CS108, Stanford
Winter 2011

Handout #32
Young

JSPs

One limitation of Servlets is that they require a lot of println statements to output a webpage of any real complexity. As an alternative you can use JSPs (JavaServer Pages). A JSP is essentially an HTML page, where in specific parts of the page, you ask the webserver to execute Java. This Java is executed on the webserver, and the web browser only sees the resulting HTML. JSPs are ultimately implemented using Servlets, so you can think of a JSP as simply an alternative method for specifying a JSP.

Let's take a look at a simple JSP:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"
/>
<title>A JSP Example</title>
</head>
<body>
It is currently <%= new java.util.Date() %>
</body>
</html>
```

While this mostly looks like HTML, the key line here is:

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

Which appears in the body of the webpage. The "<%= … %>" tells the web server that we are providing Java code. It should execute the Java code between the delimiters and replace the entire thing with whatever the Java code evaluations to. In this case, it will replace the "<%= … %>" with the current date.

The standard file extensions for JSPs is naturally *.jsp.

JSP Elements

There are several different types of JSP elements we can use:

Expressions

The `<%= new java.util.Date() %>` is what we refer to as an expression. Expressions should evaluate to something which is inserted directly into the HTML file.

Notice that expressions do not have a semi-colon at the end of the line. As previously mentioned JSPs are actually automatically converted to servlets. A JSP expression is placed inside of a println to the HttpServletResponse's PrintWriter. Our Java Date code will be converted as follows:

```
out.println(new java.util.Date());
```

If we were to put a semicolon inside of our expression like this:

```
<%= new java.util.Date(); %>
```

the generated Java would be:

```
out.println(new java.util.Date(););
```

which is clearly a syntax error.

Scriplets

While Expressions are handy, because the results generated appear directly in the HTML file, sometimes you'll need to do more complex operations. This is what Scriplets are for. In a Scriplet you can perform whatever Java you want. Here's an example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>JSP Example</title>
</head>
<body>
<%
java.util.Date now = new java.util.Date();
String comment = "";

switch(now.getMonth()) {
case 0: case 1:
   comment = "very cold"; break;
case 2: case 3: case 4:
   comment = "getting warmer"; break;
case 5: case 6: case 7:
   comment = "summer!"; break;
case 8:
   comment = "school time"; break;
case 9:
   comment = "getting colder"; break;
case 10:
   comment = "Thanksgiving"; break;
case 11:
   comment = "break time"; break;
};
```

```
out.println("It is " + comment);

%>

</body>
</html>
```

As you can see between the “<%” and the “%>” we have access to regular Java code. The code in a scriptlet section is executed when the webpage loads. The code only modifies the final HTML if it explicitly interacts with the HttpServletResponse PrintWriter. In our example, you can see this on the last line of the scriptlet:

```
out.println("It is " + comment);
```

You may notice that we never actually define the variable `out`. Scriplets have access to several pre-defined variables these include:

`out` – gives access to a PrintWriter for the response.

`request` – gives access to the HttpServletRequest object.

`response` – gives access to the HttpServletReponse object.

`session` – gives access to the HttpSession object.

`application` – gives access to the ServletContext object.

`config` – gives access to the ServletConfig object.

Declarations

The code in our scriplets and expressions goes into a single method called the _jspService method. This method gets called when our JSP receives a GET or POST request. Sometimes we’ll want to provide code which is independent of this method. We’ll do this using a JSP Declaration.

JSP declarations are indicated using “<%! … %>”. Code in these sections will go into the servlet created for our JSP, but they won’t go inside a specific method. That means we can use a JSP declaration to declare instance variables, helper methods, or even nested classes. Here’s the previous example rewritten with a helper method using a JSP declaration:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Insert title here</title>
</head>
```

```
<body>
<%!
String getComment(java.util.Date date) {
 String comment = "";

 switch(date.getMonth()) {
 case 0: case 1:
    comment = "very cold"; break;
 case 2: case 3: case 4:
    comment = "getting warmer"; break;
 case 5: case 6: case 7:
    comment = "summer!"; break;
 case 8:
    comment = "school time"; break;
 case 9:
    comment = "getting colder"; break;
 case 10:
    comment = "Thanksgiving"; break;
 case 11:
    comment = "break time"; break;
 };

 return comment;
}
%>

<%
java.util.Date now = new java.util.Date();
out.println("It is " + getComment(now));
%>

</body>
</html>
```

Defining the getComment method isn't particularly effective in this case, since we only call it once. But this example does illustrate how the declaration can be used to define a method which can be used anywhere in your JSP.

Comments

We can write a JSP comment like this:

```
<%-- This is a comment --%>
```

A JSP comment differs from an HTML comment in that an HTML comment is actually sent to the web client, and a user using "View Source" will see it. A JSP comment is ignored when generating the HTML and does not appear in the final HTML file.

Importing in JSPs

You'll notice that intead of writing Date, I instead wrote java.util.Date. This is because we hadn't yet learned how to import into our JSP. We can't simply put an import statement at the top of a scriplet or declaration. Instead we use another JSP element called the directive. Here's what your import should look like:

```
<%@ page import="java.util.*" %>
```

Notice that this line is different from a standard Java import. It has an equal sign after the keyword import. The package being imported is placed in quotes. There is also no semicolon at the end of the line.

We can import multiple packages y separating them with a comma:

```
<%@ page import="java.util.*, core.*" %>
```

Place your import write before the `<!DOCTYPE …>` and `<html>` tags and write after Eclipse's auto-generated:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
```

The javax.servlet libraries are automatically imported for you.

JSPs and the "Default" Package

This is important! JSPs will be automatically compiled and placed into their own package. This package is different from the "default" package. Do not define any classes in the default package if you want to use them from within a JSP. There is no way to import from the "default" package to the JSP's package.

Compiling JSPs

JSPs are converted to servlets when the first visitor to a website visits a webpage which is a JSP. This means that you can have compile-time errors occur at what you might consider run-time.

Some types of web servers can pre-compile JSPs, but there is no standardized way to force a webserver to pre-compile.

Forwarding with Servlets and JSPs

In the Servlet handout, I showed how, instead of generating code directly, a servlet could instead forward to an HTML webpage. This same technique can be used to forward to a JSP.

This technique is particularly powerful with JSPs, as the Servlet can handle most of the processing, it can then store data as attributes on the HttpServletRequest object. This same request object will be passed on to the JSP, which can read the attributes and use them to generate the new webpage.

When to Use JSPs

One question that arises once JSPs and Servlets are learned is when to use one vs. the other. In general, if you have a lot of processing, you should use servlets. If you have lots of complex HTML you may consider using a JSP. If you have lots of processing and you also have complex HTML, use a servlet which forwards to a JSP.

XML JSPs

Some web servers support an alternative form of JSP elements which use XML. For these servers your expressions, scriplets, and declarations would look like this:

```
<jsp:expression> ... </jsp:expression>

<jsp:scriplet> ... </jsp:scriplet>

<jsp:declaration> ... </jsp:declaration>
```