JSP Introduction and Overview

▶ Understanding the need for JSP

▶ Evaluating the benefits of JSP

▶ Comparing JSP to other technologies

▶ JSP Elements

▶

What are JavaServer Pages?

▸ JSP is a specification to create dynamic web pages based on the servlet specifications

▸ Server side processing

▸ Seperates the graphical design from the dynamic content

Is JSP meant to replace Servlets?

JSP is not meant to replace Servlets

- JSP is an extentsion of the servlet technology, and it is a common practice to use both servlets and JSP pages in the same web application (MVC)
- Servlet technology may be efficient, scalable, platform independent, and buzzword compliant
  - but it is far from practical when building Web applications
- Servlets become too inflexible to survive in the dynamic environment of a Web application when used in generating the user interface
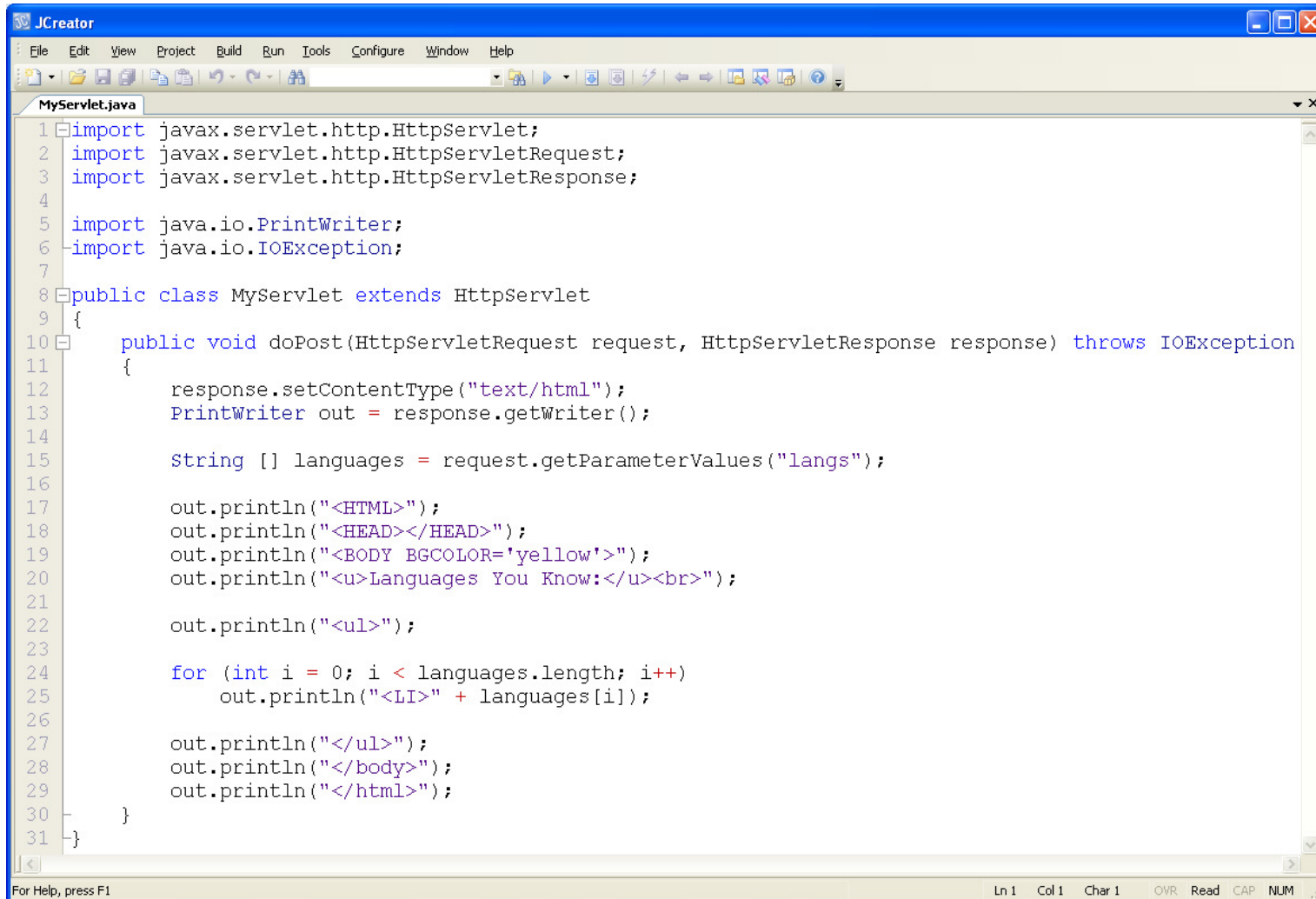- JSP is a way to counter the shortcomings of servlets

What is wrong with Servlets?

▶ Servlet programmers know how cumbersome it is to program with Servlets, especially when you have to send a long HTML page that includes little code.

Take a look at the following Servlet code

```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.PrintWriter;
import java.io.IOException;

public class MyServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String [] languages = request.getParameterValues("langs");

        out.println("<HTML>");
        out.println("<HEAD></HEAD>");
        out.println("<BODY BGCOLOR='yellow'>");
        out.println("<u>Languages You Know:</u><br>");

        out.println("<ul>");

        for (int i = 0; i < languages.length; i++)
            out.println("<LI>" + languages[i]);

        out.println("</ul>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

- More than half of the content sent from doPost method is static HTML.
- However, each HTML tag must be embedded in a String and sent using the println method of the PrintWriter object.
- It is a tedious chore
- Worse still, the HTML page may be much longer

What if we want to change the BGCOLOR to #FF0000?

▸ Another disadvantage of using Servlets is that every single change will require the intervention of the Servlet programmer.

▸ Even a slight modification, such as changing the value of bgcolor, will need to be done by the programmer.

▸

What is the solution?

▸ Sun understood this problem and soon developed a solution.

▸ The result was JSP technology.

▸ According to the Sun's website, "JSP technology is an extension of the Servlet technology created to support authoring of HTML.

▸ JSP solves drawbacks in the Servlet technology by allowing the programmer to intersperse code with static content, for example.

▸ If the programmer has to work with an HTML page template written by a web designer, the programmer can simply add code into the HTML page, and save it as a .jsp file.

▸ If at a later stage the web designer needs to change the HTML body background color, he or she can do it without wasting the programmer's time. He or she can just open the .jsp file and edit it accordingly.

▸

Demo

- ▸ Working with an HTML page template written by a web designer, the programmer can simply add code into the HTML page, and save it as a .jsp file.

With servlets, it is easy to

▸ Read form data

▸ Read HTTP request headers

▸ Set HTTP status codes and response headers

▸ Use cookies and session tracking

▸ Share data among servlets

▸ Remember data between requests

▸ Get fun, high-paying jobs

▷

With servlets, it sure is a pain to

▸ Use those println statements to generate HTML

▸ Maintain that HTML

Architectural Overview

▶ JSP page is a simple text file consisting of HTML or XML content along with JSP elements (sort of shorthand for Java code)

▶ Inside the JSP container is a special servlet called the *page compiler.*

▶ The Servlet container is configured to forward to this page compiler all HTTP requests with URLs that match the .jsp extension.

▶ When a .jsp page is first called, the page compiler parses and compiles the .jsp page into a Servlet class.

▶ If compilation is succesfull, the jsp servlet class is loaded into the memory.

▶

JSP Processing
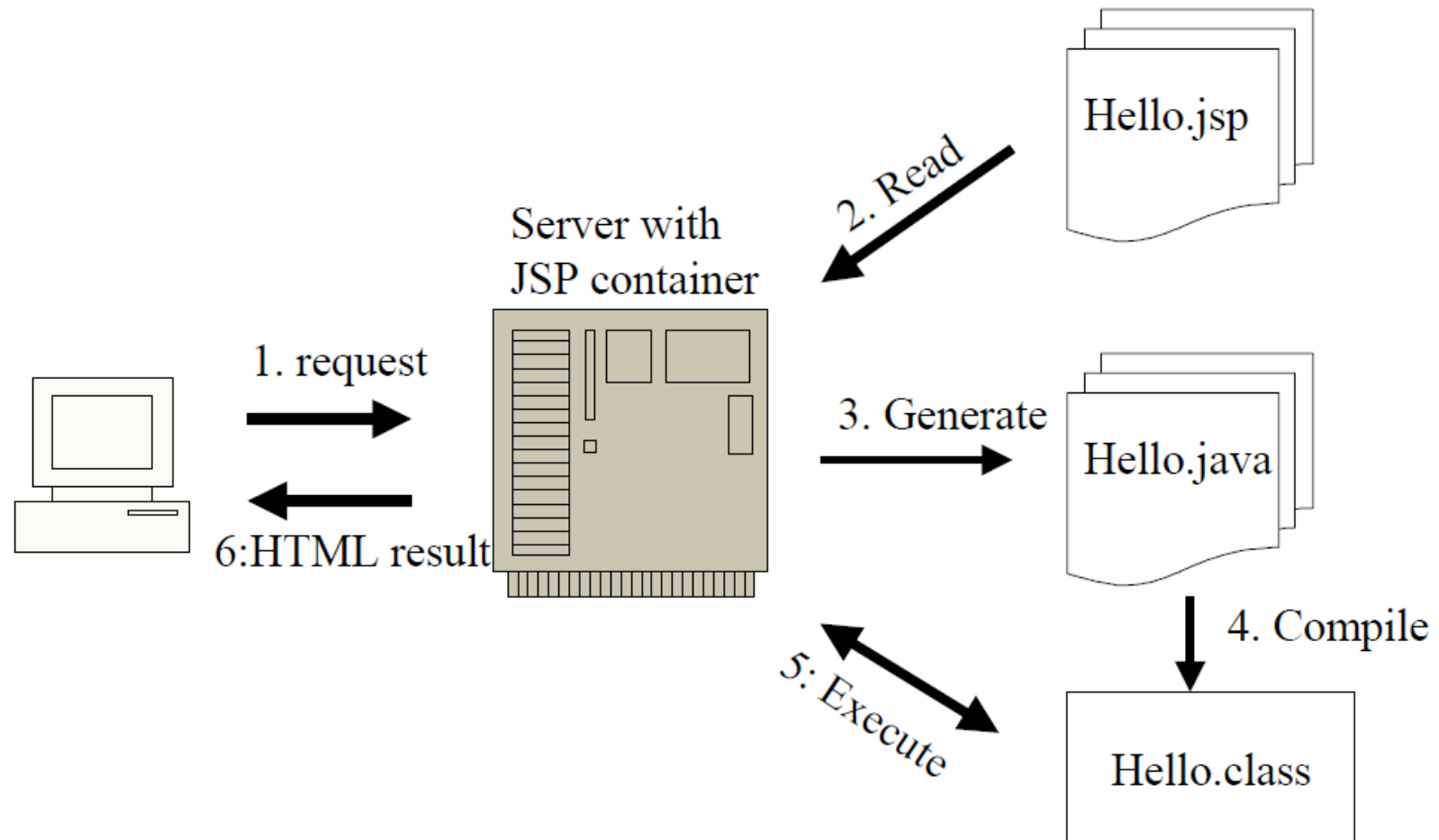
- If JSP is already compiled, compilation is skipped

Why do you think that after the deployment, the first user requests for a .jsp page will experience slow response?

▸ It is because of the time spent for compiling .jsp page into a servlet class.

Confusing Translation Time with Request Time

▸ A JSP page is converted into a servlet. The servlet is compiled, loaded into the server's memory, initialized, and executed.

▸ But which step happens when?

  ▸ To answer that question, remember two points:

    ▸ The JSP page is translated into a servlet and compiled only the first time it is accessed after having been modified.

    ▸ Loading into memory, initialization, and execution follow the normal rules for servlets.

	JSP page translated into servlet	Servlet compiled	Servlet loaded into server's memory	`jspInit` called	`_jspService` called
Page first written					
Request 1	Yes	Yes	Yes	Yes	Yes
Request 2	**No**	**No**	**No**	**No**	Yes
Server restarted					
Request 3	**No**	**No**	Yes	Yes	Yes
Request 4	No	No	No	No	Yes
Page modified					
Request 5	Yes	Yes	Yes	Yes	Yes
Request 6	No	No	No	No	Yes

JSP vs Servlets

▸ JSP pages are translated into servlets. So, fundamentally, any task JSP pages can perform could also be accomplished by servlets.

▸ However, this underlying equivalence does not mean that servlets and JSP pages are equally appropriate in all scenarios.

▸ The issue is not the power of the technology, it is the convenience, productivity, and maintainability of one or the other.

▸ After all, anything you can do on a particular computer platform in the Java programming language you could also do in assembly language.

▸ But it still matters which you choose.

▸

JSP provides the following benefits over servlets alone:

▸ **It is easier to write and maintain the HTML.**

   ▸ Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.

▸ **You can use standard Web-site development tools.**

   ▸ For example, most people use Dreamweaver or FrontPage for creating JSP pages. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.

▸ **You can divide up your development team.**

   ▸ The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

▸

▶ Now, this discussion is not to say that you should stop using servlets and use only JSP instead.

▶ By no means. Almost all projects will use both.

  ▶ For some requests in your project, you will use servlets.

  ▶ For others, you will use JSP.

  ▶ For still others, you will combine them with the MVC architecture (will discuss next week).

▷

JSP developers need to know servlets for four reasons:

- 1. JSP pages get translated into servlets. You can't understand how JSP works without understanding servlets.
- 2. JSP consists of static HTML, special-purpose JSP tags, and Java code.
  - What kind of Java code?
    - Servlet code! You can't write that code if you don't understand servlet programming.
- 3. Some tasks are better accomplished by servlets than by JSP.
  - JSP is good at generating pages that consist of large sections of fairly well structured HTML or other character data.
  - Servlets are better for generating binary data, building pages with highly variable structure, and performing tasks (such as redirection) that involve little or no output.
- 4. Some tasks are better accomplished by a combination of servlets and JSP than by either servlets or JSP alone (MVC)

JSP vs. JavaScript

- Java Scripts provide client-side execution ability
  - Interpreted
  - Cumbersome and error prone
  - Non-portable

- JSP provide server-side execution
  - Compiled
  - Portable
  - Robust
  - Not integrated with HTML – Java creates HTML

JSP vs. Servlets

▶ Similarities
  ▶ <mark>Server-side execution</mark>
  ▶ Provide identical results to the end user
  ▶ JSP will convert to Servlet.

▶ Differences
  ▶ Servlet: "HTML in Java Code"
    ▶ Mixes presentation with logic

  ▶ JSP: "Java Code Scriptlets in HTML"
    ▶ Separates presentation from logi

JSP vs. ASP

- Similarities
  - Server-side execution
  - Separates presentation from logic
  - JSP is Java's answer to ASP.

- Differences
  - **ASP**
    - Microsoft programming language, such as VB and etc.
    - Microsoft platform
    - Microsoft Product
    - Although ASP are cached, they are always interpreted.

  - **JSP**
    - Java technology
    - Platform independent
    - Specification
    - Compiled

JSP vs PHP or ColdFusion

▸ Better language for dynamic part

▸ Portable to multiple servers and operating systems

Different ways to generate dynamic content 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.

▸ Each of these approaches has a legitimate place; the size and complexity of the project is the most important factor in deciding which approach is appropriate.

▸ However, be aware that people err on the side of placing too much code directly in the page much more often than they err on the opposite end of the spectrum.

▸ Although 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.

▸

JSP Scripting Elements

- ## 1. Expressions
  - <%= Java Expression %>
    - which are evaluated and inserted into the servlet's output.

- ## 2. Scriptlets
  - <% Java Code %>
    - which are inserted into the servlet's _jspService method

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

Limiting the Amount of Java Code in JSP Pages

▸ You have 25 lines of Java code that you need to invoke. You have two options:

  ▸ (1) put all 25 lines directly in the JSP page,

  ▸ (2) put the 25 lines of code in a separate Java class, put the Java class in WEB-INF/classes/*directoryMatchingPackageName, and use* one or two lines of JSP-based Java code to invoke it.

▸ Which is better?

▸

Second Option Better

▸ The second. The second! And all the more so if you have 50, 100, 500, or 1000 lines of code.

▸ **Here's why:**

▸ **Development.**

  ▸ You generally write regular classes in a Java-oriented environment (e.g., an IDE like JBuilder or Eclipse, etc). You generally write JSP in an HTMLoriented environment like Dreamweaver. The Java-oriented environment is typically better at balancing parentheses, providing tooltips, checking the syntax, colorizing the code, and so forth.

▸ **Compilation.**

  ▸ To compile a regular Java class, you press the Build button in your IDE or invoke javac.

  ▸ To compile a JSP page, start the server, open a browser, and enter the appropriate URL.

▸ **Debugging.**

  ▸ We know this never happens to you, but when *we write* Java classes or JSP pages, we occasionally make syntax errors. If there is a syntax error in a regular class definition, the compiler tells you right away and it also tells you what line of code contains the error. If there is a syntax error in a JSP page, the server typically tells you what line *of the servlet (i.e., the servlet into which the JSP page was translated)* contains the error. For tracing output at runtime, with regular classes you can use simple System.out.println statements if your IDE provides nothing better. In JSP, you can sometimes use print statements, but where those print statements are displayed varies from server to server.

▸

▶ **Division of labor**

    ▶ Many large development teams are composed of some people who are experts in the Java language and others who are experts in HTML but know little or no Java. The more Java code that is directly in the page, the harder it is for the Web developers (the HTML experts) to manipulate it.

▶ **Testing**

    ▶ Suppose you want to make a JSP page that outputs random integers between designated 1 and some bound (inclusive). You use Math.random, multiply by the range, cast the result to an int, and add 1.

        ▶ Hmm, that sounds right. But are you sure? If you do this directly in the JSP page, you have to invoke the page over and over to see if you get all the numbers in the designated range but no numbers outside the range. After hitting the Reload button a few dozen times, you will get tired of testing.

        ▶ But, if you do this in a static method in a regular Java class, you can write a test routine that invokes the method inside a loop, and then you can run hundreds or thousands of test cases with no trouble. For more complicated methods, you can save the output, and, whenever you modify the method, compare the new output to the previously stored results.

▶ **Reuse**

    ▶ You put some code in a JSP page. Later, you discover that you need to do the same thing in a different JSP page. What do you do? Cut and paste?

        ▶ Repeating code in this manner is a cardinal sin because if you change your approach, you have to change many different pieces of code. Solving the code reuse problem is what object-oriented programming is all about. Don't forget all your good OOP principles just because you are using JSP to simplify the generation of HTML.

▶

Limit the amount of Java code that is in JSP pages

▸ "But wait!" you say, "I have an IDE that makes it easier to develop, debug, and and compile JSP pages." OK, good point. There is no hard and fast rule for exactly how much Java code is too much to go directly in the page.

    ▸ But no IDE solves the testing and reuse problems, and your general design strategy should be centered around putting the complex code in regular Java classes and keeping the JSP pages relatively simple.

▸ Almost all experienced developers have seen gross excesses: JSP pages that consist of many lines of Java code followed by tiny snippets of HTML. That is obviously bad: it is harder to

    ▸ develop, compile, debug, divvy up among team members, test, and reuse.

▸ A servlet would have been far better.

    ▸ However, some of these developers have overreacted by flatly stating that it is *always wrong to have any Java code directly in the JSP* page.

▸ Certainly, on some projects it is worth the effort to keep a strict separation between the content and the presentation and to enforce a style where there is no Java syntax in any of the JSP pages.

▸ But this is not always necessary (or even beneficial).

    ▸ A few people go even further by saying that *all pages in all applications should use* the Model-View-Controller (MVC) architecture, preferably with the Apache Struts framework. This is also an overreaction.

    ▸ Yes, MVC (will be discussed next week) is a great idea, and we use it all the time on real projects.

    ▸ And, yes, Struts is a nice framework; we are using it on a large project as the book is going to press.

▸ The approaches are great when the situation gets moderately (MVC in general) or highly (Struts) complicated.

▸ But simple situations call for simple solutions.

    ▸ all the approaches have a legitimate place; it depends mostly on the complexity of the application and the size of the development team.

    ▸ Still, be warned: beginners are much more likely to err by making hard-to-manage JSP pages chock-full of Java code than they are to err by using unnecessarily large and elaborate frameworks.

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 thedate/time that the page was requested.

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

JSP/Servlet Correspondence

▶ Now, we just stated that a JSP expression is evaluated and inserted into the page output.

▶ Although this is true, it is sometimes helpful to understand what is going on behind the scenes.

▶ It is actually quite simple: 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.

▶

Seeing the exact code that your server generates

Example: JSP Expressions

```
<HTML>
<HEAD>
<TITLE>JSP Expressions</TITLE>
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
<LI>Current time: <%= new java.util.Date() %>
<LI>Server:      <%= application.getServerInfo() %>
<LI>Session ID:   <%= session.getId() %>
<LI>The testParam form parameter: <%= request.getParameter("testParam") %>
</UL>
</BODY>
</HTML>
```

Writing Scriptlets

If you want 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 (which is called by service).

Scriptlets have the following form:

<% Java Code %>

Scriptlets have access to the same automatically defined variables as do expressions (request, response, session, out, etc.). So, for example, if you want to explicitly send output to the resultant page, you could use the out variable, as in the following example.

<%

String queryData = request.getQueryString();

out.println("Attached GET data: " + queryData);

%>

JSP/Servlet Correspondence

It is easy to understand how JSP scriptlets correspond to servlet code: the scriptlet code is just directly inserted into the _jspService method: no strings, no print statements, no changes whatsoever.

Scriptlet Example

```
<HTML>
<HEAD>
<TITLE>Color Testing</TITLE>
</HEAD>
<%
   String bgColor = request.getParameter("bgColor");
   if ((bgColor == null) || (bgColor.trim().equals("")))
   {
         bgColor = "WHITE";
   }
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Testing a Background of "<%= bgColor %>"</H2>
</BODY>
</HTML>
```

▷

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.

▶ In principle, 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.

▶ In practice, however, declarations almost always contain field or method definitions.

▶

JSP/Servlet Correspondence

▸ JSP declarations result in code that is placed inside the servlet class definition but outside the _jspService method. Since fields and methods can be declared in any order, it does not matter whether the code from declara tions goes at the top or bottom of the servlet.

▸

Declaration Example

```
<HTML>
<HEAD>
<TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
HREF="JSP-Styles.css"
TYPE="text/css">
</HEAD>
<BODY>
<H1>JSP Declarations</H1>
<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>
</BODY></HTML>
```