JSP Tag Libraries

GAL SHACHOR
ADAM CHACE
MAGNUS RYDIN

MANNING
Greenwich
(74° w. long.)

For electronic information and ordering of this and other Manning books, go to www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department
Manning Publications Co.
209 Bruce Park Drive Fax: (203) 661-9018
Greenwich, CT 06830 email: orders@manning.com

Library of Congress Cataloging-in-Publication Data
Shachor, Gal.
JSP tag libraries / Gal Shachor, Adam Chace, Magnus Rydin.
p. cm.
Includes bibliographical references and index.
ISBN 1-930110-09-X
1. Java (Computer program language) 2. JavaServer Pages. I. Chace, Adam. II. Rydin, Magnus. III. Title.

QA76.73.J38.S44 2001
005.2'762--dc21

2001030933

Manning Publications Co.
32 Lafayette Place
Greenwich, CT 06830

Copyeditors: Elizabeth Martin, Sharon Mullins
Typesetter: Tony Roberts
Cover designer: Leslie Haimes

Printed in the United States of America

1 2 3 4 5 6 7 8 9 10 – VHG – 05 04 03 02 01

12 Custom tags and J2EE

In this chapter

- **Introducing J2EE**
- **Introducing EJB**
- **Developing tags that work with J2EE resources**
- **Developing tags that work with EJB**

12.6 Using EJBs from within JSP

The advantages of J2EE for accessing powerful resources are becoming apparent, and fussing over the intricate details of how those resources are shared and implemented is left to the application server vendor. One of these powerful resources is the EJB layer. As explained earlier, EJBs are commonly used as a controller layer (a place to deposit business logic and processes) and as a way of persisting objects. The server vendor is responsible for ensuring that the objects are stored correctly, and that simultaneous access is handled properly. With EJBs, it is possible to write the entities that represent our system without having to design the underlying data structure for storage. We can use session beans to control access to these entities and offer utility operations that the client developers may utilize without having to be versed in their implementation.

12.6.1 Writing custom tags for EJB access

It should come as no surprise that our preferred method of accessing EJBs in JSP is through custom tags. To facilitate utilization of the EJB layer, we will create tags that assist us in accessing them. We do so by writing a tag library that will contain two tags:

- `<home>` allows instantiation of a home interface of a specific EJB.
- `<use>` grants the capacity to use the home interface to find existing entities or create new session or entity EJBs.

After an EJB has been retrieved, we treat it as any other bean with our tags. You can use any of the JavaBean tags we've developed in the book so far, such as `<show>`, to display EJB entity fields in a JSP page, and so forth.

As RMI over IIOP is used behind the scenes whenever an EJB method is called, all returned home or remote interfaces must be narrowed before being used. Both the `<home>` and `<use>` tags will therefore narrow the instances before adding them

to the given scope. We also need to extend our tags for iteration so that all remote interfaces in a collection are narrowed before usage.[2] We therefore develop a third EJB tag that will be an `<iterator>` for collections of remote interfaces.

12.6.2 Retrieving the EJB home interface

The `<home>` tag retrieves home interfaces for EJBs that are defined as EJB references in our web applications deployment descriptor. The tag then narrows and adds the home interface to the JSP `page` scope so that it can be used to our heart's desire. In listing 12.17, you can see the implementation of this tag.

Listing 12.17 The implementation of the home tag

```
package book.j2ee;

import javax.ejb.*;
import javax.naming.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import book.util.ExTagSupport;

public class HomeTag extends ExTagSupport
{
  protected String name;
  protected String type;

  public void setName(String name)
  {
    this.name = name;
  }

  public void setType(String type)
  {
    this.type = type;
  }

  public int doStartTag() throws JspException
  {
    try{
      InitialContext context = new InitialContext();     (1)
      ClassLoader classLoader=                                    (2)
              pageContext.getPage().getClass().getClassLoader();
      EJBHome home =                                              (3)
        (EJBHome)javax.rmi.PortableRemoteObject.narrow(
            context.lookup("java:comp/env/"+name),
            Class.forName(type, true, classLoader));
```

[2] The narrow method is used to check whether a certain remote or abstract interface can be cast to a given type.

```
      pageContext.setAttribute(this.getId(), home);
      return SKIP_BODY;
    }
    catch(NamingException e)
    {
      throw new JspTagException(
              "NamingException: " + e.getMessage());
    }
    catch(ClassNotFoundException e){
      throw new JspTagException(
              "ClassNotFoundException: " + e.getMessage());
    }
  }

  protected void clearServiceState()
 {
    name = null;
    type = null;
  }
}
```

❶ **Creates an `InitialContext`** The tag creates an `InitialContext` which it will use to look up the EJB.

❷ **Retrieves the page's `ClassLoader`** The tag retrieves the current page's `ClassLoader`, which will be used to instantiate a class of the type specified by the user.

❸ **Narrows the returned home to the specified class type** The tag uses the `InitialContext` to look up a JNDI path consisting of the root string `java:comp/env/` plus the JNDI name given by the user (such as `ejb/MyHome`). The returned class is then narrowed to the specific home type specified by the user. Finally, we add the home interface to the `page` scope.

HomeTEI

We must now define how this home interface is available throughout the page, which requires us to make it available as a scripting variable. As we have seen several times in this book, the way to specify a tag published scripting variable is to define a `TagExtraInfo` object for the tag. We do this in listing 12.18.

Listing 12.18 The implementation of HomeTEI

```
package book.j2ee;

import javax.ejb.*;
import javax.naming.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
```

```
public class HomeTEI extends TagExtraInfo
{
  public VariableInfo[] getVariableInfo(TagData data)
  {
    return new VariableInfo[]
    {
      new VariableInfo(
        data.getId(),
        data.getAttributeString("type"),
        true,
        VariableInfo.AT_BEGIN
      ),
    };
  }
}
```

Here we specify that a variable with the given ID and the given type will be added to the `page` scope from the start of this tag to the end of the page.

The HomeTag TLD

Now we need to write a tag library descriptor and the `HomeTag` will be complete. Listing 12.19 is the descriptor we will need.

Listing 12.19 The HomeTag TLD

```
<?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>ejb-tags</shortname>
  <uri>http://www.manning.com/jsptagsbook/ejb-taglib</uri>
  <info>EJB Taglib</info>
  <tag>
    <name>home</name>
    <tagclass>book.j2ee.HomeTag</tagclass>
    <teiclass>book.j2ee.HomeTEI</teiclass>
    <bodycontent>empty</bodycontent>
    <info>Adds a EJB Home interface to the page scope</info>
    <attribute>
      <name>id</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>name</name>
      <required>true</required>
```

```
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>type</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

In the tag library descriptor, we state that the tag library will contain a `<home>` tag that requires an ID, a name, and a type, which cannot be a runtime expression (so that it will be available in the translation phase).

HomeTag in action

Since the `CatalogueEntry` EJB has been given an EJB reference in an applications deployment configuration, we may access its home interface from JSP in the way described in listing 12.20.

Listing 12.20 Accessing the CatalogueEntry home interface from JSP

```
<%@ taglib uri="http://www.manning.com/jsptagsbook/ejb-taglib"
           prefix="ejb" %>
<HTML>
<HEAD>
  <TITLE>Accessing an EJB home interface</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
  <H1>Accessing an EJB home interface</H1>
  <HR>
  Retrieving the EJB home interface..<BR>
  <ejb:home id="home"
            type="book.ejb.catalogue.CatalogueEntryHome"
            name="ejb/catalogueEntry"/>
  The EJB home interface is retrieved.<BR>
  <HR>
</BODY>
</HTML>
```

❶

❶ **Adds the home interface to the `page` scope.**

In the page, the `<home>` tag is passed the ID to use when adding our home interface to the `page` scope. We also ascribe the tag with the type of class that we want returned when we look up our EJB home with the given name.

12.6.3 Using the EJB home interface

Now that the home interface of an EJB is retrievable, we may access its methods with the use of tags from earlier in the book. For example, we could use the `<iterate>` tag in a fashion as described in listing 12.21.

Listing 12.21 Using the EJB home interface

```
…
<ejb:home id="home"
          name="ejb/catalogueEntry"
          type="book.ejb.catalogue.CatalogueEntryHome"/>
<iter:iterate id="entry"
              type="book.ejb.catalogue.CatalogueEntry"
              object="<%=home.findByType(\"pda\").iterator()%>">
  <bean:show name="entry"
             property="serial"/><BR>
</iter:iterate>
…
```

In this listing, we let the `<iterate>` tag iterate over the collection of entries as returned from the finder method, introduce the entries using their remote interface, and display the serial number property for each `CatalogueEntry` of the type `pda`. As previously stated, executing this JSP on an application server that uses RMI over IIOP could throw an exception, as the remote interfaces that the `<iterate>` tag returns to the page have not been narrowed. Later we will solve this by extending the `<iterate>` tag further. Yet, the narrowing problem is not unique for the `<iterate>` tag and is going to appear for any remote interface returned by an EJB, so we need a tag that allows us to use the home interface by retrieving and narrowing a remote interface. We'll now build the `UseTag`, which will do just that.

UseTag

Our `UseTag` is somewhat analogous to the standard JSP `<jsp:useBean>` tag, but differs in that it lets us use and put into scope an EJB, instead of a standard JavaBean (listing 12.22).

Listing 12.22 The implementation of UseTag

```
package book.j2ee;

import javax.ejb.*;
import javax.naming.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import book.util.ExTagSupport;
```

```
public class UseTag extends ExTagSupport
{
  protected String type;
  protected EJBObject instance;

  public void setType(String type)
  {
    this.type = type;
  }

  public void setInstance(EJBObject instance)
  {
    this.instance = instance;
  }

  public int doStartTag() throws JspException
  {
    EJBObject object=null;
    try{
      ClassLoader classLoader=                                          ❶
              pageContext.getPage().getClass().getClassLoader();
      object = (EJBObject)javax.rmi.PortableRemoteObject.narrow(        ❷
              instance, Class.forName(type, true, classLoader));
    }
    catch(ClassNotFoundException e){
      throw new JspTagException(
              "ClassNotFoundException: " + e.getMessage());
    }
    pageContext.setAttribute(this.getId(),object);
    return SKIP_BODY;
  }

 protected void clearServiceState()
   {
    type = null;
    instance = null;
  }
}
```

❶ **Retrieves the page's `ClassLoader`.**

❷ **Narrows the instance to the specified class type.**

`UseTag` is given an EJB instance as a parameter through the `instance` attribute. It then adds the instance given to the `page` scope after narrowing it into an `EJBObject` that matches the type that was provided as attribute. The code is very similar to the code used to retrieve the home interface, the only major difference being that we needn't use lookup since the instance was given to us as a parameter to the tag. Next let's look at the TEI given in listing 12.23.

UseTEI

`UseTag` will put an instance of any EJB it is used with into `page` scope. Here's a look at how this is accomplished.

Listing 12.23 The implementation of UseTEI

```
package book.j2ee;

import javax.ejb.*;
import javax.naming.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class UseTEI extends TagExtraInfo
{
  public VariableInfo[] getVariableInfo(TagData data)
  {
    return new VariableInfo[]
    {
      new VariableInfo(
        data.getId(),                                    (1)
        (String)data.getAttribute("type"),               (2)
        true,
        VariableInfo.AT_BEGIN
      ),
    };
  }
}
```

(1) Publishes the scripting variable with the `id` that is specified by tag attribute.

(2) Publishes a variable of a type that is also specified by a tag attribute.

In listing 12.23, we specify that a variable named with the given ID and the given type will be added to the `page` scope from the start of this tag to the end of the page, just as in `HomeTEI`.

Updating the TLD to include UseTag

We should now update the EJB tag library descriptor by adding the tag descriptor as in listing 12.24.

Listing 12.24 Adding the tag to the descriptor

```
…
<tag>
    <name>use</name>
    <tagclass>book.j2ee.UseTag</tagclass>
    <teiclass>book.j2ee.UseTEI</teiclass>
```

```
    <bodycontent>empty</bodycontent>
    <info>Adds a EJB Remote Interface to the page scope</info>
    <attribute>
      <name>id</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>instance</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>type</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
  </tag>
…
```

In the addition to the tag library descriptor, we state that the library will also contain the `<use>` tag, which requires an ID, an instance, and a type as parameters. Of these parameters, only instance can be a runtime expression.

UseTag in action

Because the `CatalogueEntry` EJB described in the beginning of this chapter has received an EJB reference in an application's deployment configuration, we may use the EJB's home interface from JSP in the way described in listing 12.25.

Listing 12.25 Using the CatalogueEntry home interface in JSP

```
<%@ taglib uri="http://www.manning.com/jsptagsbook/ejb-taglib"
           prefix="ejb" %>
<HTML>
<HEAD>
  <TITLE>Accessing an EJB home interface</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
  <H1>Accessing an EJB home interface</H1>
  <HR>
  Retrieving the EJB home interface..<BR>
  <ejb:home id="home"                                              (1)
            type="book.ejb.catalogue.CatalogueEntryHome"
            name="ejb/catalogueEntry"/>
  The EJB home interface is retrieved.<BR>
  Finding the Entry with serial ABC123..<BR>
```

```
  <ejb:use id="entry"                                                    ❷
           type="book.ejb.catalogue.CatalogueEntry"
           instance="<%=home.findByPrimaryKey(\"ABC123\")%>"/>
  The entry was found.<BR>
  <HR>
</BODY>
</HTML>
```

❶ **Defines an instance of `CatalogueEntryHome` into the JSP scope** Walking through the JSP, you'll see that we add the home interface for the `CatalogueEntry` EJB to the `page` scope with an ID of `home`.

❷ **Uses the home interface to find a `CatalogueEntry` and to return its remote interface** Next, we use the `<use>` tag to narrow and add the remote interface of `CatalogueEntry` EJB to the `page` scope, by asking the home interface to find the `CatalogueEntry` that has a serial of `"ABC123"`. If such an entry is found, we can refer to `"entry"` as a scripting variable through the remainder of the JSP, calling methods of the EJB as desired.

IterateEJBTag

In order to iterate through collections of remote interfaces returned by methods called on the EJB home interfaces, we need a tag that works similarly to the `<iterate>` tag of chapter 10. The only difference will be that this tag will try to narrow the remote interface to the specified type before adding it to the given scope (see listing 12.26). Note that there is an alternative to the updated `<iterate>` tag. We could use the `<use>` tag to narrow the iterator exported from the `<iterate>` tag prior to using it. However this forces the JSP coder to develop insight into the intrinsics of EJB, something we chose to avoid.

Listing 12.26 The implementation of IterateEJBTag

```
package book.j2ee;

import java.util.Enumeration;
import java.util.Iterator;
import javax.ejb.*;
import javax.servlet.jsp.JspException;
import book.iteration.*;

public class IterateEJBTag extends IterateTag
{

  protected String type;

  public void setType(String type)    ❶
```

```
  {
    this.type=type;
  }

  public String getType()
  {
    return type;
  }

  protected void exportVariables() throws JspException
  {
    try{
      current = elementsList.getNext();
      ClassLoader classLoader=
              pageContext.getPage().getClass().getClassLoader();
      EJBObject object =
         (EJBObject)javax.rmi.PortableRemoteObject.narrow(
         current,Class.forName(getType(),
         true, classLoader));
      pageContext.setAttribute(id, object);
    }catch(ClassNotFoundException cnfe){
      throw new JspException(cnfe.getMessage());
    }
  }
}
```

❷

❸

❶ **Overriding `setType()` to save the type** The tag overrides `setType()` of the `IterateTag` class and adds a method for reading this value. We will need the type's value in order to narrow the iterator object into a concrete type in `exportVariables()`.

❷ **Retrieves the page's `ClassLoader`** ❸ **Narrows the next remote interface to the specified class type** The tag overrides `exportVariables()` of the `IterationTagSupport` class. In that method, the tag first retrieves the current page's `ClassLoader`. The tag then narrows the next item in the collection into the remote interface type specified by the user. Finally, we add the remote interface to the `page` scope.

As the new tag can use the same TEI as the original `<iterate>`, we needn't create a TEI for this tag.

The IterateEJBTag TLD

We now need to update the EJB TLD by adding the tag descriptor as in listing 12.27.

Listing 12.27 Adding the tag to the descriptor

```
…
<tag>
  <name>iterate</name>
  <tagclass>book.j2ee.IterateEJBTag</tagclass>
  <teiclass>book.iteration.IterateTagExtraInfo</teiclass>
  <bodycontent>JSP</bodycontent>
  <info>
    Iterate over an Object. The object can be an array, an Iterator
    or an Enumeration of Remote interfaces.
  </info>
  <attribute>
    <name>id</name>
    <required>true</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
    <name>type</name>
    <required>true</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
    <name>object</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>name</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
    <name>scope</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
    <name>index</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>property</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
</tag>
…
```

In this addition to the tag library descriptor, we state that the library will also contain the `<iterate>` tag, which requires the same parameters as the original `<iterate>` tag, with the difference that the attribute `type` is now required.

IterateEJBTag in action

Let's change the example given in listing 12.21 so that it takes advantage of our new tag as described in listing 12.28.

Listing 12.28 Using the EJB home interface with the EJB iterate tag

```
…
<ejb:home id="home"
          name="ejb/catalogueEntry"
          type="book.ejb.catalogue.CatalogueEntryHome"/>
<ejb:iterate id="entry"
             type="book.ejb.catalogue.CatalogueEntry"
             object="<%=home.findByType(\"pda\").iterator()%>">
  <bean:show name="entry"
             property="serial"/><BR>
</ejb:iterate>
…
```

Here, we first use the `<home>` tag to retrieve a home interface. We then use the new `<iterate>` tag from the EJB tag library to iterate through the collection returned by the home interface's `findByType()` method. Every remote interface iterated, will be narrowed before it is added to the `page` scope.

Final thoughts

From the listings it is clear that using tags to access the EJB layer is easily done and allows us to write less code than if we were to use a servlet, or, if we were unconcerned with using scriptlets. The complexity of the EJB as seen in this chapter is virtually the tip of the EJB iceberg. If you are looking for a pervasive way to incorporate EJBs into your JSP, tags are probably your only alternative.

A common practice with EJBs is to write session EJBs that provide access methods to the various entities (and entity EJBs) that comprise your applications. This approach makes it easy to add or change the sanctioned methods of retrieving these entities. Such session EJBs usually include utility methods that perform functions, such as returning all entities as a `Collection`, making it even easier for the developers to divide the application into separate layers and pare down still more of the code required for the presentation.

Note that the EJB tags we present could be even further optimized. For instance, there are utility methods in the EJB home interface that find out the type

of the named EJB's home and remote interfaces. With a little knowledge of EJB, these tags can be enhanced so that the user has only to specify the name of the EJB, and the tags will themselves find out what types to use for them. We have not taken on this topic, as it would divert attention from tags to advanced EJB usage, which is not the scope of this book.

JSP tag libraries provide a simple, elegant way of embedding dynamic server-side request handling into a JSP page. WebLogic Server provides a tag library with custom tags that you may use in your JSP pages; this library defines the cache, repeat, and process tags. Other tags also are supplied, though these are not discussed, as much of their functionality can now be found in more standard tag library implementations, such as the Java Standard Template Library (JSTL). JSP Standard Tag Library (JSTL): Folks over at Oracle have developed a common set of tags you can use in your JSP pages. JSP Custom Tags: Here you can write your own custom code and implement that as JSP tags. Iâ€™ll show you how to write your custom tags later. Sample Use Case. In order to better understand JSP tags, letâ€™s consider this problem. Your boss comes up to you and asks you to display the price of a specific stock on your JSP pages. Now you can go about solving this problem in two different ways.