CONTENTS INCLUDE:

- Configuring the Web Application
- Actions
- Configuring Actions
- Result Types
- Interceptors
- Hot Tips and more...

Struts2

By Ian Roughley

ABOUT STRUTS2

Struts2 is the next generation of model-view-controller web application frameworks. It aims at providing increased productivity through reduced XML configuration, smart conventions, and a modular and loosely-coupled architecture. This refcard refers to Struts2 version 2.0.x.

CONFIGURING THE WEB APPLICATION

To configure Struts2, a filter needs to be configured in the applications `web.xml` file:

```
<web-app>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>action2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

ACTIONS

Actions are the basic building blocks of Struts:

```
public class UserAction {

  private int age;
  private UserService service;

  public int getAge() { return age; }
  public void setAge( int age ) { this.age = age; }

  public void setUserService( UserService service ) {
    this.service = service;
  }

  public String execute() throws Exception {
    service.updateAge(age);
    return "success";
  }
}
```

Features of a Struts2 action are:

- An action doesn't need to extend classes or implement interfaces (it's a POJO)
- Use getters and setter to access data in your view and transfer data to and from the action
- Data conversion is done for you by Struts2 (all basic type conversion is available, and you can define your own more complex conversions)

Actions, continued

- Pluggable dependency injection is used for services (injecting a Spring Framework-managed bean is as simple as placing a setter on the action with a name that matches the bean's id in the Spring configuration)
- The method providing the logic is called `execute` by convention—but it could be called anything—as long as it returns a `String` and has no parameters (it can also throw `Exception`)

Hot Tip Even though an action isn't required to extend another class, it sometimes makes sense. The class `ActionSupport` is one such class, providing default implementations for validation support, internationalization, etc. so you don't have to.

CONFIGURING ACTIONS

The `struts.xml` file (accessed via the classpath) provides configuration for Struts2.

```
<struts>
<constant name="struts.devMode" value="true" />
<package name="test" extends="struts-default"
namespace="/tests" >

   <default-interceptor-ref name="basicStack" />

   <global-results>
     <result name="error" type="dispatcher">
     /error.jsp</result>
   </global-results>

   <global-exception-mappings>
     <exception-mapping
     exception="java.lang.Exception" result="error" />
   </global-exception-mappings>
```

→

Configuring Actions, continued

```
<default-action-ref name="testMe" />
  <action name="updateTest"
  method="update"class="com.fdar.s2.MyAction" >
    <result name="success" type="dispatcher">/WEB-INF
    /jsp/found.jsp</result>
    <interceptor-ref name="altStack" />
    <exception-mapping
    exception="java.lang.Exception"
    result="exception" />
    <param name="version">2.0.9</param>
  </action>
</package>
<include file="struts-module1.xml" />
</struts>
```

Hot Tip Many of the configuration options are now available as annotations, but not all of them. So it's important to know how to use the struts.xml configuration file.

Tag Name	Description
constant	**Changes the value of a configuration property.** ▪ `name`—the name of the property for the value to change ▪ `value`—the new value to assign
package(*)	**Provides a way to hierarchically split an application into smaller units using the URL path namespace.** ▪ `name`—a unique name (across all packages) ▪ `extends`—the package to extend (this package) ▪ `namespace`—the unique URL path namespace to access the actions in this package
default-interceptor-ref	**The interceptors to apply to all actions in the package by default** ▪ `name`—the interceptor or interceptor stack to use
global-results	**Contains a list of result tags (see `result` tag definition below in this table), that can be referred to by any action in the package (or extending packages)**
global-exception-mappings	**Contains a list of `exception-mapping` tags (see exception-mapping definition below in this table) to apply to all actions in the package by default.**
exception-mapping (*)	**Maps an exception to the result that should be rendered when the exception is thrown (requires the `exception` interceptor).** ▪ `exception`—the package and class of the exception ▪ `result`—the result to forward the user to when the exception is encountered
default-action-ref	**The action to invoke when the URL doesn't match any configured.** ▪ `name`—the action name
action	**Describes an action that can be invoked** ▪ `name`—the name of the action ("`.action`" is added as an extension when used in the URL) ▪ `method` (not required)—the method of the class to invoke (defaults to "`execute`") ▪ `class`—the class that provides the action logic
result	**Provides the view options for the result being returned from the action classes logic method (more than one result is allowed).** ▪ `name` (not required)—the String value returned from the action logic to match (defaults to "`success`") ▪ `type` (not required)—the result type to use to render the result (defaults to "dispatcher") The value within the tags provides the template name to render
interceptor-ref	**The interceptors to apply to this action (can use more than one)** ▪ `name`—the name of the interceptor or interceptor stack
param	**Allows parameters to be assigned to the action from configuration files.** ▪ `name`—the name of the parameter being assigned The value within the tags provides the value to assign—may contain OGNL (denoted as `${OGNLExpression}`)
include	**Includes another configuration file, allowing a large application to have multiple configuration files.** ▪ `file`—the name of the file to include

Table 1. struts.xml Configuration Elements

(*) Some attributes have been omitted because they have limited usage, see http://struts.apache.org/2.x/docs/configuration-elements.html for the complete list of configuration attributes available.

For a complete list of configuration properties that can be modified, take a look at http://struts.apache.org/2.x/docs/strutsproperties.html.

Action Annotations

The annotations currently available to actions are listed in Table 2.

Annotation Name	Description
@Namespace	**The value is the name of the namespace to use for the action**
@ParentPackage	**The value is the name of the package that the action will inherit from**

@Results	**Used when more than one @Result annotation is configured for the action.**
	`@Results({`
	`    @Result(…),`
	`    @Result(…)`
	`})`

@Result	**Defines the results for an action.**
	▪ `name`—the result from the action to match (defaults to "success")
	▪ `type`—the result type class to use (i.e. JSP rendering result type)
	▪ `value`—the value for the result type (i.e. the JSP to render)
	▪ `params` (not required)—additional parameters to pass to the result type
	`@Result(`
	`    name="success"`
	`    type= ServletActionRedirectResult.class,`
	`    value="selectLocation",`
	`    params={"method","input"})`

Table 2. Action Annotations

When using action-based annotation, there is additional configuration required in web.xml:

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>
   org.apache.struts2.dispatcher.
   FilterDispatcher</filter-class>
  <init-param>
    <param-name>actionPackages</param-name>
    <param-value>com.fdar.apress.s2.actions</param-value>
  </init-param>
</filter>
```

Validation Annotations

Note To activate annotation-based validation for an action, the class must first be annotated with @Validation. This allows Struts2 to further interrogate only those classes that are known to have validation annotations.

Each of the validations in Table 3 are method level validations, and can be applied to setters or the execute method. As well as their individual attributes, every annotation has the following common attributes:

- **message:** the message to display to the user
- **key (not required):** an i18n key from a language specific resource
- **shortCircuit (not required):** whether to abort other validations if this one fails

Additionally, validators may have the following (annotated in Table 3 as applicable):

a. **fieldName (not required):** specifies the field being acted upon

→

Validation Annotations, continued

b. **type:** `Validator.FIELD` or `Validator.SIMPLE` (defaults to `ValidatorType.FIELD`)

Annotation Name	Description
@ConversationErrorFieldValidator (a)(b)	**Validates that there are no conversion errors for a field.**

@DateRangeFieldValidator (a)(b)	**Validates that a date falls between a range.**
	▪ min (not required)—the minimum valid date
	▪ max (not required)—the maximum valid date
	`@DoubleRangeFieldValidator(`
	`    message = "Please enter a date this year",`
	`    key = "validate.thisYear",`
	`    min = "2008/01/01",`
	`    max = "2008/12/31")`

@DoubleRangeFieldValidator (a)(b)	**Validates that a double falls between a range.**
	▪ minInclusive (not required)—the inclusive minimum valid date
	▪ maxInclusive (not required)—the inclusive maximum valid date
	▪ minExclusive (not required)—the exclusive minimum valid date
	▪ maxExclusive (not required)—the exclusive maximum valid date
	`@DateRangeFieldValidator(`
	`    message = "Please enter a date this year",`
	`    key = "validate.thisYear",`
	`    minInclusive = "2008/01/01",`
	`    maxInclusive = "2008/12/31")`

@EmailValidator (a)(b)	**Validates that the email has a valid format.**

@ExpressionValidator	**Validates that an expression evaluates to true.**
	▪ expression—the OGNL expression to evaluate
	`@ExpressionValidator(`
	`    message = "Please confirm password",`
	`    key = "confirm.password",`
	`    shortCircuit = true,`
	`    expression =`
	`    "password.equals(confirmPassword)" )`

@FieldExpressionValidator (a)	**Validates that a field expression evaluates to true.** ▪ expression—the OGNL expression to evaluate
IntRangeFieldValidator (a)(b)	**Validates that an int falls between a range.** ▪ min (not required)—the minimum valid date ▪ max (not required)—the maximum valid date
@RequiredFieldValidator (a)(b)	**Validates that the field is not null.**
@RegexFieldValidator (a)(b)	**Validates that a string field matches a regular expression.** ▪ expression—the regular expression to evaluate
@RequiredStringValidator (a)(b)	**Validates that the field is not empty (i.e. not null and length > 0)**
@StringLengthFieldValidator (a)(b)	**Validates that a String is of a certain length.** ▪ trim (not required)—removes white space padding ▪ minLength (not required)—the minimum length the String must be ▪ maxLength (not required)—the maximum length the String must be
@UrlValidator (a)(b)	**Validates that the field has a valid URL format**

@VisitorFieldValidator (a)	**Steps into action properties to continue validation. This keeps validation for models separate and re-useable across multiple actions.**
	▪ context (not required)—the validation context. Multiple contexts allow for different validation rules in different circumstances (defaults to action context)
	▪ appendPrefix (not required)—whether the property name (of the action) should be pre-pended to the field name. i.e. "user.name" vs. "name" (defaults to true).
	`@VisitorFieldValidator(`
	`    message = "Error validating User",`
	`    key = "user.error",`
	`    shortCircuit = true,`
	`    context = "model",`
	`    appendPrefix = true)`

@CustomValidator (a)(b)	**Used to specify a custom validator. In addition, an array of @ValidationParameter annotations can be used to pass parameter to the custom validator. Custom validators extend the ValidatorSupport or FieldValidatorSupport class.**
	`@CustomValidator(`
	`  type ="myUserValidator",`
	`  fieldName = "user",`
	`  parameters = {`
	`    @ValidationParameter(`
	`      name = "source",`
	`      value = "admin" ) }`
	`)`

Table 3. Validation Annotations

Updated documentation on the validators can be found at: http://struts.apache.org/2.x/docs/annotations.html.

The `@Validations` validator allows you to specify multiple validators on the `execute()` method. It has the following parameters:

Parameter	Description
requiredFields	a list of RequiredFieldValidators
customValidators	a list of CustomValidators
conversionErrorFields	a list of ConversionErrorFieldValidators
dateRangeFields	a list of DateRangeFieldValidators
emails	a list of EmailValidators
fieldExpressions	a list of FieldExpressionValidators
intRangeFields	a list of IntRangeFieldValidators
requiredStrings	a list of RequiredStringValidators
stringLengthFields	a list of StringLengthFieldValidators
urls	a list of UrlValidators
visitorFields	a list of VisitorFieldValidators
regexFields	a list of RegexFieldValidator
expressions	a list of ExpressionValidator

```
@Validations(
  requiredFields = {
    @RequiredFieldValidator(
      fieldname="userName",
      message="Username is required")},
  emails = {
    @EmailValidator(fieldName="emailAddress",
      message="Email address is required")}
)
```

Conversion Annotations

Similar to validation annotations, when using conversion annotations you must add the class-level `@Conversion` annotation to the class.

Once this is complete, conversion annotations from Table 4 can be used. These annotations can be applied at the method or property level.

Annotation Name	Description

@TypeConversion	**Provides custom conversion for a property or method. Custom converters extend the `StrutsTypeConverter` class.**
	▪ key (not required)—the property or key name (defaults to property name)
	▪ type (not required)—determines the scope of the conversion: `ConversionType.APPLICATION` or `ConversionType.CLASS` (defaults to `ConversionType.CLASS`)
	▪ rule (not required)—the conversion rule: `ConversionRule.PROPERTY, ConversionRule.MAP, ConversionRule.KEY, ConversionRule.KEY_PROPERTY, ConversionRule.ELEMENT, ConversionRule.CREATE_IF_NULL` (defaults to `ConversionRule.PROPERTY`)
	▪ converter (converter or value required)—the class name of the converter
	▪ value (converter or value required)—the value to set when using `ConversionRule.KEY_PROPERTY`
	`@TypeConversion(`
	`  type = ConversionType.APPLICATION,`
	`  property = "java.util.Date",`
	`  converter =`
	`  "com.opensymphony.xwork2.util.XWorkBasic-Converter")`

Table 4. Conversion Annotations

Hot Tip

There are more conversion annotations available, although with generics they are mostly unused. If you're interested, the full list can be found at **http://struts.apache.org/2.x/docs/annotations.html**.

RESULT TYPES

As well as JSP templates, a Struts2 action can render a variety of other options. Each of those available are listed in Table 5.

Result Type Name	Description

Chain Result (*)
ActionChainResult.class
	Chains one action to another action.
	▪ actionName (default)—the action to invoke next
	▪ namespace (not required)—the namespace of the action being chained to (defaults to current namespace)
	▪ method (not required)—the method on the action to execute (defaults to execute)
	`<result type="chain">`
	`  <param name="actionName">listAction</param>`
	`  <param name="namespace">/user</param>`
	`</result>`

Dispatcher Result
ServletDispatcherResult.class
	Renders a JSP template.
	▪ location (default)—the template to render
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	`<result name="success"`
	`    type="dispatcher">user.jsp</result>`
	or (using the defaults)
	`<result>user.jsp</result>`

Freemarker Result (*)
FreemarkerResult.class
	Renders a Freemarker template.
	▪ location (default)—the template to render
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	`<result name="success"`
	`  type="freemarker">user.ftl</result>`

HttpHeader Result
HttpHeaderResult.class
	Returns HTTP headers back to the client.
	▪ status—the HTTP response status code to return
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	▪ headers (not required)—header values to return
	▪ error (not required)—the error code to return
	▪ errorMessage (not required)—error message to return (if error is set)
	`<result name="notAuthorized" type="httpheader">`
	`  <param name="status">401</param>`
	`  <param name="headers.user">${username}</param>`
	`  <param name="headers.resource">/deleteUser</param>`
	`</result>`

Redirect Result
ServletRedirectResult.class
	Performs a URL redirect rather than rendering a template.
	▪ location (default)—the URL to redirect to
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	`<result name="success" type="redirect">`
	`  <param name="location">viewUser.jsp</param>`
	`  <param name="parse">false</param>`
	`</result>`

Redirect Action Result
ServletActionRedirectResult.class
	Performs a URL redirect to another Struts2 action.
	▪ actionName (default)—the action to redirect to
	▪ namespace (not required)—the namespace of the action being redirected to (default to current namespace)
	`<result type="redirectAction">`
	`  <param name="actionName">dashboard</param>`
	`  <param name="namespace">/secure</param>`
	`</result>`

Velocity Result
VelocityResult.class
	Renders a Velocity template.
	▪ location (default)—the template to render
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	`<result name="success" type="velocity">`
	`  <param name="location">user.vm</param>`
	`</result>`

Stream Result (*)
StreamResult.class
	Streams raw data back to the client.
	▪ contentType (not required)—the mime-type of the response (defaults to text/plain)
	▪ contentLength (not required)—the stream length in bytes
	▪ inputName (not required)—the name of the InputStream to return to the client (defaults to inputStream)
	▪ bufferSize (not required)—the buffer size when copying from input to output (default 1024)
	`<result name="success" type="stream">`
	`  <param name="contentType">image/jpeg</param>`
	`  <param name="inputName">imageStream</param>`
	`</result>`

Table 5. Available Result Types

Result Types, continued

Result Type Name	Description

XSL Result
XSLTResult.class
	Renders XML by serializing attributes of the action, which may be parsed through an XSL template.
	▪ location (default)—the template to render
	▪ parse (not required)—whether to parse OGNL expressions (true by default)
	▪ matchingPattern (not required)—a pattern to match the desired elements
	▪ excludingPattern (not required)—a pattern to eliminate unwanted elements
	`<result name="success" type="xslt">`
	`  <param name="location">user.xslt</param>`
	`  <param name="matchingPattern">^/result/[^/*]$<param>`
	`</result>`

Table 5. Available Result Types, continued

Hot Tip

It's not just information from the configuration file that can be used in the result configuration. Expressions and values from the Value Stack can be accessed by placing the expression with the "${" and "}" characters. (i.e. <result>/user/${user.name}</result>).

(*) Some have additional less commonly used parameters. These parameters can be found at http://struts.apache.org/2.x/docs/result-types.html.

The online documentation for Result Types can be found at http://struts.apache.org/2.x/docs/result-types.html.

INTERCEPTORS

Interceptors play a large role in providing core framework features in Struts2. Table 6 provides a list of all the interceptors available in Struts2.

(a) denotes those interceptors implementing `MethodFilterInterceptor`. These interceptors have the following additional parameters:

- **excludeMethods**: method names to be excluded from interceptor processing
- **includeMethods**: method names to be included in interceptor processing

Name/ Configuration Value	Description/Attributes

Alias Interceptor
alias
	Allows parameters in the request to be set on the action under a different name.
	▪ aliasesKey (not required)—the name of the action parameter that contains the name-to-alias map (defaults to aliases).
	`<action name="test" class="com.examples.TestAction">`
	`    <param name="aliases">#{ 'action' : 'alias' }</param>`
	`</action>`

Chaining Interceptor chain	**Works with the chain result to copy data from one action to another.** ▪ excludes (not required)—the list of parameter names to exclude from copying (all others will be included). ▪ includes (not required)—the list of parameter names to include when copying (all others will be excluded).
Checkbox Interceptor checkbox	**Looks for a hidden identification field that specifies the original value of the checkbox. Sets the value of checkbox elements that aren't submitted.** ▪ setUncheckedValue (not required)—the value to set as the unchecked value (defaults to false)
Cookie Interceptor cookie	**Sets values in the Value Stack based on the cookie name and value—name and value must match for value to be set.** ▪ cookiesName—comma separated list of cookie names to be injected into the Value Stack (all cookies can be specified with an asterisk). ▪ cookiesValue—comma separated list of cookie values to match (all cookies names can be specified by using an asterisk for the value)

Table 6. Available Interceptors

→

Interceptors, continued

Name/ Configuration Value	Description/Attributes
Conversation Error Interceptor conversionError	**Sets the conversion errors from the ActionContext into the Action's field errors.**
Create Session Interceptor createSession	**Creates a HttpSession.**
Execute and Wait Interceptor execAndWait	**Starts a long-running action in the background on a separate thread, while preventing the HTTP request from timing out. While still in progress, a "wait" result is returned and rendered for the user (i.e. for an updating progress meter).** ▪ threadPriority (not required)—the priority to assign the processing thread (default Thread.NORM_PRIORITY) ▪ delay (not required)—an initial delay before the wait page is displayed ▪ delaySleepInterval (not required)—how long to wait between wait page refreshing (only used with delay, default is 100 milliseconds)
Exception Interceptor exception	**Allows exception to be handled declaratively (via configuration).** ▪ logEnabled (not required)—whether to log exceptions ▪ logLevel (not required)—the logging level to use (default is debug) ▪ logCategory (not required)—the logging category to use (default is com.opensymphony.xwork2.interceptor.Exception MappingInterceptor)
File Upload Interceptor fileUpload	**Allows the multi-part uploading of files. Three setters are required on the action for each property (the property being the name of the HTML form element)—{property}: the actual File, {property}ContentType: the files content type, and {property}FileName: the name of the file uploaded** ▪ maximumSize (not required)—the maximum size in bytes for the file (default to ~2MB) ▪ allowedTypes (not required)—a comma separated list of allowed content types, i.e. text/html (defaults to allow all types)
Internationalization Interceptor i18n	**Allows the setting and switching of user locales.** ▪ parameterName (not required)—the name of the HTTP request parameter that can switch the locale (default is request_locale) ▪ attributeName (not required)—the name of the session key to store the selected locale (default is WW_TRANS_I18N_LOCALE)
Logger Interceptor logger	**Logs the start and end of the action's execution (logged at the INFO level).**
Message Store Interceptor store	**Stores the action's ValidationAware messages, errors and field errors into HTTP Session so they can be accessed after the current HTTP request.** ▪ allowRequestParameterSwitch (not required)—enables the request parameter that can switch the operation mode of the interceptor ▪ requestParameterSwitch (not required)—the request parameter that will indicate what mode this interceptor is in. ▪ operationMode (not required) – the operation mode, 'STORE': stores messages; 'RETRIEVE': retrieves stored messages, or 'NONE': do nothing (defaults to 'NONE')
Model Driven Interceptor modelDriven	**Places the model (exposed via implementing the the Model-Driven interface on actions) from the action into the Value Stack above the action.**
Scoped Model Driven Interceptor scopedModelDriven	**Retrieves the model (specified by the ScopedModelDriven interface) before an action executes and stores the model after execution.** ▪ className (not required)—the model class name (defaults to the model class name) ▪ name (not required)—the key to store the model under (defaults to the model class name). ▪ scope (not required)—the scope to store the model under (defaults to 'request' but can also be 'session')
Parameters Interceptor (a) params	**This interceptor sets all HTTP parameters onto the Value Stack. Actions that want to programmatically define acceptable parameters can implement ParameterNameAware interface.** ▪ ordered (not required)—set to true if you want the top-down property setter behavior
Prepare Interceptor (a) prepare	**Calls a method for pre-execute logic for classes implementing the Preparable interface. The method called is either prepare{methodName}, where {methodName} is usually execute, or a generic prepare method.** ▪ alwaysInvokePrepare (not required)—determines whether the prepare method will always be invoked (defaults to true)

Table 6. Available Interceptors, continued

Name/ Configuration Value	Description/Attributes
Scope Interceptor scope	**Sets action properties from the HTTP session before an action is executed, and stores them back into the HTTP session after execution.** ▪ session (not required)—a comma delimited list of properties to be stored in HTTP session scope ▪ application (not required)—a comma delimited list of properties to be stored in HTTP application scope ▪ key (not required)—the key to store the properties under, can be CLASS (generates a unique key based on the class name), ACTION (generates a unique key based on the action name), any supplied value ▪ type (not required)—'start': all properties are set to the actions default values; 'end': all properties are removed once the action is run; anything else keeps default behavior ▪ sessionReset (not required)—when set to true all properties are reset
Servlet Configuration Interceptor servletConfig	**Allows the action to access HTTP information via interfaces. The interfaces that this interceptor supports are: ServletContextAware, ServletRequestAware, ServletResponseAware, ParameterAware, RequestAware, SessionAware, ApplicationAware and PrincipalAware.**
Static Parameters Interceptor staticParams	**Populates the action with the static parameters defined in the action configuration. If the action implements Parameterizable, a map of the static parameters will also be passed directly to the action.**
Roles Interceptor roles	**The action is invoked only if the user has the necessary role (supplied via the HttpServletRequest).** ▪ allowedRoles—roles allowed to access the action ▪ disallowedRoles—roles not allowed to access the action
Timer Interceptor timer	**Logs the execution time of the request (in milliseconds).** ▪ logLevel (not required)—the logging level to use (default is info) ▪ logCategory (not required)—the logging category to use (default is com.opensymphony.xwork2.interceptor TimerInterceptor)
Token Interceptor (a) token	**Ensures that only one request per token (supplied via the token tag) is processed—prevents double submitting of forms.**
Token Session Interceptor (a) tokenSession	**Builds off of the Token Interceptor, providing advanced logic for handling invalid tokens (providing intelligent fail-over in the event of multiple requests using the same session).**
Validation Interceptor (a) validation	**Runs the validations for the action.**
Workflow Interceptor (a) workflow	**Redirects user to an alternative result when validation errors are present (does not perform validation).** ▪ inputResultName (not required)—the result to return when validation errors exist (defaults to input)
Parameter Filter Interceptor (not pre-configured)	**Blocks parameters from entering the Value Stack and being assigned to the action.** ▪ allowed (not required)—a comma delimited list of parameter prefixes that are allowed ▪ blocked—a comma delimited list of parameter prefixes that are not allowed to pass ▪ defaultBlock—if true, all parameters are blocked and only those matching the allowed attribute will be allowed to pass (default to false)
Profiling Interceptor profiling	**Enables simple profiling (to the logger) when developer mode is enabled.** ▪ profilingKey—the key to use to activate profiling

Table 6. Available Interceptors, continued

The online documentation for interceptors can be found at http://struts.apache.org/2.x/docs/interceptors.html.

Interceptors are configured in `struts.xml` within the package tag. For single interceptors, the `interceptor` tag is used specifying a unique (across individual interceptors and interceptor stacks) name and the implementing class. To configure interceptor stacks, the `interceptor-stack` tag is used; listing the interceptor's using the `interceptor-ref` tag.

```
<interceptors>
  <interceptor name="breadcrumb"
  class="com.fdar.BreadCrumbInterceptor" />
  <interceptor-stack name="appStack">
    <interceptor-ref name="basicStack" />
    <interceptor-ref name="breadcrumb" />
  </interceptor-stack>
</interceptors>
```

Interceptors, continued

Hot Tip It's important not only to have the correct interceptors but ensure that they are executed in the correct order. So make sure your interceptor stacks are defined in the order you want the interceptors executed!

The parameters for interceptors can be configured in two ways. Parameters can be added using the param tag when configuring the interceptor:

```
<interceptor-ref name="validation">
  <param name="excludeMethods">input,back,cancel,
  browse</param>
</interceptor-ref>
```

The other option is within an actions' configuration, by specifying the param tag inside the `interceptor-ref` tag. In this case, the interceptor name prepends the parameter being set on the interceptor:

```
<action name="testMe"
class="com.fdar.apress.s2.MyAction">
  <interceptor-ref name="defaultStack">
    <param name="validation.excludeMethods">
    prepare,findById</param>
  </interceptor-ref>
</action>
```

In addition to the methods that need to be implemented in the Interceptor interface, interceptors can provide lifecycle callbacks. The callbacks methods are denoted by the annotations in Table 7.

Annotation Name	Description
@After	**Denotes methods on the interceptor to execute after the execute() method is invoked.** ▪ priority (not required)—the order to execute @After annotations
@Before	**Denotes methods on the interceptor to execute before the execute() method is invoked.** ▪ priority (not required)—the order to execute @Before annotations
@BeforeResult	**Denotes methods on the interceptor to execute before the result is rendered.** ▪ priority (not required)—the order to execute @BeforeResult annotations

Table 7. Interception Annotations

ABOUT THE AUTHOR

Ian Roughley

Ian Roughley is a speaker, author, and consultant. For more than ten years he has been helping clients ranging in size from Fortune 10 companies to start-ups. Focused on a pragmatic and results-based approach, he is a proponent for open source, as well as process and quality improvements through agile development techniques.

Publications
Author of *Starting Struts2* and *Practical Struts2 Web 2.0 Projects*; Java editor for *InfoQ.com*

Web Site
http://www.fdar.com

Email
ian@fdar.com

DZone

DZone communities deliver over 3.5 million pages per month to more than 1.5 million software developers, architects and designers. DZone offers something for every developer, including news, tutorials, blogs, cheatsheets, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-17-2
ISBN-10: 1-934238-17-1
50795
9 781934 238172
$7.95

Version 1.0