Using Servlet Technology for Web Pages Design

Lecturer PhD. Marian CRISTESCU
"Lucian Blaga" University of Sibiu, Faculty of Sciences, Department of Computer Sciences
E-mail: marian.cristescu@ulbsibiu.ro

The rise of server-side Java applications is one of the most interesting trends in Java programming. The Java language was originally design for use with small devices, and it was first used as a language for developing elaborate client-side wed content in the form of applets. But in the last years, Java has shown a great potential for developing server-sides, and now has come to be recognized as a language ideally suited for server-side development.
***Keywords**: servlet technology, web design, session tracking, HTTP protocol.*

1 Advantages against the others web design technology

A servlet is a particular Java server extension which can be loaded dynamically to expand the functionality of a server, and servlets are commonly used with web servers, where they can take the place of CGI scripts. A servlet is similar to a proprietary server extension, except that it runs inside a Java Virtual Machine (JVM) on the server, as it shown in figure 1, so it is safe and portable. Servlets operate solely within the domain of the server: unlike applets, they do not require support for Java in the web browser.

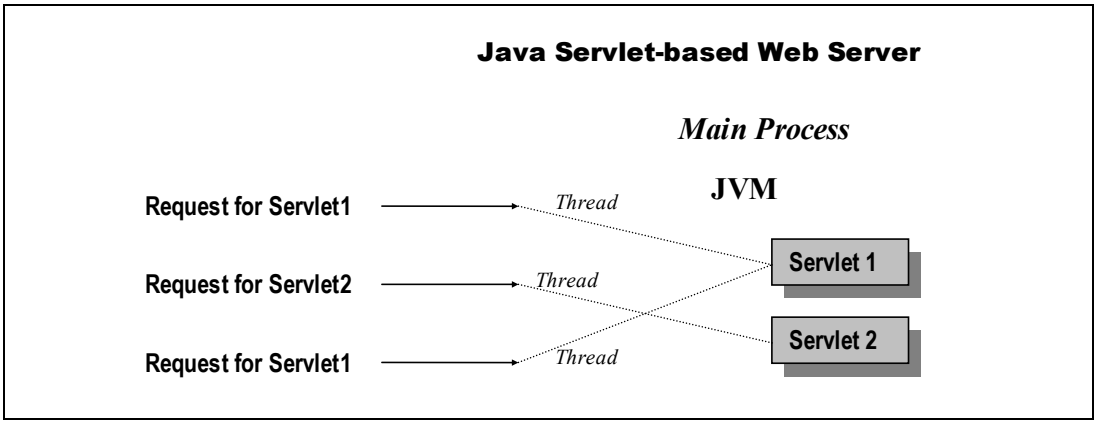

Fig.1. The servlet life cycle

Unlike CGI and FastCGI, which use multiple processes to handle separate programs and/or separate requests, servlets are all handled by separate threads within the web server process or by threads within multiple processes spread across a number of servers.
Because servlets run within the web server, they can interact very closely with the server to do things that are not possible with CGI scripts. In addition to this, servlets have the advantage to be portable: both across operating system as it is Java and also across web servers.

2. Genereting a web page using servlet technology

Normaly, every servlet must implement the `javax.servlet.Servlet` interface, but most of the servlets implement it by extending one of the next two classes: `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. Unlike a regular Java program, and just like an applet, a servlet does not have a `main()` method. Instead, certain methods of a servlet are invoked by the server in the process of handling requests. Each time the server dis-

patches a request to a servlet, it invokes the servlet's `service()` method.

A generic servlet should override its `service()` method to handle requests properly for the servlet. The service() method takes two parameters: a request object and a response object. Figure 2 shows how a generic servlet handles requests.

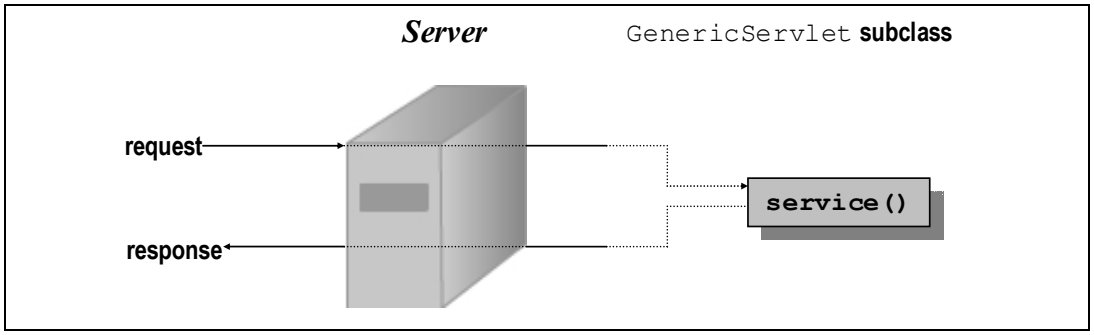

Fig.2. A generic servlet handling a request

An HTTP servlet usually does not override the `service()` method. Instead, it overrides `doGet()` to handle GET requests and `doPost()` to handle POST requests. The `service()` method of HttpServlet handles the setup and dispatching to all the `doXXX()` methods, which is why it usually should not be overridden. Figure 3 shows how an HTTP servlet handles GET and POST requests.

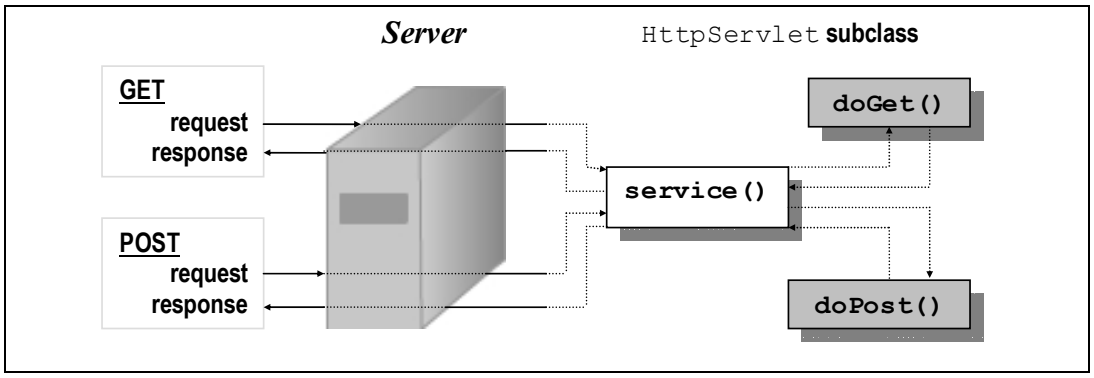

Fig.3. An HTTP servlet handling GET and POST requests

The most basic type of HTTP servlet generates a full HTML page. Such a servlet has access to the same information usually sent to a CGI script, plus a bit more. Example 1 shows an HTTP servlet that generates a complete HTML page.

Example 1. A HTTP servlet that generates a login page:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HelloWorldServlet extends HttpServlet
{
protected void doGet(HttpServletRequest aRequest, HttpServletResponse aRe-
sponse) throws ServletException,IOException
      {     aResponse.setContentType("text/html");
            PrintWriter out = aResponse.getWriter();
```

```
            out.println("<html>");
            out.println("<head><title>Login </title></head>");
            out.println("<body>");
            out.println("<form action=/servlet/loginhandler
                              method = post>")
            out.println("<center>");
            out.println("Username: <input type=text name=”user”
                              value=’’ size=20><br>");
            out.println("Password: <input type=password name=”password”
                              value=’’ size=20><br>");
            out.println("<input type=sumit value=’OK’>");
            out.println("</center>");
            out.println("</body></html>");
      }
}
```

Just like applets, servlets can define `init()` and `destroy()` methods. The server calls a servlet's `init()` method after the server constructs the servlet instance and before the servlet handles any requests. The server calls the `destroy()` method after the servlet has been taken out of service and all the requests to the servlet have completed or timed out.

The `init()` method is typically used to perform servlet initialization—creating or loading objects that are used by the servlet in the handling of its requests. This method can take also take parameters, which are given to a servlet itself and are not associated with any single request.

In the `destroy()` method, a servlet frees any resources it has acquired that will not be garbage collected. The destroy() method also gives a servlet a chance to write out its unsaved cached information or any persistent information that should be read during the next call to `init()`.

3. Session tracking

A server that uses HTTP protocol for communication can not recognize that a sequence of requests were all originated from the same user, so it is necessary for the user to introduce himself every times he makes a request, in fact to attach in request a unique identifier that lets the server identify it. For solving this there are two possibilities: using the traditional session-tracking techniques or using of the Session Tracking API.

The traditional session-tracking techniques, which can be used also by the CGI developers, are easily implemented with servlet theology:

• ***user authentication***: is possible by using of `getRemoteUser()` method which returns the username, after the client's login.

Example 2. A servlet that uses user authenication:

```
...
String name = reg.getRemoteUser();
if (name == null)
{     // Message that explains why the access in forbidden
}
else
{     // Instructions that handle the requests
}
...
```

• ***hidden form fields***: the advantages of this method is that hidden fields are supported in all the popular browsers that works with HTML, they demand no special server requirements and can be with clients that haven't registered or logged in. Hidden form fields are included in a HTML file like this:

Example 3. A servlet that uses hidden form fields:

```
<FORM ACTION = “/servlet/servletName” METHOD = “POST”>
...
<INPUT TYPE = hidden NAME = “number” VALUE = “1234”>
```

```
<INPUT TYPE = hidden NAME = "character" VALUE = "Java">
...
</FORM>
```

• ***URL rewriting***: with URL rewriting the clients can include extra information to the end of every URL for identify the session, so that the server can be able to associate the extra information with the information that he has about the session. The extra information can be in the form of extra path information, added parameters, or some custom, server-specific URL change;

• ***cookies***: cookies offer an elegant, efficient and easy way to implement session tracking. For each request, a cookie can automatically provide a client's session ID or perhaps a list of client's preferences. But the biggest problem with cookies is that browsers don't always accept cookies. Sometimes because the browser doesn't support cookies, but more often because the user has configured the browser to refuse cookies.

Example 4. A servlet that uses cookies:

```
... ... ...
String sessionId;
Coockie[] cookies =reg.getCoockies();
If (cookies != null)
{     for(int i = 0;  i < cookies.lenght;  i++)
      {     if (cookies[i].getName.equals("sessionid"))
            {     sessionId = cookies[i].getValue();
                  break;
            }
}
... ... ...
```

In addition to traditional session-tracking techniques there is support for session-tracking in the Servlet API, which makes easily for to developers to handle this problem by using servlets. Session Tracking API represents that part from the Servlet API which handles this problem. A `HttpSession` object makes an association between a HTTP client and a HTTP server. This association is maintained through a set of connections and requests over a specify period of time. Associations maintain information about the state and information about the user.

A servlet uses its request object's `getSession(boool create)` method to retrieve the current `HttpSession` object. This method returns the current session associated with the user making the request. If the user has no current valid session, this method creates one if `create` is `true` or returns `null` if `create` is `false`.

Example 5. Un servlet care utilizeaza HTTPSession:

```
... ... ...
HTTPSession session = req.getSession();
String sessionAtribute = session.getAtribute("clientName");
... ... ...
session.setAtribute("clientName", otherAtribute);
... ... ...
```

4. Using databases

Using databases in designing web sides has become a common thing due to the advantages that this kind of data storing offers: fast access to information, selections after different criteria and an easy actualization. The only problem with using databases in designing web sites is that these sites are harder to implement than usual sites.

The biggest advantage for servlets with regard to database connectivity is that the servlet life cycle allows servlets to maintain open

database connections. An existing connection can trim several seconds from a response time, compared to a CGI script that has to re-establish its connection for every invocation.

Example 6. Doing a SQL statement:

```
... ... ...
try
{     db = new DBHelper();
}
catch (Exception e)
{     throw new SQLException(e.getMessage());
}
      Connection con = null;
try
{     con = db.getConnection();
      DBHelper.executeUpdate(con,
            "insert into PROIECTE (PROI_NUME, CLIE_ID) values (?, ?)",
            new Object[] {
                  NumeProiect,
                  IdClient});
            }
finally
{     DBHelper.close(con);
}
... ... ...
```

The previous example presents a simple way through which a servlet connects to a database, realize a SQL statement and in the end it closes the connection.

5. Conclusions

Servlet technology allows creating web pages with dynamic content in an elegant way, based on object-oriented technology, and also solving, through predefined classes, the most important problems of web pages: handling requests, session tracking and database connectivity.

For more, an application made by using servlet technology presents the advantage to be portable over the operating system and also over the web server. The only thing that a client needs to run the application is a simple web browser.

References:

[HALL00] – Hall M., "*Core Servlets and JavaServer Pages*", Sun Microsystems Press, 2000;

[HUCR01] – Hunter J., Crawford W. "*JAVA Servlet Programming, 2nd Edition*". O'Reilly, 2001;

[****] - www.corewebprogramming.com

[****] - www.javaworld.com

[****] - java.sun.com/products/servlet/2.2/javadoc/