

Developing 3-Tier Database Applications with Java Servlets

by

Chád Darby

The drive to create a successful web site has resulted in web applications that are interactive and informative. A wealth of information is stored in corporate databases and there is a rush to publish this information on the web. Corporation's traditional client / server applications are being edged-out by web-based applications. This occurrence is possible thanks to the universal client, the web browser.

This article is the second of a three-part series on Java servlets. Last month (JDJ Jan 1998), I gave you an overview of the Java servlet technology and how to migrate your existing CGI scripts to Java servlets. This article will not reintroduce those concepts. I assume that you are familiar with the basics of the Java Servlet API and the Java Database Connection API.

In this article, you will learn how to build a 3-tier database application that uses Java Servlets and the Java Database Connection (JDBC). You will witness the construction of each tier and understand the techniques used to create Java Servlets with database connectivity.

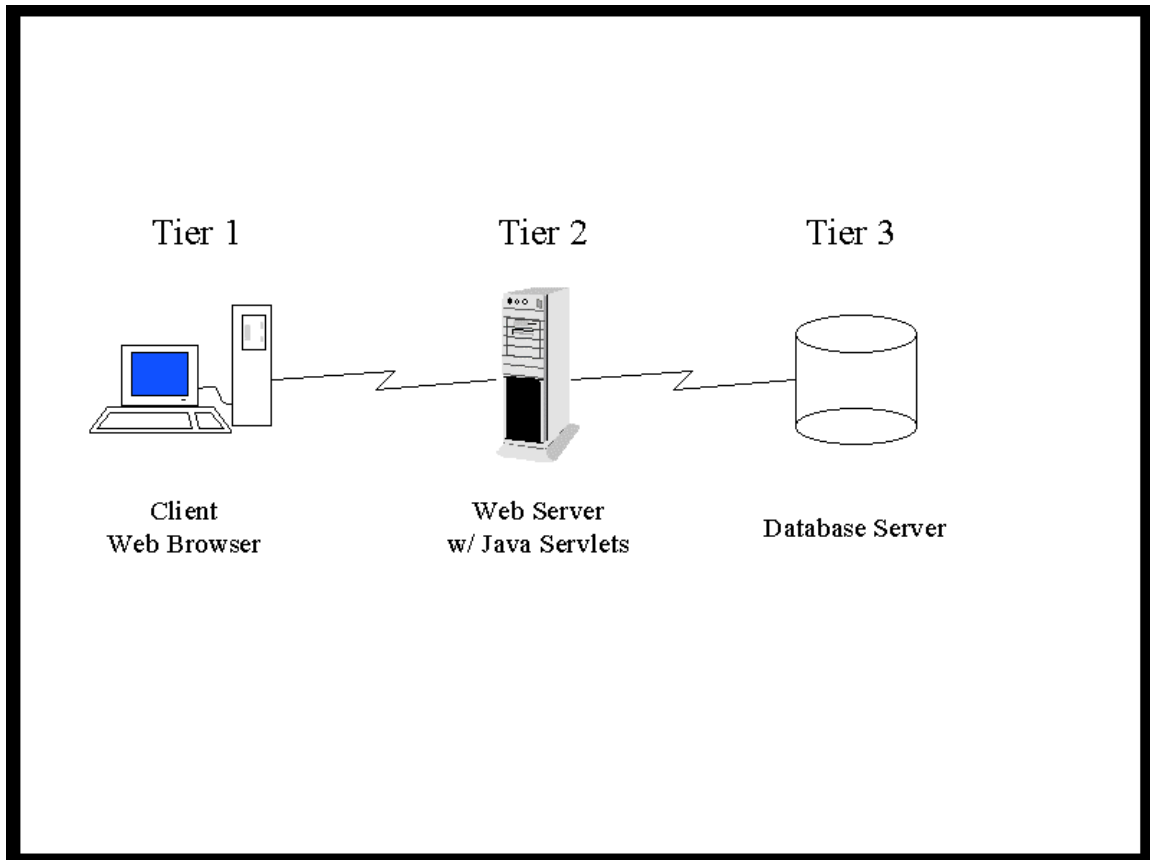
The Challenge

A prominent public speaker keeps track of students who attend her Internet seminars. After each seminar, she exchanges business cards with the interested students. She then enters the student data into her database program.

Instead of entering the data from each business card, she envisions a web application that would do the work for her. At each seminar, the web application is setup on several web terminals. Each student registers at a web terminal and provides their name, company, e-mail address, course title and expectations. The web application also has to option to display an updated list of all students.

3-Tier Solution

The web application is made up of three tiers: web browser, servlet middleware, and database server. The three tiers are illustrated below in figure 1.



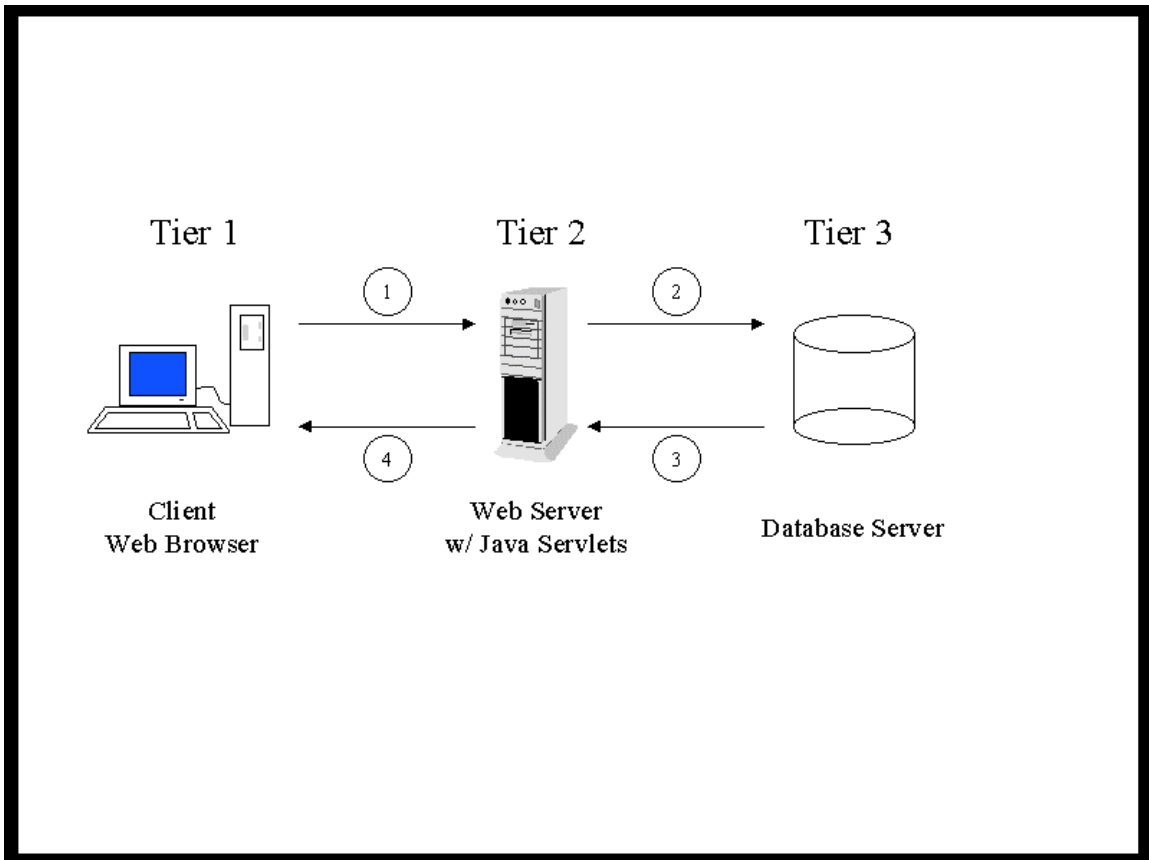
The first tier uses a web browser to take advantage of the installed user base of this universal client. An HTML form is used for user-input. The results of the database query are returned as an HTML page. Using HTML for user-input and displaying the data lowers the requirement of the client's browser version. This web application does not impose the requirement of a Java-enabled browser with the latest JDK patch.

The second tier is implemented with a web server running Java servlets. The Java servlet is able to access the database and return an HTML page listing the data. Please note that Java servlets are not restricted to Sun Microsystem's Java Web Server. You can also use Java servlets with the following servers: Microsoft IIS, Netscape FastTrack and Enterprise Server and O'Reilly WebSite Professional. Servlet functionality is possible with Live Software's JRun product, <http://www.livesoftware.com>. Sun's Java Server page, <http://jserv.javasoft.com>, also has a list of servlet-enabled web servers.

The third tier is the back-end database server. The Java servlet can access information in the database provided a JDBC driver exists. In our situation, the public speaker's database is MS-Access so we can use the JDBC-ODBC driver that is bundled with the Java Development Kit versions 1.1 and higher.

Application Interaction

As you can see, the application is partitioned into three different tiers. Figure 2 illustrates the interaction between the different tiers of the application.



Each step of the interaction is described below.

Step 1:

The user enters information into an HTML form. The form data is passed to the Java servlet running on the web server.

Step 2:

The Java servlet parses the form data and constructs an SQL statement. The SQL statement is passed to the database server using the Java Database Connection (JDBC).

Step 3:

The database server executes the SQL statement and returns a result set to the Java servlet.

Step 4:

The Java servlet processes the result set and constructs an HTML page with the data. The HTML page is then returned to the user's web browser.

Analyzing the Database Schema

Our public speaker is currently storing the student information in a MS Access database. The database contains one data table called Students.

The data fields are defined in the table below

Field Name	Data Type	Length
ID	Autonumber	-

LastName	Text	50
FirstName	Text	50
Email	Text	50
Company	Text	50
CourseTitle	Text	50
CourseLocation	Text	50
CourseStartDate	Date/Time	-

Designing the Web Browser Interface

The browser interface is composed of a main menu page. This page presents the user with the option of student registration or displaying the students in the database. HTML code for the main menu is provided in Listing 1.

Student Registration Form

The students register using an HTML form. The form collects name, e-mail address, company name and other course information. A snapshot of the student registration form is given in figure 3.

The screenshot shows a Netscape browser window titled "Student Registration - Netscape". The page content includes:

- Student Registration** (Main Title)
- Instructions**
 1. Enter your information in the fields below.
 2. Press the **Register** button to enter your information into the course database.
- Form Fields:**
 - First Name:
 - Last Name:
 - E-Mail:
 - Company:
 - Course Title:
 - Course Start Date:
 - Course Location:
- Course Expectations**
 -

The browser's status bar at the bottom shows "Document: Done".

Once the user enters their information then the "Register" button sends the data to the Java servlet.

Developing the Servlet Middleware

The servlet middleware encapsulates the business logic of the application. The servlet parses the form data and constructs an SQL statement. The SQL statement is then passed to the database server. After

executing the SQL statement, the database server returns a result set back to the servlet. At this time, the servlet processes the result set and constructs an HTML page for the user.

The servlet being created is called StudentDBServlet. The StudentDB servlet has methods to perform the following functions: initialization, servicing requests, displaying students, and registering a student. Let's look at each of these functions in detail.

Initializing the Servlet

In the life-cycle of a servlet, the `init()` method is called the first time the servlet is invoked. Listing 3 has the code listing for the `init()` method.

For the StudentDBServlet, a database connection is opened and prepared statements are created for displaying a student list and registering a student. The database connection is left open for the lifetime of the servlet. Depending on your design, you can open and close a connection for each SQL query. However, in this application, the database connection is opened only once.

Servicing User Requests

Whenever a servlet is invoked the `service()` method is called. The `service()` method is the main entry point for servlets. However, if this is the first time the servlet is being invoked, then the `init()` method is called followed by the `service()` method.

The `service()` method in this application is used to branch the request to the appropriate method. The student registration form has a hidden field called *Register*. The service method checks the value of the Register field. If the value is non-null then the `registerStudent()` method is called. If the field does not exist on the HTML page then a null value is returned. A null value results in the execution of the `displayStudents()` method.

Displaying the Student List

The `displayStudents()` method encapsulates the business logic to access the database and display the student list. This is accomplished by using a supporting Student class. The code for the Student class is given in Listing 4.

The Student class has data members to hold information for one student. The Student class also has constructors that can create an object based on form data or a database result set. The code below demonstrates how a student's last name is accessed from the form data.

```
lastName = request.getParameter("LastName");
```

The *request* object is an instance of `HttpServletRequest`. The *request* object contains the form data. The form data is accessed by calling the `getParameter()` method and providing the name of the form field. The student registration form has a *Last Name* field. Refer to my article in the January 1998 issue of JDJ for a detailed discussion of accessing form data with servlets.

The *Student* class has methods for accessing its data members and for providing a string representation of its data. Listing 4 contains the code listing for the methods of the Student class. The `toString()` method returns a normal string version of the data members. The `toWebString()` methods returns the data as an HTML formatted unordered list. The `toTableString()` method returns the data as an HTML formatted table row. These methods are used to build the student list.

Constructing an HTML page creates the student list. In the `displayStudents()` method of Listing 3, the heading of the HTML page is created. Next the table heading is created to display the following information.

Student Name	E-mail	Company	Course Expectations
--------------	--------	---------	---------------------

The servlet sends a request to the database server to get a list of students. The following SQL statement was prepared in the `init()` method.

```
select * from Students order by LastName;
```

The SQL statement will return a list of students listed in alphabetical order based on the last name. The result set is used to create the body of the HTML table. A while-loop is created to iterate through each record of the result set. The code fragment for the while-loop is shown below.

```
int rowNumber = 1;
while (dataResultSet.next())
{
    aStudent = new Student(dataResultSet);
    tableBody += aStudent.toTableString(rowNumber);
    rowNumber++;
}
```

Each record is used to create a new `Student` object. The `toTableString()` method is called to get a string representation of the student data. Recall the `toTableString()` method returns the data as an HTML formatted table row.

After the body of the table is constructed then the result set is closed. At the bottom of the web page, navigation links are provided to the main menu page.

A large amount of server-side processing has taken place. However, we are not finished yet. The HTML page must be returned to the web browser. This is accomplished by opening an output stream on the `response` object. The `response` object is an instance of `HttpServletResponse`. The `response` object is used to respond to the client. The code for returning the HTML page to the user is shown below.

```
PrintWriter outputToBrowser = new PrintWriter(response.getOutputStream());
response.setContentType("text/html");
outputToBrowser.println(htmlPage);
outputToBrowser.close();
```

The content-type is set for HTML and the `htmlPage` string is returned to the browser using the `println()` method. Figure 4 is a sample student list that is returned by the `StudentDBServlet`.

	Student Name	E-mail	Company	Course Expectations
1	berls, Clifton	clif@berls.com	Fly Me Inc	Let's see if Java is all that its cracked up to be. What about some networking, database and GUI stuff all in one week.
2	Brown, Fozxy	fozxy@nana.firm	Fozxy Productions	Data is Queen!
3	Browski, Karl	browski@marine.com	Marine Man	okay, what ever
4	Bundy, Ted	bundy@life.com	Flesh & Bone	I just want the meaty parts of Java
5	Coleman, Rosco	rosco@coleman.com	Youth United	Build a student database just like this
6	Escobar, Nas	nas@qb.firm	The Firm Biz	Database access for SoSi
7	Fowler, Charles	charles@jb.com	JB Lexus	Manage auto inventory using JDBC
8	Hardy, Antonio	kane@bdk.com	Smooth Ops	Create Java apps for SmoothApps database
9	James, Jesse	james@wildwest.com	Wild West	RMI, JDBC, CORBA and DCOM Integration

Registering A Student

The registerStudent() method creates a new Student object based on the HTML form data. The Student object is used to set the parameters on the SQL statement prepared in the init() method. The code fragment below shows how a parameter is set.

```
registerStatement.setString(LAST_NAME_POSITION, aStudent.getLastName());
```

Once all of the parameters are set then the SQL statement is executed. After the statement is executed the new student data is successfully inserted into the database.

A confirmation page is constructed for the user. The confirmation page contains a list of the data that was successfully entered into the database. The Student.toWebString() method is called to provide an HTML string for an unordered list.

Pulling It All Together

At this point, all three tiers of the application are constructed. Collections of HTML pages represent the user interface component for the browser. The only requirement on the browser is the ability to display HTML tables. The two leading browsers available from Microsoft and Netscape easily satisfy this requirement, thus making the web application browser friendly.

The back-end database was developed with Microsoft Access. However, any database could have been used provided that a JDBC driver was available for the database. In our scenario, the public speaker was already tracking student data with MS Access. The web application gave her the ability to access her legacy data and build on it.

The middleware was the critical piece of the application. The servlet middleware encapsulated the business logic and provided the “glue” between the web browser interface and the backend database information. The database access was possible by using a nice blend Java based technologies: Java Servlet API and JDBC.

Each of the components in the 3-tier application can reside on different computers. The application can easily be distributed across the network. With the world wide reach of the web browser, a user can enter information from a networked computer. The Java servlet middleware can reside on any servlet-enabled web server. The servlet can in turn interact with any networked database server in a different location.

Future Enhancements

This 3-tier web application allowed students to register their student information. The application also gave the option of displaying an updated student list. However, the application is by no means complete. There are a number of enhancements that can be made to the application.

The application can be enhanced to display students from a specific city, company or course. A user could create a custom query to generate the student listing. Also, the application can be enhanced to allow remote database administration features such as updating and deleting student entries. Currently, the application does not perform error checking on the form data entered. Client-side JavaScript can be used to verify user data. The application does not consider the case where a student will attend multiple seminars. The application can be enhanced to hold multiple course titles for a student.

Conclusion

This article presented the basic components and techniques to build a 3-tier database application. You can use this information to quickly and easily build a web interface to your existing corporate database. The Java Servlet technology coupled with the Java Database Connection are the key components in creating a web application that is information and interactive.

Resources

Article Source Code Listings

<http://www.j-nine.com/pubs/dbservlets>

Sun Microsystem's Java Server Page

<http://jserv.javasoft.com>

Live Software, JRun 2.0

<http://www.livesoftware.com>

Author By-Line

Chád (shod) Darby is a Java consultant for J9 Consulting, www.j-nine.com. He specializes in developing server-side Java applications and database applications. In his spare time he enjoys running 10K races and half-marathons. Chád can be reached at darby@j-nine.com.