

Web Programming with Servlets and JSPs

Dr Dimitris C. Dracopoulos

email: d.dracopoulos@westminster.ac.uk

1 The Need for Servlets

HTML pages are static. Each time the web server sends it out, it sends exactly the same information.

Ideally we would like:

- Pass some parameters to the server.
- The server runs a program to which it passes these parameters.
- A new HTML page is created based on the program it run, and this is returned to the client (browser).

Servlets and JSPs can create dynamically such information.

2 Why Servlets?

Before servlets CGI scripts were used to generate such dynamically created HTML information. Typically written in Perl or other scripting language.

- Servlets give you full flexibility using Java.
- All the Java API (e.g. JDBC) are available to use in servlets.
- Enable the separation of business logic from presentation (see 3-tier architecture).
- Portability: Available in all platforms, assuming you are using a servlet-enabled web server.

3 Web servers supporting servlets

Many commercial products.

However, an open source product is:

- Tomcat: Apache's Java-based Jakarta project.
- Tomcat can be used as:
 - A standalone web server.
 - An addition to other web servers to process Java requests.

The following table shows the compatibility between the different versions of Servlet/JSP APIs and Tomcat:

Servlet container	Version	Compatible with
JServ	ver.1.1.3	Servlet 2.0, JSP 1.0
Tomcat	ver.3.2.1	Servlet 2.2, JSP 1.1
Tomcat	ver.4.0	Servlet 2.3, JSP 1.2
Tomcat	ver.5.x	Servlet 2.4, JSP 2.0

JServ was an earlier product than Tomcat and Tomcat was based on it. See jakarta.apache.org for more information.

4 Setting up Tomcat

To use Tomcat with servlets the following environmental variables need to be set up:

```
set TOMCAT_HOME = c:\tomcat\5.0
set JAVA_HOME = c:\jdk1.5
```

The actual values of the variables are system dependent (based on what software you have installed and the paths that they are installed). Depending on the OS the above variables will be set in a different way.

- Tomcat has configuration files which determine on which port the server is listening to and how the locations of the servlets will be found. The configuration files can be found in the `TOMCAT_HOME/conf` directory where `TOMCAT_HOME` is the directory you install tomcat. The configuration files are in XML format.
- The web administrator starts and stops the web server by issuing the corresponding *startup/shutdown* commands.

5 Servlet performance

For performance reasons, a servlet is first loaded into the server, the first time it is requested.

- Once it is loaded it always remains in memory unless the web server is restarted.
- This implies that changes in a servlet will not take effect unless the web server is restarted.
- Tomcat can be reconfigured so as to reload a servlet every time it is requested. This can be used during the development cycle.
- Once the development is done, the servlets are deployed to the customer, therefore for performance reasons Tomcat will be configured so as to load the servlet only the first time it is requested.

6 Running a Servlet

1. Compile the servlet using:

```
javac -Djava.ext.dirs=$TOMCAT_HOME/common/lib petservlet.java
```

where `$TOMCAT_HOME` the directory that tomcat is installed. For Windows machines replace slashes with backslashes.

2. Place the compiled *class* files corresponding to the servlet under the directory:

```
$TOMCAT_HOME/webapps/servlets-examples/WEB-INF/classes
```

3. Copy the HTML `petform.html` form in directory:

```
$TOMCAT_HOME/webapps/servlets-examples
```

4. Access (run) the servlet using the following URL in a browser:

```
http://localhost:8080/petform.html
```

5. Add the following lines to the configuration file `$TOMCAT_HOME/webapps/servlets-examples/WEB-INF/w`

```
<servlet>
  <servlet-name>petservlet</servlet-name>
  <servlet-class>petservlet</servlet-class>
</servlet>
```

and

```
<servlet-mapping>
  <servlet-name>petservlet</servlet-name>
  <url-pattern>/servlet/petservlet</url-pattern>
</servlet-mapping>
```

7 The Http GET and POST commands

Browsers and web servers communicate using the HTTP protocol.

There are two ways (methods) for sending the requesting data:

- *GET*: sends the data appended to the URL as a string:

```
http://www.yahoo.com/stocks.htm?ibm=32.45
```

It is limited to a small amount of data. Also as many servers log requests, if you would not like some information you submit to be logged, then *GET* should not be used.

- *POST*: passes the data as a series of fields in the body of the HTTP request. While *GET* responses can be cached (by the browser, proxy server, web server), *POST* goes through all of the caching layers and extracts the information again. Therefore *POST* should be used for counting accesses, etc.

8 Creating a Servlet

To write a servlet:

1. Inherit from one of the `javax.servlet` classes, typically `HttpServlet` which is an abstract class.
2. Must override at least one of these methods: `doGet()`, `doPost()`. These
3. `init()`, `destroy()` can be overridden for initialisation and deallocation of resources respectively. Note that `init()` will only be executed once when the class is first loaded.
4. Use the `HttpServletRequest`, `HttpServletResponse` objects to receive information about the request from the client and to send information back to the client respectively. These two objects are passed as arguments to both the `doGet()`, `doPost()` methods.
5. The information sent to the client should be formatted in HTML, therefore this is the responsibility of the servlet.

The skeleton of a servlet is:

```
public class MyServlet extends HttpServlet {  
  
    public void init() {...}  
    public void doGet() {...}  
    public void doPost() {...}  
    public void destroy() {...}  
}
```

where

```

public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException
public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException

```

Both `doGet()`, `doPost()` have `protected` access, therefore they can only be accessed by subclasses and by classes in the same package.

9 The Hello World servlet

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}

```

10 Writing HTML to invoke a Servlet

A common way to invoke a servlet is to create a HTML form which collects information from the user (as input) and submits this to the web server. The server will provide all of the user provided information to the servlet (as part of the `HttpServletRequest` object) and run it.

```

<body>
  <center><h2>Choosing a pet</h2> </center>
  <form action=/examples/servlet/petservlet method=post >
  <h4>Preferred weight (lbs): </h4>
    <input type=text name=weight size=3>

  <h4>Number of legs: </h4>

```

```



```

11 The Pet Request Servlet

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.text.*;
import java.util.*;

public class petservlet extends HttpServlet {

    private String recommendedPet(int weight, int legs) {
        if (legs ==0) return "a goldfish";
        if (legs ==4) {
            if (weight<20) return "a cat";
            if (weight<100) return "a dog";
        }
        return "a house plant";
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp )
        throws ServletException, IOException {

        // get the input field values
        int petWeight = 0, petLegs = 0;
        try {
            petWeight = Integer.parseInt(req.getParameter("weight"));
            petLegs = Integer.parseInt(req.getParameter("legs"));
        } catch (NumberFormatException nfe) {
            // indicates that we got an invalid number
            petWeight=petLegs=-1;
        }

        resp.setContentType("text/html");

        PrintWriter out = resp.getWriter();

        out.println(" <html> <body> <h1>Recommended Pet</h1> <p>");

```

```

        out.println("You want a " + petLegs + "-legged pet weighing "
                    + petWeight + "lbs.");

        String pet = recommendedPet(petWeight, petLegs);
        out.println("<P> We recommend getting <b>" + pet );
        out.println("</b> <hr> </body> </html> ");

        out.close();
    }
}

```

12 Multiple Threads and Servlets

Servlets typically run on multithreaded servers, so that a servlet must handle concurrent requests.

- The servlet object stays in memory once it is loaded.
- Each incoming request to the server creates a new thread.
- If multiple clients try to access the same servlet then the same code will be executed at the same time.
- Synchronisation access to shared resources is needed. Synchronise instance variables, and static data.

13 Java Server Pages (JSP)

A combination of static HTML pages and Java statement. Unlike servlets the HTML which does not change does not have to be generated “manually”.

- JSP use special tag to separate HTML from Java. Java is included between the <% and %> tags.
- Another character might be following <% to further specialise its meaning.
- The Java code fragments in a JSP will be compiled automatically to create a complete servlet.

Opening JSP Tag	Meaning
<%	Everything up to the closing tag is Java code
<%=	Evaluates the Java expression that follows
<%@	Followed by <code>method</code> , <code>import</code> , <code>implements</code> , <code>extends</code> .
<%!	Followed by a Java declaration

14 How to run a JSP

- Place the jsp in the directory `$TOMCAT_HOME/webapps/jsp-examples/`, where `TOMCAT_HOME` the directory where you installed tomcat.
- Browse the following URL (assuming your jsp is called `time.jsp`):

`http://localhost:8080/jsp-examples/time.jsp`

- The JSP will be compiled the first time it is accessed. Therefore the subsequent times it is accessed, it will appear like running faster.

15 The Date JSP

```
<html>
<body bgcolor=white>
<P>
<UL>
  <b>current time is:
    <%= new java.util.Date() %>
  </b>
</ul>
</body>
```

When this is browsed it produces:

```
current time is: Mon Mar 10 10:51:25 GMT 2003
```

16 Getting the headers of a request Example

```
<html>

<body bgcolor="white">
<h1>The Echo JSP</h1>

  <%
    java.util.Enumeration eh = request.getHeaderNames();
    while (eh.hasMoreElements()) {
      String h = (String) eh.nextElement();
      out.print("<br> header: " + h );
      out.println(" value: " + request.getHeader(h) );
    }
  %>

</body>
</html>
```

The output of this is:

```

header: host value: localhost:8080
header: user-agent value: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0) Gecko/20020623
  Debian/1.0.0-0.woody.1
header: accept value: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
  plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1
header: accept-encoding value: gzip, deflate, compress;q=0.9
header: accept-charset value: ISO-8859-1, utf-8;q=0.66, *,q=0.66
header: keep-alive value: 300
header: connection value: keep-alive
header: cookie value: JSESSIONID=7596B9338285F331B3F0E379E49769B3

```

It is possible to shorten this example by using the `<%= JSP` tag which automatically converts the output to a string:

```

<html>
<body bgcolor="white">
<h1>The Echo 2 JSP</h1>

<%
  java.util.Enumeration eh = request.getHeaderNames();
  while (eh.hasMoreElements()) {
    String h = (String) eh.nextElement();
  %>
<br> header: <%= h %>
  value: <%= request.getHeader(h) %>

  <%
    }
  %>
</body>
</html>

```

17 Calling a Servlet from a JSP

```

<html>
<!--
  Copyright (c) 1999 The Apache Software Foundation. All rights
  reserved.
-->
<body bgcolor="white">

<!-- Forward to a servlet -->
<jsp:forward page="/servlet/servletToJsp" />

</html>

```

18 Calling a JSP from a Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;

public class servletToJsp extends HttpServlet {
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response) {
        try {
            // Set the attribute and Forward to hello.jsp
            request.setAttribute ("servletName", "servletToJsp");
            getServletConfig().getServletContext().getRequestDispatcher(
                "/jsp/jsptoserv/hello.jsp").forward(request, response);
        } catch (Exception ex) {
            ex.printStackTrace ();
        }
    }
}
```

```
<html>
<!--
  Copyright (c) 1999 The Apache Software Foundation.  All rights
  reserved.
-->
<body bgcolor="white">

<h1>
I have been invoked by
<% out.print (request.getAttribute("servletName").toString()); %>
Servlet.
</h1>

</html>
```

19 Deploying Servlets and JSPs

To facilitate the deployment of Servlets and JSPs we can use WAR (Web Archive) files. This can include both servlets and JSPs.

- These are similar with *jar* files and in the similar format.
- The *jar* utility can be used to create them.
- The war files include as `WEB-INF` directory and a `web.xml` file that described the application to tomcat.