JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8651-INTERNET PROGRAMMING

(Regulation 2017)

Year/Semester: III / 06 CSE

2020 – 2021

Prepared by

Dr. K. Tamilarasi

Associate Professor /CSE

UNIT III

SERVER SIDE PROGRAMMING

Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies- Installing and Configuring Apache Tomcat Web Server- DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example - JSP: Understanding Java Server Pages-JSP Standard Tag Library (JSTL)-Creating HTML forms by embedding JSP code.

3.1 Servlets: Java Servlet Architecture

- **Java** Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.
  - Performance is significantly better.
  - Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
  - Servlets are platform-independent because they are written in Java.
  - Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
  - The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

3.1.1 Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.

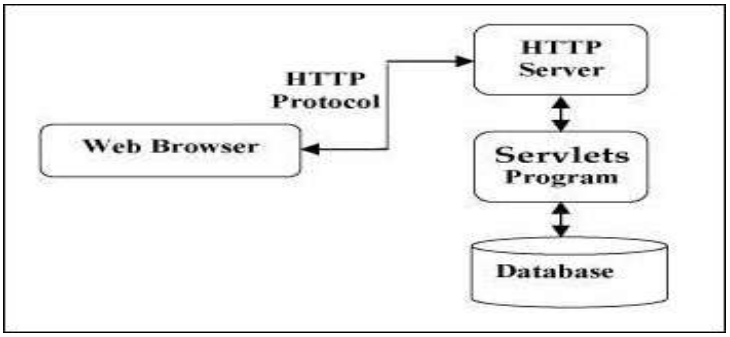


Fig: 3.1 Servlet Architecture

3.1.2 Servlets Tasks

Servlets perform the following major tasks –

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages

- Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.
- Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.
- These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.
- Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

3.1.3 Basic Structure of a Servlet

```
public class firstServlet extends HttpServlet {
  public void init() {
    /* Put your initialization code in this method,
     * as this method is called only once */
  }
  public void service() {
    // Service request for Servlet
  }
  public void destroy() {
```

```
    // For taking the servlet out of service, this method is called only once
  }
}
```

3.2 SERVLET LIFE CYCLE

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the init() method.
- The servlet calls service() method to process a client's request.
- The servlet is terminated by calling the destroy() method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

3.2.1 The init() Method

- The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this –

```
Public void init() throws ServletException {

// Initialization code

}
```

3.2.2 The service() Method

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.

- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Here is the signature of this method −

```
public void service(ServletRequest request, ServletResponse response)
   throws ServletException, IOException {
}
```

- The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
- The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

3.2.3 The doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

3.2.4 The doPost() Method

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

3.2.5 The destroy() Method

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

```
public void destroy() {

  // Finalization code...

}
```

3.2.6 Architecture Diagram

- The following figure depicts a typical servlet life-cycle scenario.
- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

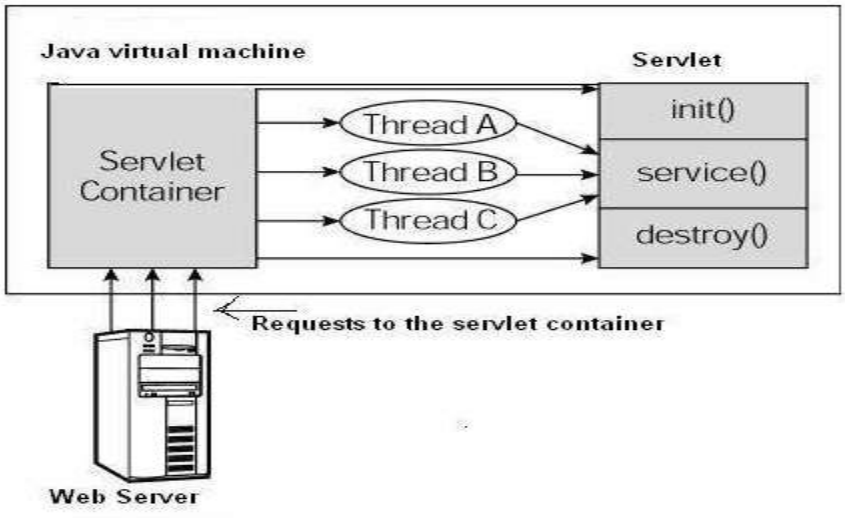


Fig 3.2 Servlet Life Cycle

3.3 FORM GET AND POST ACTIONS

3.3.1 GET Method

- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** (question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

3.3.2 POST Method

- A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

3.3.3 Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

3.3.4 GET Method Example using URL

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

- Given below is the HelloForm.java servlet program to handle input given by web browser. We are going to use getParameter() method which makes it very easy to access passed information −

```
// Import required java libraries

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {
```

```
        // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";
          out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
          "</ul>\n" +
        "</body>" +
      "</html>"
    );
  }
}
```

- Assuming your environment is set up properly, compile HelloForm.java as follows −

$ javac HelloForm.java

- If everything goes fine, above compilation would produce HelloForm.class file. Next you would have to copy this class file in <Tomcat installationdirectory>/webapps/ROOT/WEB-

INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
  <servlet-name>HelloForm</servlet-name>
  <servlet-class>HelloForm</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloForm</servlet-name>
  <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

- Now type http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result –

Output

Using GET Method to Read Form Data

First Name: ZARA

Last Name: ALI

3.3.5 GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
  <body>
    <form action = "HelloForm" method = "GET">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
```

```
</html>
```

3.3.6 POST Method Example Using Form

- Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
          out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
```

```
                + request.getParameter("first_name") + "\n" +
                "  <li><b>Last Name</b>: "
                + request.getParameter("last_name") + "\n" +
              "</ul>\n" +
            "</body>"
          "</html>"
      );
   }
   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      doGet(request, response);
   }
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

```
<html>
   <body>
      <form action = "HelloForm" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

3.3.7 Get vs. Post

S.No	GET	POST
1	In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is

		sent in body.
2	Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
3	Get request can be bookmarked.	Post request cannot be bookmarked.
4	Get request is idempotent . It means second request will be ignored until response of first request is delivered	Post request is non-idempotent.
5	Get request is more efficient and used more than Post.	Post request is less efficient and used less than get.

Two common methods for the request-response between a server and client are:

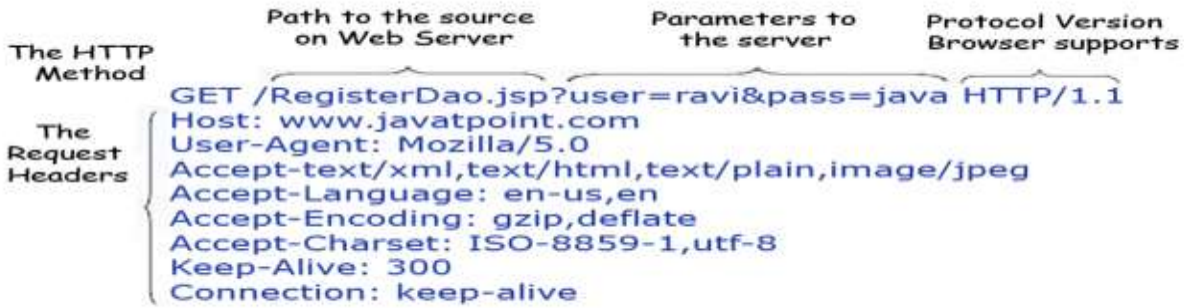
- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

Anatomy of Get Request

- The query string (name/value pairs) is sent inside the URL of a GET request:

GET/RegisterDao.jsp?name1=value1&name2=value2

- As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information is sent to the server.



- Some other features of GET requests are:
  - It remains in the browser history
  - It can be bookmarked
  - It can be cached
  - It have length restrictions
  - It should never be used when dealing with sensitive data
  - It should only be used for retrieving the data

Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a **POST request:**

POST/RegisterDao.jsp HTTP/1.1

Host: www. javatpoint.com

name1=value1&name2=value2

As we know, in case of post request original data is sent in message body. Let's see how information is passed to the server in case of post request.

- Some other features of POST requests are:
  - This requests cannot be bookmarked
  - This requests have no restrictions on length of data
  - This requests are never cached
  - This requests do not retain in the browser history

3.4 SESSION HANDLING

- HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Still there are following three ways to maintain session between web client and web server

3.4.1 Cookies

- A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the recieved cookie.

- This may not be an effective way because many time browser does not support a cookie, so I would not recommend to use this procedure to maintain the sessions.

3.4.2 Hidden Form Fields

- A web server can send a hidden HTML form field along with a unique session ID as follows −

<input type = "hidden" name = "sessionid" value = "12345">

- This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.
- This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

3.4.3 URL Rewriting

- You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.
- For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.
- URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies.
- The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

3.4.5 The HttpSession Object

- Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.
- You would get HttpSession object by calling the public method getSession() of HttpServletRequest, as below −

HttpSession session = request.getSession();

- *request.getSession()* before you send any document content to the client.

Sr.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew()** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Session Tracking Example

- This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
// Import required java libraries

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import java.util.*;

 // Extend HttpServlet class

public class SessionTrack extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)

      throws ServletException, IOException {
```

```
        // Create a session object if it is already not  created.
    HttpSession session = request.getSession(true);
        // Get session creation time.
    Date createTime = new Date(session.getCreationTime());
        // Get last access time of this web page.
    Date lastAccessTime = new Date(session.getLastAccessedTime());
    String title = "Welcome Back to my website";
    Integer visitCount = new Integer(0);
    String visitCountKey = new String("visitCount");
    String userIDKey = new String("userID");
    String userID = new String("ABCD");
    // Check if this is new comer on your web page.
    if (session.isNew()) {
      title = "Welcome to my website";
      session.setAttribute(userIDKey, userID);
    } else {
      visitCount = (Integer)session.getAttribute(visitCountKey);
      visitCount = visitCount + 1;
      userID = (String)session.getAttribute(userIDKey);
    }
    session.setAttribute(visitCountKey,  visitCount);
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

```
String docType =
  "<!doctype html public \"-//w3c//dtd html 4.0 " +
  "transitional//en\">\n";
out.println(docType +
  "<html>\n" +
    "<head><title>" + title + "</title></head>\n" +
          "<body bgcolor = \"#f0f0f0\">\n" +
      "<h1 align = \"center\">" + title + "</h1>\n" +
      "<h2 align = \"center\">Session Infomation</h2>\n" +
      "<table border = \"1\" align = \"center\">\n" +
                  "<tr bgcolor = \"#949494\">\n" +
        "  <th>Session info</th><th>value</th>
      </tr>\n" +
                    "<tr>\n" +
        "  <td>id</td>\n" +
        "  <td>" + session.getId() + "</td>
      </tr>\n" +
                  "<tr>\n" +
        "  <td>Creation Time</td>\n" +
        "  <td>" + createTime + "  </td>
      </tr>\n" +
                  "<tr>\n" +
        "  <td>Time of Last Access</td>\n" +
        "  <td>" + lastAccessTime + "  </td>
```

```
                </tr>\n" +
                        "<tr>\n" +
                  "  <td>User ID</td>\n" +
                  "  <td>" + userID + "  </td>
                </tr>\n" +
                        "<tr>\n" +
                  "  <td>Number of visits</td>\n" +
                  "  <td>" + visitCount + "</td>
                </tr>\n" +
              "</table>\n" +
            "</body>
          </html>"
      );
    }
```

3.4.6 Deleting Session Data

When you are done with a user's session data, you have several options

- **Remove a particular attribute** − You can call *public void removeAttribute(String name)* method to delete the value associated with a particular key.
- **Delete the whole session** − You can call *public void invalidate()* method to discard an entire session.
- **Setting Session timeout** − You can call *public void setMaxInactiveInterval(int interval)* method to set the timeout for a session individually.
- **Log the user out** − The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration** − If you are using Tomcat, apart from the above mentioned methods, you can configure session time out in web.xml file as follows.

```
<session-config>
  <session-timeout>15</session-timeout>
```

</session-config>

- The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.
- The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

3.5 UNDERSTANDING COOKIES

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.
- There are three steps involved in identifying returning users –
  - Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

3.5.1 The Anatomy of a Cookie

- Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A servlet that sets a cookie might send headers that look something like this −

HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html

3.5.4 Setting Cookies with Servlet

- Setting cookies with servlet involves three steps –
  i. Creating a Cookie object − You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

Cookie cookie = new Cookie("key","value");

  ii. Keep in mind, neither the name nor the value should contain white space or any of the following characters −

[ ] ( ) = , " / ? @ : ;

iii. Setting the maximum age − You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

cookie.setMaxAge(60 * 60 * 24);

iv. Sending the Cookie into the HTTP response headers − You use response.addCookie to add cookies in the HTTP response header as follows −

response.addCookie(cookie);

Example

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 // Extend HttpServlet class
public class HelloForm extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
          // Create cookies for first and last names.
      Cookie firstName = new Cookie("first_name", request.getParameter("first_name"));
      Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));
      // Set expiry date after 24 Hrs for both the cookies.
      firstName.setMaxAge(60*60*24);
      lastName.setMaxAge(60*60*24);
      // Add both the cookies in the response header.
      response.addCookie( firstName );
      response.addCookie( lastName );
      // Set response content type
      response.setContentType("text/html");
       PrintWriter out = response.getWriter();
      String title = "Setting Cookies Example";
      String docType =
```

```
"<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";out.println(docType +
"<html>\n" +
  "<head>
    <title>" + title + "</title>
  </head>\n" +
  "<body bgcolor = \"#f0f0f0\">\n" +
    "<h1 align = \"center\">" + title + "</h1>\n" +
    "<ul>\n" +
      "  <li><b>First Name</b>: "
      + request.getParameter("first_name") + "\n" +
      "  <li><b>Last Name</b>: "
      + request.getParameter("last_name") + "\n" +
    "</ul>\n" +
  "</body>
</html>"); }}
```

3.5.3 Servlet Cookies Methods

Sr.No.	Method & Description
1	public void setDomain(String pattern) This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	public String getDomain() This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	public void setMaxAge(int expiry) This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	public int getMaxAge() This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
5	public String getName() This method returns the name of the cookie. The name cannot be changed after creation.

6	public void setValue(String newValue) This method sets the value associated with the cookie
7	public String getValue() This method gets the value associated with the cookie.
8	public void setPath(String uri) This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	public String getPath() This method gets the path to which this cookie applies.
10	public void setSecure(boolean flag) This method sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
11	public void setComment(String purpose) This method specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.
12	public String getComment() This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

3.5.4 Reading Cookies with Servlet

- To read cookies, you need to create an array of javax.servlet.http.Cookie objects by calling the getCookies() method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

3.5.6 Delete Cookies with Servlet

- To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps –
  - Read an already existing cookie and store it in Cookie object.
  - Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
  - Add this cookie back into response header.

3.6 INSTALLING AND CONFIGURING APACHE TOMCAT WEB SERVER

- A development environment is where you would develop your Servlet, test them and finally run them.
- Like any other Java program, you need to compile a servlet by using the Java compiler javac and after compilation the servlet application, it would be deployed in a configured environment to test and run..
- This development environment setup involves the following steps −

3.6.1 Setting up Java Development Kit

- This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.
- You can download SDK from Oracle's Java site − Java SE Downloads.
- Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
- If you are running Windows and installed the SDK in C:\jdk1.8.0_65, you would put the following line in your C:\autoexec.bat file.

set PATH = C:\jdk1.8.0_65\bin;%PATH%

set JAVA_HOME = C:\jdk1.8.0_65

- Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.
- On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.8.0_65 and you use the C shell, you would put the following into your .cshrc file.

setenv PATH /usr/local/jdk1.8.0_65/bin:$PATH

setenv JAVA_HOME /usr/local/jdk1.8.0_65

- Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

3.6.2 Setting up Web Server − Tomcat

- A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine –
- Download latest version of Tomcat from https://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-8.0.28 on windows, or /usr/local/apache-tomcat-8.0.289 on Linux/Unix and create CATALINA_HOME environment variable pointing to these locations.
- Tomcat can be started by executing the following commands on windows machine −

%CATALINA_HOME%\bin\startup.bat

or

C:\apache-tomcat-8.0.28\bin\startup.bat

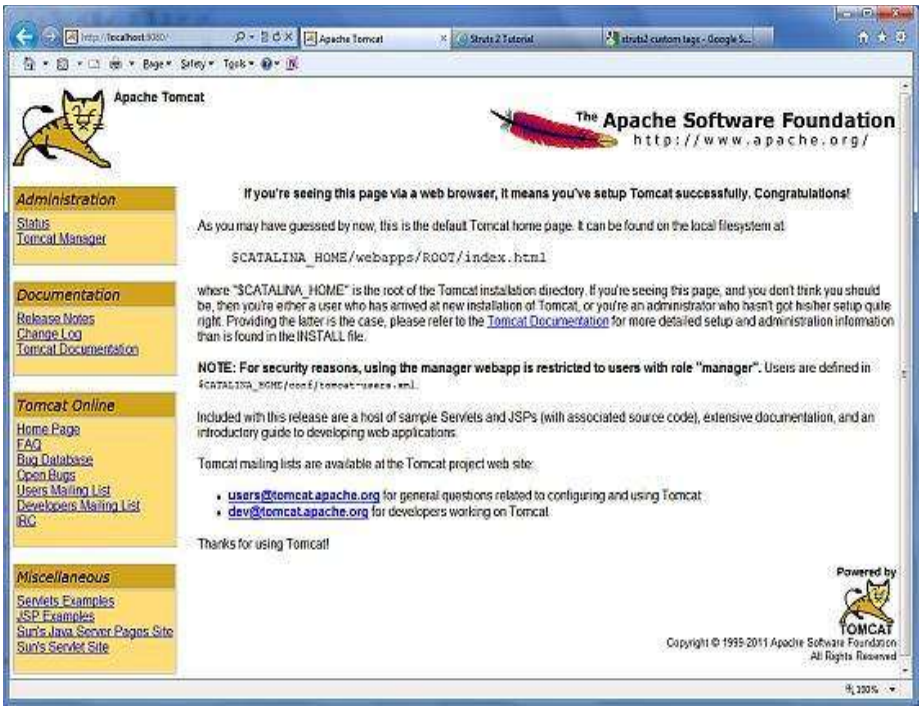
- Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine −

$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-8.0.28/bin/startup.sh

- After startup, the default web applications included with Tomcat will be available by visiting http://localhost:8080/. If everything is fine then it should display following result −



- Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site − http://tomcat.apache.org
- Tomcat can be stopped by executing the following commands on windows machine −

C:\apache-tomcat-8.0.28\bin\shutdown

- Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine −

/usr/local/apache-tomcat-8.0.28/bin/shutdown.sh

3.6.3 Setting Up the CLASSPATH

- Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.
- If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.

set CATALINA = C:\apache-tomcat-8.0.28

set CLASSPATH = %CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%

- Alternatively, on Windows NT/2000/XP, you could go to My Computer −> Properties −> Advanced −> Environment Variables. Then, you would update the CLASSPATH value and press the OK button.
- On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your .cshrc file.

setenv CATALINA = /usr/local/apache-tomcat-8.0.28

setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH

- **NOTE** − Assuming that your development directory is C:\ServletDevel (Windows) or /usr/ServletDevel (Unix) then you would need to add these directories as well in CLASSPATH in similar way as you have added above.

3.7 DATABASE CONNECTIVITY

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

Fig 3.3 Java Db Connectivity

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLException

2) public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the Oracle database

```
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example to create the statement object

```
Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close();
```

3.7.1 Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

- **Driver class:** The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.
- **Connection URL:** The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
- **Username:** The default username for the oracle database is system.
- **Password:** It is the password given by the user at the time of installing the oracle database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from emp table. Here, system and oracle are the username and password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
  //step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
  //step3 create the statement object
Statement stmt=con.createStatement();
  //step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
  //step5 close the connection object
con.close();
  }catch(Exception e){ System.out.println(e);}
  }
}
```

Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

1. C:>set classpath=c:\folder\ojdbc14.jar;.;

How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

3.7.2 Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));

Example to Connect Java Application with mysql database

In this example, soon is the database name, root is the username and password both.

```
1. import java.sql.*;
2. class MysqlCon{
3. public static void main(String args[]){
4. try{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. //here sonoo is database name, root is username and password
9. Statement stmt=con.createStatement();
10. ResultSet rs=stmt.executeQuery("select * from emp");
11. while(rs.next())
12. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
13. con.close();
14. }catch(Exception e){ System.out.println(e);}
15. }
16. }
```

The above example will fetch all the records of emp table.

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

download the jar file mysql-connector.jar

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

3.8 JSP: UNDERSTANDING JAVA SERVER PAGES:

- Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

Why to Learn JSP?

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.
- Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

3.8.1 Applications of JSP

✓ **JSP vs. Active Server Pages (ASP)**

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

✓ **JSP vs. Pure Servlets**

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

✓ **JSP vs. Server-Side Includes (SSI)**

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

✓ **JSP vs. JavaScript**

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

✓ **JSP vs. Static HTML**

Regular HTML, of course, cannot contain dynamic information.

3.9 JSP STANDARD TAG LIBRARY (JSTL)

- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
- It also provides a framework for integrating the existing custom tags with the JSTL tags.

Install JSTL Library

To begin working with JSP tages you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps −

- Step 1 − Download the binary distribution from Apache Standard Taglib and unpack the compressed file.
- Step 2 − To use the Standard Taglib from its Jakarta Taglibs distribution, simply copy the JAR files in the distribution's 'lib' directory to your application's webapps\ROOT\WEB-INF\lib directory.
- To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions

Core Tags

The core group of tags are the most commonly used JSTL tags. Following is the syntax to include the JSTL Core library in your JSP −

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

Following table lists out the core JSTL Tags −

S.No.	Tag & Description
1	<c:out> Like <%= ... >, but for expressions
2	<c:set > Sets the result of an expression evaluation in a 'scope'
3	<c:remove > Removes a scoped variable (from a particular scope, if specified).
4	<c:catch> Catches any Throwable that occurs in its body and optionally exposes it.
5	<c:if> Simple conditional tag which evalutes its body if the supplied condition is true.
6	<c:choose> Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>.
7	<c:when> Subtag of <choose> that includes its body if its condition evalutes to 'true'.
8	<c:otherwise > Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluated to 'false'.
9	<c:import> Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
10	<c:forEach > The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .
11	<c:forTokens> Iterates over tokens, separated by the supplied delimeters.
12	<c:param> Adds a parameter to a containing 'import' tag's URL.

13	<c:redirect > Redirects to a new URL.
14	<c:url> Creates a URL with optional query parameters

Formatting Tags

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites. Following is the syntax to include Formatting library in your JSP −

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

Following table lists out the Formatting JSTL Tags −

S.No.	Tag & Description
1	<fmt:formatNumber> To render numerical value with specific precision or format.
2	<fmt:parseNumber> Parses the string representation of a number, currency, or percentage.
3	<fmt:formatDate> Formats a date and/or time using the supplied styles and pattern.
4	<fmt:parseDate> Parses the string representation of a date and/or time
5	<fmt:bundle> Loads a resource bundle to be used by its tag body.
6	<fmt:setLocale> Stores the given locale in the locale configuration variable.
7	<fmt:setBundle> Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
8	<fmt:timeZone> Specifies the time zone for any time formatting or parsing actions nested in its body.
9	<fmt:setTimeZone> Stores the given time zone in the time zone configuration variable
10	<fmt:message> Displays an internationalized message.
11	<fmt:requestEncoding> Sets the request character encoding

SQL Tags

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

Following is the syntax to include JSTL SQL library in your JSP −

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

Following table lists out the SQL JSTL Tags –

S.No.	Tag & Description
1	<sql:setDataSource> Creates a simple DataSource suitable only for prototyping
2	<sql:query> Executes the SQL query defined in its body or through the sql attribute.
3	<sql:update> Executes the SQL update defined in its body or through the sql attribute.
4	<sql:param> Sets a parameter in an SQL statement to the specified value.
5	<sql:dateParam> Sets a parameter in an SQL statement to the specified java.util.Date value.
6	<sql:transaction > Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

XML tags

The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with the XML data. This includes parsing the XML, transforming the XML data, and the flow control based on the XPath expressions.

<%@ taglib prefix = "x"

uri = "http://java.sun.com/jsp/jstl/xml" %>

Before you proceed with the examples, you will need to copy the following two XML and XPath related libraries into your <Tomcat Installation Directory>\lib −

- XercesImpl.jar − Download it from https://www.apache.org/dist/xerces/j/
- xalan.jar − Download it from https://xml.apache.org/xalan-j/index.html

Following is the list of XML JSTL Tags –

S.No.	Tag & Description
1	<x:out> Like <%= ... >, but for XPath expressions.
2	<x:parse> Used to parse the XML data specified either via an attribute or in the tag body.
3	<x:set > Sets a variable to the value of an XPath expression.
4	<x:if >

	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
5	<x:forEach> To loop over nodes in an XML document.
6	<x:choose> Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise> tags.
7	<x:when > Subtag of <choose> that includes its body if its expression evalutes to 'true'.
8	<x:otherwise > Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluates to 'false'.
9	<x:transform > Applies an XSL transformation on a XML document
10	<x:param > Used along with the transform tag to set a parameter in the XSLT stylesheet

JSTL Functions

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP −

<%@ taglib prefix = "fn"

uri = "http://java.sun.com/jsp/jstl/functions" %>

Following table lists out the various JSTL Functions −

S.No.	Function & Description
1	fn:contains() Tests if an input string contains the specified substring.
2	fn:containsIgnoreCase() Tests if an input string contains the specified substring in a case insensitive way.
3	fn:endsWith() Tests if an input string ends with the specified suffix.
4	fn:escapeXml() Escapes characters that can be interpreted as XML markup.
5	fn:indexOf() Returns the index withing a string of the first occurrence of a specified substring.
6	fn:join() Joins all elements of an array into a string.
7	fn:length() Returns the number of items in a collection, or the number of characters in a string.
8	fn:replace() Returns a string resulting from replacing in an input string all occurrences with a given string.
9	fn:split() Splits a string into an array of substrings.
10	fn:startsWith()

	Tests if an input string starts with the specified prefix.
11	fn:substring() Returns a subset of a string.
12	fn:substringAfter() Returns a subset of a string following a specific substring.
13	fn:substringBefore() Returns a subset of a string before a specific substring.
14	fn:toLowerCase() Converts all of the characters of a string to lower case.
15	fn:toUpperCase() Converts all of the characters of a string to upper case.
16	fn:trim() Removes white spaces from both ends of a string.

3.10 CREATING HTML FORMS BY EMBEDDING JSP CODE.

- The information in exactly the same way as the GET method, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.
- This message comes to the backend program in the form of the standard input which you can parse and use for your processing.
- JSP handles this type of requests using **getParameter()** method to read simple parameters and **getInputStream()** method to read binary data stream coming from the client.

Reading Form Data using JSP

JSP handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call **request.getParameter()** method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.
- **getInputStream()** − Call this method to read binary data stream coming from the client.

GET Method Example Using URL

The following URL will pass two values to HelloForm program using the GET method.

http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI

Below is the **main.jsp** JSP program to handle input given by web browser. We are going to use the **getParameter()** method which makes it very easy to access the passed information −

```
<html>
  <head>
    <title>Using GET Method to Read Form Data</title>
  </head>
    <body>
    <h1>Using GET Method to Read Form Data</h1>
    <ul>
      <li><p><b>First Name:</b>
        <%= request.getParameter("first_name")%>
      </p></li>
      <li><p><b>Last  Name:</b>
        <%= request.getParameter("last_name")%>
      </p></li>
    </ul>
    </body>
</html>
```

Now type ***http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI*** in your browser's **Location:box**. This will generate the following result −

Using GET Method to Read Form Data

- **First Name**: ZARA
- **Last Name**: ALI

GET Method Example Using Form

Following is an example that passes two values using the HTML FORM and the submit button. We are going to use the same JSP main.jsp to handle this input.

```
<html>
  <body>
     <form action = "main.jsp" method = "GET">
      First Name: <input type = "text" name = "first_name">
      <br />
```

```
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
   </form>
      </body>
</html>
```

POST Method Example Using Form

```
<html>
  <head>
    <title>Using GET and POST Method to Read Form Data</title>
  </head>
    <body>
    <center>
    <h1>Using POST Method to Read Form Data</h1>
        <ul>
      <li><p><b>First Name:</b>
        <%= request.getParameter("first_name")%>
      </p></li>
      <li><p><b>Last  Name:</b>
        <%= request.getParameter("last_name")%>
      </p></li>
   </ul>
   </body>
</html>
```

Following is the content of the **Hello.htm** file −

```
<html>
  <body>
        <form action = "main.jsp" method = "POST">
      First Name: <input type = "text" name = "first_name">
```

```
<br />
Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
</body>
</html>
```

Passing Checkbox Data to JSP Program

- Checkboxes are used when more than one option is required to be selected.

Following is an example **HTML code, CheckBox.htm**, for a form with two checkboxes.

```
<html>
  <body>
        <form action = "main.jsp" method = "POST" target = "_blank">
    <input type = "checkbox" name = "maths" checked = "checked" /> Maths
    <input type = "checkbox" name = "physics"  /> Physics
    <input type = "checkbox" name = "chemistry" checked = "checked" /> Chemistry
    <input type = "submit" value = "Select Subject" />
  </form>
    </body>
</html>
```

The above code will generate the following result −

☑ Maths ☐ Physics ☑ Chemistry

Following is main.jsp JSP program to handle the input given by the web browser for the checkbox button.

```
<html>
  <head>
    <title>Reading Checkbox Data</title>
  </head>
    <body>
    <h1>Reading Checkbox Data</h1>
```

```
        <ul>
      <li><p><b>Maths Flag:</b>
        <%= request.getParameter("maths")%>
      </p></li>
      <li><p><b>Physics Flag:</b>
        <%= request.getParameter("physics")%>
      </p></li>
      <li><p><b>Chemistry Flag:</b>
        <%= request.getParameter("chemistry")%>
      </p></li>
    </ul>
    </body>
</html>
```

Reading All Form Parameters

- **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.
- Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using the ***hasMoreElements()*** method to determine when to stop and using the ***nextElement()*** method to get each parameter name.

```
<%@ page import = "java.io.*,java.util.*" %>
<html>
  <head>
    <title>HTTP Header Request Example</title>
  </head>
  <body>
    <center>
      <h2>HTTP Header Request Example</h2>
      <table width = "100%" border = "1" align = "center">
```

```
        <tr bgcolor = "#949494">
          <th>Param Name</th>
          <th>Param Value(s)</th>
        </tr>
        <%
          Enumeration paramNames = request.getParameterNames();
          while(paramNames.hasMoreElements()) {
            String paramName = (String)paramNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n");
            String paramValue = request.getHeader(paramName);
            out.println("<td> " + paramValue + "</td></tr>\n");
          }
        %>
      </table>
    </center>
          </body>
</html>
```

Following is the content of the **Hello.htm** −

```
<html>
  <body>
        <form action = "main.jsp" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics"  /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
      <input type = "submit" value = "Select Subject" />
    </form>
    </body>
</html>
```