CHAPTER 10: THE APPLICATION EVENTS FRAMEWORK

Topics in This Chapter

- Understanding the general event-handling strategy
- Monitoring servlet context initialization and shutdown
- Setting application-wide values
- Detecting changes in attributes of the servlet context
- Recognizing creation and destruction of HTTP sessions
- Analyzing overall session usage
- Watching for changes in session attributes
- Tracking purchases at an e-commerce site
- Using multiple cooperating listeners
- Packaging listeners in JSP tag libraries

Chapter 10

Developers have many tools at their disposal for handling the life cycle of individual servlets or JSP pages. The servlet `init` method (Section 2.3) fires when a servlet is first instantiated. JSP pages use the nearly identical `jspInit` method (Section 3.3). Both methods can use initialization parameters that are specified with the `init-param` subelement of the *web.xml* `servlet` element (Section 5.5). Requests are handled with `service` and `_jspService`, and destruction is handled with `destroy` and `jspDestroy`.

This is all fine for *individual* resources. But what if you want to respond to major events in the life cycle of the Web application itself? What if you want to create application-wide connection pools, locate resources, or set up shared network connections? For example, suppose you want to record the email address of the support group at your company, an address that will be used by many different servlets and JSP pages. Sure, you can use the following to store the information:

```
context.setAttribute("supportAddress", "balmer@microsoft.com");
```

Better yet, you could use the *web.xml* `context-param` element (Section 5.5) to designate the address, then read it with the `getInitParameter` method of `ServletContext`. Fine. But which servlet or JSP page should perform this task? Or you could read the address from a database. Fine. But which servlet or JSP page should establish the database connection? There is no good answer to this question; you don't know which resources will be accessed first, so the code that performs these tasks would have to be repeated many different places. You want more global control than any one servlet or JSP page can provide. That's where application life-cycle event listeners come in.

There are four kinds of event listeners that respond to Web application life-cycle events.

- **Servlet context listeners.** These listeners are notified when the servlet context (i.e., the Web application) is initialized and destroyed.
- **Servlet context attribute listeners.** These listeners are notified when attributes are added to, removed from, or replaced in the servlet context.
- **Session listeners.** These listeners are notified when session objects are created, invalidated, or timed out.
- **Session attribute listeners.** These listeners are notified when attributes are added to, removed from, or replaced in any session.

Using these listeners involves six basic steps. I'll give a general outline here, then provide listener-specific details in the following sections.

1. **Implement the appropriate interface.** Use `ServletContextListener`, `ServletContextAttributeListener`, `HttpSessionListener`, or `HttpSessionAttributeListener`. The first two interfaces are in the `javax.servlet` package; the second two are in `javax.servlet.http`.
2. **Override the methods needed to respond to the events of interest.** Provide empty bodies for the other methods in the interface. For example, the `ServletContextListener` interface defines two methods: `contextInitialized` (the Web application was just loaded and the servlet context was initialized) and `contextDestroyed` (the Web application is being shut down and the servlet context is about to be destroyed). If you wanted to define an application-wide servlet context entry, you could provide a real implementation for `contextInitialized` and an empty body for `contextDestroyed`.
3. **Obtain access to the important Web application objects.** There are six important objects that you are likely to use in your event-handling methods: the servlet context, the name of the servlet context attribute that changed, the value of the servlet context attribute that changed, the session object, the name of the session attribute that changed, and the value of the session attribute that changed.
4. **Use these objects.** This process is application specific, but there are some common themes. For example, with the servlet context, you are most likely to read initialization parameters (`getInitParameter`), store data for later access (`setAttribute`), and read previously stored data (`getAttribute`).

5. **Declare the listener.** You do this with the `listener` and `listener-class` elements of the general Web application deployment descriptor (*web.xml*) or of a tag library descriptor file.
6. **Provide any needed initialization parameters.** Servlet context listeners commonly read context initialization parameters to use as the basis of data that is made available to all servlets and JSP pages. You use the `context-param` *web.xml* element to provide the names and values of these initialization parameters.

If servlet and JSP filters are the most important new feature in version 2.3 of the servlet specification, then application life-cycle events are the second most important new capability. Remember, however, that these event listeners work only in servers that are compliant with version 2.3 of the servlet specification. If your Web application needs to support older servers, you cannot use life-cycle listeners.

Core Warning

Application life-cycle listeners fail in servers that are compliant only with version 2.2 or earlier versions of the servlet specification.

10.1 Monitoring Creation and Destruction of the Servlet Context

The `ServletContextListener` class responds to the initialization and destruction of the servlet context. These events correspond to the creation and shutdown of the Web application itself. The `ServletContextListener` is most commonly used to set up application-wide resources like database connection pools and to read the initial values of application-wide data that will be used by multiple servlets and JSP pages. Using the listener involves the following six steps.

1. **Implement the `ServletContextListener` interface.** This interface is in the `javax.servlet` package.
2. **Override `contextInitialized` and `contextDestroyed`.** The first of these (`contextInitialized`) is triggered when the Web application is first loaded and the servlet context is created. The two most common tasks performed by this method are creating application-wide data (often by reading context initialization parameters) and storing that data in an easily accessible location (often in attributes of the servlet context). The second method (`contextDestroyed`) is

triggered when the Web application is being shut down and the servlet context is about to be destroyed. The most common task performed by this method is the releasing of resources. For example, `context-Destroyed` can be used to close database connections associated with a now-obsolete connection pool. However, since the servlet context will be destroyed (and garbage collected if the server itself continues to execute), there is no need to use `contextDestroyed` to remove normal objects from servlet context attributes.

3. **Obtain a reference to the servlet context.** The `context-Initialized` and `contextDestroyed` methods each take a `ServletContextEvent` as an argument. The `ServletContext-Event` class has a `getServletContext` method that returns the servlet context.
4. **Use the servlet context.** You read initialization parameters with `getInitParameter`, store data with `setAttribute`, and make log file entries with `log`.
5. **Declare the listener.** Use the `listener` and `listener-class` elements to simply list the fully qualified name of the listener class, as below.

```
<listener>
  <listener-class>somePackage.SomeListener</listener-class>
</listener>
```

   For now, assume that this declaration goes in the *web.xml* file (immediately before the `servlet` element). However, in Section 10.5 you'll see that if you package listeners with tag libraries, you can use the identical declaration within the TLD (tag library descriptor) file of the tag library.
6. **Provide any needed initialization parameters.** Once you have a reference to the servlet context (see Step 3), you can use the `get-InitParameter` method to read context initialization parameters as the basis of data that will be made available to all servlets and JSP pages. You use the `context-param` *web.xml* element to provide the names and values of these initialization parameters, as follows.

```
<context-param>
  <param-name>name</param-name>
  <param-value>value</param-value>
</context-param>
```

10.2 Example: Initializing Commonly Used Data

Suppose that you are developing a Web site for a dot-com company that is a hot commodity. So hot, in fact, that it is constantly being bought out by larger companies. As a result, the company name keeps changing. Rather than changing zillions of separate servlets and JSP pages each time you change the company name, you could read the company name when the Web application is loaded, store the value in the servlet context, and design all your servlets and JSP pages to read the name from this location. To prevent confusion among customers, the site can also prominently display the former company name, initializing and using it in a manner similar to the current company name.

The following steps summarize a listener that accomplishes this task.

1. **Implement the `ServletContextListener` interface.** Listing 10.1 shows a class (`InitialCompanyNameListener`) that implements this interface.
2. **Override `contextInitialized` and `contextDestroyed`.** The `InitialCompanyNameListener` class uses `contextInitialized` to read the current and former company names and store them in the servlet context. Since the `contextDestroyed` method is not needed, an empty body is supplied.
3. **Obtain a reference to the servlet context.** The `contextInitialized` method calls `getServletContext` on the `ServletContextEvent` argument and stores the result in the `context` local variable.
4. **Use the servlet context.** The listener needs to read the `companyName` and `formerCompanyName` initialization parameters and store them in a globally accessible location. So, it calls `getInitParameter` on the context variable, checks for missing values, and uses `setAttribute` to store the result in the servlet context.
5. **Declare the listener.** The listener is declared in the deployment descriptor with the `listener` and `listener-class` elements, as below.

```
<listener>
  <listener-class>
    moreservlets.listeners.InitialCompanyNameListener
  </listener-class>
</listener>
```

The *web.xml* file is shown in Listing 10.2.

6. **Provide any needed initialization parameters.** The `companyName` and `formerCompanyName` init parameters are defined in *web.xml* (Listing 10.2) as follows.

```
<context-param>
  <param-name>companyName</param-name>
  <param-value>not-dot-com.com</param-value>
</context-param>
<context-param>
  <param-name>formerCompanyName</param-name>
  <param-value>hot-dot-com.com</param-value>
</context-param>
```

Listings 10.3 and 10.4 present two JSP pages that use the predefined `application` variable (i.e., the servlet context) to access the `companyName` and `formerCompanyName` attributes. Figures 10–1 and 10–2 show the results. See Section 3.3 for a full list of the predefined JSP variables (`request`, `response`, `application`, etc.).

Listing 10.1 *InitialCompanyNameListener.java*

```
package moreservlets.listeners;

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

/** Listener that looks up the name of the company when
 *  the Web application is first loaded. Stores this
 *  name in the companyName servlet context attribute.
 *  Various servlets and JSP pages will extract it
 *  from that location.
 *  <P>
 *  Also looks up and stores the former company name and
 *  stores it in the formerCompanyName attribute.
 */

public class InitialCompanyNameListener
    implements ServletContextListener {
  private static final String DEFAULT_NAME =
    "MISSING-COMPANY-NAME";
```

Listing 10.1 *InitialCompanyNameListener.java (continued)*

```
/** Looks up the companyName and formerCompanyName
 *  init parameters and puts them into the servlet context.
 */

 public void contextInitialized(ServletContextEvent event) {
    ServletContext context = event.getServletContext();
    setInitialAttribute(context,
                        "companyName",
                        DEFAULT_NAME);
    setInitialAttribute(context,
                        "formerCompanyName",
                        "");
  }

  public void contextDestroyed(ServletContextEvent event) {}

  // Looks for a servlet context init parameter with a given name.
  // If it finds it, it puts the value into a servlet context
  // attribute with the same name. If the init parameter is missing,
  // it puts a default value into the servlet context attribute.

  private void setInitialAttribute(ServletContext context,
                                   String initParamName,
                                   String defaultValue) {
    String initialValue =
      context.getInitParameter(initParamName);
    if (initialValue != null) {
      context.setAttribute(initParamName, initialValue);
    } else {
      context.setAttribute(initParamName, defaultValue);
    }
  }

  /** Static method that returns the servlet context
   *  attribute named "companyName" if it is available.
   *  Returns a default value if the attribute is unavailable.
   */

public static String getCompanyName(ServletContext context) {
    String name =
      (String)context.getAttribute("companyName");
    if (name == null) {
      name = DEFAULT_NAME;
    }
    return(name);
  }
```

Listing 10.1 *InitialCompanyNameListener.java (continued)*

```
  /** Static method that returns the servlet context
    *  attribute named "formerCompanyName" if it is available.
    *  Returns an empty string if the attribute is
    *  unavailable.
    */

  public static String getFormerCompanyName
                                    (ServletContext context) {
    String name =
      (String)context.getAttribute("formerCompanyName");
    if (name == null) {
      name = "";
    }
    return(name);
  }
}
```

Listing 10.2 *web.xml* (Excerpt for initial company name listener)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <!-- Since the company name changes so frequently,
       supply it as a servlet context parameter instead
       of embedding it into lots of different servlets and
       JSP pages. The InitialCompanyNameListener will
       read this value and store it in the servlet context. -->
  <context-param>
    <param-name>companyName</param-name>
    <param-value>not-dot-com.com</param-value>
  </context-param>

  <!-- Also store the previous company name. -->
  <context-param>
    <param-name>formerCompanyName</param-name>
    <param-value>hot-dot-com.com</param-value>
  </context-param>
  <!-- ... -->
```

Listing 10.2 *web.xml* (Excerpt for initial company name listener) *(continued)*

```
  <!-- Register the listener that sets up the
       initial company name. -->
  <listener>
    <listener-class>
      moreservlets.listeners.InitialCompanyNameListener
    </listener-class>
  </listener>
  <!-- ... -->

<!-- If URL gives a directory but no filename, try index.jsp
       first and index.html second. If neither is found,
       the result is server specific (e.g., a directory
       listing).  Order of elements in web.xml matters.
       welcome-file-list needs to come after servlet but
       before error-page.
  -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <!-- ... -->
</web-app>
```

Listing 10.3 *index.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<%@ page import="moreservlets.listeners.*" %>
<%
String companyName =
  InitialCompanyNameListener.getCompanyName(application);
String formerCompanyName =
  InitialCompanyNameListener.getFormerCompanyName(application);
String formerCompanyDescription = "";
if (!formerCompanyName.equals("")) {
  formerCompanyDescription =
    "(formerly " + formerCompanyName + ")";
}
%>
```

Listing 10.3 *index.jsp (continued)*

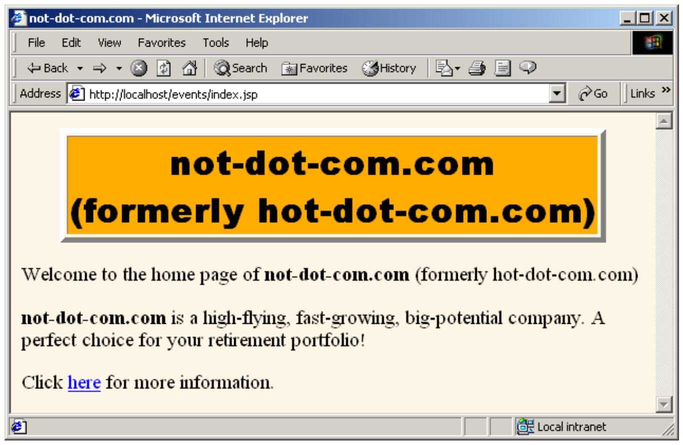
```
<TITLE><%= companyName %></TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      <%= companyName %><BR>
      <%= formerCompanyDescription %>
</TABLE>
<P>
Welcome to the home page of <B><%= companyName %></B>
<%= formerCompanyDescription %>
<P>
<B><%= companyName %></B> is a high-flying, fast-growing,
big-potential company. A perfect choice for your
retirement portfolio!
<P>
Click <A HREF="company-info.jsp">here</A> for more information.
</BODY>
</HTML>
```

Listing 10.4 *company-info.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<%@ page import="moreservlets.listeners.*" %>
<%
String companyName =
  InitialCompanyNameListener.getCompanyName(application);
String formerCompanyName =
  InitialCompanyNameListener.getFormerCompanyName(application);
String formerCompanyDescription = "";
if (!formerCompanyName.equals("")) {
  formerCompanyDescription =
    "(formerly " + formerCompanyName + ")";
}
%>
```

Listing 10.4 *company-info.jsp (continued)*

```
<TITLE><%= companyName %></TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      <%= companyName %><BR>
      <%= formerCompanyDescription %>
</TABLE>
<P>
Learn more about <B><%= companyName %></B>
<%= formerCompanyDescription %>
<UL>
  <LI><A HREF="products.jsp"><%= companyName %> products</A>
  <LI><A HREF="services.jsp"><%= companyName %> services</A>
  <LI><A HREF="history.jsp"><%= companyName %> history</A>
  <LI><A HREF="invest.jsp">investing in <%= companyName %></A>
  <LI><A HREF="contact.jsp">contacting <%= companyName %></A>
</UL>

</BODY>
</HTML>
```


not-dot-com.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/index.jsp
Go Links
Local intranet

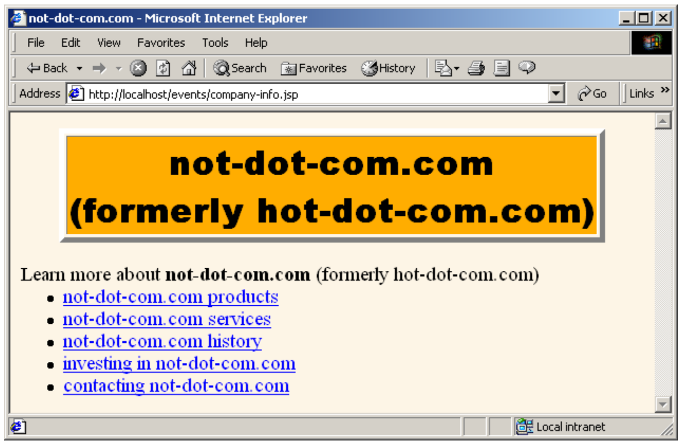
not-dot-com.com
(formerly hot-dot-com.com)

Welcome to the home page of **not-dot-com.com** (formerly hot-dot-com.com)

not-dot-com.com is a high-flying, fast-growing, big-potential company. A perfect choice for your retirement portfolio!

Click here for more information.

Figure 10–1 Home page for the company with the frequently changing name.


not-dot-com.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/company-info.jsp
Go Links
Local intranet

not-dot-com.com
(formerly hot-dot-com.com)

Learn more about **not-dot-com.com** (formerly hot-dot-com.com)

- not-dot-com.com products
- not-dot-com.com services
- not-dot-com.com history
- investing in not-dot-com.com
- contacting not-dot-com.com

Figure 10–2 Informational page for the company with the frequently changing name.

10.3 Detecting Changes in Servlet Context Attributes

OK, when the Web application is loaded, you can set up *initial* values of resources and store references to them in the servlet context. But what if you want to be notified whenever these resources *change*? For example, what if the value of resource B depends on the value of resource A? If resource A changes, you need to automatically update the value of resource B. Handling this situation is the job of servlet context attribute listeners. Using them involves the following steps.

1. **Implement the `ServletContextAttributeListener` interface.** This interface is in the `javax.servlet` package.
2. **Override `attributeAdded`, `attributeReplaced`, and `attributeRemoved`.** The `attributeAdded` method is triggered when a new attribute is added to the servlet context. When a new value is assigned to an existing servlet context attribute, `attribute-Added` is triggered with the new value and `attributeReplaced` is triggered with the old value (i.e., the value being replaced). The `attributeRemoved` method is triggered when a servlet context attribute is removed altogether.
3. **Obtain references to the attribute name, attribute value, and servlet context.** Each of the three `ServletContextAttribute-Listener` methods takes a `ServletContextAttributeEvent` as an argument. The `ServletContextAttributeEvent` class has

three useful methods: `getName` (the name of the attribute that was changed), `getValue` (the value of the changed attribute—the new value for `attributeAdded` and the previous value for `attributeReplaced` and `attributeRemoved`), and `getServletContext` (the servlet context).

4. **Use the objects.** You normally compare the attribute name to a stored name to see if it is the one you are monitoring. The attribute value is used in an application-specific manner. The servlet context is usually used to read previously stored attributes (`getAttribute`), store new or changed attributes (`setAttribute`), and make entries in the log file (`log`).
5. **Declare the listener.** Use the `listener` and `listener-class` elements to simply list the fully qualified name of the listener class, as below.

```
<listener>
  <listener-class>somePackage.SomeListener</listener-class>
</listener>
```

For now, assume that this declaration goes in the *web.xml* file immediately before any `servlet` elements. However, in Section 10.5 you'll see that if you package listeners with tag libraries, you can use the identical declaration within the TLD (tag library descriptor) file of the tag library.

The following section gives an example.

10.4 Example: Monitoring Changes to Commonly Used Data

Section 10.2 shows how to read the current and former company names when the Web application is loaded and how to make use of those values in JSP pages. But what if you want to change the company name during the execution of the Web application? It is reasonable to expect a routine that makes this change to modify the `companyName` servlet context attribute. After all, in this context, that's what it means to change the company name. It is not reasonable, however, to expect that routine to modify (or even know about) the `formerCompanyName` attribute. But, if the company name changes, the former company name must change as well. Enter servlet context attribute listeners!

The following steps summarize a listener that automatically updates the former company name whenever the current company name changes.

1. **Implement the `ServletContextAttributeListener` interface.** Listing 10.5 shows a class (`ChangedCompanyNameListener`) that implements this interface.
2. **Override `attributeAdded`, `attributeReplaced`, and `attributeRemoved`.** The `attributeReplaced` method is used to detect modification to context attributes. Empty bodies are supplied for the `attributeAdded` and `attributeRemoved` methods.
3. **Obtain references to the attribute name, attribute value, and servlet context.** The `attributeReplaced` method calls `getName` and `getValue` on its `ServletContextAttributeEvent` argument to obtain the name and value of the modified attribute. The method also calls `getServletContext` on its argument to get a reference to the servlet context.
4. **Use the objects.** The attribute name is compared to `"companyName"`. If the name matches, the attribute value is used as the new value of the `formerCompanyName` servlet context attribute.
5. **Declare the listener.** The listener is declared in the deployment descriptor with the `listener` and `listener-class` elements, as below.

```
<listener>
  <listener-class>
    moreservlets.listeners.ChangedCompanyNameListener
  </listener-class>
</listener>
```

The *web.xml* file is shown in Listing 10.6.

Listing 10.7 presents a JSP page containing a form that displays the current company name, lets users enter a new name, and submits the new name to the `ChangeCompanyName` servlet (Listing 10.8). Since changing the company name is a privileged operation, access to the form and the servlet should be restricted.

So, the form is placed in the *admin* directory and the `servlet` and `servlet-mapping` elements are used to assign the servlet a URL that also starts with */admin*. See Section 5.3 (Assigning Names and Custom URLs) for details on `servlet` and `servlet-mapping`; see the deployment descriptor in Listing 10.6 for the usage in this example.

Next, the `security-constraint` element is used to stipulate that only authenticated users in the `ceo` role can access the *admin* directory. Then, the `login-config` element is used to specify that form-based authentication be used, with *login.jsp* (Listing 10.9) collecting usernames and passwords and *login-error.jsp* (Listing 10.10) displaying messages to users who failed authentication. Listing 10.11 shows a Tomcat-specific password file used to designate a user who is in the `ceo` role. See Section 7.1 (Form-Based Authentication) for details on these types of security settings; see the deployment descriptor in Listing 10.6 for the usage in this example.

Figures 10–3 through 10–8 show the results of logging in, changing the company name, and revisiting the pages that display the current and former company names.

Listing 10.5 *ChangedCompanyNameListener.java*

```
package moreservlets.listeners;

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

/** Listener that monitors changes in the company
 *  name (which is stored in the companyName attribute
 *  of the servlet context).
 */

public class ChangedCompanyNameListener
    implements ServletContextAttributeListener {

  /** When the companyName attribute changes, put
   *  the previous value into the formerCompanyName
   *  attribute.
   */

  public void attributeReplaced
                 (ServletContextAttributeEvent event) {
    if (event.getName().equals("companyName")) {
      String oldName = (String)event.getValue();
      ServletContext context = event.getServletContext();
      context.setAttribute("formerCompanyName", oldName);
    }
  }

  public void attributeAdded
                 (ServletContextAttributeEvent event) {}

  public void attributeRemoved
                 (ServletContextAttributeEvent event) {}
}
```

Listing 10.6 *web.xml* (Excerpt for changed company name listener)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Register the listener that monitors changes to
       the company name.
  -->
  <listener>
    <listener-class>
      moreservlets.listeners.ChangedCompanyNameListener
    </listener-class>
  </listener>
  <!-- ... -->

  <!-- Assign the name ChangeCompanyName to
       moreservlets.ChangeCompanyName. -->
  <servlet>
    <servlet-name>ChangeCompanyName</servlet-name>
    <servlet-class>moreservlets.ChangeCompanyName</servlet-class>
  </servlet>

  <!-- Give a name to the servlet that redirects users
       to the home page.
  -->
  <servlet>
    <servlet-name>Redirector</servlet-name>
    <servlet-class>moreservlets.RedirectorServlet</servlet-class>
  </servlet>

  <!-- Assign the URL /admin/ChangeCompanyName to the
       servlet that is named ChangeCompanyName.
  -->
  <servlet-mapping>
    <servlet-name>ChangeCompanyName</servlet-name>
    <url-pattern>/admin/ChangeCompanyName</url-pattern>
  </servlet-mapping>
```

Listing 10.6 *web.xml* (Excerpt for changed company name listener) *(continued)*

```
  <!-- Turn off invoker. Send requests to index.jsp. -->
  <servlet-mapping>
    <servlet-name>Redirector</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
  <!-- ... -->

  <!-- Protect everything within the "admin" directory.
       Direct client access to this directory requires
       authentication.
  -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Admin</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>ceo</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Tell the server to use form-based authentication. -->
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/admin/login.jsp</form-login-page>
      <form-error-page>/admin/login-error.jsp</form-error-page>
    </form-login-config>
  </login-config>
</web-app>
```

Listing 10.7 *change-company-name.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<%@ page import="moreservlets.listeners.*" %>
<%
String companyName =
  InitialCompanyNameListener.getCompanyName(application);
%>
<TITLE>Changing Company Name</TITLE>
<LINK REL=STYLESHEET
      HREF="../events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Changing Company Name
</TABLE>
<P>

<FORM ACTION="ChangeCompanyName">
New name:
<INPUT TYPE="TEXT" NAME="newName" VALUE="<%= companyName %>">
<P>
<CENTER><INPUT TYPE="SUBMIT" VALUE="Submit Change"></CENTER>
</FORM>

</BODY>
</HTML>
```

Listing 10.8 *ChangeCompanyName.java*

```
package moreservlets;

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

/** Servlet that changes the company name. The web.xml
 *  file specifies that only authenticated users in the
 *  ceo role can access the servlet. A servlet context
 *  attribute listener updates the former company name
 *  when this servlet (or any other program) changes
 *  the current company name.
 */

public class ChangeCompanyName extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    boolean isNameChanged = false;
    String newName = request.getParameter("newName");
    if ((newName != null) && (!newName.equals(""))) {
      isNameChanged = true;
      getServletContext().setAttribute("companyName",
                                       newName);
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    String title = "Company Name";
    out.println
      (docType +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\">\n" +
       "<H2 ALIGN=\"CENTER\">" + title + "</H2>");
    if (isNameChanged) {
      out.println("Company name changed to " + newName + ".");
    } else {
      out.println("Company name not changed.");
    }
    out.println("</BODY></HTML>");
  }
}
```

Listing 10.9 *login.jsp*

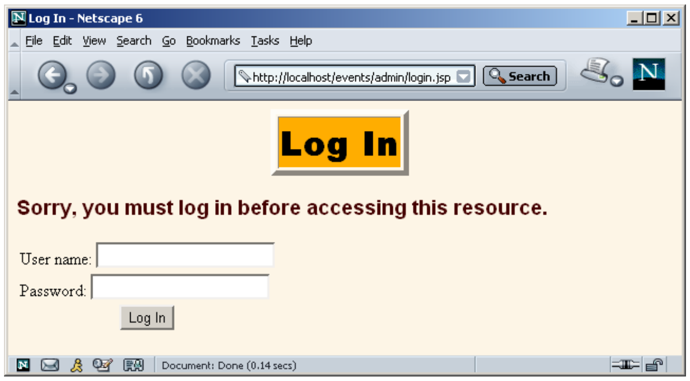
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Log In</TITLE>
<LINK REL=STYLESHEET
      HREF="../events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Log In</TABLE>
<P>
<H3>Sorry, you must log in before accessing this resource.</H3>
<FORM ACTION="j_security_check" METHOD="POST">
<TABLE>
<TR><TD>User name: <INPUT TYPE="TEXT" NAME="j_username">
<TR><TD>Password: <INPUT TYPE="PASSWORD" NAME="j_password">
<TR><TH><INPUT TYPE="SUBMIT" VALUE="Log In">
</TABLE>
</FORM>

</BODY>
</HTML>
```

Listing 10.10 *login-error.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Begone!</TITLE>
<LINK REL=STYLESHEET
      HREF="../events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Begone!</TABLE>

<H3>Begone, ye unauthorized peon.</H3>

</BODY>
</HTML>
```

Listing 10.11 *tomcat-users.xml* (Excerpt for events examples)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tomcat-users>
  <!-- ... -->
  <user name="gerstner" password="lou"
        roles="ceo" />
</tomcat-users>
```

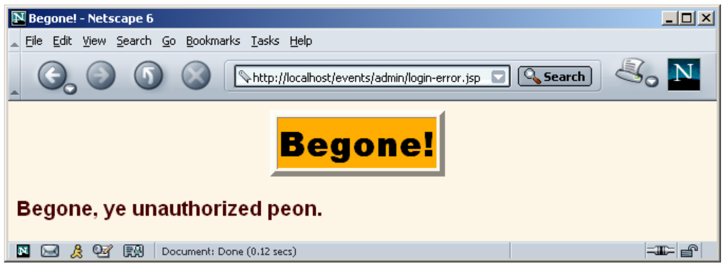

Log In - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/login.jsp
Search
Log In

Sorry, you must log in before accessing this resource.

User name:
Password:
Log In

Document: Done (0.14 secs)

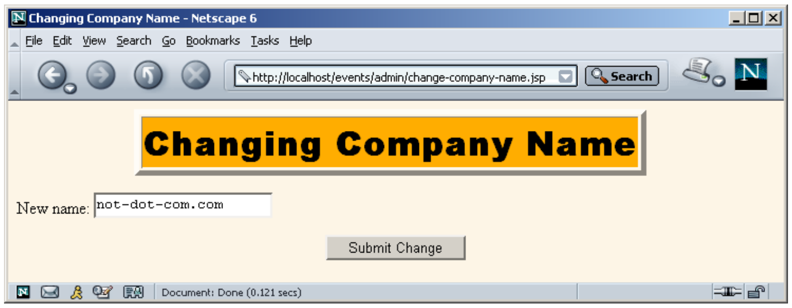
Figure 10–3 Only users who are in the `ceo` role can access the form that changes the company name.


Begone! - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/login-error.jsp
Search
Begone!

Begone, ye unauthorized peon.

Document: Done (0.12 secs)

Figure 10–4 A failed login attempt.

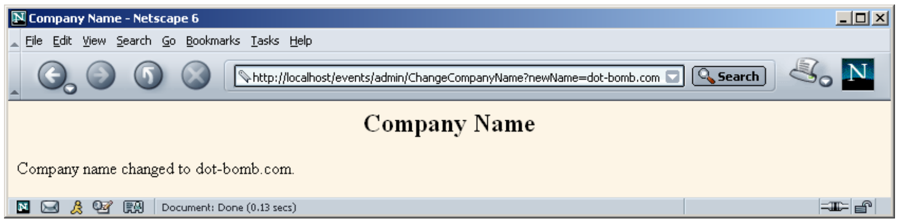

Changing Company Name - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/change-company-name.jsp
Search
Document: Done (0.121 secs)

Changing Company Name

New name: not-dot-com.com

Submit Change

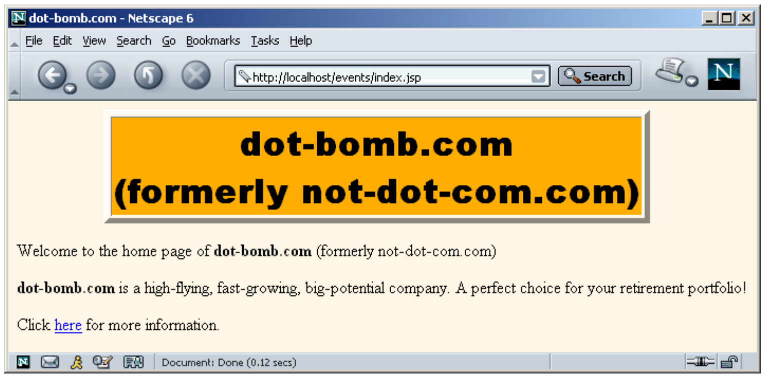
Figure 10–5 The form to change the company name when the page is accessed by an authenticated user who is in the `ceo` role.


Company Name - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/ChangeCompanyName?newName=dot-bomb.com
Search
Document: Done (0.13 secs)

Company Name

Company name changed to dot-bomb.com.

Figure 10–6 The name change confirmation page.


dot-bomb.com - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/index.jsp
Search
Document: Done (0.12 secs)

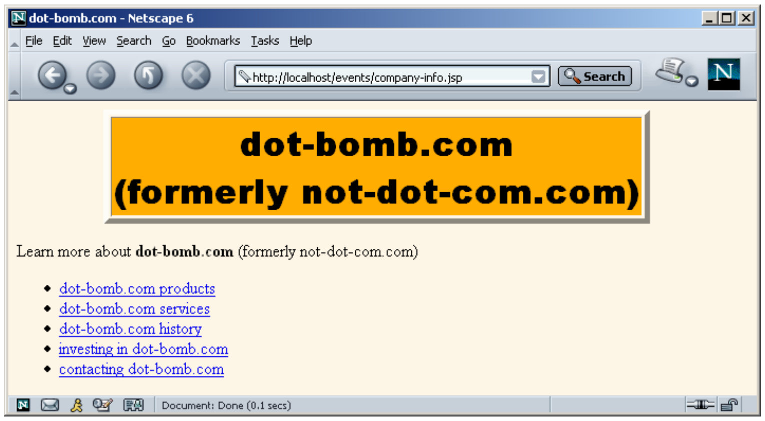
dot-bomb.com (formerly not-dot-com.com)

Welcome to the home page of **dot-bomb.com** (formerly not-dot-com.com)

dot-bomb.com is a high-flying, fast-growing, big-potential company. A perfect choice for your retirement portfolio!

Click here for more information.

Figure 10–7 When the company name changes, the company home page (Listing 10.3) is automatically updated.


dot-bomb.com - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/company-info.jsp
Search
Document: Done (0.1 secs)

dot-bomb.com
(formerly not-dot-com.com)

Learn more about **dot-bomb.com** (formerly not-dot-com.com)

- dot-bomb.com products
- dot-bomb.com services
- dot-bomb.com history
- investing in dot-bomb.com
- contacting dot-bomb.com

Figure 10–8 When the company name changes, the company information page (Listing 10.4) is also updated automatically.

10.5 Packaging Listeners with Tag Libraries

JSP tag libraries (Section 3.7, Chapter 11) provide a great way of encapsulating content that will be accessed by multiple JSP pages. But what if that content depends on life-cycle event listeners? If the `listener` and `listener-class` elements of *web.xml* were the only option for declaring listeners, tag library maintenance would be much more difficult. Normally, the user of a tag library can deploy it by simply dropping a JAR file in *WEB-INF/lib* and putting a TLD file in *WEB-INF*. Users of the tag libraries need no knowledge of the individual classes within the library, only of the tags that the library defines. But if the tag libraries used listeners, users of the libraries would need to discover the name of the listener classes and make *web.xml* entries for each one. This would be significantly more work.

Fortunately, the JSP 1.2 specification lets you put the listener declarations in the tag library descriptor file instead of in the deployment descriptor. But, wait! Event listeners need to run when the Web application is first loaded, not just the first time a JSP page that uses a custom library is accessed. How does the system handle this? The answer is that, when the Web application is loaded, the system automatically searches *WEB-INF* and its subdirectories for files with *.tld* extensions and uses all listener declarations that it finds. This means that your TLD files *must* be in the *WEB-INF* directory or a subdirectory thereof. In fact, although few servers enforce the restriction, the JSP 1.2 specification requires all TLD files to be in *WEB-INF* anyhow. Besides, putting the TLD files in *WEB-INF* is a good strategy to prevent users from retrieving them. So, you should make *WEB-INF* the standard TLD file location, regardless of whether your libraries use event handlers.

Core Approach

Always put your TLD files in the WEB-INF *directory or a subdirectory thereof.*

Unfortunately, there is a problem with this approach: Tomcat 4.0 improperly ignores TLD files at Web application startup time unless there is also a `taglib` entry in *web.xml* of the following form:

```
<taglib>
  <taglib-uri>/someName.tld</taglib-uri>
  <taglib-location>/WEB-INF/realName.tld</taglib-location>
</taglib>
```

As discussed in Section 5.13 (Locating Tag Library Descriptors), this entry is a good idea when the name of your tag library changes frequently. However, the JSP 1.2 specification does not require its use, and servers such as ServletExec 4.1 properly handle listener declarations in TLD files when there is no such entry. Nevertheless, Tomcat 4.0 requires it.

Core Warning

Tomcat 4.0 only reads listener declarations from TLD files that have `taglib` *entries in* web.xml.

Since listener declarations are a new capability in version 1.2 of the JSP specification, you must use the JSP 1.2 format of the tag library descriptor file. This format differs in two ways from the JSP 1.1 format.

First, the `DOCTYPE` declaration must use the JSP 1.2 Document Type Definition (DTD), not the 1.1 one. Here is the JSP 1.2 version:

```
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
```

Here is the JSP 1.1 version:

```
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
```

Second, a number of elements within the TLD file have changed their names slightly. In particular, hyphens were added to the `tlibversion`, `jspversion`, `shortname`, `tagclass`, and `bodycontent` elements. Also, the `info` element was renamed to `description`. These changes are summarized in Table 10.1.

Table 10.1 Changes in tag library element names. You must use the new element names if you use the 1.2 DTD (which is required if you use new capabilities such as listeners).

JSP 1.2 Name	*JSP 1.1 Name*
`tlib-version`	`tlibversion`
`jsp-version`	`jspversion`
`short-name`	`shortname`
`description`	`info`
`tag-class`	`tagclass`
`body-content`	`bodycontent`

Given the changes to the `DOCTYPE` declaration and the element names, Listing 10.12 shows the template for a TLD file in JSP 1.2. For comparison, Listing 10.13 shows the JSP 1.1 template.

Listing 10.12 JSP 1.2 Tag Library Descriptor (Template)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>some-name</short-name>
  <description>
    Tag library documentation.
  </description>
```

Listing 10.12 JSP 1.2 Tag Library Descriptor (Template) *(continued)*

```
<taglib>
  <!-- listener elements, if any, each of the following form:
    <listener>
      <listener-class>somePackage.SomeListener</listener-class>
    </listener>
  -->

  <tag>
    <name>tagName</name>
    <tag-class>somePackage.SomeTag</tag-class>
    <body-content>...</body-content>
    <description>Tag documentation.</description>
  </tag>

  <!-- Other tag elements, if any. -->
</taglib>
```

Listing 10.13 JSP 1.1 Tag Library Descriptor (Template)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>some-name</shortname>
  <info>
    Tag library documentation.
  </info>

  <taglib>
    <tag>
      <name>tagName</name>
      <tagclass>somePackage.SomeTag</tagclass>
      <bodycontent>...</bodycontent>
      <info>Tag documentation.</info>
    </tag>

    <!-- Other tag elements, if any. -->
  </taglib>
```

10.6 Example: Packaging the Company Name Listeners

The listeners shown in Sections 10.2 and 10.4 are very effective in keeping track of the current and former company names. However, the pages that display the names (*index.jsp*, Listing 10.3 and *company-info.jsp*, Listing 10.4) are a bit difficult to read and maintain. This difficulty is due to the need for checking if the former company name is missing before trying to display it, a test that results in quite a bit of explicit Java code in the JSP page. A perfect job for a simple custom tag!

Listings 10.14 and 10.15 show custom tags that print out the current and former company names, respectively. The first tag simply prints the current company name. The second tag uses a `fullDescription` attribute to decide whether to simply print the former company name (e.g., `some-company.com`) or the company name inside parentheses (e.g., `(formerly some-company.com)`). Listing 10.16 shows the TLD file for this library: the `listener` elements of Sections 10.2 and 10.4 are moved out of the *web.xml* file and into the TLD file, which is then placed in the *WEB-INF* directory. Listing 10.17 shows the *web.xml* file: the previous `listener` elements are removed, and a `taglib` entry is added that makes it easier to update the name of the TLD file and lets the listeners be detected at Web application startup time by Tomcat 4.0.

Finally, Listings 10.18 and 10.19 show the company home page (see Listing 10.3) and company information page (see Listing 10.4) reworked with the new custom tags. Note that for the `uri` attribute of the `taglib` directive, these pages use `"/company-name-taglib.tld"` (the alias defined with the *web.xml* `taglib` element), not `"/WEB-INF/company-name-taglib.tld"` (the real location). Figures 10–9 and 10–10 show the results—identical to those shown earlier in Figures 10–1 and 10–2.

Listing 10.14 *CompanyNameTag.java*

```
package moreservlets.tags;

import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
import moreservlets.listeners.*;
```

Listing 10.14 *CompanyNameTag.java (continued)*

```
/** The InitialCompanyNameListener class has static
 *  methods that permit access to the current and former
 *  company names. But, using these methods in JSP requires
 *  explicit Java code, and creating beans that provided
 *  the information would have yielded a cumbersome result.
 *  So, we simply move the code into a custom tag.
 */

public class CompanyNameTag extends TagSupport {
  public int doStartTag() {
    try {
      ServletContext context = pageContext.getServletContext();
      String companyName =
        InitialCompanyNameListener.getCompanyName(context);
      JspWriter out = pageContext.getOut();
      out.print(companyName);
    } catch(IOException ioe) {
      System.out.println("Error printing company name.");
    }
    return(SKIP_BODY);
  }
}
```

Listing 10.15 *FormerCompanyNameTag.java*

```
package moreservlets.tags;

import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
import moreservlets.listeners.*;

/** The InitialCompanyNameListener class has static
 *  methods that permit access to the current and former
 *  company names. But, using these methods in JSP requires
 *  explicit Java code, and creating beans that provided
 *  the information would have yielded a cumbersome result.
 *  So, we simply move the code into a custom tag.
 */
```

Listing 10.15 *FormerCompanyNameTag.java (continued)*

```
public class FormerCompanyNameTag extends TagSupport {
  private boolean useFullDescription = false;

  public int doStartTag() {
    try {
      ServletContext context = pageContext.getServletContext();
      String formerCompanyName =
       InitialCompanyNameListener.getFormerCompanyName(context);
      JspWriter out = pageContext.getOut();
      if (useFullDescription) {
        String formerCompanyDescription = "";
        if (!formerCompanyName.equals("")) {
          formerCompanyDescription =
            "(formerly " + formerCompanyName + ")";
        }
        out.print(formerCompanyDescription);
      } else {
        out.print(formerCompanyName);
      }
    } catch(IOException ioe) {
      System.out.println("Error printing former company name.");
    }
    return(SKIP_BODY);
  }

  /** If the user supplies a fullDescription attribute
   *  with the value "true" (upper, lower, or mixed case),
   *  set the useFullDescription instance variable to true.
   *  Otherwise, leave it false.
   */

  public void setFullDescription(String flag) {
    if (flag.equalsIgnoreCase("true")) {
      useFullDescription = true;
    }
  }

  /** Servers are permitted to reuse tag instances
   *  once a request is finished. So, this resets
   *  the useFullDescription field. This method
   *  is automatically called after the system is
   *  finished using the tag.
   */

  public void release() {
    useFullDescription = false;
  }
}
```

Listing 10.16 *company-name-taglib.tld*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<!-- a tag library descriptor -->

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>company-name-tags</short-name>
  <description>
    A tag library to print out the ever-changing current
    and former company names (which are monitored by event
    listeners).
  </description>

  <!-- Register the listener that sets up the
       initial company name. -->
  <listener>
    <listener-class>
      moreservlets.listeners.InitialCompanyNameListener
    </listener-class>
  </listener>

  <!-- Register the listener that monitors changes to
       the company name.
  -->
  <listener>
    <listener-class>
      moreservlets.listeners.ChangedCompanyNameListener
    </listener-class>
  </listener>

  <!-- Define a tag that prints out the current name. -->
  <tag>
    <name>companyName</name>
    <tag-class>moreservlets.tags.CompanyNameTag</tag-class>
    <body-content>empty</body-content>
    <description>The current company name</description>
  </tag>
```

Listing 10.16 *company-name-taglib.tld (continued)*

```
  <!-- Define a tag that prints out the previous name. -->
  <tag>
    <name>formerCompanyName</name>
    <tag-class>moreservlets.tags.FormerCompanyNameTag</tag-class>
    <body-content>empty</body-content>
    <description>The previous company name</description>
    <attribute>
      <name>fullDescription</name>
      <required>false</required>
    </attribute>
  </tag>
</taglib>
```

Listing 10.17 *web.xml* (Excerpt for custom tags)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Removed declarations for initial and changed company
       name listeners. They are now in TLD file. -->
  <!-- ... -->

  <!-- Register the company-name tag library. -->
  <taglib>
    <taglib-uri>
      /company-name-taglib.tld
    </taglib-uri>
    <taglib-location>
      /WEB-INF/company-name-taglib.tld
    </taglib-location>
  </taglib>

  <!-- ... -->
</web-app>
```

Listing 10.18 *index2.jsp*

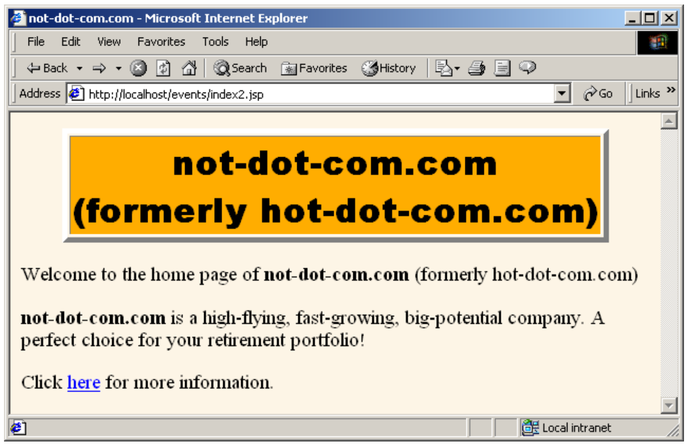
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<%@ taglib uri="/company-name-taglib.tld" prefix="msajsp" %>
<TITLE><msajsp:companyName/></TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      <msajsp:companyName/><BR>
      <msajsp:formerCompanyName fullDescription="true"/>
</TABLE>
<P>
Welcome to the home page of <B><msajsp:companyName/></B>
<msajsp:formerCompanyName fullDescription="true"/>
<P>
<B><msajsp:companyName/></B> is a high-flying, fast-growing,
big-potential company. A perfect choice for your
retirement portfolio!
<P>
Click <A HREF="company-info2.jsp">here</A> for more information.
</BODY>
</HTML>
```

Listing 10.19 *company-info2.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<%@ taglib uri="/company-name-taglib.tld" prefix="msajsp" %>
<TITLE><msajsp:companyName/></TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>
```

Listing 10.19 *company-info2.jsp (continued)*

```
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      <msajsp:companyName/><BR>
      <msajsp:formerCompanyName fullDescription="true"/>
</TABLE>
<P>
Learn more about <B><msajsp:companyName/></B>
<msajsp:formerCompanyName fullDescription="true"/>
<UL>
  <LI><A HREF="products.jsp"><msajsp:companyName/> products</A>
  <LI><A HREF="services.jsp"><msajsp:companyName/> services</A>
  <LI><A HREF="history.jsp"><msajsp:companyName/> history</A>
  <LI><A HREF="invest.jsp">investing in <msajsp:companyName/></A>
  <LI><A HREF="contact.jsp">contacting <msajsp:companyName/></A>
</UL>

</BODY>
</HTML>
```


not-dot-com.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/index2.jsp
Go Links
Local intranet

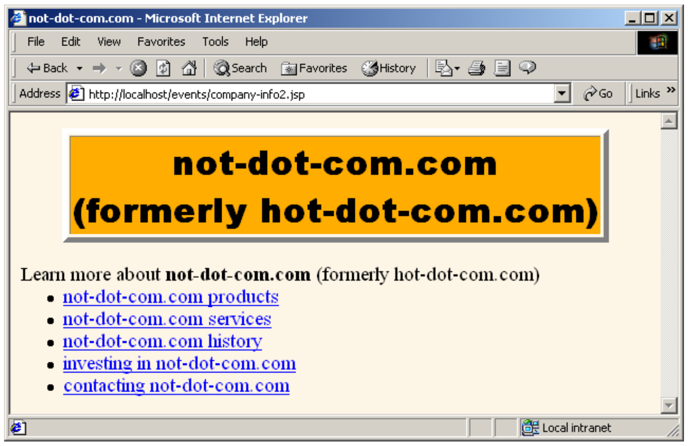
not-dot-com.com
(formerly hot-dot-com.com)

Welcome to the home page of **not-dot-com.com** (formerly hot-dot-com.com)

not-dot-com.com is a high-flying, fast-growing, big-potential company. A perfect choice for your retirement portfolio!

Click here for more information.

Figure 10–9 Reworking the company home page to use custom tags results in an identical appearance (compare Figure 10–1) but yields JSP code that is significantly easier to read and maintain.


not-dot-com.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/company-info2.jsp
Go Links
Local intranet

not-dot-com.com
(formerly hot-dot-com.com)

Learn more about **not-dot-com.com** (formerly hot-dot-com.com)

- not-dot-com.com products
- not-dot-com.com services
- not-dot-com.com history
- investing in not-dot-com.com
- contacting not-dot-com.com

Figure 10–10 Reworking the company information page to use custom tags results in an identical appearance (compare Figure 10–2) but yields JSP code that is significantly easier to read and maintain.

10.7 Recognizing Session Creation and Destruction

Classes that implement the `ServletContextListener` and `ServletContext-AttributeListener` interfaces respond to changes in the servlet context, which is shared by all servlets and JSP pages in the Web application. But, with session tracking (Section 2.10), data is stored in per-user `HttpSession` objects, not in the servlet context. What if you want to monitor changes to this user-specific data? That's the job of the `HttpSessionListener` and `HttpSessionAttributeListener` interfaces. This section discusses `HttpSessionListener`, the listener that is notified when a session is created or destroyed (either deliberately with `invalidate` or by timing out). Section 10.9 discusses `HttpSessionAttributeListener`, the listener that is notified when session attributes are added, replaced, or removed.

Using `HttpSessionListener` involves the following steps.

1. **Implement the `HttpSessionListener` interface.** This interface is in the `javax.servlet.http` package.
2. **Override `sessionCreated` and `sessionDestroyed`.** The first of these (`sessionCreated`) is triggered when a new session is created. The second method (`sessionDestroyed`) is triggered when a a session is destroyed. This destruction could be due to an explicit call to the `invalidate` method or because the elapsed time since the last client access exceeds the session timeout.

3. **Obtain a reference to the session and possibly to the servlet context.** Each of the two `HttpSessionListener` methods takes an `HttpSessionEvent` as an argument. The `HttpSessionEvent` class has a `getSession` method that provides access to the session object. You almost always want this reference; you occasionally also want a reference to the servlet context. If so, first obtain the session object and then call `getServletContext` on it.
4. **Use the objects.** Surprisingly, one of the only methods you usually call on the session object is the `setAttribute` method. You do this in `sessionCreated` if you want to guarantee that all sessions have a certain attribute. Wait! What about `getAttribute`? Nope; you don't use it. In `sessionCreated`, there is nothing in the session yet, so `getAttribute` is pointless. In addition, all attributes are removed *before* `sessionDestroyed` is called, so calling `getAttribute` is also pointless there. If you want to clean up attributes that are left in sessions that time out, you use the `attributeRemoved` method of `HttpSessionAttributeListener` (Section 10.9). Consequently, `sessionDestroyed` is mostly reserved for listeners that are simply keeping track of the number of sessions in use.
5. **Declare the listener.** In the *web.xml* or TLD file, use the `listener` and `listener-class` elements to simply list the fully qualified name of the listener class, as below.

```
<listener>
  <listener-class>somePackage.SomeListener</listener-class>
</listener>
```

10.8 Example: A Listener That Counts Sessions

Session tracking can significantly increase the server's memory load. For example, if a site that uses session tracking has 1,000 unique visitors per hour and the server uses a two-hour session timeout, the system will have approximately 2,000 sessions in memory at any one time. Reducing the timeout to one hour would cut the session memory requirements in half but would risk having active sessions prematurely time out. You need to track typical usage before you can decide on the appropriate solution.

So, you need a listener that will keep track of how many sessions are created, how many are destroyed, and how many are in memory at any one time. Assuming that you have no explicit calls to `invalidate`, the session destructions correspond to expired timeouts.

The following steps summarize a listener that accomplishes this task.

1. **Implement the `HttpSessionListener` interface.** Listing 10.20 shows a class (`SessionCounter`) that implements this interface.
2. **Override `sessionCreated` and `sessionDestroyed`.** The first of these (`sessionCreated`) increments two counters: `totalSessionCount` and `currentSessionCount`. If the current count is greater than the previous maximum count, the method also increments the `maxSessionCount` variable. The second method (`sessionDestroyed`) decrements the `currentSessionCount` variable.
3. **Obtain and use the servlet context.** In this application, no specific use is made of the session object. The only thing that matters is the fact that a session was created or destroyed, not any details about the session itself. But, the session counts have to be placed in a location that is easily accessible to servlets and JSP pages that will display the counts. So, the first time `sessionCreated` is called, it obtains the session object, calls `getServletContext` on it, and then calls `setAttribute` to store the listener object in the servlet context.
4. **Declare the listener.** Listing 10.21 shows the *web.xml* file. It declares the listener with the `listener` and `listener-class` elements, as below.

```
<listener>
  <listener-class>
    moreservlets.listeners.SessionCounter
  </listener-class>
</listener>
```

Listing 10.22 shows a JSP page that displays the session counts. Figure 10–11 shows a typical result.

In order to test session creation and timeout, I made three temporary changes.

First, I disabled cookies in my browser. Since my servers are set to use cookies for session tracking, this had the result of making each request be a new session. See the following subsection for information on disabling cookies in Netscape and Internet Explorer.

Second, I created an HTML page (Listing 10.23, Figure 10–12) that used frames with four rows and four columns to request the same JSP page (Listing 10.24) 16 times. In an environment that has cookies disabled, a request for the framed page results in 16 new sessions being created on the server (recall that JSP pages perform session tracking automatically unless the `session` attribute of the `page` directive is set to `false`—see Section 3.4).

Third, I chose an extremely low session timeout: two minutes. This saved me from waiting for hours to test the session-counting listener. Changing the default session timeout is discussed in Section 5.10, but it simply amounts to creating a `session-config` entry in *web.xml*, as follows.

```
  <session-config>
    <session-timeout>2</session-timeout>
  </session-config>
```

Listing 10.20 *SessionCounter.java*

```
package moreservlets.listeners;

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

/** Listener that keeps track of the number of sessions
 *  that the Web application is currently using and has
 *  ever used in its life cycle.
 */

public class SessionCounter implements HttpSessionListener {
  private int totalSessionCount = 0;
  private int currentSessionCount = 0;
  private int maxSessionCount = 0;
  private ServletContext context = null;

  public void sessionCreated(HttpSessionEvent event) {
    totalSessionCount++;
    currentSessionCount++;
    if (currentSessionCount > maxSessionCount) {
      maxSessionCount = currentSessionCount;
    }
    if (context == null) {
      storeInServletContext(event);
    }
  }

  public void sessionDestroyed(HttpSessionEvent event) {
    currentSessionCount--;
  }

  /** The total number of sessions created. */

  public int getTotalSessionCount() {
    return(totalSessionCount);
  }

  /** The number of sessions currently in memory. */

  public int getCurrentSessionCount() {
    return(currentSessionCount);
  }
```

Listing 10.20 *SessionCounter.java (continued)*

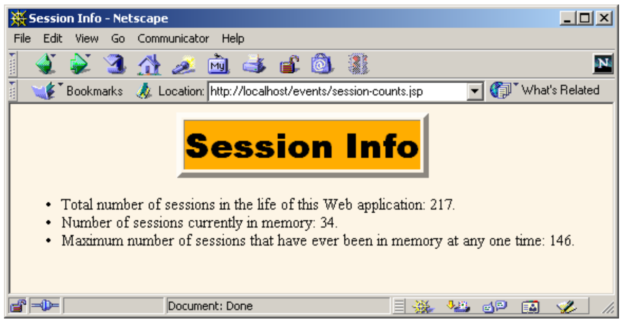
```
  /** The largest number of sessions ever in memory
   *  at any one time.
   */

  public int getMaxSessionCount() {
    return(maxSessionCount);
  }

  // Register self in the servlet context so that
  // servlets and JSP pages can access the session counts.

  private void storeInServletContext(HttpSessionEvent event) {
    HttpSession session = event.getSession();
    context = session.getServletContext();
    context.setAttribute("sessionCounter", this);
  }
}
```

Listing 10.21 *web.xml* (Excerpt for session counting listener)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Register the session counting event listener. -->
  <listener>
    <listener-class>
      moreservlets.listeners.SessionCounter
    </listener-class>
  </listener>
  <!-- ... -->

  <!-- Set the default session timeout to two minutes. -->
  <session-config>
    <session-timeout>2</session-timeout>
  </session-config>

  <!-- ... -->
</web-app>
```

Listing 10.22 *session-counts.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Session Info</TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Session Info</TABLE>
<P>

<jsp:useBean class="moreservlets.listeners.SessionCounter"
             id="sessionCounter" scope="application" />
<UL>
<LI>Total number of sessions in the life of this
    Web application:
    <jsp:getProperty name="sessionCounter"
                     property="totalSessionCount" />.
<LI>Number of sessions currently in memory:
    <jsp:getProperty name="sessionCounter"
                     property="currentSessionCount" />.
<LI>Maximum number of sessions that have ever been in
    memory at any one time:
    <jsp:getProperty name="sessionCounter"
                     property="maxSessionCount" />.
</UL>

</BODY>
</HTML>
```


Session Info - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/events/session-counts.jsp
What's Related
Document: Done

Session Info

- Total number of sessions in the life of this Web application: 217.
- Number of sessions currently in memory: 34.
- Maximum number of sessions that have ever been in memory at any one time: 146.

Figure 10–11 The `SessionCounter` listener keeps track of the sessions used in the Web application.

Listing 10.23 *make-sessions.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
  <TITLE>Session Testing...</TITLE>
</HEAD>

<FRAMESET ROWS="*,*,*,*" COLS="*,*,*,*">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <NOFRAMES><BODY>
    This example requires a frame-capable browser.
  </BODY></NOFRAMES>
</FRAMESET>
</HTML>
```

Listing 10.24 *test.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- The purpose of this page is to force the system
     to create a session. -->
<HTML>
<HEAD><TITLE>Test</TITLE></HEAD>

<%@ page import="moreservlets.*" %>
<BODY BGCOLOR="<%= ColorUtils.randomColor() %>">

</BODY></HTML>
```

Listing 10.25 *ColorUtils.java*

```
package moreservlets;

/** Small utility to generate random HTML color names. */

public class ColorUtils {
  // The official HTML color names.
  private static String[] htmlColorNames =
    { "AQUA", "BLACK", "BLUE", "FUCHSIA", "GRAY", "GREEN",
      "LIME", "MAROON", "NAVY", "OLIVE", "PURPLE", "RED",
      "SILVER", "TEAL", "WHITE", "YELLOW" };

  public static String randomColor() {
    int index = randomInt(htmlColorNames.length);
    return(htmlColorNames[index]);
  }

  // Returns a random number from 0 to n-1 inclusive.

  private static int randomInt(int n) {
    return((int)(Math.random() * n));
  }
}
```

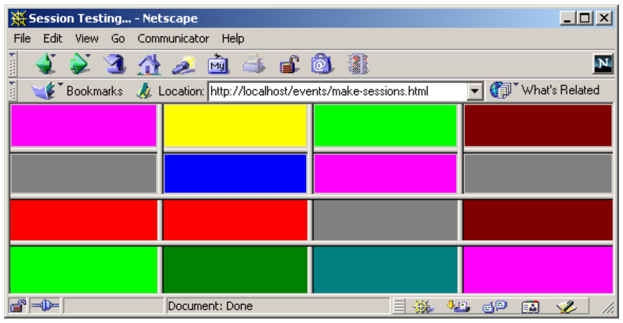
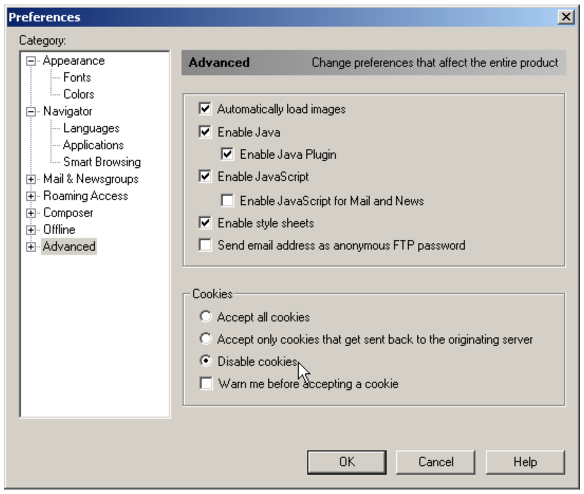

Session Testing... - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/events/make-sessions.html
What's Related
Document: Done

Figure 10–12 Session management was tested with a frame-based page that was invoked after cookies were disabled. So, each request resulted in 16 different sessions.

Disabling Cookies

Figures 10–13 through 10–15 summarize the approach to disabling cookies in Netscape 4, Netscape 6, and Internet Explorer 5. As discussed in the previous subsection, temporarily disabling cookies is useful for testing session usage.



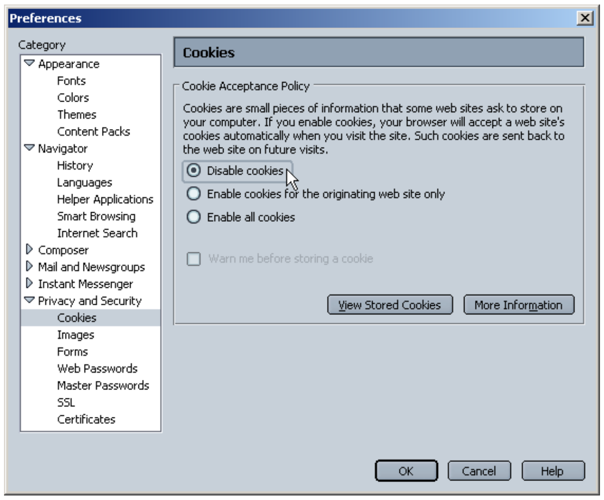
Advanced — Change preferences that affect the entire product

- [✓] Automatically load images
- [✓] Enable Java
  - [✓] Enable Java Plugin
- [✓] Enable JavaScript
  - [ ] Enable JavaScript for Mail and News
- [✓] Enable style sheets
- [ ] Send email address as anonymous FTP password

Cookies

- ○ Accept all cookies
- ○ Accept only cookies that get sent back to the originating server
- ● Disable cookies
- [ ] Warn me before accepting a cookie

Figure 10–13 To disable cookies in Netscape 4, choose the Edit menu, then Preferences, then Advanced. Select “Disable cookies.” Reset the browser after you are done testing; my preferred setting is “Accept only cookies that get sent back to the originating server.”



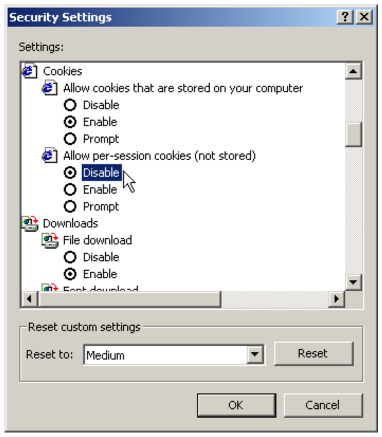
Cookies

Cookie Acceptance Policy

Cookies are small pieces of information that some web sites ask to store on your computer. If you enable cookies, your browser will accept a web site's cookies automatically when you visit the site. Such cookies are sent back to the web site on future visits.

- ● Disable cookies
- ○ Enable cookies for the originating web site only
- ○ Enable all cookies
- [ ] Warn me before storing a cookie

Figure 10–14 To disable cookies in Netscape 6, choose the Edit menu, then Preferences, then Privacy and Security, then Cookies. Select “Disable cookies.” Reset the browser after you are done testing; my preferred setting is “Enable cookies for the originating web site only.”


Security Settings

Settings:

Cookies

Allow cookies that are stored on your computer

Disable [✓ Enable] Prompt

Allow per-session cookies (not stored)

[✓ Disable] Enable Prompt

Downloads

File download

Disable [✓ Enable]

Figure 10–15 To disable cookies in Internet Explorer 5, choose the Tools menu, then Internet Options, then Security, then Custom Level. Since the goal here is to generate multiple sessions, you only need to disable per-session cookies. When done testing, reset the browser by changing the setting back to Enable.

10.9 Watching for Changes in Session Attributes

OK, so `HttpSessionListener` lets you detect when a session is created or destroyed. But, since session attributes are removed before session destruction, this listener does not let you clean up attributes that are in destroyed sessions. That's the job of the `HttpSessionAttributeListener` interface. Use of this interface involves the following steps.

1. **Implement the `HttpSessionAttributeListener` interface.** This interface is in the `javax.servlet.http` package.
2. **Override `attributeAdded`, `attributeReplaced`, and `attributeRemoved`.** The `attributeAdded` method is triggered when a new attribute is added to a session. When a new value is assigned to an existing session attribute, `attributeAdded` is triggered with the new value and `attributeReplaced` is triggered with the old value (i.e., the value being replaced). The `attribute-Removed` method is triggered when a session attribute is removed altogether. This removal can be due to an explicit programmer call to `removeAttribute`, but is more commonly due to the system remov-

ing all attributes of sessions that are about to be deleted because their timeout expired.

3. **Obtain references to the attribute name, attribute value, session, and servlet context.** Each of the three `HttpSession-AttributeListener` methods takes an `HttpSessionBinding-Event` as an argument. The `HttpSessionBindingEvent` class has three useful methods: `getName` (the name of the attribute that was changed), `getValue` (the value of the changed attribute—the new value for `attributeAdded` and the previous value for `attribute-Replaced` and `attributeRemoved`), and `getSession` (the `HttpSession` object). If you also want access to the servlet context, first obtain the session and then call `getServletContext` on it.
4. **Use the objects.** The attribute name is usually compared to a stored name to see if it is the one you are monitoring. The attribute value is used in an application-specific manner. The session is usually used to read previously stored attributes (`getAttribute`) or to store new or changed attributes (`setAttribute`).
5. **Declare the listener.** In the *web.xml* or TLD file, use the `listener` and `listener-class` elements to simply list the fully qualified name of the listener class, as below.

```
<listener>
  <listener-class>somePackage.SomeListener</listener-class>
</listener>
```

10.10 Example: Monitoring Yacht Orders

You're "promoted" to sales manager. (OK, ok, so that is too horrible a fate to contemplate. All right then, you are asked to help the sales manager.) You want to track buying patterns for a specific item (a yacht, in this case). Of course, you could try to find all servlets and JSP pages that process orders and change each one to record yacht purchases. That's an awful lot of work for what sounds like a simple request, though. Pretty hard to maintain, anyhow.

A much better option is to create a session attribute listener that monitors the attributes corresponding to order reservations or purchases and that records the information in the log file for later perusal by the sales manager.

The following steps summarize a listener that accomplishes this task.

1. **Implement the `HttpSessionAttributeListener` interface.** Listing 10.26 shows a class (`YachtWatcher`) that implements this interface.

2. **Override `attributeAdded`, `attributeReplaced`, and `attributeRemoved`.** The first of these (`attributeAdded`) is used to log the fact that a yacht was reserved (tentative) or purchased (permanent). The other two methods are used to print retractions of order reservations (but not purchases—all sales are final).
3. **Obtain references to the attribute name, attribute value, session, and servlet context.** Each of the three methods calls `getName` and `getValue` on its `HttpSessionBindingEvent` argument to obtain the name and value of the modified attribute. The methods also call `getServletContext` on the session object (obtained with `getSession`) to get a reference to the servlet context.
4. **Use the objects.** The attribute name is compared to `"orderedItem"` (attribute addition, replacement, and removal) and `"purchasedItem"` (attribute addition only). If the name matches, then the attribute value is compared to `"yacht"`. If that comparison also succeeds, then the `log` method of the servlet context is called.
5. **Declare the listener.** Listing 10.27 shows the *web.xml* file. It declares the listener with the `listener` and `listener-class` elements, as below.

```
<listener>
  <listener-class>
    moreservlets.listeners.YachtWatcher
  </listener-class>
</listener>
```

Listings 10.28 and 10.29 show a servlet that handles orders and an HTML form that sends it data, respectively. Figures 10–16 through 10–19 show the results. Listing 10.30 shows a portion of the resultant log file.

Listing 10.26 *YachtWatcher.java*

```
package moreservlets.listeners;

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

/** Listener that keeps track of yacht purchases by monitoring
 *  the orderedItem and purchasedItem session attributes.
 */

public class YachtWatcher
    implements HttpSessionAttributeListener {
```

Listing 10.26 *YachtWatcher.java (continued)*

```
private String orderAttributeName = "orderedItem";
private String purchaseAttributeName = "purchasedItem";
private String itemName = "yacht";

/** Checks for initial ordering and final purchase of
 *  yacht. Records "Customer ordered a yacht" if the
 *  orderedItem attribute matches "yacht".
 *  Records "Customer finalized purchase of a yacht" if the
 *  purchasedItem attribute matches "yacht".
 */

public void attributeAdded(HttpSessionBindingEvent event) {
  checkAttribute(event, orderAttributeName, itemName,
                 " ordered a ");
  checkAttribute(event, purchaseAttributeName, itemName,
                 " finalized purchase of a ");
}

/** Checks for order cancellation: was an order for "yacht"
 *  cancelled?  Records "Customer cancelled an order for
 *  a yacht" if the orderedItem attribute matches "yacht".
 */

public void attributeRemoved(HttpSessionBindingEvent event) {
  checkAttribute(event, orderAttributeName, itemName,
                 " cancelled an order for a ");
}

/** Checks for item replacement: was "yacht" replaced
 *  by some other item? Records "Customer changed to a new
 *  item instead of a yacht" if the orderedItem attribute
 *  matches "yacht".
 */

public void attributeReplaced(HttpSessionBindingEvent event) {
  checkAttribute(event, orderAttributeName, itemName,
                 " changed to a new item instead of a ");
}

private void checkAttribute(HttpSessionBindingEvent event,
                            String orderAttributeName,
                            String keyItemName,
                            String message) {
  String currentAttributeName = event.getName();
  String currentItemName = (String)event.getValue();
  if (currentAttributeName.equals(orderAttributeName) &&
      currentItemName.equals(keyItemName)) {
```

Listing 10.26 *YachtWatcher.java (continued)*

```
      ServletContext context =
        event.getSession().getServletContext();
      context.log("Customer" + message + keyItemName + ".");
    }
  }
}
```

Listing 10.27 *web.xml* (Excerpt for yacht-watching listener)

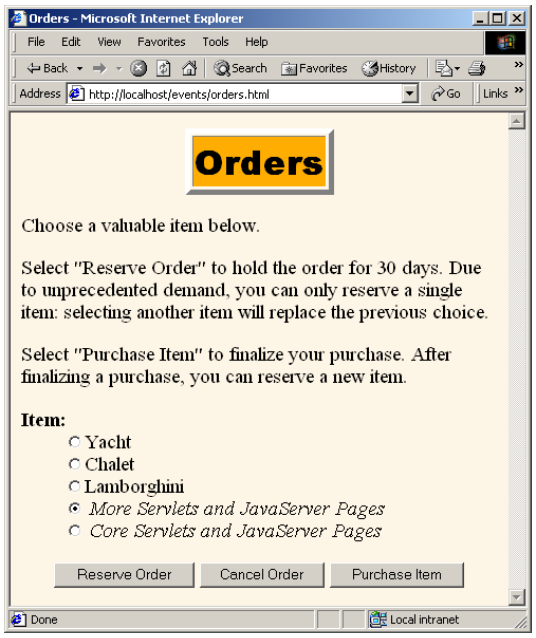
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Register the yacht-watching event listener. -->
  <listener>
    <listener-class>
      moreservlets.listeners.YachtWatcher
    </listener-class>
  </listener>
  <!-- ... -->

<!-- Assign the name OrderHandlingServlet to
       moreservlets.OrderHandlingServlet. -->
  <servlet>
    <servlet-name>OrderHandlingServlet</servlet-name>
    <servlet-class>
      moreservlets.OrderHandlingServlet
    </servlet-class>
  </servlet>
  <!-- ... -->

  <!-- Assign the URL /HandleOrders to the
       servlet that is named OrderHandlingServlet.
  -->
  <servlet-mapping>
    <servlet-name>OrderHandlingServlet</servlet-name>
    <url-pattern>/HandleOrders</url-pattern>
  </servlet-mapping>

  <!-- ... -->
</web-app>
```

Listing 10.28 *OrderHandlingServlet.java*

```
package moreservlets;

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

/** Servlet that handles submissions from the order form. If the
 *  user selects the "Reserve Order" button, the selected item
 *  is put into the orderedItem attribute. If the user selects
 *  the "Cancel Order" button, the orderedItem attribute is
 *  deleted. If the user selects the "Purchase Item" button,
 *  the selected item is put into the purchasedItem attribute.
 */

public class OrderHandlingServlet extends HttpServlet {
  private String title, picture;

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    String itemName = request.getParameter("itemName");
    if ((itemName == null) || (itemName.equals(""))) {
      itemName = "<B>MISSING ITEM</B>";
    }
    String message;
    if (request.getParameter("order") != null) {
      session.setAttribute("orderedItem", itemName);
      message = "Thanks for ordering " + itemName + ".";
    } else if (request.getParameter("cancel") != null) {
      session.removeAttribute("orderedItem");
      message = "Thanks for nothing.";
    } else {
      session.setAttribute("purchasedItem", itemName);
      message = "Thanks for purchasing " + itemName + ".";
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println
      (docType +
       "<HTML>\n" +
       "<HEAD><TITLE>" + message + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\">\n" +
       "<H2 ALIGN=\"CENTER\">" + message + "</H2>\n" +
       "</BODY></HTML>");
  }
}
```

Listing 10.29 *orders.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Orders</TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Orders
</TABLE>
<P>
Choose a valuable item below.
<P>
Select "Reserve Order" to hold the order for 30 days. Due to
unprecedented demand, you can only reserve a single item:
selecting another item will replace the previous choice.
<P>
Select "Purchase Item" to finalize your purchase. After
finalizing a purchase, you can reserve a new item.
<FORM ACTION="HandleOrders">
<DL>
  <DT><B>Item:</B>
  <DD><INPUT TYPE="RADIO" NAME="itemName" VALUE="yacht">Yacht
  <DD><INPUT TYPE="RADIO" NAME="itemName" VALUE="chalet">Chalet
  <DD><INPUT TYPE="RADIO" NAME="itemName" VALUE="car">Lamborghini
  <DD><INPUT TYPE="RADIO" NAME="itemName" VALUE="msajsp" CHECKED>
      <I>More Servlets and JavaServer Pages</I>
  <DD><INPUT TYPE="RADIO" NAME="itemName" VALUE="csajsp">
      <I>Core Servlets and JavaServer Pages</I>
</DL>
<CENTER>
<INPUT TYPE="SUBMIT" NAME="order" VALUE="Reserve Order">
<INPUT TYPE="SUBMIT" NAME="cancel" VALUE="Cancel Order">
<INPUT TYPE="SUBMIT" NAME="purchase" VALUE="Purchase Item">
</CENTER>
</FORM>

</BODY>
</HTML>
```


Orders - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/orders.html
Go Links
Done
Local intranet

Orders

Choose a valuable item below.

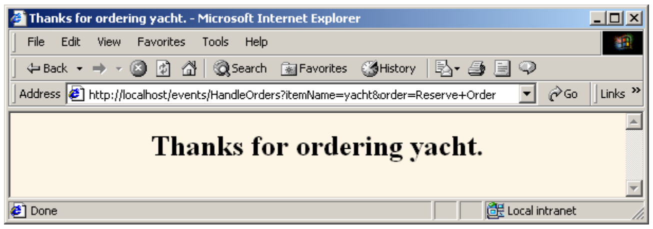
Select "Reserve Order" to hold the order for 30 days. Due to unprecedented demand, you can only reserve a single item: selecting another item will replace the previous choice.

Select "Purchase Item" to finalize your purchase. After finalizing a purchase, you can reserve a new item.

Item:
- ○ Yacht
- ○ Chalet
- ○ Lamborghini
- ● *More Servlets and JavaServer Pages*
- ○ *Core Servlets and JavaServer Pages*

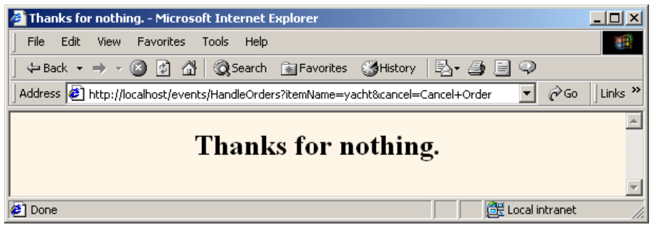
Reserve Order | Cancel Order | Purchase Item

Figure 10–16 The order form that sends data to the order handling servlet (Listing 10.28). That servlet adds, replaces, and removes values in the `orderedItem` and `purchasedItem` session attributes, which in turn triggers the yacht-watching listener (Listing 10.26).


Thanks for ordering yacht. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/HandleOrders?itemName=yacht&order=Reserve+Order
Go Links
Done
Local intranet

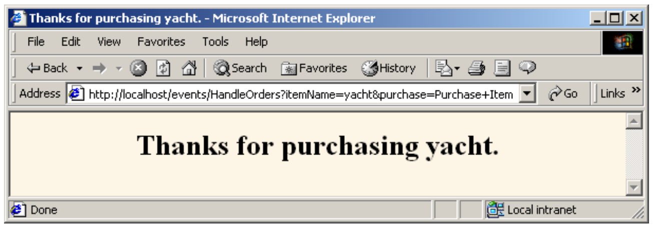
Thanks for ordering yacht.

Figure 10–17 Result of reserving an order for a yacht. The yacht-watching listener makes an entry in the log file (Listing 10.30) saying that a customer ordered a yacht.


Thanks for nothing. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/HandleOrders?itemName=yacht&cancel=Cancel+Order
Go Links
Done
Local intranet

Thanks for nothing.

Figure 10–18 Result of cancelling an order. If the user had previously reserved an order for a yacht, the yacht-watching listener makes an entry in the log file (Listing 10.30) saying that a customer replaced a yacht order with something else.


Thanks for purchasing yacht. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/events/HandleOrders?itemName=yacht&purchase=Purchase+Item
Go Links
Done
Local intranet

Thanks for purchasing yacht.

Figure 10–19 Result of purchasing a yacht. The yacht-watching listener makes an entry in the log file (Listing 10.30) saying that a customer purchased a yacht.

Listing 10.30 Sample Log File Entries

```
2001-11-07 11:50:59 Customer ordered a yacht.
2001-11-07 11:51:06 Customer changed to a new item instead of a
yacht.
2001-11-07 11:52:37 Customer cancelled an order for a yacht.
2001-11-07 11:53:05 Customer finalized purchase of a yacht.
2001-11-07 11:53:35 Customer ordered a yacht.
2001-11-07 11:53:50 Customer cancelled an order for a yacht.
2001-11-07 11:54:20 Customer changed to a new item instead of a
yacht.
2001-11-07 11:54:27 Customer changed to a new item instead of a
yacht.
2001-11-07 11:54:42 Customer cancelled an order for a yacht.
2001-11-07 11:54:44 Customer ordered a yacht.
2001-11-07 11:54:47 Customer changed to a new item instead of a
yacht.
```

10.11 Using Multiple Cooperating Listeners

Now, the listeners discussed in this chapter are all well and good. There are plenty of applications where one of them is useful. However, there are also plenty of applications where no *single* listener can, in isolation, accomplish the necessary tasks. *Multiple* listeners need to work together.

For example, suppose that your yacht-watching listener was so successful that you are asked to expand it. Rather than tracking buying patterns of a fixed item such as a yacht, you should track orders for the current daily special to let management discover if their specials are effective. Accomplishing this task requires *three* listeners to cooperate: a `ServletContextListener` to set up application-wide information about the session attributes that store daily specials, a `ServletContextAttributeListener` to monitor changes to the attributes that store the information, and an `HttpSessionAttributeListener` to keep a running count of orders for the daily special.

The three listeners are described in more detail in the following subsections.

Tracking Orders for the Daily Special

As the first step in creating an order-tracking system, you need a servlet context listener to read initialization parameters that specify which session attributes correspond to orders and which items are the current daily specials. These values should be stored in the servlet context so other resources can determine what the daily specials are. Listing 10.31 shows this listener.

Second, you need a session attribute listener to keep a running count of orders for the daily special. The count will be incremented every time a designated attribute name is added with any of the daily specials as its value. The count will be decremented every time a designated attribute is replaced or removed and the previous value is one of the daily specials. Listing 10.32 shows this listener.

Listing 10.33 shows the deployment descriptor that registers the two listeners and sets up the servlet context initialization parameters that designate the names of order-related session attributes and the names of the daily specials.

Listing 10.34 shows a JSP page that prints the current order count. Figures 10–20 through 10–22 show some typical results.

Listing 10.31 *DailySpecialRegistrar.java*

```
package moreservlets.listeners;

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

/** Listener that records how to detect orders
 *  of the daily special. It reads a list of attribute
 *  names from an init parameter: these correspond to
 *  session attributes that are used to record orders.
 *  It also reads a list of item names: these correspond
 *  to the names of the daily specials. Other listeners
 *  will watch to see if any daily special names appear
 *  as values of attributes that are hereby designated
 *  to refer to orders.
 */

public class DailySpecialRegistrar
    implements ServletContextListener {

  /** When the Web application is loaded, record the
   *  attribute names that correspond to orders and
   *  the attribute values that are the daily specials.
   *  Also set to zero the count of daily specials that have
   *  been ordered.
   */

  public void contextInitialized(ServletContextEvent event) {
    ServletContext context = event.getServletContext();
    addContextEntry(context, "order-attribute-names");
    addContextEntry(context, "daily-special-item-names");
    context.setAttribute("dailySpecialCount", new Integer(0));
  }

  public void contextDestroyed(ServletContextEvent event) {}

  /** Read the designated context initialization parameter,
   *  put the values into an ArrayList, and store the
   *  list in the ServletContext with an attribute name
   *  that is identical to the initialization parameter name.
   */
```

Listing 10.31 *DailySpecialRegistrar.java (continued)*

```
  private void addContextEntry(ServletContext context,
                               String initParamName) {
    ArrayList paramValues = new ArrayList();
    String attributeNames =
      context.getInitParameter(initParamName);
    if (attributeNames != null) {
      StringTokenizer tok = new StringTokenizer(attributeNames);
      String value;
      while(tok.hasMoreTokens()) {
        value = tok.nextToken();
        paramValues.add(value);
      }
      context.setAttribute(initParamName, paramValues);
    }
  }

  /** Returns a string containing the daily special
   *  names. For insertion inside an HTML text area.
   */

  public static String dailySpecials(ServletContext context) {
    String attributeName = "daily-special-item-names";
    ArrayList itemNames =
      (ArrayList)context.getAttribute(attributeName);
    String itemString = "";
    for(int i=0; i<itemNames.size(); i++) {
      itemString = itemString + (String)itemNames.get(i) + "\n";
    }
    return(itemString);
  }

  /** Returns a UL list containing the daily special
   *  names. For insertion within the body of a JSP page.
   */

  public static String specialsList(ServletContext context) {
    String attributeName = "daily-special-item-names";
    ArrayList itemNames =
      (ArrayList)context.getAttribute(attributeName);
    String itemString = "<UL>\n";
    for(int i=0; i<itemNames.size(); i++) {
      itemString = itemString + "<LI>" +
                   (String)itemNames.get(i) + "\n";
    }
    itemString = itemString + "</UL>";
    return(itemString);
  }
}
```

Listing 10.32 *DailySpecialWatcher.java*

```
package moreservlets.listeners;

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

/** Listener that keeps track of orders of the
 *  current daily special.
 */

public class DailySpecialWatcher
    implements HttpSessionAttributeListener {
  private static int dailySpecialCount = 0;

  /** If the name of the session attribute that was added
   *  matches one of the stored order-attribute-names AND
   *  the value of the attribute matches one of the
   *  stored daily-special-item-names, then increment
   *  the count of daily specials ordered.
   */

  public void attributeAdded(HttpSessionBindingEvent event) {
    checkForSpecials(event, 1);
  }

  /** If the name of the session attribute that was removed
   *  matches one of the stored order-attribute-names AND
   *  the value of the attribute matches one of the
   *  stored daily-special-item-names, then decrement
   *  the count of daily specials ordered.
   */

  public void attributeRemoved(HttpSessionBindingEvent event) {
    checkForSpecials(event, -1);
  }

  /** If the name of the session attribute that was replaced
   *  matches one of the stored order-attribute-names AND
   *  the value of the attribute matches one of the
   *  stored daily-special-item-names, then increment
   *  the count of daily specials ordered. Note that the
   *  value here is the old value (the one being replaced);
   *  the attributeAdded method will handle the new value
   *  (the replacement).
   */
```

Listing 10.32 *DailySpecialWatcher.java (continued)*

```
public void attributeReplaced(HttpSessionBindingEvent event) {
  checkForSpecials(event, -1);
}

// Check whether the attribute that was just added or removed
// matches one of the stored order-attribute-names AND
// the value of the attribute matches one of the
// stored daily-special-item-names. If so, add the delta
// (+1 or -1) to the count of daily specials ordered.

private void checkForSpecials(HttpSessionBindingEvent event,
                              int delta) {
  ServletContext context =
    event.getSession().getServletContext();
  ArrayList attributeNames =
    getList(context, "order-attribute-names");
  ArrayList itemNames =
    getList(context, "daily-special-item-names");
  synchronized(attributeNames) {
    for(int i=0; i<attributeNames.size(); i++) {
      String attributeName = (String)attributeNames.get(i);
      for(int j=0; j<itemNames.size(); j++) {
        String itemName = (String)itemNames.get(j);
        if (attributeName.equals(event.getName()) &&
            itemName.equals((String)event.getValue())) {
          dailySpecialCount = dailySpecialCount + delta;
        }
      }
    }
  }
  context.setAttribute("dailySpecialCount",
                       new Integer(dailySpecialCount));
}

// Get either the order-attribute-names or
// daily-special-item-names list.

private ArrayList getList(ServletContext context,
                          String attributeName) {
  ArrayList list =
    (ArrayList)context.getAttribute(attributeName);
  return(list);
}
```

Listing 10.32 *DailySpecialWatcher.java (continued)*

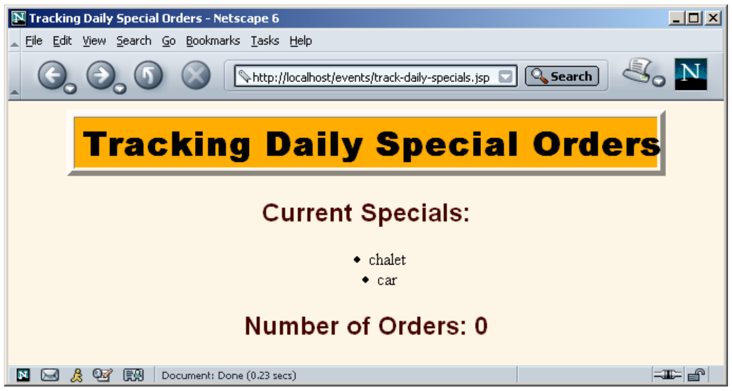
```
  /** Reset the count of daily specials that have
   *  been ordered. This operation is normally performed
   *  only when the daily special changes.
   */

  public static void resetDailySpecialCount() {
    dailySpecialCount = 0;
  }
}
```

Listing 10.33 *web.xml* (Excerpt for tracking daily special orders)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Register the listener that sets up the entries
       that will be used to monitor orders for the daily
       special. -->
  <listener>
    <listener-class>
      moreservlets.listeners.DailySpecialRegistrar
    </listener-class>
  </listener>

  <!-- Register the listener that counts orders for the daily
       special. -->
  <listener>
    <listener-class>
      moreservlets.listeners.DailySpecialWatcher
    </listener-class>
  </listener>

  <!-- ... -->
</web-app>
```

Listing 10.34 *track-daily-specials.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Tracking Daily Special Orders</TITLE>
<LINK REL=STYLESHEET
      HREF="events-styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Tracking Daily Special Orders
</TABLE>

<H2>Current Specials:</H2>
<%@ page import="moreservlets.listeners.*" %>
<%= DailySpecialRegistrar.specialsList(application) %>

<H2>Number of Orders:
<%= application.getAttribute("dailySpecialCount") %>
</H2>

</CENTER>
</BODY>
</HTML>
```


Tracking Daily Special Orders - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/track-daily-specials.jsp
Search
Document: Done (0.23 secs)

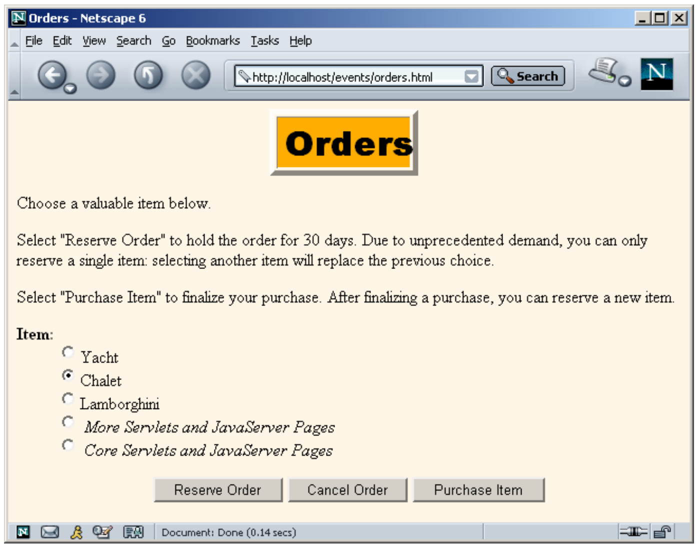
Tracking Daily Special Orders

Current Specials:

- chalet
- car

Number of Orders: 0

Figure 10–20 Initial result of *track-daily-specials.jsp*.


Orders - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/orders.html
Search

Orders

Choose a valuable item below.

Select "Reserve Order" to hold the order for 30 days. Due to unprecedented demand, you can only reserve a single item: selecting another item will replace the previous choice.

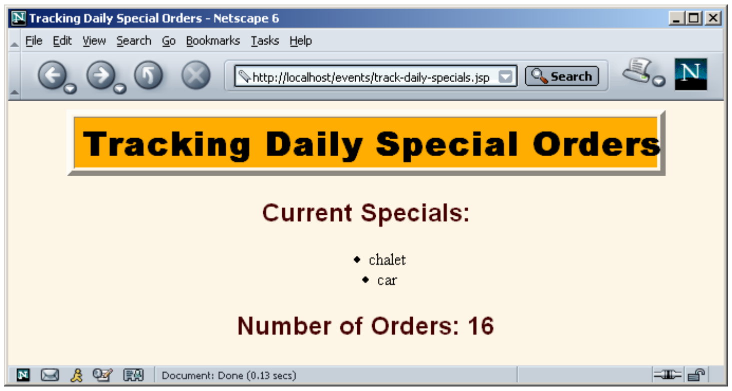
Select "Purchase Item" to finalize your purchase. After finalizing a purchase, you can reserve a new item.

Item:

- Yacht
- [✓ Chalet]
- Lamborghini
- *More Servlets and JavaServer Pages*
- *Core Servlets and JavaServer Pages*

Reserve Order | Cancel Order | Purchase Item

Figure 10–21 Ordering the daily special.


Tracking Daily Special Orders - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/track-daily-specials.jsp
Search

Tracking Daily Special Orders

Current Specials:

- chalet
- car

Number of Orders: 16

Document: Done (0.13 secs)

Figure 10–22 Result of *track-daily-specials.jsp* after several clients placed orders.

Resetting the Daily Special Order Count

The two listeners shown in the previous subsection are sufficient if you restart the server every time you change the daily specials.

However, if you change the daily specials while the server is running, you need a servlet context attribute listener to detect changes in the attribute that stores the names of the daily specials. In particular, when the daily specials change, you need to reset the running count of orders for the specials. Listing 10.35 shows this listener.

Listing 10.36 shows a JSP page that displays the current daily specials in a text area. It lets the user change the values and send them to a servlet (Listing 10.37) that records the changes in the servlet context. The JSP page is in the *admin* directory and the servlet is assigned a URL beginning with */admin* (see the *web.xml* file in Listing 10.38), so the security restrictions discussed in Section 10.4 apply.

When an authorized user changes the names of the daily specials, the order count is reset. Figures 10–23 through 10–27 show some representative results.

Listing 10.35 *ChangedDailySpecialListener.java*

```
package moreservlets.listeners;

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

/** Listener that monitors changes to the names
 *  of the daily specials (which are stored in
 *  the daily-special-item-names attribute of
 *  the servlet context). If the names change, the
 *  listener resets the running count of the number
 *  of daily specials being ordered.
 */

public class ChangedDailySpecialListener
    implements ServletContextAttributeListener {

  /** When the daily specials change, reset the
   *  order counts.
   */

  public void attributeReplaced
                 (ServletContextAttributeEvent event) {
    if (event.getName().equals("daily-special-item-names")) {
      ServletContext context = event.getServletContext();
      context.setAttribute("dailySpecialCount",
                           new Integer(0));
      DailySpecialWatcher.resetDailySpecialCount();
    }
  }

  public void attributeAdded
                 (ServletContextAttributeEvent event) {}

  public void attributeRemoved
                 (ServletContextAttributeEvent event) {}
}
```

Listing 10.36 *change-daily-specials.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Changing Daily Specials</TITLE>
<LINK REL=STYLESHEET
      HREF="../events-styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Changing Daily Specials
</TABLE>
<P>
<FORM ACTION="ChangeDailySpecial">
New specials:<BR>
<%@ page import="moreservlets.listeners.*" %>
<TEXTAREA NAME="newSpecials" ROWS=4 COLS=30>
<%= DailySpecialRegistrar.dailySpecials(application) %>
</TEXTAREA>
<P>
<INPUT TYPE="SUBMIT" VALUE="Submit Change">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Listing 10.37 *ChangeDailySpecial.java*

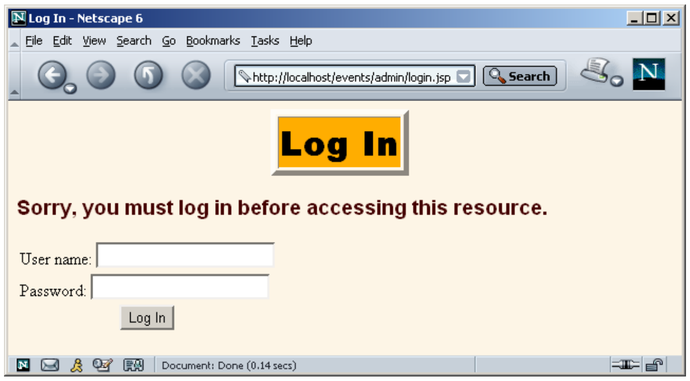
```
package moreservlets;

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

/** Servlet that changes the daily specials. The web.xml
 *  file specifies that only authenticated users in the
 *  ceo role can access the servlet. A servlet context
 *  attribute listener resets the count of daily special
 *  orders when this servlet (or any other program) changes
 *  the daily specials.
 */
```

Listing 10.37 *ChangeDailySpecial.java (continued)*

```
public class ChangeDailySpecial extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
    String dailySpecialNames =
      request.getParameter("newSpecials");
    if ((dailySpecialNames == null) ||
        (dailySpecialNames.equals(""))) {
      dailySpecialNames = "MISSING-VALUE";
    }
    ArrayList specials = new ArrayList();
    StringTokenizer tok =
      new StringTokenizer(dailySpecialNames);
    while(tok.hasMoreTokens()) {
      specials.add(tok.nextToken());
    }
    ServletContext context = getServletContext();
    context.setAttribute("daily-special-item-names",
                         specials);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    String title = "New Daily Specials";
    out.println
      (docType +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\">\n" +
       "<H2 ALIGN=\"CENTER\">" + title + "</H2>\n" +
       "<UL>");
    String special;
    for(int i=0; i<specials.size(); i++) {
      special = (String)specials.get(i);
      out.println("<LI>" + special);
    }
    out.println("</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Listing 10.38 *web.xml* (Excerpt for resetting order counts)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->

  <!-- Register the listener that resets the order counts
       when the names of the daily specials change. -->
  <listener>
    <listener-class>
      moreservlets.listeners.ChangedDailySpecialListener
    </listener-class>
  </listener>
  <!-- ... -->

  <!-- Assign the name ChangeDailySpecial to
       moreservlets.ChangeDailySpecial. -->
  <servlet>
    <servlet-name>ChangeDailySpecial</servlet-name>
    <servlet-class>
      moreservlets.ChangeDailySpecial
    </servlet-class>
  </servlet>
  <!-- ... -->

  <!-- Assign the URL /admin/ChangeDailySpecial to the
       servlet that is named ChangeDailySpecial.
  -->
  <servlet-mapping>
    <servlet-name>ChangeDailySpecial</servlet-name>
    <url-pattern>/admin/ChangeDailySpecial</url-pattern>
  </servlet-mapping>

  <!-- ... -->
</web-app>
```


Log In - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/login.jsp
Search
Document: Done (0.14 secs)

Log In

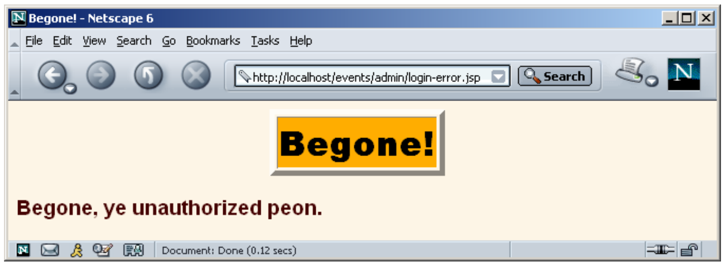
Sorry, you must log in before accessing this resource.

User name:

Password:

Log In

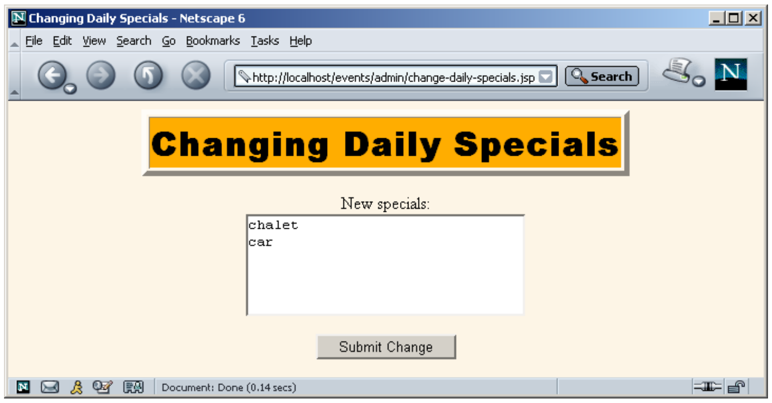
Figure 10–23 Requests by unauthenticated users for *change-daily-specials.jsp* get sent to the login page (Listing 10.9).


Begone! - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/login-error.jsp
Search
Document: Done (0.12 secs)

Begone!

Begone, ye unauthorized peon.

Figure 10–24 Users who fail authentication are shown the login-failure page (Listing 10.10).


Changing Daily Specials - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/change-daily-specials.jsp
Search
Document: Done (0.14 secs)

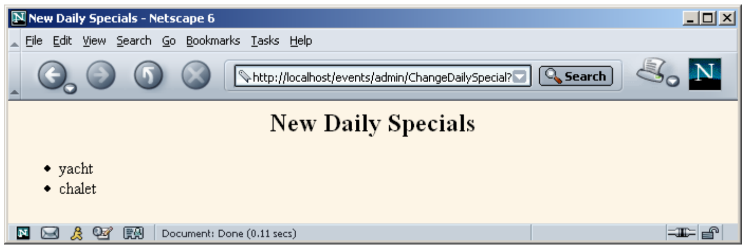
Changing Daily Specials

New specials:

```
chalet
car
```

Submit Change

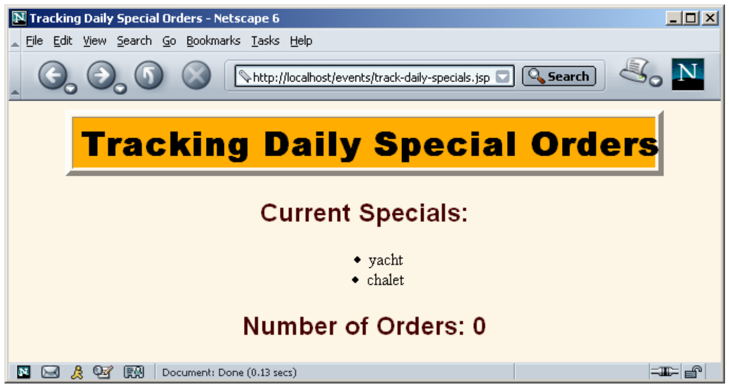
Figure 10–25 Users who pass authentication and are in the designated role (`ceo`) are shown the form for changing the daily specials (Listing 10.36). The current daily specials are displayed as the initial value of the text area.


New Daily Specials - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/admin/ChangeDailySpecial?
Search
Document: Done (0.11 secs)

New Daily Specials

- yacht
- chalet

Figure 10–26 Result of submitting the form for changing daily specials after `yacht` and `chalet` are entered in the text area.


Tracking Daily Special Orders - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/events/track-daily-specials.jsp
Search
Document: Done (0.13 secs)

Tracking Daily Special Orders

Current Specials:

- yacht
- chalet

Number of Orders: 0

Figure 10–27 When the daily specials are changed, the servlet context attribute listener (Listing 10.35) resets the order count.

10.12 The Complete Events Deployment Descriptor

The previous sections showed various excerpts of the *web.xml* file for the application events examples. This section shows the file in its entirety.

Listing 10.39 *web.xml* (Complete version for events examples)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- Order matters in web.xml! For the elements
       used in this example, this order is required:
          context-param
          listener
          servlet
          servlet-mapping
          session-config
          welcome-file-list
          taglib
          security-constraint
          login-config
  -->

  <!-- Since the company name changes so frequently,
       supply it as a servlet context parameter instead
       of embedding it into lots of different servlets and
       JSP pages. The InitialCompanyNameListener will
       read this value and store it in the servlet context. -->
  <context-param>
    <param-name>companyName</param-name>
    <param-value>not-dot-com.com</param-value>
  </context-param>

  <!-- Also store the previous company name. -->
  <context-param>
    <param-name>formerCompanyName</param-name>
    <param-value>hot-dot-com.com</param-value>
  </context-param>

  <!-- Declare the names of the session attributes that
       are used to store items that customers are
       purchasing. The daily special listener will
       track changes to the values of these attributes. -->
  <context-param>
    <param-name>order-attribute-names</param-name>
    <param-value>
      orderedItem
      purchasedItem
    </param-value>
  </context-param>
```

Listing 10.39 *web.xml* (Complete version for events examples) *(continued)*

```
  <!-- The item names of the current daily specials. -->
  <context-param>
    <param-name>daily-special-item-names</param-name>
    <param-value>
      chalet
      car
    </param-value>
  </context-param>

  <!-- Register the listener that sets up the
       initial company name. -->
<!-- Listener declaration moved to tag library...
  <listener>
    <listener-class>
      moreservlets.listeners.InitialCompanyNameListener
    </listener-class>
  </listener>
-->

  <!-- Register the listener that monitors changes to
       the company name.
  -->
<!-- Listener declaration moved to tag library...
  <listener>
    <listener-class>
      moreservlets.listeners.ChangedCompanyNameListener
    </listener-class>
  </listener>
-->

  <!-- Register the session counting event listener. -->
  <listener>
    <listener-class>
      moreservlets.listeners.SessionCounter
    </listener-class>
  </listener>

  <!-- Register the yacht-watching event listener. -->
  <listener>
    <listener-class>
      moreservlets.listeners.YachtWatcher
    </listener-class>
  </listener>
```

Listing 10.39 *web.xml* (Complete version for events examples) *(continued)*

```
<!-- Register the listener that sets up the entries
     that will be used to monitor orders for the daily
     special. -->
<listener>
  <listener-class>
    moreservlets.listeners.DailySpecialRegistrar
  </listener-class>
</listener>

<!-- Register the listener that counts orders for the daily
     special. -->
<listener>
  <listener-class>
    moreservlets.listeners.DailySpecialWatcher
  </listener-class>
</listener>

<!-- Register the listener that resets the order counts
     when the names of the daily specials change. -->
<listener>
  <listener-class>
    moreservlets.listeners.ChangedDailySpecialListener
  </listener-class>
</listener>

<!-- Assign the name ChangeCompanyName to
     moreservlets.ChangeCompanyName. -->
<servlet>
  <servlet-name>ChangeCompanyName</servlet-name>
  <servlet-class>moreservlets.ChangeCompanyName</servlet-class>
</servlet>

<!-- Assign the name OrderHandlingServlet to
     moreservlets.OrderHandlingServlet. -->
<servlet>
  <servlet-name>OrderHandlingServlet</servlet-name>
  <servlet-class>
    moreservlets.OrderHandlingServlet
  </servlet-class>
</servlet>
```

Listing 10.39 *web.xml* (Complete version for events examples) *(continued)*

```
<!-- Assign the name ChangeDailySpecial to
     moreservlets.ChangeDailySpecial. -->
<servlet>
  <servlet-name>ChangeDailySpecial</servlet-name>
  <servlet-class>
    moreservlets.ChangeDailySpecial
  </servlet-class>
</servlet>

<!-- Give a name to the servlet that redirects users
     to the home page.
-->
<servlet>
  <servlet-name>Redirector</servlet-name>
  <servlet-class>moreservlets.RedirectorServlet</servlet-class>
</servlet>

<!-- Assign the URL /admin/ChangeCompanyName to the
     servlet that is named ChangeCompanyName.
-->
<servlet-mapping>
  <servlet-name>ChangeCompanyName</servlet-name>
  <url-pattern>/admin/ChangeCompanyName</url-pattern>
</servlet-mapping>

<!-- Assign the URL /HandleOrders to the
     servlet that is named OrderHandlingServlet.
-->
<servlet-mapping>
  <servlet-name>OrderHandlingServlet</servlet-name>
  <url-pattern>/HandleOrders</url-pattern>
</servlet-mapping>

<!-- Assign the URL /admin/ChangeDailySpecial to the
     servlet that is named ChangeDailySpecial.
-->
<servlet-mapping>
  <servlet-name>ChangeDailySpecial</servlet-name>
  <url-pattern>/admin/ChangeDailySpecial</url-pattern>
</servlet-mapping>
```

Listing 10.39 *web.xml* (Complete version for events examples) *(continued)*

```
<!-- Turn off invoker. Send requests to index.jsp. -->
<servlet-mapping>
  <servlet-name>Redirector</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>

<!-- Set the default session timeout to two minutes. -->
<session-config>
  <session-timeout>2</session-timeout>
</session-config>

<!-- If URL gives a directory but no filename, try index.jsp
     first and index.html second. If neither is found,
     the result is server specific (e.g., a directory
     listing).  Order of elements in web.xml matters.
     welcome-file-list needs to come after servlet but
     before error-page.
-->
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

<!-- Register the company-name tag library. -->
<taglib>
  <taglib-uri>
    /company-name-taglib.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/company-name-taglib.tld
  </taglib-location>
</taglib>

<!-- Protect everything within the "admin" directory.
     Direct client access to this directory requires
     authentication.
-->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Admin</web-resource-name>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>ceo</role-name>
  </auth-constraint>
</security-constraint>
```

Listing 10.39 *web.xml* (Complete version for events examples) *(continued)*

```
  <!-- Tell the server to use form-based authentication. -->
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/admin/login.jsp</form-login-page>
      <form-error-page>/admin/login-error.jsp</form-error-page>
    </form-login-config>
  </login-config>
</web-app>
```