Interceptors

The default Interceptor stack is designed to serve the needs of most applications. Most applications will **not** need to add Interceptors or change the Interceptor stack.

Many Actions share common concerns. Some Actions need input validated. Other Actions may need a file upload to be pre-processed. Another Action might need protection from a double submit. Many Actions need drop-down lists and other controls pre-populated before the page displays.

The framework makes it easy to share solutions to these concerns using an "Interceptor" strategy. When you request a resource that maps to an "action", the framework invokes the Action object. But, before the Action is executed, the invocation can be intercepted by another object. After the Action executes, the invocation could be intercepted again. Unsurprisingly, we call these objects "Interceptors."

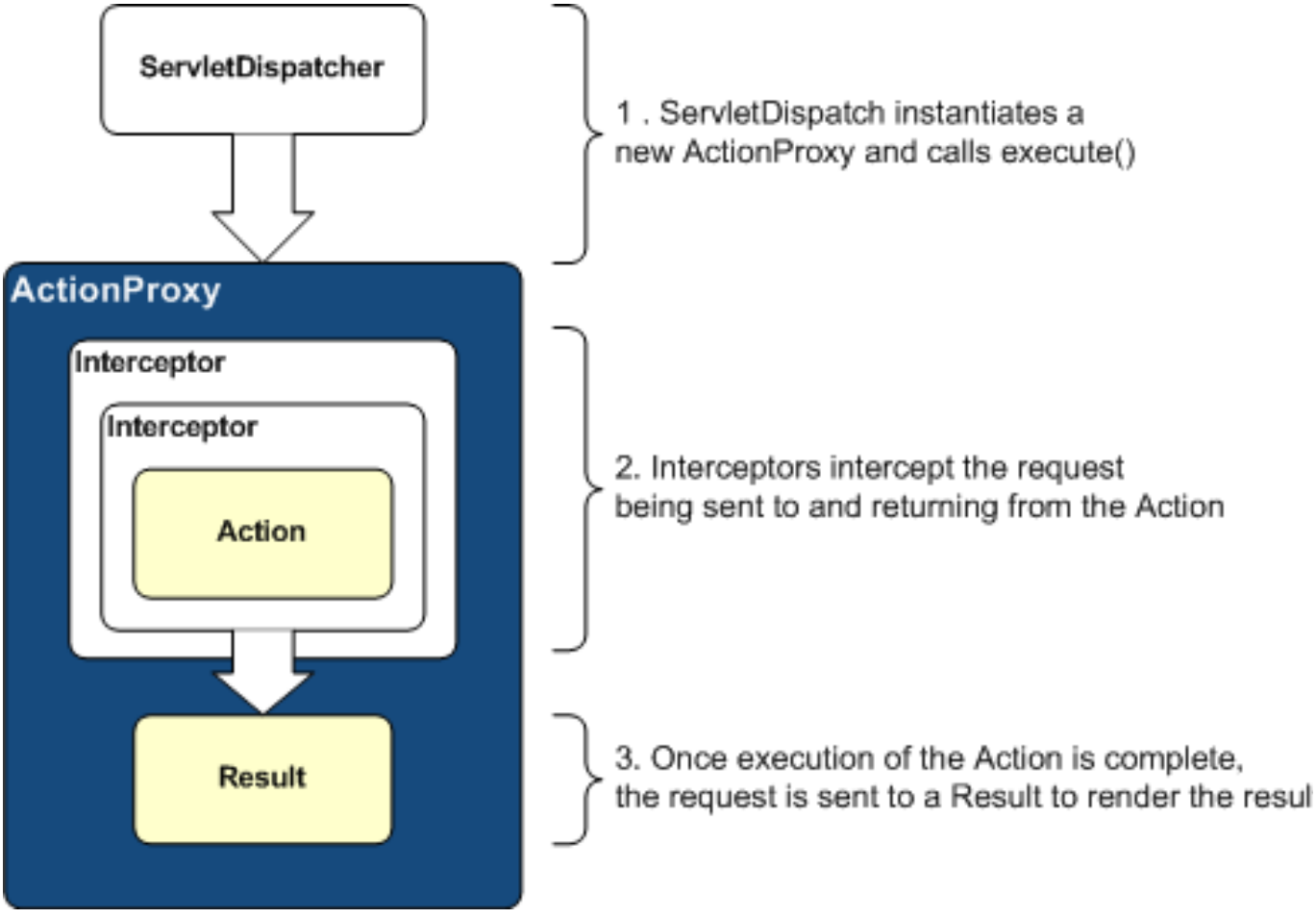
2

Understanding Interceptors

Interceptors can execute code before and after an Action is invoked. Most of the framework's core functionality is implemented as Interceptors. Features like double-submit guards, type conversion, object population, validation, file upload, page preparation, and more, are all implemented with the help of Interceptors. Each and every Interceptor is pluggable, so you can decide exactly which features an Action needs to support.

Interceptors can be configured on a per-action basis. Your own custom Interceptors can be mixed-and-matched with the Interceptors bundled with the framework. Interceptors "set the stage" for the Action classes, doing much of the "heavy lifting" before the Action executes.

Action Lifecyle



In some cases, an Interceptor might keep an Action from firing, because of a double-submit or because validation failed. Interceptors can also change the state of an Action before it executes.

The Interceptors are defined in a stack that specifies the execution order. In some cases, the order of the Interceptors on the stack can be very important.

Configuring Interceptors

xmlstruts.xml<package name="default" extends="struts-default"> <interceptors> <interceptor name="timer" class=".."/> <interceptor name=" logger" class=".."/> </interceptors> <action name="login" class="tutorial.Login"> <interceptor-ref name="timer"/> <interceptor-ref name="logger"/> <result name="input">login.jsp</result> <result name="success" type="redirectAction">/secure/home</result> </action> </package>

Stacking Interceptors

With most web applications, we find ourselves wanting to apply the same set of Interceptors over and over again. Rather than reiterate the same list of Interceptors, we can bundle these Interceptors together using an Interceptor Stack.

xmlstruts.xml<package name="default" extends="struts-default"> <interceptors> <interceptor name="timer" class=".."/> <interceptor name=" logger" class=".."/> <interceptor-stack name="myStack"> <interceptor-ref name="timer"/> <interceptor-ref name="logger"/> </interceptor-stack> < /interceptors> <action name="login" class="tutuorial.Login"> <interceptor-ref name="myStack"/> <result name="input">login.jsp</result> <result name="success" type="redirectAction">/secure/home</result> </action> </package>

Looking inside `struts-default.xml`, we can see how it's done.

The Default Configuration

{snippet:id=all|lang=xml|url=struts2/core/src/main/resources/struts-default.xml}Since the `struts-default.xml` is included in the application's configuration by default, all of the predefined interceptors and stacks are available "out of the box".

Framework Interceptors

Interceptor classes are also defined using a key-value pair specified in the Struts configuration file. The names specified below come specified in struts-default.xml. If you extend the `struts-default` package, then you can use the names below. Otherwise, they must be defined in your package with a name-class pair specified in the <interceptors> tag.

Interceptor	Name	Description
Alias Interceptor	alias	Converts similar parameters that may be named differently between requests.
Chaining Interceptor	chain	Makes the previous Action's properties available to the current Action. Commonly used together with <result type="chain"> (in the previous Action).
Checkbox Interceptor	checkbox	Adds automatic checkbox handling code that detect an unchecked checkbox and add it as a parameter with a default (usually 'false') value. Uses a specially named hidden field to detect unsubmitted checkboxes. The default unchecked value is overridable for non-boolean value'd checkboxes.
Cookie Interceptor	cookie	Inject cookie with a certain configurable name / value into action. (Since 2.0.7.)
CookieProvider Interceptor	cookieProvider	Transfer cookies from action to response (Since 2.3.15.)
Conversion Error Interceptor	conversionError	Adds conversion errors from the ActionContext to the Action's field errors
Create Session Interceptor	createSession	Create an HttpSession automatically, useful with certain Interceptors that require a HttpSession to work properly (like the TokenInterceptor)
DebuggingInterceptor	debugging	Provides several different debugging screens to provide insight into the data behind the page.
Execute and Wait Interceptor	execAndWait	Executes the Action in the background and then sends the user off to an intermediate waiting page.

Exception Interceptor	exception	Maps exceptions to a result.
File Upload Interceptor	fileUpload	An Interceptor that adds easy access to file upload support.
I18n Interceptor	i18n	Remembers the locale selected for a user's session.
Logger Interceptor	logger	Outputs the name of the Action.
Message Store Interceptor	store	Store and retrieve action messages / errors / field errors for action that implements ValidationAware interface into session.
Model Driven Interceptor	modelDriven	If the Action implements ModelDriven, pushes the `getModel` Result onto the Value Stack.
Scoped Model Driven Interceptor	scopedModelDriven	If the Action implements ScopedModelDriven, the interceptor retrieves and stores the model from a scope and sets it on the action calling `setModel`.
Parameters Interceptor	params	Sets the request parameters onto the Action.
Prepare Interceptor	prepare	If the Action implements Preparable, calls its `prepare` method.
Scope Interceptor	scope	Simple mechanism for storing Action state in the session or application scope.
Servlet Config Interceptor	servletConfig	Provide access to Maps representing HttpServletRequest and HttpServletResponse.
Static Parameters Interceptor	staticParams	Sets the `struts.xml` defined parameters onto the action. These are the <param> tags that are direct children of the <action> tag.
Roles Interceptor	roles	Action will only be executed if the user has the correct JAAS role.
Timer Interceptor	timer	Outputs how long the Action takes to execute (including nested Interceptors and View)
Token Interceptor	token	Checks for valid token presence in Action, prevents duplicate form submission.
Token Session Interceptor	tokenSession	Same as Token Interceptor, but stores the submitted data in session when handed an invalid token
Validation Interceptor	validation	Performs validation using the validators defined in *action*-validation.xml
Default Workflow Interceptor	workflow	Calls the `validate` method in your Action class. If Action errors are created then it returns the `INPUT` view.
Parameter Filter Interceptor	N/A	Removes parameters from the list of those available to Actions
Profiling Interceptor	profiling	Activate profiling through parameter
Multiselect Interceptor	multiselect	Like the checkbox interceptor detects that no value was selected for a field with multiple values (like a select) and adds an empty parameter
NoOp Interceptor	noop	Does nothing, just passes invocation further, used in empty stack

Since 2.0.7, Interceptors and Results with hyphenated names were converted to camelCase. (The former model-driven is now modelDriven.) The original hyphenated names are retained as "aliases" until Struts 2.1.0. For clarity, the hyphenated versions are not listed here, but might be referenced in prior versions of the documentation.

Method Filtering

{snippet:id=javadoc|javadoc=true|url=com.opensymphony.xwork2.interceptor.MethodFilterInterceptor}

Interceptor Parameter Overriding

Interceptor's parameter could be overridden through the following ways :

Method 1:

xml<action name="myAction" class="myActionClass"> <interceptor-ref name="exception"/> <interceptor-ref name="alias"/> <interceptor-ref name="params"/> <interceptor-ref name="servletConfig"/> <interceptor-ref name="prepare"/> <interceptor-ref name="i18n"/> <interceptor-ref name="chain"/> <interceptor-ref name="modelDriven"/> <interceptor-ref name="fileUpload"/> <interceptor-ref name="staticParams"/> <interceptor-ref name="params"/> <interceptor-ref name="conversionError"/> <interceptor-ref name="validation"> <param name=" excludeMethods">myValidationExcudeMethod</param> </interceptor-ref> <interceptor-ref name="workflow"> <param name="excludeMethods" >myWorkflowExcludeMethod</param> </interceptor-ref> </action>

Method 2:

xml<action name="myAction" class="myActionClass"> <interceptor-ref name="defaultStack"> <param name="validation.excludeMethods" >myValidationExcludeMethod</param> <param name="workflow.excludeMethods">myWorkflowExcludeMethod</param> </interceptor-ref> < /action>

In the first method, the whole default stack is copied and the parameter then changed accordingly.

In the second method, the `interceptor-ref` refers to an existing interceptor-stack, namely `defaultStack` in this example, and override the `v alidator` and `workflow` interceptor `excludeMethods` attribute. Note that in the `param` tag, the name attribute contains a dot (.) the word before the dot(.) specifies the interceptor name whose parameter is to be overridden and the word after the dot (.) specifies the parameter itself. The syntax is as follows:

<interceptor-name>.<parameter-name>

Note also that in this case the `interceptor-ref` name attribute is used to indicate an interceptor stack which makes sense as if it is referring to the interceptor itself it would be just using Method 1 describe above.

Method 3:

xml<interceptors> <interceptor-stack name="parentStack"> <interceptor-ref name="defaultStack"> <param name="params.excludeParams" >token</param> </interceptor-ref> </interceptor-stack> </interceptors> <default-interceptor-ref name="parentStack"/>

Interceptor Parameter Overriding Inheritance

Parameters override are not inherited in interceptors, meaning that the last set of overridden parameters will be used. For example, if a stack overrides the parameter "defaultBlock" for the "postPrepareParameterFilter" interceptor as:

xml<interceptor-stack name="parentStack"> <interceptor-ref name="postPrepareParameterFilter"> <param name="defaultBlock">true</param> < /interceptor-ref> </interceptor-stack>

and an action overrides the "allowed" for "postPrepareParameterFilter":

xml<package name="child2" namespace="/child" extends="parentPackage"> <action name="list" class="SomeAction"> <interceptor-ref name=" parentStack"> <param name="postPrepareParameterFilter.allowed">myObject.name</param> </interceptor-ref> </action> </package>

Then, only "allowed" will be overridden for the "postPrepareParameterFilter" interceptor in that action, the other params will be null.

Lazy parameters

This functionality was added in Struts 2.5.9

It is possible to define an interceptor with parameters evaluated during action invocation. In such case the interceptor must be marked with `WithL azyParams` interface. This must be developer's decision as interceptor must be aware of having those parameters set during invocation and not when the interceptor is created as it happens in normal way.

Params are evaluated as any other expression starting with from action as a top object.

xml<action name="LazyFoo" class="com.opensymphony.xwork2.SimpleAction"> <result name="success">result.jsp</result> <interceptor-ref name="lazy"> <param name="foo">${bar}</param> </interceptor-ref> </action>javapublic class MockLazyInterceptor extends AbstractInterceptor implements WithLazyParams { private String foo = ""; public void setFoo(String foo) { this.foo = foo; } public String intercept(ActionInvocation invocation) throws Exception { .... return invocation.invoke(); } }

Please be aware that order of interceptors can matter when want to access parameters passed via request as those parameters are set by Parameters Interceptor.

Order of Interceptor Execution

Interceptors provide an excellent means to wrap before/after processing. The concept reduces code duplication (think AOP).

xml<interceptor-stack name="xaStack"> <interceptor-ref name="thisWillRunFirstInterceptor"/> <interceptor-ref name="thisWillRunNextInterceptor" /> <interceptor-ref name="followedByThisInterceptor"/> <interceptor-ref name="thisWillRunLastInterceptor"/> </interceptor-stack>

Note that some Interceptors will interrupt the stack/chain/flow ... so the order is very important.

Interceptors implementing `com.opensymphony.xwork2.interceptor.PreResultListener` will run after the Action executes but before the Result executes.

thisWillRunFirstInterceptor thisWillRunNextInterceptor followedByThisInterceptor thisWillRunLastInterceptor MyAction1 MyAction2 (chain) MyPreResultListener MyResult (result) thisWillRunLastInterceptor followedByThisInterceptor thisWillRunNextInterceptor thisWillRunFirstInterceptor

FAQ

- How do we configure an Interceptor to be used with every Action?
- How do we get access to the session?
- How can we access the HttpServletRequest?
- How can we access the HttpServletResponse?
- How can we access request parameters passed into an Action?
- How do we access static parameters from an Action?
- Can we access an Action's Result?
- How do I obtain security details (JAAS)?
- Why isn't our Prepare interceptor being executed?
- How do we upload files?

Next: Writing Interceptors