Unit 5

Web Technology

UNIT 5

CO5. Develop dynamic web pages using client side programming (Java Script) and server side programming (Servlet and Java Server Pages).

Objective:

- To understand overview and architecture of servlet
- To understand the API of servlet and JSP)
- To understand the life cycle of servlet and JSP.
- To understand the concepts of Redirection in servlet and JSP
- To understand the concepts of JDBC(Java Database Connectivity) driver
- To understand the concepts of session tracking in servlet
- To understand the all types of tags tag in JSP
- To understand the concepts of implicit objects and custom tab libraries in JSP.

UNIT 5

Topics: Servlets: Servlet Overview and Architecture, Interface Servlet and the Servlet Life Cycle, HandlingHTTP get Requests, Handling HTTP post Requests, Redirecting Requests to Other Resources, Session Tracking, Cookies, Session Tracking with HttpSession **(CO5)**

Objective of the above topics:

- To understand the concepts of servlet API and its developement
- To understand the concepts of developing a servlet class and deploying it
- To understand the concepts of handling HTTP get and post request
- To understand the various types of session tracking concepts .

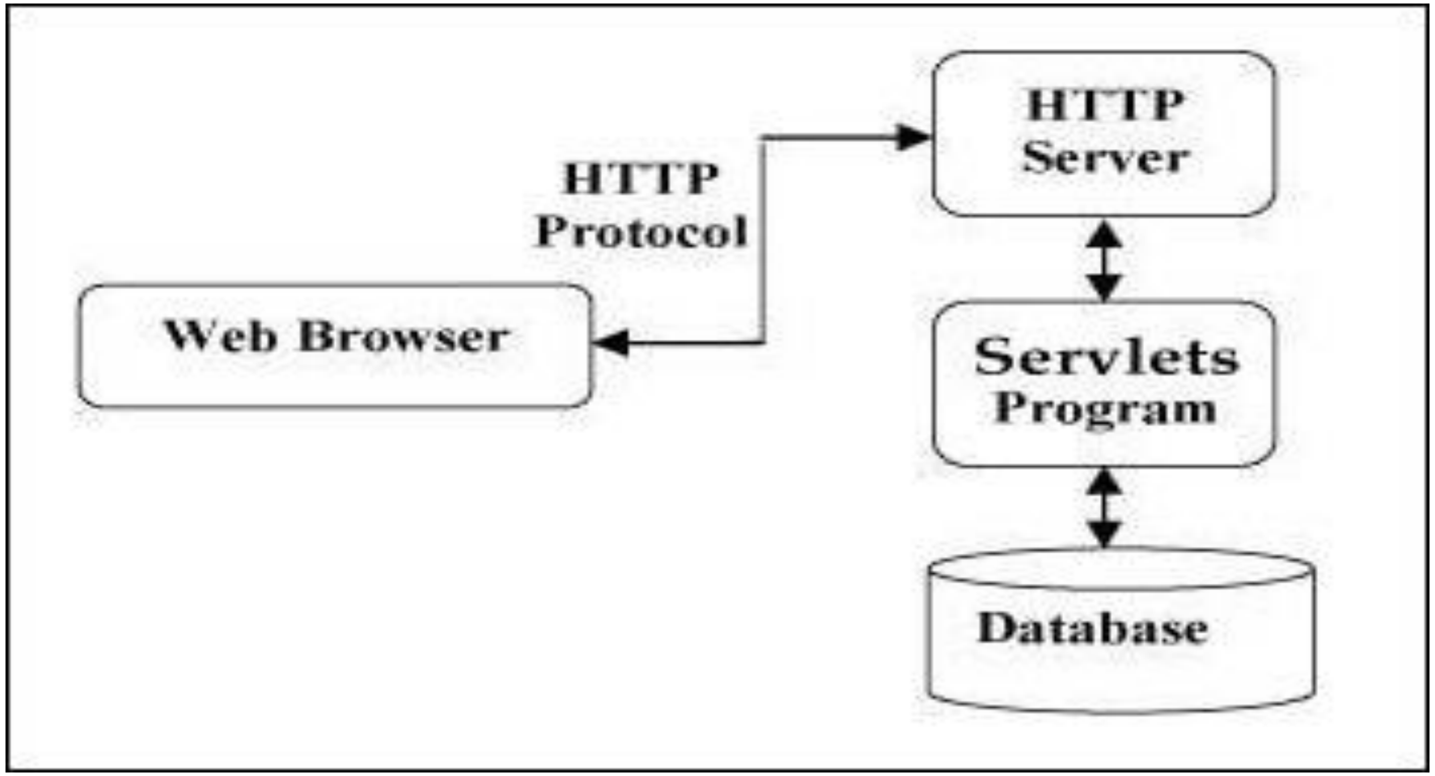
Introduction to Servlet

• Servlet Overview and Architecture

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Introduction to Servlet(cont..)

•**Servlet Architecture**



Introduction to Servlet(cont..)

- **Servlet API Basics**

  - 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.

  - The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Introduction to Servlet(cont..)

- **Interfaces in javax.servlet package**
  - Servlet
  - ServletRequest
  - ServletResponse
  - RequestDispatcher
  - ServletConfig
  - ServletContext
  - SingleThreadModel

Introduction to Servlet(cont..)

- **Classes in javax.servlet package**
  - GenericServlet
  - ServletInputStream
  - ServletOutputStream
  - ServletException

Introduction to Servlet(cont..)

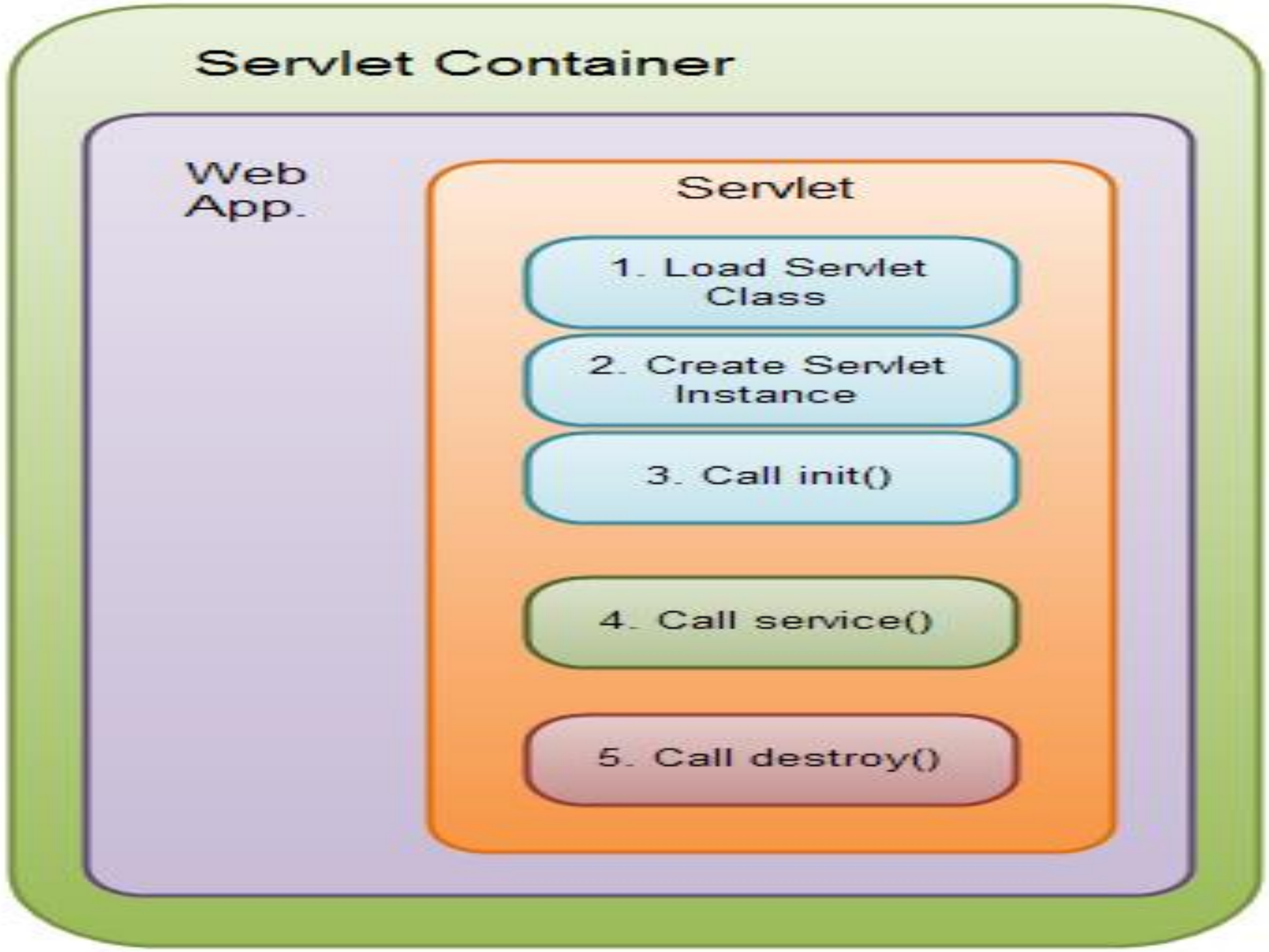
- **Interfaces in javax.servlet.http package**
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
  - HttpSessionListener

Introduction to Servlet(cont..)

- **Life Cycle Of Servlet**
  - Load the Servlet Class.
  - Create the Instance of Servlet.
  - Call the servlet’s init() method.
  - Call the servlet’s service() method.
  - Call the servlet’s destroy() method.

Introduction to Servlet(cont..)

- **Life Cycle Of Servlet(cont..)**



Introduction to Servlet(cont..)

- **Running First Servlet Program**

```
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.io.IOException;
 public class FirstServlet extends GenericServlet
{
 public  void  service(ServletRequest  request,  ServletResponse
response)
        throws ServletException, IOException
 {

      PrintWriter out=res.getWriter();
             out.println("Hello This is First Servlet");   }
```

Introduction to Servlet(cont..)

- **Configuring and Mapping a Servlet**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
    <servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-classFirstServlet </servlet-class>
    </servlet>

     <servlet-mapping>
   <servlet-name> FirstServlet </servlet-name>
   <url-pattern >/FirstServlet/* </url-pattern>
  </servlet-mapping>
 </web-app>
```

Introduction to Servlet(cont..)

- **Handling HTTP Request**

  - The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

- **Handling HTTP GET requests**

  - Handling GET requests involves overriding the doGet method

  - By handling HTTP get request, the client URL shows as string and this can be accessed any time.

Introduction to Servlet(cont..)

- **Example of Handling HTTP get request**

```
public class BookDetailServlet extends HttpServlet
 {
public void doGet (HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException
{ ... // set content-type header before accessing the Writer
response.setContentType("text/html");
response.setContentType("text/html"); PrintWriter out =
response.getWriter();
out.println("<html>" + "<head><title>Book
Description</title></head>" + ...);
```

Introduction to Servlet(cont..)

- **Example of Handling HTTP get request(cont..)**

```
String bookId = request.getParameter("bookId"); if
(bookId != null)
{ // and the information about the book and print it ...
} out.println("</body></html>");
out.close();
}
...
}
```

Introduction to Servlet(cont..)

- **Handling POST Requests**

  - Handling POST requests involves overriding the doPost method.

  - Handling HTTP post request URL does not shows as string and data can not be accessed by another user.

  - This method is much secure

Introduction to Servlet(cont..)

- **Example of Handling HTTP get request**

```
public class ReceiptServlet extends HttpServlet
{
 public void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException
 { ... // set content type header before accessing the Writer
response.setContentType("text/html"); PrintWriter out =
response.getWriter(); // then write the response
out.println("<html>" + "<head><title> Receipt </title>" + ...);
out.println("<h3>Thank you for purchasing your books from us "
+ request.getParameter("cardname") + ...); out.close();  } ... }
```

Introduction to Servlet(cont..)

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Introduction to Servlet(cont..)

- **Redirecting Requests to Other Resources**

  - The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource

  - This method actually makes the client(browser) to create a new request to get to the resource.

  - The client can see the new url in the browser.

  - sendRedirect() accepts relative URL, so it can go for resources inside or outside the server.

Introduction to Servlet(cont..)

- **Example of Redirecting Requests to Other Resources**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RedirectServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletRespo
nse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
  res.sendRedirect("http://www.google.com");
  }
}
```

Introduction to Servlet(cont..)

- **Session Tracking in Servlet**

  - Session simply means a particular interval of time.

  - Session Tracking is a way to maintain state (data) of an user.

  - 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.

Introduction to Servlet(cont..)

- **Cookies In Servlet**

  - A cookie is a small piece of information that is persisted between the multiple client requests.

  - The cookie class provides an easy way for servlet to read, create, and manipulate HTTP-style cookies, which allows servlets to store small amount of data.

  - A servlet uses the getCookies() method to retrieve cookies as request.

  - The addCookie() method sends a new cookie to the browser

Introduction to Servlet(cont..)

- **Cookies Example**

<cookie.html>

```
<html>
   <body>
<center>
   <h1>Cookies Example in Java</h1>
      <form action=““"http://localhost:8080/cookies/co”
method="Post">
        First name: <input type="text" name="fname">
        Last name: <input type="text" name="lname">
        <input type="submit“ value="SUBMIT">
        </form>
</center> </body>  </html
```

Introduction to Servlet(cont..)

- **Cookies Example(cont..)**

<AddCookie.java>

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class AddCookie extends HttpServlet
{
   public void doPost(HttpServletRequest req,HttpServletResponse
res) throws ServletException,IOException
    {
```

Introduction to Servlet(cont..)

- **Cookies Example(cont..)**

<AddCookie.java>(cont..)

```
 String fname=req.getParameter("fname");
String lname=req.getParameter("lname");

Cookie f=new Cookie("first_name",fname);
Cookie l=new Cookie("last_name",lname);
res.addCookie(f);
res.addCookie(l);

res.sendRedirect("http://localhost:8080/cookies/st”);
}
}
```

Introduction to Servlet(cont..)

- **Cookies Example(cont..)**

<GetCookie.java>

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class GetCookie extends HttpServlet
{
   public void doGet(HttpServletRequest
req,HttpServletResponse res) throws
ServletException,IOException
   {
   PrintWriter out=res.getWriter();
out.println("<h1>");
   Cookie[]
c=req.getCookies();
      for(Cookie k:c)
      {
     out.println(k.get
Value());
      }

out.println("</h1>");
   }
}
```

Introduction to Servlet(cont..)

- **Session Tracking using HTTPSession interface**

•The HttpServletRequest interface provides two methods to get the object of HttpSession

- public HttpSession getSession()
- public HttpSession getSession(boolean create)

Introduction to Servlet(cont..)

- **Example of Session Tracking using HTTPSession interface**

<Index.html>

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

Introduction to Servlet(cont..)

- **Example of Session Tracking using HTTPSession interface**

<FirstServlet.java>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class FirstServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResp
onse response){
        try{
          response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String s1=request.getParameter("userName");
        out.print("Welcome "+s1);
```

Introduction to Servlet(cont..)

- **Example of Session Tracking using HTTPSession interface(cont..)**

<FirstServlet.java>

```
HttpSession session=request.getSession();
        session.setAttribute("First",s1);

        out.print("<a href='servlet2'>visit</a>");

        out.close();

                }catch(Exception e){System.out.println(e);}
    }
  }
```

Introduction to Servlet(cont..)

- **Example of Session Tracking using HTTPSession interface(cont..)**

<SecondServlet.java>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
  public class SecondServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServle
tResponse response)
        try{
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession(false);
```

Introduction to Servlet(cont..)

- **Example of Session Tracking using HTTPSession interface(cont..)**

<SecondServlet.java>

```
String n=(String)session.getAttribute(“First");
        out.print("Hello "+n);

        out.close();

                }catch(Exception e){System.out.println(e);}
    }

  }
```

Previous Topics: Recap

- The previous topic was focused on the concepts of servlet API and its developement
- It was focused mainly on concepts of developing a servlet class and deploying it.
- It was discussed about handling HTTP get and post request.
- It was also focused on various types of session tracking concepts .

UNIT 5

Topics: JavaServer Pages (JSP): Introduction, JavaServer Pages Overview, A First JavaServer Page Example, Implicit Objects, Scripting, Standard Actions, Directives, Custom Tag Libraries **(CO5)**

Objective of the above topics:

- To understand the concepts of Java Server Pages (JSP)
- To understand the API and life cycle of JSP.
- To understand the various types of tags and implicit objects in JSP.
- To understand the concept of using custom tag libraries in JSP.

Introduction to Java Server Pages(JSP)

- **Introduction to Java Server Pages Overview**

  - JSP technology is used to create web application just like Servlet technology.

  - 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.

  - JSP pages are converted into Servlet by the Web Container. The Container translates a JSP page into servlet class source(.java) file and then compiles into a Java Servlet class.

Introduction to JSP(cont..)

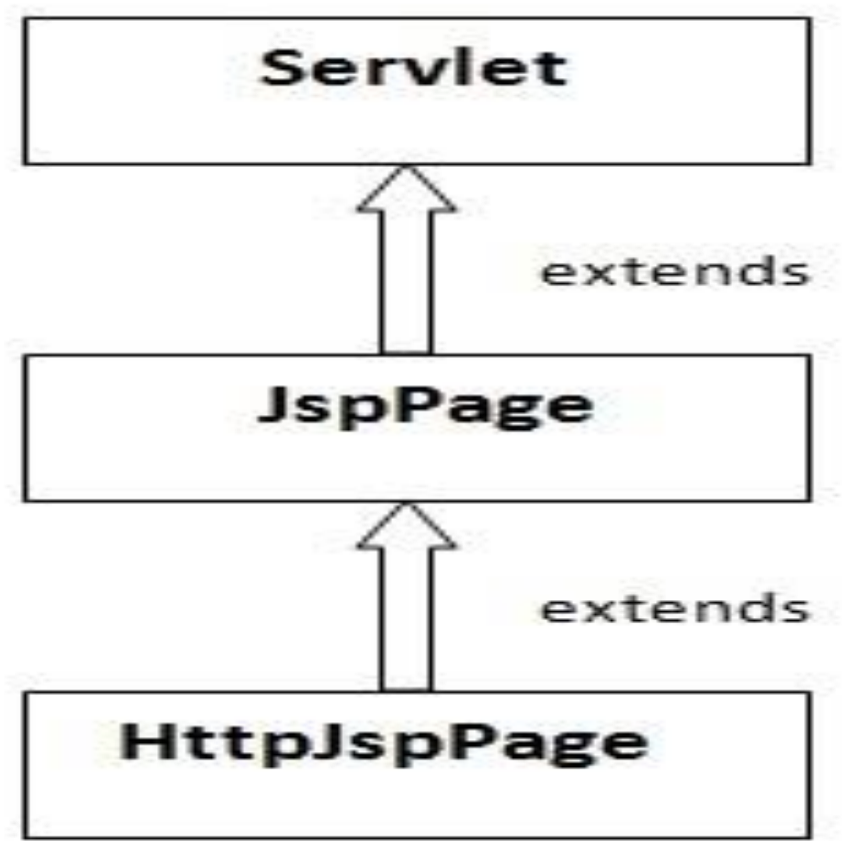
- **Advantage of JSP over Servlet**
    - Extension to Servlet
    - Easy to maintain
    - Fast Development: No need to recompile and redeploy
    - Less code than Servlet

Introduction to JSP(cont..)

- **Life Cycle Of JSP**
  - Translation of JSP Page
  - Compilation of JSP Page
  - Classloading (class file is loaded by the classloader)
  - Instantiation (Object of the Generated Servlet is created).
  - Initialization (jspInit() method is invoked by the container).
  - Reqeust processing (_jspService() method is invoked by the container).
  - Destroy (jspDestroy() method is invoked by the container).

Introduction to JSP(cont..)

- **JSP API**



Introduction to JSP(cont..)

- **A First Java Server Page Example**

<First.html>

```
<html>
<body>
<form action="exp.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

Introduction to JSP(cont..)

- **A First Java Server Page Example(cont..)**

<First.jsp>

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

Introduction to JSP(cont..)

- **Nine Implicit Objects in JSP**

Object	**Type**
out	**JspWriter**
request	**HttpServletRequest**
response	**HttpServletResponse**
config	**ServletConfig**
application	**ServletContext**
session	**HttpSession**
pageContext	**PageContext**
Page	**Object**
exception	**Throwable**

Introduction to JSP(cont..)

- **JSP out implicit object**

  - For writing any data to the buffer, JSP provides an implicit object named out.

  - It is the object of JspWriter

  - **Example**

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Introduction to JSP(cont..)

- **JSP request implicit object**

index.html

```
<form action="welcome.jsp
">
<input type="text" name="
uname">
<input type="submit" value
="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.get
Parameter("uname");
out.print("welcome "+na
me);
%>
```

Introduction to JSP(cont..)

JSP response implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><b
r/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("
http://www.google.com"
);

%>
```

Introduction to JSP(cont..)

JSP config implicit object

- In JSP, config is an implicit object of type *ServletConfig*.

- This object can be used to get initialization parameter for a particular JSP page.

  - **Example**

```
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
```

Introduction to JSP(cont..)

JSP application implicit object

- In JSP, application is an implicit object of type *ServletContext*.

- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

  - **Example**

```
String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
```

Introduction to JSP(cont..)

JSP session implicit object

- In JSP, session is an implicit object of type HttpSession.

•The Java developer can use this object to set,get or remove attribute or to get session information.

- **Example**

```
session.setAttribute("user",name);
```

Introduction to JSP(cont..)

JSP page implicit object

- In JSP, page is an implicit object of type Object class.

•This object is assigned to the reference of auto generated servlet class.

- **Example**

```
<%=page.getClass().getName() %>
<% this.log("message"); %>
```

Introduction to JSP(cont..)

JSP pageContext implicit object

- The pageContext object is an instance of a javax.servlet.jsp.PageContext object.

- The pageContext object is used to represent the entire JSP page.

  - **Example**
    pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

Introduction to JSP(cont..)

JSP exception implicit object

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- In JSP, there are two ways to perform exception handling
  - By **errorPage** and **isErrorPage** attributes of page directive
  - By **<error-page>** element in web.xml file

Introduction to JSP(cont..)

- **JSP Scripting Element**

  - The scripting elements provides the ability to insert java code inside the jsp.

  - There are three types of scripting elements

    - scriptlet tag
    - expression tag
    - declaration tag

Introduction to JSP(cont..)

- **JSP Scripting Element**

  - The scripting elements provides the ability to insert java code inside the jsp.

  - There are three types of scripting elements

    - scriptlet tag
    - expression tag
    - declaration tag

Introduction to JSP(cont..)

- **JSP Scripting Element(cont..)**

JSP scriptlet tag

•A scriptlet tag is used to execute java source code in JSP

Example:

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Introduction to JSP(cont..)

- **JSP Scripting Element(cont..)**

JSP expression tag

- The code placed within JSP expression tag is *written to the output stream of the response*.

- So you need not write out.print() to write data.

- It is mainly used to print the values of variable or method.

Eg. <%= "welcome to jsp" %>

Introduction to JSP(cont..)

- **JSP Scripting Element(cont..)**
  - **JSP Declaration tag**
    - The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet

Syntax:-

<%! field or method declaration %>

Introduction to JSP(cont..)

- **JSP Scripting Element(cont..)**
  - **JSP Declaration tag**
    - The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet

Syntax:-
<%! field or method declaration %>

Introduction to JSP(cont..)

- **JSP Directive tags**
  - **Directive Tag** gives special instruction to Web Container at the time of page translation.
  - Directive tags are of three types: **page**, **include** and **taglib**.

Directive	Description
`<%@ page ... %>`	defines page dependent properties such as language, session, errorPage etc.
`<%@ include ... %>`	defines file to be included.
`<%@ taglib ... %>`	declares tag library used in the page

Introduction to JSP(cont..)

- **JSP Directive tags (cont..)**
  - **JSP page directive**
    - The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

```
<%@ page attribute="value" %>
```

Introduction to JSP(cont..)

- **Attributes of JSP page directive**
  - Import
  - contentType
  - Extends
  - Info
  - Buffer
  - Language
  - isELIgnored
  - isThreadSafe
  - autoFlush
  - Session
  - pageEncoding
  - errorPage
  - isErrorPage

Introduction to JSP(cont..)

•JSP include directive

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.

- The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

•Example : <%@ include file="header.html" %>

Introduction to JSP(cont..)

- **JSP Taglib directive**

  - The JSP taglib directive is used to define a tag library that defines many tags.

  - We use the TLD (Tag Library Descriptor) file to define the tags.

Example:
```
<%@ taglib uri="http://www.google.com/tags" prefix="mytag"
%>
<mytag:currentDate/>
```

Introduction to JSP(cont..)

- **JSP Standard Action Tags**

  - Each JSP action tag is used to perform some specific tasks.

  - The action tags are used to control the flow between pages and to use Java Bean.

Introduction to JSP(cont..)

- **JSP Standard Action Tags(cont..)**

jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

Introduction to JSP(cont..)

- **JSP Custom Tag Library**
    - The JavaServer Pages Standard Tag Library JSTL is a collection of useful JSP tags which encapsulates 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 existing custom tags with JSTL tags.

Introduction to JSP(cont..)

- **Classification of The JSTL Tags**
  - Core Tags
  - Formatting tags
  - SQL tags
  - XML tags
  - JSTL Function

Introduction to JSP(cont..)

- **Classification of The JSTL Tags(cont..)**
  - **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" %>
```

Introduction to JSP(cont..)

- **Classification of The JSTL Tags(cont..)**

•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" %>

Introduction to JSP(cont..)

- **Classification of The JSTL Tags(cont..)**

  - **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" %>
```

Introduction to JSP(cont..)

- **Classification of The JSTL Tags(cont..)**

  - **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.

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

Introduction to JSP(cont..)

- **Classification of The JSTL Tags(cont..)**

  - **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"
%>
```

Previous Topics: Recap

- The previous topic was mainly focused on the concepts of Java Server Pages (JSP) API and its life cycle.

- It was focused on the various types of tags and implicit objects in JSP.

- It was also focused on the concept of using custom tag libraries in JSP.