OBJECT™ TECHNOLOGIES

JSP I

What will be covered

OBJECT TECHNOLOGIES

- What is JSP
- Need of JSP
- Limitations of Servlets
- JSP Complementary for servlets
- JSP life cycle
- JSP elements
- First simple page in JSP

What is JSP

OBJECT TECHNOLOGIES

- Java Server Pages (JSP) is a server side technology for developing dynamic web pages.
- This is mainly used for implementing presentation layer (GUI Part) of an application.
- A complete JSP code is more like a HTML with bits of java code in it.
- JSP is an extension of servlets and every JSP page first gets converted into servlet by JSP container before processing the client's request.
- JSP files has an extension of .jsp.

Need of JSP

- Initially only servlet technology was available to genarate dymanic pages of a web application.
- Servlets generate web pages by embedding HTML directly in programming language code. This pushes the creation of dynamic web pages exclusively into the realm of programmers.
- The JSP technology--which abstracts servlets to a higher level--is an open, freely available specification developed by Sun Microsystems as an alternative to Microsoft's Active Server Pages (ASP) technology, and a key component of the Java 2 Enterprise Edition (J2EE) specification.
- Many of the commercially available application servers (such as BEA WebLogic, IBM WebSphere, Live JRun, Orion, and so on) support JSP technology.

Limitations of Servlets

OBJECT TECHNOLOGIES

- Servlet is a picture of both Presentation logic(HTML) and Business Logic(Java) . At the time of development of Servlet by using both of the above (Presentation and Business Logic), it may become imbalance, because a Servlet developer can't be good in both Presentation logic and Business logic.
- Servlet never provides separation between or clarity between Presentation Logic and Business Logic. So that servlets do not give Parallel Development.
- If we do any changes in a servlet, then we need to do Re-deployment process i.e. Servlets modifications requires redeployment, which is one of the time-consuming process.
- The presentation logic (HTML code) will be mixed up with Java code (pw.println()). Lots of coding is written inside pw.println() to write html code. Writing HTML code in Servlet programming is a complex process and it makes Servlet looks bulky.
- No editor can support writing HTML inside pw.println() statement.

JSP Complementary for servlets

OBJECT

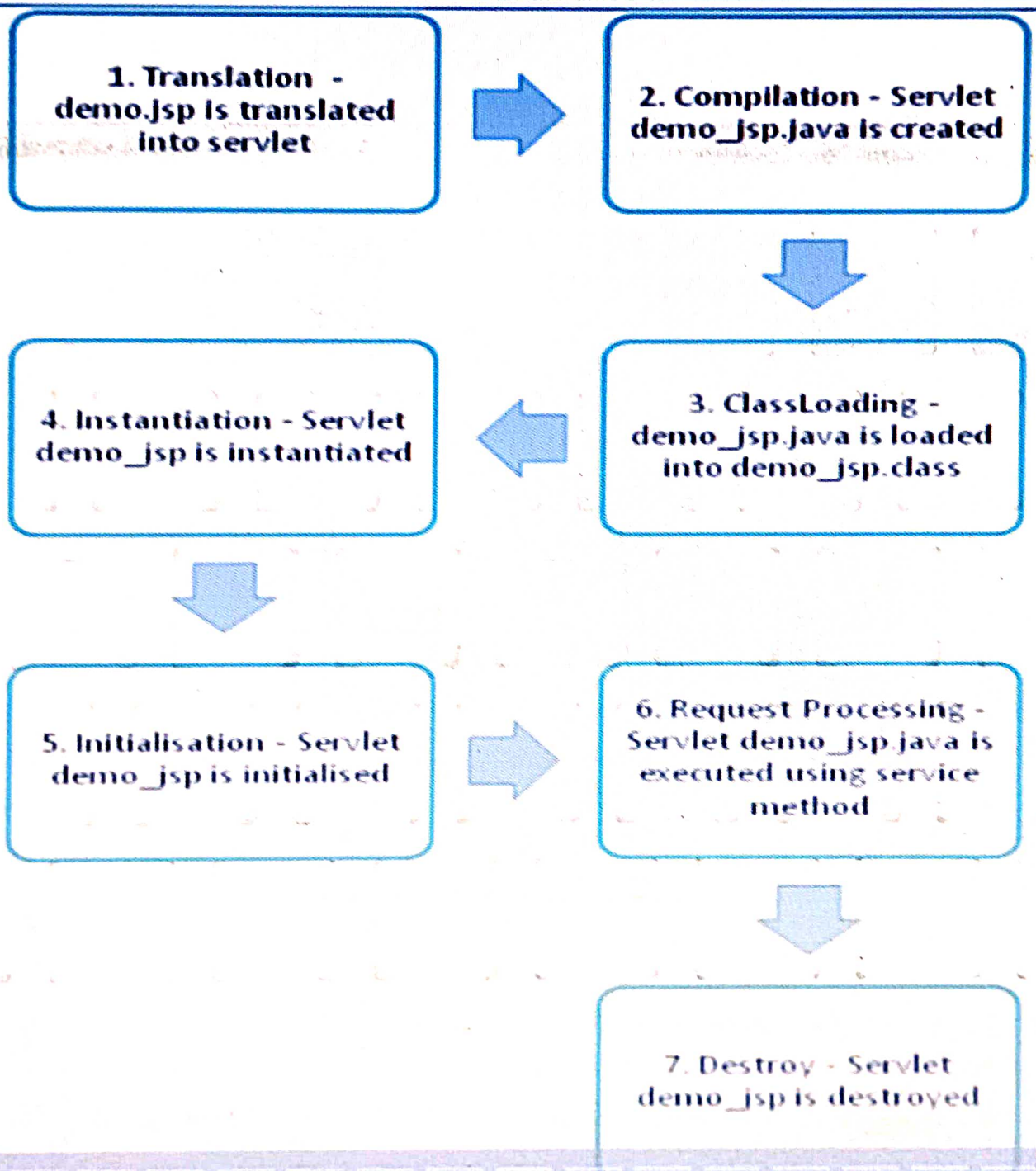
- When Sun introduced JSP software, some were quick to claim that Servlets had been replaced as the preferred request-handling mechanism in Web-enabled enterprise architectures.
- Although JSP technology will be a powerful successor to basic Servlets, the two have an evolutionary relationship and can be used in a cooperative and complementary manner.
- Purpose of JSP technology was never to replace existing servlet technology.
- Whenever business logic is more, servlets are preferred and when view is more, JSP is used.
- Virtually servlets and JSP both contain java and HTML but we should see to it that servlets should have less/no HTML and JSP should have less/no Java.

JSP life cycle

OBJECT TECHNOLOGIES

- JSP Life Cycle is defined as translation of JSP Page into servlet as a JSP Page needs to be converted into servlet first in order to process the service requests. The Life Cycle starts with the creation of JSP and ends with the disintegration of that.
- When the browser asks for a JSP, JSP engine first checks whether it needs to compile the page. If the recent modification is done in JSP after it's previous compilation, then the JSP engine compiles the page.
- Compilation process of JSP page involves three steps:
  - Parsing of JSP
  - Turning JSP into servlet
  - Compiling the servlet

JSP life cycle



JSP life cycle

- **Translation of the JSP Page:**
- A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.
- **Compilation of the JSP Page**
- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class
- **Classloading**
- Servlet class that has been loaded from JSP source is now loaded into the container

JSP life cycle

- **Instantiation**
- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.
- **Initialization**
- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

JSP life cycle

OBJECT TECHNOLOGIES

- **Request processing**
- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.
- **Destroy**
- _jspdestroy() method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files

JSP elements

- There are three types of JSP elements you can use: directive, action, and scripting. A new construct added in JSP 2.0 is an Expression Language (EL) expression; let's call this a forth element type, even though it's a bit different than the other three.
- **Scripting elements :**
- Java provides various scripting elements that allow you to insert Java code from your JSP code into the servlet. Scripting elements have different components that are allowed by JSP.
- **Directive elements :**
- The directive elements, specify information about the page itself for example, if session tracking is required or not, buffering requirements, and the name of a page that should be used to report errors, if any.
- **Action elements :**
- Action elements typically perform some action based on information that is required at the exact time the JSP page is requested by a browser

JSP elements

- **HTML tags**
- <h1> Welcome to JSP </h1>
- Any valid tag in HTML is valid to be written in JSP. At the time of translation phase, html tags get translated into out.write() statement which gets added in service method of generated servlet.
- **HTML comment**
- <!-- This is HTML comment -->
- HTML comments are converted into out.write() statement which gets added in service method of generated servlet. It travels from server to client but it is not getting displayed in clinet's browser
- **JSP Comment**
- <%-- This is JSP comment --%>
- JSP comments do not have any place in generated servlet.

JSP elements

- **Template text**

  JSP is complementary for servlet
- Text without any markups is considered as template text. It is translated similarly like HTML tags
- **Scriptlets**

```
<%
    out.println("Your IP address is " + request.getRemoteAddr());
%>
```

- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- At the time of translation phase, only symbols are removed and content of scriptlet are added in service method of generated servlet

JSP elements

- **Expressions**
- <%= expression %>
- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.
- At the time of compilation, expressions get converted to out.print() statement and added in the service method of generated servlet.
- Today's date: <%= (new java.util.Date()).toLocaleString()%>
- **JSP Declarations**
- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file

JSP elements

- <%! declaration; [declaration;]+ ... %>.
- The jsp declaration tag can declare variables as well as methods.
- The declaration of jsp declaration tag is placed outside the _jspService() method in the generated servlet class.
- **JSP Directives**
- A JSP directive affects the overall structure of the servlet class. It usually has the following form –
- <%@ directive attribute="value" %>
- There are three types of directive tag – page, include, taglib.
- Directives can have a number of attributes which can be listed down as key-value pairs and separated by spaces.
- **Std Actions**
- **EL**
- **JSTL**
- **Custom tags**

JSP elements

- Basically, Page directives are used to give the instructions to the JSP compiler like what package to import, what language we are writing, etc. Page directives are basically used for supplying compile-time information to the container for generating a servlet. Page directive should be first statement in JSP. The scope of the page directive is applicable to the current JSP page only. JSP program allows only one-page directive at a time.
- The include directive is employed to incorporate a file during the interpretation phase. This directive tells the container to merge the content of other external files with the present JSP. This tag is given for code inclusion, not for output inclusion.
- The main purpose of Taglib Directives is to make available user-defined tag library into the present JSP pages.

First simple page in JSP

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="java.util.Calendar"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1> Welcome to JSP </h1>
    This is template text <br/>
    <!--  This is HTML comment -->
    <%-- This is JSP comment. No place in generate servlet --%>
    <br/>
    <%
        int n = 0;
        out.println("n = "+(++n));
    %>
    <br/>
    <%= Calendar.getInstance().getTime().toString() %>
    <%! int num; %>
    <br/>
    <%= "num = "+(++num) %>
</body>
</html>
```

Std Actions

What is JSP Action?

1. JSP actions use the construct in XML syntax to control the behavior of the servlet engine.
2. We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions like include and forward.
3. Unlike directives, actions are re-evaluated each time the page is accessed.

Syntax:

```
<jsp:action_name attribute="value" />
```

The JSP Standard Actions are used within the JSP page and are used to eliminate or remove scriptlet code in a JSP page because now-a-days, scriptlet tag is not recommended.

The standard tags are as follows.

1. jsp:include
2. jsp:forward

<jsp:include> :

Instead of loading the text of the included file in the original file like "include directive", it actually calls the included target at run-time (the way a browser would call the included target. In practice, this is actually a simulated request rather than a full round-trip between the browser and the server).

Following is an example of jsp:include usage :

includestandardaction.jsp

```
<HTML>
<BODY>
Going to include hello.jsp...<BR>
<jsp:include page="hello.jsp"/>
</BODY>
</HTML>
```

Following is an example of jsp:include usage :

includestandardaction.jsp

```
<HTML>
<BODY>
Going to include hello.jsp...<BR>
<jsp:include page="hello.jsp"/>
</BODY>
</HTML>
```

<jsp:forward> :

The <jsp:forward> element forwards the request object containing the client request information from one JSP file to another file. The target file can be an HTML file, another JSP file, or a servlet, as long as it is in the same application context as the forwarding JSP file. The lines in the source JSP file the <jsp:forward> element are not processed.

This sample code shows the use of tag. This checks the percentage of free memory and based on that opens new page using this tag.

checkmemory.jsp

```
<html>
<%
double freeMemory = Runtime.getRuntime().freeMemory();
double totalMemory = Runtime.getRuntime().totalMemory();
double percent = freeMemory/totalMemory;
if(percent<0.5){
%>
<jsp:forward page="one.jsp"/>
<%}else{%>
```

checkmemory.jsp

```
<html>
<%
double freeMemory = Runtime.getRuntime().freeMemory();
double totalMemory = Runtime.getRuntime().totalMemory();
double percent = freeMemory/totalMemory;
if(percent<0.5){
%>
<jsp:forward page="one.jsp"/>
<%}else{%>
<jsp:forward page="two.html"/>
<%}%>
</html>
```

one.jsp

```
<html>
<body>
<font color="red">
VM Memory usage less than 50 percent
</font>
</body>
</html>
```

two.jsp

```
<html>
<body>
<font color="red">
VM Memory usage greater than 50 percent
```

```
<html>
<body>
<font color="red">
VM Memory usage greater than 50 percent
</font>
</body>
</html>
```

Bean Related Std Actions

JSP Standard action are predefined tags which perform some action based on information that is required at the exact time the JSP page is requested by a browser. An action can, for instance, access parameters sent with the request to do a database lookup. It can also dynamically generate HTML, such as a table filled with information retrieved from an external system.

They can be used to include a file at the request time, to find or instantiate a JavaBean, to forward a request to a new page, to generate a browser-specific code, etc. The JSP standard actions affect the overall runtime behavior of a JSP page and also the response sent back to the client.

Given below some action elements :

<jsp:useBean>	Makes a JavaBeans component available in a page
<jsp:getProperty>	Gets a property value from a JavaBeans component and adds it to the response
<jsp:setProperty>	Sets a JavaBeans component property value
<jsp:include>	Includes the response from a servlet or JSP page during the request processing phase
<jsp:forward>	Forwards the processing of a request to servlet or JSP page
<jsp:param>	Adds a parameter value to a request handed off to another servlet or JSP page using <jsp:include> or <jsp:forward>
<jsp:plugin>	Generates HTML that contains the appropriate browser-dependent elements (OBJECT or EMBED) needed to execute an applet with the Java Plug-in software

Examples

Examples

<jsp:useBean> Action

1. This action is useful when you want to use Beans in a JSP page, through this tag you can easily invoke a bean.
2. The jsp: setProperty activity tag is utilized to set property estimations in a bean utilizing setter method. The setProperty uses java bean for the development of web application.

Syntax

```
<jsp:setProperty name=?instanceOfBean? property= ?*?   |
property=?propertyName? param=?parameterName?  |
property=?propertyName? value=?{ string | <%= expression %>}? />
```

The jsp: getProperty action tag is utilized to get property estimations in a bean utilizing getter strategy. This getProperty is used for web application with java bean.

Syntax

```
<jsp:getProperty name=?instanceOfBean? property=?propertyName? />
```

Syntax of <jsp:useBean>:

```
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name" />
```

Example of <jsp:useBean>, <jsp:setProperty> & <jsp:getProperty>:

Once Bean class is instantiated using above statement, you have to use jsp:setProperty and jsp:getProperty actions to use the bean?s parameters. we will see both setProperty and getProperty after this action tag.

EmployeeBeanTest.jsp

```
<html>
  <head>
    <title>JSP Page to show use of useBean action</title>
  </head>
  <body>
```

EmployeeBeanTest.jsp

```
<html>
  <head>
    <title>JSP Page to show use of useBean action</title>
  </head>
  <body>
    <h1>Demo: Action</h1>
    <jsp:useBean id="student" class="javabeansample.StuBean"/>
    <jsp:setProperty name="student" property="*"/>
    <h1>
      name:<jsp:getProperty name="student" property="name"/><br>
      empno:<jsp:getProperty name="student" property="rollno"/><br>
    </h1>
  </body>
</html>
```

StudentBean.java

```
package javabeansample;
public class StuBean {
  public StuBean() {
  }
  private String name;
  private int rollno;
//get/set

}
```

<jsp:setProperty> Action

This action tag is used to set the property of a Bean, while using this action tag, you may need to specify the Bean?s unique name (it is nothing but the id value of useBean action tag).

<jsp:setProperty> Action

This action tag is used to set the property of a Bean, while using this action tag, you may need to specify the Bean?s unique name (it is nothing but the id value of useBean action tag).

syntax of <jsp:setProperty>

```
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name" />
....
....
<jsp:setProperty name="unique_name_to_identify_bean" property="property_name" />
OR
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name">
....
....
<jsp:setProperty name="unique_name_to_identify_bean" property="property_name" />
</jsp:useBean>
```

In property_name, you can also use ?*?, which means any request parameter which matches to the Bean?s property will be passed to the corresponding setter method.

<jsp:getProperty> Action

It is used to retrieve or fetch the value of Bean?s property.

syntax of <jsp:getProperty>

```
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name" />
....
....
<jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />
OR
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name">
....
....
<jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />
</jsp:useBean>
```

<jsp:plugin> Action

This tag is used when there is a need of a plugin to run a Bean class or an Applet.

MVC Architecture

What is MVC?

MVC is an architecture that separates business logic, presentation and data. In MVC,

M stands for Model
V stands for View
C stands for controller.

MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message.

Model Layer:

1. This is the data layer which consists of the business logic of the system.
2. It consists of all the data of the application
3. It also represents the state of the application.
4. It consists of classes which have the connection to the database.
5. The controller connects with model and fetches the data and sends to the view layer.
6. The model connects with the database as well and stores the data into a database which is connected to it.

View Layer:

1. This is a presentation layer.
2. It consists of HTML, JSP, etc. into it.
3. It normally presents the UI of the application.
4. It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.
5. This view layer shows the data on UI of the application.

Controller Layer:

1. It acts as an interface between View and Model.
2. It intercepts all the requests which are coming from the view layer.
3. It receives the requests from the view layer and processes the requests and does the necessary validation for the

Controller Layer:

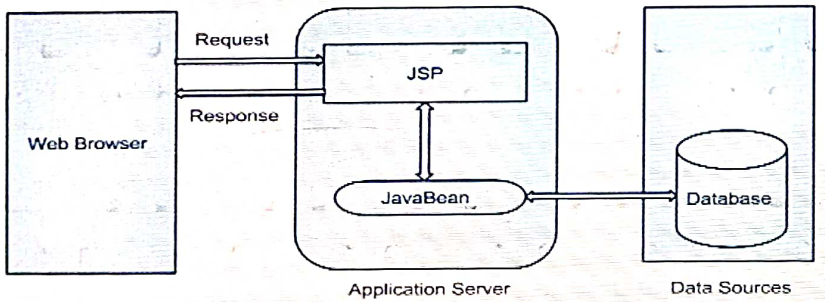
1. It acts as an interface between View and Model.
2. It intercepts all the requests which are coming from the view layer.
3. It receives the requests from the view layer and processes the requests and does the necessary validation for the request.
4. This requests is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view.

Two types of development models are used for web applications.

1. **Model-1 architecture**
2. **Model-2 architecture**

Model-1 Architecture

Servlet and JSP are the main technologies to develop the web applications. This model uses JSP to design applications which is responsible for all the activities provided by the application. The following figure shows Model-1 architecture.



In Model-1 architecture web applications are developed by combining both business and presentation logic. In this model, JSP pages receive requests which are transferred to data sources by JavaBeans. After the requests are serviced, JSP pages send

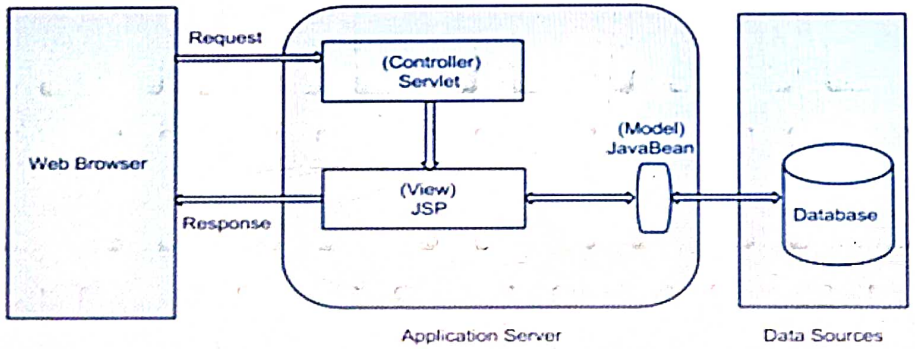
In Model-1 architecture web applications are developed by combining both business and presentation logic. In this model, JSP pages receive requests which are transferred to data sources by JavaBeans. After the requests are serviced, JSP pages send responses back to client.

Disadvantages

1. This architecture is not suitable for large applications.
2. It increases the complexity of the program so it is difficult to debug the program.
3. It needs more time to develop custom tags in JSP.
4. A single change in one page needs changes in other pages. So it is difficult to maintain.

Model-2 (MVC) Architecture

The Model-2 architecture is based on MVC design model i.e. Model, View and Controller.
Following figure shows Model-2 Architecture.



As shown in above model Servlet acts as Controller, JSP acts as View and JavaBean acts as Model. In this model, Controller receives requests which are transferred to data sources by View component and Model component. After the requests are

As shown in above model Servlet acts as Controller, JSP acts as View and JavaBean acts as Model. In this model, Controller receives requests which are transferred to data sources by View component and Model component. After the requests are serviced, JSP pages i.e. View component send responses back to client.

The advantages of MVC are:

1. Easy to maintain
2. Easy to extend
3. Easy to test
4. Navigation control is centralized

MVC Disadvantage

If we change the code in controller, we need to recompile the class and redeploy the application.

Example of MVC architecture

1. In this example, we are going to show how to use MVC architecture in JSP.
2. We are taking the example of a form with two variables "email" and "password" which is our view layer.
3. Once the user enters email, and password and clicks on submit then the action is passed in mvc_servlet where email and password are passed.
4. This mvc_servlet is controller layer. Here in mvc_servlet the request is sent to the bean object which act as model layer.
5. The email and password values are set into the bean and stored for further purpose.
6. From the bean, the value is fetched and shown in the view layer.

Mvc_example.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>MVC Guru Example</title>
</head>
<body>
<form action="Mvc_servlet" method="POST">
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>MVC Guru Example</title>
</head>
<body>
<form action="Mvc_servlet" method="POST">
Email: <input type="text" name="email">
<br />
Password: <input type="text" name="password" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Mvc_servlet.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
// TODO Auto-generated method stub
String email=request.getParameter("email");
String password=request.getParameter("password");

TestBean testobj = new TestBean();
testobj.setEmail(email);
testobj.setPassword(password);
request.setAttribute("gurubean",testobj);
RequestDispatcher rd=request.getRequestDispatcher("mvc_success.jsp");
rd.forward(request, response);
}
```

TestBean.java

TestBean.java

```
public class TestBean implements Serializable{

public String getEmail() {
return email;
}
public void setEmail(String email) {
this.email = email;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}
private String email="null";
private String password="null";


}
```

Mvc_success.jsp

```
<body>

<%
TestBean testguru=(TestBean)request.getAttribute("gurubean");
out.print("Welcome, "+testguru.getEmail());
%>
</body>
```

Assignments

1. Accept information from the user about about name, emailid, conatctno and adress. Keep all the information in the request

Mvc_success.jsp

```
<body>

<%
TestBean testguru=(TestBean)request.getAttribute("gurubean");
out.print("Welcome, "+testguru.getEmail());
%>
</body>
```

Assignments

1. Accept information from the user about about name, emailid, conatctno and adress. Keep all the information in the request level attribute and display in another JSP.(Use std actions)
2. Repeat above assignment with MVC implementation using servlet for creating the attribute and JSP for showing the properties.
3. Develop a blogging application with follwing functionality.
   Display all the available topics as in the form of hyperlinks.
   When user clicks the topic, then display all the comments made by all users on that topic. Controller will communicate with database to fetch the information about all comments on a perticular topic.
   Note : Use MVC architecture. Do not generate response in servlet. Response should be generated in JSP using std actions or EL or JSTL