Java Server Pages and Java Beans

Java server pages (JSP) and Java beans work together to create a web application. Java server pages are html pages that also contain regular Java code, which is included between special tags. Java beans are Java programs that follow some specific rules. Together they make up part of a web application.

There are several advantages to using Java server pages and beans. One is that the html and the Java code can be kept separate. Another is that the JSP file is a special kind of html file so that it can be placed anywhere on the server and can include any html tags, including references to other pages, images, applets, etc.

Java server pages are not only html pages. They are also Java programs, so they must be compiled. This is done the first time that they are accessed. After that, the compiled code resides on the server and can be used as is by any succeeding requests. They are first translated into Java servlets, and then the servlets are compiled into class files. On a stand-alone system, you can find both in the folder **work/Standalone/localhost/_**.

Java Server Pages

In a JSP file the Java code is contained between tags that begin with <% and end with %>. These tags are also used in active server pages (asp). There are several different kinds of JSP tags depending upon their use.

- <%= … %> is used for expressions.
- <%! … %> is used for declarations.
- <% … %> is used for straight Java code.
- <%@ … %> is used to include another file such as an html file or a package such as java.sql.*.

There are some reserved words that may be used by JSP files without further definition. These should be familiar from similar terms used with Java servlets.

- request – an instance of HttpServletRequest.
- response – an instance of HttpServletResponse.
- out – a PrintWriter object for the response.
- session – the HttpSession object associated with the session.

Java Beans

Java beans are regular Java programs with a few specific restrictions. The constructor may not have any parameters, and the variables all have get and set accessor and mutator methods. The Java server page uses the accessor and mutator methods of the bean to send values to the bean and to get resulting data back from the bean.

In a Java bean, you can instantiate other classes, access databases, create lists, tables, and anything else you might want to do. You can also have methods that receive request data from the JSP file. They have the usual request parameter as in the following example:

```
public void processRequest (HttpServletRequest request) { … }
```

JSP files must declare which bean they are going to use. This is done with tags that follow xml syntax. They are case sensitive and must have closing tags or a closing ‘/’. A bean declaration for a simple hello world example follows:

```
<jsp:useBean id = "hello" scope = "session" class = "greetings.HelloBean" />
```

The id for the bean is used in place of a name. The declaration links it to the bean class, HelloBean.class. The Java server page may be located anywhere, but the class file goes in the same classes folder as the servlet classes.

HelloBean Example

The bean, HelloBean.java, is very simple. We will see more complicated beans later on. It doesn't have a separate constructor, only the default one without parameters. It also does not do anything other than provide a storage place for the data, the name and the email address. Notice it is in a package called greetings.

```
package greetings;
public class HelloBean
{
	private String name = "";
	private String email = "";

	public String getName() {return name;}
	public String getEmail() {return email;}

	public void setName (String n) {name = n;}
	public void setEmail (String e) {email = e;}
} // HelloBean
```

The Java server page must also set the properties of the bean, i.e. the bean's instance data. This is done with jsp:setProperty tags. These too use xml syntax.

```
	<jsp:setProperty name="hello" property="name" value='<%= request.getParameter ("name") %>' />
	<jsp:setProperty name="hello" property="email" value='<%= request.getParameter ("email") %>' />
```

The name attribute is the same as the bean id, hello. The property is the name of the instance variable in the bean. And the value is read from the html input data.

The html page used for this example is also very simple.

```
<html>
<head><title>Hello</title></head>

<body bgcolor="white">
<font size=5 color="blue">
<form method="get" action="hello.jsp">
	<br />Enter your name and email address:
	<br /><input type="text" name="name" value="" size="20"> Name
	<br /><input type="text" name="email" value="" size="20"> Email
	<p> <input type="submit" name="Send" value="Send"></p>
</form>
</font></body></html>
```

The JSP file combines both html and Java code. One that can tie together the html and bean files above follows:

```
<html>
<head><title>Hello JSP</title></head>
```

```
<body bgcolor="white">
<font size=5 color="blue">
<%! String name, email; %>

<jsp:useBean id="hello" scope="session" class="greetings.HelloBean" />
<jsp:setProperty name="hello" property="name" value='<%= request.getParameter ("name") %>' />
<jsp:setProperty name="hello" property="email" value='<%= request.getParameter ("email") %>' />

<%  name = hello.getName();
      email = hello.getEmail();
%>

Hello, your name is <% out.println (name); %>
<br />And your email address is <% out.println (email); %>
</font></body></html>
```

The result looks like:

Hello, your name is Alice Lee
And your email address is alee@aol.com

where the request data sent by the html file was Alice Lee for the name and alee@aol.com for the e-mail address.

The setProperty tags in the Java server page can be replaced by

```
<jsp:setProperty name = "hello" property = "*" />
```

if the names in the JSP and bean files are the same. This doesn't always work, but in most cases it saves a lot of writing. However, when using this shortcut, there are several conventions that must be followed. First, the accessor and mutator methods must be of the form

```
getEmail ()
setEmail (String e)
```

where the identifier in the web page and bean is 'email'. If instead, the variable was called 'eMail', the get and set methods would be

```
getEMail ()
setEMail (String e)
```

The first letter after the get or set must be capitalized. The rest of the letters follow the capitalization in the variable. This also means that all variables should begin with lower case letters.

FindBean Example

A somewhat more complicated example can be created to find an object in a database. The database access is done in the bean while the html display is created by the Java server page. The following shows a simple Access table.

ProduceTable : Table

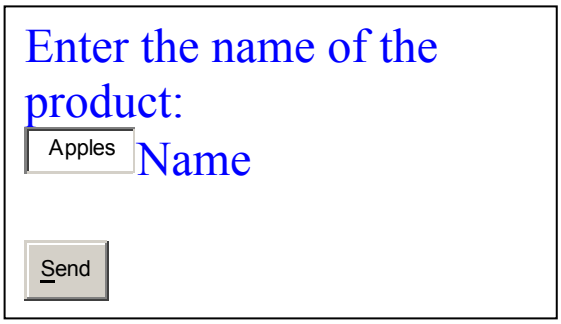
ID	Type	Name	Variety	Price
AF136	Fruit	Apples	Red Delicious	1.25
AV213	Vegetable	Broccoli	Flowerets	2.65
BF214	Fruit	Oranges	Florida	2.30
BV247	Vegetable	Carrots	Baby	1.25
CF159	Fruit	Pear	Bosc	2.35
CV364	Vegetable	Beans	Yellow	1.35
DF123	Fruit	Bananas	Small	0.89
EF514	Fruit	Grapes	Red Seedless	3.95
GV524	Vegetable	Peas	Snow	1.25

Record: 1 of 9

A small html page can be used to find the product given the product's name.

```
<html>
<head><title>Find Product</title></head>

<body bgcolor="white">
<form method="get" action="find.jsp">
        <br>
        <font size=5 color="blue">
        Enter the name of the product: <br>
        <input type="text" name="name" value="" size="20"> Name <br />
        <br /> <input type="submit" name="action" value="Send">
        </font>
</form>
</body></html>
```

In a browser, this looks like:

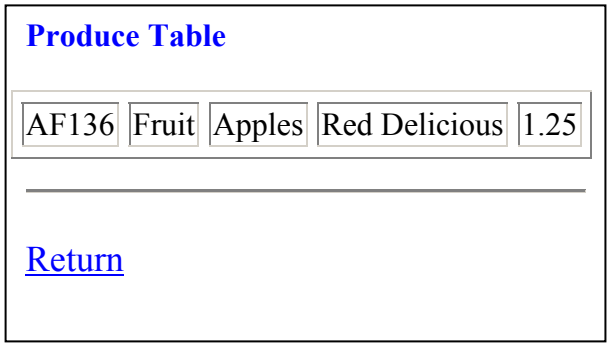


The Java server page, find.jsp, contains code to both use the bean, FindBean, and to display the results of the search. The bean is in a package called produce, so it is referred to as "produce.FindBean". The bean id may be anything, but calling it FindBean makes it clear what bean is meant. The only method in FindBean other than gets and sets is called processRequest. It has no parameters.

```
<html>
<head><title> Find Produce JSP. </title></head>

<body bgcolor = "white">
<font size = "4" color="blue">

<jsp:useBean id = "findBean" scope = "session" class = "produce.FindBean" />
<jsp:setProperty name = "findBean" property = "*" />

<% findBean.processRequest(); %>

<% if (findBean.getFound ()) { %>
<h4>Produce Table</h4>
<table border = "1" cellspacing = "5">
      <tr>
      <td><% out.println (findBean.getId()); %></td>
      <td><% out.println (findBean.getType()); %></td>
      <td><% out.println (findBean.getName()); %></td>
      <td><% out.println (findBean.getVariety()); %></td>
      <td><% out.println (findBean.getPrice()); %></td>
      </tr>
</table>
<% } else { %>
      <h4>The product is not in the database.</h4>
<% } %>
<hr>
<p><a href  = "../market/find.html">Return</a></p>
</font>
</body></html>
```

If the product is in the database, the result is a table row with the results. But if the product was not found, an error message will be displayed.

The Return reference takes you back to find.html.



The Java bean, FindBean.java, is quite brief. It accesses the database and uses the results to set the values for the instance variables.

```
package produce;
import java.sql.*;
import java.io.*;

// FindBean locates the product in the database and gets values for the instance variables.
public class FindBean
{
	private String id, type, name, variety, price;
	private boolean found;

	// The accessor methods.
	public String getId() {return id;}
	public String getType() {return type;}
	public String getName() {return name;}
	public String getVariety() {return variety;}
	public String getPrice() {return price;}
	public boolean getFound () {return found;}

	public void setName (String n) {name = n;} // The only set method needed.

	public void processRequest ()
	{
		try
		{
			// Get a jdbc-odbc bridge and connect to produce.mdb.
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection con = DriverManager.getConnection ("jdbc:odbc:produce");

			// Create a statement to query the database.
			Statement stmt = con.createStatement ();
			String query = "Select * From ProduceTable Where Name = '" + name + "'";
			ResultSet rs = stmt.executeQuery (query);

			// If the result set is not empty, use the result set to get values for the instance variables.
			if (rs.next ())
			{
				id = rs.getString ("ID");
				type = rs.getString ("Type");
				name = rs.getString ("Name");
				variety = rs.getString ("Variety");
				price = rs.getString ("Price");
				found = true;
			}
			else found = false;
		} catch (ClassNotFoundException e){System.out.println ("Class Not Found exception.\n");}
		  catch (SQLException e){System.out.println ("SQL Exception");}
	} // processRequest
} // FindBean
```

Displaying the Database

The result set that a Select query returns cannot be sent to the Java server page. But the data can be stored in a data structure such as an array, list or hashtable. The following example stores it as a two dimensional array where row 0 contains the column names and the data starts in row 1 column 1. This means that column 0 is never used. But that really doesn't matter.

The Java bean for this application has to contact the database and then create the table. The table may then be used by the Java server page to display the data in an html table. The Java bean is shown first.

DisplayBean.java

```
//     DisplayBean gets data from a database and creates a table containing the data.

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

//     DisplayBean gets the data from the database and creates a table called tableData.
public class DisplayBean
{
      private String [][] tableData;
      private int rows;
      private int columns;

      private Connection con;

      public String [][] getTableData () {return tableData;}
      public int getRows () {return rows;}
      public int getColumns () {return columns;}

      public void setTableData (String [][] t) {tableData = t;}
      public void setRows (int r) {rows = r;}
      public void setCols (int c) {columns = c;}

      public void processRequest ()
      {
            try
            {
                  // Get a jdbc-odbc bridge and connect to produce.mdb.
                  Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                  con = DriverManager.getConnection ("jdbc:odbc:produce");

                  // Create a statement to query the database.
                  Statement stmt = con.createStatement ();
                  String query = "Select * From ProduceTable";
                  ResultSet rs = stmt.executeQuery (query);
                  ResultSetMetaData metaData = rs.getMetaData ();
                  columns = metaData.getColumnCount ();
```

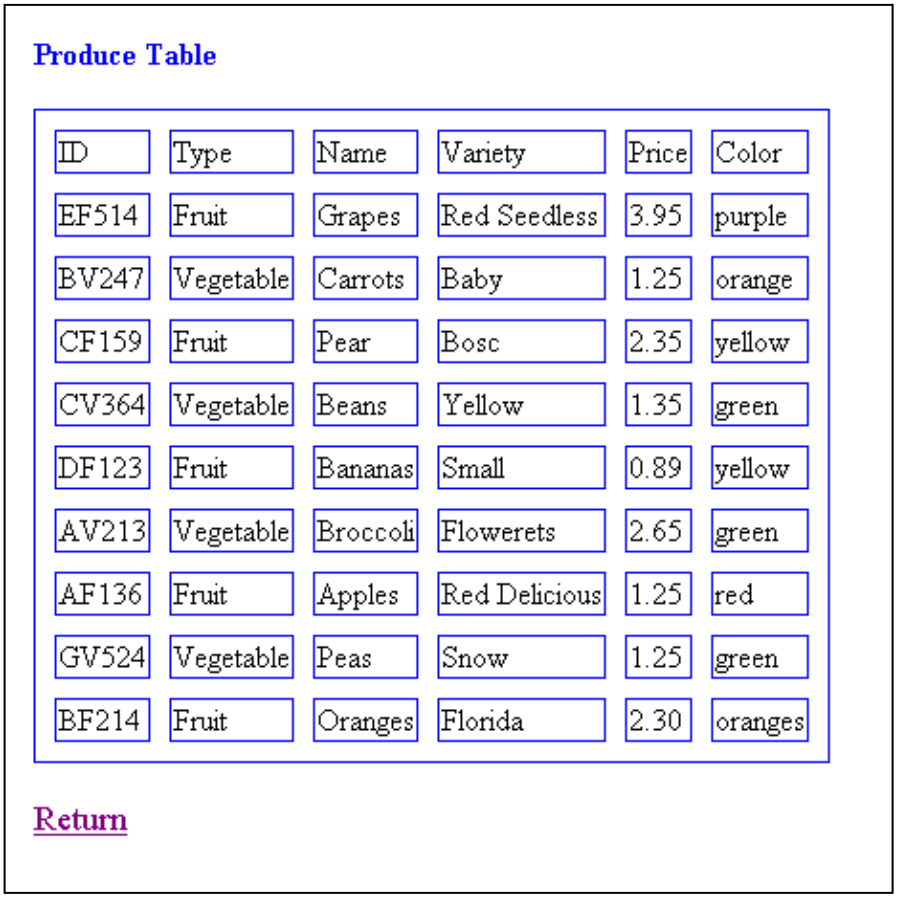
```
                // Get a new instance of the table and fill it.
                tableData = new String [20][columns+1];
                for (int count = 1; count <= columns; count ++)
                      tableData [0][count] = metaData.getColumnName (count);
                rows = 1;
                while (rs.next ())
                {
                      for (int col = 1; col<= columns; col ++)
                            tableData [rows][col] = rs.getString (col);
                      rows ++;
                }
                con.close ();
          } catch (ClassNotFoundException e){System.out.println ("Class Not Found exception.\n");}
            catch (SQLException e){System.out.println ("SQL Exception\n");}
      } // processRequest
} // class DisplayBean
```

The Java server page receives the table and uses it to display the results on an html page.

display.jsp

```
<html>
<head><title> Display Products JSP. </title></head>

<body bgcolor="white">
<font size=4 color="blue">

<%! String [][] tableData;
      int rows, columns, row, col;
%>
<jsp:useBean id="displayBean" scope="session" class="produce.DisplayBean" />
<jsp:setProperty name="displayBean" property="*" />

<% displayBean.processRequest(); %>
<h4>Produce Table</h4>

<%
      tableData = displayBean.getTableData ();
      rows = displayBean.getRows ();
      columns = displayBean.getColumns ();
%>

<table border='1' bordercolor='blue' cellspacing='10'>
      <%  for (int row = 0; row < rows; row ++) { %>
                <tr>
                <%  for (int col = 1; col <= columns; col ++)  { %>
                      <td> <% out.println (tableData [row][col]); %></td>
                      <% } %>
                </tr>
          <% } %>
</table>
```

```
<p><a href="../market/index.html">Return</a></p>
</font></body></html>
```



Produce Table

ID	Type	Name	Variety	Price	Color
EF514	Fruit	Grapes	Red Seedless	3.95	purple
BV247	Vegetable	Carrots	Baby	1.25	orange
CF159	Fruit	Pear	Bosc	2.35	yellow
CV364	Vegetable	Beans	Yellow	1.35	green
DF123	Fruit	Bananas	Small	0.89	yellow
AV213	Vegetable	Broccoli	Flowerets	2.65	green
AF136	Fruit	Apples	Red Delicious	1.25	red
GV524	Vegetable	Peas	Snow	1.25	green
BF214	Fruit	Oranges	Florida	2.30	oranges

Return

References

1. Marty Hall & Larry Brown, *Core Servlets and Java Server Pages*, First Edition, Sun Microsystems Press/Prentice-Hall PTR Book, 2003.
2. W3 Schools Online Web Tutorials, http://www.w3schools.com.