Introduction to JSP

http://www.tutorialspoint.com/jsp/index.htm

Java Server Pages

- Java Server Pages (JSP) is a technology that lets you mix regular, static HTML with dynamically-generated HTML using Java code
- Similar to PHP
- Servlet – process and produce input using out.
- JSP – process and produce output in plain HTML, for most cases

Java Server Pages

- ***Separation of dynamic and static content***
  - The JavaServer Pages technology enables the separation of static content from dynamic content that is inserted into the static template.
  - This greatly simplifies the creation of content.
  - This separation is supported by beans specifically designed for the interaction with server-side objects, and, specially, by the tag extension mechanism (using MVC, not by using JSP alone.

hello.jsp

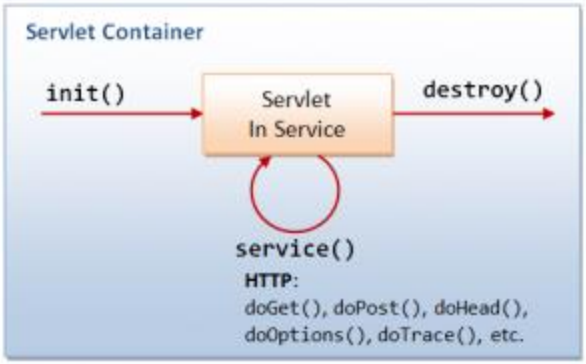
```
<HTML>
<HEAD><TITLE>Hello World</TITLE></HEAD>
<BODY>
<% String name = request.getParameter(“name”); %>
<% String ic = request.getParameter(“ic”); %>
Hello <B><%= name %>[<%= ic %>]</B> welcome to JSP World
</BODY>
</HTML>
```

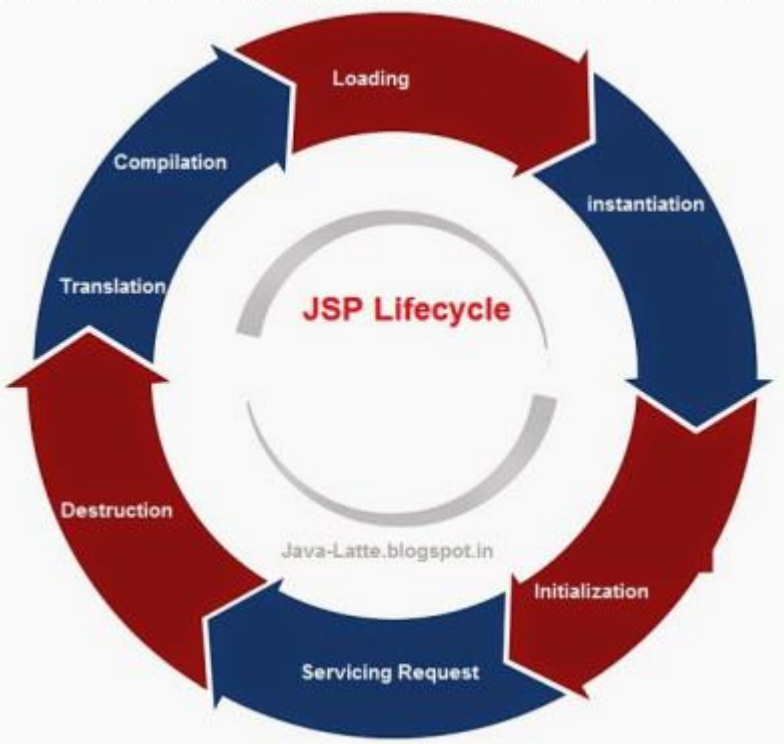
Advantage JSP over Servlet

- Problem with servlet - content and presentation in one place
- A lot of out.println()
- Servlet programmer also a web page designer
- Create the HTML pages (by the WEB designer), save it as .jsp extention, and leave the content to the coded by the JSP programmer

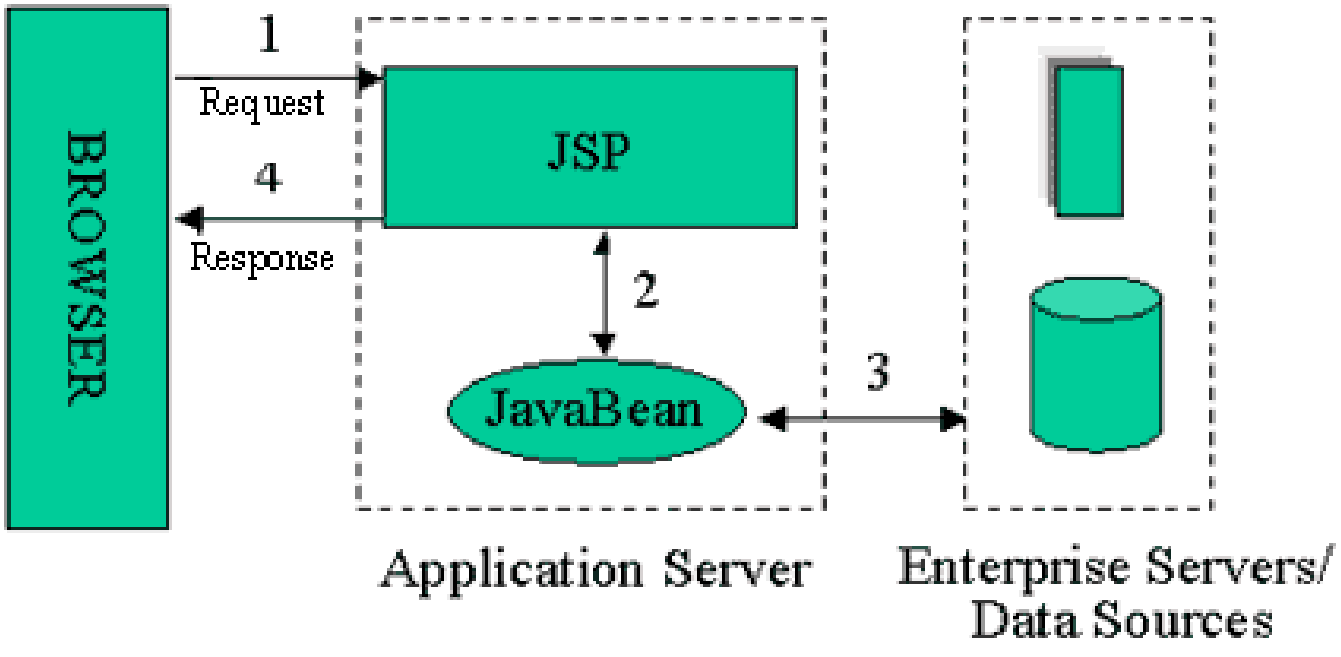
- https://www.upgrad.com/blog/jsp-vs-servlet/

Servlet	JSP
Servlets are faster as compared to JSP, as they have a short response time.	JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code.
Servlets are Java-based codes.	JSP are HTML-based codes.
Servlets are harder to code, as here, the HTML codes are written in Java.	JSPs are easier to code, as here Java is coded in HTML.
In an MVC architecture, Servlets act as the controllers.	In MVC architectures, the JSPs act as a view to present the output to the users.
The Servlets are capable of accepting all types of protocol requests.	The JSPs are confined to accept only the HTTP requests.
Modification in Servlets is a time-consuming and challenging task, as here, one will have to reload, recompile, and then restart the servers.	Modification is easy and faster in JSPs as we just need to refresh the pages.
Servlets require the users to enable the default sessions management explicitly, as Servlets do not provide default session management.	JSPs provide session management by default.
Servlets do not provide the facility of writing custom tags.	JSPs can provide the facility of building the JSP tags easily, which can directly call javaBeans.
In Servlets, we do not have implicit objects.	In JSPs, we have support for implicit objects.
Servlets are hosted and executed on Web Servers.	JSP is compiled in Java Servlets before their execution. After that, it has a similar lifecycle as Servlets.


Servlet Container
init()
Servlet
In Service
destroy()
service()
HTTP:
doGet(), doPost(), doHead(),
doOptions(), doTrace(), etc.


Loading
instantiation
Initialization
Servicing Request
Destruction
Translation
Compilation
JSP Lifecycle
Java-Latte.blogspot.in

JSP Model 1 Architecture



Java Server Pages Operation

- Create JSP pages like normal HTML file
- Once invoked - automatically converted to normal servlet, with the static HTML simply being printed out to output stream associated with the servlet's service method
- Translation normally done only once the first time the page is requested
- to ensure that the first real user doesn't get a momentary delay when the JSP page is translated into a servlet and compiled, developers can simply request the page themselves after installing it

Template Text: Static HTML

- In many cases, a large percent of your JSP page just consists of static HTML, known as template text.
- This HTML looks just like normal HTML, follows all the same syntax rules, and is simply "passed through" to the client by the servlet OUT created to handle the page.
- Not only does the HTML look normal, it can be created by whatever tools you already are using for building Web pages (Textpad/Dreamweaver).

JSP Scripting Elements

- JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:
  - Expressions of the form `<%= expression %>` that are evaluated and inserted into the output,
  - Scriptlets of the form `<% code %>` that are inserted into the servlet's service method, and
  - Declarations of the form `<%! code %>` that are inserted into the body of the servlet class, outside of any existing methods. (Global variable / other method)

JSP Scripting Elements

- **JSP Expressions**
  - A JSP expression is used to insert Java values directly into the output. It has the following form:

```
<%= Java Expression %>
```

  - The Java expression is evaluated, converted to a string, and inserted in the page. - NOTICE NO COMA!
  - This evaluation is performed at run-time (when the page is requested), and has full access to information about the request.
  - For example:

```
Current time: <%= new java.util.Date() %>
```

JSP Scripting Elements

- **JSP Expressions (continue)**
  - To simplify these expressions, there are a number of predefined variables that you can use.
  - These variables call implicit objects, the most important ones are:
  - **request**, the HttpServletRequest
  - **response**, the HttpServletResponse
  - **session**, the HttpSession associated with the request (if any);
  - **out**, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.
  - Here's an example:

```
Your hostname: <%= request.getRemoteHost() %>
```

JSP Scripting Elements

- **JSP Scriptlets**
  - If you want to do something more complex than insert a simple expression, JSP scriptlets let you insert arbitrary code into the servlet method that will be built to generate the page.
  - Scriptlets have the following form:

<% Java Code; %>

  - Scriptlets have access to the same automatically defined variables as expressions.
  - So, for example, if you want output to appear in the resultant page, you would use the **out** variable.

JSP Scripting Elements

- **JSP Scriptlets (cont.)**
  - Example:

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
```

  - Note that code inside a scriptlet gets inserted exactly as written, and any static HTML (template text) before or after a scriptlet gets converted to print statements.
  - For example, the following JSP fragment, containing mixed template text and scriptlets

JSP Scripting Elements

- **JSP Scriptlets (cont.) - greetings.jsp /**

```
<% if (Math.random() < 0.5) { %>
Have a <B>nice</B> day!
<% } else { %>
Have a <B>lousy</B> day!
<% } %>
```

- will get converted to something like:

```
if (Math.random() < 0.5) {
  out.println("Have a <B>nice</B> day!");
} else {
  out.println("Have a <B>lousy</B> day!");
}
```

JSP Scripting Elements

- **JSP Declarations**
  - A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (outside of the service method processing the request).
  - It has the following form:

```
<%! Java Code %>
```

  - Since declarations do not generate any output, they are normally used in conjunction with JSP expressions or scriptlets. - AccessCounts.jsp

JSP Comments

- <%-- comment --%>
  - A JSP comment.
  - Ignored by JSP-to-scriptlet translator.
  - Not to be found in the resultant HTML
- <!-- comment -->
  - An HTML comment.
  - Passed through to resultant HTML as a comment in the HTML.

JSP Directive

- A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@ directive attribute="value" %>

JSP Directive

- There are two main types of directive:
  - page, which lets you do things like import classes, customize the servlet superclass, and the like;
  - and include, which lets you insert a file into the servlet class at the time the JSP file is translated into a servlet.

JSP page Directive

- The page directive lets you define one or more of the following case-sensitive attributes:
  - **import attribute**
    - import=*"package.class"* or import=*"package.class1,...,package.classN"*
    - This lets you specify what packages should be imported.
    - For example:
    - **<%@ page import="java.util.*" %>**
    - The import attribute is the only one that is allowed to appear multiple times.

JSP include Directive

- This directive lets you include files at the time the JSP page is translated into a servlet. The directive looks like this:

- <%@ include file="relative url" %>

- The URL specified is normally interpreted relative to the JSP page that refers to it,
- but, as with relative URLs in general, you can tell the system to interpret the URL relative to the home directory of the Web server by starting the URL with a forward slash.
- The contents of the included file are parsed as regular JSP text,
- and thus can include static HTML, scripting elements, directives, and actions.

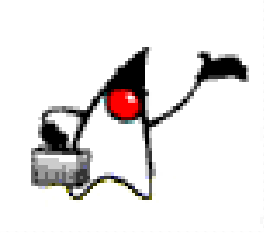
Presentation Outline

- Introduction / Motivation
- What is JSP?
- Advantages of using JSP
- How does JSP work?
- Syntax
- Examples

Introduction / Motivation

- Need to present dynamic content to web site users for applications such as e-commerce, customized web sites, etc
- Need to be able to access database or other server-side resources
- Want to make development as fast and easy as possible

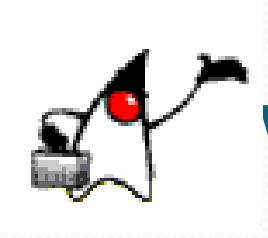
Is there a solution?



What is JSP?

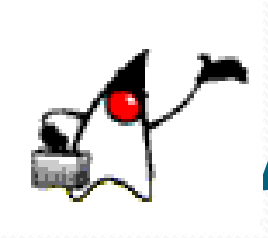
Why yes, there is!

- Server-side scripting language developed by Sun Microsystems to create dynamic/interactive web content
- Scripting done by Java code embedded within static HMTL using XML-like JSP tags and ‘scriptlets’
- Allows for seamless integration of static HTML with server-side Java



What is JSP?

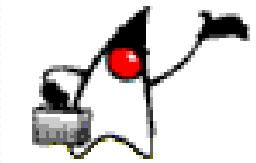
- An extension to the Servlet API:
  - Provides an abstraction above the level of the Servlet
  - Provides usage of the same core features and services
  - Allows integration with existing Java classes and JavaBeans



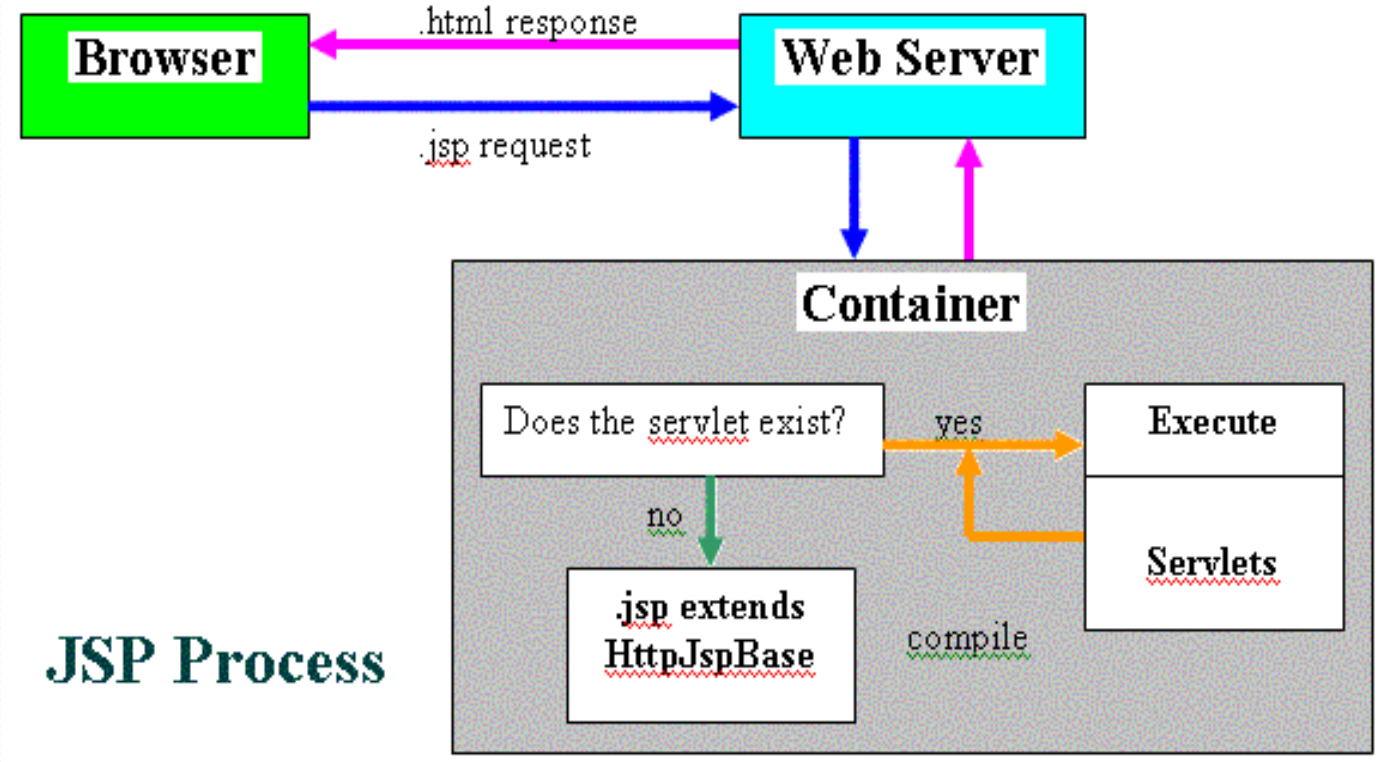
Advantages

- JSP programming is easy!

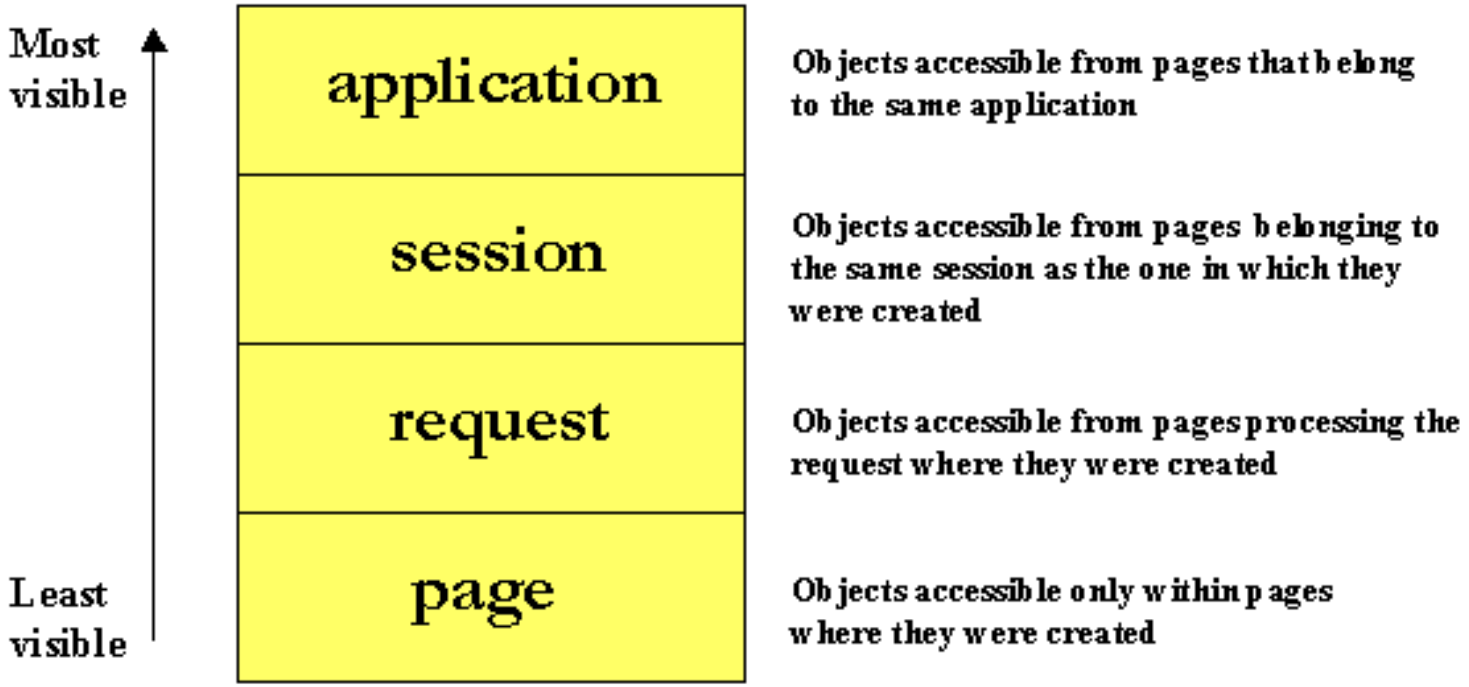
  (For anyone familiar with HTML and Java)
- No need to explicitly compile
- Can be deployed on virtually any platform; only requires Apache web server and Tomcat
- Allows separation of dynamic and static content



Process


Browser
.html response
Web Server
.jsp request
Container
Does the servlet exist?
yes
Execute
no
Servlets
.jsp extends
HttpJspBase
compile
JSP Process

Scope


Most visible
Least visible
application
session
request
page
Objects accessible from pages that belong to the same application
Objects accessible from pages belonging to the same session as the one in which they were created
Objects accessible from pages processing the request where they were created
Objects accessible only within pages where they were created

Synchronization

- Default – Servlets instantiated only once
- Multithreaded to serve multiple instances
- Shared class variables, concurrence problems
- <%@ page is ThreadSafe = “false” %>
  - Slow round robin service

Syntax

- **Expressions**

  Expression is evaluated and placed in output

  <%= expression %>

  <%= new java.util.Date( ) %>

- **Scriptlets**

  Code is inserted in service method.

  <% code %>

Syntax

- **Declarations**

  Code is inserted in body of servlet class, outside of service method.

  <%! code %>

  <%! private int accessCount = 0; %>

- **Directives**

  Messages that enable the programs to set the overall structure of the resulting servlet.

  <%@ settings %>

Syntax

- **Page Directives**

  Directions to the servlet engine about general setup.

  <%@ page att="val" %>

  <%@ page import ="java.util.*" %>

- **Include Directives**

  A file is inserted when the JSP page is translated.

  <%@ include file=“Relative url" %>

Syntax

- **Actions**

  Predefined tasks that are processed by the JSP container at request time.

  <jsp:include> Action

  Includes a file at the time the page is requested.

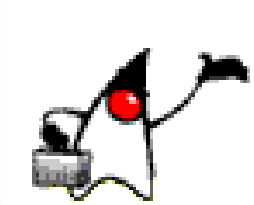
  <jsp: include page="banner.html" flush = "true" />

Syntax

<jsp:useBean> Action

Declares a Java Bean instance for use in the JSP page.

```
<jsp:useBean id="courseBean"
        class="coursepack.CourseListBean" />
```



Syntax

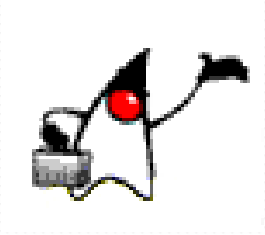
- **<jsp: getProperty> Action**

  Gets a property in the specified JavaBean instance.

  <jsp:getProperty name="courseBean"
  property="courseColor" />

- Equivalent to expression:

  <%= courseBean.getCourseColor(courseNumber) %>



Syntax

- **<jsp: setProperty> Action**

  Sets a property in the specified JavaBean instance.

  <jsp:setProperty name="courseBean"
  property="courseColor" value="blue" />

  Equivalent to expression:

  <%= courseBean.setCourseColor("red") %>