UNIT – IV

WEB SERVERS & SERVLETS

JSP APPLICATION DEVELOPMENT

Introduction:

Java Servlet is a server side program that is called by the user interface or another J2EE in component and contains the business logic to process a request. The Java Servlet can call other components such as EJB that handle processing.

J2EE applications use a light weight client, typically a browser that contains little (or) no processing logic. Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate Web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multi Purpose Internet Mail Extension (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language source code of a web page has a MIME type of text/HTML.

Now consider the dynamic content. Assuming that an online store uses a database to store information about business, This would include items for sale, prices, availability, orders and so on. It wishes to make this information accessible to customer via web pages. The contents of those web pages must be dynamically generated in order to reflect the largest information in the database.

In the early days of web, a server could be dynamic construct a page by creating a separate process to handle each client request. The process would open connecting to one or more database in order to obtain the necessary information. It communicated within the web server buyer and interface known as Common Gateway Interface(CGI). CGI allowed the separate process to read data from the HTTP request write the data to the HTTP response. A variety of different languages were used to build CGI programs. These include C, C++, Perl etc. However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition, CGI programs were not platform - independent. Therefore, other techniques were introduced, Among these are "Servlets".

Benefits of Servlets over CGI:

Service offer several advantages in comparison with CGI:

- The 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. A number of web servers from different vendors offer the servlet API. Programs developed for this API can be moved to any of these environment without recompilation.
- The Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases (or) other software via the sources and RMI mechanisms.

The Life Cycle of a Servlet:

1. init()
2. service()
3. destroy()

They are implemented by every servlet and invoked at specific times by the server. Assuming that a user enters a URL to a web browser. The browser then generates an HTTP request for this ?URL. This request is then sent to the appropriate server. This HTTP request is received by the web server. The server maps this request to a particular sevlet. The servlet is dynamically retrived and loaded into the address space of the server. The server invokes “init()” method of the servlet. This method is invoked only when the servlet is first loaded in to the memory. It is possible to pass the initialization parameters to the servlet. So, it may configure itself.

The server invokes the “service()” method of the servlet. This method is called to process the HTTP request. It is possible to the servlet to read the data that has been provided in the HTTP request. It may also formulate an HTTP response for the-client.

The servlet remaining in the address space of the server and is available to process any other HTTP request received from clients. The service() method is called for each HTTP request.

The server may decide to unload the servlet from its memory. The algorithms by which this determination is made or specific to each server. The server calls “destroy()” to relinquish any resources as file handles that are allocated for the servlet. Important data may be stored to a persistent

store. The memory allocated for the servlet and its objects can then be garbage collected.

A Simple Servlet:

To create and test a simple servlet we have to follow the following steps:

1. Create a project folder hierarchy as follows:

P1

|

WEB-INF

|

Classes

2. Open notepad and write servlet class as follows:

Source Code:

```
import java.io.*;
import javax.servlet.*;
public class MyServlet extends GenericServlet{
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException{
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<html><head><title>This is My Servlet </title>");
pw.println("</head><body><h1>Hi This message came from my server</h1>");
pw.println("</body></html>");
plw.close();
}
}
```

Above code imports "javax.servlet" package. This package contains the classes and interfaces required to build servlets. The program defines "MyServlet" as a subclass of "GenericServlet". The GenericServlet class provides functionality that makes it is easy to handle requests and responses.

Inside MyServlet, "service()" is overridden. This method handles requests from a client. The first argument "ServletRequest" object that enables the servlet to read data that is provided via client requests. The second argument is a "ServletResponse" object that enables the servlet to formulate a

response for the client.

The call to “setContentType()” method establishes the “MIME” type of the HTTP response. In this the MIME type is text/html that indicates that the browser should interpret the content as HTML data.

Now, the getWriter() method obtains the “PrintWriter()” anything written to this stream is sent to the client as part of a HTTP response. The println() method is used to write some simple HTML.

3. Save the above file with the name MyServlet.java.
4. Set the classpath to servlet-api.jar file by the following command

   setClassPath= %CLASSPATH%;D:/tomcat6.0/lib/servlet-api.jar ;

   (or)

setClassPath= %CLASSPATH%;D:/tomcat5.5/commom/lib/servlet-api.jar;

1. Compile the servlet by the following command

   javac MyServlet.java

2. Copy te MyServlet.class file in the P1/WEB-INF/classes folder
3. Open notepad and write the description of te servlet as shown below:

```
<web-app>
<servlet>
<servlet-name>abc</servlet-name>
<sservlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>abc</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>
</web-app>
```

4. Save this file with the name web.xml in P1/WEB-INF folder.
5. Copy the project folder P! in d:/tomcat6.0/webapps folder.
6. Start tomcat server. The application P1 is deployed and running.
7. Open web browser window.
8. Type the following in address field of the browser window:

http://localhost:8080/P1/hello

Where P1=name of the application

Hello= url-pattern of MyServlet

9. Servlet executes one generates the response to clients as:

10. Hi, this message came from MyServlet in client's browser window.

The Servlet API:

Two packages contain the classes and interfaces that are required to build servlets. These are:

1. Javax.servlet
2. Javax.servlet.http

The above two packages constitute the servlet API. These two packages are not part of the java core packages. Instead they are standard extensions. Therefore, they are not included in the Java Software Development Kit. We must download tomcat to obtain their functionality.

The javax.servlet Package:

The javax.servlet package contains a number of interfaces and classes that establish the framework in which the servlets operate.

List of Interfaces:

Interface	Description
Servlet	Declares lifecycle methods for a servlet
ServletConfig	Allows servlet to get initialization parameters.
ServletContext	Enables servlet to events and access information about their environment
ServletRequest	Used to read data from a client request
ServletResponse	Used to write data to a client response
SingleThreadModel	Indicates that the Servlet is thread safe

1.The Servlet Interface:

All Servlets must implement the "Servlet" interface. It declares the init(), service and destroy() methods that are called by server during the life cycle of servlet. A method is also provided that allows a servlet to obtain any initialization parameters. Various methods designed by the servlet interfaces are:

Method	Description
void destroy()	Called when the servlet is unloaded
ServletConfig getServletConfig()	Returns a ServletConfig object that contains any initialization parameters
String getServletInfo()	Returns a string describing the servlet
void init(ServletConfig sc) throws ServletException	Called when the servlet is initialized. Initialization parmeters for the servlet can be obtained from sc. An unavilable Exception should be thrown if the servlets cannot be initialized.
void service(ServletRequest req, ServletResponse res) throws ServletException, IoException	Called to process a request from a client .The request from req and the response to the client can be written res. An exception is generated if a servlet (or) Io problem occurs.

The init(), servlet() and destroy() methods are the life cycle methods of the server. These are invoked by the server the getServeltConfig() method is called by the servlet to obtain imitialize paramaters. A servlet devloper overrides getServletInfo() method to provide a string useful information.

2.The ServletConfig Interface:

The servletConfig interface is implemented by the server. It allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are:

Method	Description
servletContext getServletContext()	Returns the context of the servlet.
string getInitParameter(String param)	Returns the value of the initialization parameter named param.
string getServletParameterName()	Returns an enumeration of all initialization parameter names.
string getServletName()	Returns the name of the invoking servlet.

3.The ServletContext Interface:

The servletContext interface is implemented by the server it enables servlet to obtain information

about their enviroment several methods designed by this interface are:

Methods	Description
string getiIMEType(String file)	Returns the MIME type of file
string getServletInfo()	Returns information about the server.
void log(string s)	Writes to the servlet log
void log(string s, throwable e)	Write s and stack trace for e to the servlet log

The ServletRequest Interface:

The ServletRequest Interface is implemented by the server. It enables a servlet to obtain the information a client request. The various methods are:

Method	Description
Int getContentLength()	Returns the size of the request. The value -1 is returned if the size is unavailable.
String getContentType()	Returns the type of the request. A null value is returned if the type cannot be determined.
ServletInputStream getInputStream() throws IOException	Returns a servlet that can be used to read binary data from the request. An IllegalStateException is thrown if getReader has already been invoked for this request
String getCharacterEncoding()	Returns the character encoding of the request
String getParameter(string pname)	Returns the value of the parameter named pname
Enumeration getParameterNames()	Returns an enumeration of the parameter names for this request.
String[] getParameterValues(string name)	Returns an array containing values associated with the parameter specify by name

4.The ServletResponse Interface:

The ServletResponse Interface is implemented by the server. It enables a server to formulate a response for a client. Several methods defined by this interface are:

Method	Description
String getCharacterEncoding()	Returning the character encoding for the response
ServletOutputStream getOutputStream() throws IOException	Returns a ServletOutputStream that can be used to write binary data to a response. An IlleagalStateException is thrown if getWriter() has already been invoked for this request
PrintWriter getWriter() throws IOException	Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream() has already been invoked for this request
Void SetContentLength(int size)	Sets the content length for the response to size
Void SetContentType(string type)	Sets the content type for the response to type.

The various core classes that are provided in the javax.servlet Package are:

Class	Description
GenericServlet	Implement the servlet and servletConfig Interfaces.
ServletInputStream	Provides an input stram for reading request from a client.
ServletOutputStream	Provides an outputStream for writing responses to a client.
ServletException	Indicates a servlet error has occures.
Unavailable Exception	Indicates a servlet in unavailable.

The SingleThread Model Interface:

This interface is used to indicate that only a single thread will execute the servlet() metod of a servlet at a given time. It defines no constants and declares no methods.

If a servlet implements this interface, the server has two options:

1. It can create servlet instances of the servlet when a client request arrives it is dent to an available

instance of the servlet.

2. It can synchronize access to the servlet.

Javax.servlet classes:

1. The GenericServlet Class:

The GenericServlet class provides implementations of the basic life cycle methods of a servlet and is typically subclass by servlet developers. GenericServlet implements "servlet" and ServletConfig interfaces. In addition, a method to append a String to the server log file is available.the signatures of this method are shown as:

- Void log(String s)
- Void log(String s)

Here, s is the strong to be appended to the log , and e is the exception that occured.

2. The ServletInputStream class:

The ServletInputStream class extends InputStream. It is implemented by the server and provides an input stream that a servlet developer can use to read the data from a client request .It defines the default constructor. In addition, a method is provided to read bytes from the stream. Its syntax is:

int readLine(byte[] buffer,int offset, int size) throws IOException

Here, buffer is the array into which size bytes are placed starting at offfset. The method returns the actual number of bytes read (or) -1 if an end-of-stram condition is encountered.

1. The ServletOutputStream class:

The ServletOutputStream class extends OutputSream. It is implemented by the server and provides on OutputStream, that a developer can use to write data to a client response. A default constructor is defined. It also defines print() and println() methods, which output data to the stream.

2. The ServletException class:

Javax.Servlet defines two exceptions. The first is ServletException, which indicates a servlet problem as occurred. The second is UnavailableException, which extends ServletException and indicates tha a servlet is unavailable.

The javax.Servlet.http package:

The javax.Servlet.http package contains a number of interfaces and classes that are commonly used by servlet developers. Various interfaces defined by the javax.Servlet.http are:

Interface	Description
HttpServletRequest	Enables Servlet to read data from a HTTP Request
HttpServletResponse	Enables Servlet to write data to an Http Response
HttpSession	Allows session data to be read or written
HttpSessionBindingListener	Informs an object that is bound to (or) unbound from a session.

1.The HttpServerRequest Interface:

The HttpServletRequest interface is implemented by the server and it enables a servlet to obtain information about a client request. Several methods defined by this interface are:

Method	Description
Cookie[] getCookies()	Returns an array of cookies in this request
Long getDateHeader(String field)	Returns the value of the date header field named field
String getHeader(string field)	Returns the value of the header field named field
Enumeration getHeaderNames()	Returns an enumeration of the header names
String getQueryString()	Returns any query string in the URL
StringBuffer getRequestURL()	Returns the URL
HttpSession getSession()	Returns session for this request. If a session does not exist, one is created and then returned.

2.The HttpServletResponse interface:

The HttpServletResponse interface is implemented by the server. It enables a servlet to formulate an Http Response to a client. Several constraints are defined. These correspond to the different status codes that can be assigned to an Http Response. Several methods of this interface are:

Methods and it's Description:

- void addCookie(Cookie cookie) : adds cookie to the HTTPresponse
- String encodeURL(String url) : determines if the session ID must be encoded in the URL identified as URL. If so it returns the modified version of URL. Otherwise returns URL. All URL's generated by a servlet should be processed by this method
- void sendError(int c) throws IOExecption : sends the error code c to the client.
- void sendError(int c, String S) throws IOExecption : sends the error code c and message S to the client.

- void sendRedirect(String url) throws IOExecption : redirects the client to url.
- void sendHeader(String field, String value) : adds field to the header with value equal to value.
- void setState(int code) : sets the states code for this response to code.

3.HttpSession interface :

The HttpSession interface is implemented by the server and it enables a servlet to read and write the state information that is associate with an HttpSession. Various methods defined by this interface are

Methods :

Object getAttribute(String attr) : returns the value associated with the name passed in attr. Returns null if attr is not found.

Enumeration getAttributeNames() : returns an enumeration of the attribute names associated with the session.

long getCreationTime() : returns the time (in milliseconds) when this session was created.

long getLastAccessedTime() : returns the time (int milliseconds) when the user last made a request for this session.

void invalidate() : invalidates this session and removes it from the context.

3. HttpSessionBindingListener interface :

The HttpSessionBindingListener is an interface extending from base interface java.util.EventListener interface. This interface will notify an object when it is bound to or unbound from a session.

Methods :

Void valueBound(HttpSessionBindingEvent e)

Void valueUnbound(HttpSessionBindingEvent e)

Here, e is the event object that describes the binding various classes that are provided by javax.servlet.httpPacakge are

Class & Their Description:

Cookie : allows state information to be stored on a client machine

HttpServlet : provides methods to handle Http requests and responses

HttpSessionEvent : encapsulates a session changed event

HttpSessionBindingEvent : indicates when a listener is bound or unbound from a session value or

that a session attributes

1) The Cookie Class:

The Cookie class encapsulates a cookie. A cookie contains state information sent by a servlet to web browser, and is saved by th e browser and stored on a client.

It later is sent back to server. Cookies are valuable for tracking user's activities. For example, assuming that a user visits an online store. A cookie can save the user's name, address and other information. The user does not need to enter this data each time he (or) she visits the store.

A servlet can write a cookie in to a user's machine via the addcookie() method of the HttpServletResponse interface, The data for that cookie is then included in the header of the Http response that is sent to the browser.

The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expression date of the cookie
- The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is

deleted when the current browser Session ends. Otherwise, the cookie is saved in a file on the user's machine.

The domain and path of the cookie determine when it is included in the header of the HTTP request. If the user enters a URL whose domain and path match

these values, the cookie is then supplied to web server. Otherwise it is not. There is one constructor defined for cookie. It has signature:

Cookie(string name, string value)

Here, name and value of the cookie are supplied as arguments to the constructor. Various other methods defined by cookie class are:

Method	Description
string getDomain()	Return the domain
int getMaxAge()	Return the age(in seconds)

string getName()	Return the name
string getPath()	Return the path
string getValue()	Return the value
void setDomain(string d)	Sets the domain to d
void setMaxAge(int secs)	Sets the maximum age of the cookie to secs. This is the number of seconds after which the cookie is deleted.passing -1 causes the cookie to be removed when the browser is terminuted
void setValue(string v)	sets the value to v.
void setpath(string p)	sets the path to p

2) The HttpServlet class:

The HttpServlet class extends GenericServlet, It is commonly used when developing servlets that receive and process HTTP requests.The methods of HttpServlet can be summarized as:

Method	Description
void doGet(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Performs on HTTP GET
void doPost(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Performs on HTTP POST
void service(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Called by the server, When an HTTP request arrives for the servlet. The arguments provide access to HTTP request and response

3) The HttPSessionEvent Class:

The HttpSessionEvent Class encapsulates session events.It extends EventObject and is generated when a change occurs to the session.It defines a constructor. HttpSessionEvent(HttpSession session):- Here,Session is the source of the event.

HttPSessionEvent defines one method:

HttPSession getSession():- It returns the session in which the event is occurred.

4) The HttpSessionBindingEvent Class:

The HttpSessionBindingEvent Class extends The HttpSessionEvent Class.It is generated when a listener is bound (or) unbound from a value in a HttpSession object.It is also generated when an attribute is bound (or) unbound.

The constructors defined are:

HttPSessionBindingEvent(HttPSession session String name)

HttPSessionBindingEvent(HttPSession session String name,Object val)

Here,Session is the source of the event and name is the name associated with the object that is being bound (or) unbound.If an attribute is being bound (or) unbound,its value is passed in val.

The various other methods are:

String getName():

This method obtains the name that is being bound (or) unbound.

HttpSession getSession():

Obtains the session to which the listeneris bound (or) unbound.

Object getValue():

This method obtains the value of the attribute that is being bound (or) unbound.

Reading Servlet Parameters

The ServletRequest includes methods that allow us to read the names and values of parameters that are included in a client request. In order to illustrate this consider the below example. The example contains two files PostParameter.html and PostParameterServlet.java. The PostParameter.html defines two labels and two text fields and a submit button.

PostParameters.html:

```
<html>
    <head>
        <title>Reading Servlet Parameters</title>
    </head>
    <body>
        <center>
```

```
<form name=”form1” method=”post”      action=http://localhost:8080/p1/PostParamservlet>
                    <table>
                              <tr>
                              <td><b>Employee</td>
                               <td><input type=”text” name=”ename” size=”25”       value=””></td>
                              </tr>
                              <tr>
<td><b>phone</td>
          <td><input type=”text” name=”phone” size=”25”        value=””></td>
                              </tr>
                              <tr>
                                        <td><input type=”submit” value=”submit”></td>
                                        <td><input type=”reset” value=”reset”></td>
                              </tr>
                    </table>
          </body>
</html>
```

The source code for PostParametersServlet.java can be shown as:

PostParametersServlet.java:

```
import java.io.*  ;
import javax.Servlet.*;
import java.util.*;
public class PostParametersServlet extends GenericServlet{
          public      void      Service(ServletRequest      req,      ServletResponse      res)      throws
ServletException,IOException {
          PrintWriter pw=res.getwriter();
          Enumeration e=req.getParameterNames();
          While(e.hasMoreElements()){
                              String pname=(string)e.nextElement();
```

```
pw.println(pname+"=");
                    String pvalue=req.getParameter(pname);
                    Pw.println();
                }
                pw.close();
        }
}
```

The getParameterNames() method return an enumeration of the parameter names. The parameter value is obtained through the getParameter() method. The output will be the parameter name and value are displayed to the client.

Place the html file in P1 folder and .class file in classes folder and web.xml in WEB-INF folder and follow the following steps:

1. Start Tomcat
2. Display the web page in the browser
3. Enter employee name and phone number in the text fields
4. Submit the web page

The browser will display the response that is dynamically generated by the servlets.

Reading Initialization Parameters:

The Initialization Parameters are supplied by server developer in the development descriptor i.e web.xml. The Initialization Parameters are given to a server as follows:

```
<init-parameter>
<param-name>name</param-name>
<param-value>value</param-value>
</init-parameter>
```

The servlet container reads the parameters and supplies them to the servlet using servletConf8g object. We call the getParameter() method on the ServletConfig object in the Servlet to receive the parameter values .

For ex: init() method can be implemented as follows.

```
public void init(ServletConfig con) throws ServletException
{
        String value=con.getInitParameter("name");
```

}

Whatever name is given to the parameter in the deployment descriptor file that name should be supplied to the getInitParameter() .

It receives the value specified the deployment descriptor file as as string object .

We can supply any number of Initialization Parameters to the Servlet.

Installing the Java Software development kit(JSDK):

1.Download the Java development kit from the sun's website https://java.sun.com/

2.Double click on the downloaded jdk installer to start JDK and now we could see the window as license agreement. Change the 4adio button option for selecting accepting terms and conditions.

3.Now just click on the next button then cases for the description folder where we go to install the Java. By default it 8s installed in c drive

C:/Program Files/java/jdk1.5.0_01

4.Now click the next button and the installation progresses that will take few minutes.

After the installation of Java we need to set the environment variables. The steps are:

1. Go to desktop find MyComputer icon and right click on it
2. From the pop up menu select properties option
3. Click on the advanced which appears on the top right corner.
4. Click on environment variables button and we could see a window which contains the user variables and system variables.
5. Now click on the new variable button in the user variable option to create the PATH variable

Variable Name: PATH

Variable Value:C:\Program files\Java\jdk1.5.0.

Then click on OK button.

6. Now again click on the new variable button in the user variable option to set the JAVA_HOME Variable as:

Variable Name: JAVA_HOME

Variable Value:C:\Programfiles\Java\jdk1.5.0

7. Click on ok button.

Handling HTTP Requests and Responses:

The HttpServlet class provides specialized methods that handles the various types of Http requests. A servlet developer typically overrides one of these methods. However, the GET and POST methods are commonly used when handling from input.

Handling HTTP GET Requests and Responses: In GET method the servlet container must write the headers before committing the response, because in Http the header must be sent before the

response body. If the response is incorrectly formatted, do Get returns an Http “Bad Request” message.

Example:

Here we will develop a servlet that handles an HTTP GET request. The Servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined and a servlet is defined.

ColorGet.html:

```
<html>
<head>
        <title>Handling GET Request</title>
</head>
<body>
<center>
<form name=”form1” method=”GET” action= http://localhost:8080/p1/ColorGetServlet>
<b>Color:
<select name=”Color” size=”1”>
        <option value=”red”>Red</option>
          <option value=”Green”>Green</option>
          <option value=”Blue”>Blue</option>
</select><br><br>
<input type=”submit” value=”submit”>
</form>
</center>
</body>
</html>
```

The above html defines a select element and a submit button .The action of the form tag specifies a URL that identifies a servlet to process the HTTP GET request.

ColorGetServlet.java:

```
Import java.io.*;
Import javax.servlet.*;
```

```
Import javax.servlet.http.*;
Public class ColorGetServlet extends HttpServlet{
        Public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException, IoException{
        String color=req.getParameter("color");
        Res.setContentType("text/html");
        PrintWriter pw= res.getWriter();
        Pw.println("<b> The Selected color is:");
        Pw.println(color);
        Pw.close();
}
}
```

Compile the servlet and perform these steps to test this example:

1.Start Tomcat.

2.Display web page in the browser.

3.Select a color.

4.Submit the web page.

After Completing these steps ,the browser will display the response that is dynamically generated by the servlet. The Parameters for an HTTP GET request are included as part of URL. That is sent to the web server. For example, assuming that the user selects red option and submits the form. The URL sent from the browser to the server is : http://localhost:8080/p1/ColorGetServlet?color=Red. The Characters to the right of the question mark are known as "Query String" .

Handling HTTP POST Request and Responses:

The HTTP POST method allows the client to send data of unlimited length to the webserver at a time .The servlet container must write the headers before committing the responses, because in HTTP the header must be sent before the response body. If the HTTP POST Request is incorrectly formatted,doPost returns an HTTP "Bad Request" message.

Example:

Here we will devlop a servlet that handles an HTTP POST request. The Servlet is invoked when a form on a web page is submitted. The example contains two files. A webpage is defined and a servlet

is defined.

ColorPost.html:

```
<html>
<head>
<title>Handling post req</title>
</head>
<body>
<center>
 <form name=”form1” method=”POST” action=http://localhost:8080/p1/ColorPostServlet>
<b>Color:
<select name=”color” size=”1”>
<option value=”red”> Red</option>
<option value=”Blue”>Blue</option>
</select><br><br>
<input type=”submit” value=”submit”>
</form>
</center>
</body>
</html>
```

ColorPostServlet.java:

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Public class ColorPostServlet extends HttpServlet{
Public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException{
String color=req.getParameter(“color”);
Res.setContentType(“text/html”);
PrintWriter pw=res.getWriter();
pw.println(“<b> the selected color is:”);
```

```
pw.println(color);
pw.close();
}
}
```

Parameters for an HTTP POST request are not included as part of URL that is sent to web server URL sent will be: http://localhost:8080/p1/ColorPostServlet.

Handling Cookies:

AddCookie.html:-

```
<html>
    <head>
        <title> Cookies demo </title>
    </head>
    <body>
        <center>
<form name="form1" method="Post"
action="http://localhost:8080/P1/AddCookieServlet">
        <b>Enter a value for my cookie:
        <input type="text" name="data" size=25 value=" ">
        <br><input type=submit value="submit">
        </form>
     </center>
  </body>
</html>
```

The above page contains a text field in which a value can be entered. There is also a submit button on the page when this button is pressed, the value in the text field is sent to AddCookieServlet via an HTTP POST request.

AddCookieServlet.Java:-

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
```

```
public class AddCookieServlet extends HttpServlet
{
    Public void doPost (HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException
   {
         String data=req.getParameter(“data”);
          Cookie cookie=new Cookie(“MyCookie”,data);
           res.addCookie(cookie);
           PrintWriter pw=res.getWriter();
           pw.println(“<br>MyCookie has been sent to”);
           pw.println(data);
           pw.close();
   }
}
```

Above code gets the value of parameter name data. It then creates cookie object that has the name “MyCookie” and contains the value of the “data” parameter. The cookie is then added to the header of the HTTP response via the addCookie() method. A feedback message is then written to the browser.

GetCookiesServlet.java:-

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
public class GetCookieServlet extends HttpServlet
{
    Public void doGet (HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException
   {
          Cookie[] cookies=req.getCookies();
          res.setContentType(“text/html”);
           PrintWriter pw=res.gerWriter();
```

```
        pw.println("<b>");
        for(int i=0;i<cookies.length;i++)
        {
              String name=cookies[i].getName();
               String value=cookies[i].getValue();
               pw.println("name="+name+";value="+value);
        }
        pw.close();
  }
}
```

Compile the servlet and perform these steps:-

1. Start Tomcat
2. Display AddCookie.html in the bottom
3. Enter a value for my cookie
4. Submit the webpage

   After completing these steps we will observe that a feedback message is displayed by the browser.

Next request the following URL via browser. http://localhost:8000/p1/GetCookiesServlet

The name and value of the cookie will be displayed in the browser.

Session Tracking:

Http is a stateless protocol. Each request is independent of the previous one. However in some applications it is necessary to save state information so that information can be collected from several interactions between a browser and a server sessions provide such a mechanism.

A session can be defined as a series of related interactions between a single client and the web server, which take place over a period of time.

A session can be created via the getSession() method of HttpServerRequest. An HttpSession object is requested this object can store a set of bindings that associate names in objects the SetAttribute(), getAttribute(), getAttributeNames() and removeAttributes() methods of HttpSession manage these bindings. It is important to note that session state is shared among all the servlets that are associated with a particular client.

Example:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DateServlet extends HttpServlet{
 public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException{
  res.setContentType(“text/html”);
  HttpSession hs=req.getSession(true);
  PrintWriter pw=res.getWriter();
  pw.print(“<b>”);
  Date d=(Date)hs.getAttribute(“date”);
  If (d!=null){
   Pw.print(“Last accessed date:”+d+”<br>”);
  }
  date = new Date();
  hs.setAttribute(“date”,d);
  pw.print(“current date:”+d);
  pw.close();
 }
```

JSP Application Development

Generating Dynamic Content:

JSP is all about generating dynamic content i.e. content that differs based on user input, time of day, the state of an external system or any other runtime conditions. JSP provides us with loss of tools for generating this content.

JSP pages should have the file extension. JSP which tells the server that the page needs to be processed by the JSP container without this extension, the server is unable to distinguish a JSP page from the other type of file and sends it unprocessed to the browser.

When working with the JSP's, we need a regular text editor such as notepad on windows. However, some Interactive Development Environment (IDE) includes a small web container that allows us to easily execute and debug page during development. There are also several webpage authoring tools that are often used when developing HTML pages that support JSP.

Example JSP that shows the current date and time:

```
<% page import =”java.util.Date” session=”false” %>
<html>
<head><title>Example JSP </title></head>
<body>
<h1>Hello, Welcome to JSP world </h1>
<% out.println(“<p><b>Hello to the JSP world”);
Out.println(“<p><b>Current Date is:”);%>
<%=new Date()%>
</body></html>
```

Using Scripting Element:

We can create Dynamic content by accessing Java programming language objects within scripting elements.

We can access a variety of objects, including enterprise beans and Java Bean components within a JSP page. JSP technology automatically makes some objects available, and we can also create and access application application-specific objects.

Implicit JSP Objects:

Implicit Objects are created by the web container and contain information related to a particular request, Page (or) application. Many of the objects are defined by the Java Server technology underlying JSP Technology.

Implicit objects are those objects which are available in JSP by default. Web container executes the JSP page as a servlet. Servlet operates in the request-response model. All objects that are associated with any servlet by default are part of this implicit object group. The following is a list of JSP implicit Objects:

1. Request
2. Response
3. Page content
4. Session
5. Application
6. Out
7. Config
8. Page
9. Exception

JSP implicit objects:

These implicit objects are created by the JSP engine during translation phase. They are being created inside service method, so we can directly use them within scriptlet without initializing ad declaring them. There are total 9 implicit objects available in JSP.

1. Out – Instance of the javax.servlet.JSP.JSPWriter
2. Request – javax.servlet.http.HttpServletRequest
3. Response – javax.servelet.http.HttpServletRequest
4. Session – javax.servlet.http.HttpSession
5. Application – javax.servlet.ServletContext
6. Exception – javax.servlet.JSP.JSPException
7. Page – java.lang.object
8. Page Context – javax.Sevlet.JSP.PageContext
9. Config – javax.servlet.Servletconfig

- **Out:**

This object is used for writing content to the client. It has several methods which can be used for properly formatting output message to the browser.

Differences between JSP writer and PrintWriter:

- ➔ Every JSP Writer is associated with 8kb of the internal buffer. Whereas PrintWriter doesn't associated with any buffer.
- ➔ The methods of the JSPWriter class are designed to throw java.io.IOException whereas methods of PrintWriter class doesn't throw any exception.
- ➔ JSPWriter is an abstract class present in the javax.servlet package.
- ➔ PrintWriter is a class defined in java.io package.

Example:

```
<html>
<body>
<% int a=10; Int b=20;
Int sum=a+b;
out.print("a value is:"+a);
out.print("b value is:"+b);
out.print("sum is:"+sum); %></body></html>
```

- **Request:**

The main purpose of request implicit object is to get data on a JSP page which has entered by the user on the previous JSP page.

Example:

```
<% string name=request.getParameter("vid");
out.print("Welcome"+name); %>
```

- **Response:**

It is basically used for modifying on dealing with the response which is being sent to the client after processing the request.

Example:

```
request.setContentType("text\html");
```

out.print(name);

- **Session:**

It is most frequently used implicit object, which is used for storing the user's data to make it available on the other JSP pages till the uses session is awake.

Example:

<html><body>

<form action="/welcome">

<input type="text" name="uname">

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

</form></body></html>

Welcome.jsp:

<html><body><% string name=request.getParameter("uname");

out.print("Welcome:"+name);

session.setAttribute("user",name);

< a href="showuserinfo.jsp">Show underable %>

</body>

</html>

Showuserinfo.jsp:

<html><body> <% string name=(string)session.getAttribute("user");

out.print("Hello"+name);

%></body></html>

- **Application:**

This is used for getting application wide initialization processing and to maintain useful data across whole JSP application.

- **Exception:**

Exception implicit object is used in exception handling for exploring the error messages. This object is only available to the JSP pages which has ErrorPage set to true.

<% @page isErrorPage="true" %>

<html><body><%= exception %></body></html>

- **Page:**

Page is implicit object used to represent the servlet instance i.e. converted servlet, generated during translation phase from a JSP page.

<% =page.getServletInfo() %>

- **Config:**

This is a Servlet configuration object and mainly used for accessing getting configuration information such as ServletContext, Servlet name,configuration parameters etc.

Conditional Processing:

Stud.html:

```
<html><body><h3>Conditional Processing </h3>
<form method=”get” action=”http://localhost:page/p1/con.jsp”>
<b>Enter Student Name:<input type=”text” name=”t1”><br>
<b>Enter Student Marks:<input type=”text” name=”t2”><br>
<input type=”submit” value=”Result”>
</form></body></html>
```

Con.jsp:

```
<html><body>
<%! String name;
String marks1;
Int marks;
%>
<b>Following is the Student Result
<%
name=request.getParameter(“t1”);
marks1=request.getParameter(“t2”);
marks=Integer.ParseInt(marks1);
if(marks>70)
out.println(“Distinction”);
else if(marks>60 && marks<70)
out.println(“First Class”);
else if(marks>50 && marks<60)
```

```
out.println("Second class");
else
out.println("Failed");
%>
<% out.println("<b>Student name:"+name);
out.println("<b>Student marks are:"+marks); %>
</body></html>
```

Declaring Variables are Methods:

Declaration tag is a block of java code for declaring class wide variables, methods and classes.

<%! Declaration %>

Example:

```
<html><head><title>Declaration Tag</title></head>
<% String name="abc"
        int age=30; %>
<b> The Name is:<%= name %><br>
<b>The age is:<%= age %><br>
<%! int sum(int num1,int num2,int num3){
        return num1+num2+num3;
} %>
<b>The Result is: <%= sum(10,40,50) %>
</body></html>
```

Displaying values using an expression to set an attribute:

```
<html><head><title>JSP Expression</title></head>
<body><h2>JSP Expression </h2>
<ul>
<li>current time: <%= new java.util.Date()%>
<li>host name: <%= request.getResponseHost()%>
<li>Session id: <%= session.getId() %>
</ul></body></html>
```

Sharing data between JSP pages using Session Object:

first.jsp:

```
<%@ Page language=”java”%>
<html><head><title>First</title></head>
<body><form method=”post” action=”second.jsp”>
<b>Username: <input type=”text” name=”uname”><br>
<b> Password: <input type=”password” name=”password”><br>
<input type=”submit” value=”Enter”>
</form></body></html>
```

Second.jsp:

```
<%@Page language=”java” %>
<% string uname=request.getParameter(“uname”);
String password=request.getParameter(“Password”);
Session.getAttribute(“user”,uname);
Session.getAttribute(“password”,password);
%><html><body>< a href=”third.jsp”>Welcome </u> to view Session data</body></html>
```

third.jsp:

```
<@Page language=”java” %>
<% string uname=(string)session.getAttribute(“user”);
String pwd=(string)session.getAttribute(“password”); %>
<html><body>
<b>username is: <%= uname%> <br>
<b> password is:<%= pwd%> </body></html>
```

Scripting Elements:

The Basic Functionality of scripting elements is to allow jsp developers to embed java code directly into jsp. We can classify scripting elements as follows:

i) Declaration

ii) Expression

iii) Scriptlet

1.Declaration:

A JSP declaration lets us define methods or variables in a JSP page. The methods and variables become direct members of the containers generated servlet code. We can define both instance and static members in a declaration. A declaration has the following form:

Example:

```
<%! private int count=1;
Static int total;
Int getcount(){
return count;
}
static int getTotal(){
return total;
}
%>
```

(1) In a declaration we can define a static block of variables and a class also.

(2) In a JSP page we can have any number of declarations in any number of declaring in any order.

(3) Variables declared in a declaration are initialized to java default values.

2. Expressions:

An Expression scripting element is an embedded java.expr that is evaluated and converted to text string. The resulting text string is placed in a JSP output in the location where the element appear. Expressions act as Placeholders for java language expressions. An Expression is evaluated each time the page is accessed and in the value is then embedded into the output HTML.

Example:

```
<%= a+b %>
<%= new java.util.Date()%>
```

(1) Expressions are inserted into the service method of the container generated servlet.

(2) We can have any number of expression per page.

(3) Embedded Java expression should not be terminated with a semicolon.

(4) We can print the value of any object of any primitive datatype to the output stream usng an expression.

(5) We can also use as expression in a JSP action attribute. In this case the value of the expression does not go to the output Stream. It is evaluated at the request time and its value is assigned to the attribute. An expression used in this way is known as request time attribute expression.

3.Scriptlets:

A scriptlet is used to include complete fragments of java code in the body of JSP page. This Scripting element differs from others in two ways:

(1) It is not limited to the declaration of methods and variables.

(2) It does not generate a string output directly as expressions do.

```
<% int sum=0;
for(int i=0;i<10;i++)
sum++;
%>
```

(1) Scriptlet can be placed anywhere within the jsp body.

(2) We can place any number of scriptlet number of scriptlet in JSP.

(3) Each Scriptlet is inserted into the service method of container generated servlet.

(4) The scriptlet is executed each time the page is requested.

(5) The scriptlets contain any java code, they are used for embedding computing logic withing a JSP page.

Comments in a JSP:

In JSP there are three kinds of comments:

(1) Java Comments/ Scripting Comments

(2) HTML Comments/ Content Comments

(3) JSP Comments/ Hidden Comments

In a Declaration or a scriptlet we can have single line multiline java comments.

HTML Comment is of the form <!-- -->

A JSP Comment is of the form <%-- -->

Simple JSP that makes use of the all Scripting Elements:

```
<html><head><title>Example JSP</title></head>
<body><unset><h1>Example JSP</h1>
<%! String names[]={“aaa”,”bbb”,”ccc”};
```

```
private string getName(int index){
return names[index];}
%>
<b>The third name is:<%= getName(2)%><br>
<% for(int i=0;i<=5;i++){  %>
<b>This line is displayed 5 times <br>
<% } %>
</center></body></html>
```

Including content in JSP Page:

There are two mechanisms for including another web resource in a JSP page:

(a) The include Directive

(b) The JSP include element

The include directive is processed when a JSP page is translated into a servlet class. The effect of the directive is to insert the text contained in another file either a static content or another JSP Page in the current JSP page. We would probables use the include directive to the include banner comment, copyright information or my piece of the content that we might want to reuse in another page. The Syntax for include directive is:

<%@ include file=”file name” %>

Because we must practically put an include directive in each file that returns the resource referenced by the directive, this approach has its own limitations.

The JSP:include element is processed when a JSP page is executed. The include action allows us to include either a static or dynamic resource in a JSP file. The results of including static or dynamic resources are quite different. If the resource is static, its content is inserted into the calling JSP file. If the resource is dynamic, the request is sent to the included resource, the included page is executed and then the result is included in the response from the calling JSP page. The syntax for the jsp:include is as follows:

<%@ include page=”include page”%>