BACK END DEVELOPMENT

MODULE=03

SERVLETS

- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- **Servlet** technology is robust and scalable because of java language.
- What is a Servlet?
- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

CGI (COMMON GATEWAY INTERFACE)

- A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.
- CGI (Common Gateway Interface)
- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

CGI

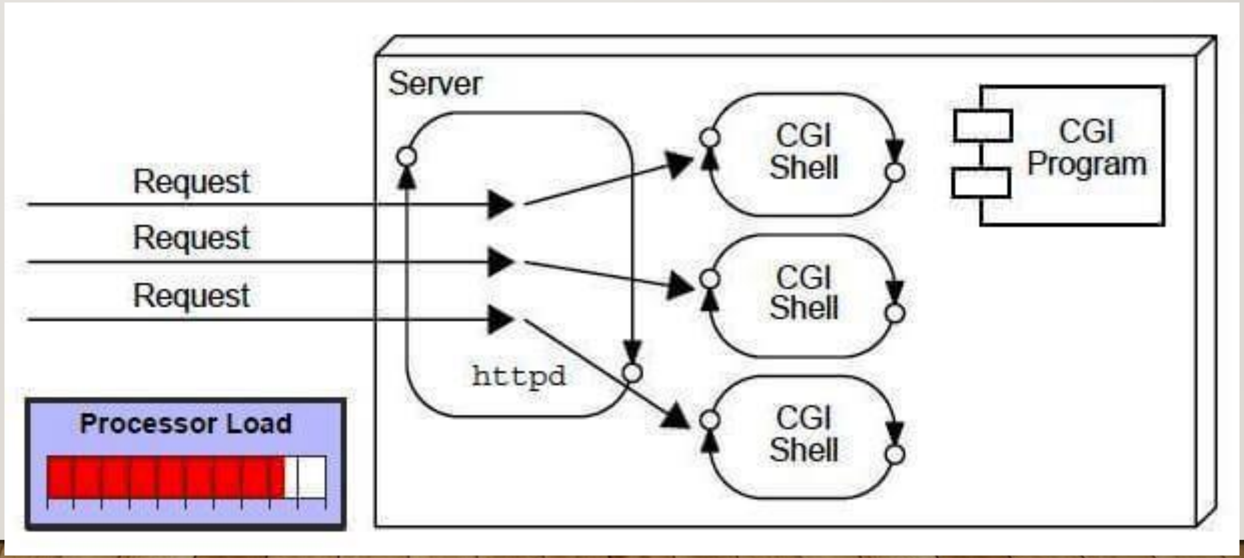
Disadvantages of CGI

There are many problems in CGI technology:

1.If the number of clients increases, it takes more time for sending the response.

2.For each request, it starts a process, and the web server is limited to start processes.

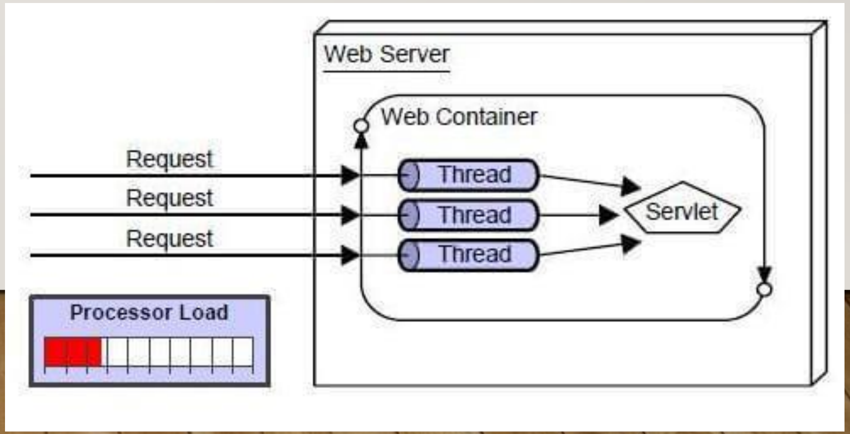
3.It uses platform dependent language e.g. C, C++, perl.



ADVANTAGES OF SERVLET

- There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.



SERVLETS ARCHITECTURE

•**Servlets Tasks:**

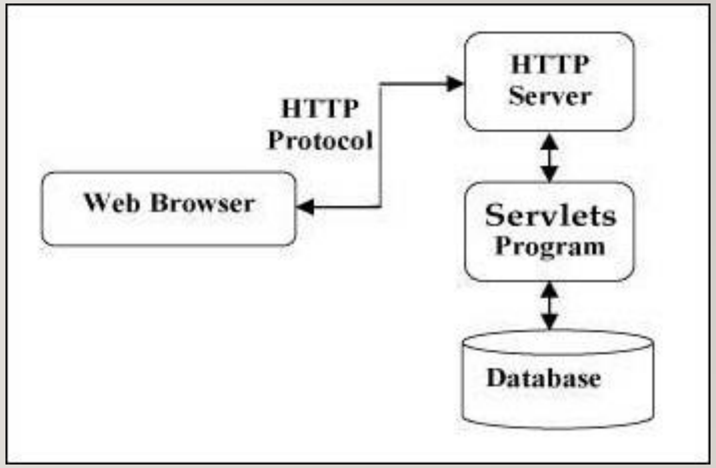
•Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.

•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.

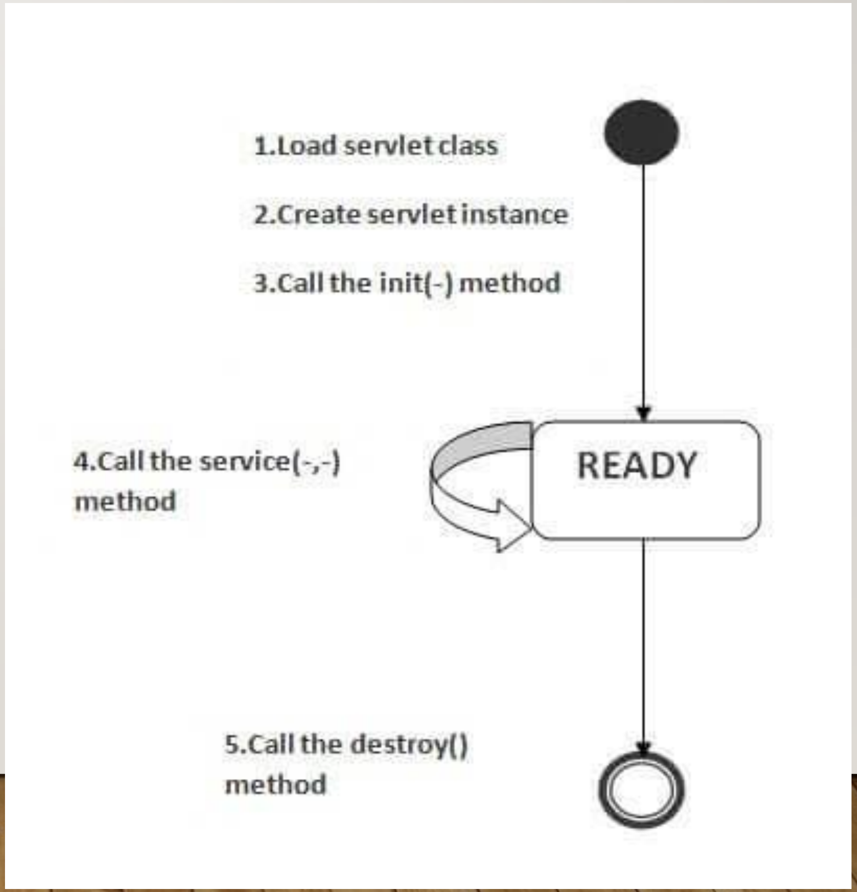
•Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.



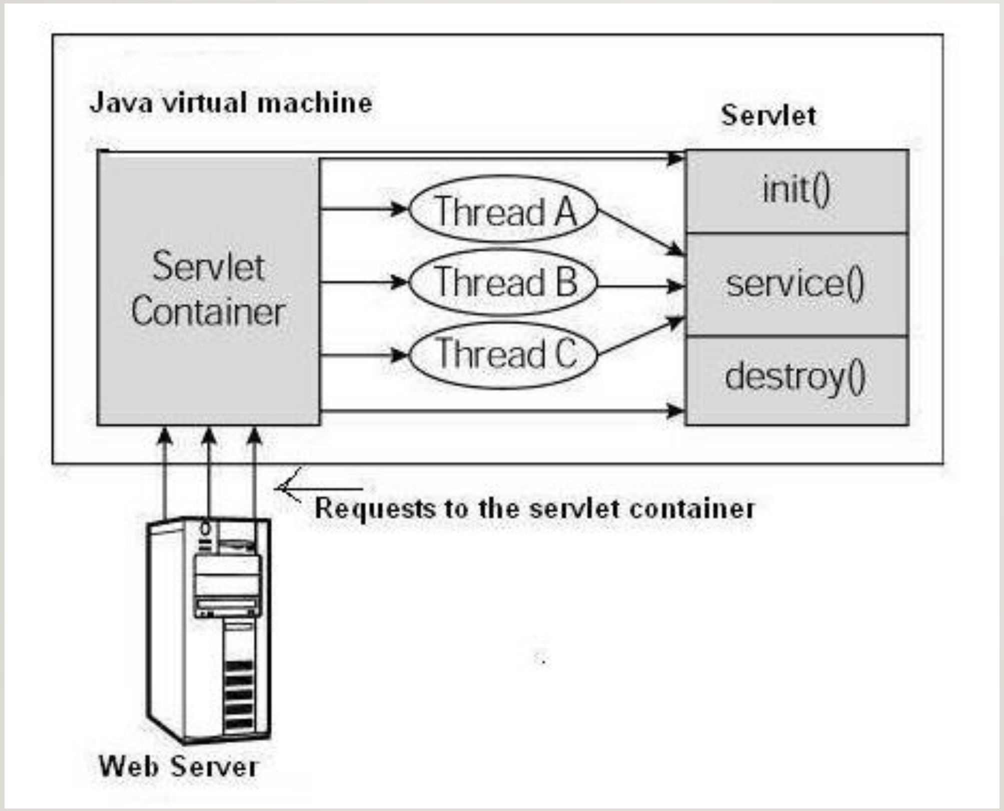
LIFE CYCLE OF A SERVLET

- The web container maintains the life cycle of a servlet instance.

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

SERVLET LIFECYCLE



LIFE CYCLE OF A SERVLET

1) Servlet class is loaded

The class loader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once

- 5) destroy method is invoked
- The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

SERVLET API

- The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

SERVLET API

- There are many interfaces in javax.servlet package. They are as follows:

1. Servlet - This interface describes and connects all the methods that a Servlet must implement.
2. ServletRequest - used to retrieve the information data from the user
3. ServletResponse - used to estimate the response to the end-user on the system
4. RequestDispatcher
5. ServletConfig -used for providing the information data to the servlet classes external to explicitly.
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

HTTPSERVLET CLASS
GET AND POST METHOD

- The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.
- **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.

1. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
2. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

EXAMPLE

- The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

- There are given 6 steps to create a **servlet example**

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

```
import javax.servlet.http.*;

import javax.servlet.*;

import java.io.*;

public class DemoServlet extends HttpServlet{

public void doGet(HttpServletRequest req,HttpServletRespon
se res)

throws ServletException,IOException

{

res.setContentType("text/html");//setting the content type

PrintWriter pw=res.getWriter();//get the stream to write the
data

//writing html in the stream

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

pw.close();//closing the stream

}}
```

SERVLET CODE TO PERFORM ADDITION OF TWO NUMBERS

- package servletjsp;
- import javax.servlet.http.HttpServlet;
- import javax.servlet.http.HttpServletRequest;
- import javax.servlet.http.HttpServletResponse;
- public class AddServlet extends HttpServlet
- {
-     public void service(HttpServletRequest req,HttpServletResponse res)
-     {
-         int i=Integer.parseInt(req.getParameter("num1"));
-         int j=Integer.parseInt(req.getParameter("num2"));
-         int k=i+j;
-         System.out.println("the result is "+ k);} }
- **JAVA file**

HTML FILE

```
• <!DOCTYPE html>
• <html>
•     <body>
•         <form action="add">
•             Enter 1st number: <input type="text"  name=num1><br>
•             Enter 2nd number: <input type="text"  name=num2><br>
•             <input type="submit">
•         </form>
•     </body>
• </html>
```

WEB.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" id="WebApp_ID"
        version="3.1">
    <servlet>
<servlet-name>abc</servlet-name>
        <servlet-class>servletjsp.AddServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/add</url-pattern>
    </servlet-mapping>
</web-app>
```

WEB.XML

The web server uses this configuration **to identify the servlet to handle a given request and call the class method that corresponds to the request method**.

```
<web-app> <servlet>
<servlet-name>abc</servlet-name>
        <servlet-class>servletjsp.AddServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/add</url-pattern>
    </servlet-mapping>
</web-app>
```

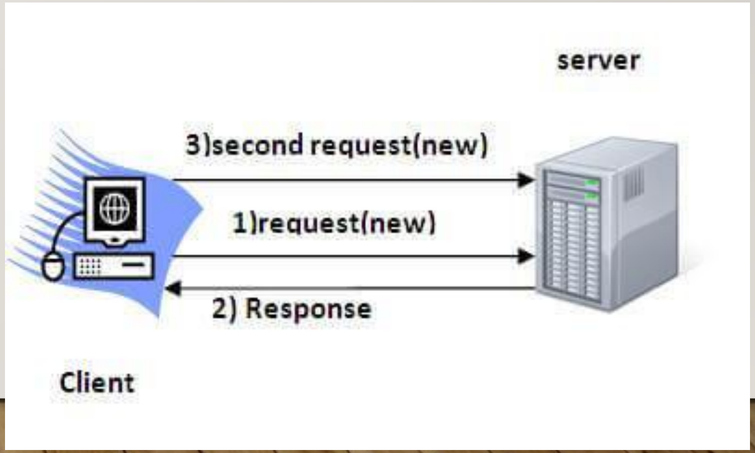
SESSION TRACKING

- Session
- HTTP Protocol

SESSION TRACKING/HANDLING IN SERVLETS

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new

request.



- Why use Session Tracking?
- **To recognize the user** It is used to recognize the particular user.
- Session Tracking Techniques
- There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

HIDDEN FORM FIELD

- The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user’s view.

```
<input type = 'hidden' name = 'session' value = '12345' >
```

- Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

- Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

URL REWRITING

- With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.
- **url?name1=value1&name2=value2&??**
- Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

- Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

HTTPSESSION

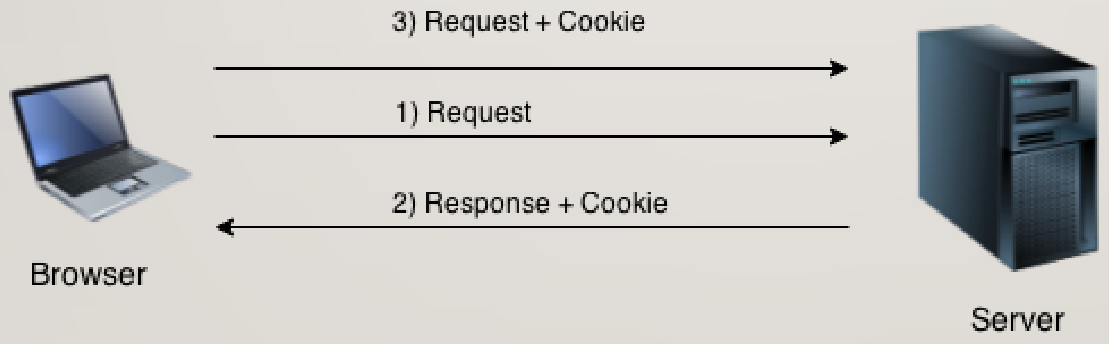
- A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.
-
```
HttpSession session = request.getSession( );
Session.setAttribute("username", "password");
```

COOKIES IN SERVLET

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

HOW COOKIE WORKS

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



TYPES OF COOKIE

- There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

- Non-persistent cookie
- It is **valid for single session** only. It is removed each time when user closes the browser.
- Persistent cookie
- It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

- Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

- Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

- Cookie class
- **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

HOW TO CREATE COOKIE?

```
Cookie ck=new Cookie("user",“Ajay");//creating cookie object

response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

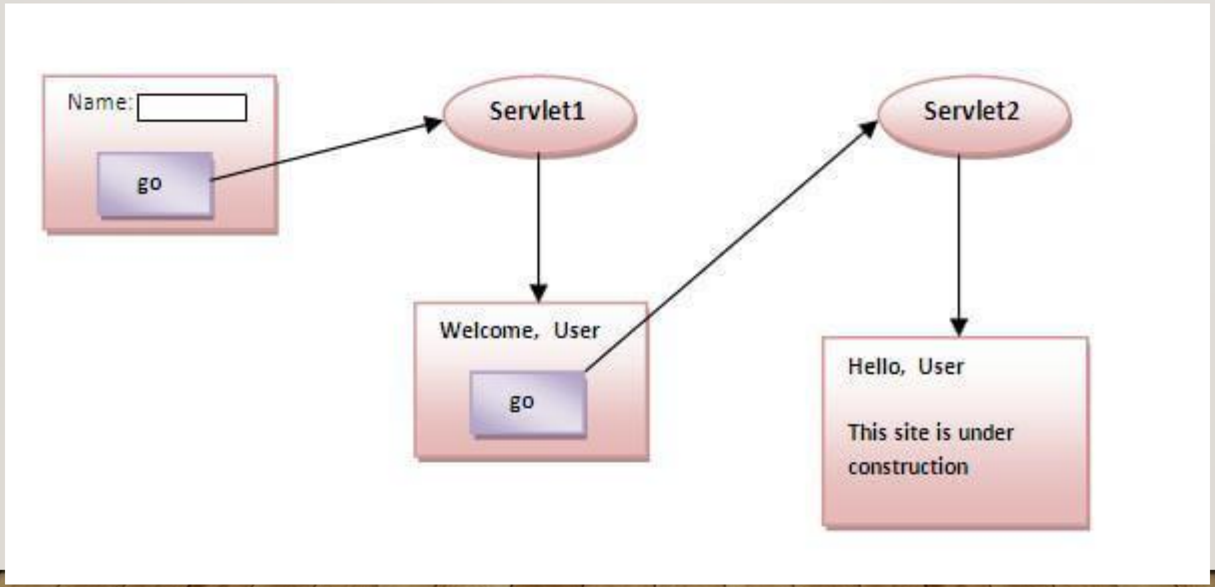
```
Cookie ck=new Cookie("user","");//deleting value of cookie

ck.setMaxAge(0);//changing the maximum age to 0 seconds

response.addCookie(ck);//adding cookie in the response
```

HOW TO GET COOKIES?

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```



INSTALLING AND CONFIGURING APACHE TOMCAT WEB SERVER

- Step 1: Install JDK
- Step 2: Set up the class path variable of JDK.
- Step 3:
- Download Apache Tomcat from following link- **Apache Tomcat 8** , **Tomcat 9** and Tomcat 10
- https://dlcdn.apache.org/tomcat/tomcat-10/v10.0.11/bin/apache-tomcat-10.0.11.exe
- Step 4:Next unpack the zip folder and start executing.
- Step 5:After installation check the path in browser http://localhost:8080

TO START AND STOP SERVER

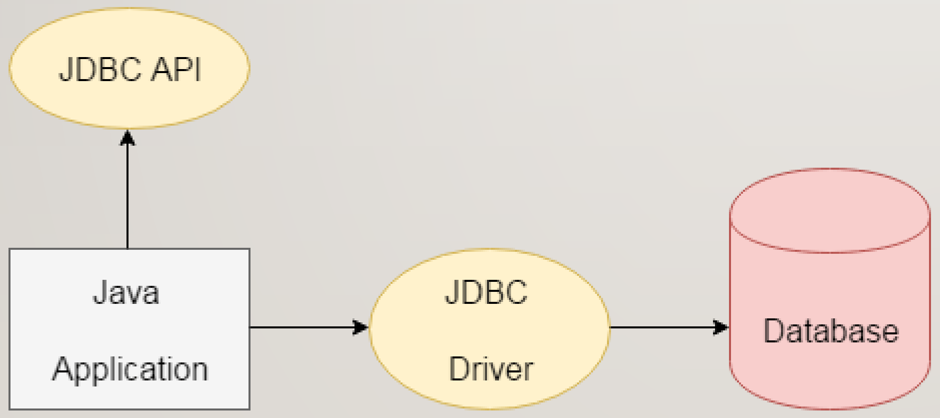
- After successful installation, go to BIN folder directly under Tomcat folder. You will find two batch files with names **startup.bat** and **shutdown.bat**. You can create shortcuts of these batch files on the desktop or inside Startup Menu for easily starting and stopping Tomcat server whenever required.
- Simply use Command Prompt to start and stop Tomcat server in other way.
- C:\Tomcat8\bin>startup.bat
- C:\Tomcat8\bin>shutdown.bat

DATABASE CONNECTIVITY -JDBC

- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:
- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

JDBC

- We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



- The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API.
- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

JDBC DRIVER

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC BRIDGE DRIVER

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

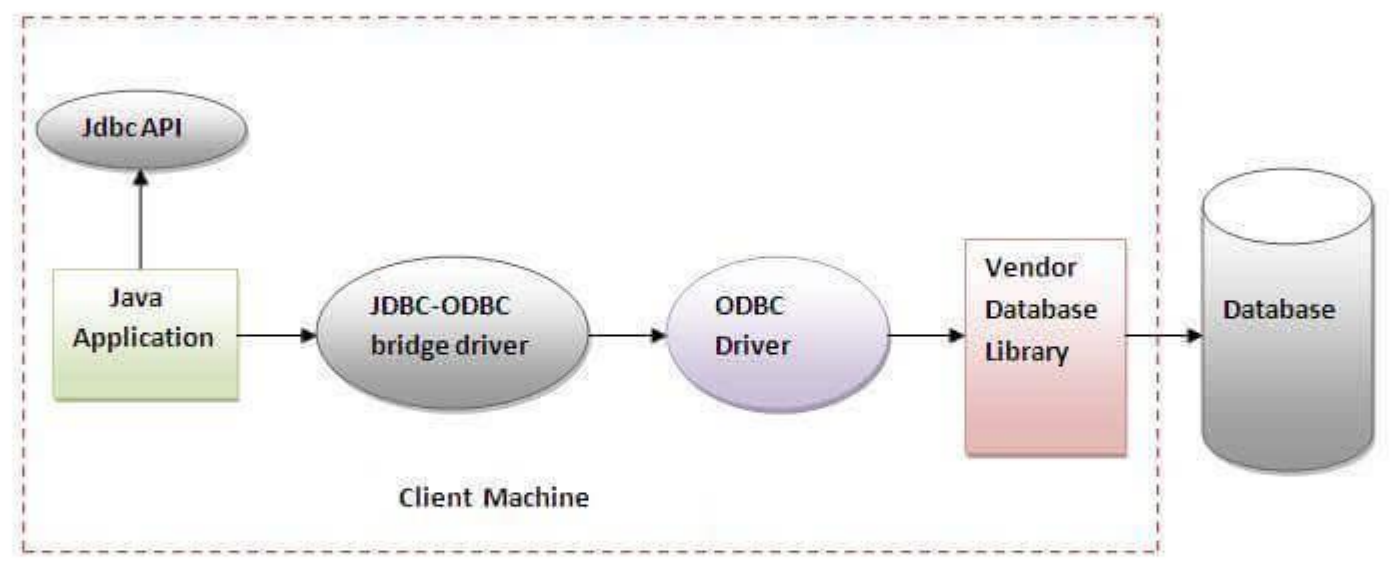


Figure- JDBC-ODBC Bridge Driver

- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.
- Advantages:
- easy to use.
- can be easily connected to any database.
- Disadvantages:
- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) NATIVE-API DRIVER

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

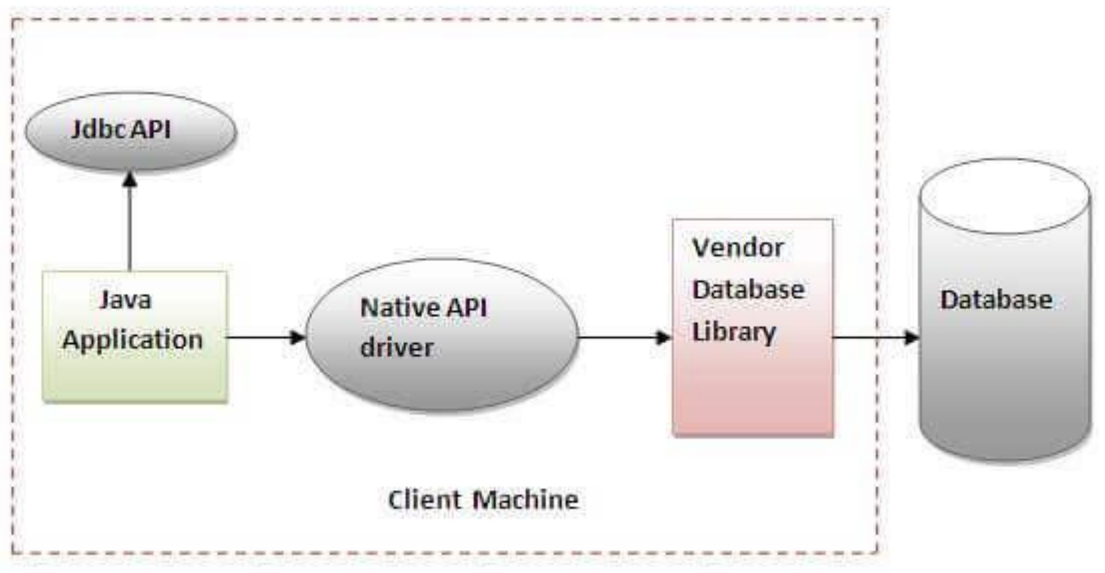


Figure- Native API Driver

- Advantage:
- performance upgraded than JDBC-ODBC bridge driver.
- Disadvantage:
- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) NETWORK PROTOCOL DRIVER

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

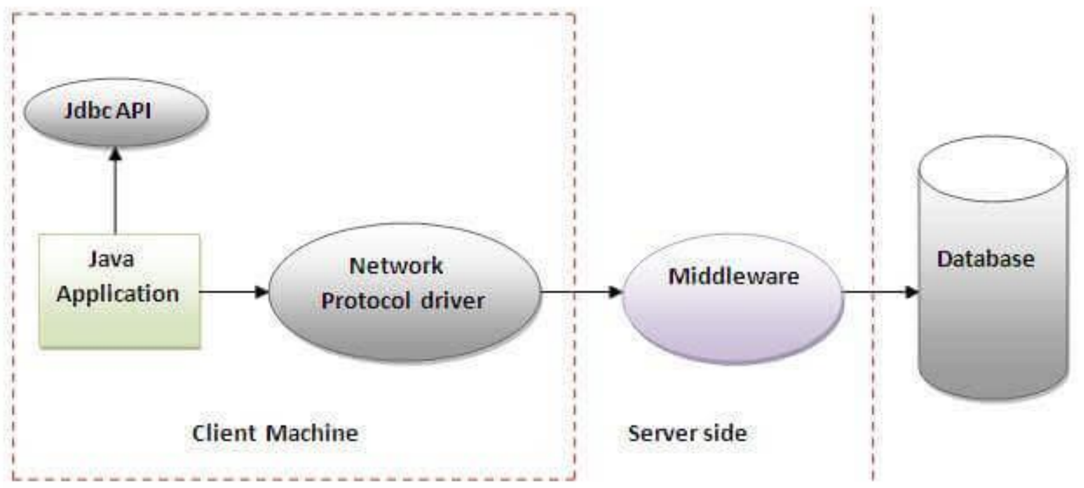


Figure- Network Protocol Driver

4) THIN DRIVER

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.
- Disadvantage:
- Drivers depend on the Database.

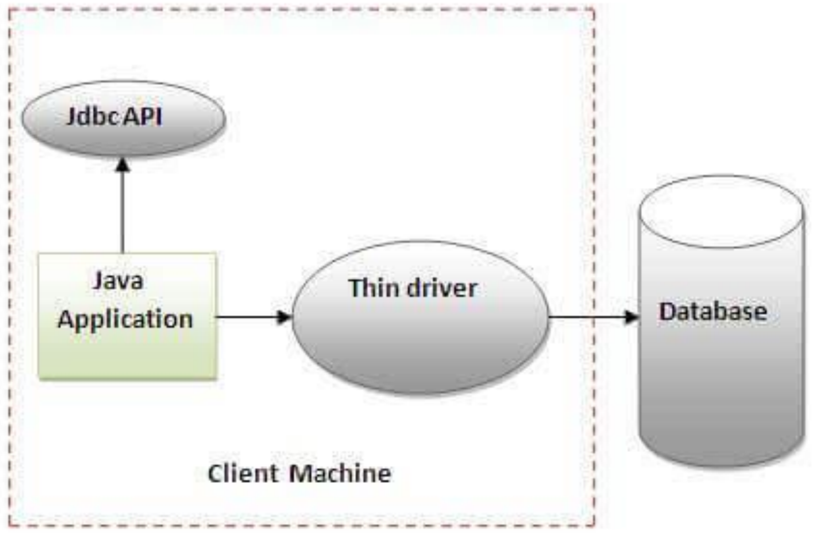


Figure- Thin Driver

JAVA DATABASE CONNECTIVITY WITH 5 STEPS

- 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

- **1) Register 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.

1. 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

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

Example to establish connection with the Oracle database

```
1. 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

1. 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

1. **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

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

- Example to close connection

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

JAVA DATABASE CONNECTIVITY WITH MYSQL

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/emp**
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.

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

EXAMPLE TO CONNECT JAVA APPLICATION WITH MYSQL DATABASE

```
import java.sql.*;

class MysqlCon{

public static void main(String args[]){

try{

Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/emp","root","root");

//here emp is database name, root is username and passwor
d
```

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getSt
ring(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

JSP (JAVA SERVER PAGES)

- **JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

ADVANTAGES OF JSP OVER SERVLET

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

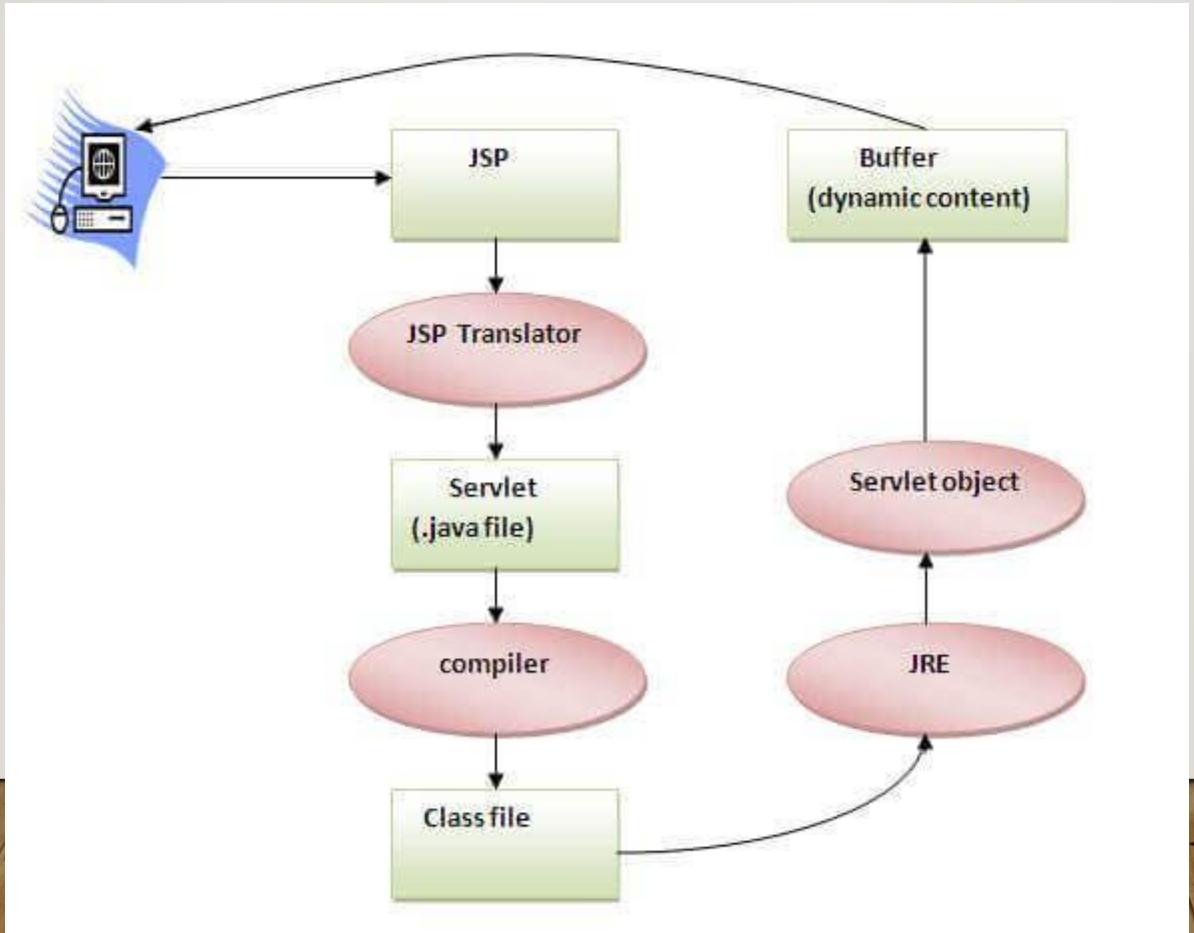
3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

- JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.



THE LIFECYCLE OF A JSP PAGE

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

CREATING A SIMPLE JSP PAGE

- To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

JSTL (JSP STANDARD TAG LIBRARY)

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplify the JSP.
2. **Code Reusability** We can use the JSTL tags on various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL MAINLY PROVIDES FIVE TYPES OF TAGS:

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The URL for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**.
SQL tags	The JSTL SQL tags provide SQL support. The URL for the SQL tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

JSTL CORE <C:OUT> TAG

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${'Welcome to JSTL'}"/>
</body>
</html>
```

CREATING HTML FORMS BY EMBEDDING JSP CODE

- JSP handles GET and POST 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

```
<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>
```

- The following URL will pass two values to HelloForm program using the GET method.
- **http://localhost:8080/main.jsp?first_name=AJAY &last_name=ALI**
- **RESULT**
  - Using GET Method to Read Form Data
- **First Name**: AJAY
- **Last Name**: ALI

GET METHOD EXAMPLE USING FORM

```
<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>
```

PASSING CHECKBOX DATA TO JSP PROGRAM

HTML Code

```
<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>
```

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>
```