JSP Example Tutorial for Beginners

In last few posts, a wrote a lot about **Java Servlet** and got very good response from our readers. So I have started another series on JSP tutorials and this is the first post of the series.

In this tutorial, we will look into the basics of JSP, advantages of JSP over Servlets, Life Cycle of JSP, JSP API interfaces and Classes and where can we put JSP files in the web application.

We will also look into the JSP Comments, Scriptlets, Directives, Expression, Declaration and JSP attributes in brief detail. Some of these topics are very important and we will look into them in more detail in future posts.

1. What is JSP and why do we need JSP?

JSP (JavaServer Pages) is server side technology to create dynamic java web application. JSP can be thought as an extension to servlet technology because it provides features to easily create user views.

JSP Page consists of HTML code and provide option to include java code for dynamic content. Since web applications contain a lot of user screens, JSPs are used a lot in web applications. To bridge the gap between java code and HTML in JSP, it provides additional features such as JSP Tags, Expression Language, Custom tags. This makes it easy to understand and helps a web developer to quickly develop JSP pages.

2. Advantages of JSP over Servlets?

- We can generate HTML response from servlets also but the process is cumbersome and error prone, when it comes to writing a complex HTML response, writing in a servlet will be a nightmare. JSP helps in this situation and provide us flexibility to write normal HTML page and include our java code only where it's required.
- JSP provides additional features such as tag libraries, expression language, custom tags that helps in faster development of user views.
- JSP pages are easy to deploy, we just need to replace the modified page in the server and container takes care of the deployment. For servlets, we need to recompile and deploy whole project again.

Actually Servlet and JSPs compliment each other. We should use Servlet as server side controller and to communicate with model classes whereas JSPs should be used for presentation layer.

3. Life cycle of JSP Page

JSP life cycle is also managed by container. Usually every web container that contains servlet container also contains JSP container for managing JSP pages.

JSP pages life cycle phases are:

- **Translation** – JSP pages doesn't look like normal java classes, actually JSP container parse the JSP pages and translate them to generate corresponding servlet source code. If JSP file name is home.jsp, usually its named as home_jsp.java.
- **Compilation** – If the translation is successful, then container compiles the generated servlet source file to generate class file.

- **Class Loading** – Once JSP is compiled as servlet class, its lifecycle is similar to servlet and it gets loaded into memory.
- **Instance Creation** – After JSP class is loaded into memory, its object is instantiated by the container.
- **Initialization** – The JSP class is then initialized and it transforms from a normal class to servlet. After initialization, ServletConfig and ServletContext objects become accessible to JSP class.
- **Request Processing** – For every client request, a new thread is spawned with ServletRequest and ServletResponse to process and generate the HTML response.
- **Destroy** – Last phase of JSP life cycle where it's unloaded into memory.

4. Life cycle methods of JSP

JSP lifecycle methods are:

A. **jspInit()** declared in JspPage interface. This method is called only once in JSP lifecycle to initialize config params.
B. **_jspService(HttpServletRequest request, HttpServletResponse response)** declared in HttpJspPage interface and response for handling client requests.
C. **jspDestroy()** declared in JspPage interface to unload the JSP from memory.

5. Simple JSP Page Example with Eclipse and Tomcat

We can use Eclipse IDE for building dynamic web project with JSPs and use Tomcat to run it. Please read Java Web Applications tutorial to learn how can we easily create JSPs in Eclipse and run it in tomcat.

A simple JSP page example is:

home.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>First JSP</title>
</head>
<%@ page import="java.util.Date" %>
<body>
<h3>Hi Pankaj</h3><br>
<strong>Current Time is</strong>: <%=new Date() %>

</body>
</html>
```

If you have a simple JSP that uses only JRE classes, we are not required to put it as WAR file. Just create a directory in the tomcat webapps folder and place your JSP file in the newly created directory. For example, if your JSP is located at apache-`tomcat/webapps/test/home.jsp`, then you can access it in browser with URL `http://localhost:8080/test/home.jsp`. If your host and port is different, then you need to make changes in URL accordingly.

6. JSP Files location in Web Application WAR File

We can place JSP files at any location in the WAR file, however if we put it inside the WEB-INF directory, we wont be able to access it directly from client.

We can configure JSP just like servlets in web.xml, for example if I have a JSP page like below inside WEB-INF directory:

test.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Test JSP</title>
</head>
<body>
Test JSP Page inside WEB-INF folder.<br>
Init Param "test" value =<%=config.getInitParameter("test") %><br>
HashCode of this object=<%=this.hashCode() %>
</body>
</html>
```

And I configure it in web.xml configuration as:

web.xml

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

  <servlet>
  <servlet-name>Test</servlet-name>
  <jsp-file>/WEB-INF/test.jsp</jsp-file>
  <init-param>
    <param-name>test</param-name>
    <param-value>Test Value</param-value>
  </init-param>
  </servlet>

  <servlet-mapping>
  <servlet-name>Test</servlet-name>
  <url-pattern>/Test.do</url-pattern>
  </servlet-mapping>

  <servlet>
  <servlet-name>Test1</servlet-name>
```

```
    <jsp-file>/WEB-INF/test.jsp</jsp-file>
    </servlet>

    <servlet-mapping>
    <servlet-name>Test1</servlet-name>
    <url-pattern>/Test1.do</url-pattern>
    </servlet-mapping>
</web-app>
```

Then I can access it with both the URLs http://localhost:8080/FirstJSP/Test.do and http://localhost:8080/FirstJSP/Test1.do

Notice that container will create two instances in this case and both will have their own servlet config objects, you can confirm this by visiting these URLs in browser.

For Test.do URI, you will get response like below.

```
Test JSP Page inside WEB-INF folder.
Init Param "test" value =Test Value
HashCode of this object=1839060256
```

For Test1.do URI, you will get response like below.

```
Test JSP Page inside WEB-INF folder.
Init Param "test" value =null
HashCode of this object=38139054
```

Notice the init param value in second case is null because it's not defined for the second servlet, also notice the hashcode is different.
If you will make further requests, the hashcode value will not change because the requests are processed by spawning a new thread by the container.

Did you noticed the use of **config** variable in above JSP but there is no variable declared, it's because its one of the 9 implicit objects available in JSP page, read more about them at JSP Implicit Objects.

7. JSP API Interfaces and Classes

All the core JSP interfaces and classes are defined in `javax.servlet.jsp` package. Expression Language API interfaces are classes are part of `javax.servlet.jsp.el` package. JSP Tag Libraries interfaces and classes are defined in `javax.servlet.jsp.tagext` package.

Here we will look into interfaces and classes of Core JSP API.

• JspPage Interface

JspPage interface extends Servlet interface and declares jspInit() and jspDestroy() life cycle methods of the JSP pages.

- ## HttpJspPage Interface

  HttpJspPage interface describes the interaction that a JSP Page Implementation Class must satisfy when using the HTTP protocol. This interface declares the service method of JSP page for HTTP protocol as *public void _jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException*.

- ## JspWriter abstract Class

  Similar to PrintWriter in servlets with additional facility of buffering support. This is one of the implicit variables in a JSP page with name “out”. This class extends java.io.Writer and container provide their own implementation for this abstract class and use it while translating JSP page to Servlet. We can get it’s object using PageContext.getOut() method. Apache Tomcat concrete class for JspWriter is `org.apache.jasper.runtime.JspWriterImpl`.

- ## JspContext abstract Class

  JspContext serves as the base class for the PageContext class and abstracts all information that is not specific to servlets. The JspContext provides mechanism to obtain the JspWriter for output, mechanism to work with attributes and API to manage the various scoped namespaces.

- ## PageContext abstract Class

  PageContext extends JspContext to provide useful context information when JSP is used for web applications. A PageContext instance provides access to all the namespaces associated with a JSP page, provides access to several page attributes, as well as a layer above the implementation details. Implicit objects are added to the pageContext automatically.

- ## JspFactory abstract Class

  The JspFactory is an abstract class that defines a number of factory methods available to a JSP page at runtime for the purposes of creating instances of various interfaces and classes used to support the JSP implementation.

- ## JspEngineInfo abstract Class

  The JspEngineInfo is an abstract class that provides information on the current JSP engine.

- ## ErrorData final Class

Contains information about an error, for error pages.

- ### JspException Class

  A generic exception known to the JSP container, similar to ServletException.If JSP pages throw JspException then errorpage mechanism is used to present error information to user.

- ### JspTagException Class

  Exception to be used by a Tag Handler to indicate some unrecoverable error.

- ### SkipPageException Class

  Exception to indicate the calling page must cease evaluation. Thrown by a simple tag handler to indicate that the remainder of the page must not be evaluated. This exception should not be thrown manually in a JSP page.

8. JSP Comments

Since JSP is built on top of HTML, we can write comments in JSP file like html comments as

`<-- This is HTML Comment -->`

These comments are sent to the client and we can look it with view source option of browsers.

We can put comments in JSP files as:

`<%-- This is JSP Comment--%>`

This comment is suitable for developers to provide code level comments because these are not sent in the client response.

9. JSP Scriptlets

Scriptlet tags are the easiest way to put java code in a JSP page. A scriptlet tag starts with `<%` and ends with `%>`.

Any code written inside the scriptlet tags go into the `_jspService()` method.

For example:

```
<%
Date d = new Date();
System.out.println("Current Date="+d);
%>
```

10. JSP Expression

Since most of the times we print dynamic data in JSP page using *out.print()* method, there is a shortcut to do this through JSP Expressions. JSP Expression starts with `<%=` and ends with `%>`.

`<% out.print("Pankaj"); %>` can be written using JSP Expression as `<%= "Pankaj" %>`

Notice that anything between `<%= %>` is sent as parameter to `out.print()` method. Also notice that scriptlets can contain multiple java statements and always ends with semicolon (;) but expression doesn't end with semicolon.

11. JSP Directives

JSP Directives are used to give special instructions to the container while JSP page is getting translated to servlet source code. JSP directives starts with `<%@` and ends with `%>`

For example, in above JSP Example, I am using *page* directive to to instruct container JSP translator to import the Date class.

12. JSP Declaration

JSP Declarations are used to declare member methods and variables of servlet class. JSP Declarations starts with `<%!` and ends with `%>`.

For example we can create an int variable in JSP at class level as `<%! public static int count=0; %>`

13. JSP transformed Servlet Source Code and Class File location in Tomcat

Once JSP files are translated to Servlet source code, the source code (.java) and compiled classes both are place in **Tomcat/work/Catalina/localhost/FirstJSP/org/apache/jsp** directory. If the JSP files are inside other directories of application, the directory structure is maintained.

For JSPs inside WEB-INF directory, its source and class files are inside **Tomcat/work/Catalina/localhost/FirstJSP/org/apache/jsp/WEB_002dINF** directory.

Here is the source code generated for above test.jsp page.

test_jsp.java

```java
/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/7.0.32
 * Generated at: 2013-08-21 03:40:59 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp.WEB_002dINF;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_depe

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;

  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }

  public void _jspInit() {
    _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServl
    _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactor
  }

  public void _jspDestroy() {
  }

  public void _jspService(final javax.servlet.http.HttpServletRequest requ
        throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;


    try {
      response.setContentType("text/html; charset=US-ASCII");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\n");
      out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitiona
      out.write("<html>\n");
      out.write("<head>\n");
      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; ch
      out.write("<title>Test JSP</title>\n");
      out.write("</head>\n");
      out.write("<body>\n");
      out.write("Test JSP Page inside WEB-INF folder.<br>\n");
      out.write("Init Param \"test\" value =");
      out.print(config.getInitParameter("test") );
      out.write("<br>\n");
      out.write("HashCode of this object=");
      out.print(this.hashCode() );
      out.write("\n");
      out.write("</body>\n");
      out.write("</html>");
    } catch (java.lang.Throwable t) {
      if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          try { out.clearBuffer(); } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageExcep
        else throw new ServletException(t);
      }
    } finally {
      _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

Notice following points in above servlet code;

- The package of class starts with org.apache.jsp and if JSPs are inside other folders, it includes directory hierarchy too. Usually we dont care about it.
- The generates servlet class is final and can't be extended.
- It extends `org.apache.jasper.runtime.HttpJspBase` that is similar to HttpServlet except that it's internal to Tomcat JSP Translator implementation. HttpJspBase extends HttpServlet and implements HttpJspPage interface.
- Notice the local variables at the start of _jspService() method implementation, they are automatically added by JSP translator and available for use in service methods, i.e in scriptlets.

As a java programmer, sometimes it helps to look into the generated source for debugging purposes.

14. JSP init parameters

We can define init parameters for the JSP page as shown in above example and we can retrieve them in JSP using **config** implicit object, we will look into implicit objects in JSP in more detail in future posts.

15. Overriding JSP init() method

We can override JSP init method for creating resources to be used by JSP service() method using JSP Declaration tags, we can override jspInit() and jspDestroy() or any other methods also. However we should never override _jspService() method because anything we write in JSP goes into service method.

16. Attributes in a JSP

Apart from standard servlet attributes with request, session and context scope, in JSP we have another scope for attributes, i.e Page Scope that we can get from pageContext object. We will look it’s importance in custom tags tutorial. For normal JSP programming, we don’t need to worry about page scope.

Thats all for JSP tutorial for beginners, I hope it helps you in understanding the basic concepts of JSPs and help you in getting started. We will look into other JSP features in future posts.