Java Server Pages

Sandeep, SunBeam

PowerPoint

Introduction

- It is specification defined by Sun/Oracle.
- tomcat/lib contains "**jsp-api.jar**" file which contains implementation of JEE specification.
- JSP is built on top of servlets, but JSP is much easier to program.
- Internally JSP is translated Servlet.
- In case of servlet, java class can contain HTML code whereas in case of JSP, HTML code can contain java code.
- JSP and Servlets are complementary technologies for producing dynamic Web pages via Java. While Servlets are the foundation for server side Java, they are not always the most efficient solution with respect to development time.

Advantages of JSP

1. JSP allows us to separate presentation logic from business logic.
    - In context of JSP, we should use:
        1. JSP for presentation logic
        2. Java Beans for business logic
        3. DAO for data manipulation logic
2. Generally JSP pages contains only tag hence Web Designer can design it using IDE or JSP designer tool.
3. JSP needs no compilation by the Programmer.
4. If we use JSP then we need lesser code hence development becomes faster.

JSP API Version

Sr. No.	JSP API Version	Platform
1	JSP 1.1	J2EE 1.2
2	JSP 1.2	J2EE 1.3
3	JSP 2.0	J2EE 1.4
4	JSP 2.1	Java EE 5
5	JSP 2.2	Java EE 6
6	JSP 2.3	Java EE 7
7	JSP 2.4	Java EE 8

First JSP Page

```
<%@ page language="java"  contentType="text/html; charset=UTF-8” pageEncoding="UTF-8"%>

<html>

    <head>

        <meta http-equiv="Content-Type” content="text/html; charset=UTF-8">

        <title>Hello Page</title>

    </head>

    <body>

        <h3>Hello from JSP.</h3>

    </body>

</html>
```

Translated JSP

- If we put .jsp file in tomcat/webapps then container keep translated servlet inside tomcat/work directory.
- If we run .jsp file from eclipse IDE then container keep translated servlet inside .metadata directory of current workspace:

  "/home/sandeep/Projects/Servlets/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/work/Catalina/localhost/WebApp_Name/org/apache/jsp"

JSP Life Cycle

- Much like Servlets, understanding JSP requires understanding the simple life cycle that JSP follows.
- Handling life cycle of JSP is job of web container(SC / JC).
- JSP follows a Five-phase life cycle:
    1. Translation Phase(JC)
    2. Compilation Phase(JC)
    3. Initialization Phase(SC)
    4. Request Processing Phase(SC)
    5. Destruction Phase(SC)

Translation Phase

- JSP Container i.e. jasper handles translation phase.
- Web browser sends request to the JSP(Hello.jsp) very first time.
- During this phase java server page is translated into servlet.
- As shown in above code "Hello.jsp" gets translated into "Hello_jsp.java".
- If we make any syntactical mistake in JSP then it gets detected in this phase.

Compilation Phase

- JSP Container i.e jasper handles compilation phase.
- In this phase, jasper compiles Hello_jsp servlet automatically i.e "**Hello_jsp.java**" is compiled into "**Hello_jsp.class**".
- Java's syntax errors in translated servlet are detected in this phase.

Initialization Phase

- Servlet Container i.e. Catalina handles initialization phase.

- After compilation, Servlet Container will locate, load, instantiate and initialize translated servlet using jspInit() method. This method gets called only once.
- "**void jspInit( )**" is a method of javax.servlet.jsp.JspPage interface.

Request Processing Phase

- Servlet Container i.e Catalina handles request processing phase.
- To handle the request and to generate dynamic response, servlet container invokes _jspService() method. This method gets called per client request.
- The _jspService()method corresponds to the body of the JSP page. This method is defined automatically by the JSP container and should never be defined by the JSP page author.
- Signature:

  void _jspService(*ServletRequestSubtype* req, *ServletResponseSubtype* resp) throws SE, IOE

- The specific signature depends on the protocol supported by the JSP page.

  void _jspService(HttpServletRequest req, HttpServletResponse resp) throws SE, IOE;

- It is a method of javax.servlet.jsp.HttpJspPage interface.

Destruction Phase

- Before servlet instance gets removed from memory, servlet container gives call to the jspDestroy() method.

- If we stop web application or re-deploy the web application or if we shut down server then servlet container removes servlet instance from memory.

- “**void jspDestroy()**” is a method of javax.servlet.jsp.JspPage interface.

JSP implicit objects

- There are 9 implicit objects of JSP

Sr. No.	**Object**	**Type**
1	out	JspWriter
2	request	HttpServletRequest
3	response	HttpServletResponse
4	session	HttpSession
5	application	ServletContext
6	config	ServletConfig
7	exception	Throwable
8	page	Object
9	pageContext	PageContext

JSP Comments

- There are two types of comments in JSP:
  1. Server side comment
  2. Client side comment
- **Server side comment :**
  - JSP comment is called as server side comment.
  - Syntax : **<%-- JSP Comment --%>**
  - During translation phase JSP container ignore server side comment.
- **Client side comment :**
  - HTML/XML comment is called as client side comment.
  - Syntax : **<!-- HTML Comment -->**
  - JSP Container consider client side comment but web browser ignore it.

Scripting Elements

- If we want to make JSP dynamic then we should use scripting elements.

- There are three different types of scripting elements available for use in JSP:
    1. Scriptlet
    2. Expression
    3. Declaration

Scriptlet

- A scriptlet is defined with a start "<%" and end "%>" with code between.
- Using scriptlets, we can insert java code inside JSP.
- Scriptlets are good for providing low level functionality such as loops, and conditional statements.
- Consider the following example:

```
 <body>
        <%
                Date date = new Date();
                out.println(date);
        %>
</body>
```

- During translation phase, **JSP container insert scriptlet code into _jspService()** method.

Expression

- An expression is defined with a start "<%=", end "%>" and an expression between.
- Expressions provide an easy method of sending out dynamic strings to a client.
- Consider the following example:

```
    <body>
          <%= new Date() %>
    </body>
```

- Above code gets added in _jspService as :   **out.print(new Date());**
- <%= new Date(); %> // JasperException
- Above code gets added in _jspService as :   **out.print(new Date(););**
- **Scriptlet and experession gets inserted into _jspService() method, jsp implicit objects are accessible in scriplet and expression.**

Declaration

- Declaration is the third and final scripting element available for use in JSP.
- A declaration is defined with a start "<%!" and end "%>" with code between.
- If we want declare variable at class scope or override /define method at class scope then we should use declaration.
- Consider the following example:

```
<body>
        <%!
                int count = 0;
        %>
        Count:<%= ++ this.count %>
</body>
```

- Code embedded by a declaration appears outside of the_jspService() method.

Directives

- Directives are messages to a JSP container. They do not send output to a client.

- All directives use the following syntax:

```
<%@ directive {attribute="value"} %>
```

- There are three different JSP directives for use on a page:

  1. **page**
  2. **include**
  3. **taglib**

Page Directive(<%@ page %>)

- The page directive provides page-specific information to a JSP container.

- Attributes for the page directive are as follows:
  1. language
  2. import
  3. session
  4. isErrorPage
  5. errorPage
  6. isThreadSafe

Include Directive(<%@ include %>)

- The include directive is used to include text and/or code at translation time of a JSP.
- The include directive always follows the same syntax:

```
<%@ include file="relativeURL" %>
```

  Files included must be part of a Web Application.

- Since include directives take place at translation time, they are equivalent of directly including the source code in the JSP before compilation and do not result in performance loss at runtime.
- A good example to use is including a common header and footer with multiple pages of content.
- Example:

```
 <body>
         <%@ include file="Header.jsp" %>
         <h2 align="center">Body</h2>
         <%@ include file="Footer.jsp" %>
</body>
```

Taglib Directive(<%@ taglib %>)

- If we want to use custom tag library then we should use taglib directive.

- The **taglib** directive has following syntax

  <%@ taglib uri="uri" prefix="prefixOfTag"%>

  - **uri** attribute value resolves to a location the container understands.
  - The **prefix** attribute informs a container what markup are custom actions.

Exception Handling

- There are three generic types of Throwable objects thrown by Servlets and JSP
    1. IOException
    2. ServletException
    3. JspException
- Exceptions thrown from a Servlet or JSP can be handled on an individual basis or on an application-wide basis.
- In addition to the try-catch-finally statement, JSP can use the page directive to specify a page that uncaught exceptions are passed to.
- The page directive's "**errorPage**" attribute can be assigned a relative URL value representing a JSP or Servlet especially designed to be an error page.

```
<%@ page errorPage="ErrorPage.jsp" %>
```

- A JSP designed to be an error page can set the page directive isErrorPage attribute to true; this makes the exception implicit object automatically available to represent a passed exception.

```
<%@ page isErrorPage="true" %>
```

Exception Handling

- Error pages can be defined on a per Web Application basis by a Web Application Deployment Descriptor, web.xml.
- web.xml Error Page for All Checked Exceptions:

```
     <error-page>
           <exception-type> java.lang.Exception </exception-type>
           <location> /ErrorPage.jsp </location>
    </error-page>
```

- web.xml Error Page Entry for HTTP Status Code:

```
    <error-page>
           <error-code>404</error-code>
           <location>/FileNotFound.jsp</location>
    </error-page>
```

Exception Handling

- **ErrorData** is a final class declared in **javax.servlet.jsp** package.
- It contains information about an error, for error pages.
- Methods:
    1. public String **getRequestURI**()
    2. public int **getStatusCode**()
    3. public String **getServletName**()
    4. public Throwable **getThrowable**()

Scope in JSP

- There are 4 types of scope in JSP:

    1. page
    2. request
    3. session
    4. application.

Thank you