CHAPTER 9

Configuring Struts

Up until now, I've presented portions of Struts along the lines of the MVC design pattern. In Labs 6, 7, and 8, you've implemented these portions yourself for the LILLDEP webapp. What's missing is the way Struts ties together the various Model-View-Controller portions.

Struts uses a single configuration file called `struts-config.xml` to store this information. The Struts distribution comes bundled with samples of this file, which you can copy and amend for your own use.

The Structure of struts-config.xml

Unsurprisingly, `struts-config.xml` holds data in XML format. There are several sections, each of which handles configuration for specific portions of Struts:

- **Form bean declarations**: This is where you map your `ActionForm` subclass to a name. You use this name as an alias for your `ActionForm` throughout the rest of the `struts-config.xml` file, and even on your JSP pages.
- **Global exceptions**: This section defines handlers for exceptions thrown during processing.
- **Global forwards**: This section maps a page on your webapp to a name. You can use this name to refer to the actual page. This avoids hardcoding URLs on your web pages.
- **Form handlers**: Remember the form handlers I mentioned in the previous chapter? This is where you declare them. Form handlers are also known as "action mappings."
- **Controller declarations**: This section configures Struts internals. Rarely used in practical situations.
- **Message resources**: This section tells Struts where to find your properties files, which contain prompts and error messages.

- **Plug-in declarations**: This is where you declare extensions to Struts. You will use two important plug-ins: **Tiles** and **Validator**.

These sections must be placed in this order. Not all sections need be present. For example, you might not want to take advantage of the global exception handling facility. In this case, your `struts-config.xml` file would not contain the global exceptions section.

■ **Note** Earlier versions of Struts had a "data sources" section before the form bean declarations. This data sources section allowed you to preconfigure JDBC data sources for your webapp. This section has since been deprecated.

Among the seven sections, the form bean, form handler (or action mapping), and message resources sections are the most important, and you must master them before you can use Struts.

Before I introduce the various sections, let's take a look a simple `struts-config.xml` file, the one for the Registration webapp.

Configuring the Registration Webapp

The `struts-config.xml` file (shown in Listing 9-1) for the Registration webapp requires just the form bean, global exceptions, global forwards, form handler (or action mapping), and message resources sections.

Listing 9-1. *struts-config.xml for the Registration Webapp*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
   "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>

   <form-beans>
     <form-bean
        name="RegistrationForm"
        type="net.thinksquared.registration.struts.RegistrationForm"/>
   </form-beans>
```

```
  <global-exceptions>
    <exception key="reg.error.io-unknown"
               type="java.io.IOException"
               handler="net.thinksquared.registration.ErrorHandler"/>

    <exception key="reg.error.unknown"
               type="java.lang.Exception"
               path="/errors.jsp" />
  </global-exceptions>

  <global-forwards>
    <forward name="ioError" path="/errors.jsp"/>
  </global-forwards>

  <action-mappings>
    <action
       path="/Registration"
       type="net.thinksquared.registration.struts.RegistrationAction"
       name="RegistrationForm"
       scope="request"
       validate="true"
       input="/Registration.jsp">

        <forward name="success" path="/Success.jsp"/>

    </action>
  </action-mappings>

  <message-resources parameter="Application"/>

</struts-config>
```

The three main sections should be immediately obvious in Listing 9-1: the form beans section is enclosed by the `<form-beans>` tag, the form handlers section by the `<action-mappings>` tag, and the single message resource section by the `<message-resources>` tag. I'll describe these sections in detail next.

Declaring Form Beans

The form bean section is where you give your `ActionForm` subclasses names, which can be used both in `struts-config.xml` and on your JSP pages.

The declaration consists of a single enclosing <form-beans> tag (note the plural), and one or more <form-bean> (note the singular) tags, as shown in Listing 9-2.

Listing 9-2. *The Form Beans Section*

```
<form-beans>
  <form-bean
       name="RegistrationForm"
       type="net.thinksquared.registration.struts.RegistrationForm"/>
</form-beans>
```

A <form-bean> tag has two attributes:

- name: The name attribute is a unique label for an ActionForm subclass. In Listing 9-2, the name is RegistrationForm, but it could be anything you like, as long as it is unique among other form bean declarations and qualifies as an XML attribute value.
- type: The type attribute is the fully qualified class name for that ActionForm subclass.

With this declaration, you can use the name RegistrationForm to refer to the ActionForm subclass, net.thinksquared.registration.struts.RegistrationForm.

Declaring Global Exceptions

Global exceptions allow you to catch uncaught runtime exceptions that occur in your Action subclasses, displaying them with a custom error message. This certainly lends a little more polish to your application, compared to the default Tomcat error page. Listing 9-3 shows two ways you can define an exception handler.

Listing 9-3. *Declaring Global Exception Handlers*

```
<global-exceptions>
  <exception key="reg.error.io-unknown"
             type="java.io.IOException"
             handler="net.thinksquared.registration.IOErrorHandler"/>

  <exception key="reg.error.unknown"
             type="java.lang.Exception"
             path="/errors.jsp"/>

</global-exceptions>
```

The <global-exceptions> tag contains global exception handlers, each represented by an <exception> tag. Each <exception> tag has two required attributes:

- key: An error key. When an exception handler is fired, an ActionMessage with this key is created and put on the request. This error message gets pasted on the JSP containing an <html:errors> tag that is finally displayed. Note that key is a required attribute, which you have to specify even if you don't use it.

- type: Describes the type of error that is caught. Struts will first try to find the error class that matches the declared types. If none matches exactly, then Struts goes up that error's superclass tree, until it finds a match with a declared exception.

There are two ways you can handle the caught exceptions: using a path attribute to point to a JSP page that displays the error message, or using a handler attribute, which is a fully qualified class name of your ExceptionHandler subclass—that is, your handler subclasses:

```
org.apache.struts.action.ExceptionHandler
```

You only have to override the execute() function:

```
public ActionForward execute(Exception ex,
                             ExceptionConfig ae,
                             ActionMapping mapping,
                             ActionForm form,
                             HTTPRequestServlet request,
                             HTTPResponseServlet response)
```

As you can see, this is nearly identical to Action.execute(), apart from the extra first two arguments. You should declare the "next" page ins a <global-forwards> section (see the next section), so that mapping can resolve it.

In most instances, using a custom ExceptionHandler subclass is overkill unless you have a special requirement to do so.

Declaring Global Forwards

You use global forwards to define forwarding paths accessible to all Actions or ExceptionHandlers. In fact, all such paths will be accessible as long as you have an ActionMapping instance initialized by Struts. This is certainly the case for the execute() functions in Action or ExceptionHandler.

Global forwards are defined within an enclosing <global-forwards> tag, within which you may place as many <forward> tags as you wish, as shown in Listing 9-4.

Listing 9-4. *Declaring a Global Forward*

```
<global-forwards>
   <forward name="ioError" path="/errors.jsp"/>
</global-forwards>
```

Each <forward> tag has two attributes:

- name: A globally unique name for this forward.
- path: The path to the JSP or form handler to which this forward refers. In both cases, the paths must begin with a slash (/).

In Listing 9-4, a forward named ioError is declared, which refers to the path errors.jsp. You can access this path from the mapping object available in execute():

```
mapping.findForward("ioError")
```

Declaring Form Handlers

Form handlers are defined within a single <action-mappings> enclosing tag, as shown in Listing 9-5.

Listing 9-5. *The Form Handler Declaration*

```
<action-mappings>
  <action
    path="/Registration"
    type="net.thinksquared.registration.struts.RegistrationAction"
    name="RegistrationForm"
    scope="request"
    validate="true"
    input="/Registration.jsp">

     <forward name="success" path="/Success.jsp"/>

  </action>
</action-mappings>
```

The <action-mappings> tag acts as a container for each form handler, described by an <action> tag. The <action> tag contains a few attributes that configure the handler:

- path: Describes the name of the form handler.
- type: The fully qualified class name of the Action subclass that handles business logic.
- name: The name of the form bean associated with this handler.
- validate: Tells Struts whether it should perform simple validation on the form data.
- scope: Sets the scope of the form data. Only request or session scopes are allowed.
- input: The relative path of the page that forms the input of this page.

The path attribute must start with a slash. This has a meaning I will cover in a later chapter. You might want to recall how form handlers are used on your JSPs, as the action attribute of the <html:form> tag:

```
<html:form action="Registration.do" ...>
```

In this snippet, the form handler is Registration, and this corresponds to the path attribute in the form handler declaration of Listing 9-5. The .do extension tells the servlet container that this is a Struts form handler. This extension is the default, and is defined in the web.xml file distributed with Struts.

Just as in the form bean declaration, the type attribute is the fully qualified name of your Action subclass that will process business logic. In this case, it is RegistrationAction.

The name attribute is the name of the form bean associated with the handler. In this case, it is RegistrationForm, which is an obvious alias to the RegistrationForm subclass.

The validate : attribute, which can either be true or false, tells Struts whether the validate() method on the form bean's ActionForm subclass should be called. In other words, Struts performs simple validation only when validate="true".

The scope : attribute, which can either be request or session, tells Struts in which scope to put the form bean that contains the form data.

Lastly, the input attribute is just the path to the input page. This is how mapping.getInputForward() (see Chapter 7) knows what the input page was. In Listing 9-5, the input page is Registration.jsp.

Note that the path to the input page must begin with a slash, which denotes the "root" directory of the webapp. This corresponds to the root directory on the WAR file you create to deploy the webapp. Or, as another way of looking at it, when the webapp is installed, the root directory is just the \webapps\<app name>\ directory hosted under the servlet container.

For example, if the Registration webapp were deployed in a WAR file called registration.war, then the root directory on the servlet container would be \webapps\registration\.

One last thing about the `input` attribute. If I had put in a global forwards section, I could have declared a global forward for `Registration.jsp`, and used the name of the global forward as the value of `input`.

Forwards

Each `<action>` tag can contain zero or more `<forward>` tags. These forwards differ from the `<forward>`s declared in the global forwards section in that they are visible only to the enclosing form handler. Forwards declared the global forwards section are visible everywhere.

`<forward>`s represent the possible "next" pages for that form handler. Like the `<forward>`s in the global forwards section, each `<forward>` tag has two attributes:

- `name`: A name for the "next" page
- `path`: The actual page's path, relative to the webapp's root directory

The `name` attribute identifies this "next" page in the `Action` subclass associated with the form handler. This is how `mapping.findForward()` works (see Listing 7-2):

```
return mapping.findForward("success");
```

The label `success` is just the value of the `name` attribute.

The `path` must start with a slash if it's a true path. If you had declared the page as a global forward, then you could use the name of the global forward instead. In this case, you'd have no initial slash, because the `path` value is just a label, not a true path.

Controller Declaration

The controller section is probably the least used among the seven sections. It is used to manually override some default Struts settings. I will only mention a few useful ones here:

- `maxFileSize`: Specifies the upper size limit on uploaded files. You can use a number followed by `K`, `M`, or `G` to indicate a size in kilobytes, megabytes, or gigabytes, respectively. For example, `maxFileSize="2M"` limits the file size to 2MB. We will deal with file uploading in Chapter 11.
- `nocache`: Tells Struts whether it should cache content. Setting `nocache="true"` disables content caching.
- `contentType`: Specifies the default content type of pages. For example, if you're delivering XML pages by default, set `contentType="text/xml"`.

- The following snippet shows how you'd declare a controller section:

```
<controller maxFileSize="1.618M" contentType="text/svg" />
```

This sets the maximum uploadable file size to a golden 1.618MB, and the default content type to SVG.

Message Resources

The message resources section declares the location of your properties file that stores the key/value pairs of your application. Recall that the contents of this properties file was implicitly used to create error messages:

```
errors.add("userid",new ActionMessage("reg.error.userid.exists"));
```

and prompts on the JSP page:

```
<bean:message key="registration.jsp.prompt.userid"/>
```

Unlike the previous two sections, this one does *not* have an enclosing tag.

```
<message-resources parameter="Application"/>
```

The main attribute of the `<message-resources>` tag is the `parameter` attribute, which gives the location of the properties file for your application relative to the `\WEB-INF\classes\` directory of your webapp. So, in the previous declaration, the message resource file is

```
\WEB-INF\classes\Application.properties
```

Note that in the declaration, the `.properties` extension is implied. If you had placed the properties file further up the package, say, in

```
\WEB-INF\classes\net\thinksquared\registration\struts\resources\
```

the `parameter` attribute value would be

```
net.thinksquared.registration.struts.resources.Application
```

Declaring Plug-ins

Plug-ins are custom extensions to Struts. An example is the Tiles framework, which was developed independently of Struts. In earlier (pre-1.2) versions of Struts, Tiles had to be downloaded separately. Although it is now part of the 1.2 distribution, its earlier independent existence is still apparent because you have to declare a plug-in section for Tiles in order for you to use it in Struts.

The plug-in section tells Struts what plug-ins to initialize and what data they require (usually paths to various configuration files required by the plug-in). Listing 9-6 shows a typical plug-in declaration.

Listing 9-6. *A Possible Plug-in Declaration for Tiles*

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
    <set-property property="definitions-config"
                    value="/WEB-INF/tiles-defs.xml"/>
</plug-in>
```

Each `<plug-in>` tag declares a single plug-in. There is no limit to how many of these you might have in your `struts-config.xml` file. The `className` attribute is required, and it points to the plug-in class that Struts calls. This class is unique to each plug-in.

Each `<plug-in>` tag may contain zero or more `<set-property>` tags to set the various properties needed by the plug-in. There are only two attributes here:

- `property`: Defines the property being set
- `value`: Specifies the corresponding value

Needless to say, each plug-in will have to be configured differently, and you'll have to get the details from the plug-in's manual.

Lab 9a: Configuring LILLDEP

In this lab, you will configure LILLDEP and deploy it on Tomcat. Make the following changes to `\web\WEB-INF\struts-config.xml` for LILLDEP:

1. Declare a form bean called `ContactFormBean` for the `ActionForm` subclass you implemented in Lab 6.
2. Declare a form handler for the form in `full.jsp`. What should the name of the form handler be?
3. The `Action` subclass for the form handler should be the one you implemented in Lab 7. The form bean used should be the one in step 1.
4. Set the `scope` attribute to `request`.
5. Give a value for the form handler's `input` attribute. Where is this used? What happens if you omit this attribute from the form handler's declaration?

6. Create a forward for this form handler to the page `full.jsp`. What should be the `name` attribute of this forward? (Hint: Check the code for the `Action` subclass.) What happens if you omit this forward declaration?

7. Run `compile.bat` to produce the WAR file, then deploy and test your application. Figure 9-1 shows what the LILLDEP start page should look like. You should be able to key data into LILLDEP. Test out the simple validations you wrote in Lab 6.

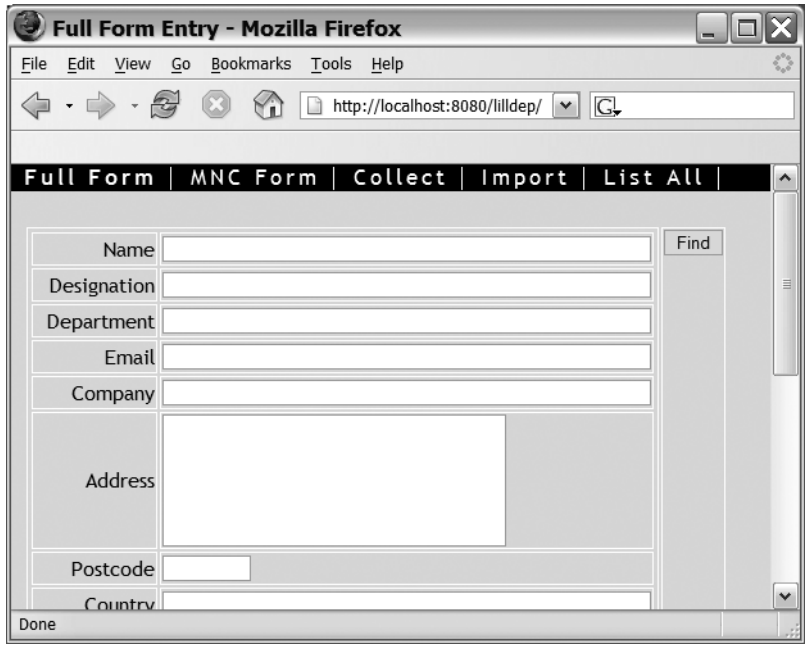

Figure 9-1. *The LILLDEP start page*

Code Reuse

One reason why form handlers are declared in the `struts-config.xml` file instead of being hardcoded is because this promotes code reuse.

In Lab 9a, you implemented a form handler specific to the `full.jsp` page. But what if you wanted a different page to submit data to this form handler? You obviously can't now, because this form handler's only "next" page is `full.jsp`.

One way to do it would be to declare a new `<forward>`, and amend `ContactAction` to dispatch to the forward corresponding to the input page. This "solution" is hardly a good one because you'd have to amend and recompile `ContactAction` each time you wanted to add or remove a page.

The solution Struts offers is much better. The idea is to declare a *new* form handler in `struts-config.xml` to handle input from a different page. You can then reuse form beans and `Action` subclasses in this new form handler, but use a different `<forward>`.

The next lab session shows you how to do this.

Lab 9b: The MNC Page

In LILLDEP, users frequently have to fill in contacts from multinational corporations (MNCs). For these companies, users want a separate form, containing just the following seven fields:

- Name
- Designation
- Department
- Email
- Company
- Address
- Postcode

The Classification field should *automatically* be set to the string value `mnc`. You can use the `<html:hidden>` tag

```
<html:hidden property="myFormProperty" value="myFixedValue"/>
```

to automatically set the Classification field. The other fields should be left blank. Figure 9-2 shows how the MNC page appears to the user.

1. Complete the implementation of `mnc.jsp`. The form handler should be called `ContactFormHandlerMNC`.
2. Add a new form handler to `struts-config.xml` to accept data from `mnc.jsp`. The (single) forward should point back to `mnc.jsp`. What should the name of the forward be? (Hint: `ContactAction` knows only one forward label.)
3. Run `compile.bat` to produce the WAR file.
4. Stop Tomcat, then delete the `\webapps\lilldep\` folder. Only then re-deploy the LILLDEP webapp.

Test out your application as you did in Lab 9a.

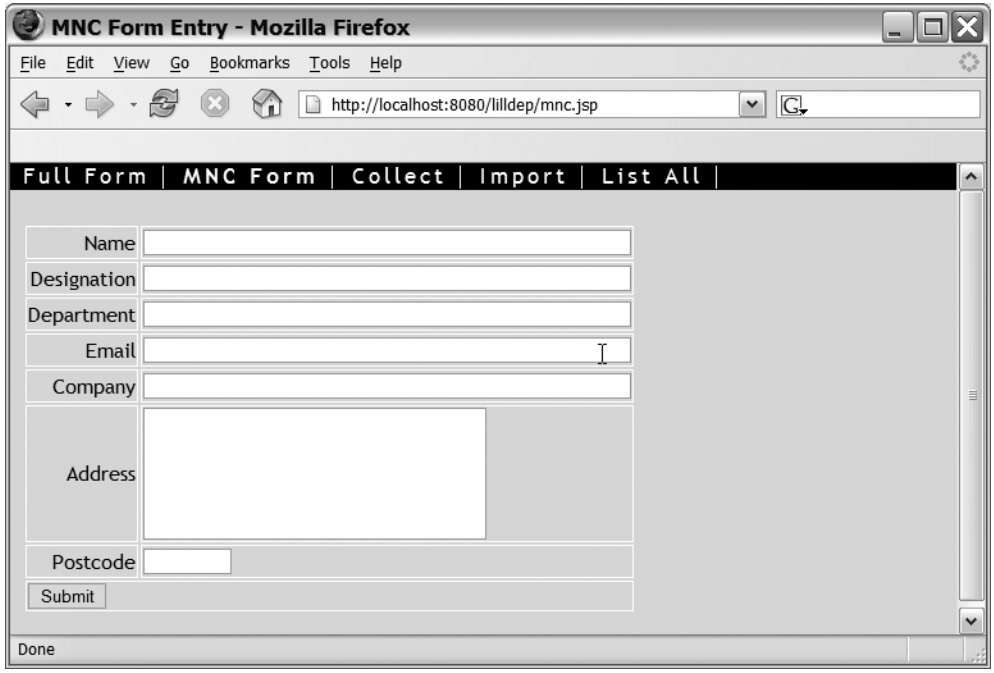

Figure 9-2. *The MNC data entry page*

Summary

In this chapter, you've learned the basics of configuring Struts:

- The `struts-config.xml` file ties together the View and Controller sections of your webapp.
- `struts-config.xml` is deployed in the `\WEB-INF\` directory of your webapp.
- `ActionForm` subclasses are exposed to your webapp in the form bean declarations.
- A form handler declaration ties together form beans, an `Action` subclass, and one or more "next" pages.
- The properties files for your application are declared in the message resources section of `struts-config.xml`.