**Struts 2 Ek Bilgiler**

**Interceptors**

Interceptors provide a simple way to add processing logic around the method being called on

the action. They allow for cross-functional features to be applied to actions in a convenient

and consistent way, avoiding the need for adding code to each and every action that over time

would create additional maintenance overhead



**The Value Stack and OGNL**

As you saw in Figure 3-2, the *Value Stack* is a central concept in the Struts2 framework. All of

the core components interact with it in one way or another to provide access to context information

as well as to elements of the execution environment.

Underneath, the Value Stack is exactly what it says—a stack implementation. However,

there are differences between a traditional stack implementation and the Value Stack. The first

difference is that the contents of the stack are made up of four levels:

• *Temporary objects*: These objects need temporary storage during the processing of a

request, for example, the current element in a collection that is being iterated over.

• *Model object*: When the action implements the ModelDriven interface, the model object

is placed on the stack in front of the action that is being executed; this level will not be

present if the interface is not implemented by the action.

• *Action object*: This is the action that is currently being executed.

• *Named objects*: Any object can be assigned an identifier, making it a named object.

Named objects can be developer created but also included are #application, #session,

#request, #attr, and #parameters—each corresponding to an equivalent HTTP scoped

object collection.

Another difference is how the stack is used. Traditionally, when using a stack, you would

push objects on and pop objects off when you wanted to use them. With the Value Stack, you

are searching for, or evaluating, a particular expression using OGNL (Object Graph Navigational

Language) syntax.

Like other expression languages, such as JSTL (JSP Standard Tag Library) or MVEL

(MVFLEX Expression Language), OGNL provides a mechanism to navigate object graphs

using a dot notation and evaluate expressions, including calling methods on the objects

being retrieved. Table 3-3 provides some concrete examples of what is possible in an OGNL

expression.

**Results and Result Types**

After an action has been processed, it’s time to send the resulting information back to the user.

In Struts2, this task is split into two parts: the result type and the result itself.

The result type provides the implementation details for the type of information that is

returned to the user. Result types are usually preconfigured in Struts2 (as shown in Table 3-4)

or provided via plug-ins, but developers can provide custom result types as well. Configured

as the default result type is the dispatcher, which uses a JSP to render the response to the user.

After a result type is defined, it can be used many times by different action results.



**OGNL**

|  |  |
| --- | --- |
| **OGNL Example** | **Description** |
| address.postcode | Returns the value of calling getAddress().getPostcode().  This accessor pattern is known as dot notation in that each  property name is separated by a period to provide  navigation within an object graph. |
| #session['user'] | Obtains the user object from the HTTP session. |
| !required | The expression returns true if the call to the isRequired()  method returns false. |
| required && result.size()>1 | Returns the result of performing a logical AND of the result  of calling isRequired() and executing the logic to  determine if the collection result has a size greater than 1. |
| hasActionErrors() | Returns the value from calling the hasActionErrors()  method. |
| [2].id | Calls getId() on the third element of the Value Stack (index  0 is the top). This is most useful when you know the exact  contents of the Value Stack, and the getId() method is  present on objects higher than the index of the object you  are after. |
| top | Returns the object on the top of the Value Stack. |
| results.{name} | Returns a collection consisting of calling getName() on  each of the elements in the collection results. This is  known as *projection.* |
| role in {'admin','user'} | Determines whether the value returned from calling |
| role not in {'admin','user'} | getRole() is either in, or not in, the collection of 'admin'  and 'user'. |
| @com.static.Constants@getRoles() | Returns the value of calling the static method getRoles()  on the class Constants. |
| @com.static.Constants@USER\_NAME | Returns the static property value of USER\_NAME on the class  Constants. |

# Mapping Interceptors To Action

Struts 2 developers are used to declare the actions belong to a package that extend the “**struts-default**“, which contains the default set of interceptors.

|  |
| --- |
| **<package** name="default" namespace="/" extends="struts-default"**>**  **<action** name="testingAction"  class="com.mkyong.common.action.TestingAction" **>**  **<result** name="success"**>**pages/result.jsp**</result>**  **</action>**  **</package>** |

The default set of interceptors are grouped as “**defaultStack**” in **struts-default.xml** file, which is located in the **struts2-core.jar** file. The “**defaultStack**” provides all the core Struts 2 functionality, which is suit the need of most application.

Try study the **struts-default.xml** file, it’s always the best interceptors reference.

|  |
| --- |
| **<package** name="default" namespace="/" extends="struts-default"**>**  **<action** name="testingAction"  class="com.mkyong.common.action.TestingAction" **>**  **<interceptor-ref** name="timer"**/>**  **<interceptor-ref** name="logger"**/>**  **<result** name="success"**>**pages/result.jsp**</result>**  **</action>**  **</package>** |

In above snippet code, it map the “**timer**” and “**logger**” interceptors to the “**TestingAction**” action class via “**interceptor-ref**” element.

The interceptors will fire in the order they’re declared.

Since the “**TestingAction**” is declared it’s own interceptors, **it’s immediate loses all the inherit default set of interceptors**, you must explicitly declare the “**defaultStack**” in order to use it, see below example.

|  |
| --- |
| **<package** name="default" namespace="/" extends="struts-default"**>**  **<action** name="testingAction"  class="com.mkyong.common.action.TestingAction" **>**  **<interceptor-ref** name="timer"**/>**  **<interceptor-ref** name="logger"**/>**  **<interceptor-ref** name="defaultStack"**/>**  **<result** name="success"**>**pages/result.jsp**</result>**  **</action>**  **</package>** |

# Override The Interceptor Parameters

In Struts 2, you can set or override the interceptor parameters via the generic **<param>** tag. See below example

|  |
| --- |
| **<package** name="default" namespace="/" extends="struts-default"**>**  **<action** name="whateverAction"  class="com.mkyong.common.action.WhateverAction" **>**  **<interceptor-ref** name="workflow"**>**  **<param** name="excludeMethods"**>**whateverMethod**</param>**  **</interceptor-ref>**  **<result** name="success"**>**pages/whatever.jsp**</result>**  **</action>**  **</package>** |

However, in above snippet, the action class is declared it’s own interceptor, and it will cause the immediate lose of the inherit “**defaultStack**” interceptors.

What if you want to keep the “**defaultStack**” interceptors, and override the **workflow’s excludeMethods** parameter as well? No problem, try this

|  |
| --- |
| **<package** name="default" namespace="/" extends="struts-default"**>**  **<action** name="whateverAction"  class="com.mkyong.common.action.WhateverAction" **>**  **<interceptor-ref** name="defaultStack"**>**  **<param** name="workflow.excludeMethods"**>**whateverMethod**</param>**  **</interceptor-ref>**  **<result** name="success"**>**pages/whatever.jsp**</result>**  **</action>**  **</package>** |

The if tag can then be used to determine whether the user has logged in, using the

expression #session['user']==null. In Chapter 3, you saw how the Value Stack contains various

objects, including the HTTP Session, which can be referenced using the #session identifier.

In this expression, you are selecting the attribute with the name of “user” and testing to

determine whether it has a null value. If it does, you render the link.

|  |
| --- |
| <s:if test="#session['user']==null">  <s:url id="register" action="findUser" namespace="/user" />  <s:a href="%{register}">Register</s:a>  </s:if> |

To render the link, two more Struts2 tags are used. The first is the url tag. Like the form

tag, it uses an action attribute and a namespace attribute to generate a correct URL, which is

then placed on the Value Stack under the lookup value of the id attribute. The a tag then uses

this value (notice the %{register} value, so that the URL is rendered and not simply the text

“register” rendered). Figure 5-4 shows the starting page with the Register link visible.

If the HTTP Session attribute is not empty, the update link needs to be rendered. The

prepare() method of the action class uses the emailId property to lookup the User domain

object, so you need to find the e-mail address of the user that is currently logged in and supply

this as a request attribute on the URL.

The else tag works in partnership with the if tag from the Register link, so that the

Update link is only rendered when an HTTP Session object is available. Generating and displaying

the link is exactly the same as shown previously. The only addition being the param tag,

which is used within the url tag to provide request attributes. In the code example, the param

tag is used to provide the e-mail name/value pair. Because we need to provide the e-mail

address, and we know that the HTTP Session object for the key “user” is not null, we can

assign the email property directly from the session’s object.

|  |
| --- |
| <s:else>  <s:url id="update" action="findUser" namespace="/user" >  <s:param name="emailId" value="#session['user'].email" />  </s:url>  <s:a href="%{update}">Update Profile</s:a>  </s:else>  The complete page becomes  <%@ taglib uri="/struts-tags" prefix="s" %>  <head>  <title>Home</title>  </head>  <body>  <s:if test="#session['user']==null">  <s:url id="register" action="findUser" namespace="/user" />  <s:a href="%{register}">Register</s:a>  </s:if>  <s:else>  <s:url id="update" action="findUser" namespace="/user" >  <s:param name="emailId" value="#session['user'].email" />  </s:url>  <s:a href="%{update}">Update Profile</s:a>  </s:else> |