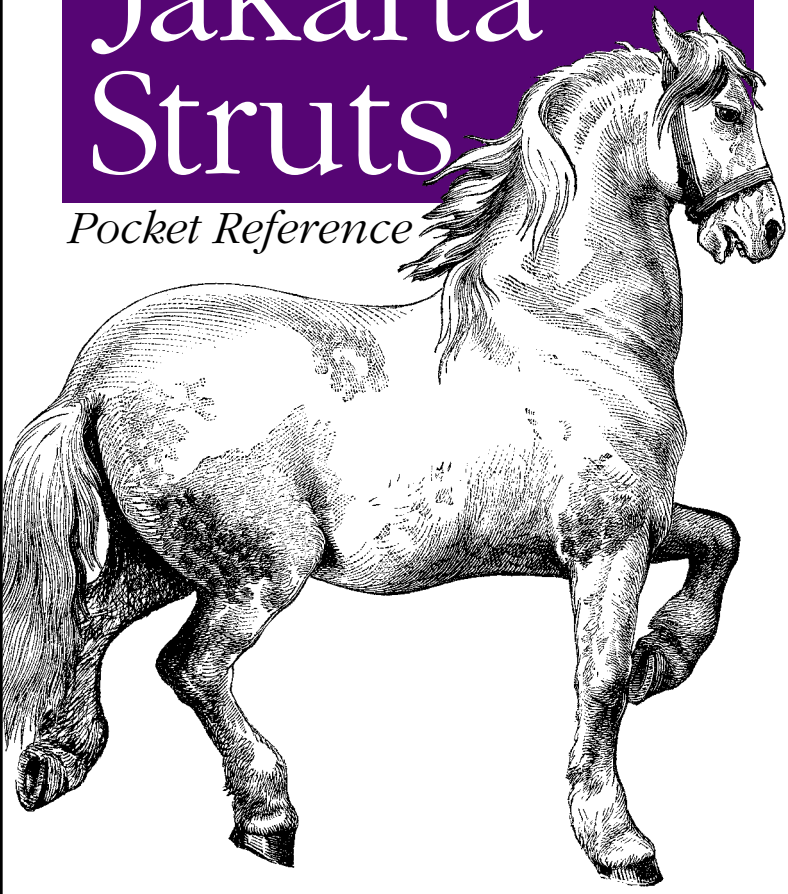
Building Web Applications with Servlets & JSPs

Jakarta Struts

Pocket Reference



O'REILLY®

Chuck Cavaness & Brian Keeton

Jakarta Struts

Pocket Reference

Jakarta Struts
Pocket Reference

Chuck Cavaness and Brian Keeton

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

PART I

Configuring Struts

For every Struts application, there are at least two configuration files that must be present: the web application deployment descriptor and a Struts configuration file. This part discusses each one in turn.

Configuring web.xml

Each Struts application must include a web application deployment descriptor named *web.xml*, which must be placed in the *WEB-INF* directory. The web container reads and parses the descriptor file at startup and uses the settings to configure the runtime environment for the installed web application.

Although there are many available configuration settings that can affect the container's runtime environment, it's not necessary to configure all of the settings for a Struts application. In many cases, the absence of a setting or the container's default values will be sufficient. Only those settings that pertain to Struts applications will be examined here.

For Struts applications, the following configuration settings are typically configured within the *web.xml* file:

- Struts `ActionServlet` settings
- Initialization parameters
- `load-on-startup` settings
- Welcome file list
- Tag libraries mappings

Configuring the Struts ActionServlet

The Struts `ActionServlet` is designed to receive all incoming requests for the web application. Two steps are necessary when configuring the `ActionServlet` in the *web.xml* file. The first step is to use the `servlet` element to configure the fully qualified Java class name of the `ActionServlet`:

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet
  </servlet-class>
 </servlet>
</web-app>
```

In this *web.xml* example, the `servlet` element declares two child elements, `servlet-name` and `servlet-class`. The `servlet-class` element specifies the fully qualified class that will function as the front controller for the Struts application. The Java class specified must be a descendant of the `org.apache.struts.action.ActionServlet`. If you don't have a need for specialized behavior, you can safely use the default controller as shown in the previous *web.xml* fragment.

The `servlet-name` element acts as a logical name for the `ActionServlet`. It is used in other elements within the deployment descriptor. You can specify whatever value you like here, as long as it adheres to the Servlet Specification naming guidelines.

The second step required to configure the `ActionServlet` is to inform the web container which URL requests should be directed to the `ActionServlet` controller. This is done using the `servlet-mapping` element:

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>
   org.apache.struts.action.ActionServlet
  </servlet-class>
 </servlet>
```

```
  <servlet-mapping>
   <servlet-name>storefront</servlet-name>
   <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

In the example `servlet` and `servlet-mapping` elements shown, any request containing a URL that matches **.do* would be processed by the servlet named *storefront*.

Declaring Initialization Parameters

Within the `servlet` element, you can specify multiple `init-param` elements. The parameters listed in Table 1 are used by the `ActionServlet` to configure the Struts application runtime environment.

Table 1. ActionServlet initialization parameters

Name	Purpose
`config`	Comma-separated list of context-relative path(s) to the XML resource(s) containing the configuration information for the default module. The default value is */WEB-INF/struts-config.xml*.
`config/${module}`	Comma-separated list of context-relative path(s) to the XML resource(s) containing the configuration information for the module that will use the specified prefix (`/${module}`). This can be repeated as many times as required for multiple modules.
`convertNull`	Forces simulation of the Struts 1.0 behavior when populating forms. If set to `true`, the numeric Java wrapper class types such as `java.lang.Integer` default to `null` (rather than 0). The default value is `false`.
`rulesets`	Comma-delimited list of fully qualified class names of additional `org.apache.commons.digester.RuleSet` instances that should be added to the `Digester` and that will process the Struts configuration files. By default, only the `RuleSet` for the standard configuration elements is loaded.
`validating`	Specifies whether a validating XML parser should be used to process the configuration file. The default value is `true`.

To configure one of the initialization parameters, an `init-param` element is added within the `servlet` element as shown here:

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet
  </servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
 </servlet>

 <servlet-mapping>
  <servlet-name>storefront</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
</web-app>
```

You can add as many `init-param` elements as the application requires.

Using the load-on-startup Setting

The `load-on-startup` element directs the web container to instantiate an instance of the specified `servlet-class` and invoke the `init( )` method.

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name>storefront</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
</web-app>
```

The integer value specified in the `load-on-startup` element tells the container the order in which the servlets should be called, in case there are more than one specified in the *web.xml* file. If the value is negative or not present, the container is free to load the servlets in any order.

WARNING

For Struts applications, it's essential that you include the `load-on-startup` element to ensure the proper initialization of resources.

Setting Up the Welcome File List

The `welcome-file-list` element specifies a starting page for the web application other than the default for the container. For example, if you want the page *index.jsp* to be executed when a client enters the root URL for the web application, add the following XML element:

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name>storefront</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
</web-app>
```

Since all requests should go through the Struts controller (in order for the correct module to be selected), it's sometimes necessary to use a combination of the `welcome-file-list` and

the forward JSP tag to invoke an Action using the default startup page. For example, the *index.jsp* page included in the previous welcome-file element might look like:

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
 <body>
  <logic:forward name="welcome"/>
 </body>
</html>
```

In this example, when the container invokes the *index.jsp* page, the Action named welcome will be invoked.

Configuring the Tag Libraries

The Struts framework includes several JSP tag libraries that can be used within your JSPs. In order for the container to find the descriptor files for a tag library, include a taglib element in the *web.xml* file for each library:

```
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name>storefront</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

 <taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld
  </taglib-location>
 </taglib>
```

```
 <taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld
  </taglib-location>
 </taglib>

 <taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld
  </taglib-location>
 </taglib>
</web-app>
```

The Struts Configuration Files

The Struts framework uses an XML-based configuration file to declaratively configure all aspects of the Struts application. By default, this file is named *struts-config.xml*, although you can call it whatever you like. The file is normally placed within the *WEB-INF* directory. The location can actually be specified within the *web.xml* file using the `config` initialization parameter.

Using Multiple Struts Configuration Files

Each Struts application can have one or more Struts configuration files per module. For each module, the framework parses the separate configuration files, merges all of the configuration settings in memory, and presents them as a single set of instructions for the application. The configuration settings for each module are kept separate from one another.

To configure multiple Struts configuration files for a single module, include the filenames (separated by a comma) within the `config` initialization parameter element. For example, the following XML fragment illustrates adding two different configuration files for the default module:

```
<init-param>
 <param-name>config</param-name>
 <param-value>
   /WEB-INF/struts-config.xml,/WEB-INF/struts-config2.xml
 </param-value>
</init-param>
```

The data-sources Element

The `data-sources` element allows you to set up a datasource that you can utilize from within the Struts framework. The `data-sources` element contains zero or more `data-source` elements as shown here:

```
<!ELEMENT data-sources (data-source*)>
```

Table 2 lists the `data-source` element's attributes.

Table 2. The data-source element's attributes

Name	Description
`className`	The implementation class of the configuration bean that will hold the datasource information. If specified, it must be a descendant of `org.apache.struts.config.DataSourceConfig`, which is the default value.
`key`	The servlet context attribute under which this datasource object will be stored. The default value is `Globals.DATA_SOURCE_KEY`.
`type`	The fully qualified Java class name of the datasource implementation class. The class specified here must implement the `javax.sql.DataSource` interface and be configurable from JavaBeans properties. The default value is `org.apache.struts.util.GenericDataSource`.

The `data-source` element allows for multiple `set-property` elements to be specified:

```
<!ELEMENT data-source (set-property*)>
```

The `set-property` element configures properties that are specific to your datasource implementation.

NOTE

The `set-property` element defines three attributes: the `id` attribute (which is seldom used), the `property` attribute, and the `value` attribute. The `property` attribute is the name of the JavaBeans property whose setter method will be called. The `value` attribute is a string representing the value that will be passed to the setter method after proper conversion. Although the `set-property` element is used within many configuration elements, it is only explained here since the other uses are identical.

The following `data-sources` example illustrates how to set up a datasource that utilizes an Oracle database:

```
<data-sources>
 <data-source>
  <set-property property="autoCommit" value="false"/>
  <set-property property="description"
    value="Oracle Datasource"/>
  <set-property
    property="driverClass"
    value="oracle.jdbc.driver.OracleDriver"/>
  <set-property property="maxCount" value="5"/>
  <set-property property="minCount" value="1"/>
  <set-property property="user" value="scott"/>
  <set-property property="password" value="tiger"/>
  <set-property
    property="url"
    value="jdbc:oracle:thin:@localhost:1521:DEV"/>
 </data-source>
</data-sources>
```

By default, the Struts framework will utilize the DBCP component from the Commons project. By using the `type` attribute, you can substitute other datasource implementations. For example, the following `data-source` fragment shows how to use the datasource implementation included with the Oracle JDBC driver:

```
<data-sources>
 <data-source type="oracle.jdbc.pool.OracleDataSource">
```

```
    <set-property property="description"
      value="Oracle Datasource"/>
    <set-property
      property="driverClass"
      value="oracle.jdbc.driver.OracleDriver"/>
    <set-property property="user" value="scott"/>
    <set-property property="password" value="tiger"/>

    <set-property
      property="url"
      value="jdbc:oracle:thin:@localhost:1521:DEV"/>
   </data-source>
  </data-sources>
```

The form-beans Element

The `form-beans` element configures multiple `ActionForm` classes for use within the `Action` classes and the view. Within the `form-beans` section, you can configure zero or more `form-bean` child elements. Each `form-bean` element represents a physical `ActionForm` class and has several child elements:

```
<!ELEMENT form-bean (icon?, display-name?, description?,
set-property*, form-property*)>
```

Each `form-bean` element has four attributes that you can specify (see Table 3).

Table 3. The form-bean attributes

Name	Description
`className`	If you don't want to use the standard configuration bean `org.apache.struts.config.FormBeanConfig`, you can specify your own class here. It must be a descendant of the `FormBeanConfig` class.
`dynamic`	You no longer have to set this attribute. It's included here because it was deprecated during the 1.1 Beta process. The framework determines this value based on the value of the `type` attribute. If the class specified in the `type` attribute is a descendant of the `DynaActionForm` class, the `dynamic` property will be set to `true`.

Table 3. The form-bean attributes (continued)

Name	Description
name	The unique identifier for this form bean; referenced by the action element to specify which form bean to use with its request. This value is required and must be unique within a module.
type	The fully qualified name of a Java class that extends the Struts ActionForm class. This attribute is required.

The following form-beans element illustrates how to specify a nondynamic ActionForm.

```
<form-beans>
 <form-bean
   name="loginForm"
   type="com.cavaness.banking.web.user.LoginForm"/>
</form-beans>
```

You can pass one or more dynamic properties to the org.apache.struts.action.DynaActionForm class using the form-property element. Dynamic properties are supported only when the type attribute of the surrounding form-bean element is org.apache.struts.action.DynaActionForm or a descendant class. Each form-property element also has five attributes that you can specify (see Table 4).

Table 4. The attributes of the form-property element

Name	Description
className	If you don't want to use the standard configuration bean org.apache.struts.config.FormPropertyConfig, you can specify your own class here.
initial	A string representation of the initial value for this property. Property values are initialized based on standard Java conventions.
name	The name of the property being described by this element. This attribute is required.
size	The number of array elements to create if the value of the type attribute specifies an array. The default value is 0.
type	The fully qualified Java class name of the implementation class of this bean property, optionally followed by "[]" to indicate that this field is indexed. This attribute is required.

The following form-bean fragment illustrates using the form-property element to specify a dynamic ActionForm:

```
<form-bean
  name="checkoutForm"
    type="org.apache.struts.action.DynaActionForm">
    <form-property name="firstName"
      type="java.lang.String"/>
    <form-property name="lastName"
      type="java.lang.String"/>
    <form-property name="age" type="java.lang.Integer"
      initial="18"/>
  </form-bean>
```

The global-exceptions Element

The global-exceptions section allows you to configure exception handlers declaratively. The global-exceptions element can contain zero or more exception elements:

```
<!ELEMENT global-exceptions (exception*)>
```

The exception element can also be specified in the action element. If an exception element is configured for the same type of exception both as a global exception and within an action element, the action level will take precedence. If no exception element mapping is found at the action level, the framework will then look for an exception mapping defined for the exception's parent class. Eventually, if a handler is not found, a ServletException or IOException will be thrown, depending on the type of the original exception.

The declaration of the exception element illustrates that it also has several child elements:

```
<!ELEMENT exception (icon?, display-name?, description?,
set-property*)>
```

Table 5 lists the exception element's attributes.

Table 5. The attributes of the exception element

Name	Description
`className`	The implementation class of the configuration bean that will hold the exception information. If specified, it must be a descendant of `org.apache.struts.config.ExceptionConfig`, which is the default class when no value is specified.
`handler`	The fully qualified Java class name of the exception handler that will process the exception. If no value is specified, the default class `org.apache.struts.action.ExceptionHandler` will be used. If a class is specified for this attribute, it must be a descendant of the `ExceptionHandler` class.
`key`	The message resources key specifying the error message associated with this exception.
`path`	The module-relative path of the resource to forward to if this exception occurs during the processing of an `Action`. This attribute is required.
`scope`	The identifier of the scope level where the `ActionError` instance should be stored. The attribute value must be either `request` or `session`. The default value is `request`.
`type`	The fully qualified Java class name of the exception that is to be handled. This attribute is required.
`bundle`	The servlet context attribute that identifies a resource bundle from which the key attribute of this element should come. If this attribute is not set, the default message resource for the current module is assumed.

The following example illustrates how to add an `exception` element at the global level:

```
<global-exceptions>
  <exception
    key="global.error.invalidlogin"
    path="/security/signin.jsp"
    scope="request"
    type="com.oreilly.struts.InvalidLoginException"/>
 </global-exceptions>
```

The global-forwards Element

Every action finishes by forwarding or redirecting to a view. In most cases, this view is a JSP page or a static HTML page,

but it can be another type of resource as well. Instead of referring to the view directly, the Struts framework uses the concept of a *forward* to associate a logical name to the resource. So, instead of referring to *login.jsp* directly, a Struts application might refer to this resource as the `login` forward.

The `global-forwards` section allows you to configure forwards that can be used by all actions within a module. The `global-forwards` section consists of zero or more `forward` elements:

```
<!ELEMENT global-forwards (forward*)>
```

The `forward` element maps a logical name to an application-relative URI. The application can then perform a forward or redirect using the logical name rather than the literal URI. This helps to decouple the controller and model logic from the view. The `forward` element can be defined in the `global-forwards` section as well as an `action` element. If a forward with the same name is defined in both places, the `action` level will take precedence.

The declaration of the `forward` element illustrates that it also has several child elements:

```
<!ELEMENT forward(icon?, display-name?, description,
set-property*)>
```

Table 6 lists the `forward` element's attributes.

Table 6. The attributes of the forward element

Name	Description
`className`	The implementation class of the configuration bean that will hold the forward information. The `org.apache.struts.action.ActionForward` is the default class when no value is specified.
`contextRelative`	Indicates whether the value of the `path` attribute should be considered context-relative if it starts with a slash (and is therefore not prefixed with the module prefix). The default value is `false`.
`name`	A unique value that is used to reference this forward in the application. This attribute is required.

Table 6. The attributes of the forward element (continued)

Name	Description
path	If the contextRelative attribute is true, the path is context-relative within the current web application (even if we are in a named module). If the contextRelative property is false, the path is the module-relative portion of the URL. This attribute is required and must begin with a slash (/) character.
redirect	A Boolean value that determines whether the RequestProcessor should perform a forward or a redirect when using this forward mapping. The default value is false, which means that a forward will be performed.

The following is an example of a global-forwards element:

```
<global-forwards>
  <forward name="Login" path="/security/signin.jsp"
    redirect="true"/>
  <forward name="SystemFailure"
    path="/common/systemerror.jsp"/>
  <forward
    name="SessionTimeOut"
    path="/common/sessiontimeout.jsp"
    redirect="true"/>
</global-forwards>
```

The action-mappings Element

The action-mappings element contains a set of action elements for a Struts application. The action-mappings element can contain zero or more action elements:

```
<!ELEMENT action-mappings (action*)>
```

The action element describes an ActionMapping object that is to be used to process a request for a specific module-relative URI. The action element also describes a mapping from a specific request path to a corresponding Action class. The controller selects a particular mapping by matching the URI path in the request with the path attribute in one of the action elements. The action element contains the following child elements:

```
<!ELEMENT action (icon?, display-name?, description,
set-property*, exception*, forward*)>
```

The attributes listed in Table 7 are available for the action element.

Table 7. The attributes of the action element

Name	Description
className	The implementation class of the configuration bean that will hold the action information. The org.apache.struts.action.ActionMapping class is the default class when no value is specified.
attribute	The name of the request or session scope attribute under which the form bean for this action can be accessed. A value is allowed here only if there is a form bean specified in the name attribute. This attribute has no default value. If both this attribute and the name attribute contain a value, this attribute will take precedence.
forward	A context-relative path of the web application resource that will process this request via RequestDispatcher.forward( ), instead of instantiating and calling the Action class. The attributes forward, include, and type are mutually exclusive.
include	A context-relative path of the web application resource that will process this request via RequestDispatcher.include( ), instead of instantiating and calling the Action class. The attributes forward, include, and type are mutually exclusive.
input	A context-relative path of the input form to which control should be returned if a validation error is encountered. This attribute is required if name is specified and the input bean returns validation errors.
name	The name of the form bean that is associated with this action. This value must be the name attribute from one of the form-bean elements defined earlier. This attribute has no default value.
path	A context-relative path of the submitted request, starting with a slash (/) character and omitting any filename extension if extension mapping is being used.
parameter	A general-purpose configuration parameter that can be used to pass extra information to the action instance selected by this action mapping. The Struts framework does not use this value in any way. If you provide a value here, you can obtain the value in your Action by calling the getParameter( ) method on the ActionMapping passed to the execute( ) method.

Table 7. The attributes of the action element (continued)

Name	Description
prefix	Used to match request parameter names to form bean property names. For example, if all of the properties in a form bean begin with "pre_", you can set the prefix attribute so the request parameters will match to the ActionForm properties. You can provide a value here only if the name attribute is specified.
roles	A comma-delimited list of security role names that are allowed to invoke this Action. When a request is processed, the RequestProcessor verifies the user has at least one of the roles identified within this attribute.
scope	Used to identify the scope within which the form bean is placed. It can be either request or session. It can be specified only if the name attribute is present. The default value is session.
suffix	Used to match request parameter names to form bean property names. For example, if all of the properties in a form bean end with "_foo", you can set the suffix attribute so the request parameters will match to the ActionForm properties. You can provide a value here only if the name attribute is specified.
type	A fully qualified Java class name that extends the org.apache.struts.action.Action class. It is used to process the request if the forward or include attributes are not specified. The attributes forward, include, and type are mutually exclusive.
unknown	A Boolean value indicating whether this action should be configured as the default for this application. If this attribute is set to true, this action will handle any request that is not handled by another action. Only one action mapping per application can have this value set to true. The default value is false.
validate	A Boolean value indicating whether the validate( ) method of the form bean, specified by the name attribute, should be called prior to calling the execute( ) method of this action. The default value is true.

The following is an example of an action element:

```
<action
  path="/signin"
  type="com.oreilly.strutssecurity.LoginAction"
  scope="request"
  name="loginForm"
  validate="true"
  input="/security/signin.jsp">
```

```
    <forward
      name="Success"
      path="/index.jsp" redirect="true"/>
    <forward
      name="Failure"
      path="/security/signin.jsp"
      redirect="true"/>
  </action>
```

The controller Element

If you're familiar with Struts Version 1.0, you'll notice that many of the parameters that were configured in the *web.xml* for the controller servlet are now configured using the `controller` element. Since the controller and its attributes are defined in *struts-config.xml*, you can define a separate `controller` element for each module. The declaration of the `controller` element illustrates that it has a single child element:

```
<!ELEMENT controller (set-property*)>
```

The `controller` element can contain zero or more `set-property` elements and many different attributes, which are listed in Table 8.

Table 8. The attributes for the controller element

Name	Description
`className`	The implementation class of the configuration bean that will hold the controller information. If specified, it must be a descendant of `ControllerConfig`, which is the default class when no value is specified.
`bufferSize`	The size of the input buffer used when processing file uploads. The default value is `4096`.
`contentType`	The default content type and optional character encoding that are set for each response. The default value is `text/html`. Even when a value is specified here, an action or a JSP page can override it.

Table 8. The attributes for the controller element (continued)

Name	Description
`debug`	The debugging level for this application. This value is used throughout the framework to determine how verbose the logging information should be for events that take place internally. The larger the value, the more verbose the logging is. This attribute is not required. The default value is 0, which causes little or no logging information to be written out. This attribute has been deprecated; you should now use your specific logging implementation to control the verbosity of log messages.

`forwardPattern`	A replacement pattern defining how the `path` attribute of a `forward` element is mapped to a context-relative URL when it starts with a slash (and when the `contextRelative` property is `false`). This value can consist of any combination of the following:
	`$M`
	This is replaced by the module prefix of this module.
	`$P`
	This is replaced by the `path` attribute of the selected `forward` element.
	`$$`
	This causes a literal dollar sign to be rendered
	`$x` *(where* `x` *is any character not defined above)*
	This is reserved for future use.
	If not specified, the default `forwardPattern` is `$M$P`, which is consistent with the previous behavior of forwards.

`inputForward`	Set to `true` if you want the `input` attribute of the `action` element to be the name of a local or global `forward` element. Set to `false` to treat the `input` attribute of `action` elements as a module-relative path.
`locale`	A Boolean value indicating whether the user's preferred `Locale` is stored in the user's session (if it is not already present). The default value is `true`.
`maxFileSize`	The maximum size (in bytes) of a file to be accepted as a file upload. This value can be expressed as a number followed by a `K`, `M`, or `G`, which is interpreted to mean kilobytes, megabytes, or gigabytes, respectively. The default value is `250M`.

Table 8. The attributes for the controller element (continued)

Name	Description
`memFileSize`	The maximum size (in bytes) of a file whose contents will be retained in memory after uploading. Files larger than this threshold will be written to some alternative storage medium, typically a hard disk. This value can be expressed as a number followed by a `K`, `M`, or `G`, which is interpreted to mean kilobytes, megabytes, or gigabytes, respectively. The default value is `256K`.
`multipartClass`	The fully qualified Java class name of the multipart request handler class to be used when uploading files from a user's local filesystem to the server. The default value is the `org.apache.struts.upload.CommonsMultipartRequestHandler` class.
`nocache`	A Boolean value indicating whether the framework should set `nocache` HTTP headers in every response. The default value is `false`.

`pagePattern`	A replacement pattern defining how the page attribute of custom tags is mapped to a context-relative URL of the corresponding resource. This value can consist of any combination of the following:
	`$M`
	This is replaced by the module prefix of this module.
	`$P`
	This is replaced by the `path` attribute of the selected `forward` element.
	`$$`
	This causes a literal dollar sign to be rendered.
	`$x` *(where* `x` *is any character not defined above)*
	This is reserved for future use.
	If not specified, the default `pagePattern` is `$M$P`, which is consistent with the previous behavior of URL calculation.

`processorClass`	The fully qualified Java class name of the request processor class to be used to process requests. The value specified here should be a descendant of `RequestProcessor`, which is the default value.
`tempDir`	Specifies the temporary working directory that is used when processing file uploads. This attribute is not required; the servlet container will assign a default value for each web application.

The ControllerConfig class is used to represent the information configured in the controller element in memory. The following fragment shows an example of how to configure the controller element:

```
<controller
 contentType="text/html;charset=UTF-8"
 debug="3"
 locale="true"
 nocache="true"
 processorClass=
 "com.oreilly.struts.CustomRequestProcessor"/>
```

The message-resources Element

The message-resources element specifies characteristics of the message resource bundles that contain the localized messages for an application. Each Struts configuration file can define one or more message resource bundles; therefore, each module can define its own bundles. The declaration of the message-resources element shows that it contains only a set-property element:

```
<!ELEMENT message-resources (set-property*)>
```

Table 9 lists the attributes for the message-resources element.

Table 9. The message-resources element's attributes

Name	Description
className	The implementation class of the configuration bean that will hold the message-resources information. If specified, it must be a descendant of MessageResourcesConfig, which is the default class when no value is specified.
factory	The fully qualified Java class name of the MessageResourcesFactory class that should be used. The class PropertyMessageResources is the default.
key	The servlet context attribute under which this message resources bundle will be stored. The default attribute is the value specified by the string constant at Globals.MESSAGES_KEY. The application module prefix (if any) is appended to the key (${*key*}${*prefix*}).

Table 9. The message-resources element's attributes (continued)

Name	Description
`null`	A Boolean value indicating how the `MessageResources` subclass should handle the case when an unknown message key is used. If this value is set to `true`, an empty string will be returned. If set to `false`, a message that looks something like "???global.label.missing???" will be returned. The actual message will contain the bad key. The default value is `true`.
`parameter`	This attribute is the base name of the resource bundle. For example, if the name of your resource bundle is *ApplicationResources.properties*, you should set the parameter value to `ApplicationResources`. This attribute is required. If your resource bundle is within a package, you must provide the fully qualified name in this attribute.

The following example shows how to configure multiple `message-resources` elements for a single application. Notice that the second element has to specify the `key` attribute, since there can be only one stored with the default key:

```
<message-resources
  null="false"
  parameter="StorefrontMessageResources"/>

<message-resources
  key="IMAGE_RESOURCE_KEY"
  null="false"
  parameter="StorefrontImageResources"/>
```

The plug-in Element

Plug-ins, a new feature to Struts, allow Struts applications to discover resources dynamically at startup. You must provide a nonabstract Java class that implements the `org.apache.struts.action.PlugIn` interface and add a `plug-in` element to the configuration file.

The `plug-in` element specifies a fully qualified class name of a general-purpose application plug-in module that receives notification of application startup and shutdown events. At startup, the framework will create an instance of each `PlugIn`

class specified. The `init()` method is called when the application is started, and the `destroy()` method is called when the application is stopped.

The declaration of the `plug-in` element shows that it may contain zero or more `set-property` elements so that extra configuration information can be passed to your `PlugIn` class:

```
<!ELEMENT plug-in          (set-property*)>
```

There is a single attribute for the `plug-in` element, which is listed in Table 10.

Table 10. The attribute for the plug-in element

Name	Description
`className`	The fully qualified Java class name of the plug-in class. It must implement the `PlugIn` interface.

The following fragment shows two `plug-in` elements being used:

```
<plug-in
  className=
  "com.cavaness.banking.service.memory.MemoryDatabasePlugIn">

  <set-property property="pathname"
    value="/WEB-INF/database.xml"/>
</plug-in>

<plug-in
  className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames" value=
    "/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```