JAVA SERVER PAGES AND MODEL VIEW CONTROLLER ARCHITECTURE

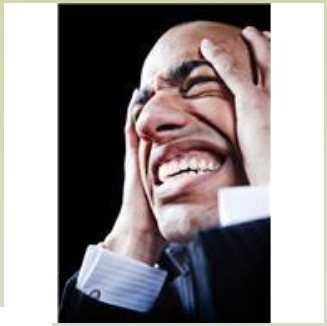
Knower, Known, and Process of Knowing

Main point 1

The web container generates a servlet from a JSP file the first time the JSP is requested from a web application. Since a JSP is essentially a servlet, one should understand servlets to effectively deal with JSPs. **Science of Consciousness:** Actions in accord with fundamental levels of knowledge promote success in dealing with more expressed values

Main idea of JSP

- separate display from processing, i.e., separate html from java
  - servlet is java with some html mixed in
    - a little plain java code
    - and lots of enclosed HTML
      - out.println(“ … \” … \” … “)
    - writing and maintaining quickly becomes a headache



  - Jsp is html with a little java mixed in
    - a little java code bracketed in
      - <% …. %> , etc
    - and lots of plain HTML

- There are two types of data in a JSP page:
  - Template Data: The static part, copied directly to response
    - static HTML
  - JSP Elements: The dynamic part, translated and executed by the container
    - typically generate additional HTML portions of the page
    - can execute arbitrary Java code

Four types of JSP elements:

- **Script**: embedded Java statements <% %> <%= %> …
- **Directive**: page level operations for the translation phase.

<%@ page %>

- **EL Expression**: more convenient and powerful than a JSP expression ${ } vs <%= %>
- **Action**: JSP “functions” that encapsulate some commonly used functionality <c:foreach />

JSP scripting elements:

- **Declaration**:
  - <%! Inserts instance variable and method declarations into servlet p292
  - <%! Java declaration statements %>
  - <%! int count = 0; %>
- **Scriptlet**:
  - Inserts Java statements inside service method
    - <% Java statements %>
    - <% count = count * 10; %>
    - <% inserts into the service method p282
- **Expression**:
  - expects a Java expression, which it puts inside ‘out.print’ in service method
  - <%= Java expression %>
  - wraps it inside a print statement p286
  - <%= ++count %> becomes … out.print(++count); … in service method
- **Comment**:
  - <%-- jsp comment --%> p302
  - contrast with HTML comment
    - <!– Comment -->
    - HTML comments get sent to the browser (part of the HTML)
    - JSP elements all processed by container and do not appear in generated HTML

JSP Demo

- Inspect the JSP demo code
- Implement and run
- Insert the scripting elements shown after step 6
- Find the generated servlet
- Find the inserted JSP statements
- Find the implicit objects

```
 public void _jspService(HttpServletRequest request, HttpServletResponse response)   throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
//SNIP
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
//SNIP
      out.write("        <title> Introduction to JSP demo – postback page </title>\n");
      out.write("\n");
      out.write("    </head>\n");
      out.write("\n");
      out.write("    <body>\n");
      out.write("\n");
      out.write("      <p>This is the postback message</p>\n");
      out.write("      ");
      out.write("\n");
      out.write("      The count is now:  \n");
      out.write("      ");
      out.print( ++count );
out.write("      ");
out.write("      <!-- This is an html comment inserted by the increment comment -->\n");
      out.write("      ");
 count = count * 10;
      out.write("\n");
```

JSP Demo

- Inspect the JSP demo code
- Implement and run
- Insert the scripting elements shown after step 6
- Find the generated servlet
- Find the inserted JSP statements
- Find the implicit objects

JSP Lifecycle

LOADING PHASE

1. JSP Page Translation

A java servlet file is generated from the JSP source file. The container validates the syntactic correctness of the JSP pages and tag files. The container interprets the standard directives and actions, and generates java code for a servlet.

2. JSP Page Compilation

The generated java servlet file is compiled into a java servlet class.

3. Class Loading

The java servlet class that was compiled from the JSP source is loaded into the container.

EXECUTION PHASE

1. Initialization

jspInit() method is called immediately after the instance is created. It is called only once during JSP life cycle.

2. _jspService()

This method is called for every request of this JSP during its life cycle.

3. jspDestroy()

The servlet completes its purpose and submits itself to servlet heaven (garbage collection).

jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.

http://javapapers.com/jsp/jsp-life-cycle-explain/

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

JSP Element – Directive

- message from a JSP page to the JSP container that controls processing of entire page.

```
 <%@ page import="java.util.Date" %>
<HTML>
<BODY>
<%     System.out.println( "Evaluating date now" );  Date date = new Date(); %>
Hello!  The time is now <%= date %>
</BODY>
</HTML>
```

```
<HTML>
<BODY>
Going to include hello.jsp...<BR>
<%@ include file="hello.jsp" %>
</BODY>
</HTML>
```

```
<%@ taglib uri="http://www.jspcentral.com/tags" prefix="public" %>
```

Main point 1

The web container generates a servlet from a JSP file the first time the JSP is requested from a web application. Since a JSP is essentially a servlet, one should understand servlets to effectively deal with JSPs. **Science of Consciousness:** Actions in accord with fundamental levels of knowledge promote success in dealing with more expressed values

Main point 2

An EL expression is a compact expression of a systematic evaluation of the page, request, session and application scopes. **Science of Consciousness:** The laws of nature are compact expressions that control the infinite diversity of the manifest creation.

EL (Expression Language)

- recall JSP “expression”
  - <%= myMovieList.get(i) %>
  - evaluates the expression and writes it out to the HTML page at its location on the page
  - <%= ((Person) request.getAttribute(“person”)).getDog().getName() %>
- recall that attributes are where web app stores model values
  - values computed in model and then accessed in page for display
    - no model code on the JSP page
- EL simplifies JSP expression syntax
  - ${person.dog.name}

High level description of EL

${something}

- container evaluates this as follows

- checks page scope for an attribute named "something",
  - if found use it.
- otherwise check request scope for an attribute named "something",
  - if found use it.
- otherwise check session scope for an attribute named "something",
  - if found use it
- otherwise check application scope for an attribute named "something",
  - if found use it.
- otherwise ignore the expression.

More detailed description

${firstThing}

- if firstThing is not an implicit EL object search page, request, session and application scopes until attribute "firstThing" is found

${firstThing.secondThing}

- if firstThing is a bean then secondThing is a property of the bean
- if firstThing is a map then secondThing is a key of the map

${firstThing[secondThing]}

- if firstThing is a bean then secondThing is a property of the bean
- if firstThing is a map then secondThing is a key of the map
- if firstThing is a List then secondThing is an index into the List

EL is “null friendly”

- if EL cannot find a value for the attribute it ignores it
  - caution
  - no warning or error message
- in arithmetic, treats null value as 0
  - could be a surprise
  - ${750 + myBankAccount.balance}
    - “Aaaahhh, where’s all my money!”
- in logical expressions, nulls become “false”
  - more surprises ??

Main point 2

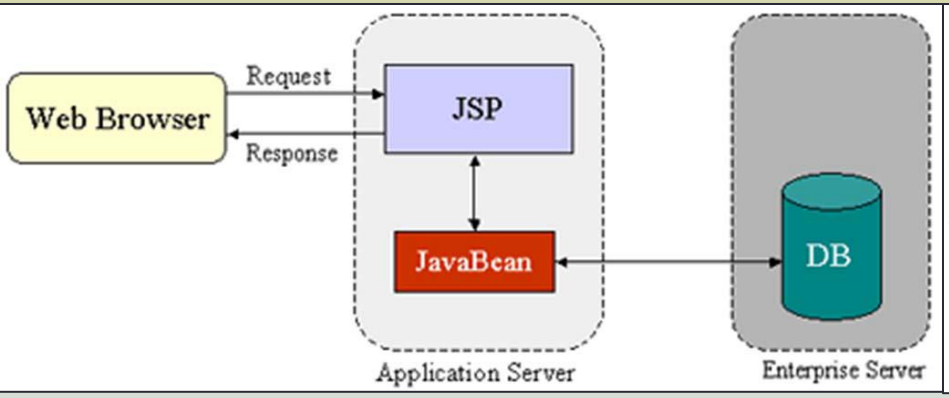
An EL expression is a compact expression of a systematic evaluation of the page, request, session and application scopes. **Science of Consciousness:** The laws of nature are compact expressions that control the infinite diversity of the manifest creation.

Main point 3

When you use JSP pages according to a Model 2 architecture, there is a servlet that acts as a controller (process of knowing) that sets attribute values based on computations and results from a business model (knower), then dispatches the request to the servlet generated by the JSP page (known). The JSP servlet then retrieves the attribute values and inserts them into the designated places in the HTML being sent to the browser. **Science of Consciousness:** Complete knowledge is the wholeness of knower, known, and process of knowing.

JSP Model 1 and Model 2 Architectures

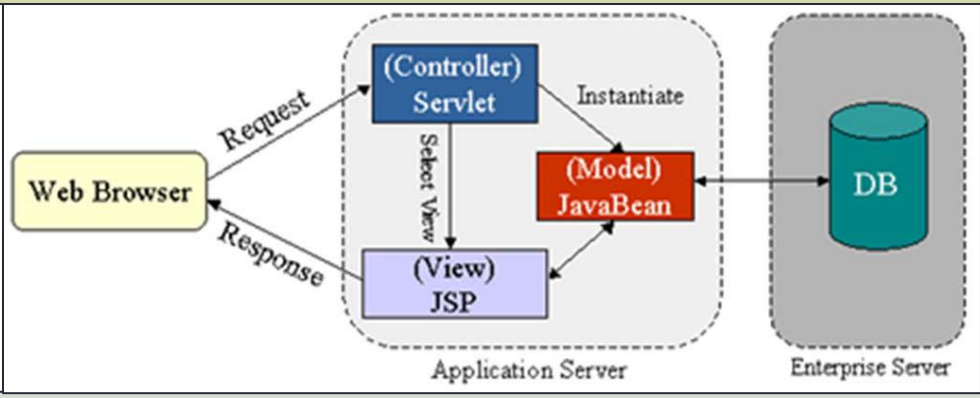
Model 1



- Simple architecture
- For small applications
- Issues

  - Pages are coupled, need to know about each other.
  - Expensive to provide another presentation for same application.
  - Java code in HTML

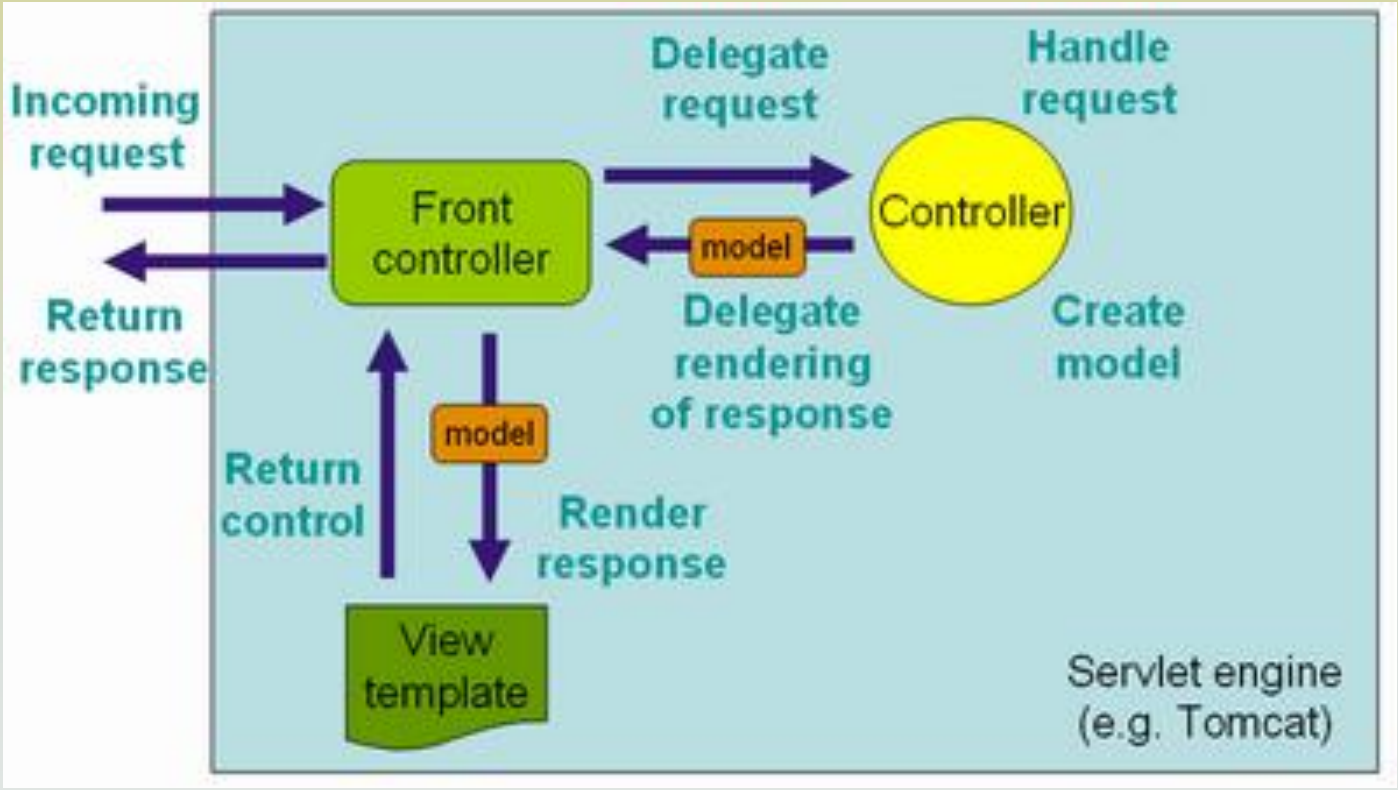
Model 2



- More complex architecture
- For medium and large applications
- Advantages

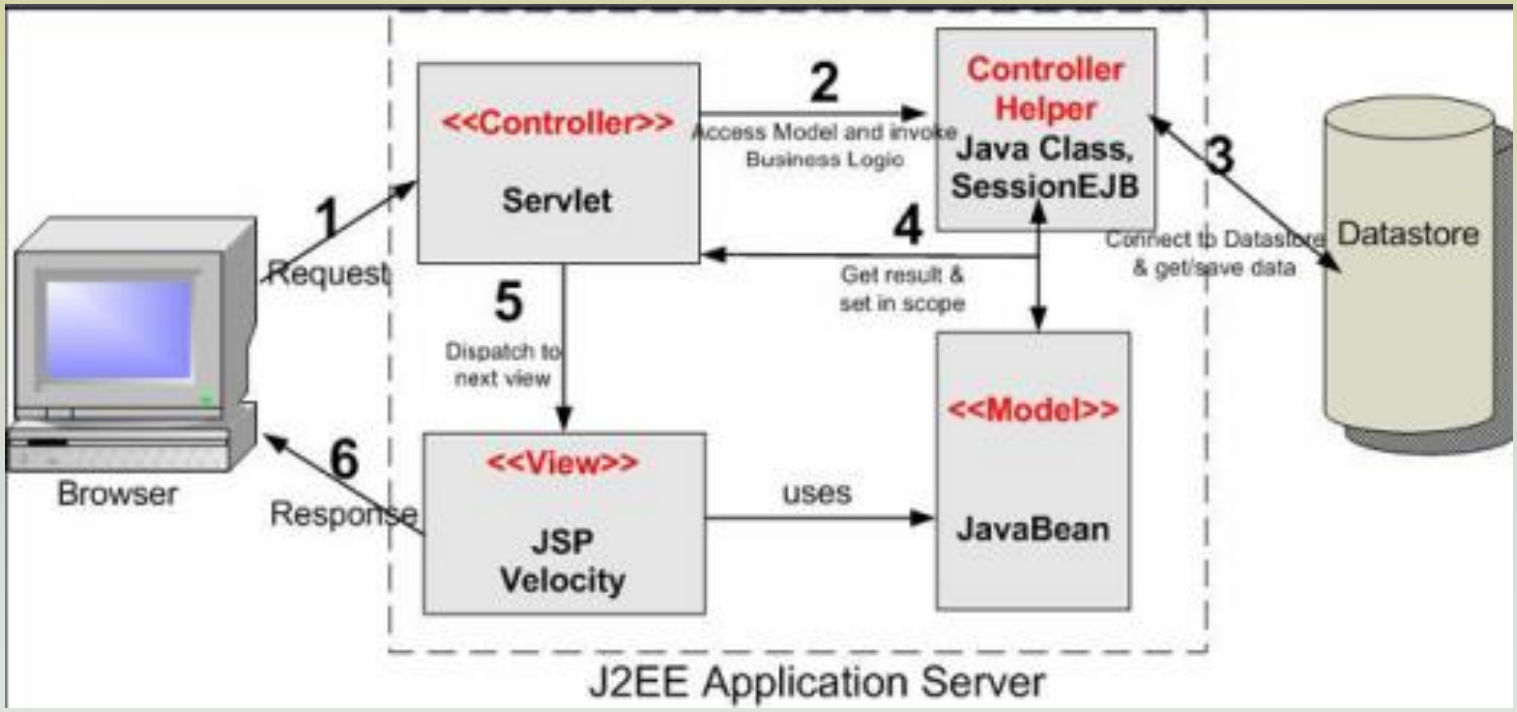
  - Each team can work on different pages. Easy to understand each page.
  - Separation of presentation from control, making it easy to change one without affecting the other.
  - no Java code in HTML

JSP Model 2 Architecture



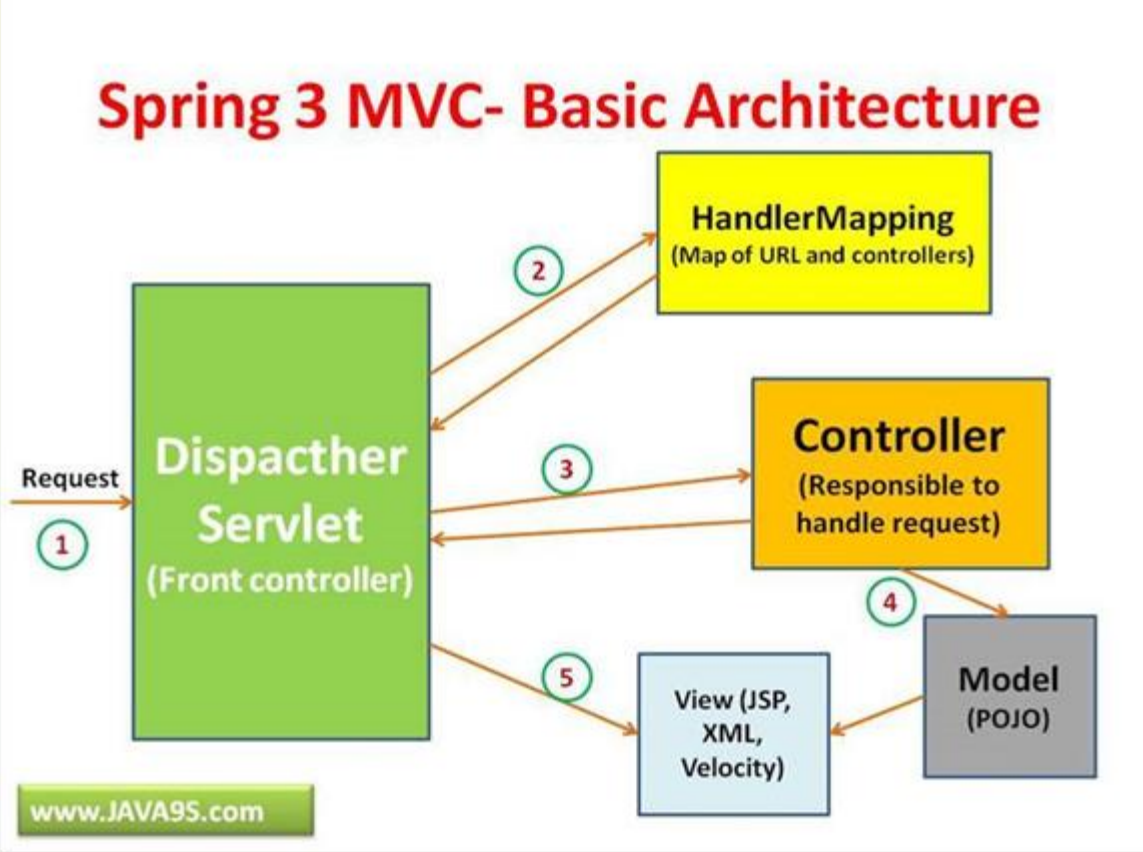
- http://stackoverflow.com/questions/20314098/spring-mvc-without-servlets
- SpringMVC site

JSP Model 2 Architecture



- http://jaiswaltraining.com/struts/ch1.php
- Struts site

JSP Model 2 Architecture



- http://java9s.com/spring-3-mvc/spring-3-mvc-introduction-spring-3-mvc-architecture

Model 1 vs 2 architecture (cont)

- Model 1:
  - JSP acts as both controller and view
  - JavaBean (POJO) is model
  - Problems:
    - JSPs became very complicated,
    - JSPs contains page navigation logic,
    - JSPs perform validation and conversion of string parameters
- Model 2:
  - Have a single controller servlet that takes requests
  - Gets request parameters
  - Converts and validates them
  - Calls object with business logic to do processing
  - Forwards results to jsp page for display
- http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Example model 1 JSP

```
<H1>Time JSP</H1>

<%
    //the parameter "zone" shall be equal to a number between 0 and 24 (inclusive)
    TimeZone timeZone = TimeZone.getDefault();  //returns the default TimeZone
    if (request.getParameterValues("zone") != null)
        {
           // gets a TimeZone. For this example we're just going to assume
           // its a positive argument, not a negative one.
           String timeZoneArg = request.getParameterValues("zone")[0];
           timeZone = TimeZone.getTimeZone("GMT+" + timeZoneArg + ":00");
        }
    /*
     since we're basing our time from GMT, we'll set our Locale to
     Brittania, and get a Calendar.
    */
    Calendar myCalendar = Calendar.getInstance(timeZone, Locale.UK);
%>

The current time is:
<%= myCalendar.get(Calendar.HOUR_OF_DAY) %>:
<%= myCalendar.get(Calendar.MINUTE) %>:
<%= myCalendar.get(Calendar.SECOND) %>
```

Main point 3

When you use JSP pages according to a Model 2 architecture, there is a servlet that acts as a controller (process of knowing) that sets attribute values based on computations and results from a business model (knower), then dispatches the request to the servlet generated by the JSP page (known). The JSP servlet then retrieves the attribute values and inserts them into the designated places in the HTML being sent to the browser. **Science of Consciousness:** Complete knowledge is the wholeness of knower, known, and process of knowing.

Demo 2

- inspect the demo steps
  - what HTTP message will be sent as a result of step 5?
  - what response will be sent as a result of that message?
  - What is the problem that step 8 wants you to fix? Why is it a problem?
  - Do you need to use an attribute?
  - What scope is required?