little of the XML syntax in a normal JSP page as you like—this is to encourage you to migrate your JSP pages to XML syntax at a pace to suit.
55. Is a page written in bona fide XML a JSP document and will be treated as such?
    a. No. The page will continue to work, but the JSP container is likely to treat it as a standard syntax page, unless you tell in some way that it is a JSP document.
56. How can you make sure the container identifies a page as a JSP document?
    a. There are three approaches that identify a page as a JSP document:
        i. Ensure that your web application deployment descriptor web.xml is at version level 2.4 and that the file with your JSP page source has the extension .jspx.
        ii. Ensure that your web application deployment descriptor web.xml is at version level 2.4, and include some appropriate settings in deployment descriptor's <jsp-config> element. Here's an example configuration:
            1. <jsp-config>
            2. <jsp-property-group>
            3. <url-pattern>/jspx/*</url-pattern>

4. <is-xml>true</is-xml>
5. </jsp-property-group>
6. </jsp-config>

This says that any for any file accessed with a URL ending in /jspx /anythingatall.any within the web application, treat this as a JSP document. The <is-xml> element takes two valid values: true (treat these as JSP documents with XML syntax) or false (treat these documents as JSP pages with standard syntax). The <url-pattern> element works in just the same way we saw within the <servlet-mapping> element

iii. Enclose your page source with the root element <jsp:root>. This element is backward-compatible with previous versions of the JSP container, so it doesn't rely on a particular version level for web.xml.

1. <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page">

57. List two advantages for using <jsp:root> element for making a file a JSP document?
    a. If you need to remain compatible with older containers, or older applications in newer containers, this will work. An older-style web.xml won't matter.
    b. If your source files can't have the .jspx extension for some reason, and if their URL patterns are too diverse to warrant defining inside the <jsp-config> element.
58. In a JSP document, by default an XML header statement appears at the very beginning of the page output, looking like this: <?xml version="1.0" encoding="UTF-8"?>. If what you're producing is not XML, how can you suppress this?
    a. There are a couple of approaches:
        i. Use <jsp:root> as your root element. This suppresses the XML header statement by default (if you're using <jsp:root> for some other reason, there are ways to retain the XML header statement if you actually need it).
        ii. Include a <jsp:output> element as follows: <jsp:output omit-xmldeclaration=" true" />.
59. By default, a JSP document wants to produce XML. Is it dependent on the MIME type?
    a. No. This is regardless of the MIME type that you set with <jsp:directive.page contentType="..." />.
60. What is the importance of EL in Java?

   a. Expression Language (EL) is all about the EL-imination of Java syntax from your pages. The EL can be used to easily access data from the JSP pages. The EL simplifies writing script-less JSP pages that do not use Java scriptlets or Java expressions and thus have a more controlled interaction with the rest of the Web Application. Expression Language provides an alternative to the expression aspect of Java language scripting—<jsp:expression>...</jsp: expression> or <%...%> . EL by itself is not a replacement for scriptlets.

61. Why is EL used most often in conjunction with JSTL?
    a. Expression Language began life as part of the JSP Standard Tag Library (JSTL). EL is now incorporated as part of the JSP 2.0 specification and is entirely independent of JSTL. However, it's only with JSTL that it fully comes into its own. EL can supply only the equivalent of the "right-hand side of the equal sign" in a typical computing statement. For example, EL lacks any looping constructs. JSTL supplies the missing pieces, and hence EL used most often in conjunction with JSTL.
62. How can you enable EL?
    a. Expression Language can be enabled or disabled in three different ways.
        i. The page directive attribute isELEnabled can turn on EL for a single page—or not.
        ii. The <jsp-propertygroup> element, which has a subelement <el-enabled>. The <jsppropertygroup> element is the subelement of <jsp-config>.
        iii. EL is enabled at an application level by having a deployment descriptor at servlet level 2.4. A previous deployment descriptor level indicates that EL should be switched off.
63. Give the syntax of EL usage and its importance?
    a. An expression begins with ${ and ends with }. The part between the curly braces must be a valid EL expression. The string in the JavaServerPage source code is subject to translation, like anything else in the page. Translation checks syntax validity but won't check that the variables you use actually exist (remember that translation incorporates compilation).

b. At run time, the string representing the expression is sent to a method called resolveVariable(), in an object supplied by your JSP container provider of type VariableResolver. This returns an object, which is sent to the JSP output stream—typically via an out.print() statement in your generated servlet source. Mostly, the expression will resolve one way or another. Even if your variables don't exist, sensible defaults are provided, which mostly prevent the expression ending in a run-time error. EL is equally valid in standard JSP syntax or JSP document (XML) syntax.

64. Will the below syntax work?

    <% request.setAttribute("anAttribute", ${aValueFromEL});

    a. No, it won't work. You do have to keep Java code (such as scriptlets) free of EL (after all, EL is not valid Java syntax).

65. What do you know about EL Literals?

    a. EL has a smaller range of literals than Java. The ones it does use are similar. The different values are Boolean, Integer, floating point, Strings and null. Because you don't declare variables or assign to variables in EL, there are no explicit keywords for types; nonetheless, there are five that are defined.

66. List down the types of operators in EL?

    a. EL operators (like EL literals) offer a subset of what's available in the Java language:

        iv. Arithmetic

            1. There are five arithmetic operators—for addition (+), subtraction (-), multiplication (*), division (/), and modulo (%). As you can see, the operator symbols are identical to Java. However, there are alternative forms for the division and modulo operators—div and mod, respectively.

        v. Relational

            1. EL has a full complement of relational operators, which have conventional and alternate forms. Alternative forms exist to make writing JSP documents that much easier. For example Greater than can be '>' or gt. To avoid having to use escape sequences such as

&gt;= every time you want to express “greater than or equals” in an expression, use ge instead. e.g ${9 ge 3}. The result of a relational operation is boolean true or false. Under the covers, the String equals() method is invoked rather than a straight comparison of objects. In general, EL relational evaluation will invoke useful comparison methods on objects (such as equals() and compareTo()) when they are appropriate and available.

vi. Logical

1. EL has a more limited set of logical operators than the Java language. There is a symbolic and alternative form. Logical “and” can be expressed as ‘&&’ and ‘and’. These operators allow you to join conditional tests together to return a composite boolean result. For example, ${9 > 3 && "z" gt "a"} would return true. Like Java, EL will evaluate only the left-hand side of an expression involving && and ||, if that is sufficient to intuit the overall result.

vii. Empty

1. EL’s empty operator can be invoked like this: ${empty obj}. This expression will evaluate to true if obj represents something null—as would happen if the obj attribute didn’t exist. There are other circumstances where ${empty obj} results in true, which is any of the following:
    a. obj is an empty string (“”).
    b. obj is an empty array.
    c. obj is an empty Map or an empty Collection.

67. Describe the addition operation in EL?

a. Addition is expressed like this: ${a + b}. If either of attributes a and b doesn’t exist, and is null, that’s not a problem—they are treated as zero values. A zero-length string—“”—is likewise treated as zero. The addition operator in EL—unlike Java—is not overloaded to handle string concatenation. The following calculation won’t work: ${"Not a Number" + 3.0}. You will get a javax.servlet.jsp.el.ELException, complaining that “Not a Number” cannot be converted to a java.lang.Double value.

68. How is the Subtraction expressed in EL?
    a. Subtraction is expressed as: ${a — b}.
69. How is the Multiplication expressed in EL?
    a. Multiplication is expressed ${a * b}.
70. Describe the division operation in EL?
    a. Division is expressed ${a / b} or ${a div b}. Even if the inputs are both integers, double division is performed. There is no direct EL equivalent for Java’s integer division behavior.
    b. Being as EL division is always double division, it behaves like Java floating decimal division, so divide by zero is not an error but results in an answer of “**Infinity**.”
71. Describe the modulo operation in EL?
    a. Modulo is expressed ${a % b} or ${a mod b}. This time, unlike, division, integers are respected as integers, but a double for either input causes the calculation to be worked as a double.
72. Are the precedence rules same for EL and normal java?
    a. For all the operators— arithmetic, relational, logical—the precedence rules work in the same way as for the equivalent operators in the Java language. Parentheses can (and should) be used to clarify potential misinterpretations of code. Expressed crudely, the order is as follows:
        i. not, empty;
        ii. multiplication, division, modulo;
        iii. addition, subtraction;
        iv. relational;
        v. and, or.
73. How can you access maps, arrays and lists in EL?
    a. EL can access properties on beans with the dot operator (.) or square brackets([]). Similarly EL can access a map to get the value with the dot operator (.) or square brackets([]). When using the square brackets([]) syntax the bracket should either

contain a key literal or an attribute which contain the key literal. EL can access items in arrays or java.util.List objects with square brackets.

b. ${appl.properties.name} will execute appl.getProperties() and get the value for a key 'name' from the Map returned. ${appl.properties[prop]} will execute appl.getProperties() and get the value for a key which is the value of the attribute prop from the Map returned.

74. Describe the use of the '.' and [ ] Operators for accessing object properties?

a. For using EL to display the properties of an object, object has to be a JavaBean—at least in the sense of having "getter" methods for those properties. Then we can access the properties using either a '.' or '[ ]' Operator.

    vi. ${currentDog.name}
    vii. ${currentDog["name"]}

b. You're not limited to one level, either. You can get the properties of objects within objects:

    viii. ${currentDog.father.name}
    ix. ${currentDog["father"]["name"]}

c. You can even mix and match double quotes and single quotes, as long as you are consistent within any particular pair of square brackets.

    x. ${currentDog["father"]["name"]}
    xi. ${currentDog['father']['name']}
    xii. ${currentDog["father"]['name']}

75. Give output and explain:

    xiii. <% String[] dayArray = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
    xiv. pageContext.setAttribute("days", dayArray); %>
    xv. <% pageContext.setAttribute("two", new Integer(2));
    xvi. pageContext.setAttribute("three", "3"); %>
    xvii. ${days[two]}
    xviii. ${days[three]}

a. It would output "Wed Thu":

a. The first page attribute, called “two,” is set to an Integer with a value of 2. So ${days[two]} gets the third value in the array—“Wed.”
b. The second page attribute, called “three,” still works when loaded with a String, provided that a method like Integer.parseInt() can extract an int value from the String.

76. What will happen if we have an array of 5 values and we call ${days[7]}?
    a. EL silently suppresses this problem—you just get blank output. Even if you use an attribute name that doesn’t exist—${days[notAnAttributeName]}—nothing goes wrong; you just get blank output.
77. Given this page attribute, <% pageContext.setAttribute(“four”, “the_Word_Four”); %> the expression ${days[four]} would result in what output?
    a. This would end in a run-time error (ELException). This is because the attribute
    b. supplied is a valid attribute but can’t be converted to an integer value. However if you use an attribute name that doesn’t exist—${days[notAnAttributeName]}—nothing goes wrong; you just get blank output.
78. Give output:

```
xix.    <jsp:directive.page import="java.util.*" />
xx.     <% Map longDays = new HashMap();
xxi.    longDays.put("WED", "Wednesday");
xxii.   longDays.put("THU", "Thursday");
xxiii.  longDays.put("FRI", "Friday");
xxiv.   longDays.put("SAT", "Saturday");
xxv.    pageContext.setAttribute("longDays", longDays);
xxvi.   pageContext.setAttribute("wed", "WED");
xxvii.  %>
xxviii. <br /> ${longDays[wed]}
xxix.   <br /> ${longDays["THU"]}
xxx.    <br /> ${longDays.FRI}
```

    a. The output from this code is
        a. Wednesday
        b. Thursday

        c. Friday

    b. In general terms, expressions accessing a Map work on this principle: ${nameOfMap[keyValue]}. It doesn't matter if the key value is a literal ("THU") or derived from an attribute (wed). If you use a Map's key value as if it were a property name on a bean, you will still find the corresponding value.

79. What are **EL Implicit Objects**?
    a. It has its own set of implicit objects. All the EL implicit objects, with the exception of pageContext, are of type java.util.Map. The available implicit objects are pageContext, pageScope, requestScope, sessionScope, applicationScope, param, paramValues, header, headerValues, cookie, initParam.
80. Describe the use of pageScope, requestScope, sessionScope, and applicationScope implicit objects of EL?
    a. These implicit objects are used to access attributes in a given scope. If you want to target only a session scope attribute, then ${sessionScope.myAttribute} will do the trick. All the alternative Map syntaxes will work as well—${applicationScope["myAttribute"]} to find the attribute in application scope, for example.
81. If you call an attribute like ${myAttribute} which scope will it refer to?
    a. Under the covers, PageContext.find("myAttribute") is used to search all scopes through page, request, session, and application, stopping when it finds an attribute of the right name.
82. Let's suppose that your HTTP header request contains the following query string: ?myParm=firstValue&myParm=secondValue. Let's also suppose this is a GET request, so there are no additional parameter values for myParm hidden in a POSTed request body. What would be the result of ${param.myParm} and ${paramValues.myParm[1]} and paramValues["myParm "][1] and ${param.myParm[1]}?
    a. The result of ${param.myParm} is "firstValue." The result of ${paramValues.myParm[1]} and paramValues["myParm "][1] is "secondValue.". However ${param.myParm[1]} will throw exception.
    b. The implicit object param can be used solely to access the first value of a parameter.

c. **paramValues** returns all the values associated with a named ServletRequest parameter as a Map. To get on the values array for the key num, we can either use the dot operator based syntax (paramValues. myParm) or the square bracket based syntax (paramValues["myParm "]). And then use the square bracket based syntax on the array of values (paramValues.myParm[1] or paramValues["myParm "][1]).
d. EL implicit object param returns the first value associated with a named ServletRequest parameter as a String. We cannot have an array operation on that. It will throw exception.

83. Describe the initParam implicit object of EL?
    a. This is used to access ServletContext initialization parameters, whose values are available across the entire web application. The syntax is exactly as for param, so ${initParam.myParm} is used to return the value of an initialization parameter named “myParm.”
84. Which of the below syntax is correct for an EL:
    xxxi. ${cookie.JSESSIONID.value}
    xxxii. ${cookie["JSESSIONID"].value}
    xxxiii. ${cookie["JSESSIONID"]["value"]}
    a. All are correct.
85. Briefly describe the cookie implicit object of EL?
    a. A cookie object is a map of HttpServletRequest cookie names and cookie objects. So cookie names will be the keys and cookie objects will be the values. So we can use the EL map syntax as **cookie.cookieName** to retrieve a value.
86. Suppose that I set up a page attribute called “header”; then ${header} would refer to the implicit object header or page attribute?
    a. An implicit object name always takes precedence. Even if I set up a page attribute called “header”; ${header} would still refer to the implicit object header, not my page attribute (or attribute in any other scope, come to that).
87. What is the significance of using type attribute in **<jsp:useBean>** without the class attribute?

    a. Using type in **<jsp:useBean>** without the class attribute relies on the fact that the bean has already been created. You cannot use **type** attribute alone while creating a bean.

88. What is the significance of using type attribute along with class attribute in **<jsp:useBean>**?
    a. A **<jsp:useBean>** with class and type attributes instantiates a bean from the class named in class and assigns the bean the data type you specify in type. The value of type can be the same as class, a superclass of class, or an interface implemented by class.
89. What is the significance of having a set of <jsp:setProperty> elements inside the body of a **<jsp:useBean>** ?
    a. The presence of a body a set of <jsp:setProperty> elements signifies the below:
        i. If the bean doesn’t exist, it will be created, and the <jsp:setProperty> tags will execute to set up some default values.
        ii. If bean exists already, it will be left alone, and the <jsp:setProperty> tags will not execute, so any property values already set will remain unchanged.
90. Can we have a **<jsp:useBean>** with scope = session and without any scope with same id in the same JSP file?
    a. No. Specifying the scope attribute creates or uses an attribute in the specified session and make it available as a local variable in the page scope with that name. So we cannot have two variables with same name in page scope and hence all ids for **<jsp:useBean>** should be unique.
91. **More notes**
    a. The xmlns attribute is required in XHTML but it is invalid in HTML.