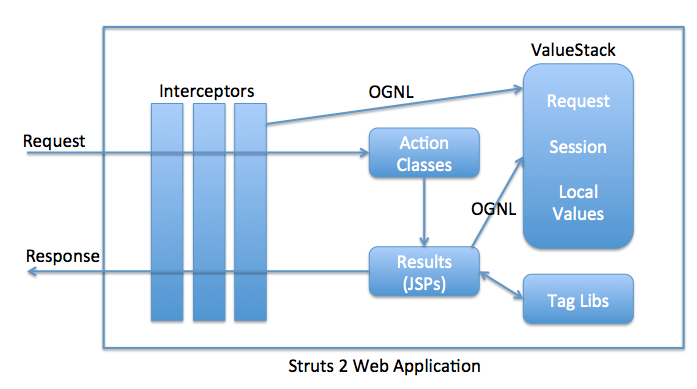
# Struts 2 :



### What is interceptor in Struts2?

Interceptors are the backbone of Struts2 Framework. Struts2 interceptors are responsible for most of the processing done by the framework, such as passing request params to action classes, making Servlet API request, response, session available to Action classes, validation, i18n support, etc.

ActionInvocation is responsible to incapsulate Action classes and interceptors and to fire them in order. The most important method for use in ActionInvocation is invoke() method that keeps track of the interceptor chain and invokes the next interceptor or action. This is one of the best example of Chain of Responsibility pattern in Java EE frameworks.

### What are different ways to create Action classes in Struts2?

Struts2 provide different ways to create action classes.

1. By implementing Action interface
2. Using Struts2 @Action annotation
3. By extending ActionSupport class
4. Any normal java class with execute() method returning String can be configured as Action class.

### What is ValueStack and OGNL?

ValueStack is the storage area where the application data is stored by Struts2 for processing the client requests. The data is stored in ActionContext objects that use ThreadLocal to have values specific to the particular request thread.

Object-Graph Navigation Language (OGNL) is a powerful Expression Language that is used to manipulate data stored on the ValueStack. As you can see in architecture diagram, both interceptors and result pages can access data stored on ValueStack using OGNL.

### Name some useful annotations introduced in Struts2?

Some of the important annotations introduced in Struts2 are:

* 1. @Action to create action class
  2. @Actions to configure single class for multiple actions
  3. @Namespace and @Namespaces for creating different modules
  4. @Result for result pages
  5. @ResultPath for configuring result pages location

### Provide some important Struts2 constants that you have used?

Some of the Struts2 constants that I have used are:

* 1. **struts.devMode** to run our application in development mode. This mode does reload properties files and provides extra logging and debugging feature. It’s very useful while developing our application but we should turn it off while moving our code to production.
  2. **struts.convention.result.path** to configure the location of result pages. By default Struts2 look for result pages at {WEBAPP-ROOT}/{Namespace}/ and we can change the location with this constant.
  3. **struts.custom.i18n.resources** to define global resource bundle for i18n support.
  4. **struts.action.extension** to configure the URL suffix to for Struts2 application. Default suffix is .action but sometimes we might want to change it to .do or something else.

We can configure above constants in struts.xml file like below.

<constant name="struts.devMode" value="true"></constant>

<constant name="struts.action.extension" value="action,do"></constant>

<constant name="struts.custom.i18n.resources" value="global"></constant>

<constant name="struts.convention.result.path" value="/"></constant>

### What is the use of namespace in action mapping in Struts2?

Struts2 namespace configuration allows us to create modules easily. We can use namespace to separate our action classes based on their functionality, for example admin, user, customer etc.

### Which interceptor is responsible for mapping request parameters to action class Java Bean properties?

com.opensymphony.xwork2.interceptor.ParametersInterceptor interceptor is responsible for mapping request parameters to the Action class java bean properties. This interceptor is configured in struts-default package with name “params”. This interceptor is part of basicStack and defaultStack interceptors stack.

### Which interceptor is responsible for i18n support?

com.opensymphony.xwork2.interceptor.I18nInterceptor interceptor is responsible for i18n support in Struts2 applications. This interceptor is configured in struts-default package with name “i18n” and it’s part of i18nStack and defaultStack.

### What is the difference in using Action interface and ActionSupport class for our action classes, which one you would prefer?

We can implement Action interface to create our action classes. This interface has a single method execute() that we need to implement. The only benefit of using this interface is that it contains some constants that we can use for result pages, these constants are SUCCESS, ERROR, NONE, INPUT and LOGIN.

ActionSupport class is the default implementation of Action interface and it also implements interfaces related to Validation and i18n support. ActionSupport class implements Action, Validateable, ValidationAware, TextProvider and LocaleProvider interfaces. We can override validate() method of ActionSupport class to include field level validation login in our action classes.

Depending on the requirements, we can use any of the approaches to create struts 2 action classes, my favorite is ActionSupport class because it helps in writing validation and i18n logic easily in action classes.

### How can we get Servlet API Request, Response, HttpSession etc Objects in action classes?

Struts2 action classes doesn’t provide direct access to Servlet API components such as Request, Response and Session. However sometimes we need these access in action classes such as checking HTTP method or setting cookies in response.

Thats why Struts2 API provides a bunch of \*Aware interfaces that we can implement to access these objects. Struts2 API uses dependency injection to inject Servlet API components in action classes. Some of the important Aware interfaces are SessionAware, ApplicationAware, ServletRequestAware and ServletResponseAware.

You can read more about them in How to get [Servlet API Session in Struts2 Action Classes](http://www.journaldev.com/2203/get-servlet-session-request-response-context-attributes-struts-2-action) tutorial.

# How to get Servlet Session, Request, Response, Context Attributes in Struts 2 Action

[**Struts 2 Action**](http://www.journaldev.com/2173/struts-2-actions-example-tutorial) classes doesn’t provide any methods to get Servlet API Request, Response, Session and attributes. But sometimes we need to access these in action classes, for example checking HTTP method or to work with session attributes or to set cookies or headers in response.

That’s why Struts 2 API provides a bunch of interfaces that we can implement in action classes to inject these objects in Action classes. All of these interfaces end with “Aware” and defined in org.apache.struts2.interceptor package.

All of these interfaces declares setter methods through which Struts 2 API injects **[Servlet API](http://www.journaldev.com/2114/servlet-jsp-tutorial" \o "Java Servlet JSP Tutorial with Example Programs)** components in action classes. It’s a great example of [**Dependency Injection**](http://www.journaldev.com/2394/java-dependency-injection-design-pattern-example-tutorial) in Java EE frameworks.

These \*Aware interfaces are:

1. **SessionAware**: This interface provides access to session attributes in action classes and declare a single method setSession(Map sessionAttributes). Note that we can’t get HttpSession by implementing this interface, this is just to get access to the session attributes.
2. **ApplicationAware**: This interface provides access to context attributes in action classes as Map. We can add attributes to application context by putting values in the context map. This interface declares single method setApplication(Map applicationAttributes).
3. **RequestAware**: This interface provides access to request attributes in action classes, it contains single method setRequest(Map requestAttr). This interface is only applicable if Action classes are used in Servlet environment. Since this interface makes the Action tied to a servlet environment, so it should be avoided if possible since things like unit testing will become more difficult.
4. **ServletRequestAware**: We can implement this interface in action classes to get access to HttpServletRequest object. This interface is only relevant if the Action is used in a servlet environment. It defines a single method as setServletRequest(HttpServletRequest request).
5. **ServletResponseAware**: Struts 2 action classes can implement this interface to get access to the HttpServletResponse object. We can then use response object to add headers or cookies. It defines a single method as setServletResponse(HttpServletResponse response).
6. **CookiesAware**: This interface is intended to provide access to cookies in request in the form of Map. It contains single method setCookiesMap(Map cookies). To work with this interface, there are two interceptors defined in struts-default package as:
7. <interceptor name="cookie" class="org.apache.struts2.interceptor.CookieInterceptor"/>

<interceptor name="cookieProvider" class="org.apache.struts2.interceptor.CookieProviderInterceptor"/>

But they are not part of defaultStack interceptors stack, so we need to include them for action class like below.

<action name="home" class="com.journaldev.struts2.actions.HomeAction">

<interceptor-ref name="cookie"></interceptor-ref>

<interceptor-ref name="cookieProvider"></interceptor-ref>

<interceptor-ref name="defaultStack"></interceptor-ref>

<result name="success">/home.jsp</result>

</action>

However these interceptors are very new and in my testing with Struts 2 version 2.3.15.1, I didn’t find it to be working. I looked into Struts 2 API docs but didn’t find any help on this. I will update the post if I find anything or if you know what is missing, please let us know through comments. The workaround is to use ServletRequestAware and ServletResponseAware interface to get the request cookies or to set cookies in response.

1. **PrincipalAware**: We can implement this interface in action class to get Principal information from HttpServletRequest object. This interface works with PrincipalProxy to provide user id, principal details.

### What is the use of execAndWait interceptor?

Struts2 provides execAndWait interceptor for long running action classes. We can use this interceptor to return an intermediate response page to the client and once the processing is finished, final response is returned to the client. This interceptor is defined in the struts-default package and implementation is present in ExecuteAndWaitInterceptor class.

Check out [Struts2 execAndWait interceptor example](http://www.journaldev.com/2296/struts2-execandwait-interceptor-example-for-long-running-actions) to learn more about this interceptor and how to use it.

### What is the use of token interceptor in Struts2?

One of the major problems with web applications is the double form submission. If not taken care, double form submission could result in charging double amount to customer or updating database values twice. We can use token interceptor to solve the double form submission problem. This interceptor is defined in struts-default package but it’s not part of any interceptor stack, so we need to include it manually in our action classes.

Read more at [Struts2 token interceptor](http://www.journaldev.com/2281/struts2-token-interceptor-example) example.

### How can we upload files in Struts2 application?

File Upload is one of the common task in a web application. Thats why Struts2 provides built in support for file upload through FileUploadInterceptor. This interceptor is configured in struts-default package and provide options to set the maximum size of a file and file types that can be uploaded to the server.

Read more about FileUpload interceptor at [Struts2 File Upload Example](http://www.journaldev.com/2192/struts-2-file-upload-example).

### What is the default suffix for Struts2 action URI and how can we change it?

The default URI suffix for Struts2 action is .action, in Struts1 default suffix was .do. We can change this suffix by defining struts.action.extension constant value in our Struts2 configuration file as:

<constant name="struts.action.extension" value="action,do"></constant>

### What is the default location of result pages and how can we change it?

By default Struts2 looks for result pages in {WEBAPP-ROOT}/{Namespace}/ directory but sometimes we want to keep result pages in another location, we can provide struts.convention.result.path constant value in Struts2 configuration file to change the result pages location.

Another way is to use @ResultPath annotation in action classes to provide the result pages location.

### What is life cycle of an interceptor?

Interceptor interface defines three methods – init(), destroy() and intercept(). init and destroy are the life cycle methods of an interceptor. Interceptors are Singleton classes and Struts2 initialize a new thread to handle each request. init() method is called when interceptor instance is created and we can initialize any resources in this method. destroy() method is called when application is shutting down and we can release any resources in this method.

intercept() is the method called every time client request comes through the interceptor.

### What is an interceptor stack?

An interceptor stack helps us to group together multiple interceptors in a package for further use. struts-default package creates some of the mostly used interceptor stack – basicStack and defaultStack. We can create our own interceptor stack at the start of the package and then configure our action classes to use it.

In the web.xml file, Struts defines its FilterDispatcher, the Servlet Filter class that initializes the Struts framework and handles all requests. This filter can contain initialization parameters that affect what, if any, additional configuration files are loaded and how the framework should behave.

<filter>

        <filter-name>struts2</filter-name>

        <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>

    </filter>

**Model Driven**

Struts 2 does not have "forms" like Struts 1 did. In Struts 2 request parameters are bound directly to fields in the actions class, and this class is placed on top of the stack when the action is executed.

If an action class implements the interface com.opensymphony.xwork2.ModelDriven then it needs to return an object from the getModel() method. Struts will then populate the fields of this object with the request parameters, and this object will be placed on top of the stack once the action is executed. Validation will also be performed on this model object, instead of the action. Please read about [VisitorFieldValidator Annotation](https://struts.apache.org/docs/visitorfieldvalidator-annotation.html) which can help you validate model's fields.

## Interceptor

To use ModelDriven actions, make sure that the [Model Driven Interceptor](https://struts.apache.org/docs/model-driven-interceptor.html) is applied to your action. This interceptor is part of the default interceptor stack defaultStack so it is applied to all actions by default.

## Example

Action class:

|  |
| --- |
| public class ModelDrivenAction implements ModelDriven {      public String execute() throws Exception {          return SUCCESS;      }        public Object getModel() {          return new Gangster();      }  } |

Gangster class (model):

|  |
| --- |
| public class Gangster implements Serializable {      private String name;      private int age;      private String description;      private boolean bustedBefore;        public int getAge() {          return age;      }      public void setAge(int age) {          this.age = age;      }      public boolean isBustedBefore() {          return bustedBefore;      }      public void setBustedBefore(boolean bustedBefore) {          this.bustedBefore = bustedBefore;      }      public String getDescription() {          return description;      }      public void setDescription(String description) {          this.description = description;      }      public String getName() {          return name;      }      public void setName(String name) {          this.name = name;      }  } |

JSP for creating a Gangster:

|  |
| --- |
| <s:form action="modelDrivenResult" method="POST" namespace="/modelDriven">      <s:textfield label="Gangster Name" name="name" />      <s:textfield label="Gangster Age"  name="age" />      <s:checkbox  label="Gangster Busted Before" name="bustedBefore" />      <s:textarea  cols="30" rows="5" label="Gangster Description" name="description" />      <s:submit />  </s:form> |

To create a ModelDriven Action our Action class should implement the **ModelDriven interface** and should include the**modelDriven interceptor**. The modelDriven interceptor is already included in the default stack.

The next step is to implement the **getModel()** method in such a way that it returns the application domain object, in our example we return the User object.

**Action Chaining:**

**As a rule, Action Chaining is not recommended**

The [Chain Result](https://struts.apache.org/docs/chain-result.html) is a result type that invokes an Action with its own Interceptor Stack and Result. This Interceptor allows an Action to forward requests to a target Action, while propagating the state of the source Action. Below is an example of how to define this sequence.

|  |
| --- |
| <package name="public" extends="struts-default">      <!-- Chain creatAccount to login, using the default parameter -->      <action name="createAccount" class="...">          <result type="chain">login</result>      </action>        <action name="login" class="...">          <!-- Chain to another namespace -->          <result type="chain">              <param name="actionName">dashboard</param>              <param name="namespace">/secure</param>          </result>      </action>  </package>    <package name="secure" extends="struts-default" namespace="/secure">      <action name="dashboard" class="...">          <result>dashboard.jsp</result>      </action>  </package> |

**The redirect-after-post technique is a common** pattern in web application development. It simply means making your Action, after it has successfully executed, result in a redirect. You can do this by using the [Redirect Result](https://struts.apache.org/docs/redirect-result.html) or the [Redirect Action Result](https://struts.apache.org/docs/redirect-action-result.html).

[Redirect Result](https://struts.apache.org/docs/redirect-result.html):

Calls the {@link HttpServletResponse#sendRedirect(String) sendRedirect} method to the location specified. The response is told to redirect the browser to the specified location (a new request from the client). The consequence of doing this means that the action (action instance, action errors, field errors, etc) that was just executed is lost and no longer available. This is because actions are built on a single-thread model. The only way to pass data is through the session or with web parameters (url?name=value) which can be OGNL expressions.

# Parameters

* **location (default)** - the location to go to after execution.
* **parse** - true by default. If set to false, the location param will not be parsed for Ognl expressions.
* **anchor** - Optional. Also known as "fragment" or colloquially as "hash". You can specify an anchor for a result.

This result follows the same rules from StrutsResultSupport.

# Examples

|  |
| --- |
| <!--    The redirect URL generated will be:    /foo.jsp#FRAGMENT  -->  <result name="success" type="redirect">    <param name="location">foo.jsp</param>    <param name="parse">false</param>    <param name="anchor">FRAGMENT</param>  </result> |
| <package name="passingRequestParameters" extends="struts-default" namespace="/passingRequestParameters">     <-- Pass parameters (reportType, width and height) -->     <!--     The redirect url generated will be - the namespace of current acction will be appended as location doesn't start with "/":     /passingRequestParameters/generateReport.jsp?reportType=pie&width=100&height=100#summary     -->     <action name="gatherReportInfo" class="...">        <result name="showReportResult" type="redirect">           <param name="location">generateReport.jsp</param>           <param name="reportType">pie</param>           <param name="width">100</param>           <param name="height">100</param>           <param name="parse">false</param>           <param name="anchor">summary</param>        </result>     </action>  </package> |

**Redirect Action Result:**

This result uses the ActionMapper provided by the ActionMapperFactory to redirect the browser to a URL that invokes the specified action and (optional) namespace. This is better than the ServletRedirectResult because it does not require you to encode the URL patterns processed by the ActionMapper in to your struts.xml configuration files. This means you can change your URL patterns at any point and your application will still work. It is strongly recommended that if you are redirecting to another action, you use this result rather than the standard redirect result.

See examples below for an example of how request parameters could be passed in.

See [ActionMapper](https://struts.apache.org/docs/actionmapper.html) for more details

# Parameters

* **actionName (default)** - The name of the action that will be redirected to.
* **namespace** - Used to determine which namespace the action is in that we're redirecting to. If namespace is null, the default will be the current namespace.
* **suppressEmptyParameters** - Optional boolean (defaults to false) that can prevent parameters with no values from being included in the redirect URL.
* **parse** - Boolean, true by default. If set to false, the actionName param will not be parsed for Ognl expressions.
* **anchor** - Optional. Also known as "fragment" or colloquially as "hash". You can specify an anchor for a result.

# Examples

|  |
| --- |
| <package name="public" extends="struts-default">      <action name="login" class="...">          <!-- Redirect to another namespace -->          <result type="redirectAction">              <param name="actionName">dashboard</param>              <param name="namespace">/secure</param>          </result>      </action>  </package>    <package name="secure" extends="struts-default" namespace="/secure">      <-- Redirect to an action in the same namespace -->      <action name="dashboard" class="...">          <result>dashboard.jsp</result>          <result name="error" type="redirectAction">error</result>      </action>        <action name="error" class="...">          <result>error.jsp</result>      </action>  </package>    <package name="passingRequestParameters" extends="struts-default" namespace="/passingRequestParameters">     <!-- Pass parameters (reportType, width and height) -->     <!--     The redirectAction url generated will be :     /genReport/generateReport.action?reportType=pie&amp;width=100&amp;height=100#summary     -->     <action name="gatherReportInfo" class="...">        <result name="showReportResult" type="redirectAction">           <param name="actionName">generateReport</param>           <param name="namespace">/genReport</param>           <param name="reportType">pie</param>           <param name="width">100</param>           <param name="height">100</param>           <param name="empty"></param>           <param name="suppressEmptyParameters">true</param>           <param name="anchor">summary</param>        </result>     </action>  </package> |
| <!--      Example of "anchor" param usage in conjunction with "redirectAction" result-type.        Generated URL: /displayReport.action#SUMMARY  -->    <action name="displayReport">      <result>/jsp/displayReport.jsp</result>  </action>    <action name="financeReport" class="com.mycompany.reports.FinanceReportAction">      <result name="input">/jsp/index.jsp</result>      <result name="success" type="redirectAction">          <param name="actionName">displayReport</param>          <param name="parse">false</param>          <param name="anchor">SUMMARY</param>      </result>  </action> |