MODEL VIEW CONTROLLER ARCHITECTURE AND JAVA SERVER PAGES

Knower, Known, and Process of Knowing

Web App Architecture and Java Server Pages

- The need to separate the work of web designers from that of web programmers provided impetus for two further developments in servlet programming:
  - Design standard for web architectures that conforms to the MVC pattern. This separation of concerns gives expression to the fact that different laws of nature operate at different levels of creation.
  - The JSP technology which provides extra tags for a web page that provide dynamic content and can be used by web designers comfortably

MVC: Model-View-Controller

Separation of Concerns

Separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes

Model

Maintains the data[i.e. state] of the application domain

View

Manages the display of information.

Controller

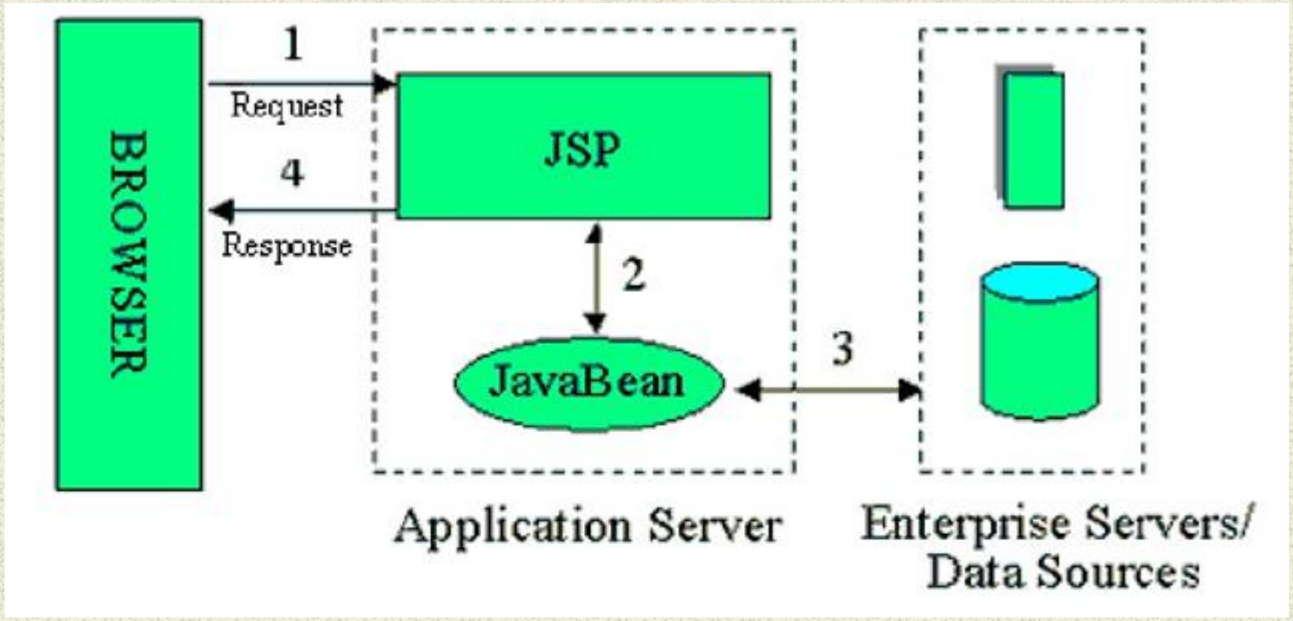
Manages user input triggering the model and/or the view to change as appropriate.

MVC I and MVC II

- Model I mixes view and business logic inside each handling servlet (or jsp). This makes it more difficult to change the view independently of that logic, and difficult to change business logic without changing the view.
- Model II cleanly separates business data and logic from the view, and the two are connected by way of a controller servlet. The model allows for multiple controllers/servlets, [e.g., one per GET/POST pair]. Typical MVC Framework implementations have just **one controller servlet** which centralizes common tasks.

JSP Model 1 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 separate controller servlet that takes requests

    Gets request parameters

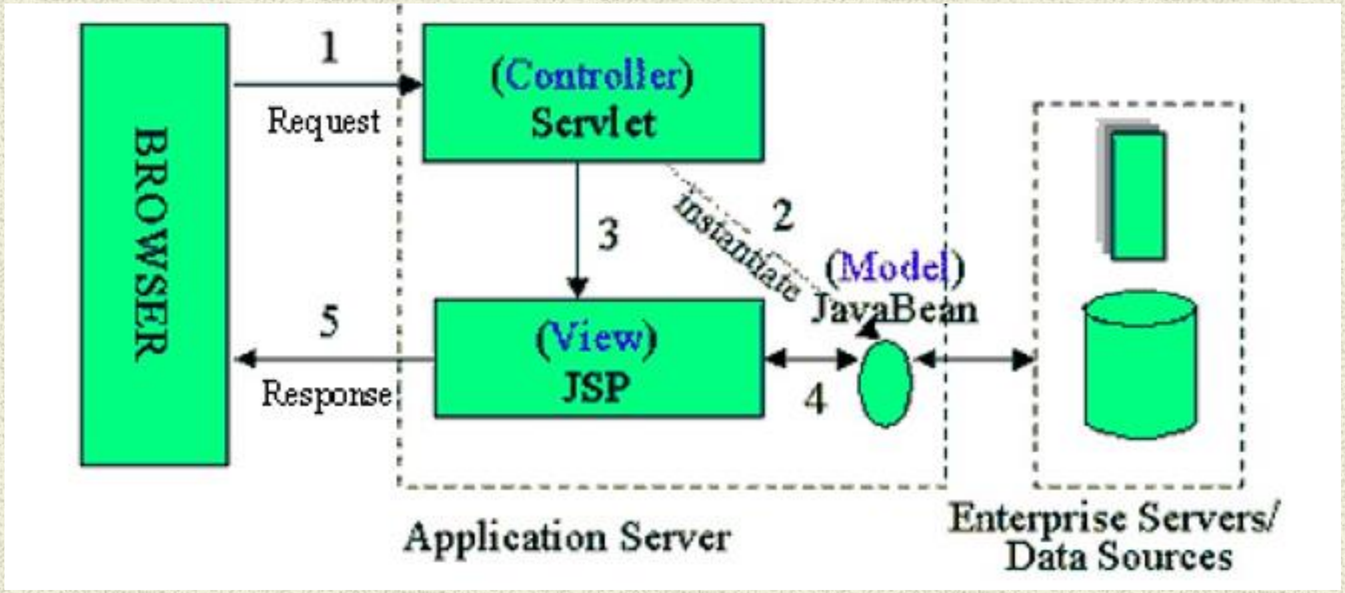
    Converts and validates them

    Calls object with business logic to do processing

  Forwards results to JSP for display

JSP Model 2 Architecture

- **Model View Controller**



DEMO

JSP Intro Demo :

MVC1 – MVC2 examples

Redirecting and forwarding requests

- Servlets may internally pass (“forward”) the request processing to another local resource or to tell the client browser to issue another HTTP request to a specified URL

- forwarding (to another servlet or jsp in same website):

```
RequestDispatcher view =
    request.getRequestDispatcher(“/hello.do”);
view.forward(request, response);
```

- redirect
  - //to another site
    - `response.sendRedirect(“http://www.mum.edu”)`
  - //within same site
    - `response.sendRedirect(“/hello.do”)`

Difference between redirect and forward

- **forward**
  - passes the request to another resource on the server
    - sometimes referred as “server side redirect”
  - request and response objects passed to destination servlet.
  - browser is completely unaware of servlet forward and hence the URL in browser address bar will remain unchanged
- **redirect**
  - server sends HTTP status code 3xx to client along with the redirect URL (usually 302 temporary redirect)
  - client then sends a new request to the URL
  - extra round trip
  - address bar will change to new URL
  - only http message sent, request and response objects cannot be sent

Urls Used in sendRedirect

- An absolute URL can be used:
  `response.sendRedirect("http://www.oreilly.com")` – this URL will appear in the browser address bar
- If a relative URL is used, there are two possibilities:
  - Without forward slash – the relative url is appended to the path specified in most recent request [i.e., folder/directory]:
    * most recent request:

http://localhost:8091/JSPIntroDemo/index.jsp

`response.sendRedirect("hello.jsp")`

* new url becomes (and this appears in address bar):

http://localhost:8091/JSPIntroDemo/Hello.jsp

(continued)

- With forward slash – the forward slash indicates an “absolute” path relative to the servlet container root [.vs. context root].
  - * previous request:

http://localhost:8091/JSPIntroDemo/index.jsp

Now do: `response.sendRedirect(“/Hello.jsp”)`

* new url becomes (and this appears in address bar):

http://localhost:8091/Hello.jsp

Urls Used in RequestDispatcher

There are two ways to obtain a RequestDispatcher, and the URLs that go with them are different

1. From a ServletRequest –
   - `request.getRequestDispatcher("result.jsp")`

     This will be relative to the location of the request.

   - `request.getRequestDispatcher("/result.jsp")`

     This will be relative to the webapp root

2. From a ServletContext – in this case the *relative* path is not allowed

   `getServletContext().getRequestDispatcher("/result.jsp")`

Main Point

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.

.

JavaServer Pages (JSP)

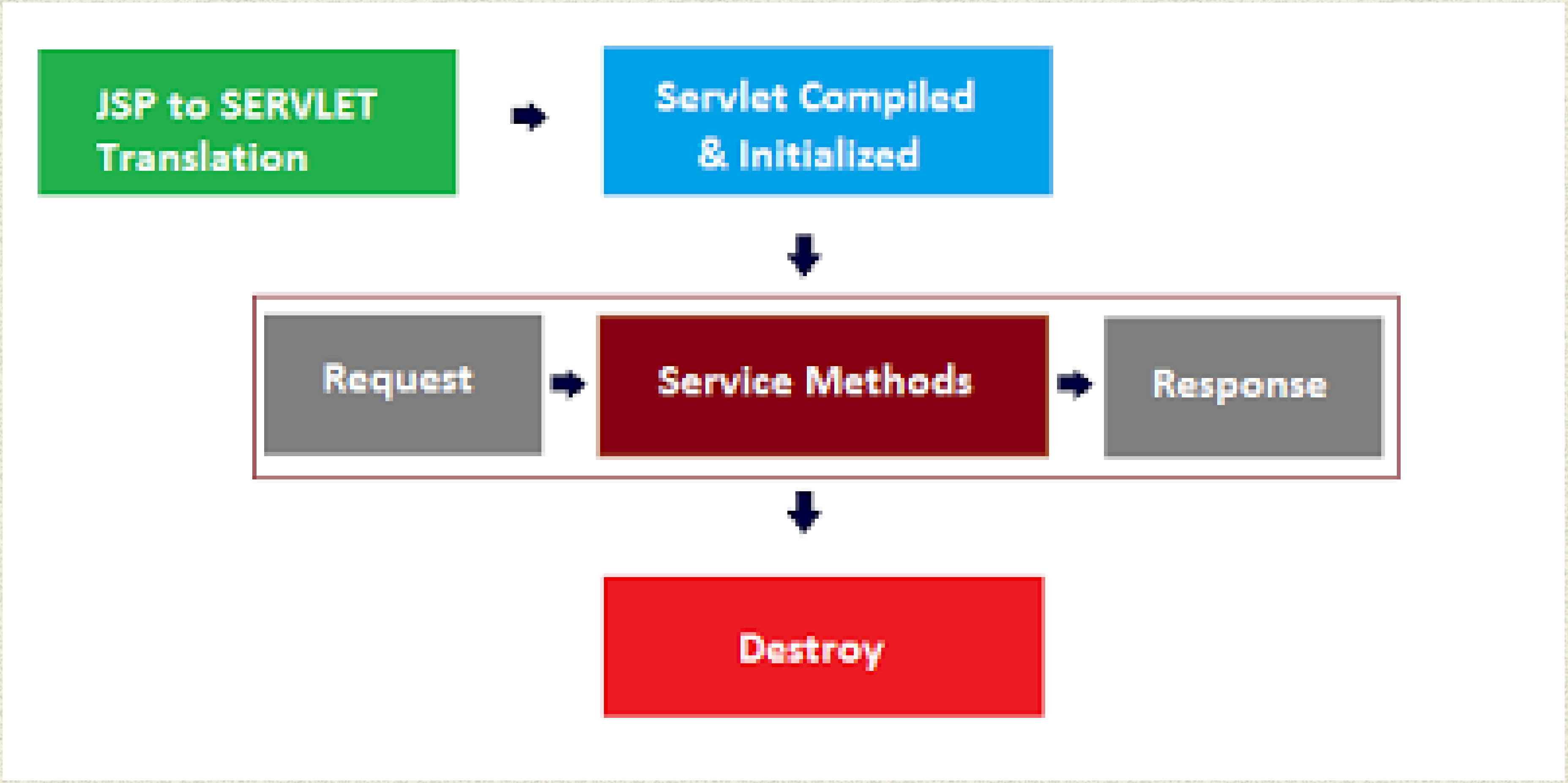
- Technology for developing web pages that support dynamic content
- Separate display from processing, i.e., separate html from java
- Developers insert java code in HTML pages
- Special JSP tags, most of which start with <% and end with %>.
- Eliminates the need to code complex output streams from the response object to produce a view

- **MODEL I MVC:**

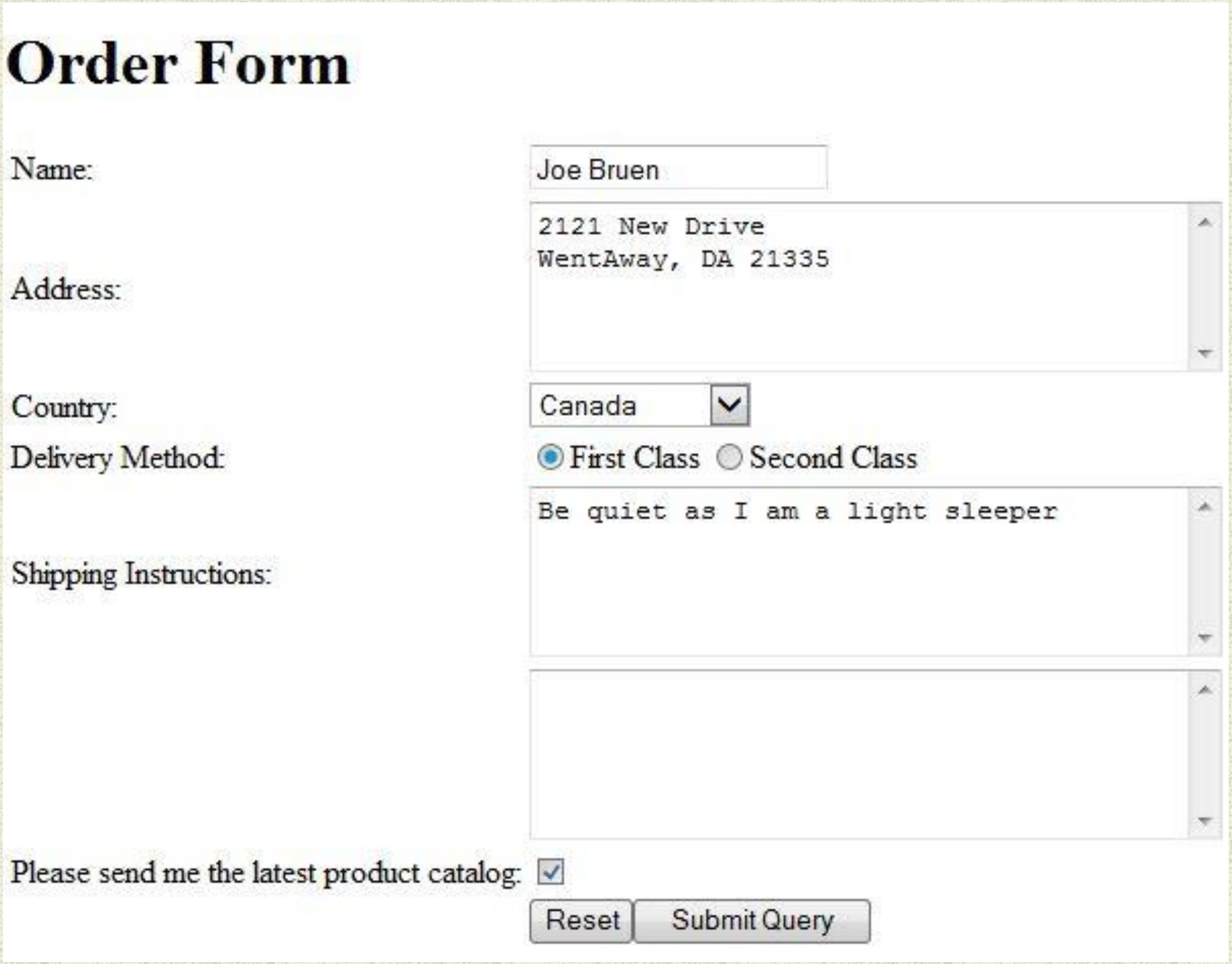
```
// invoke Hello.jsp
<form action="HelloName.jsp" method="get"><br/>
Name: <input type = "text" name = "name"/><br/>
<input type = "submit" value = "Submit"/>
</form>
```

```
//HelloName.jsp content
<body>
Hello, <%=request.getParameter("name") %>
</body>
```

JSP Lifecycle



A Simple Order Form "DEMO"



Order Form

Name:	Joe Bruen
Address:	2121 New Drive WentAway, DA 21335
Country:	Canada
Delivery Method:	[✓ First Class] Second Class
Shipping Instructions:	Be quiet as I am a light sleeper
Please send me the latest product catalog:	✓

```
•   public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse
  response)
•     try {
•       response.setContentType("text/html");
•       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("<!DOCTYPE html>\r\n");
•       out.write("<html>\r\n");
•       out.write("<head>\r\n");
•       out.write("<link rel=\"stylesheet\" type=\"text/css\" href=\"resources/mystyle.css\">\r\n");
•       out.write("<meta charset=\"ISO-8859-1\">\r\n");
•       out.write("<h4>Order</h4>\r\n");
•       out.write("</head>\r\n");
•       out.write("<body>\r\n");
•       out.write("\t<form  method=\"post\">\r\n");
•       out.write("\t\t <table> \r\n");
•       out.write("        \t<tr> \r\n");
•       out.write("        \t\t<td>Name:</td> \r\n");
•       out.write("        \t\t<td><input name='name'/></td>\r\n");
•       out.write("        \t</tr>     \t\r\n");
•       out.write("\t\t\t<tr>\r\n");
•       out.write("        \t\t<td>Address:</td> \r\n");
•       out.write("        \t\t<td><textarea name='address' cols='40' rows='5'></textarea></td> \r\n");
•       out.write("        \t</tr>  \t\r\n");
```

Types of JSP elements

- **Scripting Elements**: Embedded Java statements

```
<%!    %>      - declaration [Class level variables]
<%     %>      - scriptlet…essentially java code
<%=    %>      - expression…output statement
```

- **Directive**: Instructions for the container.

<%@ directive %> and XML variant: <jsp:directive   />

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" %>
```

[More] JSP Intro Demo

-

SCRIPLETS [UGH!]

BETTER THAN SERVLETS!!!

Main Point

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. *Actions in accord with fundamental levels of knowledge insure success in dealing with more expressed values.*

“Scripting considered harmful”

- JSP scripting originated in early days of web apps
- JSP now try to avoid scripting
- Might see in legacy code
- Might see something similar in other frameworks
  - PHP
  - ASP.net
  - ASPMVC.net
  - …?
- scripting part of “Model 1 JSP architecture”

Progress Toward Scriptless JSPs

- Tag Libraries
- There is a standard set of additional tags for JSP pages that encapsulate functionality of scriptlets (like for loops), thereby eliminating embedded code

- The JSP Expression Language (EL)
  - Syntax: ${person.name}
  - Left value is either an attribute or an implicit object (like request), right side is a property

JSP Expression Language [EL]

- An expression ${expr} prints expr to the page
- An expression ${person.name} evaluates (and prints to the page) something like: person.getName(), where person is an instance of a Person bean having a *name* variable.
- Has arithmetic and logical operators
- To test whether two objects are equal, EL uses **eq[or ==]** for equals and **ne[or !=]** for not equals
- Can use integers, floating point numbers, strings, the built-in constants true and false for boolean values, and null.

Expression Language Example

- Example: We have a person object stored as a request attribute. A person has an address and an address has a zip. How to retrieve zip?

- Scripting approach:

```
<%((pkg.Person)request.getAttribute(“person”)).
getAddress().getZip() %>
```

- EL approach:

```
${person.address.zip}
```

Main Point

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

.

JSTL – JSP Standard Tag Library

- Provides new tags for JSPs that reduce scripting

- They use the custom tag api, but have become a standard library, essentially a part of JSP language

Using JSTL

The JSTL library provides 5 kinds of tags, each having a different (standard) prefix. You “import” a library by placing a “taglib” directive at the top of your jsp page. Here are the choices:

- Core tags: (can do if, if/else, loops…)

```
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
```

- Function tags: (standard Java string manipulation)

```
<%@ taglib prefix="fn"
        uri="http://java.sun.com/jsp/jstl/functions" %>
```

`NOTE: the taglib directive has no XML syntax analog`

(continued)

- Format Tags (format numbers, dates..)

```
<%@ taglib prefix="fmt"
uri="http://java.sun.com/jsp/jstl/format" %>
```

- SQL Tags (set data source, perform queries)

```
<%@ taglib prefix="sql"
uri="http://java.sun.com/jsp/jstl/sql" %>
```

- XML Tags (for navigating/parsing/working with XML files)

```
<%@ taglib prefix="x"
uri="http://java.sun.com/jsp/jstl/xml" %>
```

(continued)

- Head First, p. 475 lists all the available JSTL library tags.
- DEMO: show how to use some of them…
  - c:set (set value of a variable)
  - c:out
  - c:if
  - c:choose
  - c:forEach
- Many of these make simple use of the EL.

Main Point

- The JSP Standard Tag Library provides convenient action tags for many common operations on a JSP page. JSTL combined with the EL could satisfy JSP needs. *Using a JSP action is analogous to the efficient use of some law of nature.*