Using JavaBeans Components in JSP Documents

Topics in This Chapter

- Understanding the benefits of beans
- Creating beans
- Installing bean classes on your server
- Accessing bean properties
- Explicitly setting bean properties
- Automatically setting bean properties from request parameters
- Sharing beans among multiple servlets and JSP pages

Chapter 14

This chapter discusses the third general strategy for inserting dynamic content in JSP pages (see Figure 14–1): by means of JavaBeans components.

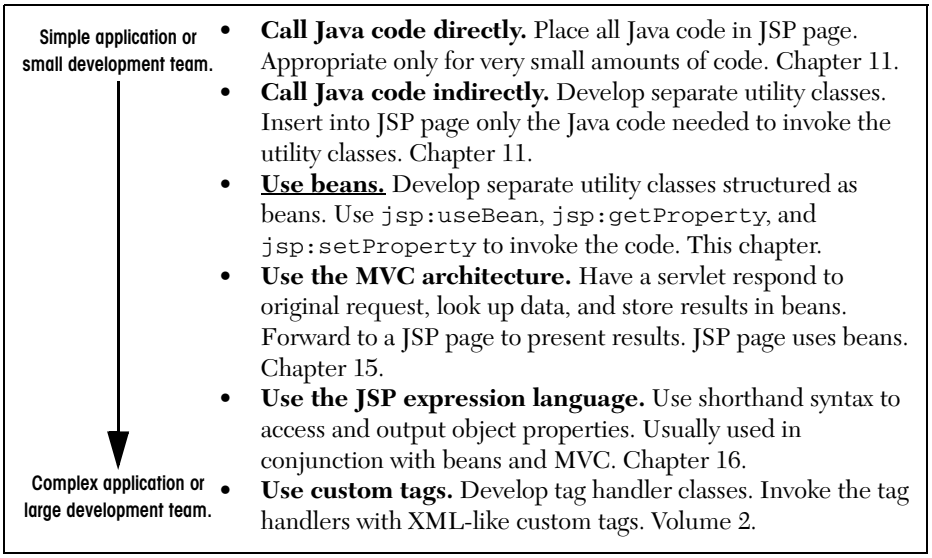

Figure 14–1 Strategies for invoking dynamic code from JSP.

14.1 Why Use Beans?

OK, so you already understand the benefit of using separate Java classes instead of embedding large amounts of code directly in JSP pages. As discussed in Section 11.3 (Limiting the Amount of Java Code in JSP Pages), separate classes are easier to write, compile, test, debug, and reuse. But what do beans provide that other classes do not? After all, beans are merely regular Java classes that follow some simple conventions defined by the JavaBeans specification; beans extend no particular class, are in no particular package, and use no particular interface.

Although it is true that beans are merely Java classes that are written in a standard format, there are several advantages to their use. With beans in general, visual manipulation tools and other programs can automatically discover information about classes that follow this format and can create and manipulate the classes without the user having to explicitly write any code. In JSP in particular, use of JavaBeans components provides three advantages over scriptlets and JSP expressions that refer to normal Java classes.

1. **No Java syntax.** By using beans, page authors can manipulate Java objects using only XML-compatible syntax: no parentheses, semicolons, or curly braces. This promotes a stronger separation between the content and the presentation and is especially useful in large development teams that have separate Web and Java developers.
2. **Simpler object sharing.** When you use the JSP bean constructs, you can much more easily share objects among multiple pages or between requests than if you use the equivalent explicit Java code.
3. **Convenient correspondence between request parameters and object properties.** The JSP bean constructs greatly simplify the process of reading request parameters, converting from strings, and putting the results inside objects.

14.2 What Are Beans?

As we said, beans are simply Java classes that are written in a standard format. Full coverage of JavaBeans is beyond the scope of this book, but for the purposes of use in JSP, all you need to know about beans are the three simple points outlined in the following list. If you want more details on beans in general, pick up one of the many books on the subject or see the documentation and tutorials at http://java.sun.com/products/javabeans/docs/.

- **A bean class must have a zero-argument (default) constructor.** You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors, which results in a zero-argument constructor being created automatically. The default constructor will be called when JSP elements create beans. In fact, as we see in Chapter 15 (Integrating Servlets and JSP: The Model View Controller (MVC) Architecture), it is quite common for a servlet to create a bean, from which a JSP page merely looks up data. In that case, the requirement that the bean have a zero-argument constructor is waived.
- **A bean class should have no public instance variables (fields).** To be a bean that is accessible from JSP, a class should use accessor methods instead of allowing direct access to the instance variables. We hope you already follow this practice since it is an important design strategy in object-oriented programming. In general, use of accessor methods lets you do three things without users of your class changing their code: (a) impose constraints on variable values (e.g., have the `setSpeed` method of your `Car` class disallow negative speeds); (b) change your internal data structures (e.g., change from English units to metric units internally, but still have `getSpeedInMPH` and `getSpeedInKPH` methods); (c) perform side effects automatically when values change (e.g., update the user interface when `setPosition` is called).
- **Persistent values should be accessed through methods called `getXxx` and `setXxx`.** 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 `getXxx` method but no corresponding `setXxx`, the class is said to have a read-only property named *xxx*.

  The one exception to this naming convention is with boolean properties: they are permitted to use a method called `isXxx` 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* naming convention.

14.3 Using Beans: Basic Tasks

You use three main constructs to build and manipulate JavaBeans components in JSP pages:

- **`jsp:useBean`.** In the simplest case, this element builds a new bean. It is normally used as follows:

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

  If you supply a `scope` attribute (see Section 14.6, "Sharing Beans"), the `jsp:useBean` element can either build a new bean or access a preexisting one.

- **`jsp:getProperty`.** This element reads and outputs the value of a bean property. Reading a property is a shorthand notation for calling a method of the form `get`*Xxx*. This element is used as follows:

  ```
  <jsp:getProperty name="beanName"
                   property="propertyName" />
  ```

- **`jsp:setProperty`.** This element modifies a bean property (i.e., calls a method of the form `set`*Xxx*). It is normally used as follows:

  ```
  <jsp:setProperty name="beanName"
                   property="propertyName"
                   value="propertyValue" />
  ```

The following subsections give details on these elements.

Building Beans: jsp:useBean

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 the following.

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

This statement usually means "instantiate an object of the class specified by `Class`, and bind it to a variable in `_jspService` with the name specified by `id`." Note, however, that you use the fully qualified class name—the class name with packages included. This requirement holds true regardless of whether you use `<%@ page import... %>` to import packages.

Core Warning

You must use the fully qualified class name for the `class` attribute of `jsp:useBean`.

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(); %>
```

Installing Bean Classes

The bean class definition should be placed in the same directories where servlets can be installed, *not* in the directory that contains the JSP file. Just remember to use packages (see Section 11.3 for details). Thus, the proper location for individual bean classes is WEB-INF/classes/*subdirectoryMatchingPackageName*, as discussed in Sections 2.10 (Deployment Directories for Default Web Application: Summary) and 2.11 (Web Applications: A Preview). JAR files containing bean classes should be placed in the WEB-INF/lib directory.

Core Approach

Place all your beans in packages. Install them in the normal Java code directories: WEB-INF/classes/subdirectoryMatchingPackageName for individual classes and WEB-INF/lib for JAR files.

Using jsp:useBean Options: scope, beanName, and type

Although it is convenient to think of `jsp:useBean` as being equivalent to building an object and binding it to a local variable, `jsp:useBean` has additional options that make it more powerful. As we'll see in Section 14.6, you can specify a `scope` attribute that associates the bean with more than just the current page. If beans can be shared, it is useful to obtain references to existing beans, rather than always building a new object. 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 declaration, as in the following example.

```
<jsp:useBean id="thread1" class="mypackage.MyClass"
                          type="java.lang.Runnable" />
```

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

```
java.lang.Runnable thread1 = new myPackage.MyClass();
```

A `ClassCastException` results if the actual class is not compatible with `type`. Also, you can use `type` without `class` if the bean already exists and you merely want to access an existing object, not create a new object. This is useful when you share beans by using the `scope` attribute as discussed in Section 14.6.

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.

A few character sequences also require special handling in order to appear inside attribute values. To get `'` within an attribute value, use `\'`. Similarly, to get `"`, use `\"`; to get `\`, use `\\`; to get `%>`, use `%\>`; and to get `<%`, use `<\%`.

Accessing Bean Properties: jsp:getProperty

Once you have a bean, you can output 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.

Core Note

With `jsp:useBean`, the bean name is given by the `id` attribute. With `jsp:getProperty` and `jsp:setProperty`, it is given by the `name` attribute.

Instead of using `jsp:getProperty`, you could use a JSP expression and explicitly call a method on the object with the variable name specified by 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 given earlier in this section, 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. If you create objects with `jsp:useBean` instead of an equivalent JSP scriptlet, be syntactically consistent and output bean properties with `jsp:getProperty` instead of the equivalent JSP expression. However, direct access to the variable is useful when you are using loops, conditional statements, and methods not represented as properties.

For you who 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 it has another method called `setFoo` that takes a `T` as an argument and stores it for later access by `getFoo`."

Setting Simple Bean Properties: jsp:setProperty

To modify bean properties, you normally use `jsp:setProperty`. This action has several different forms, but with the simplest form you 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). In Section 14.5 we present some

alternative 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.

14.4 Example: StringBean

Listing 14.1 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` property called `message`.

Listing 14.2 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"); %>
```

Please note that we do not recommend that you really mix the explicit Java syntax and the XML syntax in the same page; this example is just meant to illustrate the equivalent results of the two forms.

Core Approach

Whenever possible, avoid mixing the XML-compatible `jsp:useBean` tags with JSP scripting elements containing explicit Java code.

Figure 14–2 shows the result.

Listing 14.1 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 14.2 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 (from jsp:getProperty):
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>
<LI>Initial value (from JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                     property="message"
                     value="Best string bean: Fortex" />
    Value after setting property with jsp: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 14–2 Result of StringBean.jsp.

14.5 Setting Bean Properties: Advanced Techniques

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 14.3 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 14.4 shows a JSP file that builds an instance of the `SaleEntry` class by means of:

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

Listing 14.5 (Figure 14–3) gives the HTML form that collects the request parameters. The results are shown in Figure 14–4.

Once the bean is instantiated, using a request parameter to set the `itemID` is straightforward, as shown below.

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

Notice that we used a JSP expression for the `value` attribute. Most JSP attribute values have to be fixed strings, but the `value` attribute of `jsp:setProperty` is permitted to be a request time expression. 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. In any case, the point is that it is *possible* to use JSP expressions here, but doing so requires the use of explicit Java code. In some applications, avoiding such explicit code is the main reason for using beans in the first place. Besides, as the next examples will show, the situation becomes much more complicated when the bean property is not of type `String`. The next two subsections will discuss how to solve these problems.

Listing 14.3 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;
  }

  // In real life, replace this with database lookup.
  // See Chapters 17 and 18 for info on accessing databases
  // from servlets and JSP pages.
```

Listing 14.3 SaleEntry.java *(continued)*

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

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

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

Listing 14.4 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>
<CENTER>
<TABLE BORDER=5>
  <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") %>' />
```

Listing 14.4 SaleEntry1.jsp *(continued)*

```
<%
int numItemsOrdered = 1;
try {
  numItemsOrdered =
    Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {}
%>
<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>" />
<%
double discountCode = 1.0;
try {
  String discountString =
    request.getParameter("discountCode");
  discountCode =
    Double.parseDouble(discountString);
} catch(NumberFormatException nfe) {}
%>
<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />
<BR>
<TABLE 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>
</CENTER></BODY></HTML>
```

Listing 14.5 SaleEntry1-Form.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Invoking SaleEntry1.jsp</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">
      Invoking SaleEntry1.jsp</TABLE>
<FORM ACTION="SaleEntry1.jsp">
  Item ID: <INPUT TYPE="TEXT" NAME="itemID"><BR>
  Number of Items: <INPUT TYPE="TEXT" NAME="numItems"><BR>
  Discount Code: <INPUT TYPE="TEXT" NAME="discountCode"><P>
  <INPUT TYPE="SUBMIT" VALUE="Show Price">
</FORM>
</CENTER></BODY></HTML>
```

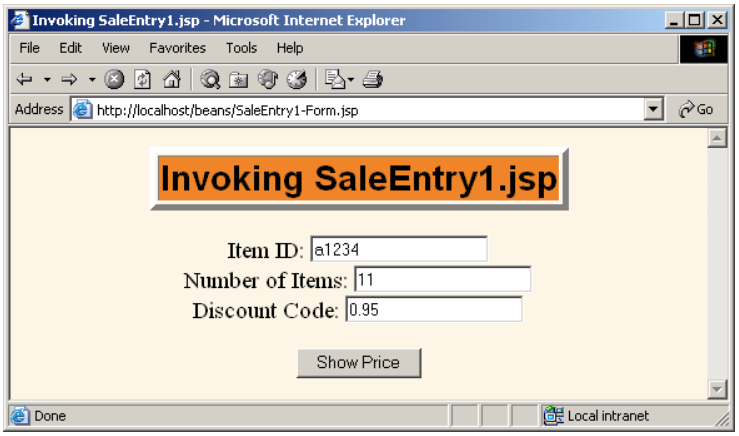

Figure 14–3 Front end to SaleEntry1.jsp.

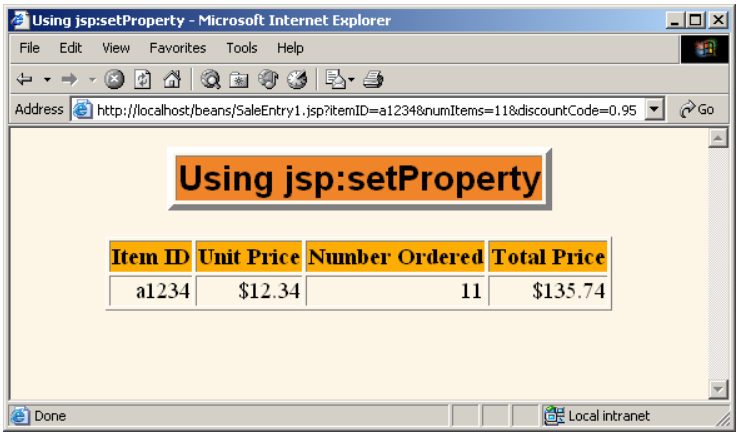

Figure 14–4 Result of SaleEntry1.jsp.

Associating Individual Properties with Input Parameters

Setting the `itemID` property is easy since its value is a `String`. Setting the `numItems` and `discountCode` properties is a bit more problematic since their values must be numbers whereas `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. It lets you associate a property with a request parameter and automatically perform 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 the named request parameter is automatically used as the value of the bean property, and type conversions from `String` to primitive types (`byte`, `int`, `double`, etc.) and wrapper classes (`Byte`, `Integer`, `Double`, etc.) are automatically performed. If the specified 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
    name="entry"
    property="numItems"
    param="numItems" />
```

You can simplify the code slightly if the request parameter name and the bean property name are the same. In that case, you can omit param as in the following example.

```
<jsp:setProperty
    name="entry"
    property="numItems" /> <%-- param="numItems" is assumed. --%>
```

We prefer the slightly longer form that lists the parameter explicitly. Listing 14.6 shows the relevant part of the JSP page reworked in this manner.

Listing 14.6 SaleEntry2.jsp

```
...
<jsp:useBean id="entry" class="coreservlets.SaleEntry" />
<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />
<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />
<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />
...
```

Associating All Properties with Request Parameters

Associating a property with a request 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 request 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 14.6 can be replaced by the following simple line. Listing 14.7 shows the relevant part of the page.

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

Listing 14.7 SaleEntry3.jsp

```
...
<jsp:useBean id="entry" class="coreservlets.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
...
```

This approach lets you define simple "form beans" whose properties correspond to the request parameters and get populated automatically. The system starts with the request parameters and looks for matching bean properties, not the other way around. Thus, no action is taken for bean properties that have no matching request parameter. This behavior means that the form beans need not be populated all at once; instead, one submission can fill in part of the bean, another form can fill in more, and so on. To make use of this capability, however, you need to share the bean among multiple pages. See Section 14.6 (Sharing Beans) for details. Finally, note that servlets can also use form beans, although only by making use of some custom utilities. For details, see Section 4.7 (Automatically Populating Java Objects from Request Parameters: Form Beans).

Although this approach is simple, three small warnings are in order.

- **No action is taken when an input parameter is missing.** In particular, the system does not supply `null` as the property value. So, you usually design your beans to have identifiable default values that let you determine if a property has been modified.
- **Automatic type conversion does not guard against illegal values as effectively as does manual type conversion.** In fact, despite the convenience of automatic type conversion, some developers eschew the automatic conversion, define all of their settable bean properties to be of type `String`, and use explicit `try/catch` blocks to handle malformed data. At the very least, you should consider the use of error pages when using automatic type conversion.
- **Bean property names and request parameters are case sensitive.** So, the property name and request parameter name must match exactly.

14.6 Sharing Beans

Up to this point, we 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`.

When you use `scope`, the system first looks for an existing bean of the specified name in the designated location. Only when the system fails to find a preexisting bean does it create a new one. This behavior lets a servlet handle complex user requests by setting up beans, storing them in one of the three standard shared locations (the request, the session, or the servlet context), 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: The Model View Controller (MVC) Architecture).

As described below, the `scope` attribute has four possible values: `page` (the default), `request`, `session`, and `application`.

- **`<jsp:useBean ... scope="page" />`**
  This is the default value; you get the same behavior if you omit the `scope` attribute entirely. The `page` scope 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. Storing the object there means that servlet code can access it by calling `getAttribute` on the predefined `pageContext` variable.

  Since every page and every request has a different `PageContext` object, using `scope="page"` (or omitting `scope`) indicates that the bean is not shared and thus a new bean will be created for each request.

- **`<jsp:useBean ... scope="request" />`**
  This value signifies that, in addition to being bound to a local variable, the bean object should be placed in the `HttpServletRequest` object for the duration of the current request, where it is available by means of the `getAttribute` method.

  Although at first glance it appears that this scope also results in unshared beans, two JSP pages or a JSP page and a servlet will share request objects when you use `jsp:include` (Section 13.1), `jsp:forward` (Section 13.3), or the `include` or `forward` methods of `RequestDispatcher` (Chapter 15).

Storing values in the request object is common when the MVC (Model 2) architecture is used. For details, see Chapter 15.

- **`<jsp:useBean ... scope="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 `getAttribute`.

  Thus, this scope lets JSP pages easily perform the type of session tracking described in Chapter 9.

- **`<jsp:useBean ... scope="application" />`**
  This value means that, in addition to being bound to a local variable, the bean will be stored in the `ServletContext` available through the predefined `application` variable or by a call to `getServletContext`. The `ServletContext` is shared by all servlets and JSP pages in the Web application. Values in the `ServletContext` can be retrieved with the `getAttribute` method.

Creating Beans Conditionally

To make bean sharing more convenient, you can conditionally evaluate bean-related elements in two situations.

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`.

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. Because `jsp:useBean` invokes the default (zero-argument) constructor, you frequently need to modify the properties after the bean is created. To mimic a constructor, however, you should make these modifications only when the bean is first created, not when an existing (and presumably updated) bean is accessed. No problem: multiple pages can contain `jsp:setProperty` statements between the start and end tags of `jsp:useBean`; only the page first accessed executes the statements.

For example, Listing 14.8 shows a simple bean that defines two properties: `accessCount` and `firstPage`. The `accessCount` property records cumulative access counts to any of a set of related pages and thus should be executed for all requests. The `firstPage` property stores the name of the first page that was accessed and thus should be executed only by the page that is first accessed. To enforce the distinction, we place the `jsp:setProperty` statement that updates the `accessCount` property in unconditional code and place the `jsp:setProperty` statement for `firstPage` between the start and end tags of `jsp:useBean`.

Listing 14.9 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 14–5 shows a typical result.

Listing 14.8 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);
  }

  public void setAccessCountIncrement(int increment) {
    accessCount = accessCount + increment;
  }
}
```

Listing 14.9 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.
<jsp:setProperty name="counter" property="accessCountIncrement"
                 value="1" />
</BODY></HTML>
```

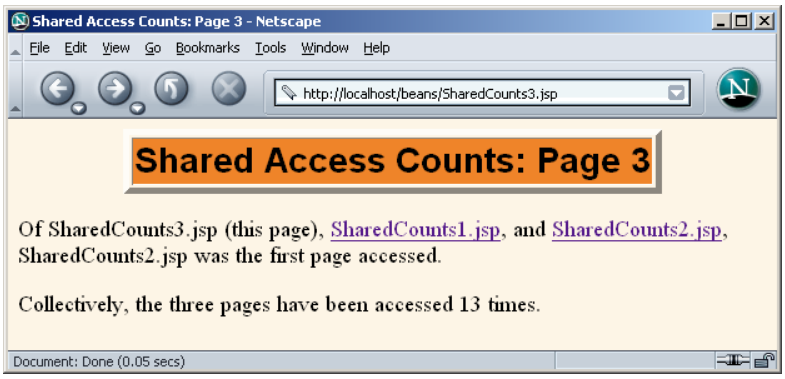

Figure 14–5 Result of a user visiting SharedCounts3.jsp. The first page visited by any user was SharedCounts2.jsp. SharedCounts1.jsp, SharedCounts2.jsp, and SharedCounts3.jsp were collectively visited a total of twelve times after the server was last started but before the visit shown in this figure.

14.7 Sharing Beans in Four Different Ways: An Example

In this section, we give an extended example that illustrates the various aspects of bean use:

- Using beans as utility classes that can be tested separately from JSP pages.
- Using unshared (page-scoped) beans.
- Sharing request-scoped beans.
- Sharing session-scoped beans.
- Sharing application-scoped (i.e., `ServletContext`-scoped) beans.

Before moving on to the examples, one caution is warranted. When you store beans in different scopes, be sure to use different names for each bean. Otherwise, servers can get confused and retrieve the incorrect bean.

Core Warning

Do not use the same bean name for beans stored in different locations. For every bean, use a unique value of `id` in `jsp:useBean`.

Building the Bean and the Bean Tester

The fundamental use of beans is as basic utility (helper) classes. We want to reiterate as strongly as possible: except for very short snippets, Java code that is directly inserted into JSP pages is harder to write, compile, test, debug, and reuse than regular Java classes.

For example, Listing 14.10 presents a small Java object that represents a food item with two properties: `level` and `goesWith` (i.e., four methods: `getLevel`, `setLevel`, `getGoesWith`, and `setGoesWith`). Perhaps this object is so simple that just looking at the source code suffices to show that it is implemented correctly. Perhaps. But how many times have you thought that before, only to uncover a bug later? In any case, a more complex class would surely require testing, so Listing 14.11 presents a test routine. Notice that the utility class represents a value in the application domain and is not dependent on any servlet- or JSP-specific classes. So, it can be tested entirely independently of the server. Listing 14.12 shows some representative output.

Listing 14.10 BakedBean.java

```
package coreservlets;

/** Small bean to illustrate various bean-sharing mechanisms. */

public class BakedBean {
  private String level = "half-baked";
  private String goesWith = "hot dogs";

  public String getLevel() {
    return(level);
  }

  public void setLevel(String newLevel) {
    level = newLevel;
  }

  public String getGoesWith() {
    return(goesWith);
  }

  public void setGoesWith(String dish) {
    goesWith = dish;
  }
}
```

Listing 14.11 BakedBeanTest.java

```
package coreservlets;

/** A small command-line program to test the BakedBean. */

public class BakedBeanTest {
  public static void main(String[] args) {
    BakedBean bean = new BakedBean();
    System.out.println("Original bean: " +
                       "level=" + bean.getLevel() +
                       ", goesWith=" + bean.getGoesWith());
    if (args.length>1) {
      bean.setLevel(args[0]);
      bean.setGoesWith(args[1]);
      System.out.println("Updated bean: " +
                         "level=" + bean.getLevel() +
                         ", goesWith=" + bean.getGoesWith());
    }
  }
}
```

Listing 14.12 Output of BakedBeanTest.java

```
Prompt> java coreservlets.BakedBeanTest gourmet caviar
Original bean: level=half-baked, goesWith=hot dogs
Updated bean: level=gourmet, goesWith=caviar
```

Using scope="page"—No Sharing

OK, after (and *only* after) we are satisfied that the bean works properly, we are ready to use it in a JSP page. The first application is to create, modify, and access the bean entirely within a single page request. For that, we use the following:

- **Create the bean:** use `jsp:useBean` with `scope="page"` (or no `scope` at all, since `page` is the default).
- **Modify the bean:** use `jsp:setProperty` with `property="*"`. Then, supply request parameters that match the bean property names.
- **Access the bean:** use `jsp:getProperty`.

Listing 14.13 presents a JSP page that applies these three techniques. Figures 14–6 and 14–7 illustrate that the bean is available only for the life of the page.

Listing 14.13 BakedBeanDisplay-page.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Baked Bean Values: page-based Sharing</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Baked Bean Values: page-based Sharing</H1>
<jsp:useBean id="pageBean" class="coreservlets.BakedBean" />
<jsp:setProperty name="pageBean" property="*" />
<H2>Bean level:
<jsp:getProperty name="pageBean" property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="pageBean" property="goesWith" /></H2>
</BODY></HTML>
```

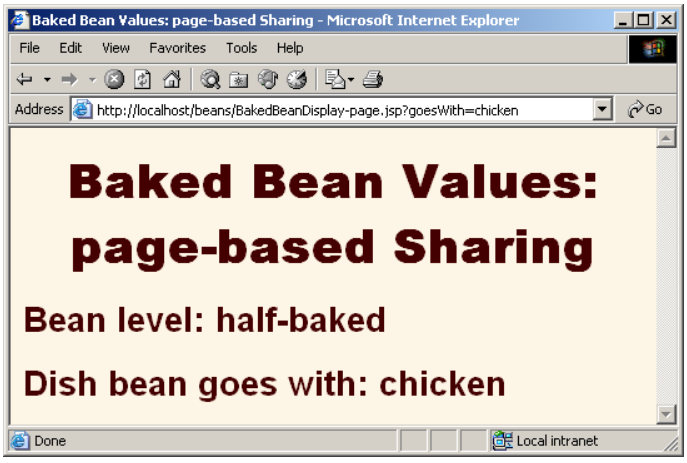

Figure 14–6 Initial request to BakedBeanDisplay-page.jsp—BakedBean properties persist within the page.

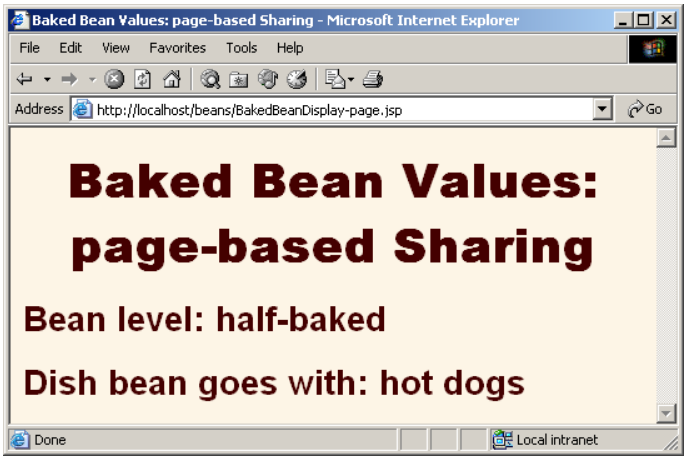

Figure 14–7 Subsequent request to BakedBeanDisplay-page.jsp—`BakedBean` properties do not persist between requests.

Using Request-Based Sharing

The second application is to create, modify, and access the bean within two different pages that share the same request object. Recall that a second page shares the request object of the first page if the second page is invoked with `jsp:include`, `jsp:forward`, or the `include` or `forward` methods of `RequestDispatcher`. To get the desired behavior, we use the following:

- **Create the bean:** use `jsp:useBean` with `scope="request"`.
- **Modify the bean:** use `jsp:setProperty` with `property="*"`. Then, supply request parameters that match the bean property names.
- **Access the bean in the first page:** use `jsp:getProperty`. Then, use `jsp:include` to invoke the second page.
- **Access the bean in the second page:** use `jsp:useBean` with the same `id` as on the first page, again with `scope="request"`. Then, use `jsp:getProperty`.

Listings 14.14 and 14.15 present a pair of JSP pages that applies these four techniques. Figures 14–8 and 14–9 illustrate that the bean is available in the second page but is not stored between requests.

Listing 14.14 BakedBeanDisplay-request.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Baked Bean Values: request-based Sharing</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Baked Bean Values: request-based Sharing</H1>
<jsp:useBean id="requestBean" class="coreservlets.BakedBean"
             scope="request" />
<jsp:setProperty name="requestBean" property="*" />
<H2>Bean level:
<jsp:getProperty name="requestBean" property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="requestBean" property="goesWith" /></H2>
<jsp:include page="BakedBeanDisplay-snippet.jsp" />
</BODY></HTML>
```

Listing 14.15 BakedBeanDisplay-snippet.jsp

```
<H1>Repeated Baked Bean Values: request-based Sharing</H1>
<jsp:useBean id="requestBean" class="coreservlets.BakedBean"
             scope="request" />
<H2>Bean level:
<jsp:getProperty name="requestBean" property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="requestBean" property="goesWith" /></H2>
```

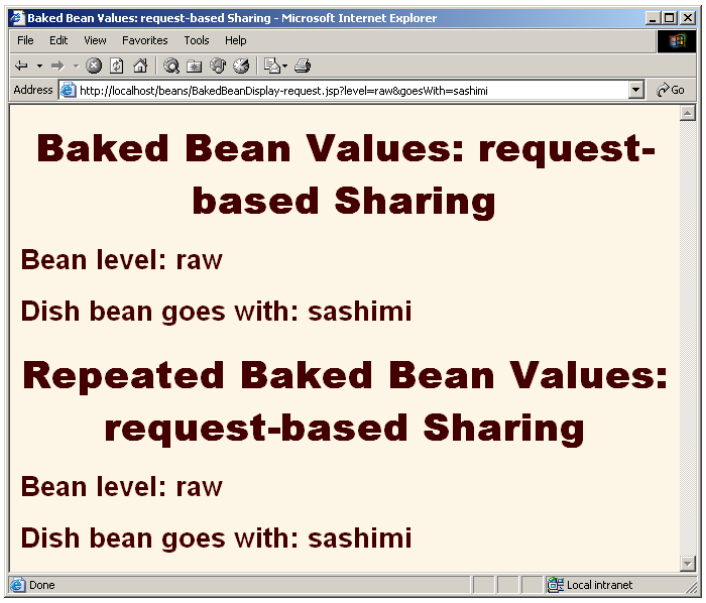

Figure 14–8 Initial request to BakedBeanDisplay-request.jsp—`BakedBean` properties persist to included pages.

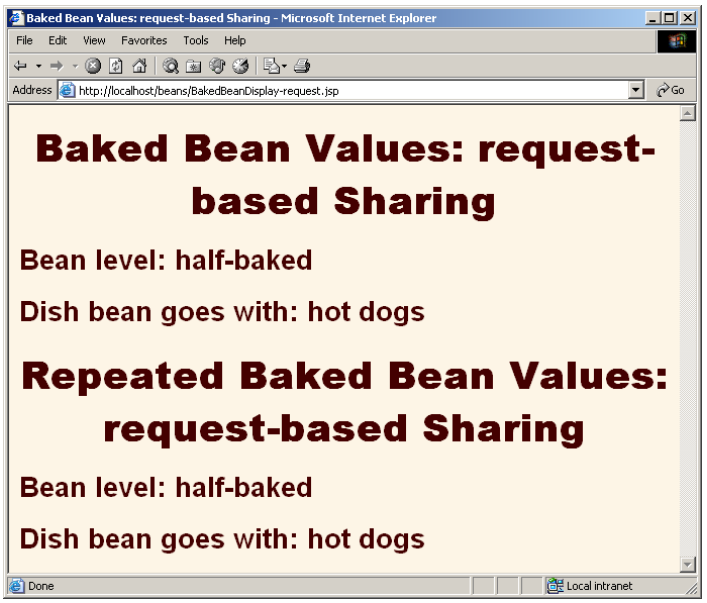

Figure 14–9 Subsequent request to BakedBeanDisplay-request.jsp—`BakedBean` properties do not persist between requests.

Using Session-Based Sharing

The third application involves two parts. First, we want to create, modify, and access the bean within a page. Second, if the *same* client returns to the page, he or she should see the previously modified bean. A classic case of session tracking. So, to get the desired behavior, we use the following:

- **Create the bean:** use `jsp:useBean` with `scope="session"`.
- **Modify the bean:** use `jsp:setProperty` with `property="*"`. Then, supply request parameters that match the bean property names.
- **Access the bean in the initial request:** use `jsp:getProperty` in the request in which `jsp:setProperty` is invoked.
- **Access the bean later:** use `jsp:getProperty` in a request that does not include request parameters and thus does not invoke `jsp:setProperty`. If this request is from the same client (within the session timeout), the previously modified value is seen. If this request is from a different client (or after the session timeout), a newly created bean is seen.

Listing 14.16 presents a JSP page that applies these techniques. Figure 14–10 shows the initial request. Figures 14–11 and 14–12 illustrate that the bean is available in the same session, but not in other sessions. Note that we would have gotten similar behavior if the `jsp:useBean` and `jsp:getProperty` code were repeated in multiple JSP pages: as long as the pages are accessed by the same client, the previous values will be preserved.

Listing 14.16 BakedBeanDisplay-session.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Baked Bean Values: session-based Sharing</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Baked Bean Values: session-based Sharing</H1>
<jsp:useBean id="sessionBean" class="coreservlets.BakedBean"
             scope="session" />
```

Listing 14.16 BakedBeanDisplay-session.jsp *(continued)*

```
<jsp:setProperty name="sessionBean" property="*" />
<H2>Bean level:
<jsp:getProperty name="sessionBean" property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="sessionBean" property="goesWith" /></H2>
</BODY></HTML>
```

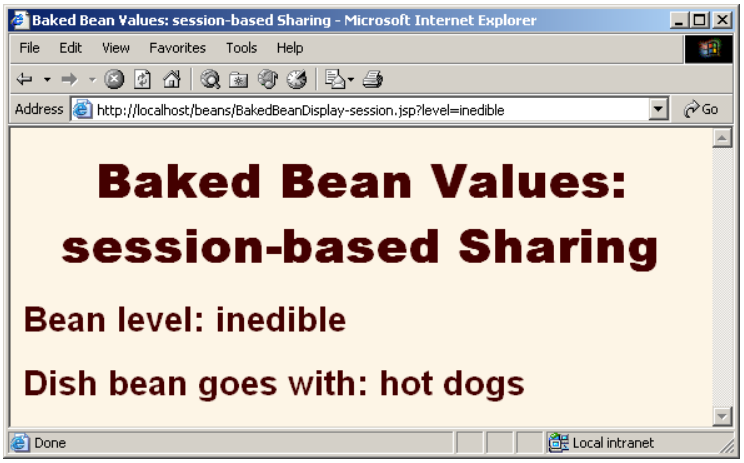

Figure 14–10 Initial request to BakedBeanDisplay-session.jsp.

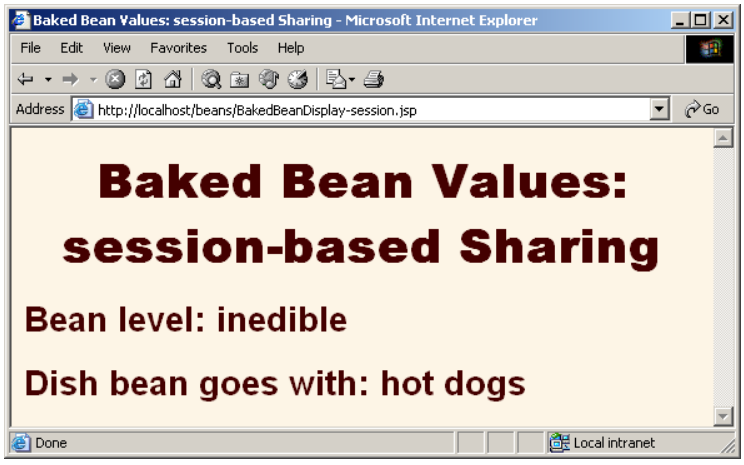

Figure 14–11 Subsequent request to BakedBeanDisplay-session.jsp—`BakedBean` properties persist between requests if the request is from the same client in the same session.

Figure 14–12 Subsequent request to BakedBeanDisplay-session.jsp—`BakedBean` properties do not persist between requests if the request is from a different client (as here) or is in a different session.

Using ServletContext-Based Sharing

The fourth and final application also involves two parts. First, we want to create, modify, and access the bean within a page. Second, if *any* client comes to the page later, he or she should see the previously modified bean. What else besides the `ServletContext` provides such global access? So, to get the desired behavior, we use the following:

- **Create the bean:** use `jsp:useBean` with `scope="application"`.
- **Modify the bean:** use `jsp:setProperty` with `property="*"`. Then, supply request parameters that match the bean property names.
- **Access the bean in the initial request:** use `jsp:getProperty` in the request in which `jsp:setProperty` is invoked.
- **Access the bean later:** use `jsp:getProperty` in a request that does not include request parameters and thus does not invoke `jsp:setProperty`. Whether this request is from the same client or a different client (regardless of the session timeout), the previously modified value is seen.

Listing 14.17 presents a JSP page that applies these techniques. Figure 14–13 shows the initial request. Figures 14–14 and 14–15 illustrate that the bean is available

to multiple clients later. Note that we would have gotten similar behavior if the `jsp:useBean` and `jsp:getProperty` code were repeated in multiple JSP pages.

Listing 14.17 BakedBeanDisplay-application.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Baked Bean Values: application-based Sharing</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Baked Bean Values: application-based Sharing</H1>
<jsp:useBean id="applicationBean" class="coreservlets.BakedBean"
             scope="application" />
<jsp:setProperty name="applicationBean" property="*" />
<H2>Bean level:
<jsp:getProperty name="applicationBean" property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="applicationBean" property="goesWith"/></
H2>
</BODY></HTML>
```

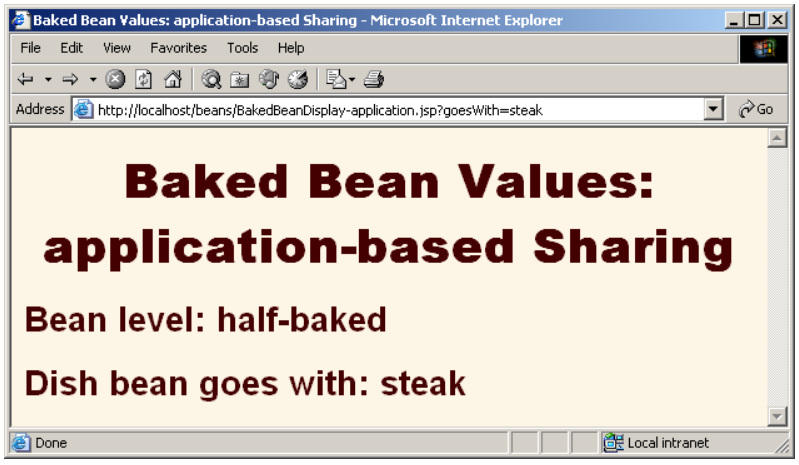

Figure 14–13 Initial request to BakedBeanDisplay-application.jsp.

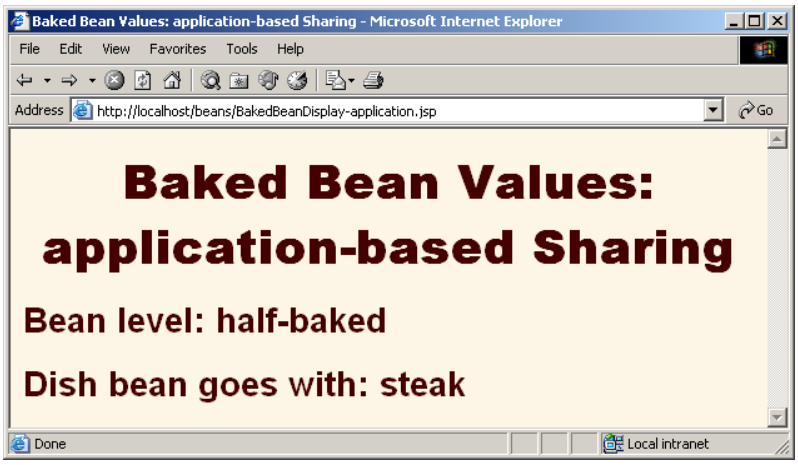

Figure 14–14 Subsequent request to BakedBeanDisplay-application.jsp—`BakedBean` properties persist between requests.

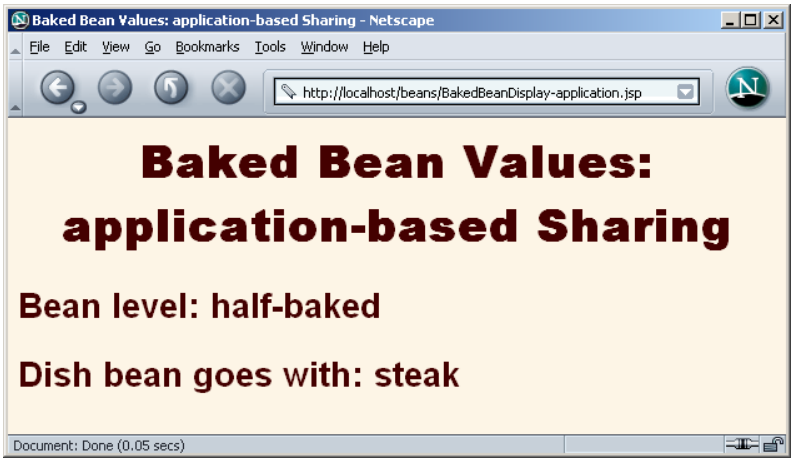

Figure 14–15 Subsequent request to BakedBeanDisplay-application—`BakedBean` properties persist between requests even if the request is from a different client (as here) or is in a different session.