JSP

**.What are the advantages of JSP over Servlet?**

JSP is a serverside technology to make content generation a simple appear.The advantage of JSP is that they are document-centric. Servlets, on the other hand, look and act like programs. A Java Server Page can contain [Java program](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs.php) fragments that instantiate and execute Java classes, but these occur inside an HTML template file and are primarily used to generate dynamic content. Some of the JSP functionality can be achieved on the client, using [JavaScript](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs.php). The power of JSP is that it is server-based and provides a framework for Web application development.

**2.What is the life-cycle of JSP?**

When a request is mapped to a JSP page for the first time, it translates the JSP page into a servlet class and compiles the class. It is this servlet that services the client requests.   
A JSP page has seven phases in its lifecycle, as listed below in the sequence of occurrence:

* Translation
* Compilation
* Loading the class
* Instantiating the class
* jspInit() invocation
* \_jspService() invocation
* jspDestroy() invocation   
  [More about JSP Life cycle](http://developersbook.com/jsp/jsp-lifecycle.php)

**3.What is the jspInit() method?**

The jspInit() method of the javax.servlet.jsp.JspPage interface is similar to the init() method of servlets. This method is invoked by the container only once when a JSP page is initialized. It can be overridden by a page author to initialize resources such as database and [network connections](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs.php), and to allow a JSP page to read persistent configuration data.

**4.What is the \_jspService() method?**

SThe \_jspService() method of the javax.servlet.jsp.HttpJspPage interface is invoked every time a new request comes to a JSP page. This method takes the HttpServletRequest and HttpServletResponse objects as its arguments. A page author cannot override this method, as its implementation is provided by the container.

**5.What is the jspDestroy() method?**

The jspDestroy() method of the javax.servlet.jsp.JspPage interface is invoked by the container when a JSP page is about to be destroyed. This method is similar to the destroy() method of servlets. It can be overridden by a page author to perform any cleanup operation such as closing a database connection.

**6.What JSP lifecycle methods can I override?**

You cannot override the \_jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

**7. How can I override the jspInit() and jspDestroy() methods within a JSP page?**

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

<%!

public void jspInit() {

. . .

}

%>

<%!

public void jspDestroy() {

. . .

}

%>

**8.What are implicit objects in JSP?**

Implicit objects in JSP are the Java objects that the JSP Container makes available to developers in each page. These objects need not be declared or instantiated by the JSP author. They are automatically instantiated by the container and are accessed using standard variables; hence, they are called implicit objects.The implicit objects available in JSP are as follows:

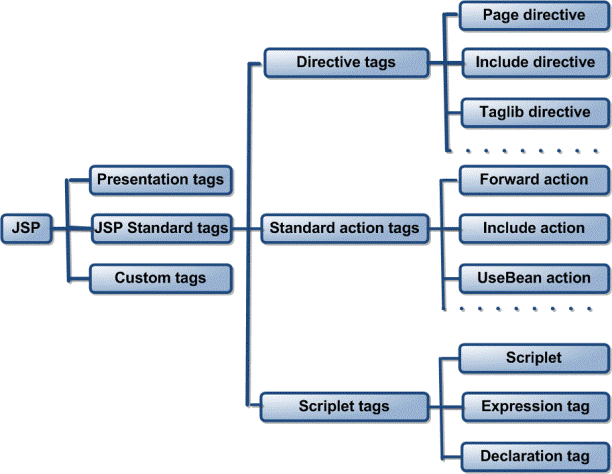
* request
* response
* pageContext
* session
* application
* out
* config
* page
* exception

The implicit objects are parsed by the container and inserted into the generated servlet code. They are available only within the jspService method and not in any declaration.

[Check more about implicit objects](http://developersbook.com/jsp/jsp-implicit-objects.php)

**9.What are the different types of JSP tags?**

The different types of JSP tags are as follows:

  
  
**10.What are JSP directives?**

* JSP directives are messages for the JSP engine. i.e., JSP directives serve as a message from a JSP page to the JSP container and control the processing of the entire page
* They are used to set global values such as a class declaration, method implementation, output content type, etc.
* They do not produce any output to the client.
* Directives are always enclosed within <%@ ….. %> tag.
* Ex: page directive, include directive, etc.

**.What is page directive?**

* A page directive is to inform the JSP engine about the headers or facilities that page should get from the environment.
* Typically, the page directive is found at the top of almost all of our JSP pages.
* There can be any number of page directives within a JSP page (although the attribute – value pair must be unique).
* The syntax of the include directive is: <%@ page attribute="value">
* Example:<%@ include file="header.jsp" %>

**12.What are the attributes of page directive?**

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

* **import**: It specifies the packages that are to be imported.
* **session**: It specifies whether a session data is available to the JSP page.
* **contentType**: It allows a user to set the content-type for a page.
* **isELIgnored**: It specifies whether the EL expressions are ignored when a JSP is translated to a servlet.
* language="**java**" ]   
            [ extends="*package*.*class*" ]   
            [ import="{*package*.*class* | *package*.*\**}, ..." ]   
            [ session="**true** | false" ]   
            [ buffer="none | **8kb** | *size*kb" ]   
            [ autoFlush="**true** | false" ]   
            [ isThreadSafe="**true** | false" ]   
            [ info="*text*" ]   
            [ errorPage="*relativeURL*" ]   
            [ contentType="*mimeType* [ ;charset=*characterSet* ]"   |   "**text/html ; charset=ISO-8859-1**" ]   
            [ isErrorPage="true | **false**" ]

**13.What is the include directive?**

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

* The include directive is used to statically insert the contents of a resource into the current JSP.
* This enables a user to reuse the code without duplicating it, and includes the contents of the specified file at the translation time.
* The syntax of the include directive is as follows:  
  <%@ include file = "FileName" %>
* This directive has only one attribute called file that specifies the name of the file to be included.

**14.What are the JSP standard actions?**

* The JSP standard actions affect the overall runtime behavior of a JSP page and also the response sent back to the client.
* They can be used to include a file at the request time, to find or instantiate a JavaBean, to forward a request to a new page, to generate a [browser](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs-2.php)-specific code, etc.
* Ex: include, forward, useBean,etc. object

**15.What are the standard actions available in JSP?**

The standard actions available in JSP are as follows:

* **<jsp:include>**: It includes a response from a servlet or a JSP page into the current page. It differs from an include directive in that it includes a resource at request processing time, whereas the include directive includes a resource at translation time.
* **<jsp:forward>**: It forwards a response from a servlet or a JSP page to another page.
* **<jsp:useBean>**: It makes a JavaBean available to a page and instantiates the bean.
* **<jsp:setProperty>**: It sets the properties for a JavaBean.
* **<jsp:getProperty>**: It gets the value of a property from a JavaBean component and adds it to the response.
* **<jsp:param>**: It is used in conjunction with <jsp:forward>;, <jsp:, or plugin>; to add a parameter to a request. These parameters are provided using the name-value pairs.
* **<jsp:plugin>**: It is used to include a Java applet or a JavaBean in the current JSP page.

**16.What is the <jsp:useBean> standard action?**

The <jsp:useBean> standard action is used to locate an existing JavaBean or to create a JavaBean if it does not exist. It has attributes to identify the object instance, to specify the lifetime of the bean, and to specify the fully qualified classpath and type.

**17.What are the scopes available in <jsp:useBean>?**

The scopes available in <jsp:useBean> are as follows:

* **page scope:**: It specifies that the object will be available for the entire JSP page but not outside the page.
* **request scope**: It specifies that the object will be associated with a particular request and exist as long as the request exists.
* **application scope**: It specifies that the object will be available throughout the entire [Web application](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs-2.php) but not outside the application.
* **session scope**: It specifies that the object will be available throughout the session with a particular client.

**18.What is the <jsp:forward> standard action?**

* The <jsp:forward> standard action forwards a response from a servlet or a JSP page to another page.
* The execution of the current page is stopped and control is transferred to the forwarded page.
* The syntax of the <jsp:forward> standard action is :    
  <jsp:forward page="/targetPage" />  
  Here, targetPage can be a JSP page, an HTML page, or a servlet within the same context.
* If anything is written to the output stream that is not buffered before <jsp:forward>, an IllegalStateException will be thrown.

*Note* : Whenever we intend to use <jsp:forward> or <jsp:include> in a page, buffering should be enabled. By default buffer is enabled.

**19.What is the <jsp:include> standard action?**

The <jsp:include> standard action enables the current JSP page to include a static or a dynamic resource at runtime. In contrast to the include directive, the include action is used for resources that change frequently. The resource to be included must be in the same context.The syntax of the <jsp:include> standard action is as follows:  
<jsp:include page="targetPage" flush="true"/>   
Here, targetPage is the page to be included in the current JSP.

**20.What is the difference between include directive and include action?**

|  |  |
| --- | --- |
| **Include directive** | **Include action** |
| The *include* directive, includes the content of the specified file during the translation phase–when the page is converted to a servlet. | The *include* action, includes the response generated by executing the specified page (a JSP page or a servlet) during the request processing phase–when the page is requested by a user. |
| The include directive is used to statically insert the contents of a resource into the current JSP. | The include standard action enables the current JSP page to include a static or a dynamic resource at runtime. |
| Use the include directive if the file changes rarely. It’s the fastest mechanism. | Use the include action only for content that changes often, and if which page to include cannot be decided until the main page is requested. |

**Differentiate between pageContext.include and jsp:include?**

The <jsp:include> standard action and the pageContext.include() method are both used to include resources at runtime. However, the pageContext.include() method always flushes the output of the current page before including the other components, whereas <jsp:include> flushes the output of the current page only if the value of flush is explicitly set to true as follows:

<jsp:include page="/index.jsp" flush="true"/>

**22.What is the jsp:setProperty action?**

You use jsp:setProperty to give values to properties of beans that have been referenced earlier. You can do this in two contexts. First, you can use jsp:setProperty after, but outside of, a jsp:useBean element, as below:

<jsp:useBean id="myName" ... />

...

<jsp:setProperty name="myName" property="myProperty" ... />

In this case, the jsp:setProperty is executed regardless of whether a new bean was instantiated or an existing bean was found.   
  
A second context in which jsp:setProperty can appear is inside the body of a jsp:useBean element, as below:

<jsp:useBean id="myName" ... >

...

<jsp:setProperty name="myName"

property="someProperty" ... />

</jsp:useBean>

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

#### **Attributes and Usage**

* name="*beanInstanceName*"

The name of an instance of a Bean that has already been created or located with a <jsp:useBean> element. The value of name must match the value of id in <jsp:useBean>. The <jsp:useBean> element must appear before <jsp:seProperty> in the JSP file.

* property="\*"

Stores all of the values the user enters in the viewable JSP page (called request parameters) in matching Bean properties. The names of the properties in the Bean must match the names of the request parameters, which are usually the elements of an HTML form. A Bean property is usually defined by a variable declaration with matching getter and setter methods (for more information, see the JavaBeans API Specification available at http://java.sun.com/beans).

The values of the request parameters sent from the client to the server are always of type String. The String values are converted to other data types so they can be stored in Bean properties. The allowed Bean property types and their conversion methods are shown in [TABLE 1-1](http://java.sun.com/products/jsp/tags/11/syntaxref11.fm13.html#17460).

**How <jsp:setProperty> Converts Strings to Other Values**

|  |  |
| --- | --- |
| **Property Type** | **String Is Converted Using** |
| boolean or Boolean | java.lang.Boolean.valueOf(String) |
| byte or Byte | java.lang.Byte.valueOf(String) |
| char or Character | java.lang.Character.valueOf(String) |
| double or Double | java.lang.Double.valueOf(String) |
| integer or Integer | java.lang.Integer.valueOf(String) |
| float or Float | java.lang.Float.valueOf(String) |
| long or Long | java.lang.Long.valueOf(String) |

|  |
| --- |
|  |

You can also use <jsp:setProperty> to set the value of an indexed property in a Bean. The indexed property must be an array of one of the data types shown in [TABLE 1-1](http://java.sun.com/products/jsp/tags/11/syntaxref11.fm13.html#17460). The array elements are converted using the conversion methods shown in the table.

If a request parameter has an empty or null value, the corresponding Bean property is not set. Likewise, if the Bean has a property that does not have a matching request parameter, the property value is not set.

* property="*propertyName*" [ param="*parameterName*" ]

Sets one Bean property to the value of one request parameter. In the syntax, property specifies the name of the Bean property and param specifies the name of the request parameter by which data is being sent from the client to the server.

If the Bean property and the request parameter have different names, you must specify both property and param. If they have the same name, you can specify property and omit param.

If a parameter has an empty or null value, the corresponding Bean property is not set.

* property="*propertyName*" value="{*string* | <%= *expression* %>}"

Sets one Bean property to a specific value. The value can be a String or an expression that is evaluated at runtime. If the value is a String, it is converted to the Bean property's data type according to the conversion rules shown above in [TABLE 1-1](http://java.sun.com/products/jsp/tags/11/syntaxref11.fm13.html#17460). If it is an expression, its value must have a data type that matches the the data type of the value of the expression must match the data type of the Bean property.

If the parameter has an empty or null value, the corresponding Bean property is not set. You cannot use both the param and value attributes in a <jsp:setProperty> element.

**23.What is the action?**

The <jsp:getProperty> action is used to access the properties of a bean that was set using the <jsp:getProperty> action. The container converts the property to a String as follows:

* If it is an object, it uses the toString() method to convert it to a String.
* If it is a primitive, it converts it directly to a String using the valueOf() method of the corresponding Wrapper class.
* The syntax of the <jsp:getProperty> method is: <jsp:getProperty name="Name" property="Property" />

Here, name is the id of the bean from which the property was set. The property attribute is the property to get. A user must create or locate a bean using the <jsp:useBean> action before using the <jsp:getProperty> action.

**24.What is the <jsp:param> standard action?**

The <jsp:param> standard action is used with <jsp:include> or <jsp:forward> to pass parameter names and values to the target resource. The syntax of the <jsp:param> standard action is as follows:   
<jsp:param name="paramName" value="paramValue"/>

**25.What is the jsp:plugin action ?**

This action lets you insert the [browser](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs-3.php)-specific OBJECT or EMBED element needed to specify that the browser run an applet using the [Java](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs-3.php) plugin.

**26.What are scripting elements?**

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

1. **Expressions** of the form <%= expression %> that are evaluated and inserted into the output,
2. **Scriptlets** of the form <% code %> that are inserted into the servlet's service method,
3. **Declarations** of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods.

**27.What is a scriptlet?**

A scriptlet contains Java code that is executed every time a JSP is invoked. When a JSP is translated to a servlet, the scriptlet code goes into the service() method. Hence, methods and variables written in scriptlets are local to the service() method. A scriptlet is written between the **<% and %>** tags and is executed by the container at request processing time.

**28.What are JSP declarations?**

As the name implies, JSP declarations are used to declare class variables and methods in a JSP page. They are initialized when the class is initialized. Anything defined in a declaration is available for the whole JSP page. A declaration block is enclosed between the **<%! and %>** tags. A declaration is not included in the service() method when a JSP is translated to a servlet.

**29.What is a JSP expression?**

A JSP expression is used to write an output without using the out.[print](http://www.developersbook.com/jsp/interview-questions/jsp-interview-questions-faqs-3.php) statement. It can be said as a shorthand representation for scriptlets. An expression is written between the **<%= and %>** tags. It is not required to end the expression with a semicolon, as it implicitly adds a semicolon to all the expressions within the expression tags.

**30.How is scripting disabled?**

Scripting is disabled by setting the scripting-invalid element of the deployment descriptor to true. It is a subelement of jsp-property-group. Its valid values are true and false. The syntax for disabling scripting is as follows:

<jsp-property-group>

<url-pattern>\*.jsp</url-pattern>

<scripting-invalid>true</scripting-invalid>

</jsp-property-group>

### **How do i include static files within a jsp page ?**

This is static include..   
you can just include using below code   
<%@ include file="header.jsp" %>   
or   
<%@ include file="footer.html" %>   
  
Dynamic include you can do like below   
<jsp:include page="header.jsp"/>   
  
or you can pass parameter also   
<jsp:include page="header.jsp">   
<jsp:param name="title" value="Java Interview Questions"/>   
</jsp:include>   
  
  
Image upload from JSP to DataBase ?

Image upload from JSP to Data Base step by step :   
  
step 1 :   
In the JSP   
<form name="regform2" method="post" enctype="multipart/form-data">   
<input type="file" name="ImageFile" id="ImageFile" onChange="uploadImage()"/>   
</form>   
  
java script :   
function uploadImage(){   
document.regform.action ="<%=request.getContextPath()%>/dfdmin?cmd=uploadimage";   
document.regform.submit();   
}   
  
  
Step 2.   
In the servlet - add the below code.   
String rtempfile = File.createTempFile("temp","1").getParent();   
MultipartRequest multi = new MultipartRequest(request, rtempfile, 500000 \* 1024);   
File rnewfile=null;   
rnewfile = new File(CommonArt.IMAGE\_PATH+"jsp"+File.separator+"images"+File.separator+"uploadImage"+File.separator);   
if(rnewfile.exists()){   
}else{   
rnewfile.mkdirs();   
}   
  
File f = multi.getFile("ImageFile");   
System.out.println(f.getName());   
FileInputStream fin =new FileInputStream(f);   
RandomAccessFile r = new RandomAccessFile(rnewfile+File.separator+f.getName(),"rw");   
filename = f.getName();   
// FileOutputStream fos =new FileOutputStream(rnewfile);   
byte sizefile[] = new byte[5000000];   
fin.read(sizefile);   
// fos.write(sizefile);   
r.write(sizefile);   
//fos.close();   
r.close();   
fin.close();   
  
  
Step 3.   
Insert into Database   
  
InputStream is = new FileInputStream(f);   
String sql = " INSERT INTO image\_upload (IMAGE) VALUES (?) ";   
pStmt = conn.prepareStatement(sql);   
pStmt.setBinaryStream(1, is, (int)(f.length()));   
pStmt.execute();   
conn.commit();   
  
  
  
This will upload multipart file to your data base.   
  
Note : get cos.jar from oreilly website

How to Disable session in JSP page ?

Disabling the session in some pages will improve the performance of your JSP container.   
  
Every time a JSP is requested, JSP creates an HttpSession object to maintain state for each unique client. The session data is accessible in the JSP as the implicit session object. In JSPs, sessions are enabled by default.   
By default <%@ page session="true" %>   
  
Session object uses the server resources. Each session object uses up a small amount of system resources as it is stored on the server side. This also increases the traffic as the session ID is sent from server to client. Client also sends the same session ID along with each request. If some of the JSP pages on your web site are getting millions of hits from internet browser and there is not need to identify the user, so its better to disable the session in that JSP page.   
  
  
You can tell the container to disable session in the JSP file by setting the session attribute to false. Set the session attribute of the page directive to false.   
  
<%@ page session="false" %>   
  
<%@ page language="java" session="false"%>   
<html>   
<head>   
<title>Session Disabled Example</title>   
</head>   
<body>   
<p>Session is Disabled in this page </p>   
</body>   
</html>

Q.How you do Session Management in JSP ?

### **Http protocol is a stateless protocol, that means that it can't persist the data. Http treats each request as a new request so every time you will send a request you will be considered as a new user. In session management whenever a request comes for any resource, a unique token is generated by the server and transmitted to the client by the response object and stored on the client machine as a cookie. We can also say that the process of managing the state of a web based client is through the use of session IDs. Session IDs are used to uniquely identify a client browser, while the server side processes are used to associate the session ID with a level of access. Thus, once a client has successfully authenticated to the web applicatiion, the session ID can be used as a stored authentication voucher so that the client does not have to retype their login information with each page request. Now whenever a request goes from this client again the ID or token will also be passed through the request object so that the server can understand from where the request is coming. Session management can be achieved by : 1. Cookies: cookies are small bits of textual information that a web server sends to a browser and that browsers returns the cookie when it visits the same site again. In cookie the information is stored in the form of a name, value pair. By default the cookie is generated. If the user doesn't want to use cookies then it can disable them browser setting. 2. URL rewriting: In URL rewriting we append some extra information on the end of each URL that identifies the session. This URL rewriting can be used where a cookie is disabled. It is a good practice to use URL rewriting. In this session ID information is embedded in the URL, which is recieved by the application through Http GET requests when the client clicks on the links embedded with a page. Exmaple : http://www.techfaq360.com/answers.jsp?sname=test 3. Hidden form fields: In hidden form fields the html entry will be like this : <input type ="hidden" name ="name" value="">. This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and submitted with the Http POST method. 4. HttpSession object : javax.servlet.http.HttpSession is an interface that provides a way to identify a user across more than one page request or visit to a web site. This is the way mainly used in webapplication. HttpSession object maintain session for you. You don't need to do any session management. session.setAttribute("name",name); String name = session.getAttribute("name"); you will get the same value which you have set. How can get the checked values from multiple check box checked ? or How to Handle multiple check box checked ?**

In the JSP page   
  
for(int j = 0; j < messageList.size(); j++) {   
  
Message msg = (Message)messageList.get(j);   
  
<input type="checkbox" name="delmesg\_<%=msg.getMessageId()%>" value="on">   
  
}   
  
  
  
Retrive the check box which is checked if more then one   
  
In the Servlet   
  
Enumeration params = request.getParameterNames();   
  
List msgIdList = new ArrayList();   
  
while ( params != null && params.hasMoreElements()){   
  
String param = (String)params.nextElement();   
  
//System.out.println("param--------"+param);   
  
if(params != null && param.startsWith("delmesg\_")){   
  
  
  
String msgId = param.substring(param.indexOf("\_") + 1 ,param.length());   
  
System.out.println("msgId"+msgId);   
  
msgIdList.add(msgId);   
  
  
  
}   
  
}   
  
  
  
// msgIdList contains number of check box checled in the JSP

Q.How to send email through JSP/Java?

In the JSP   
<%   
String from,   
  
String[] to,   
  
String subject,   
  
String textBody   
  
) {   
  
boolean sentResult = false;   
  
  
  
try {   
  
Properties mailProps = new Properties();   
  
mailProps.put("mail.smtp.host", Common.SMTP\_SERVER);   
  
Session session = Session.getDefaultInstance(mailProps, null);   
  
  
  
//create message   
  
MimeMessage message = new MimeMessage(session);   
  
  
  
//set from   
  
InternetAddress fromAdd = new InternetAddress(from);   
  
message.setFrom(fromAdd);   
  
  
  
//set to   
  
InternetAddress[] toAdd = null;   
  
  
  
if(to != null){   
  
toAdd = new InternetAddress[to.length];   
  
for(int i = 0 ; i < to.length; i++)   
  
toAdd[i] = new InternetAddress(to[i]);   
  
  
  
message.setRecipients(Message.RecipientType.TO,toAdd);   
  
}   
  
  
  
  
  
  
  
  
  
// set Subject   
  
message.setSubject(subject);   
  
  
  
  
  
Multipart mp = new MimeMultipart("alternative");   
  
if(textBody != null){   
  
MimeBodyPart tbp = new MimeBodyPart();   
  
tbp.setText(textBody, "ISO-8859-1");   
  
tbp.setHeader("Content-Type", "text/plain; charset=" + "ISO-8859-1");   
  
mp.addBodyPart(tbp);   
  
}   
  
  
  
message.setContent(mp);   
  
  
  
//set sent Date   
  
message.setSentDate(new java.util.Date());   
  
  
  
// send the mail off   
  
  
  
Transport.send(message);   
  
sentResult = true;   
  
} catch(Exception exp) {   
  
exp.printStackTrace();   
  
sentResult = false;   
  
  
  
}   
  
%>

Q.How to create and retrive a multiple selections list in jsp/servlet?

This is the code to display multiple selections list in jsp.   
List medList = DAO.getallMedium(); // this method retun list of Medium objects.   
medList list contains list of Medium objects.   
  
Java :   
bean class.   
public class Medium {   
    int medId;   
    String medName;   
  
    /\*\*   
     \* @return Returns the medId.   
     \*/   
    public int getMedId() {   
        return medId;   
    }   
    /\*\*   
     \* @param medId The medId to set.   
     \*/   
    public void setMedId(int medId) {   
        this.medId = medId;   
    }   
    /\*\*   
     \* @return Returns the medName.   
     \*/   
    public String getMedName() {   
        return medName;   
    }   
    /\*\*   
     \* @param medName The medName to set.   
     \*/   
    public void setMedName(String medName) {   
        this.medName = medName;   
    }   
}   
  
DAO Clas to retrive mediums in data base.   
public static List getMediums(){   
        PreparedStatement pStmt = null;   
     Connection conn = null;   
     boolean success = false;   
     ResultSet rs = null;   
     List medList = new ArrayList();   
       
     try{   
         conn = getConnection();   
            
         String sql = " select \* from MEDIUM ";   
         pStmt = conn.prepareStatement(sql);   
                     
         rs = pStmt.executeQuery();   
         while(rs.next()){   
             Medium med = new Medium();   
             med.setMedId(rs.getInt("MED\_ID"));   
             med.setMedName(rs.getString("MEDIUM\_NAME"));   
             medList.add(med);   
             }   
           
           
           
     }catch(Exception e){   
         e.printStackTrace();   
            
     }finally{   
         closeConnectionProp(conn,pStmt,rs);   
     }   
           
     return medList;   
       
    }   
  
JSP :   
  
<select name="Medium" id="medium" multiple=true>   
<option value="0">Choose A Medium</option>   
  
<%   
     for(int i=0; i<medList.size();i++){   
         Medium med = (Medium)medList.get(i);   
     %>   
<option value="<%=med.getMedId()%>"><%=med.getMedName()%></option>   
<%}%>   
</select>   
  
  
Servlet :   
Retrive the multiple selections from jsp   
String[] mediums = request.getParameterValues("medium");   
  
for(int i=0;i<mediums.length;i++)   
{   
System.out.println(mediums[i]);   
}

Q.How to create a drop down list in jsp?

This is the code to display drop down list in jsp.   
List medList = DAO.getallMedium(); // this method retun list of Medium objects.   
medList list contains list of Medium objects.   
  
Java :   
bean class.   
public class Medium {   
    int medId;   
    String medName;   
  
    /\*\*   
     \* @return Returns the medId.   
     \*/   
    public int getMedId() {   
        return medId;   
    }   
    /\*\*   
     \* @param medId The medId to set.   
     \*/   
    public void setMedId(int medId) {   
        this.medId = medId;   
    }   
    /\*\*   
     \* @return Returns the medName.   
     \*/   
    public String getMedName() {   
        return medName;   
    }   
    /\*\*   
     \* @param medName The medName to set.   
     \*/   
    public void setMedName(String medName) {   
        this.medName = medName;   
    }   
}   
  
DAO Clas to retrive mediums in data base.   
public static List getMediums(){   
        PreparedStatement pStmt = null;   
     Connection conn = null;   
     boolean success = false;   
     ResultSet rs = null;   
     List medList = new ArrayList();   
       
     try{   
         conn = getConnection();   
            
         String sql = " select \* from MEDIUM ";   
         pStmt = conn.prepareStatement(sql);   
                     
         rs = pStmt.executeQuery();   
         while(rs.next()){   
             Medium med = new Medium();   
             med.setMedId(rs.getInt("MED\_ID"));   
             med.setMedName(rs.getString("MEDIUM\_NAME"));   
             medList.add(med);   
             }   
           
           
           
     }catch(Exception e){   
         e.printStackTrace();   
            
     }finally{   
         closeConnectionProp(conn,pStmt,rs);   
     }   
           
     return medList;   
       
    }   
  
JSP :   
  
<select name="Medium" id="medium" >   
<option value="0">Choose A Medium</option>   
  
<%   
     for(int i=0; i<medList.size();i++){   
         Medium med = (Medium)medList.get(i);   
     %>   
<option value="<%=med.getMedId()%>"><%=med.getMedName()%></option>   
<%}%>   
</select>

Q. How to add and delete Cookie in jsp ?

Add Cookie to response object:   
Cookie cookie = new Cookie ("name",value);   
cookie.setPath("/");   
cookie.setDomain(DOMAIN\_NAME); DOMAIN\_NAME may be .techfaq360.com   
cookie.setMaxAge(2\* 7 \* 24 \* 60 \* 60);// 2 week   
response.addCookie(cookie);   
  
Get cookie from request object :   
  
Cookie myCookie = null;   
  
Cookie cookies [] = request.getCookies ();   
if (cookies != null)   
for (int i = 0; i < cookies.length; i++)   
{   
if (cookies [i].getName().equals ("name")) // the name of the cookie you have added   
{   
myCookie = cookies[i];   
break;   
}   
}   
  
Delete Cookie:   
  
You can't delete the cookie. just add maxage to 0;   
cookie.setMaxAge(0);   
cookie.setPath("/");   
cookie.setDomain(DOMAIN\_NAME);   
response.addCookie(cookie);   
  
cookie.setMaxAge(-1) means on browser close cookie will be deleted.

Q. How to Protect JSPs from direct access ?

JSPs located in the WEB-INF and its sub-directories are protected from outside access.   
If you want to go pageB.jsp from pageA.jsp   
<html:link action="gotoPageB">Go to Page B</html:link>   
  
in the struts-config.xml   
<action path="/gotoPageB"   
parameter="/WEB-INF/pageB.jsp"   
type="org.apache.struts.actions.ForwardAction"/>

Q.How to upload an image from servlet/jsp into server from clients machine?

Image upload from JSP to Data Base step by step :   
  
step 1 :   
In the JSP   
<form name="regform2" method="post" enctype="multipart/form-data">   
<input type="file" name="ImageFile" id="ImageFile" onChange="uploadImage()"/>   
</form>   
  
java script :   
function uploadImage(){   
document.regform.action ="<%=request.getContextPath()%>/dfdmin?cmd=uploadimage";   
document.regform.submit();   
}   
  
  
Step 2.   
In the servlet - add the below code.   
This will upload your image to server (D:\\) or In unix you can mention /home/user like that   
  
String rtempfile = File.createTempFile("temp","1").getParent();   
MultipartRequest multi = new MultipartRequest(request, rtempfile, 500000 \* 1024);   
File rnewfile=null;   
rnewfile = new File("D:\\"+"jsp"+File.separator+"images"+File.separator+"uploadImage"+File.separator);   
if(rnewfile.exists()){   
}else{   
rnewfile.mkdirs();   
}   
  
File f = multi.getFile("ImageFile");   
System.out.println(f.getName());   
FileInputStream fin =new FileInputStream(f);   
RandomAccessFile r = new RandomAccessFile(rnewfile+File.separator+f.getName(),"rw");   
filename = f.getName();   
// FileOutputStream fos =new FileOutputStream(rnewfile);   
byte sizefile[] = new byte[5000000];   
fin.read(sizefile);   
// fos.write(sizefile);   
r.write(sizefile);   
//fos.close();   
r.close();   
fin.close();   
  
DAO.upload(f); // Call to DAO for insert image into database.   
  
Step 3.   
  
Then call to DAO to save this image to data base.   
Insert into Database   
public void upload(File f){   
InputStream is = new FileInputStream(f);   
String sql = " INSERT INTO image\_upload (IMAGE) VALUES (?) ";   
pStmt = conn.prepareStatement(sql);   
pStmt.setBinaryStream(1, is, (int)(f.length()));   
pStmt.execute();   
conn.commit();   
}   
  
  
  
This will upload multipart file to your data base.   
  
Note : get cos.jar from oreilly website   
  
  
OR   
if your server have already image file then just do below step   
  
  
  
  
Code is here :   
File f = new File("d:\\test.jpg");   
InputStream is = new FileInputStream(f);   
String sql = " INSERT INTO image\_upload (IMAGE) VALUES (?) ";   
pStmt = conn.prepareStatement(sql);   
pStmt.setBinaryStream(1, is, (int)(f.length()));   
pStmt.execute();   
conn.commit();   
  
This will work.

Q.How do you delete a Cookie within a JSP?

Cookie mycook = new Cookie("name","value");   
response.addCookie(mycook);   
Cookie killmycook = new Cookie("mycook","value");   
killmycook.setMaxAge(0);   
killmycook.setPath("/");   
killmycook.addCookie(killmycook);

Q.How do you prevent the Creation of a Session in a JSP Page and why?

<%@ page session="false">   
By default, a JSP page will automatically create a session for the request if one does not exist.   
However, sessions consume resources and if it is not necessary to maintain a session, one should not be created. For example, a marketing campaign may suggest the reader visit a web page for more information. If it is anticipated that a lot of traffic will hit that page, you may want to optimize the load on the machine by not creating useless sessions.

Q.How does a servlet communicate with a JSP page?

In the service method   
protected void service(HttpServletRequest request,HttpServletResponse response)   
        throws ServletException, java.io.IOException {   
  
User user = (User)request.getSession().getAttribute("user");   
                   
        String imgId = (String)request.getParameter("imgId");   
        String path = request.getContextPath()+"/jsp/addnetwork.jsp";   
           
        //do some thing   
User user = DAO.getUser(imgId);   
request.getSession().setAttribute("user",user);       
         response.sendRedirect(path);   
  
  
}   
  
In the addnetwork.jsp   
  
User user = getAttribute("user);

Q.which situation you use static include and dynamic include   
in jsp?

static include : Contents are static like   
<%@ include file="header.jsp"%> ..Not changing frequently. Don't need to pass parameter.   
For example.   
a.jsp contain <%@ include file="header.jsp"%> then all the codes of header.jsp is included in a.jsp.   
  
dynamic include : Need to pass parameter. Need to execute every time. Dynamic content. Excecute the included jsp and only ouput is pasted to main jsp   
For example..   
a.jsp contain following code   
<jsp:include page="masterTemplate.jsp" flush="true">   
<jsp:param name="rColumn" value="rightDisplay.jsp" />   
<jsp:param name="mColumn" value="AlertBody.jsp" />   
<jsp:param name="title" value="Alerts" />   
</jsp:include>   
  
masterTemplate.jsp execute and output is pasted to a.jsp.

q.What's the Difference between Forward and Include?

The <jsp:forward> action enables you to forward the request to a static HTML file, a servlet, or another JSP.   
  
<jsp:forward page="url" />   
  
The JSP that contains the <jsp:forward> action stops processing, clears its buffer, and forwards the request to the target resource. Note that the calling JSP should not write anything to the response prior to the <jsp:forward> action.   
  
You can also pass additional parameters to the target resource using the <jsp:param> tag.   
  
<jsp:forward page="test.htm" >   
<jsp:param name="name1" value="value1" />   
<jsp:param name="name2" value="value2" />   
</jsp:forward>   
  
In this example, test.jsp can access the value of name1 using request.getParameter("name1").   
  
<jsp:include execute the code and force a flush of the buffer in the output stream.   
If a.jsp has the code like   
<jsp:include page="template.jsp" flush="true" >   
<jsp:param name="name1" value="value1" />   
</jsp:include>   
  
then template.jsp execute and the output is placed in a.j

Q.How do you pass control from one JSP page to another?

Use the following ways to pass control of a request from one servlet to another or one jsp to another.   
The RequestDispatcher object ?s forward method to pass the control.   
Ex. <jsp:forward page="b.jsp" />   
The response.sendRedirect method   
Ex. response.sendRedirect("b.jsp");

Q.How can I declare methods within my JSP page?

In the declarion part   
<%!   
public int add(inti,intj){   
return i+j;   
}   
%>

Q.How can I implement a thread-safe JSP page?

You can make your JSPs thread-safe by having them implement the SingleThreadModel interface. This is done by adding the directive <%@ page isThreadSafe="false" % > within your JSP page.

Q.How does JSP handle run-time exceptions?

You can use the errorPage attribute of the page directive to have uncaught runtime exceptions automatically forwarded to an error processing page.   
For example:   
redirects the browser to the JSP page error.jsp if an uncaught exception is encountered during request processing. Within error.jsp, if you indicate that it is an error-processing page, via the directive: isErrorPage=true.   
the Throwable object describing the exception may be accessed within the error page via the exception implicit object.

Q.How do I perform browser redirection from a JSP page?

You can use the response implicit object to redirect the browser to a different resource, as:   
response.sendRedirect("http://www.exforsys.com/path/error.html");   
You can also physically alter the Location HTTP header attribute, as shown below:   
You can also use the:   
Also note that you can only use this before any output has been sent to the client. I beleve this is the case with the response.sendRedirect() method as well. If you want to pass any paramateres then you can pass using >

Q.How do I include static files within a JSP page?

<%@ include file="header.jsp"%>   
Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase

Q.How many JSP scripting elements and what are they?

There are three scripting language elements:   
--declarations   
<%!   
int i=8;   
%>   
--scriptlets   
<% Java Code Logic   
i=i+29;   
%>   
  
--expressions   
<%= i %> - This will display the value of i.

Q.What are the implicit objects?

Implicit objects are objects that are created by the web container and contain information related to a particular request, page, or application. They are:   
--request   
--response   
--pageContext   
--session   
--application   
--out   
--config   
--page   
--exception

### **Q.How does a try statement determine which catch clause should be used to handle an exception?**

When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exceptionis executed. The remaining catch clauses are ignored.

Q.How do I prevent the browser from caching my dynamic content?

add the code to your jsp.   
<%   
response.setHeader( "Cache-Control", "no-cache" );   
response.setHeader( "Pragma", "no-cache" );   
response.setIntHeader( "Expires", 0 );   
%>

What is a Expression,Declaration,Scriptlet in jsp?

<b>expression </b>   
An expression tag contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file. Because the value of an expression is converted to a String, you can use an expression within text in a JSP file. Like   
<%= emp.getName()%>   
<%= (new java.util.Date()).toLocaleString() %>   
You cannot use a semicolon to end an expression.   
  
All the expression code is converting to servlet. All the expressions go to inside service() method of the convert servlet.   
  
  
<b>Declaration</b>   
A declaration declares one or more variables or methods for use later in the JSP source file.   
A declaration must contain at least one complete declarative statement. You can declare any number of variables or methods within one declaration tag, as long as they are separated by semicolons. The declaration must be valid in the scripting language used in the JSP file.   
  
<%! int i = 0; %>   
<%! int a, b, c; %   
  
You can add method to declaration part.   
<%!   
public String trimData(String str){   
return str.trim();   
}   
%>   
  
You can call the method within the jsp.   
  
All the declaration code is converting to servlet. If you add method , the method is in convert servlet. All varibales are instance variable in the convert servlet.   
  
<b>Scriptlet</b>   
Scriptlet code is like java logic. you can declare varibales in the scriptlet and do the logic.   
All the Scriptlet go to inside service() method of the convert servlet.   
  
<%   
int n=10;   
for(int i=0;i<n;i++){   
}   
%>   
What is a Visible and Hidden Comment in jsp?

On view source (browser) the Hidden Comment is not visible to end user.   
  
On view source (browser) the Visible Comment is visible to end user.   
  
Examples   
<%-- This comment will not be visible in the page source --%>   
  
<!-- This comment will visible in the page source -->

### **Q.What are the different scope valiues for the <jsp:useBean>?**

Scopes are   
page - with in the same page   
request - after forward or include also you will get the request scope data.   
session -   
after senRedirect also you will get the session scope data. All data stored in session is available to end user till session closed or browser closed.   
  
application -   
Data will be available through out the application. One user can store data in application scope and other can get the data from application scope.

Q.Explain the life-cycle mehtods in JSP?

The jspInit()- The container calls the jspInit() to initialize te servlet instance.It is called before any other method, and is called only once for a servlet instance.   
The \_jspservice()- The container calls the \_jspservice() for each request, passing it the request and the response objects.   
The jspDestroy()- The container calls this when it decides take the instance out of service. It is the last method called n the servlet instance.

## [***24. How can JSP inheritance be achieved***](http://be achieved)

Posted on September 6, 2006 by sharat

The JSP super class and sub-class need to satisfy certain requirements as defined in the sections below. First let us assume we are using the HTTP protocol. Then mapping of other protocols is an easy task.

**Requirements for the Super Class**

* It must implement the HttpJspPage interface.
* It will obviously extend the HttpServlet class. It needs to implement the three main lifecycle methods of the servlet:
  + init
  + destroy
  + service

These implementations are ‘final’. In addition, the method getServletConfig must be implemented to return the ServletConfig object, passed onto the servlet during initialization.

This super class now necessitates the base class to actually implement the JSP version of these three function calls: jspInit, jspDestroy, and \_jspService, by providing abstract declarations for the same

* + public abstract void jspInit()
  + public abstract void jspDestroy()
  + The signature for the \_jspService method depends on the kind of JSP Engine that has been deployed. For eg: if Weblogic is the JSP Engine, the signature is:

public abstract void \_jspService(HttpServletRequest request, HttpServletResponse response);

The init method is called jspInit, the destroy method – jspDestroy, and the service method as \_jspService.

**Requirements for the sub class**

The sub class may or may not provide the jspInit and jspDestroy method, based on whether the parent class provides it.

**Code snippet for the JSP sub class**

<%@ page extends=”JSPPageSuper”%>

<%

//Any server processing

%>

<HTML>

</HTML>

Note the first line where the page extends the JSPPageSuper servlet.

JSTL

There are five groups under which the JSTL tags have been organized.

They are as follows:

1)  **core**

**2) xml**

**3) sql**

**4) formatting**

**5) functions.**

The JSTL folder contains a sub-folder named 'tld'. There will be a number of tld files there such as

**c.tld, ( core)**

**x.tld, (xml)**

**fmt.tld, (format)**

**sql.tld & (sql)**

**fn.tld. (functions)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Description** |  |  |
| catch | Catches any Throwable exception that occurs in the body of the tag |  |  |
| choose | Provides conditional operation and allows for choosing between mutually exclusive options |  |  |
| if | Provides conditional operation |  |  |
| import | Imports a resource specified by a URL and exposes it to the page, variable, or reader |  |  |
| forEach | Iterates over collections |  |  |
| forTokens | Iterates over tokens |  |  |
| out | Outputs expressions to the Web page |  |  |
| otherwise | A conditional operation tag for choosing between mutually exclusive options. This tag is a subtag of the choose tag. |  |  |
| param | Used along with the import, redirect, or url tags to add a parameter to a URL |  |  |
| redirect | Redirects to a new URL |  |  |
| remove | Removes a variable from a scope |  |  |
| set | Sets the value of an attribute |  |  |
| url | Creates a URL |  |  |
| when | A conditional operation tag that includes its body when an expression evaluates to true. This tag is a subtag of the choose tag. |  |  |
| *Table 14-3 Core Tags* |  |  |  |
| **Area** | **Function** | **Tags** | **Prefix** |
| **Core** | Variable support | remove  set | c |
|  | Flow control | choose    when    otherwise  forEach  forTokens  if |  |
|  | URL management | import    param  redirect    param  url    param |  |
|  | Miscellaneous | catch  out |  |

### Variable Support Tags

The set tag sets the value of an EL variable or the property of an EL variable in any of the JSP scopes (page, request, session, or application). If the variable does not already exist, it is created.

The JSP EL variable or property can be set either from the attribute value:

<c:set var="foo" scope="session" value="..."/>

or from the body of the tag:

<c:set var="foo">

  ...

</c:set>

For example, the following sets an EL variable named bookID with the value of the request parameter named Remove:

<c:set var="bookId" value="${param.Remove}"/>

To remove an EL variable, you use the remove tag. When the bookstore JSP page [bookreceipt.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookreceipt.txt) is invoked, the shopping session is finished, so the cart session attribute is removed as follows:

<c:remove var="cart" scope="session"/>

### Flow Control Tags

To execute flow control logic, a page author must generally resort to using scriptlets. For example, the following scriptlet is used to iterate through a shopping cart:

<%

  Iterator i = cart.getItems().iterator();

  while (i.hasNext()) {

    ShoppingCartItem item =

      (ShoppingCartItem)i.next();

    ...

%>

    <tr>

    <td align="right" bgcolor="#ffffff">

    ${item.quantity}

    </td>

    ...

<%

  }

%>

Flow control tags eliminate the need for scriptlets. The next two sections have examples that demonstrate the conditional and iterator tags.

#### **Conditional Tags**

The if tag allows the conditional execution of its body according to the value of the test attribute. The following example from [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookcatalog.txt) tests whether the request parameter Add is empty. If the test evaluates to true, the page queries the database for the book record identified by the request parameter and adds the book to the shopping cart:

<c:if test="${!empty param.Add}">

  <c:set var="bid" value="${param.Add}"/>

  <jsp:useBean id="bid" type="java.lang.String" />

   <sql:query var="books"

    dataSource="${applicationScope.bookDS}">

    select \* from PUBLIC.books where id = ?

    <sql:param value="${bid}" />

  </sql:query>

  <c:forEach var="bookRow" begin="0" items="${books.rows}">

    <jsp:useBean id="bookRow" type="java.util.Map" />

    <jsp:useBean id="addedBook"

      class="database.BookDetails" scope="page" />

  ...

  <% cart.add(bid, addedBook); %>

...

</c:if>

The choose tag performs conditional block execution by the embedded when subtags. It renders the body of the first when tag whose test condition evaluates to true. If none of the test conditions of nested when tags evaluates to true, then the body of an otherwise tag is evaluated, if present.

For example, the following sample code shows how to render text based on a customer's membership category.

<c:choose>

  <c:when test="${customer.category == 'trial'}" >

    ...

  </c:when>

  <c:when test="${customer.category == 'member'}" >

    ...

  </c:when>

    <c:when test="${customer.category == 'preferred'}" >

    ...

  </c:when>

  <c:otherwise>

    ...

  </c:otherwise>

</c:choose>

The choose, when, and otherwise tags can be used to construct an if-then-else statement as follows:

<c:choose>

  <c:when test="${count == 0}" >

    No records matched your selection.

  </c:when>

  <c:otherwise>

    ${count} records matched your selection.

  </c:otherwise>

</c:choose>

#### **Iterator Tags**

The forEach tag allows you to iterate over a collection of objects. You specify the collection via the items attribute, and the current item is available through a variable named by the var attribute.

A large number of collection types are supported by forEach, including all implementations of java.util.Collection and java.util.Map. If the items attribute is of type java.util.Map, then the current item will be of type java.util.Map.Entry, which has the following properties:

* key: The key under which the item is stored in the underlying Map
* value: The value that corresponds to the key

Arrays of objects as well as arrays of primitive types (for example, int) are also supported. For arrays of primitive types, the current item for the iteration is automatically wrapped with its standard wrapper class (for example, Integer for int, Float for float, and so on).

Implementations of java.util.Iterator and java.util.Enumeration are supported, but they must be used with caution. Iterator and Enumeration objects are not resettable, so they should not be used within more than one iteration tag. Finally, java.lang.String objects can be iterated over if the string contains a list of comma-separated values (for example: Monday,Tuesday,Wednesday,Thursday,Friday).

Here's the shopping cart iteration from the preceding section, now with the forEach tag:

<c:forEach var="item" items="${sessionScope.cart.items}">

  ...

  <tr>

    <td align="right" bgcolor="#ffffff">

    ${item.quantity}

  </td>

  ...

</c:forEach>

The forTokens tag is used to iterate over a collection of tokens separated by a delimiter.

### URL Tags

The jsp:include element provides for the inclusion of static and dynamic resources in the same context as the current page. However, jsp:include cannot access resources that reside outside the web application, and it causes unnecessary buffering when the resource included is used by another element.

In the following example, the transform element uses the content of the included resource as the input of its transformation. The jsp:include element reads the content of the response and writes it to the body content of the enclosing transform element, which then rereads exactly the same content. It would be more efficient if the transform element could access the input source directly and thereby avoid the buffering involved in the body content of the transform tag.

<acme:transform>

  <jsp:include page="/exec/employeesList"/>

<acme:transform/>

The import tag is therefore the simple, generic way to access URL-based resources, whose content can then be included and or processed within the JSP page. For example, in [XML Tag Library](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL5.html#wp63716), import is used to read in the XML document containing book information and assign the content to the scoped variable xml:

<c:import url="/books.xml" var="xml" />

<x:parse doc="${xml}" var="booklist"

  scope="application" />

The param tag, analogous to the jsp:param tag (see [jsp:param Element](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSPIntro11.html#wp66264)), can be used with import to specify request parameters.

In [Session Tracking](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/Servlets11.html#wp64784) we discuss how an application must rewrite URLs to enable session tracking whenever the client turns off cookies. You can use the url tag to rewrite URLs returned from a JSP page. The tag includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged. Note that this feature requires that the URL be *relative*. The url tag takes param subtags to include parameters in the returned URL. For example, [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookcatalog.txt) rewrites the URL used to add a book to the shopping cart as follows:

<c:url var="url" value="/catalog" >

  <c:param name="Add" value="${bookId}" />

</c:url>

<p><strong><a href="${url}">

The redirect tag sends an HTTP redirect to the client. The redirect tag takes param subtags for including parameters in the returned URL.

### Miscellaneous Tags

The catch tag provides a complement to the JSP error page mechanism. It allows page authors to recover gracefully from error conditions that they can control. Actions that are of central importance to a page should *not* be encapsulated in a catch; in this way their exceptions will propagate instead to an error page. Actions with secondary importance to the page should be wrapped in a catch so that they never cause the error page mechanism to be invoked.

The exception thrown is stored in the variable identified by var, which always has page scope. If no exception occurred, the scoped variable identified by var is removed if it existed. If var is missing, the exception is simply caught and not saved.

The out tag evaluates an expression and outputs the result of the evaluation to the current JspWriter object. The syntax and attributes are as follows:

<c:out value="value" [escapeXml="{true|false}"]

  [default="*defaultValue*"] />

If the result of the evaluation is a java.io.Reader object, then data is first read from the Reader object and then written into the current JspWriter object. The special processing associated with Reader objects improves performance when a large amount of data must be read and then written to the response.

If escapeXml is true, the character conversions listed in [Table 14-4](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL4.html#wp85678) are applied.

|  |  |
| --- | --- |
| *Table 14-4 Character Conversions* |  |
| **Character** | **Character Entity Code** |
| < | &lt; |
| > | &gt; |
| & | &amp; |
| ' | &#039; |
| " | &#034; |

## ***XML Tag Library***

The JSTL XML tag set is listed in [Table 14-5](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL5.html#wp90179).

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-5 XML Tags* |  |  |  |
| **Area** | **Function** | **Tags** | **Prefix** |
| XML | Core | out  parse  set | x |
|  | Flow control | choose    when    otherwise  forEach  if |  |
|  | Transformation | transform    param |  |

A key aspect of dealing with XML documents is to be able to easily access their content. XPath (see [How XPath Works](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JAXPXSLT3.html#wp72547)), a W3C recommendation since 1999, provides an easy notation for specifying and selecting parts of an XML document. In the JSTL XML tags, XPath expressions specified using the select attribute are used to select portions of XML data streams. Note that XPath is used as a *local* expression language only for the select attribute. This means that values specified for select attributes are evaluated using the XPath expression language but that values for all other attributes are evaluated using the rules associated with the JSP 2.0 expression language.

In addition to the standard XPath syntax, the JSTL XPath engine supports the following scopes to access web application data within an XPath expression:

* $foo
* $param:
* $header:
* $cookie:
* $initParam:
* $pageScope:
* $requestScope:
* $sessionScope:
* $applicationScope:

These scopes are defined in exactly the same way as their counterparts in the JSP expression language discussed in [Implicit Objects](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html#wp71043). [Table 14-6](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL5.html#wp76812) shows some examples of using the scopes.

|  |  |
| --- | --- |
| *Table 14-6 Example XPath Expressions* |  |
| **XPath Expression** | **Result** |
| $sessionScope:profile | The session-scoped EL variable named profile |
| $initParam:mycom.productId | The String value of the mycom.productId context parameter |

The XML tags are illustrated in another version (bookstore5) of the Duke's Bookstore application. This version replaces the database with an XML representation of the bookstore database, which is retrieved from another web application. The directions for building and deploying this version of the application are in [The Example JSP Document](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSPX2.html#wp116707). A sample bookstore5.war is provided in <*INSTALL*>/j2eetutorial14/examples/web/provided-wars/.

### Core Tags

The core XML tags provide basic functionality to easily parse and access XML data.

The parse tag parses an XML document and saves the resulting object in the EL variable specified by attribute var. In bookstore5, the XML document is parsed and saved to a context attribute in [parsebooks.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore5/web/parsebooks.txt), which is included by all JSP pages that need access to the document:

<c:if test="${applicationScope:booklist == null}" >

  <c:import url="${initParam.booksURL}" var="xml" />

  <x:parse doc="${xml}" var="booklist" scope="application" />

</c:if>

The set and out tags parallel the behavior described in [Variable Support Tags](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL4.html#wp63882) and [Miscellaneous Tags](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL4.html#wp85708) for the XPath local expression language. The set tag evaluates an XPath expression and sets the result into a JSP EL variable specified by attribute var. The out tag evaluates an XPath expression on the current context node and outputs the result of the evaluation to the current JspWriter object.

The JSP page [bookdetails.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore5/web/bookdetails.txt) selects a book element whose id attribute matches the request parameter bookId and sets the abook attribute. The out tag then selects the book's title element and outputs the result.

<x:set var="abook"

  select="$applicationScope.booklist/

    books/book[@id=$param:bookId]" />

  <h2><x:out select="$abook/title"/></h2>

As you have just seen, x:set stores an internal XML representation of a *node* retrieved using an XPath expression; it doesn't convert the selected node into a String and store it. Thus, x:set is primarily useful for storing parts of documents for later retrieval.

If you want to store a String, you must use x:out within c:set. The x:out tag converts the node to a String, and c:set then stores the String as an EL variable. For example, [bookdetails.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore5/web/bookdetails.txt) stores an EL variable containing a book price, which is later provided as the value of a fmt tag, as follows:

<c:set var="price">

  <x:out select="$abook/price"/>

</c:set>

<h4><fmt:message key="ItemPrice"/>:

  <fmt:formatNumber value="${price}" type="currency"/>

The other option, which is more direct but requires that the user have more knowledge of XPath, is to coerce the node to a String manually by using XPath's string function.

<x:set var="price" select="string($abook/price)"/>

### Flow Control Tags

The XML flow control tags parallel the behavior described in [Flow Control Tags](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL4.html#wp74001) for XML data streams.

The JSP page [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore5/web/bookcatalog.txt) uses the forEach tag to display all the books contained in booklist as follows:

<x:forEach var="book"

  select="$applicationScope:booklist/books/\*">

  <tr>

    <c:set var="bookId">

      <x:out select="$book/@id"/>

    </c:set>=

    <td bgcolor="#ffffaa">

      <c:url var="url"

      value="/bookdetails" >

        <c:param name="bookId" value="${bookId}" />

        <c:param name="Clear" value="0" />

      </c:url>

      <a href="${url}">

      <strong><x:out select="$book/title"/>&nbsp;

      </strong></a></td>

    <td bgcolor="#ffffaa" rowspan=2>

      <c:set var="price">

        <x:out select="$book/price"/>

      </c:set>

      <fmt:formatNumber value="${price}" type="currency"/>

      &nbsp;

    </td>

    <td bgcolor="#ffffaa" rowspan=2>

    <c:url var="url" value="/catalog" >

      <c:param name="Add" value="${bookId}" />

    </c:url>

    <p><strong><a href="${url}">&nbsp;

      <fmt:message key="CartAdd"/>&nbsp;</a>

    </td>

  </tr>

  <tr>

    <td bgcolor="#ffffff">

    &nbsp;&nbsp;<fmt:message key="By"/> <em>

      <x:out select="$book/firstname"/>&nbsp;

      <x:out select="$book/surname"/></em></td></tr>

</x:forEach>

## ***Internationalization Tag Library***

Chapter [22](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/WebI18N.html#wp83291) covers how to design web applications so that they conform to the language and formatting conventions of client locales. This section describes tags that support the internationalization of JSP pages.

JSTL defines tags for setting the locale for a page, creating locale-sensitive messages, and formatting and parsing data elements such as numbers, currencies, dates, and times in a locale-sensitive or customized manner. [Table 14-7](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL6.html#wp86733) lists the tags.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-7 Internationalization Tags* |  |  |  |
| **Area** | **Function** | **Tags** | **Prefix** |
| I18n | Setting Locale | setLocale  requestEncoding | fmt |
|  | Messaging | bundle  message    param  setBundle |  |
|  | Number and Date Formatting | formatNumber  formatDate  parseDate  parseNumber  setTimeZone  timeZone |  |

JSTL i18n tags use a localization context to localize their data. A *localization context* contains a locale and a resource bundle instance. To specify the localization context at deployment time, you define the context parameter javax.servlet.jsp.jstl.fmt.localizationContext, whose value can be a javax.servlet.jsp.jstl.fmt.LocalizationContext or a String. A String context parameter is interpreted as a resource bundle base name. For the Duke's Bookstore application, the context parameter is the String messages.BookstoreMessages. When a request is received, JSTL automatically sets the locale based on the value retrieved from the request header and chooses the correct resource bundle using the base name specified in the context parameter.

### Setting the Locale

The setLocale tag is used to override the client-specified locale for a page. The requestEncoding tag is used to set the request's character encoding, in order to be able to correctly decode request parameter values whose encoding is different from ISO-8859-1.

### Messaging Tags

By default, the capability to sense the browser locale setting is enabled in JSTL. This means that the client determines (via its browser setting) which locale to use, and allows page authors to cater to the language preferences of their clients.

#### **The setBundle and bundle Tags**

You can set the resource bundle at runtime with the JSTL fmt:setBundle and fmt:bundle tags. fmt:setBundle is used to set the localization context in a variable or configuration variable for a specified scope. fmt:bundle is used to set the resource bundle for a given tag body.

#### **The message Tag**

The message tag is used to output localized strings. The following tag from [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookcatalog.txt) is used to output a string inviting customers to choose a book from the catalog.

<h3><fmt:message key="Choose"/></h3>

The param subtag provides a single argument (for parametric replacement) to the compound message or pattern in its parent message tag. One param tag must be specified for each variable in the compound message or pattern. Parametric replacement takes place in the order of the param tags.

### Formatting Tags

JSTL provides a set of tags for parsing and formatting locale-sensitive numbers and dates.

The formatNumber tag is used to output localized numbers. The following tag from [bookshowcart.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookshowcart.txt) is used to display a localized price for a book.

<fmt:formatNumber value="${book.price}" type="currency"/>

Note that because the price is maintained in the database in dollars, the localization is somewhat simplistic, because the formatNumber tag is unaware of exchange rates. The tag formats currencies but does not convert them.

Analogous tags for formatting dates (formatDate) and for parsing numbers and dates (parseNumber, parseDate) are also available. The timeZone tag establishes the time zone (specified via the value attribute) to be used by any nested formatDate tags.

In [bookreceipt.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookreceipt.txt), a "pretend" ship date is created and then formatted with the formatDate tag:

<jsp:useBean id="now" class="java.util.Date" />

<jsp:setProperty name="now" property="time"

  value="${now.time + 432000000}" />

<fmt:message key="ShipDate"/>

<fmt:formatDate value="${now}" type="date"

  dateStyle="full"/>.

## ***SQL Tag Library***

The JSTL SQL tags for accessing databases listed in [Table 14-8](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL7.html#wp84217) are designed for quick prototyping and simple applications. For production applications, database operations are normally encapsulated in JavaBeans components.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-8 SQL Tags* |  |  |  |
| **Area** | **Function** | **Tags** | **Prefix** |
| Database |  | setDataSource | sql |
|  | SQL | query    dateParam   param transaction  update   dateParam    param |  |

The setDataSource tag allows you to set data source information for the database. You can provide a JNDI name or DriverManager parameters to set the data source information. All of the Duke's Bookstore pages that have more than one SQL tag use the following statement to set the data source:

<sql:setDataSource dataSource="jdbc/BookDB" />

The query tag performs an SQL query that returns a result set. For parameterized SQL queries, you use a nested param tag inside the query tag.

In [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookcatalog.txt), the value of the Add request parameter determines which book information should be retrieved from the database. This parameter is saved as the attribute name bid and is passed to the param tag.

<c:set var="bid" value="${param.Add}"/>

<sql:query var="books" >

  select \* from PUBLIC.books where id = ?

  <sql:param value="${bid}" />

</sql:query>

The update tag is used to update a database row. The transaction tag is used to perform a series of SQL statements atomically.

The JSP page bookreceipt.jsp page uses both tags to update the database inventory for each purchase. Because a shopping cart can contain more than one book, the transaction tag is used to wrap multiple queries and updates. First, the page establishes that there is sufficient inventory; then the updates are performed.

<c:set var="sufficientInventory" value="true" />

<sql:transaction>

  <c:forEach var="item" items="${sessionScope.cart.items}">

    <c:set var="book" value="${item.item}" />

    <c:set var="bookId" value="${book.bookId}" />

    <sql:query var="books"

      sql="select \* from PUBLIC.books where id = ?" >

      <sql:param value="${bookId}" />

    </sql:query>

    <jsp:useBean id="inventory"

      class="database.BookInventory" />

    <c:forEach var="bookRow" begin="0"

      items="${books.rowsByIndex}">

      <jsp:useBean id="bookRow" type="java.lang.Object[]" />

      <jsp:setProperty name="inventory" property="quantity"

        value="${bookRow[7]}" />

      <c:if test="${item.quantity > inventory.quantity}">

        <c:set var="sufficientInventory" value="false" />

        <h3><font color="red" size="+2">

        <fmt:message key="OrderError"/>

        There is insufficient inventory for

        <i>${bookRow[3]}</i>.</font></h3>

      </c:if>

    </c:forEach>

  </c:forEach>

  <c:if test="${sufficientInventory == 'true'}" />

    <c:forEach var="item" items="${sessionScope.cart.items}">

     <c:set var="book" value="${item.item}" />

     <c:set var="bookId" value="${book.bookId}" />

      <sql:query var="books"

        sql="select \* from PUBLIC.books where id = ?" >

        <sql:param value="${bookId}" />

      </sql:query>

      <c:forEach var="bookRow" begin="0"

        items="${books.rows}">

        <sql:update var="books" sql="update PUBLIC.books set

          inventory = inventory - ? where id = ?" >

          <sql:param value="${item.quantity}" />

          <sql:param value="${bookId}" />

        </sql:update>

      </c:forEach>

    </c:forEach>

    <h3><fmt:message key="ThankYou"/>

      ${param.cardname}.</h3><br>

  </c:if>

</sql:transaction>

### query Tag Result Interface

The Result interface is used to retrieve information from objects returned from a query tag.

public interface Result

  public String[] getColumnNames();

  public int getRowCount()

  public Map[] getRows();

  public Object[][] getRowsByIndex();

  public boolean isLimitedByMaxRows();

For complete information about this interface, see the API documentation for the [JSTL](http://java.sun.com/products/jsp/jstl/1.1/docs/api/index.html) packages.

The var attribute set by a query tag is of type Result. The getRows method returns an array of maps that can be supplied to the items attribute of a forEach tag. The JSTL expression language converts the syntax ${*result*.rows} to a call to *result*.getRows. The expression ${books.rows} in the following example returns an array of maps.

When you provide an array of maps to the forEach tag, the var attribute set by the tag is of type Map. To retrieve information from a row, use the get("*colname*") method to get a column value. The JSP expression language converts the syntax ${*map*.*colname*} to a call to *map*.get("*colname*"). For example, the expression ${book.title} returns the value of the title entry of a book map.

The Duke's Bookstore page [bookdetails.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookdetails.txt) retrieves the column values from the book map as follows.

<c:forEach var="book" begin="0" items="${books.rows}">

  <h2>${book.title}</h2>

  &nbsp;<fmt:message key="By"/> <em>${book.firstname}

  ${book.surname}</em>&nbsp;&nbsp;

  (${book.year})<br> &nbsp; <br>

  <h4><fmt:message key="Critics"/></h4>

  <blockquote>${book.description}</blockquote>

  <h4><fmt:message key="ItemPrice"/>:

  <fmt:formatNumber value="${book.price}" type="currency"/>

  </h4>

</c:forEach>

The following excerpt from [bookcatalog.jsp](http://72.5.124.55/j2ee/1.4/docs/tutorial/examples/web/bookstore4/web/bookcatalog.txt) uses the Row interface to retrieve values from the columns of a book row using scripting language expressions. First, the book row that matches a request parameter (bid) is retrieved from the database. Because the bid and bookRow objects are later used by tags that use scripting language expressions to set attribute values and by a scriptlet that adds a book to the shopping cart, both objects are declared as scripting variables using the jsp:useBean tag. The page creates a bean that describes the book, and scripting language expressions are used to set the book properties from book row column values. Then the book is added to the shopping cart.

You might want to compare this version of bookcatalog.jsp to the versions in [JavaServer Pages Technology](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSPIntro.html#wp100465) and [Custom Tags in JSP Pages](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSPTags.html#wp74644) that use a book database JavaBeans component.

<sql:query var="books"

  dataSource="${applicationScope.bookDS}">

  select \* from PUBLIC.books where id = ?

  <sql:param value="${bid}" />

</sql:query>

<c:forEach var="bookRow" begin="0"

      items="${books.rowsByIndex}">

  <jsp:useBean id="bid" type="java.lang.String" />

  <jsp:useBean id="bookRow" type="java.lang.Object[]" />

  <jsp:useBean id="addedBook" class="database.BookDetails"

    scope="page" >

    <jsp:setProperty name="addedBook" property="bookId"

      value="${bookRow[0]}" />

    <jsp:setProperty name="addedBook" property="surname"

      value="${bookRow[1]}" />

    <jsp:setProperty name="addedBook" property="firstName"

      value="${bookRow[2]}" />

    <jsp:setProperty name="addedBook" property="title"

      value="${bookRow[3]}" />

    <jsp:setProperty name="addedBook" property="price"

      value="${bookRow[4])}" />

    <jsp:setProperty name="addedBook" property="year"

      value="${bookRow[6]}" />

    <jsp:setProperty name="addedBook"

      property="description"

      value="${bookRow[7]}" />

    <jsp:setProperty name="addedBook" property="inventory"

      value="${bookRow[8]}" />

  </jsp:useBean>

  <% cart.add(bid, addedBook); %>

  ...

</c:forEach>

## ***Functions***

[Table 14-9](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/JSTL8.html#wp84687) lists the JSTL functions

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-9 Functions* |  |  |  |
| **Area** | **Function** | **Tags** | **Prefix** |
| **Functions** | Collection length | length | **fn** |
|  | String  manipulation | toUpperCase, toLowerCase  substring, substringAfter,  substringBefore  trim  replace  indexOf, startsWith, endsWith, contains, containsIgnoreCase  split, join  escapeXml |  |

.

Although the java.util.Collection interface defines a size method, it does not conform to the JavaBeans component design pattern for properties and so cannot be accessed via the JSP expression language. The length function can be applied to any collection supported by the c:forEach and returns the length of the collection. When applied to a String, it returns the number of characters in the string.

For example, the index.jsp page of the hello1 application introduced in Chapter [3](http://72.5.124.55/j2ee/1.4/docs/tutorial/doc/WebApp.html#wp83291) uses the fn:length function and the c:if tag to determine whether to include a response page:

<%@ taglib uri="http://java.sun.com/jsp/jstl/core"

  prefix="c" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"

  prefix="fn" %>

<html>

<head><title>Hello</title></head>

...

<input type="text" name="username" size="25">

<p></p>

<input type="submit" value="Submit">

<input type="reset" value="Reset">

</form>

<c:if test="${fn:length(param.username) > 0}" >

<%@include file="response.jsp" %>

</c:if>

</body>

</html>

The rest of the JSTL functions are concerned with string manipulation:

* toUpperCase, toLowerCase: Changes the capitalization of a string
* substring, substringBefore, substringAfter: Gets a subset of a string
* trim: Trims whitespace from a string
* replace: Replaces characters in a string
* indexOf, startsWith, endsWith, contains, containsIgnoreCase: Checks whether a string contains another string
* split: Splits a string into an array
* join: Joins a collection into a string
* escapeXml: Escapes XML characters in a string

1. **Modify the web.xml file to include the TLD files.**  
     
   Finally, you must modify your web.xml, and add an entry for all eight of the tag libraries that you added. This consists of adding <taglib> directives inside the main <web-app> directive. The entries you should add are listed here.
2. <taglib>
3. <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
4. <taglib-location>/WEB-INF/fmt.tld</taglib-location>
5. </taglib>
6. <taglib>
7. <taglib-uri>http://java.sun.com/jstl/fmt-rt</taglib-uri>
8. <taglib-location>/WEB-INF/fmt-rt.tld</taglib-location>
9. </taglib>
10. <taglib>
11. <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
12. <taglib-location>/WEB-INF/c.tld</taglib-location>
13. </taglib>
14. <taglib>
15. <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
16. <taglib-location>/WEB-INF/c-rt.tld</taglib-location>
17. </taglib>
18. <taglib>
19. <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
20. <taglib-location>/WEB-INF/sql.tld</taglib-location>
21. </taglib>
22. <taglib>
23. <taglib-uri>http://java.sun.com/jstl/sql-rt</taglib-uri>
24. <taglib-location>/WEB-INF/sql-rt.tld</taglib-location>
25. </taglib>
26. <taglib>
27. <taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
28. <taglib-location>/WEB-INF/x.tld</taglib-location>
29. </taglib>
30. <taglib>
31. <taglib-uri>http://java.sun.com/jstl/x-rt</taglib-uri>
32. <taglib-location>/WEB-INF/x-rt.tld</taglib-location>
33. </taglib>

After completing the preceding three steps, you are now ready to test your installation of JSTL. This can be done by creating a JSP page that uses JSTL. A very simple example would be the "count to ten" example shown before. You should place your JSP file inside the Webroot directory (C:\Program Files\Apache Tomcat 4.0\webapps\ROOT). Once the Tomcat server is started, you should now be able to browse to <http://127.0.0.1:8080/count.jsp> to see your page.

If you do not have JSTL installed correctly, there will likely be no error message. If JSTL is not interpreting your tags, they will be passed through directly to the Web browser. The Web browser will interpret these tags as unknown HTML tags. Most browsers simply ignore the unknown HTML tags.

### **The JSTL Tag Libraries**

JSTL is often spoken of as a single-tag library. JSTL is actually four tag libraries. These tag libraries are summarized as follows.

* **Core Tag Library**—Contains tags that are essential to nearly any Web application. Examples of core tag libraries include looping, expression evaluation, and basic input and output.
* **Formatting/Internationalization Tag Library**—Contains tags that are used to and parse data. Some of these tags will parse data, such as dates, differently based on the current locale.
* **Database Tag Library**—Contains tags that can be used to access SQL databases. These tags are normally used only to create prototype programs. This is because most programs will not handle database access directly from JSP pages. Database access should be embedded in EJBs that are accessed by the JSP pages.
* **XML Tag Library**—Contains tags that can be used to access XML elements. Because XML is used in many Web applications, XML processing is an important feature of JSTL.

## ***What Is a Custom Tag?***

A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a *tag handler*. The Web container then invokes those operations when the JSP page's servlet is executed.

Custom tags have a rich set of features. They can

* Be customized via attributes passed from the calling page.
* Access all the objects available to JSP pages.
* Modify the response generated by the calling page.
* Communicate with each other. You can create and initialize a JavaBeans component, create a variable that refers to that bean in one tag, and then use the bean in another tag.
* Be nested within one another, allowing for complex interactions within a JSP page.