JSP - DEBUGGING

It is always difficult to testing/debugging a JSP and servlets. JSP and Servlets tend to involve a large amount of client/server interaction, making errors likely but hard to reproduce.

Here are a few hints and suggestions that may aid you in your debugging.

Using System.out.println():

System.out.println() is easy to use as a marker to test whether a certain piece of code is being executed or not. We can print out variable values as well. Additionally:

- Since the System object is part of the core Java objects, it can be used everywhere without the need to install any extra classes. This includes Servlets, JSP, RMI, EJB's, ordinary Beans and classes, and standalone applications.
- Compared to stopping at breakpoints, writing to System.out doesn't interfere much with the normal execution flow of the application, which makes it very valuable when timing is crucial.

Following is the syntax to use System.out.println():

```
System.out.println("Debugging message");
```

Following is a simple example of using System.out.print():

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>System.out.println</title></head>
<body>
<c:forEach var="counter" begin="1" end="10" step="1" >
   <c:out value="${counter-5}"/></br>
   <% System.out.println( "counter= " +
                    pageContext.findAttribute("counter") ); %>
</c:forEach>
</body>
</html>
```

Now if you will try to access above JSP, it will produce following result at browser:

```
-4
-3
-2
-1
0
1
2
3
4
5
```

If you are using Tomcat, you will also find these lines appended to the end of stdout.log in the logs directory.

```
counter=1
counter=2
counter=3
counter=4
counter=5
counter=6
counter=7
counter=8
counter=9
counter=10
```

This way you can pring variables and other information into system log which can be analyzed to find out the root cause of the problem or for various other reasons.

Using the JDB Logger:

The J2SE logging framework is designed to provide logging services for any class running in the JVM. So we can make use of this framework to log any information.

Let us re-write above example using JDK logger API:

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

<html>
<head><title>Logger.info</title></head>
<body>
<% Logger logger=Logger.getLogger(this.getClass().getName());%>

<c:forEach var="counter" begin="1" end="10" step="1" >
   <c:set var="myCount" value="${counter-5}" />
   <c:out value="${myCount}"/></br>
   <% String message = "counter="
                  + pageContext.findAttribute("counter")
                  + " myCount="
                  + pageContext.findAttribute("myCount");
                  logger.info( message );
   %>
</c:forEach>
</body>
</html>
```

This would generate similar result at the browser and in stdout.log, but you will have additional information in stdout.log. Here we are using **info** method of the logger because we are logging message just for informational purpose. Here is a snapshot of stdout.log file:

```
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=1 myCount=-4
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=2 myCount=-3
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=3 myCount=-2
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=4 myCount=-1
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=5 myCount=0
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=6 myCount=1
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=7 myCount=2
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=8 myCount=3
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=9 myCount=4
24-Sep-2010 23:31:31 org.apache.jsp.main_jsp _jspService
INFO: counter=10 myCount=5
```

Messages can be sent at various levels by using the convenience functions **severe(), warning(), info(), config(), fine(), finer(),** and **finest()**. Here finest() method can be used to log finest information and severe() method can be used to log severe information.

You can use Log4J Framework to log messages in different files based on their severity levels and importance.

Debugging Tools:

NetBeans is a free and open-source Java Integrated Development Environment that supports the development of standalone Java applications and Web applications supporting the JSP and servlet specifications and includes a JSP debugger as well.

NetBeans supports the following basic debugging functionalities:

- Breakpoints
- Stepping through code
- Watchpoints

You can refere to NteBeans documentation to understand above debugging functionalities.

Using JDB Debugger:

You can debug JSP and servlets with the same **jdb** commands you use to debug an applet or an application.

To debug a JSP or servlet, you can debug sun.servlet.http.HttpServer, then watch as HttpServer executing JSP/servlets in response to HTTP requests we make from a browser. This is very similar to how applets are debugged. The difference is that with applets, the actual program being debugged is sun.applet.AppletViewer.

Most debuggers hide this detail by automatically knowing how to debug applets. Until they do the same for JSP, you have to help your debugger by doing the following:

- Set your debugger's classpath so that it can find sun.servlet.http.Http-Server and associated classes.
- Set your debugger's classpath so that it can also find your JSP and support classes, typically ROOT\WEB-INF\classes.

Once you have set the proper classpath, start debugging sun.servlet.http.HttpServer. You can set breakpoints in whatever JSP you're interested in debugging, then use a web browser to make a request to the HttpServer for the given JSP (http://localhost:8080/JSPToDebug). You should see execution stop at your breakpoints.

Using Comments:

Comments in your code can help the debugging process in various ways. Comments can be used in lots of other ways in the debugging process.

The JSP uses Java comments and single line (// ...) and multiple line (/* ... */) comments can be used to temporarily remove parts of your Java code. If the bug disappears, take a closer look at the code you just commented and find out the problem.

Client and Server Headers:

Sometimes when a JSP doesn't behave as expected, it's useful to look at the raw HTTP request and response. If you're familiar with the structure of HTTP, you can read the request and response and see exactly what exactly is going with those headers.

Important Debugging Tips:

Here is a list of some more debugging tips on JSP debugging:

- Ask a browser to show the raw content of the page it is displaying. This can help identify formatting problems. It's usually an option under the View menu.
- Make sure the browser isn't caching a previous request's output by forcing a full reload of the page. With Netscape Navigator, use Shift-Reload; with Internet Explorer use Shift-Refresh.