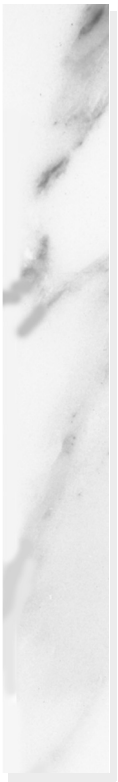
USING JAVABEANS WITH JSP



Topics in This Chapter

- Creating and accessing beans
- Installing bean classes on your server
- Setting bean properties explicitly
- Associating bean properties with input parameters
- Automatic conversion of bean property types
- Sharing beans among multiple JSP pages and servlets

Chapter 13

The JavaBeans API provides a standard format for Java classes. Visual manipulation tools and other programs can automatically discover information about classes that follow this format and can then create and manipulate the classes without the user having to explicitly write any code.

Full coverage of JavaBeans is beyond the scope of this book. If you want details, pick up one of the many books on the subject or see the documentation and tutorials at `http://java.sun.com/beans/docs/`. For the purposes of this chapter, all you need to know about beans are three simple points:

1. **A bean class must have a zero-argument (empty) constructor.** You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors, which results in an empty constructor being created automatically. The empty constructor will be called when JSP elements create beans.
2. **A bean class should have no public instance variables (fields).** I hope you already follow this practice and use accessor methods instead of allowing direct access to the instance variables. Use of accessor methods lets you impose constraints on variable values (e.g., have the `setSpeed` method of your `Car` class disallow negative speeds), allows you to change your internal data structures without changing the class interface (e.g.,

change from English units to metric units internally, but still have `getSpeedInMPH` and `getSpeedInKPH` methods), and automatically perform side effects when values change (e.g., update the user interface when `setPosition` is called).

3. **Persistent values should be accessed through methods called `get`*`Xxx`* and `set`*`Xxx`*.** For example, if your `Car` class stores the current number of passengers, you might have methods named `getNumPassengers` (which takes no arguments and returns an `int`) and `setNumPassengers` (which takes an `int` and has a `void` return type). In such a case, the `Car` class is said to have a *property* named `numPassengers` (notice the lowercase n in the property name, but the uppercase N in the method names). If the class has a `get`*`Xxx`* method but no corresponding `set`*`Xxx`*, the class is said to have a read-only property named *xxx*.

   The one exception to this naming convention is with boolean properties: they use a method called `is`*`Xxx`* to look up their values. So, for example, your `Car` class might have methods called `isLeased` (which takes no arguments and returns a `boolean`) and `setLeased` (which takes a `boolean` and has a `void` return type), and would be said to have a `boolean` property named `leased` (again, notice the lowercase leading letter in the property name).

   Although you can use JSP scriptlets or expressions to access arbitrary methods of a class, standard JSP actions for accessing beans can only make use of methods that use the `get`*`Xxx`*/`set`*`Xxx`* or `is`*`Xxx`*/`set`*`Xxx`* design pattern.

13.1 Basic Bean Use

The `jsp:useBean` action lets you load a bean to be used in the JSP page. Beans provide a very useful capability because they let you exploit the reusability of Java classes without sacrificing the convenience that JSP adds over servlets alone.

The simplest syntax for specifying that a bean should be used is:

```
<jsp:useBean id="name" class="package.Class" />
```

This usually means "instantiate an object of the class specified by `Class`, and bind it to a variable with the name specified by `id`." So, for example, the JSP action

```
<jsp:useBean id="book1" class="coreservlets.Book" />
```

can normally be thought of as equivalent to the scriptlet

```
<% coreservlets.Book book1 = new coreservlets.Book(); %>
```

Although it is convenient to think of `jsp:useBean` as being equivalent to building an object, `jsp:useBean` has additional options that make it more powerful. As we'll see in Section 13.4 (Sharing Beans), you can specify a `scope` attribute that makes the bean associated with more than just the current page. If beans can be shared, it is useful to obtain references to existing beans, so the `jsp:useBean` action specifies that a new object is instantiated only if there is no existing one with the same `id` and `scope`.

Rather than using the `class` attribute, you are permitted to use `beanName` instead. The difference is that `beanName` can refer either to a class or to a file containing a serialized bean object. The value of the `beanName` attribute is passed to the `instantiate` method of `java.beans.Bean`.

In most cases, you want the local variable to have the same type as the object being created. In a few cases, however, you might want the variable to be declared to have a type that is a superclass of the actual bean type or is an interface that the bean implements. Use the `type` attribute to control this, as in the following example:

```
<jsp:useBean id="thread1" class="MyClass" type="Runnable" />
```

This use results in code similar to the following being inserted into the `_jspService` method:

```
Runnable thread1 = new MyClass();
```

Note that since `jsp:useBean` uses XML syntax, the format differs in three ways from HTML syntax: the attribute names are case sensitive, either single or double quotes can be used (but one or the other *must* be used), and the end of the tag is marked with `/>`, not just `>`. The first two syntactic differences apply to all JSP elements that look like `jsp:`*xxx*. The third difference applies unless the element is a container with a separate start and end tag.

Core Warning

Syntax for `jsp:`xxx elements differs in three ways from HTML syntax: attribute names are case sensitive, you must enclose the value in single or

double quotes, and noncontainer elements should end the tag with `/>`*, not just* `>`*.*

There are also a few character sequences that require special handling in order to appear inside attribute values:

- To get `'` within an attribute value, use `\'`.
- To get `"` within an attribute value, use `\"`.
- To get `\` within an attribute value, use `\\`.
- To get `%>` within an attribute value, use `%\>`.
- To get `<%` within an attribute value, use `<\%`.

Accessing Bean Properties

Once you have a bean, you can access its properties with `jsp:getProperty`, which takes a `name` attribute that should match the `id` given in `jsp:useBean` and a `property` attribute that names the property of interest. Alternatively, you could use a JSP expression and explicitly call a method on the object that has the variable name specified with the `id` attribute. For example, assuming that the `Book` class has a `String` property called `title` and that you've created an instance called `book1` by using the `jsp:useBean` example just given, you could insert the value of the `title` property into the JSP page in either of the following two ways:

```
<jsp:getProperty name="book1" property="title" />
<%= book1.getTitle() %>
```

The first approach is preferable in this case, since the syntax is more accessible to Web page designers who are not familiar with the Java programming language. However, direct access to the variable is useful when you are using loops, conditional statements, and methods not represented as properties.

If you are not familiar with the concept of bean properties, the standard interpretation of the statement "this bean has a property of type `T` called `foo`" is "this class has a method called `getFoo` that returns something of type `T` and has another method called `setFoo` that takes a `T` as an argument and stores it for later access by `getFoo`."

Setting Bean Properties: Simple Case

To modify bean properties, you normally use `jsp:setProperty`. This action has several different forms, but with the simplest form you just supply three

attributes: `name` (which should match the `id` given by `jsp:useBean`), `property` (the name of the property to change), and `value` (the new value). Section 13.3 (Setting Bean Properties) discusses some alternate forms of `jsp:setProperty` that let you automatically associate a property with a request parameter. That section also explains how to supply values that are computed at request time (rather than fixed strings) and discusses the type conversion conventions that let you supply string values for parameters that expect numbers, characters, or boolean values.

An alternative to using the `jsp:setProperty` action is to use a scriptlet that explicitly calls methods on the bean object. For example, given the `book1` object shown earlier in this section, you could use either of the following two forms to modify the `title` property:

```
<jsp:setProperty name="book1"
                 property="title"
                 value="Core Servlets and JavaServer Pages" />
<% book1.setTitle("Core Servlets and JavaServer Pages"); %>
```

Using `jsp:setProperty` has the advantage that it is more accessible to the nonprogrammer, but direct access to the object lets you perform more complex operations such as setting the value conditionally or calling methods other than `get`*`Xxx`* or `set`*`Xxx`* on the object.

Installing Bean Classes

The class specified for the bean must be in the server's regular class path, not the part reserved for classes that get automatically reloaded when they change. For example, in the Java Web Server, the main bean class and all the auxiliary classes it uses should go in the *`install_dir`*`/classes` directory or be in a JAR file in *`install_dir`*`/lib`, not in *`install_dir`*`/servlets`. Since Tomcat and the JSWDK don't support auto-reloading servlets, bean classes can be installed in any of the normal servlet directories. For Tomcat 3.0, assuming you haven't defined your own Web application, the primary directory for servlet class files is *`install_dir`*`/webpages/WEB-INF/classes`; for the JSWDK, the default location is *`install_dir`*`/webpages/WEB-INF/servlets`. With all three servers, remember that a package name corresponds to a subdirectory. So, for example, a bean called `Fordhook` that declares "`package lima;`" would typically be installed in the following locations:

- **Tomcat 3.0:**
  `install_dir``/webpages/WEB-INF/classes/lima/Fordhook.class`

- **JSWDK 1.0.1:**
  install_dir/webpages/WEB-INF/servlets/lima/Fordhook.class
- **Java Web Server 2.o:**
  install_dir/classes/lima/Fordhook.class

The JSP files that *use* bean classes don't need to be installed anywhere special, however. As is usual with JSP files on a JSP-capable server, they can be placed anywhere that normal Web pages can be.

13.2 Example: StringBean

Listing 13.1 presents a simple class called `StringBean` that is in the `coreservlets` package. Because the class has no public instance variables (fields) and has a zero-argument constructor since it doesn't declare any explicit constructors, it satisfies the basic criteria for being a bean. Since `StringBean` has a method called `getMessage` that returns a `String` and another method called `setMessage` that takes a `String` as an argument, in beans terminology the class is said to have a `String` parameter called `message`.

Listing 13.2 shows a JSP file that uses the `StringBean` class. First, an instance of `StringBean` is created with the `jsp:useBean` action as follows:

```
<jsp:useBean id="stringBean" class="coreservlets.StringBean" />
```

After this, the `message` property can be inserted into the page in either of the following two ways:

```
<jsp:getProperty name="stringBean" property="message" />
<%= stringBean.getMessage() %>
```

The `message` property can be modified in either of the following two ways:

```
<jsp:setProperty name="stringBean"
                 property="message"
                 value="some message" />
<% stringBean.setMessage("some message"); %>
```

Figure 13–1 shows the result.

Listing 13.1 StringBean.java

```
package coreservlets;

/** A simple bean that has a single String property
 *  called message.
 */

public class StringBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

Listing 13.2 StringBean.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using JavaBeans with JSP</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Using JavaBeans with JSP</TABLE>

<jsp:useBean id="stringBean" class="coreservlets.StringBean" />

<OL>
<LI>Initial value (getProperty):
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>

<LI>Initial value (JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                     property="message"
                     value="Best string bean: Fortex" />
    Value after setting property with setProperty:
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>

<LI><% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>

</BODY>
</HTML>
```

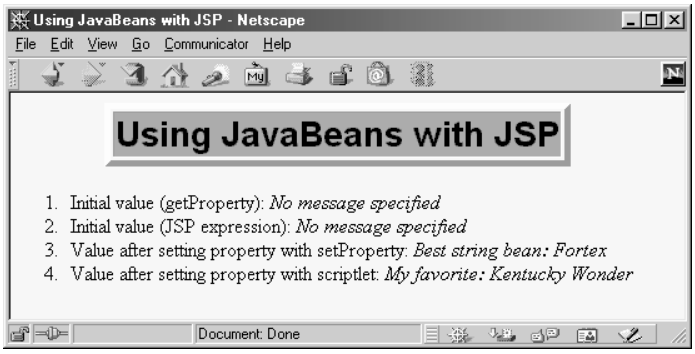


Figure 13–1 Result of `StringBean.jsp`.

13.3 Setting Bean Properties

You normally use `jsp:setProperty` to set bean properties. The simplest form of this action takes three attributes: `name` (which should match the `id` given by `jsp:useBean`), `property` (the name of the property to change), and `value` (the new value).

For example, the `SaleEntry` class shown in Listing 13.3 has an `itemID` property (a `String`), a `numItems` property (an `int`), a `discountCode` property (a `double`), and two read-only properties `itemCost` and `totalCost` (each of type `double`). Listing 13.4 shows a JSP file that builds an instance of the `SaleEntry` class by means of:

```
<jsp:useBean id="entry" class="coreservlets.SaleEntry" />
```

The results are shown in Figure 13–2.

Once the bean is instantiated, using an input parameter to set the `itemID` is straightforward, as shown below:

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

Notice that I used a JSP expression for the `value` parameter. Most JSP attribute values have to be fixed strings, but the `value` and `name` attributes of `jsp:setProperty` are permitted to be request-time expressions. If the expression uses double quotes internally, recall that single quotes can be used instead of double quotes around attribute values and that `\'` and `\"` can be used to represent single or double quotes within an attribute value.

Listing 13.3 SaleEntry.java

```
package coreservlets;

/** Simple bean to illustrate the various forms
 *  of jsp:setProperty.
 */

public class SaleEntry {
  private String itemID = "unknown";
  private double discountCode = 1.0;
  private int numItems = 0;

  public String getItemID() {
    return(itemID);
  }

  public void setItemID(String itemID) {
    if (itemID != null) {
      this.itemID = itemID;
    } else {
      this.itemID = "unknown";
    }
  }

  public double getDiscountCode() {
    return(discountCode);
  }

  public void setDiscountCode(double discountCode) {
    this.discountCode = discountCode;
  }

  public int getNumItems() {
    return(numItems);
  }

  public void setNumItems(int numItems) {
    this.numItems = numItems;
  }

  // Replace this with real database lookup.

  public double getItemCost() {
    double cost;
    if (itemID.equals("a1234")) {
      cost = 12.99*getDiscountCode();
    } else {
      cost = -9999;
```

Listing 13.3 SaleEntry.java (continued)

```
    }
    return(roundToPennies(cost));
  }

  private double roundToPennies(double cost) {
    return(Math.floor(cost*100)/100.0);
  }

  public double getTotalCost() {
    return(getItemCost() * getNumItems());
  }
}
```

Listing 13.4 SaleEntry1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using jsp:setProperty</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Using jsp:setProperty</TABLE>

<jsp:useBean id="entry" class="coreservlets.SaleEntry" />

<jsp:setProperty
    name="entry"
    property="itemID"
    value='<%= request.getParameter("itemID") %>' />

<%
int numItemsOrdered = 1;
try {
  numItemsOrdered =
    Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {}
%>
```

Listing 13.4 `SaleEntry1.jsp` (continued)

```
<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>" />

<%
double discountCode = 1.0;
try {
  String discountString =
    request.getParameter("discountCode");
  // Double.parseDouble not available in JDK 1.1.
  discountCode =
    Double.valueOf(discountString).doubleValue();
} catch(NumberFormatException nfe) {}
%>
<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />

<BR>
<TABLE ALIGN="CENTER" BORDER=1>
<TR CLASS="COLORED">
  <TH>Item ID<TH>Unit Price<TH>Number Ordered<TH>Total Price
<TR ALIGN="RIGHT">
  <TD><jsp:getProperty name="entry" property="itemID" />
  <TD>$<jsp:getProperty name="entry" property="itemCost" />
  <TD><jsp:getProperty name="entry" property="numItems" />
  <TD>$<jsp:getProperty name="entry" property="totalCost" />
</TABLE>

</BODY>
</HTML>
```

Associating Individual Properties with Input Parameters

Setting the `itemID` property was easy since its value is a `String`. Setting the `numItems` and `discountCode` properties is a bit more problematic since their values must be numbers and `getParameter` returns a `String`. Here is the somewhat cumbersome code required to set `numItems`:

```
<%
int numItemsOrdered = 1;
try {
  numItemsOrdered =
    Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {}
%>
<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>" />
```

Fortunately, JSP has a nice solution to this problem that lets you associate a property with a request parameter and that automatically performs type conversion from strings to numbers, characters, and boolean values. Instead of using the `value` attribute, you use `param` to name an input parameter. The value of this parameter is automatically used as the value of the property, and simple type conversions are performed automatically. If the specified input parameter is missing from the request, no action is taken (the system does not pass `null` to the associated property). So, for example, setting the `numItems` property can be simplified to:

```
<jsp:setProperty
```

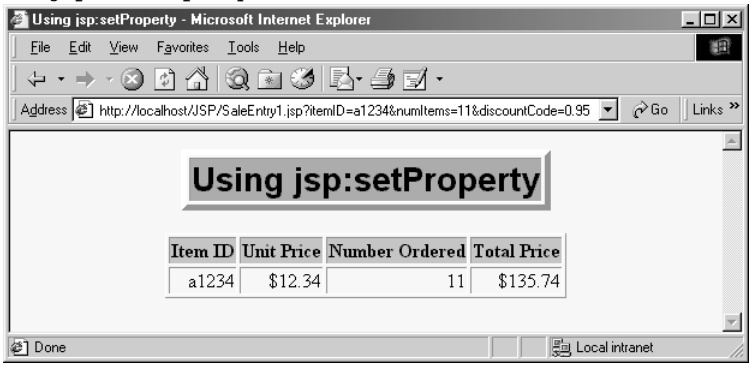


Figure 13–2 Result of `SaleEntry1.jsp`.

```
name="entry"
property="numItems"
param="numItems" />
```

Listing 13.5 shows the entire JSP page reworked in this manner.

Listing 13.5 SaleEntry2.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using jsp:setProperty</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Using jsp:setProperty</TABLE>

<jsp:useBean id="entry" class="coreservlets.SaleEntry" />

<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />

<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />

<%-- WARNING! Both the JSWDK 1.0.1 and the Java Web Server
              have a bug that makes them fail on double
              type conversions of the following sort.
--%>
<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />

<BR>
<TABLE ALIGN="CENTER" BORDER=1>
<TR CLASS="COLORED">
  <TH>Item ID<TH>Unit Price<TH>Number Ordered<TH>Total Price
<TR ALIGN="RIGHT">
  <TD><jsp:getProperty name="entry" property="itemID" />
  <TD>$<jsp:getProperty name="entry" property="itemCost" />
  <TD><jsp:getProperty name="entry" property="numItems" />
  <TD>$<jsp:getProperty name="entry" property="totalCost" />
</TABLE>

</BODY>
</HTML>
```

Automatic Type Conversions

Table 13.1 summarizes the automatic type conversions performed when a bean property is automatically associated with an input parameter. One warning is in order, however: both JSWDK 1.0.1 and the Java Web Server 2.0 have a bug that causes them to crash at page translation time for pages that try to perform automatic type conversions for properties that expect `double` values. Tomcat and most commercial servers work as expected.

Core Warning

With the JSWDK and the Java Web Server, you cannot associate properties that expect double-precision values with input parameters.

Table 13.1 Type Conversions When Properties Are Associated with Input Parameters

Property Type	*Conversion Routine*
`boolean`	`Boolean.valueOf(paramString).booleanValue()`
`Boolean`	`Boolean.valueOf(paramString)`
`byte`	`Byte.valueOf(paramString).byteValue()`
`Byte`	`Byte.valueOf(paramString)`
`char`	`Character.valueOf(paramString).charValue()`
`Character`	`Character.valueOf(paramString)`
`double`	`Double.valueOf(paramString).doubleValue()`
`Double`	`Double.valueOf(paramString)`
`int`	`Integer.valueOf(paramString).intValue()`
`Integer`	`Integer.valueOf(paramString)`
`float`	`Float.valueOf(paramString).floatValue()`
`Float`	`Float.valueOf(paramString)`
`long`	`Long.valueOf(paramString).longValue()`
`Long`	`Long.valueOf(paramString)`

Associating All Properties with Input Parameters

Associating a property with an input parameter saves you the bother of performing conversions for many of the simple built-in types. JSP lets you take the process one step further by associating *all* properties with identically named input parameters. All you have to do is to supply `"*"` for the `property` parameter. So, for example, all three of the `jsp:setProperty` statements of Listing 13.5 can be replaced by the following simple line. Listing 13.6 shows the complete page.

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

Although this approach is simple, four small warnings are in order. First, as with individually associated properties, no action is taken when an input parameter is missing. In particular, the system does not supply `null` as the property value. Second, the JSWDK and the Java Web Server both fail for conversions to properties that expect `double` values. Third, automatic type conversion does not guard against illegal values as effectively as does manual type conversion. So you might consider error pages (see Sections 11.9 and 11.10) when using automatic type conversion. Fourth, since both property names and input parameters are case sensitive, the property name and input parameter must match exactly.

Core Warning

In order for all properties to be associated with input parameters, the property names must match the parameter names ***exactly,*** *including case.*

Listing 13.6 `SaleEntry3.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using jsp:setProperty</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
```

Listing 13.6 SaleEntry3.jsp (continued)

```
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Using jsp:setProperty</TABLE>

<jsp:useBean id="entry" class="coreservlets.SaleEntry" />
<%-- WARNING! Both the JSWDK 1.0.1 and the Java Web Server
              have a bug that makes them fail on automatic
              type conversions to double values.
--%>
<jsp:setProperty name="entry" property="*" />

<BR>
<TABLE ALIGN="CENTER" BORDER=1>
<TR CLASS="COLORED">
  <TH>Item ID<TH>Unit Price<TH>Number Ordered<TH>Total Price
<TR ALIGN="RIGHT">
  <TD><jsp:getProperty name="entry" property="itemID" />
  <TD>$<jsp:getProperty name="entry" property="itemCost" />
  <TD><jsp:getProperty name="entry" property="numItems" />
  <TD>$<jsp:getProperty name="entry" property="totalCost" />
</TABLE>

</BODY>
</HTML>
```

13.4 Sharing Beans

Up to this point, I have treated the objects that were created with `jsp:useBean` as though they were simply bound to local variables in the `_jspService` method (which is called by the `service` method of the servlet that is generated from the page). Although the beans are indeed bound to local variables, that is not the only behavior. They are also stored in one of four different locations, depending on the value of the optional `scope` attribute of `jsp:useBean`. The `scope` attribute has the following possible values:

- **`page`**
  This is the default value. It indicates that, in addition to being bound to a local variable, the bean object should be placed in the `PageContext` object for the duration of the current request. In principle, storing the object there means that servlet code can access it by calling `getAttribute` on the predefined `pageContext` variable. In practice, beans created with `page` scope are almost always accessed by `jsp:getProperty`, `jsp:setProperty`, scriptlets, or expressions later in the same page.
- **`application`**
  This very useful value means that, in addition to being bound to a local variable, the bean will be stored in the shared `ServletContext` available through the predefined `application` variable or by a call to `getServletContext()`. The `ServletContext` is shared by all servlets in the same Web application (or all servlets in the same server or servlet engine if no explicit Web applications are defined). Values in the `ServletContext` can be retrieved by the `getAttribute` method. This sharing has a couple of ramifications.

  First, it provides a simple mechanism for multiple servlets and JSP pages to access the same object. See the following subsection (Conditional Bean Creation) for details and an example.

  Second, it lets a servlet *create* a bean that will be used in JSP pages, not just *access* one that was previously created. This approach lets a servlet handle complex user requests by setting up beans, storing them in the `ServletContext`, then forwarding the request to one of several possible JSP pages to present results appropriate to the request data. For details on this approach, see Chapter 15 (Integrating Servlets and JSP).
- **`session`**
  This value means that, in addition to being bound to a local variable, the bean will be stored in the `HttpSession` object associated with the current request, where it can be retrieved with `getValue`. Attempting to use `scope="session"` causes an

error at page translation time when the `page` directive stipulates that the current page is not participating in sessions. (See Section 11.4, "The session Attribute.")

- **`request`**
  This value signifies that, in addition to being bound to a local variable, the bean object should be placed in the `ServletRequest` object for the duration of the current request, where it is available by means of the `getAttribute` method. This value is only a slight variation of the per-request scope provided by `scope="page"` (or by default when no `scope` is specified).

Conditional Bean Creation

To make bean sharing more convenient, there are two situations where bean-related elements are evaluated conditionally.

First, a `jsp:useBean` element results in a new bean being instantiated only if no bean with the same `id` and `scope` can be found. If a bean with the same `id` and `scope` *is* found, the preexisting bean is simply bound to the variable referenced by `id`. A typecast is performed if the preexisting bean is of a more specific type than the bean being declared, and a `ClassCastException` results if this typecast is illegal.

Second, instead of

```
<jsp:useBean ... />
```

you can use

```
<jsp:useBean ...>
  statements
</jsp:useBean>
```

The point of using the second form is that the statements between the `jsp:useBean` start and end tags are executed *only* if a new bean is created, *not* if an existing bean is used. This conditional execution is convenient for setting initial bean properties for beans that are shared by multiple pages. Since you don't know which page will be accessed first, you don't know which page should contain the initialization code. No problem: they can all contain the code, but only the page first accessed actually executes it. For example, Listing 13.7 shows a simple bean that can be used to record cumulative access counts to any of a set of related pages. It also stores the name of the first page that was accessed. Since there is no way to predict which page in a set will be accessed first, each page that uses the shared counter has statements like the following:

```
<jsp:useBean id="counter"
             class="coreservlets.AccessCountBean"
             scope="application">
  <jsp:setProperty name="counter"
                   property="firstPage"
                   value="Current Page Name" />
</jsp:useBean>

Collectively, the pages using the counter have been accessed
<jsp:getProperty name="counter" property="accessCount" />
times.
```

Listing 13.8 shows the first of three pages that use this approach. The source code archive at `http://www.coreservlets.com/` contains the other two nearly identical pages. Figure 13–3 shows a typical result.

Listing 13.7 `AccessCountBean.java`

```
package coreservlets;

/** Simple bean to illustrate sharing beans through
 *  use of the scope attribute of jsp:useBean.
 */

public class AccessCountBean {
  private String firstPage;
  private int accessCount = 1;

  public String getFirstPage() {
    return(firstPage);
  }

  public void setFirstPage(String firstPage) {
    this.firstPage = firstPage;
  }

  public int getAccessCount() {
    return(accessCount++);
  }
}
```

Listing 13.8 SharedCounts1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Shared Access Counts: Page 1</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Shared Access Counts: Page 1</TABLE>
<P>
<jsp:useBean id="counter"
             class="coreservlets.AccessCountBean"
             scope="application">
  <jsp:setProperty name="counter"
                   property="firstPage"
                   value="SharedCounts1.jsp" />
</jsp:useBean>

Of SharedCounts1.jsp (this page),
<A HREF="SharedCounts2.jsp">SharedCounts2.jsp</A>, and
<A HREF="SharedCounts3.jsp">SharedCounts3.jsp</A>,
<jsp:getProperty name="counter" property="firstPage" />
was the first page accessed.
<P>
Collectively, the three pages have been accessed
<jsp:getProperty name="counter" property="accessCount" />
times.

</BODY>
</HTML>
```

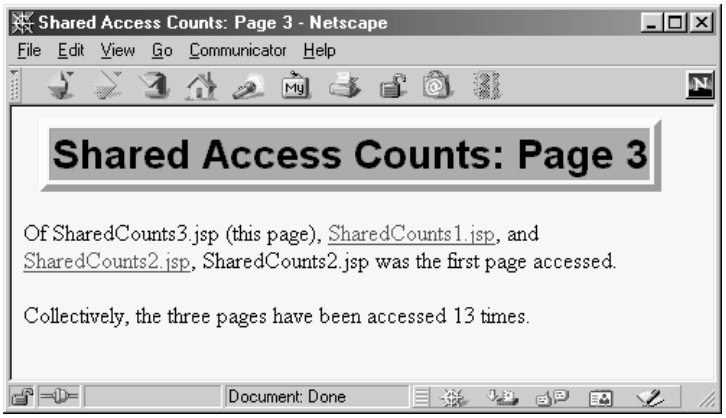


Figure 13–3 Result of a user visiting `SharedCounts3.jsp`. The first page visited by any user was `SharedCounts2.jsp`. `SharedCounts1.jsp`, `SharedCounts2.jsp`, and `SharedCounts3.jsp` were collectively visited a total of twelve times after the server was last started but prior to the visit shown in this figure.