ServletApp: Making Servlets Compile and Run Like Java Applications

Allan M. Hart and James M. Slack
Department of Computer and Information Sciences
Minnesota State University
Mankato, MN 56001
allan.hart@mnsu.edu, james.slack@mnsu.edu

Abstract

Teaching servlet programming has its rewards for both instructor and student. However, it presents unique challenges both in terms of available resources (a machine running a servlet engine of some kind is required) and risks (servlet code should not be trusted). In this paper we introduce *ServletApp,* a single 1.4MB .jar file containing all that is necessary to run a servlet as an application. Students who use ServletApp are thus provided with an environment in which it is easy to write, compile, test and debug servlet and JSP code.

1. Introduction

In an attempt to keep our undergraduate database course as "cutting edge" as possible, we have, in recent years, attempted to incorporate a JDBC[1] component into the course. This seemed an appropriate strategy since our CS1 and CS2 courses are Java[2] based and we are thus able to leverage our students' experience with Java in the database course. In order to make our students' experience with JDBC as realistic as possible, we require a semester project in which teams of 4 to 6 students create an online shopping application that includes an Oracle or SQL 2000 Server database. Access to the database is to be provided via SQL queries submitted to a JDBC based servlet[3]. A 3-tier architecture is envisioned. Thus, our students are actually exposed to two Java APIs, viz. JDBC and Servlets.

2. The Problem

While teaching JDBC is relatively straightforward, teaching Servlets is not. Problems arise because of the following considerations:

- Unless a student has Tomcat[4] or a similar servlet engine installed on his/her computer, they will have no effective way of testing their servlet code. Teaching students how to install/configure/maintain Tomcat or a similar servlet engine, while possible, would require more time than we want to devote to a topic that is really not part of a database course.
- Running Tomcat or a similar servlet engine on our personal office machines and having students upload and test their servlet code on those machines while possible is, if nothing else, highly dangerous. Servlet code, unlike Applet code, does *not* run in a "security sandbox" and while it may be possible to put enough restrictions in place to alleviate this problem, neither of us relishes the thought of our personal office machines having their hard drives erased by malicious student servlet code.
- Running Tomcat or a similar servlet engine on a machine dedicated to the task of running student servlet code while possible would still not address the problem of malicious student servlet code noted above. Moreover, while some servlet engines provide an auto reload feature that eliminates the need to restart the servlet engine when new or modified servlet code is uploaded, not all do; and, of those that do, not all do so reliably.

What is wanted then is an "environment" in which students can conveniently write, compile, test, and debug servlet code.

3. The Solution

Our solution to this dilemma comes in the form of *ServletApp*, a single 1.4MB .jar file containing a web server, servlet engine, JSP[5] engine, and browser. ServletApp makes a servlet compile and run like an ordinary Java application. Because it has a built-in browser, ServletApp is capable of launching its own browser windows. Either Internet Explorer or Netscape Navigator can be used in addition to or instead of the built-in browser. Cookies are used with Internet Explorer and Netscape Navigator in order to maintain session state while URL rewriting is employed by the built-in browser for this purpose.

4. ServletApp Usage

A servlet can be run under ServletApp in two different ways. Consider, e.g., the following simple *pure* "HelloWorld" servlet:

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

public class HelloWorld1 extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><body>");
    out.println("<p>Hello World</p>");
    out.println("</body></html>");
    out.close();
  }
}
```

To run this servlet under ServletApp, an additional Java program is needed to register the servlet with the servlet engine, then start the server and browser. The following code will do:

```
import java.io.*;
import servletapp.*;

public class RunHelloWorld1
{
  private static final String SERVLET_NAME
            = "HelloWorld1";
  private static final String SERVLET_URL
            = "/" + SERVLET_NAME;

  public static void main(String[] args)
            throws IOException
  {
    ServletApp.registerServlet(SERVLET_URL,
                               SERVLET_NAME);
    ServletApp.startBrowser("/servlets" +
                            SERVLET_URL);
  }
}
```

Compiling and running the above code requires that the student download servletapp.jar and use the following batch file commands. (We supply compile.bat and run.bat so students don't have to modify CLASSPATH.)

```
compile HelloWorld1.java
compile RunHelloWorld1.java
run RunHelloWorld1
```

The running servlet appears in ServletApp's built-in browser, as shown in Figure 1.

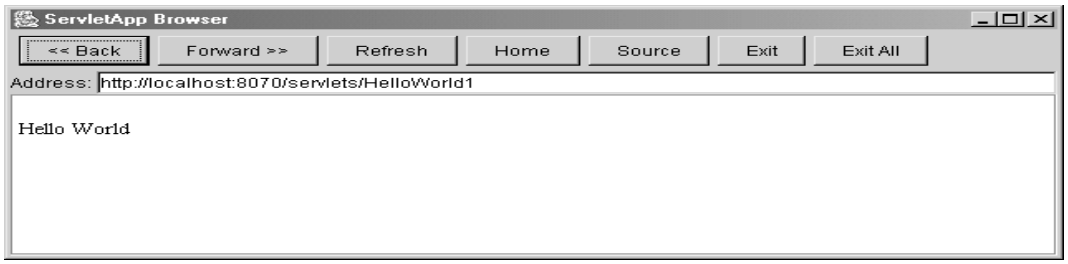

Figure 1: Running a servlet in the built-in browser.

This approach to running ServletApp servlets enjoys the advantage of keeping the servlet code relatively "pure," and also allows running several servlet classes. A second approach keeps all code in one source file, as illustrated by the following code:

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

public class HelloWorld2 extends HttpServlet
{
  private static final String SERVLET_NAME
            = "HelloWorld2";
  private static final String SERVLET_URL
            = "/" + SERVLET_NAME;

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    out.print("<html><body>");
    out.println("<p>Hello World</p>");
    out.println("</body></html>");
    out.close();
  }

  public static void main(String[] args)
            throws IOException
  {
    ServletApp.registerServlet(SERVLET_URL,
                               SERVLET_NAME);
    ServletApp.startBrowser("/servlets"
                            + SERVLET_URL);
  }
}
```

The student compiles and runs the program using the compile and run batch commands, with the same result. This second approach enjoys an advantage from both the instructor's and student's points of view. Instead of having the student submit several files and thereby risk mismatched versions, etc., the student needs to submit but one. The disadvantage of this approach is that there can be just one servlet class. However, one servlet class can play the role of several by using an "operation" parameter; we distribute an example of this kind of servlet to students. (The example is available on our web site -- see the Availability section below.)

There are two things to note concerning the code in this second approach. First, the last import, viz. "import servletapp.*;" must be included in the code for any servlet that is to be run under ServletApp. Second, note the inclusion of a main() method in the HelloWorld2 servlet. These are essentially the only deviations from "normal" servlet programming that are required.

While not demonstrated in the above code, ServletApp can open multiple ServletApp browsers simply by calling startBrowser() more than once. Multiple browser windows simulate multiple users accessing the servlet simultaneously, and can underscore the need for sessions.

5. ServletApp and JSPs

ServletApp even handles Java Server Pages (JSPs)! In order to run a JSP file, the student need only create a "docroot" folder in the current directory, place the JSP file in the docroot folder and point his/her browser at the URL http://localhost:8070/<*jspfile*>. ServletApp puts the generated servlet source and class files in an automatically-created "temp" folder in the current directory, so students can easily see how the JSP process works.

For example, the following JSP can be placed in docroot/hello.jsp:

```
Hello! Here's the date: <%= new java.util.Date() %>
```

The following ServletApp program will serve the JSP file:

```
import java.io.*;
import servletapp.*;

public class HelloJSP
{
  public static void main(String[] args)
          throws IOException
  {
    ServletApp.startBrowser("/hello.jsp");
  }
}
```

The JSP file is recompiled when changed, even when the ServletApp application is running. ServletApp also serves ordinary HTML files from the "docroot" folder.

6. How ServletApp Works

Jetty[6], an open-source HTTP server and servlet/JSP container written in 100% Java, supplies most of ServletApp's functionality. We chose Jetty (version 3.1.7) because it has a reputation of being stable and fast, and is also relatively easy to embed in applications. (This version of Jetty supports version 2.2 of the servlet API and 1.1 of the JSP API.)

ServletApp's registerServlet() method makes sure that the servlet URL starts with a forward slash and that the servlet class file exists, then it calls Jetty's ServletHandler.addServlet() method.

ServletApp's startBrowser() method starts the Jetty server if it is not already running. (There is a separate startServer() method in ServletApp that can start the server explicitly.) The startBrowser() method then loads the URL http://localhost:8070/<*supplied-url*>, where <*supplied-url*> is given as a parameter. The host and port number can be changed with method calls, if desired.

To stop the servlet "application" completely, the student clicks the "Exit All" button on the browser; this stops all browser windows and the Jetty server.

We had to write the browser code, but this task was almost trivial because Java's javax.swing.JEditorPane class handles all HTML rendering.

7. ServletApp in the Classroom

During the fall semester 2002, we used ServletApp in conjunction with code obtained from chapter 9 of Core Servlets and JavaServer Pages[7][i]. The code is quite extensive and consists of the following classes:

- CatalogPage.java: An abstract base class that displays items for sale. It looks up the identifiers for items for sale in the catalog and presents an order page to the user by using the descriptions and prices found therein. This is a servlet class.
- Catalog.java: This class maintains an array of items in inventory. It provides a getItem() method that takes an itemID String value and returns the associated Item.
- Item.java: This class contains itemId, shortDescription, longDescritipion and cost variables. It contains the usual getter and setter methods for these variables.
- ItemOrder.java: This class pairs a catalog Item with an order. It keeps track of the number of items and the total price.

- KidsBookPage.java: This is an extension of the CatalogPage file and, hence, is a servlet class. It displays a page selling books for children.
- TechBooksPage.java: This is another extension of the CartalogPage file and, hence, is also a servlet class. It displays a page selling computer books.
- ShoppingCart: This class is used to track orders. The OrderPage servlet (see next) associates an instance of this class with each user session.
- OrderPage.java: This class displays all items currently in ShoppingCart. Session tracking is incorporated. If this is the first visit to this page a new shopping cart is created. Users can bookmark this page, access it from their history list or be sent back by clicking on "Udate Order". This is a servlet class.
- ServletUtilities.java: A collection of utility methods. The most notable here are two methods for getting cookies and getting the values of cookies.

This collection of classes implements an on-line store using a shopping cart and session tracking. Figure 2 is a screen shot of the KidsBooksPage servlet in action.

Figure 2: Running the KidsBooksPage servlet.

Figure 3 shows a result of several uses of the "Add to Shopping Cart" button.

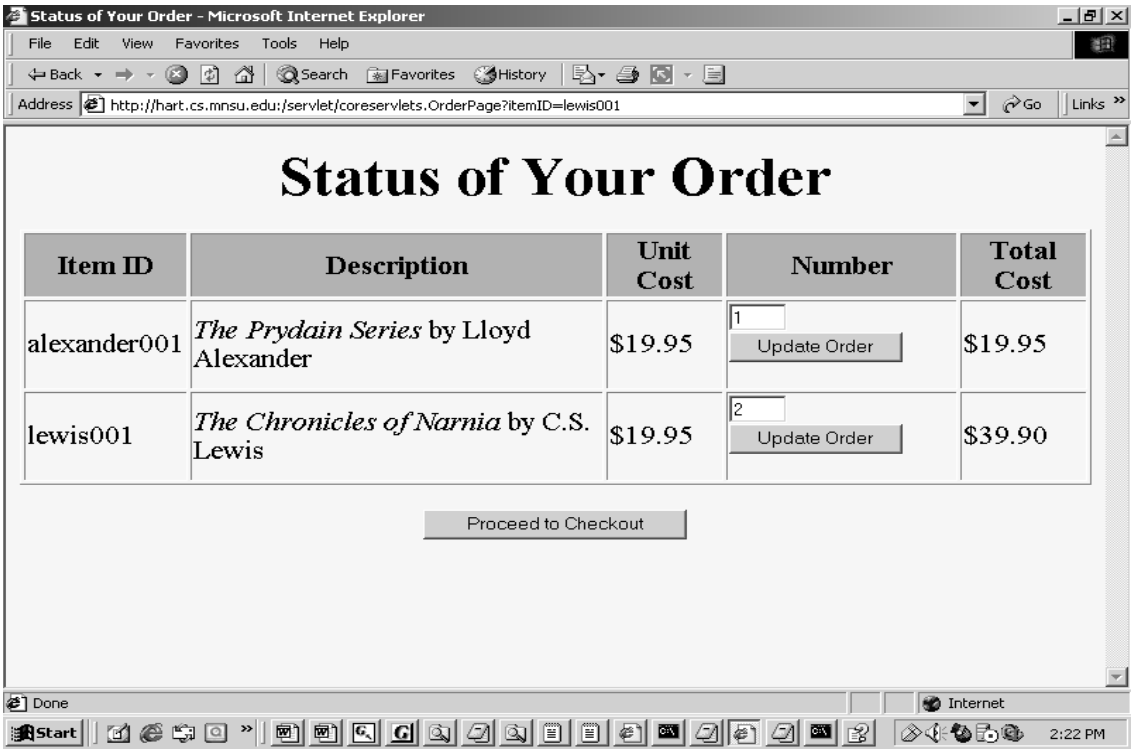

Figure 3: Result of the OrderPage servlet after some items have been placed in the shopping cart.

The missing item from this code is a connection to any real database. Clicking on the "Proceed to Checkout" button (see Figure 3) merely results in an amusing invitation to write out a check to the author of Core Servlets and JavaServer Pages. Our students were "invited" to complete the code by performing the following tasks.

1. Create an Oracle database containing the usual Customer, Product, Order and OrderLine tables (other tables were included but didn't figure into the on-line work).
2. Next, they modified the code so as to reflect the products that they were selling. This involved changing or replacing the code from the KidsBooksPage.java and TechBooksPage.java files. They also were required to modify the Catalog.java file. These changes were mechanical and merely involved some simple substitutions for the code contained in these files.
3. Lastly, our students were required to write their own servlet (SubmitOrder.java) which made a JDBC connection to their database, checked the On_Hand value in the Product table to make sure that there were enough items on hand to fulfill the order (aborting the transaction if there weren't), decremented the On_Hand value appropriately (if there were), inserted customer information into the Customer table (if the order involved a new customer), and inserted the appropriate entries into the Order and Order_Line tables. In order for SubmitOrder.java to collect the

requisite information concerning the contents of the shopping cart and the identity of the customer, two things are required. First, in the OrderPage.java file, the lines

```
ServletContext context = getServletContext();
Vector itemsOrdered = cart.getItemsOrdered();
context.setAttribute("com.orders", itemsOrdered);
```

need to be included and the lines

```
ServletContext context = getServletContext();
Vector itemsOrdered = (Vector) context.getAttribute("com.orders");
```

need to be included in SubmitOrder.java. The setAttribute() and getAttribute() methods of the ServletContext object provide a means for interservlet communication. In the code above, itemsOrdered is a Vector containing the shopping cart items. In OrderPage.java, it is first established as an attribute in the context with a name of "com.orders" via the use of setAttribute() and later retrieved via the use of getAttribute() in SubmitOrder.java. Second, a static HTML file (Checkout.html) containing a form with user information such as name, address, phone, credit card number, etc. is created. This file is referenced in OrderPage.java with the following lines:

```
String checkoutURL =
response.encodeURL("/allan/Checkout.html");
// "Proceed to Checkout" button below table
out.println
("</TABLE>\n" +
"<FORM ACTION=\"" + checkoutURL + "\">\n" +
"<BIG><CENTER>\n" +
"<INPUT TYPE=\"SUBMIT\"\n" +
"VALUE=\"Proceed to Checkout\">\n" +
</CENTER></BIG></FORM>");
```

The information contained in Checkout.html is collected with SubmitOrder.java via the use of the standard getParameter() method of the HttpServletRequest parameter of the doPost() method..

At this point, the changes to the database were commited Extra credit points were available for using the Java Mail API to send an appropriate email to the customer. Students were able to see if their database changes had taken place simply by making the appropriate queries using an Oracle client interface.

8. Future Work

We intend to continue our work with ServletApp in the future. In particular, we intend to extend ServletApp so that it can handle JSP custom tag libraries. Integration of ServletApp with JNDI[8] and Java security services is also planned.

9. Conclusions

We used a prototype of ServletApp in our database classes during the Spring semester 2002. The prototype supported only a small portion of the servlet specification, and did not support JSPs at all. Nevertheless, most students found ServletApp easy and fun to use. The current version of ServletApp was used in our database classes during the Fall semester 2002. Again our students found ServletApp an easy tool to use. ServletApp can be used in a wide variety of classroom scenarios. E.g., an instructor wishing to provide his/her students with a less involved introduction to web-based database connectivity, might have his/her students implement a simple guest book servlet using ServletApp. Because ServletApp handles JSPs, an instructor wishing to avoid introducing his/her students to the complexities involved in writing servlets, but wanting to introduce his/her students to dynamic web content via JSPs, might use ServletApp for that purpose. ServletApp has its uses not only in database courses. Any instructor whose course anticipate servlet and/or JSP programming as potential components should find ServletApp a welcome addition to their toolkits.

10. Availability

ServletApp is licensed under the Jetty License, which is open-source license. ServletApp is freely available from

http://jslack.cs.mnsu.edu/home/downloads/servletapp
http://hart.cs.mnsu.edu/home/downloads/servletapp.

11. References

[1] JDBC: http://java.sun.com/products/jdbc/

[2] Java: http://java.sun.com

[3] Servlets: http://java.sun.com/products/servlet/

[4] Tomcat: http://java.sun.com/products/jsp/tomcat/

[5] JSPs: http://java.sun.com/products/jsp/

[6] Jetty: http://jetty.mortbay.org/jetty/

[7] Core Servlets and JavaServer Pages by Marty Hall, © 2000 Prentice Hall

[8] JNDI: http://java.sun.com/products/jndi/

[i] The choice of Core Servlets and JavaServer Pages is appropriate for several reasons. First, as is evidenced by the fact that, at one time, Core Servlets and JavaServer Pages was the best selling Java book on Amazon.com, more than a few programmers have "cut their servlet programming teeth" with this book. Second, the author and publishers have adopted a "try before you buy" philosophy - one can download (for free) the entire text of the book in .pdf format simply by visiting the web site for the book at http://www.coreservelets.com – a great advantage for students with shallow pockets. Third, the author generously permits "unrestricted" use of the example source code in the book.