# Struts 2 : Interceptors

# Interceptors

- Interceptors play a crucial role
  - A high **level of separation** of concerns.
  - Interceptors remove **cross-cutting tasks from our action components** such a high level of separation of concerns.
  - **Cross-cutting, or preprocessing and postprocessing.**
  - Example
    - Logging (this task must be done for almost every request)
    - File uploading and transferring request parameters onto our action

# Interceptors

- Role of interceptors are clear,
  - instead of having a simple controller invoking an action directly,
  - we now have a **component that sits between** the **controller** and **the action.**
- The invocation of an action is a layered process
  - that always includes the execution of a stack of interceptors **prior to** and **after the actual execution of the action** itself.
- Rather than invoke the **action's execute() method directly**,
  - the framework creates an **object** called an **ActionInvocation** that encapsulates the action and all of the interceptors that have been configured to fire before and after that action executes.

# Interceptors

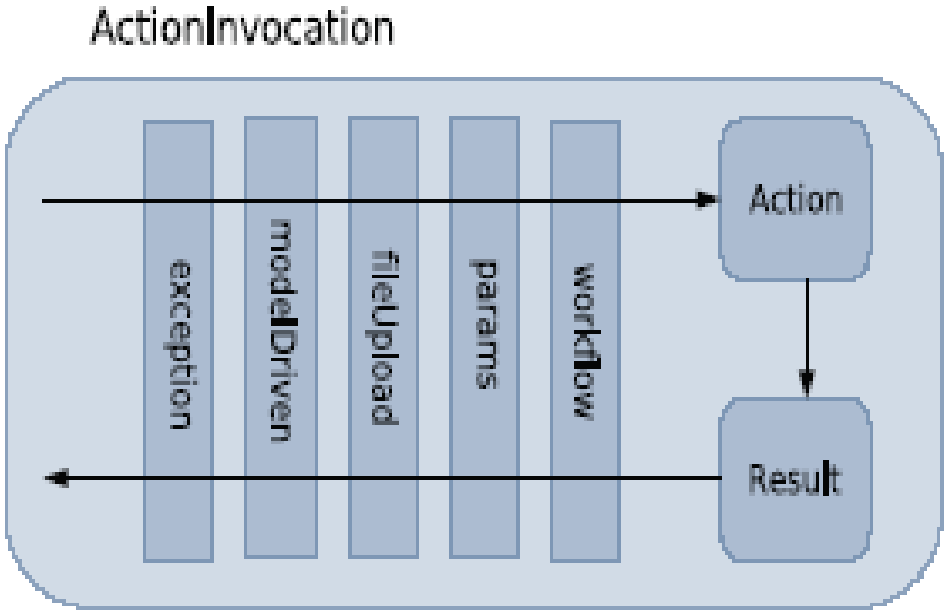


**Figure 4.1** `ActionInvocation` encapsulates the execution of an action with its associated interceptors and results.

The invocation of an action must first travel through the stack of interceptors associated with that action.

# Interceptors

- The ActionInvocation encapsulates all the processing details associated with the execution of a particular action.
  - When the framework receives a request, it first must decide to which action the URL maps.
  - An instance of this action is added to a newly created instance of ActionInvocation.
  - Next, the **framework consults the declarative architecture**, as created by the application's XML or Java annotations, to discover which **interceptors should fire, and in what sequence.**
  - References to these interceptors are added to the ActionInvocation.
  - In addition to these central elements, the ActionInvocation also holds references to other important information like the servlet request objects and a map of the results available to the action.

# How the interceptors fire

- Now that the ActionInvocation has been created and populated with all the objects and information it needs, we can start the invocation.
- The ActionInvocation exposes the **invoke() method**, which is called by the framework to start the execution of the action.
- When the framework calls this method, the ActionInvocation starts the invocation process by executing the first interceptor in the stack.
- Note that the invoke() method doesn't always map to the first interceptor; it's the responsibility of the ActionInvocation itself to keep track of what stage the invocation process has reached and pass control to the appropriate interceptor in the stack. It does this by calling that interceptor's **intercept() method**.

# How the interceptors fire

```
<interceptor-stack name="basicStack">
   <interceptor-ref name="exception"/>
   <interceptor-ref name="servletConfig"/>
   <interceptor-ref name="prepare"/>
   <interceptor-ref name="checkbox"/>
   <interceptor-ref name="params"/>
   <interceptor-ref name="conversionError"/>
</interceptor-stack>
```

# How the interceptors fire

- Continued execution of the subsequent interceptors, and ultimately the action, occurs through recursive calls to the ActionInvocation's invoke() method.
- Each time **invoke()** is called, ActionInvocation consults its state and executes whichever interceptor comes next.
- When all of the interceptors have been invoked, the invoke() method will cause the action itself to be executed

# How the interceptors fire

- Now let's look at what an interceptor can do when it fires. An interceptor has a three-stage, conditional execution cycle:
  - Do some preprocessing.
  - Pass control on to successive interceptors, and ultimately the action, by calling invoke(), or divert execution by itself returning a control string.
  - Do some postprocessing.

# How the interceptors fire

```
public String intercept(ActionInvocation invocation) throws Exception {

   long startTime = System.currentTimeMillis();      (1)

   String result = invocation.invoke();      (2)

   long executionTime = System.currentTimeMillis() - startTime;

     ... log the time ...                                              (3)

   return result;
}
```

# Built-in Struts 2 interceptors

- Struts 2 comes with a powerful set of built-in interceptors that provide most of the functionality you'll ever want from a web framework.
- Based on functionalities the interceptors have been grouped as below,
  - **Utility** interceptors
  - **Data transfer** interceptors
  - **Workflow** interceptors
  - **Miscellaneous** interceptors

# Utility interceptors

- These interceptors provide simple utilities to aid in development, tuning, and troubleshooting.
  - TIMER
    - This simple interceptor merely records the duration of an execution. Position in the interceptor stack determines what this is actually timing.

      INFO: Executed action [/chapterFour/secure/ImageUpload!execute] took 123 ms.
  - LOGGER
    - This interceptor provides a simple logging mechanism that logs an entry statement during preprocessing and an exit statement during postprocessing.

      INFO: Starting execution stack for action /chapterFour/secure/ImageUpload

      INFO: Finishing execution stack for action /chapterFour/secure/ImageUpload
    - This can be useful for debugging.

# Data transfer interceptors

- Interceptors can be used to handle data transfer.
  - PARAMS (DEFAULTSTACK)
    - It transfers the request parameters to properties exposed by the ValueStack.
  - STATIC-PARAMS (DEFAULTSTACK)
    - This interceptor also moves parameters onto properties exposed on the ValueStack. The difference is the origin of the parameters.
    - The parameters that this interceptor moves are defined in the action elements of the declarative architecture.
    - For example, suppose you have an action defined like this in one of your declarative architecture XML files:

```
<action name="exampleAction" class="example.ExampleAction">
        <param name="firstName">John</param>
        <param name="lastName">Doe</param>
</action>
```

# Data transfer interceptors

- AUTOWIRING
  - This interceptor provides an integration point for using Spring to manage your application resources.

- SERVLET-CONFIG ( DEFAULTSTACK )
  - The servlet-config interceptor provides a clean way of injecting various objects from the Servlet API into your actions.
  - This interceptor works by setting the various objects on setter methods exposed by interfaces that the action must implement.
  - The following interfaces are available for retrieving various objects related to the servlet environment.

# Data transfer interceptors

- Your action can implement any number of these.
  - ServletContextAware—Sets the ServletContext
  - ServletRequestAware—Sets the HttpServletRequest
  - ServletResponseAware—Sets the HttpServletResponse
  - ParameterAware—Sets a map of the request parameters
  - RequestAware—Sets a map of the request attributes
  - SessionAware—Sets a map of the session attributes
  - ApplicationAware—Sets a map of application scope properties

- FILEUPLOAD ( DEFAULTSTACK )
  - The fileUpload interceptor transforms the files and metadata from multipart requests into regular request parameters so that they can be set on the action just like normal parameters.

# Workflow interceptors

- WORKFLOW (DEFAULTSTACK)
  - It works with our actions to provide data validation and subsequent workflow alteration if a validation error occurs.
- VALIDATION (DEFAULTSTACK)
  - The validation interceptor, on the other hand, is part of the Struts 2 validation framework and provides a declarative means to validate your data.
  - Rather than writing validation code, the validation framework allows you to use both XML files and Java annotations to describe the validation rules for your data.
- PREPARE (DEFAULTSTACK)
  - The prepare interceptor provides a generic entry point for arbitrary workflow processing that you might want to add to your actions. The concept is simple. When the prepare interceptor executes, it looks for a prepare() method on your action.

# Workflow interceptors

- MODELDRIVEN (DEFAULTSTACK)
  - The modelDriven interceptor is considered a workflow interceptor because it alters the workflow of the execution by invoking getModel(), if present, and setting the model object on the top of the ValueStack where it'll receive the parameters from the request.
  - This alters workflow because the transfer of the parameters, by the params interceptor, would otherwise be directed onto the action object itself.
  - By placing the model over the action in the ValueStack, the modelDriven interceptor thus alters workflow.
  - This concept of creating an interceptor that can conditionally alter the effective functionality of another interceptor without direct programmatic intervention demonstrates the power of the layered interceptor architecture.

# Miscellaneous interceptors

- EXCEPTION (DEFAULTSTACK)
  - This important interceptor lays the foundation for rich exception handling in your applications.
  - The exception interceptor comes first in the defaultStack, and should probably come first in any custom stacks you create yourself.
  - The exception interceptor will catch exceptions and map them, by type, to user-defined error pages.
  - Its position at the top of the stack guarantees that it'll be able to catch all exceptions that may be generated during all phases of action invocation.
  - It can catch them because, as the top interceptor, it'll be the last to fire during postprocessing.

# Miscellaneous interceptors

- TOKEN AND TOKEN-SESSION
  - The token and token-session interceptors can be used as part of a system to prevent duplicate form submissions.
  - Duplicate form posts can occur when users click the Back button to go back to a previously submitted form and then click the Submit button again, or when they click Submit more than once while waiting for a response.
  - The token interceptors work by passing a token in with the request that is checked by the interceptor.
  - If the unique token comes to the interceptor a second time, the request is considered a duplicate.
  - These two interceptors both do the same thing, differing only in how richly they handle the duplicate request.
  - You can either show an error page or save the original result to be rendered for the user.

# Miscellaneous interceptors

- SCOPED-MODELDRIVEN (DEFAULTSTACK)
  - This nice interceptor supports wizard-like persistence across requests for your action's model object.
  - This one adds to the functionality of the modelDriven interceptor by allowing you to store your model object in, for instance, session scope.
- EXECANDWAIT
  - When a request takes a long time to execute, it's nice to give the user some feedback.
  - While the token interceptors discussed earlier can technically solve this problem, we should still do something for the user.
  - The execAndWait interceptor helps prevent your users from getting antsy.

# Declaring interceptors

- Since most of the interceptors that you'll typically need are provided by the struts-default package, we need to see the interceptor declarations made in the strutsdefault xml file.
- Basically, interceptor declarations consist of declaring the interceptors that are available and associating them with the actions for which they should fire.
- The only complication is the creation of stacks, which allow you to reference groups of interceptors all at once.
- Interceptor declarations, like declarations of all framework components, must be contained in a package element.

# Declaring interceptors

```
<package name="struts-default">

  . . .

  <interceptors>
    <interceptor name="execAndWait" class="ExecuteAndWaitInterceptor"/>
    <interceptor name="exception" class="ExceptionMappingInterceptor"/>
    <interceptor name="fileUpload" class="FileUploadInterceptor"/>
    <interceptor name="i18n" class="I18nInterceptor"/>
    <interceptor name="logger" class="LoggingInterceptor"/>
    <interceptor name="modelDriven" class="ModelDrivenInterceptor"/>
    <interceptor name="scoped-modelDriven" class= . . ./>
    <interceptor name="params" class="ParametersInterceptor"/>
    <interceptor name="prepare" class="PrepareInterceptor"/>
    <interceptor name="static-params" class=. . ./>
    <interceptor name="servlet-config" class="ServletConfigInterceptor"/>
    <interceptor name="sessionAutowiring"
                 class="SessionContextAutowiringInterceptor"/>
    <interceptor name="timer" class="TimerInterceptor"/>
    <interceptor name="token" class="TokenInterceptor"/>
    <interceptor name="token-session" class= . . . />
    <interceptor name="validation" class= . . . />
    <interceptor name="workflow" class="DefaultWorkflowInterceptor"/>
    . . .
```

❶ Interceptors element

❷ All interceptor elements

# Declaring interceptors

```
      <interceptor-stack name="defaultStack">
        <interceptor-ref name="exception"/>
        <interceptor-ref name="alias"/>
        <interceptor-ref name="servlet-config"/>
        <interceptor-ref name="prepare"/>
        <interceptor-ref name="i18n"/>
        <interceptor-ref name="chain"/>
        <interceptor-ref name="debugging"/>
        <interceptor-ref name="profiling"/>
        <interceptor-ref name="scoped-modelDriven"/>
        <interceptor-ref name="modelDriven"/>
        <interceptor-ref name="fileUpload"/>
        <interceptor-ref name="checkbox"/>
        <interceptor-ref name="static-params"/>
        <interceptor-ref name="params">
          <param name="excludeParams">dojo\..*</param>
        </interceptor-ref>
        <interceptor-ref name="conversionError"/>
        <interceptor-ref name="validation">
           <param name="excludeMethods">input,back,cancel,browse</param>
        </interceptor-ref>
        <interceptor-ref name="workflow">
          <param name="excludeMethods">input,back,cancel,browse</param>
        </interceptor-ref>
      </interceptor-stack>
    </interceptors>
    <default-interceptor-ref name="defaultStack"/>
</package>
```

3 Declaring a stack

Interceptor references 4

5 Parameters

6 Default reference

# Declaring interceptors

- XML DOCUMENT STRUCTURE
  - Before we see how we can specify the interceptors that'll fire for your specific actions, we should make a point about the sequence of elements within the XML documents we use for declarative architecture.
  - These XML documents must conform to certain rules of ordering.
  - For instance, each package element contains precisely one interceptors element, and that element must come in a specific position in the document.
  - The complete DTD, struts-2.0.dtd, can be found on the Struts 2 website.
  - For now, note the following snippet from the DTD, which pertains to the structure of listing 4.2

# Declaring interceptors

```
<!ELEMENT struts (package|include|bean|constant)*>

<!ELEMENT package (result-types?, interceptors?, default-interceptor-ref?,
default-action-ref?, global-results?, global-exception-mappings?, action*)>
```

# Mapping interceptors to actions

- Much of the time, your actions will belong to packages that extend struts-default, and you'll be content to let them use the defaultStack of interceptors they inherit from that package.
- Eventually, you'll probably want to modify, change, or perhaps just augment that default set of interceptors.
- To do this, you have to know how to map interceptors to your actions.
- **Associating an interceptor to an action is done with an interceptorref element.**
- The following code snippet shows how to associate a set of interceptors with a specific action:

# Mapping interceptors to actions

```
<action name="MyAction" class="org.actions.myactions.MyAction">
  <interceptor-ref name="timer"/>
  <interceptor-ref name="logger"/>
  <result>Success.jsp</result>
</action>
```

```
<action name="MyAction" class="org.actions.myactions.MyAction">
  <interceptor-ref name="timer"/>
  <interceptor-ref name="logger"/>
  <interceptor-ref name="defaultStack"/>
  <result>Success.jsp</result>
</action>
```

# Building your own interceptor

- ***Implementing the Interceptor interface***
  - When you write an interceptor, you'll implement the com.opensymphony.xwork2.interceptor.Interceptor interface.

```
public interface Interceptor extends Serializable {
   void destroy();
   void init();
   String intercept(ActionInvocation invocation) throws Exception;
}
```

  - Interface defines only three methods.
  - The first two are typical lifecycle methods that give you a chance to initialize and clean up resources as necessary.
  - The real business occurs in the **intercept() method**. This **method is called by the recursive ActionInvocation.invoke() method.**

# Building your own interceptor

- ***Implementing the Interceptor interface***
  - If you want to write an interceptor that has this type of parameterization, you can extend com.opensymphony.xwork2.interceptor.MethodFilterInterceptor rather than directly implementing the Interceptor interface.

# Example : AuthenticationInterceptor

- The authentication interceptor will be simple.
- If you recall the **three phases of interceptor** processing
  - Preprocessing
  - Calling ActionInvocation.invoke()
  - Postprocessing
- You can anticipate how our AuthenticationInterceptor will function.

# Example : AuthenticationInterceptor

- When a request comes to one of our secure actions, we'll want to check whether the request is coming from an authenticated user.
- This check is made during preprocessing.
  - If the user has been authenticated, the interceptor will call invoke(), thus allowing the action invocation to proceed.
  - If the user hasn't been authenticated, the interceptor will return a control string itself, thus barring further execution.
  - The control string will route the user to the login page.

# Example : AuthenticationInterceptor

- Struts 2 Portfolio application.
  - On the home page, there's a link to add an image without having logged in.
  - The add image action is a secure action.
  - Try clicking the link without having logged in.
  - You'll be automatically taken to the login page.
  - Now, log in and try the same link again.
  - The application comes with a default user, username = "Ravi" and password = "ravi".
  - This time you're allowed to access the secure add image action.
  - This is done by a custom interceptor that we've placed in front of all of our secure actions.

# Example : AuthenticationInterceptor

- Roles of the AuthenticationInterceptor
  - It doesn't do the authentication.
  - It just bars access to secure actions by unauthenticated users.
  - Authentication itself is done by the login action.
  - The login action checks to see whether the username and password are valid.
  - If they are, the user object is stored in a session-scoped map.
  - When the AuthenticationInterceptor fires, it checks to see whether the user object is present in the session.
  - If it is, it lets the action fire as usual.
  - If it isn't, it diverts workflow by forwarding to the login page.

# Example : AuthenticationInterceptor

**Listing 4.3 The Login action authenticates the user and stores the user in session scope**

```
public class Login extends ActionSupport implements SessionAware {        (1)
  public String execute(){
    User user = getPortfolioService().authenticateUser( getUsername(),
       getPassword() );                                                    (2)
    if ( user == null )
    {                                                                      (3)
      return INPUT;
    }
    else{
      session.put( Struts2PortfolioConstants.USER, user );                 (4)
    }
    return SUCCESS;
  }
  . . .
  public void setSession(Map session) {                                    (5)
    this.session = session;
  }
}
```

# Example : AuthenticationInterceptor

**Listing 4.4 Inspecting the heart of the AuthenticationInterceptor**

```
public class AuthenticationInterceptor implements Interceptor {

   public void destroy() {
   }

   public void init() {
   }

   public String intercept( ActionInvocation actionInvocation )
                                    throws Exception{

      Map session = actionInvocation.getInvocationContext().getSession();
      User user = (User) session.get( Struts2PortfolioConstants.USER );

      if (user == null) {
         return Action.LOGIN;
      }
      else {

         Action action = ( Action ) actionInvocation.getAction();

         if (action instanceof UserAware) {
            ((UserAware)action).setUser(user);
         }

         return actionInvocation.invoke();
       }
   }

}
```

Implements interceptor

Empty implementations

1

3

4

5 Continue action invocation

# Example : AuthenticationInterceptor

**Listing 4.5 Declaring our interceptor and building a new default stack**

```
<package name="chapterFourSecure" namespace="/chapterFour/secure"
 extends="struts-default">

  <interceptors>                                          (1)

    <interceptor name="authenticationInterceptor"                (2)
      class="manning.utils.AuthenticationInterceptor"/>

    <interceptor-stack name="secureStack">                       (3)
      <interceptor-ref name="authenticationInterceptor"/>
      <interceptor-ref name="defaultStack"/>
    </interceptor-stack>

  </interceptors>

  <default-interceptor-ref name="secureStack"/>           (4)

   . . .
</package>
```

# Example : AuthenticationInterceptor

- interceptors element
  - contain our interceptor and interceptor-stack declarations.
- interceptor element
  - to map our Java class to a logical name.
- interceptor-stack
  - we build a new stack that takes the defaultStack and adds our new interceptor to the top of it.
  - We put it on top because we might as well stop an unauthenticated request as soon as possible.
- default-interceptor-ref
  - we declare our new secure- Stack as the default stack for the package.
  - Note that the default-interceptor-ref element isn't contained in the interceptors element; it doesn't declare any interceptors, it just declares the default value for the package.

# Example : AuthenticationInterceptor

- The interceptor starts inside the intercept() method
- Here we can see that the interceptor uses the ActionInvocation object to obtain information pertaining to the request.
- We’re getting the session map. With the session map in hand, we retrieve the user object stored under the known key.
- If the user object is null,
  - then the user hasn’t been authenticated through the login action.
  - At this point, we return a result string, without allowing the action to continue. This result string, Action.LOGIN, points to our login page.

# Example : AuthenticationInterceptor

- If the user object exists,
  - then the user has already logged in.
  - At this point, we get a reference to the current action from the ActionInvocation and check whether it implements the UserAware interface. This interface allows actions to have the user object automatically injected into a setter method.
- With the business of authentication out of the way, the interceptor calls invoke() on the ActionInvocation object to pass control on to the rest of the interceptors and the action.