UNIT : 6

Reference :-Marty Hall, Larry Brown, “Core Servlets and JavaServer Pages Volume – 1”, Pearson Education, 2nd ed.(2004)
Chapter :- 11 : Invoking Java Code with JSP Scripting Elements

Creating Template Text

- a large percentage of your JSP document consists of static text (usually HTML), known as template text.
- if you want to have <% or %> in the output, you need to put <\% or %\> in the template text.
- if you want a comment to appear in the JSP page but not in the resultant document, use

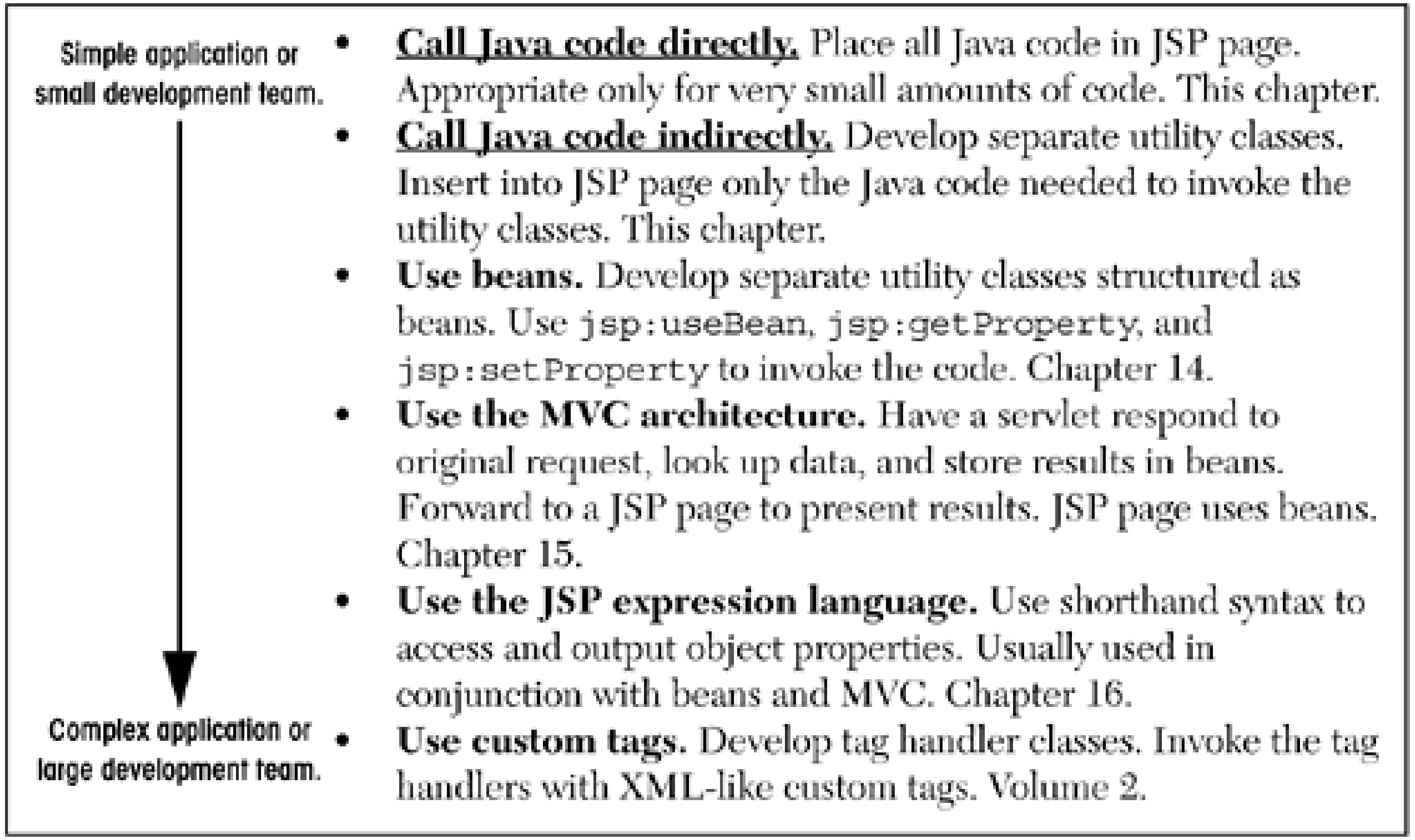
  <%-- JSP Comment --%>

  HTML comments of the form

  <!-- HTML Comment --> are passed through to the client normally.

Invoking JAVA Code from JSP

- **Strategies for invoking dynamic code from JSP**



Simple application or small development team.
Complex application or large development team.

- **Call Java code directly.** Place all Java code in JSP page. Appropriate only for very small amounts of code. This chapter.
- **Call Java code indirectly.** Develop separate utility classes. Insert into JSP page only the Java code needed to invoke the utility classes. This chapter.
- **Use beans.** Develop separate utility classes structured as beans. Use `jsp:useBean`, `jsp:getProperty`, and `jsp:setProperty` to invoke the code. Chapter 14.
- **Use the MVC architecture.** Have a servlet respond to original request, look up data, and store results in beans. Forward to a JSP page to present results. JSP page uses beans. Chapter 15.
- **Use the JSP expression language.** Use shorthand syntax to access and output object properties. Usually used in conjunction with beans and MVC. Chapter 16.
- **Use custom tags.** Develop tag handler classes. Invoke the tag handlers with XML-like custom tags. Volume 2.

Invoking JAVA Code from JSP

- The size and complexity of the project is the most important factor in deciding which approach is appropriate.

- Putting small amounts of Java code directly in JSP pages works fine for simple applications, using long and complicated blocks of Java code in JSP pages yields a result that is hard to maintain, hard to debug, hard to reuse, and hard to divide among different members of the development team

Type of JSP Scripting Elements

- JSP scripting elements let you insert Java code into the servlet that will be generated from the JSP page.

- Three forms:
- Expressions of the form <%= Java Expression %>, which are evaluated and inserted into the servlet's output.

- Scriptlets of the form <% Java Code %>, which are inserted into the servlet's _jspService method (called by service).

- Declarations of the form <%! Field/Method Declaration %>, which are inserted into the body of the servlet class, outside any existing methods.

Using JSP Expressions

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

  `<%= Java Expression %>`

- The expression is evaluated, converted to a string, and inserted in the page.

- This evaluation is performed at runtime (when the page is requested) and thus has full access to information about the request.

  For example, the following shows the date/time that the page was requested.

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

Predefined Variables

- request, the HttpServletRequest.
- response, the HttpServletResponse.
- session, the HttpSession associated with the request
- out, the Writer (a buffered version of type JspWriter) used to send output to the client.
- application, the ServletContext. This is a data structure shared by all servlets and JSP pages in the Web application and is good for storing shared data.

JSP / Servlet Correspondence

- JSP expressions basically become print (or write) statements in the servlet that results from the JSP page.

- Whereas regular HTML becomes print statements with double quotes around the text, JSP expressions become print statements with no double quotes.

- Instead of being placed in the doGet method, these print statements are placed in a new method called _jspService that is called by service for both GET and POST requests.

- Tomcat Autogenerated Servlet Source Code

  install_dir/work/Standalone/localhost/_ (The final directory is an underscore.

XML Syntax for Expressions

- XML authors can use the following alternative syntax for JSP expressions:
- <jsp:expression>Java Expression</jsp:expression>
- XML elements, unlike HTML ones, are case sensitive

Writing Scriptlets

- To do something more complex than output the value of a simple expression, JSP scriptlets let you insert arbitrary code into the servlet's _jspService method .

- Scriptlets have the following form:

  <% Java Code %>

- Scriptlets have access to the same automatically defined variables as do expressions (request, response, session, out, etc.).

Writing Scriptlets

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

  Or

- <% String queryData = request.getQueryString(); %> Attached GET data: <%= queryData %>

  Or

- Attached GET data: <%= request.getQueryString() %>

Writing Scriptlets

- scriptlets can perform a number of tasks that cannot be accomplished with expressions.
  - setting response headers and status codes,
  - invoking side effects such as writing to the server log or
  - updating a database, or
  - executing code that contains loops, conditionals, or
  - other complex constructs
- For instance, the following snippet specifies that the current page is sent to the client as Microsoft Word, not as HTML (which is the default).

```
<% response.setContentType("application/msword"); %>
```

Writing Scriptlets

- the scriptlet code is just directly inserted into the _jspService method: no strings, no print statements, no changes whatsoever.
- JSP expressions contain Java values (which do not end in semicolons), whereas most JSP scriptlets contain Java statements (which are terminated by semicolons).
- Expressions get placed inside print or write statements.

XML Syntax for Scriptlets

- The XML equivalent of <% Java Code %> is

  <jsp:scriptlet>Java Code</jsp:scriptlet>

Scriptlets to Make JSP Page Conditional

- Use of scriptlets is to conditionally output HTML or other content that is not within any JSP tag.

- code inside a scriptlet gets inserted into the resultant servlet's _jspService method (called by service) exactly as written and

- Any static HTML (template text) before or after a scriptlet gets converted to print statements.

Using Declarations

- A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (outside the _jspService method that is called by service to process the request).

- A declaration has the following form:

  <%! Field or Method Definition %>

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

Using Declarations

- JSP declarations can contain field (instance variable) definitions, method definitions, inner class definitions, or even static initializer blocks: anything that is legal to put inside a class definition but outside any existing methods.

- Do not use JSP declarations to override the standard servlet life-cycle methods (service, doGet, init, etc.).

- For initialization and cleanup in JSP pages, use JSP declarations to override jspInit or jspDestroy, not init or destroy.

Using Declarations

- Define most methods with separate Java classes, not JSP declarations.

- Moving the methods to separate classes (possibly as static methods) makes them
  - easier to write (since you are using a Java environment, not an HTML-like one),
  - easier to test (no need to run a server),
  - easier to debug (compilation warnings give the right line numbers; no tricks are needed to see the standard output), and
  - easier to reuse (many different JSP pages can use the same utility class).

JSP/Servlet Correspondence

- JSP declarations result in code that is placed inside the servlet class definition but outside the _jspService method.

- The XML equivalent of <%! Field or Method Definition %> is

  <jsp:declaration>Field or Method Definition</jsp:declaration>

XML Syntax for Declarations

- Multiple client requests to the same servlet result only in multiple threads calling the service method of a single servlet instance.
- They do not result in the creation of multiple servlet instances except possibly when the servlet implements the now-deprecated SingleThreadModel interface.
- Thus, instance variables (fields) of a normal servlet are shared by multiple requests, and accessCount does not have to be declared static.

XML Syntax for Declarations

- you couldn't use this for a real hit counter, since the count starts over whenever you restart the server.

- real hit counter would need to use jspInit and jspDestroy to read the previous count at startup and store the old count when the server is shut down.

- if you use jspDestroy, it would be possible for the server to crash unexpectedly (e.g., when a rolling blackout strikes Silicon Valley). So, you would have to periodically write the hit count to disk.

Using Predefined Variables

- Eight automatically defined local variables in _jspService, sometimes called "implicit objects.“
- Local variables. Not constants. Not JSP reserved words.
- these variables are not accessible in declarations.

Using Predefined Variables

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

Using Predefined Variables

- out

  This variable is the Writer used to send output to the client.

  However, to make it easy to set response headers at various places in the JSP page, out is not the standard PrintWriter but rather a buffered version of Writer called JspWriter.

  The out variable is used almost exclusively in scriptlets since JSP expressions are automatically placed in the output stream and thus rarely need to refer to out explicitly.

Example : out

index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Using Predefined Variables

- request

  This variable is the HttpServletRequest associated with the request;

  it gives you access to

  The request parameters, the

  request type (e.g., GET or POST), and

  the incoming HTTP headers (e.g., cookies).

Example of Request

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Using Predefined Variables

- response

  This variable is the HttpServletResponse associated with the response to the client.

Example : response

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

Using Predefined Variables

- config

This variable is the ServletConfig object for this page.

In principle, you can use it to read initialization parameters, but, in practice, initialization parameters are read from jspInit, not from _jspService.

Example : config

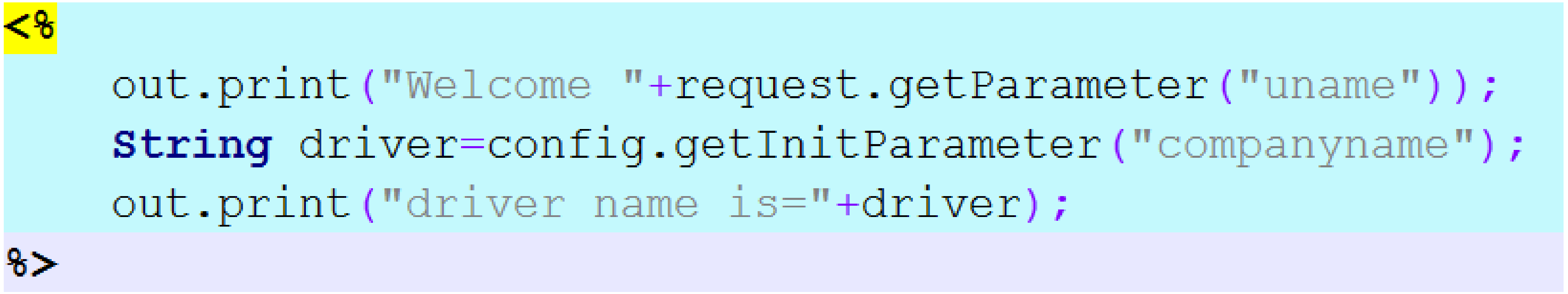
```
<form action="welcome.jsp">
    <input type="text" name="uname">
    <input type="submit" value="go"><br/>
</form>
```

```
<servlet>
    <servlet-name>test</servlet-name>
    <jsp-file>/welcome.jsp</jsp-file>

    <init-param>
        <param-name>companyname</param-name>
        <param-value>ABC Co. Ltd.</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>test</servlet-name>
    <url-pattern>/welcome.jsp</url-pattern>
</servlet-mapping>
```

Example : config



```
<%
    out.print("Welcome "+request.getParameter("uname"));
    String driver=config.getInitParameter("companyname");
    out.print("driver name is="+driver);
%>
```

Using Predefined Variables

- application

  This variable is the ServletContext as obtained by getServletContext.

  Servlets and JSP pages can store persistent data in the ServletContext object rather than in instance variables.

  ServletContext has setAttribute and getAttribute methods that let you store arbitrary data associated with specified keys

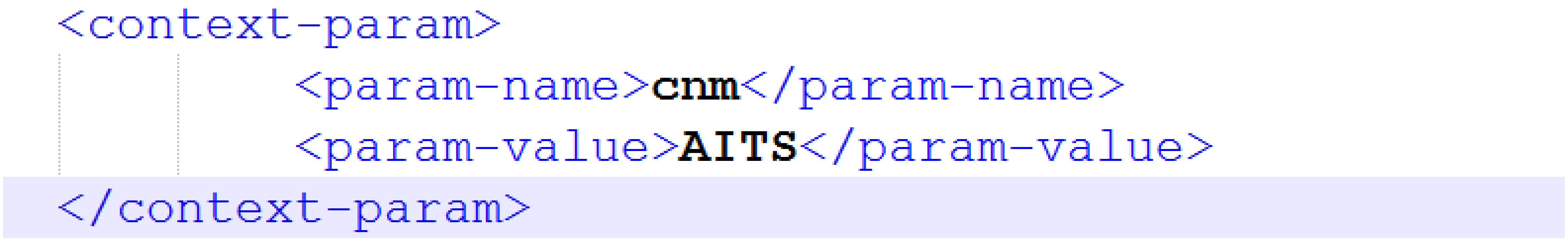
Using Predefined Variables

- application

  The difference between storing data in instance variables and storing it in the ServletContext is that the ServletContext is shared by all servlets and JSP pages in the Web application, whereas instance variables are available only to the same servlet that stored the data.

  application.getInitParameter(“cnm”);

Example : application



```
<context-param>
        <param-name>cnm</param-name>
        <param-value>AITS</param-value>
</context-param>
```

Using Predefined Variables

- session

  This variable is the HttpSession object associated with the request.

  Sessions are created automatically in JSP, so this variable is bound even if there is no incoming session reference.

Using Predefined Variables

- pageContext

  JSP introduced a class called PageContext to give a single point of access to many of the page attributes.

  The PageContext class has methods getRequest, getResponse, getOut, getSession, and so forth.

  The pageContext variable stores the value of the PageContext object associated with the current page.

  If a method or constructor needs access to multiple page-related objects, passing pageContext is easier than passing many separate references to request, response, out, and so forth.

Using Predefined Variables

- page

  This variable is simply a synonym for this and is not very useful.

  It was created as a placeholder for the time when the scripting language could be something other than Java.