Introduction to JSP

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

Netbeans – jsp.zip

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.

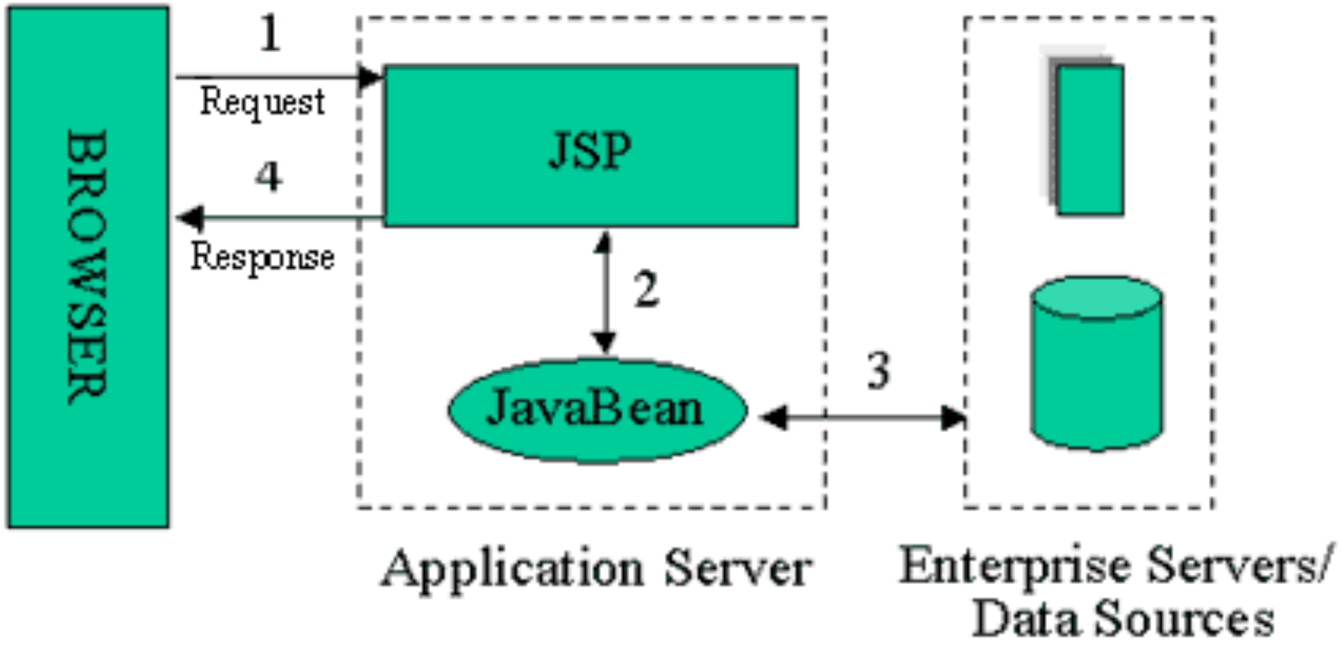
hello.jsp

```
<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

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.

JSP include Directive

- Example
  - SomeRandomPage.jsp
  - SomeRandomPage2.jsp

JSP Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine.
- You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
- Available actions include:
  - jsp:forward - Forward the requester to a new page
    - **Session_set.jsp**
    - **Session_set2.jsp**
    - **View_session.jsp**

- Shopping Cart All-Servlet example
  - VCD_Servlet
- Shopping Cart All-JSP example
  - VCD_JSP

Model View Controller Design Pattern

- To understand the relevance of MVC design pattern, we must understand the problem with the architecture of both servlet and Java Server Pages (JSP)
- **Servlet Problem**
  - the problem is associated with the coupling of data processing and data formatting which make the servlet programmer also a Web designer
  - A lot of out.println() – zillions!
  - View code `ShoppingServlet.java`
  - HTML presentations are hard coded in java classes
  - Hard to redesign / change

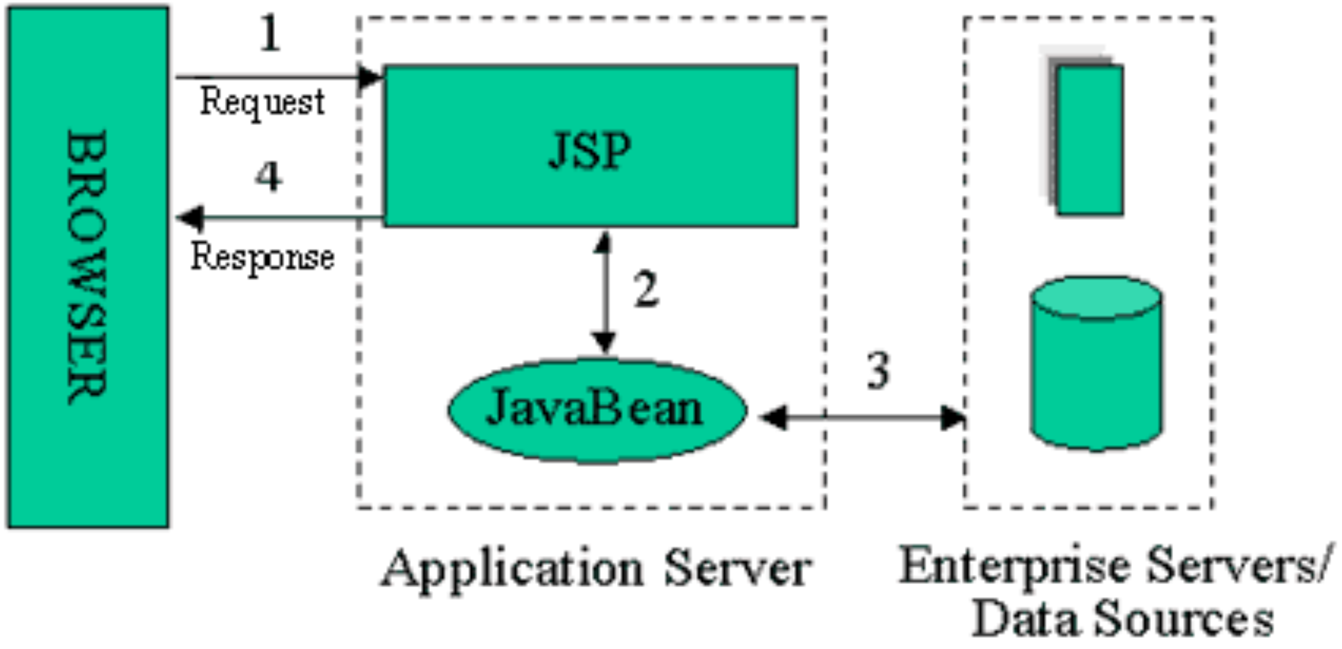
Model View Controller Design Pattern

- **Servlet Problem** - continue
  - HTML presentations are hard coded in java classes
  - Hard to redesign / change
  - Tiresome changing from html code to java string code – sometimes confusing
  - Hard to debug html problem

```
<form method=”post”
     action=”/servlet/kuantan.HelloWorld”>
out.println(“<form method=\”post\”
  action=\”/servlet/kuantan.HelloWorld\”>”);
```

  – To change html presentations, Java code have to be recompiled

JSP Model 1 Architecture



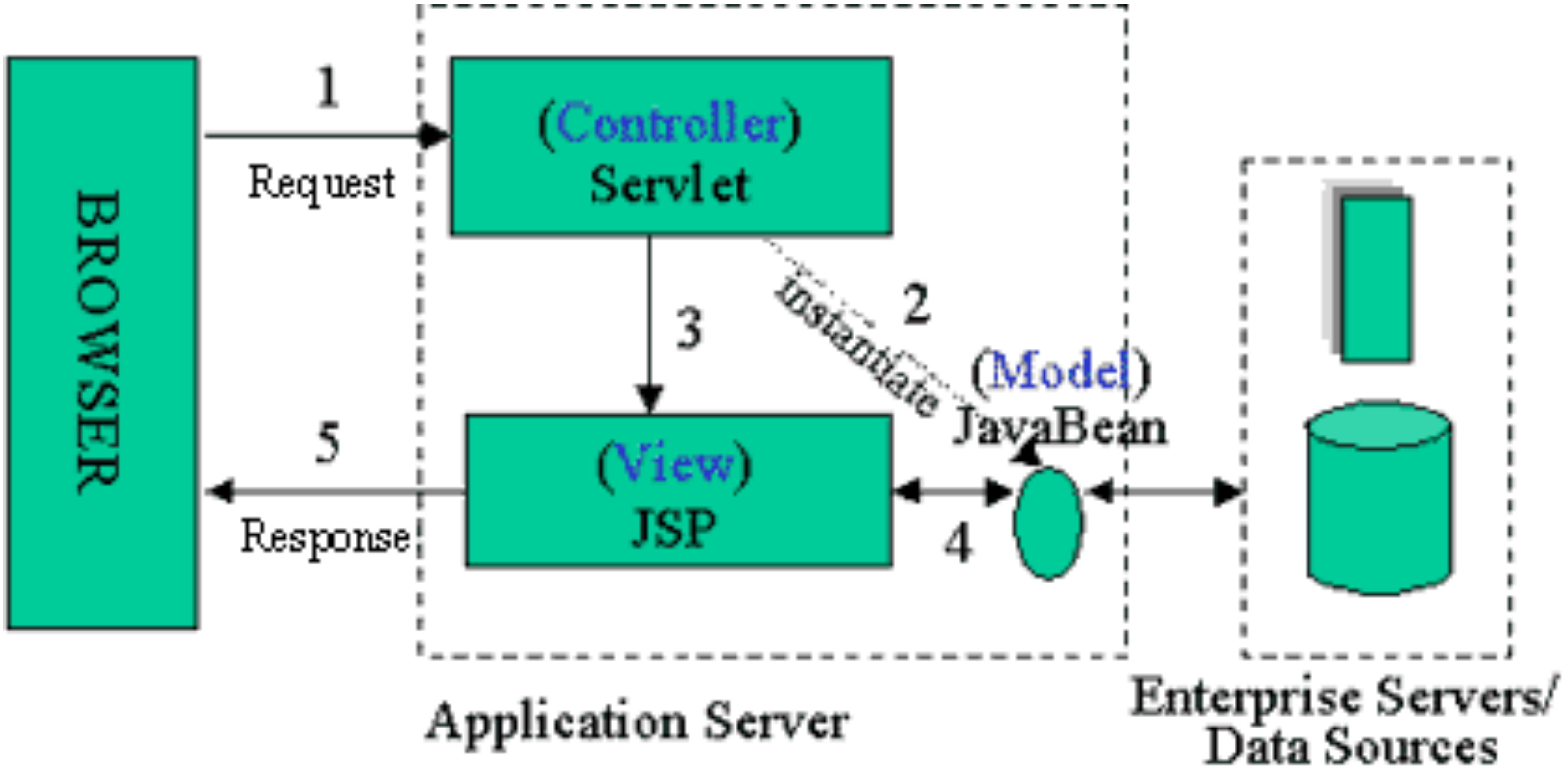
JSP Problem - Model 1 Architecture

- JSP page alone is responsible for processing the incoming request and replying back to the client
- There is still separation of presentation from content, because all data access is performed using beans
- Not suitable for complex application - usually leads to a significant amount of scriptlets or Java code embedded within the JSP page - crowded and unmanageable
- Ultimately, it may even lead to an unclear definition of roles and allocation of responsibilities, causing easily avoidable project-management headaches

Problem Solving – Combine JSP & Servlet

- Using Model View Controller Design Pattern

JSP Model 2 Architecture - MVC Design Pattern



JSP Model 2 Architecture - Perfect MVC

- **Servlet (The controller)**
  - 1 for receiving form data / GET data
  - 2 data processing
  - 2 instantiate and put data into the JavaBeans
  - 2 put JavaBeans into sessions – 3 JSP redirection
- **JSP (The view)**
  - 4 Reading JavaBeans (**The model**) from session
  - 5 Format and present JavaBeans data to the user in HTML form
- **Killing two birds using one stone**
  - Removing out.println from the servlet
  - Removing java code for data processing at JSP

JSP Model 2 Architecture - Perfect MVC

- **Conclusions**
  - hybrid approach for serving dynamic content, since it combines the use of both servlets and JSP
  - Using predominant strengths of both technologies,
  - using JSP (***view***)
    - to generate the presentation layer
  - and servlets (***controller***)
    - to perform process-intensive tasks
    - processing / creation of any beans (***model***)or objects used by the JSP (***view***)
    - deciding, depending on the user's actions, which JSP page to forward the request to
  - and JavaBeans (**model**) as data encapsulator

JSP Model 2 Architecture - Perfect MVC

- **Note** that there is no processing logic within the JSP page itself;
- it is simply responsible for retrieving any objects or beans that may have been previously created by the servlet,
- and extracting the dynamic content from that servlet for insertion within static templates
- **VCD_MVC**
- **MVC**

Authenticating, Access Control & Profile Management

- **Using FORM authentication**
  - Supplying login and password through HTML Form to log to the restricted application
  - Data send to servlet using SSL protocol – prevent from sniffer
- **Servlet for login-password processing and JSP redirecting**
  - Authenticating user login and password from the database
  - Creating user session
  - Creating user profile using JavaBeans and store in the user newly created session
  - Direct user to the session protected JSP pages

Authenticating, Access Control & Profile Management

- **JSP pages**
  - Control user access to protected resources using user session
  - Every JSP pages which involve in the restricted application should also have a section for session authentication
  - **Website (netbeans)**

- Complete System using MVC
  - Cash (netbeans)