Struts 2 Basics

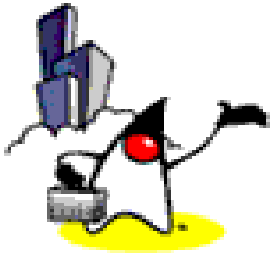
Sang Shin
www.JavaPassion.com
"Learning is fun!"

Topics

- Key core features
- Request processing life-cycle
- Improvements made in Struts 2 (over Struts 1)
- Action
- Result & Result type
- Interceptor
- Validation
- Configuration files
- Packages
- Plug-in
- Struts 2 tags

source: Apache Struts 2 documentation

Key Core Features



source: Apache Struts 2 documentation

Key Core Features

- Pluggable framework architecture that allows request lifecycles to be customized for each action.
- Flexible validation framework that allows validation rules to be decoupled from action code.
- Hierarchical approach to internationalization that simplifies localizing applications.

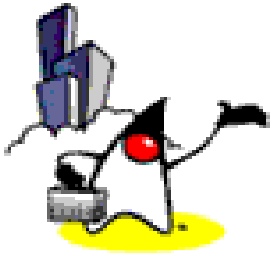
source: Apache Struts 2 documentation

Key Core Features

- Integrated dependency injection engine that manages component lifecycle and dependencies.
  - By default, the framework utilizes Spring for dependency injection
- Modular configuration files that use packages and namespaces to simplify managing large projects with hundreds of actions.
- Java 5 annotations that reduce configuration overhead
  - Java 1.4 is the minimum platform

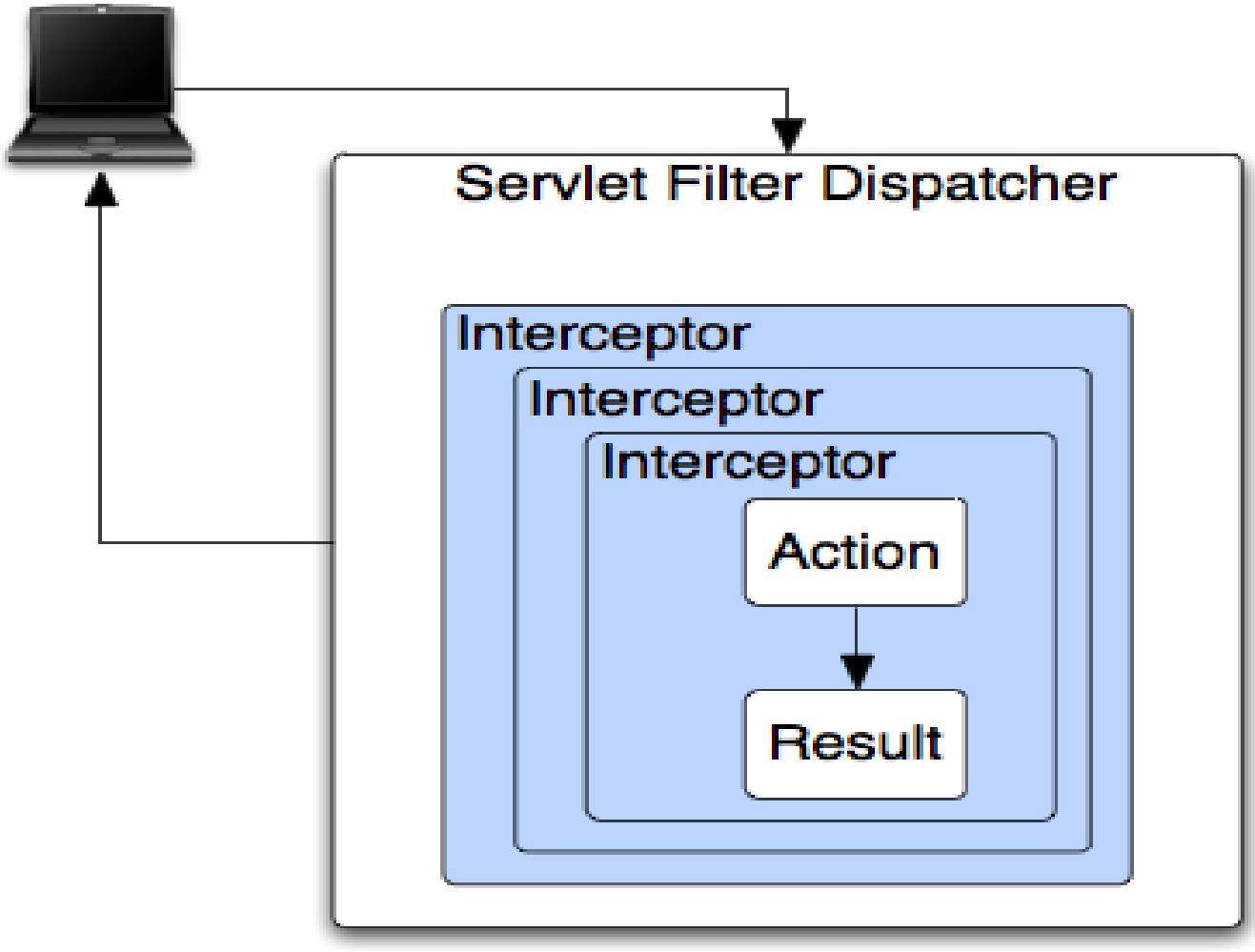
source: Apache Struts 2 documentation

Struts 2 Request Processing Life-Cycle



source: Apache Struts 2 documentation

Struts 2 Architecture



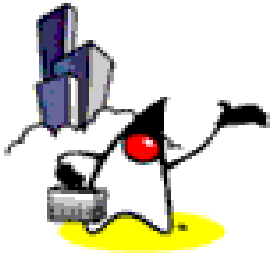
source: Apache Struts 2 documentation

Struts 2 Request Processing

1. The web browser requests a resource (/mypage.action, /reports/myreport.pdf, et cetera)
2. The Filter Dispatcher looks at the request and determines the appropriate Action
3. The Interceptors automatically apply common functionality to the request, like workflow, validation, and file upload handling
4. The Action method executes, usually storing and/or retrieving information from a database
5. The Result renders the output to the browser, be it HTML, images, PDF, or something else

source: Apache Struts 2 documentation

Improvements made in Struts 2



source: Apache Struts 2 documentation

Simplified Design

- Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent.
  - Struts 1 programming model requires implementing the abstract classes while Struts 2 uses interfaces

source: Apache Struts 2 documentation

Intelligent Defaults

- Most configuration elements have a default value which can be set according to the need.
  - "Convention over configuration" philosophy
- There are xml-based default configuration files that can be extended according to the need.

source: Apache Struts 2 documentation

Improved Results

- Unlike *ActionForwards* (of Struts 1), Struts 2 *Results* provide flexibility to create multiple type of outputs

source: Apache Struts 2 documentation

POJO Action's

- Struts 1
  - Actions in Struts1 have dependencies on the servlet API since the *HttpServletRequest* and *HttpServletResponse* objects are passed to the *execute()* method
- Struts 2
  - Any java class with *execute()* method can be used as an Action class.
  - Actions are neutral to the underlying framework

source: Apache Struts 2 documentation

No More ActionForm's

- *ActionForms* feature (of Struts 1) is no more known to the Struts 2 framework.
- Simple JavaBean flavored actions are used (in Struts 2) to put properties directly

source: Apache Struts 2 documentation

Enhanced Testability

- Struts 2 Actions are HTTP independent and framework neutral.
  - This enables to test struts applications very easily without resorting to mock objects.

source: Apache Struts 2 documentation

Better Tag Features

- Struts 2 tags enables to add style sheet-driven markup capabilities
  - You can create consistent pages with less code
  - Struts 2 tag markup can be altered by changing an underlying stylesheet.
- Struts 2 tags are more capable and result oriented.
- Both JSP and FreeMarker tags are fully supported.

source: Apache Struts 2 documentation

Annotation Support

- Java 5 annotations can be used as an alternative to XML and Java properties configuration

source: Apache Struts 2 documentation

Stateful Checkboxes

- Struts 2 checkboxes do not require special handling for false values

source: Apache Struts 2 documentation

Quick Start

- Many changes can be made on the fly without restarting a web container.

source: Apache Struts 2 documentation

Customizable Controller

- Struts 2 lets to customize the request handling per action, if desired.
  - Struts 1 lets to customize the request processor per module

source: Apache Struts 2 documentation

Easy Spring Integration

- Struts 2 Actions are Spring-aware
  - You just need to add Spring beans

source: Apache Struts 2 documentation

Easy Plug-in's

- Struts 2 extensions can be added by dropping in a JAR.
  - No additional XML or properties files.
  - Metadata is expressed through convention and annotation

source: Apache Struts 2 documentation

Ajax Support

- AJAX client side validation
- Remote form submission support (works with the submit tag as well)
- An advanced *div* template that provides dynamic reloading of partial HTML
- An advanced template that provides the ability to load and evaluate JavaScript remotely
- An AJAX-only tabbed Panel implementation
- A rich pub-sub event model
- Interactive auto complete tag

source: Apache Struts 2 documentation

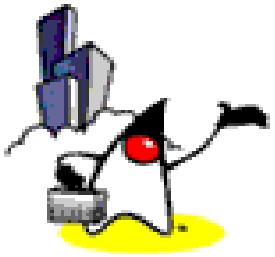
Struts 1 vs. Struts 2

- Action
- ActionForm
- ActionForward
- struts-config.xml
- ActionServlet
- RequestProcessor
- validation.xml

- Action
- Action or POJO
- Result
- struts.xml
- FilterDispatcher
- Interceptors
- *Action*-validation.xml

source: Apache Struts 2 documentation

Action



source: Apache Struts 2 documentation

Action

- Is a basic "unit-of-work"
- POJO class which has *execute()* method

source: Apache Struts 2 documentation

Action Mapping Configuration

- Maps an identifier to a *Action* class.
- When a request matches the action's name, the framework uses the mapping to determine how to process the request.
- Also specifies
  - A set of result types
  - A set of exception handlers
  - An interceptor stack

source: Apache Struts 2 documentation

Example: Action Mapping

```
<action name="Logon" class="tutorial.Logon">
  <result type="redirect-action">Menu</result>
  <result name="input">/tutorial/Logon.jsp</result>
</action>
```

source: Apache Struts 2 documentation

Action Names (*name* attribute)

- The *name* attribute is matched a part of the location requested by a browser (or other HTTP client).
- The framework will drop the host and application name and the extension, and match what's in the middle.
- Example:
  - A request for *http://www.planetstruts.org/struts2-mailreader/Welcome.do* will map to the *Welcome* action.

source: Apache Struts 2 documentation

Action Names

- Within an application, the link to an action is usually generated by a Struts Tag.
- The tag can specify the action by name, and the framework will render the default extension and anything else that is needed.

```
<s:form action="Hello">
    <s:textfield label="Please enter your name" name="name"/>
    <s:submit/>
</s:form>
```

source: Apache Struts 2 documentation

Example: Plain JSP

```
<html>
  <head><title>Add Blog Entry</title></head>
  <body>
    <form action="save.action" method="post">
        Title: <input type="text" name="title" /><br/>
        Entry: <textarea rows="3" cols="25" name="entry"></textarea>
   <br/>
        <input type="submit" value="Add"/>
    </form>
  </body>
</html>
```

source: Apache Struts 2 documentation

Example: Using Struts Tag

```
<%@ taglib prefix="s" uri="/WEB-INF/struts-tags.tld" %>

<html>
  <head><title>Add Blog Entry</title></head>
  <body>
   <s:form action="save" method="post" >
      <s:textfield label="Title" name="title" />
      <s:textarea label="Entry" name="entry" rows="3" cols="25" />
      <s:submit value="Add"/>
  </s:form>
   </body>
</html>
```

source: Apache Struts 2 documentation

Action Interface

- The default entry method to the handler class is defined by the Action interface.

```
public interface Action {
    public String execute() throws Exception;
}
```

- Implementing the Action interface is optional.
  - If Action is not implemented, the framework will use reflection to look for an *execute* method.

source: Apache Struts 2 documentation

Action Methods

- Sometimes, developers like to create more than one entry point to an Action.
  - For example, in the case of of a data-access Action, a developer might want separate entry-points for create, retrieve, update, and delete. A different entry point can be specified by the method attribute.

```
<action name="delete" class="example.CrudAction"
        method="delete">
<action name="create" class="example.CrudAction"
        method="create">
```

- If there is no execute method, and no other method, specified in the configuration, the framework will throw an exception.

source: Apache Struts 2 documentation

Wildcard Method

- Many times, a set of action mappings will share a common pattern
- Rather than code a separate mapping for each action class, you can write it once as a wildcard mapping.

source: Apache Struts 2 documentation

Example: Wildcard Method

- Example 1

```
<action name="*Crud" class="example.Crud"
  method="{1}">
```

Here, a reference to "*editCrud*" will call the *edit* method on an instance of the Crud Action class. Likewise, a reference to "*deleteCrud*" will call the *delete* method instead.

- Example 2

```
<action name="Crud_*" class="example.Crud"
  method="{1}">
```

source: Apache Struts 2 documentation

Action Default

- If you would prefer that an omnibus action handle any unmatched requests, you can specify a default action. If no other action matches, the default action is used instead

```
<package name="Hello" extends="action-default">
<default-action-ref name="UnderConstruction">

<action name="UnderConstruction">
  <result>/UnderConstruction.jsp</result>
</action>
```

source: Apache Struts 2 documentation

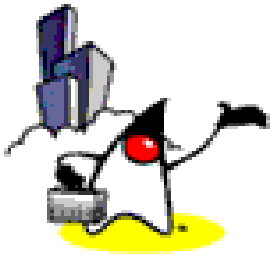
Wildcard Default

- Using wildcards is another approach to default actions.
- A wildcard action at the end of the configuration can be used to catch unmatched references.

```
<action name="*" >
  <result>/{1}.jsp</result>
</action>
```

source: Apache Struts 2 documentation

Result



source: Apache Struts 2 documentation

Result

- When an Action class method completes, it returns a String.
  - The value of the String is used to select a result element.
  - An action mapping will often have a set of results representing different possible outcomes.
- There are predefined result names (tokens)
- Applications can define other result names (tokens) to match specific cases.

source: Apache Struts 2 documentation

Pre-defined result names (tokens)

- String SUCCESS = "success";
- String NONE = "none";
- String ERROR = "error";
- String INPUT = "input";
- String LOGIN = "login";

source: Apache Struts 2 documentation

Result Element

- Provides a logical name (with *name* attribute)
  - An Action can pass back a token like "success" or "error" without knowing any other implementation details.
  - If the *name* attribute is not specified, the framework will give it the name "*success*".
- Provides a Result Type (with *type* attribute)
  - Most results simply forward to a server page or template, but other Result Types can be used to do more interesting things.
  - If a *type* attribute is not specified, the framework will use the *dispatcher*

source: Apache Struts 2 documentation

Result element

- Result element without defaults

```
<result name="success" type="dispatcher">
    <param name="location">/ThankYou.jsp</param>
</result>
```

- Result element using some defaults (as as above)

```
<result>
    <param name="location">/ThankYou.jsp</param>
</result>
```

- Result element using default for the <param> as well

```
<result>/ThankYou.jsp</result>
```

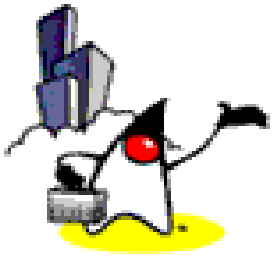
source: Apache Struts 2 documentation

Example: Multiple Results

```
<action name="Hello">
  <result>/hello/Result.jsp</result> <!-- name=”success” -->
  <result name="error">/hello/Error.jsp</result>
  <result name="input">/hello/Input.jsp</result>
</action>
```

source: Apache Struts 2 documentation

Result Types



source: Apache Struts 2 documentation

Predefined Result Types

- Dispatcher Result
- Redirect Action Result
- Chain Result
- Redirect Result
- FreeMarker Result
- Velocity Result
- PlainText Result
- Tiles Result
- HttpHeader Result
- Stream Result

source: Apache Struts 2 documentation

Setting a default Result Type

- If a *type* attribute is not specified, the framework will use the *dispatcher*.
  - The default Result Type, dispatcher, forwards to another web resource.
- A default Result Type can be set as part of the configuration for each package.

```
<result-types>
 <result-type name="dispatcher"
  class="org.apache.struts2.dispatcher.ServletDispatcherResult"
  default="true"/>
</result-types>
```

source: Apache Struts 2 documentation

Global Results

- Most often, results are nested with the action element. But some results apply to multiple actions.
  - Example: In a secure application, a client might try to access a page without being authorized, and many actions may need access to a "logon" result.
- If actions need to share results, a set of global results can be defined for each package.
- The framework will first look for a local result nested in the action. If a local match is not found, then the global results are checked.

source: Apache Struts 2 documentation

Example: Global Results

```
<global-results>
  <result name="error">/Error.jsp</result>
  <result name="invalid.token">/Error.jsp</result>
  <result name="login" type="redirect-action">Logon!input</result>
</global-results>
```

source: Apache Struts 2 documentation

Dynamic Results

- A result may not be known until execution time.
- Result values may be retrieved from its corresponding Action implementation by using EL expressions that access the Action's properties

source: Apache Struts 2 documentation

Example: Dynamic Results

- Give the following Action fragment

```
private String nextAction;
public String getNextAction() {
    return nextAction;
}
```

- You might define a result like following

```
<action name="fragment" class="FragmentAction">
  <result name="next"
            type="redirect-action">${nextAction}</result>
</action>
```

source: Apache Struts 2 documentation

Example: Dynamic Results

- In the code below, if it returns success, then the browser will be forwarded to

  /<app-prefix>/myNamespace/otherAction.action?id=<value of id>

```
<action name="myAction" class="com.project.MyAction">
    <result name="success"
            type="redirect-action">otherAction?id=${id}</result>
    <result name="back"
            type="redirect">${redirectURL}</result>
  </action>
```

source: Apache Struts 2 documentation

Action Chaining

- The framework provides the ability to chain multiple actions into a defined sequence or workflow.
- This feature works by applying a Chain Result to a given Action

source: Apache Struts 2 documentation

Chain Result

- The Chain Result 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.

source: Apache Struts 2 documentation

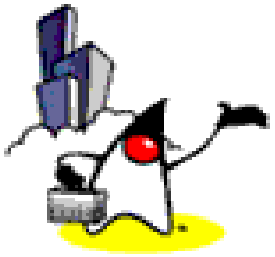
```
<package name="public" extends="struts-default">
   <!-- Chain creatAccount to login, using the default parameter -->
   <action name="createAccount" class="...">
      <result type="chain">login</result>
   </action>

   <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>
```

source: Apache Struts 2 documentation

Interceptor



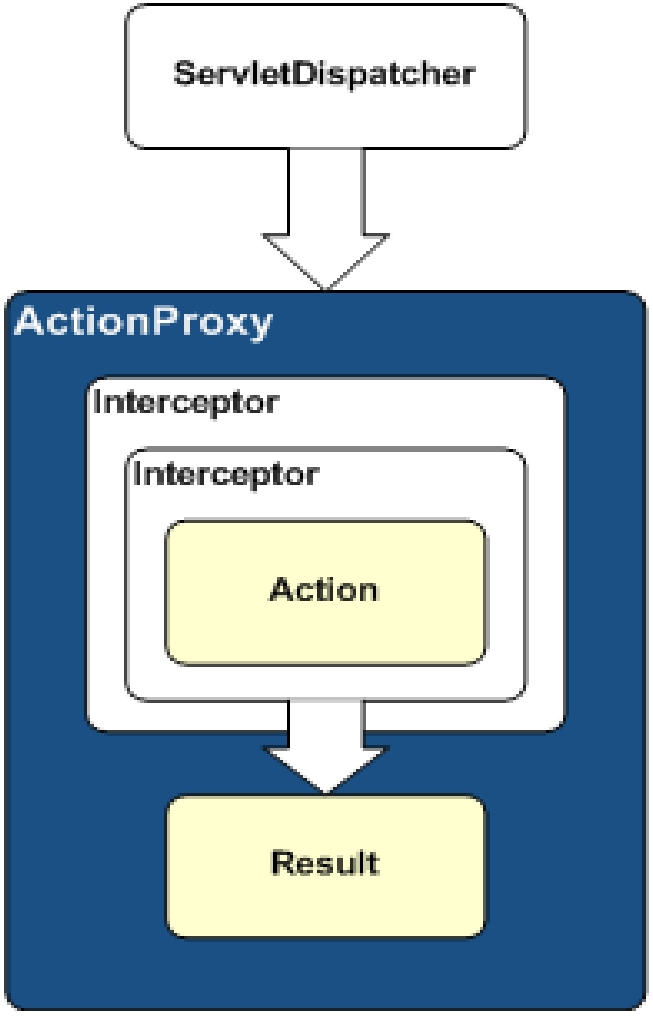
source: Apache Struts 2 documentation

Why Interceptors?

- Many Actions share common concerns.
  - Example: 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.

source: Apache Struts 2 documentation

Interceptor in Action Life-cycle



1 . ServletDispatch instantiates a new ActionProxy and calls execute()

2. Interceptors intercept the request being sent to and returning from the Action

3. Once execution of the Action is complete, the request is sent to a Result to render the results.

source: Apache Struts 2 documentation

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

source: Apache Struts 2 documentation

Interceptors & Actions

- 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.

source: Apache Struts 2 documentation

Configuration of Interceptors

- 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.
- 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.

source: Apache Struts 2 documentation

Configurating Interceptors

```
<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="redirect-action">/secure/home</result>
  </action>
</package>
```

source: Apache Struts 2 documentation

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.

source: Apache Struts 2 documentation

Stacking Interceptors

```
<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="redirect-action">/secure/home</result>
</action>
</package>
```

source: Apache Struts 2 documentation

Interceptor Interface

- Interceptors must implement the *com.opensymphony.xwork2.interceptor.Interceptor* interface
- The *AbstractInterceptor* class provides an empty implementation of init and destroy, and can be used if these methods are not going to be implemented.

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

source: Apache Struts 2 documentation

Example: Interceptor

```
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class SimpleInterceptor extends AbstractInterceptor {

    public String intercept(ActionInvocation invocation) throws
    Exception {
      MyAction action = (MyAction)invocation.getAction();
      action.setDate(new Date());
      return invocation.invoke();
    }
}
```

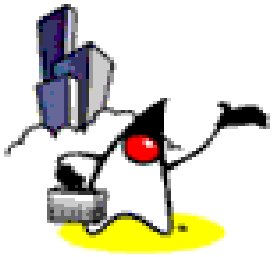
source: Apache Struts 2 documentation

Framework Interceptors

- Struts 2 framework provides an extensive set of ready-to-use interceptors
  - Parameter interceptor: Sets the request parameters onto the Action.
  - Scope interceptor: Simple mechanism for storing Action state in the session or application scope.
  - Validation interceptor: Performs validation using the validators defined in action-validation.xml
  - Many more
- Configured in *struts-default.xml*

source: Apache Struts 2 documentation

View Support



source: Apache Struts 2 documentation

View

- Reusable user interface tags that allow for easy component-oriented development using themes and templates.
- Bundled tags ranges from simple text fields to advanced tags like date pickers and tree views.
- JSTL-compatible expression language (OGNL) that exposes properties on multiple objects as if they were a single JavaBean.
- Pluggable Result Types that support multiple view technologies, including JSP, FreeMarker, Velocity, PDF, and JasperReports.

source: Apache Struts 2 documentation

View

- Optional AJAX theme that simplifes creating interactive web applications.
- Optional Interceptor plugins that can execute long-running queries in the background, prevent multiple form submissions, or handle custom security schemes.

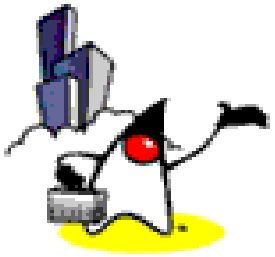
source: Apache Struts 2 documentation

View Technologies

- JSP, Velocity, FreeMaker, PDF, XSLT

source: Apache Struts 2 documentation

Validation



source: Apache Struts 2 documentation

Built-in Validators

```
<validators>
   <validator name="required" class=".."/>
   <validator name="requiredstring" class=".."/>
   <validator name="int" class=".."/>
   <validator name="double" class=".."/>
   <validator name="date" class=".."/>
   <validator name="expression" class=".."/>
   <validator name="fieldexpression" class=".."/>
   <validator name="email" class=".."/>
   <validator name="url" class=".."/>
   <validator name="visitor" class=".."/>
   <validator name="conversion" class="../>
   <validator name="stringlength" class=".."/>
   <validator name="regex" class=".."/>
</validators>
```

source: Apache Struts 2 documentation

Defining Validation Rules

- Per Action class:
  - in a file named <*ActionName*>*-validation.xml*
- Per Action alias:
  - in a file named <*ActionName-alias*>*-validation.xml*
- Inheritance hierarchy and interfaces implemented by Action class
  - Searches up the inheritance tree of the action to find default validations for parent classes of the Action and interfaces implemented

source: Apache Struts 2 documentation

Example: SimpleAction-validation.xml

```
<validators>
  <field name="bar">
     <field-validator type="required">
        <message>You must enter a value for bar.</message>
     </field-validator>
     <field-validator type="int">
        <param name="min">6</param>
        <param name="max">10</param>
        <message>bar must be between ${min} and ${max}, current
   value is ${bar}.</message>
     </field-validator>
  </field>
  <field name="bar2">
     <field-validator type="regex">
        <param name="regex">[0-9],[0-9]</param>
        <message>The value of bar2 must be in the format "x, y",
   where x and y are between 0 and 9</message>
    </field-validator>
  </field>
```

source: Apache Struts 2 documentation

```
<field name="date">
    <field-validator type="date">
        <param name="min">12/22/2002</param>
        <param name="max">12/25/2002</param>
        <message>The date must be between 12-22-2002 and 12-25-
  2002.</message>
    </field-validator>
</field>
<field name="foo">
    <field-validator type="int">
        <param name="min">0</param>
        <param name="max">100</param>
        <message key="foo.range">Could not find foo.range!</message>
    </field-validator>
</field>
<validator type="expression">
    <param name="expression">foo lt bar </param>
    <message>Foo must be greater than Bar. Foo = ${foo}, Bar = $
  {bar}.</message>
</validator>
</validators>
```

source: Apache Struts 2 documentation

Client Validation

- JavaScript client validation code is generated by the framework
- Code you write

```
<s:form action="Login" validate="true">
    <s:textfield key="username"/>
    <s:password key="password" />
    <s:submit/>
</s:form>
```

source: Apache Struts 2 documentation

Client Validation

- JavaScript code that gets generated by framework

```
<form namespace="/example" id="Login" name="Login"
   onsubmit="return validateForm_Login();" action="/struts2-login-
   clientsidevalidation/example/Login.action"
   method="post"><table class="wwFormTable">

..

<script type="text/javascript">
   function validateForm_Login() {
      form = document.getElementById("Login");
      clearErrorMessages(form);
      clearErrorLabels(form);
```

source: Apache Struts 2 documentation

Example: type=”expression”

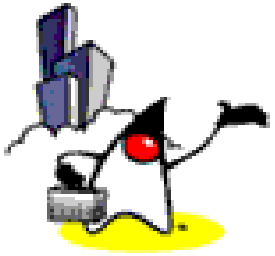
```
<field name="password">
   <field-validator type="requiredstring">
      <message key="error.password.required"/>
   </field-validator>
</field>

<field name="password2">
   <field-validator type="requiredstring">
      <message key="error.password2.required"/>
   </field-validator>
</field>

<validator type="expression">
   <param name="expression">password eq password2</param>
   <message key="error.password.match"/>
</validator>
```

source: Apache Struts 2 documentation

Configuration Files



source: Apache Struts 2 documentation

struts-default.xml

- Defines all of the default bundled results and interceptors and many interceptor stacks which you can use either as-is or as a basis for your own application-specific interceptor stacks.
- Automatically included into *struts.xml* file
- Included in the struts2.jar file.
- In order to provide your own version, you can change the *struts.configuration.files* setting in *struts.properties* file.

source: Apache Struts 2 documentation

struts-default.xml

```
<package name="struts-default">
    <result-types>
        <result-type name="chain"  class="..."/>
        <result-type name="dispatcher" class="..." default="true"/>
        ...
    </result-types>

    <interceptors>
        <interceptor name="alias" class="..."/>
        <interceptor name="autowiring" class="...r"/>
        ...

        <!-- Basic stack -->
        <interceptor-stack name="basicStack">
            <interceptor-ref name="..."/>
            <interceptor-ref name="servlet-config"/>
            <interceptor-ref name="prepare"/>
            <interceptor-ref name="checkbox"/>
            <interceptor-ref name="params"/>
            <interceptor-ref name="conversionError"/>
```

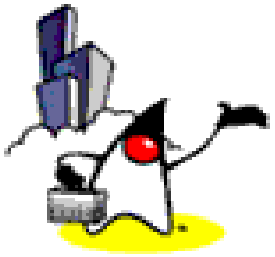
source: Apache Struts 2 documentation

struts.xml

- The core configuration file for the framework
- Should reside on the classpath of the webapp (generally /WEB-INF/classes).

source: Apache Struts 2 documentation

Packages



source: Apache Struts 2 documentation

Package

- Packages are a way to group actions, results, result types, interceptors, and interceptor-stacks into a logical configuration unit.
- Conceptually, packages are similiar to classes in that they can be extended and have individual parts that can be overridden by "sub" packages.

source: Apache Struts 2 documentation

Example: <package ..>

```
<package name="employee" extends="struts-default"
            namespace="/employee">
 <default-interceptor-ref name="crudStack"/>

 <action name="list" method="list"
  class="org.apache.struts2.showcase.action.EmployeeAction" >
   <result>/empmanager/listEmployees.jsp</result>
   <interceptor-ref name="basicStack"/>
 </action>
```

source: Apache Struts 2 documentation

Namespace attribute

- The namespace attribute subdivides action configurations into logical modules, each with its own identifying prefix.
  - Namespaces avoid conflicts between action names. Each namespace can have it's own "menu" or "help" action, each with its own implementation.

source: Apache Struts 2 documentation

Example: Namespace attribute

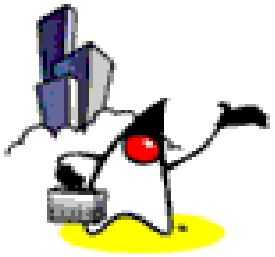
```
<package name="default">
   <action name="foo" class="mypackage.simpleAction>
      <result name="success" type="dispatcher">greeting.jsp</result>
   </action>
   <action name="bar" class="mypackage.simpleAction">
      <result name="success" type="dispatcher">bar1.jsp</result>
   </action>
</package>

<package name="mypackage1" namespace="/">
   <action name="moo" class="mypackage.simpleAction">
      <result name="success" type="dispatcher">moo.jsp</result>
   </action>
</package>

<package name="mypackage2" namespace="/barspace">
   <action name="bar" class="mypackage.simpleAction">
      <result name="success" type="dispatcher">bar2.jsp</result>
   </action>
</package>
```

source: Apache Struts 2 documentation

Plug-in



source: Apache Struts 2 documentation

Struts 2 Plug-in Architecture

- Plug-in's are used to extend the framework just by adding a JAR to the application's classpath.
- Since plugins are contained in a JAR, they are easy to share with others.

source: Apache Struts 2 documentation

Bundled Plug-in's

- JSF plug-in
- REST plug-in
- Spring plug-in
- Tiles plug-in
- SiteGraph plug-in
- Sitemesh plug-in
- JasperReports plug-in
- JFreeChart plug-in
- Config Browser plug-in
- Struts 1 plug-in

source: Apache Struts 2 documentation

Home - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://cwiki.apache.org/S2PLUGINS/home.html

Getting Started Latest Headlines

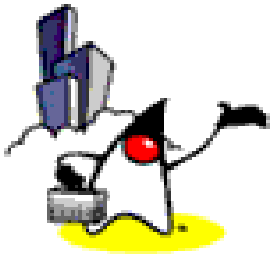
Yahoo! Finance | Google | Home

Plugins

- Breadcrumbs Plugin — is designed to capture bread crumbs. It can be configured many ways
- Codebehind Plugin — reduces mundane configuration by adding "Page Controller" conventions
- Config Browser Plugin — a simple tool to help view your Struts configuration at runtime
- Connext Graph Plugin — allows web applications to make use of the Open Flash Chart charts
- EJB3 plugin — allows EJB3 session beans to be injected into Struts 2 actions
- ExtraTags Plugin — provides a set of additional tags that complement the core UI tags
- Groovy Standalone Plugin — provides support for Actions (and Interceptors) written in the Groovy language
- Guice Plugin — allows Actions, Interceptors, and Results to be injected by Guice
- GWT Plugin — can be used to call methods on Struts actions using Google Web Toolkit (GWT)
- HDIV Plugin — integrates HDIV (HTTP Data Integrity Validator) with Struts 2 adding Security functionalities
- Image Plugin — collection of various plugins for image handling (thumbnails, remote storage, validator)
- JasperReports Plugin — enables Actions to return reports through JasperReports
- JFreeChart Plugin — allows Actions to easily return generated charts and graphs
- JSCalendar Plugin — a backport of the Webwork 2.2.6 JSCalendar tag
- JSF Plugin — provides support for JavaServer Faces components with no additional configuration
- JSON Plugin — provides a "json" result type that serializes actions into JSON
- LightBoxJS Plugin — makes it easier to use the popular LightBoxJS script
- OSGi Plugin — leverages OSGi to allow Struts 2 applications to be divided into multiple jars (bundles) and managed at runtime
- Pell Multipart Plugin — instructs Struts to use Jason Pell's multipart parser to process file uploads
- Plexus Plugin — enables Struts Actions, Interceptors, and Results to be created and injected by Plexus
- REST Plugin — provides tools to build RESTful applications
- Rome RSS-Atom Plugin — allows easy outputting of Rome SyndFeed objects (RSS, Atom)
- Scope Plugin — implements JBoss Seam-style scoped bijection and conversation management
- SiteGraph Plugin — generates graphical diagrams representing the flow of your web application

on

Struts 2 Tags



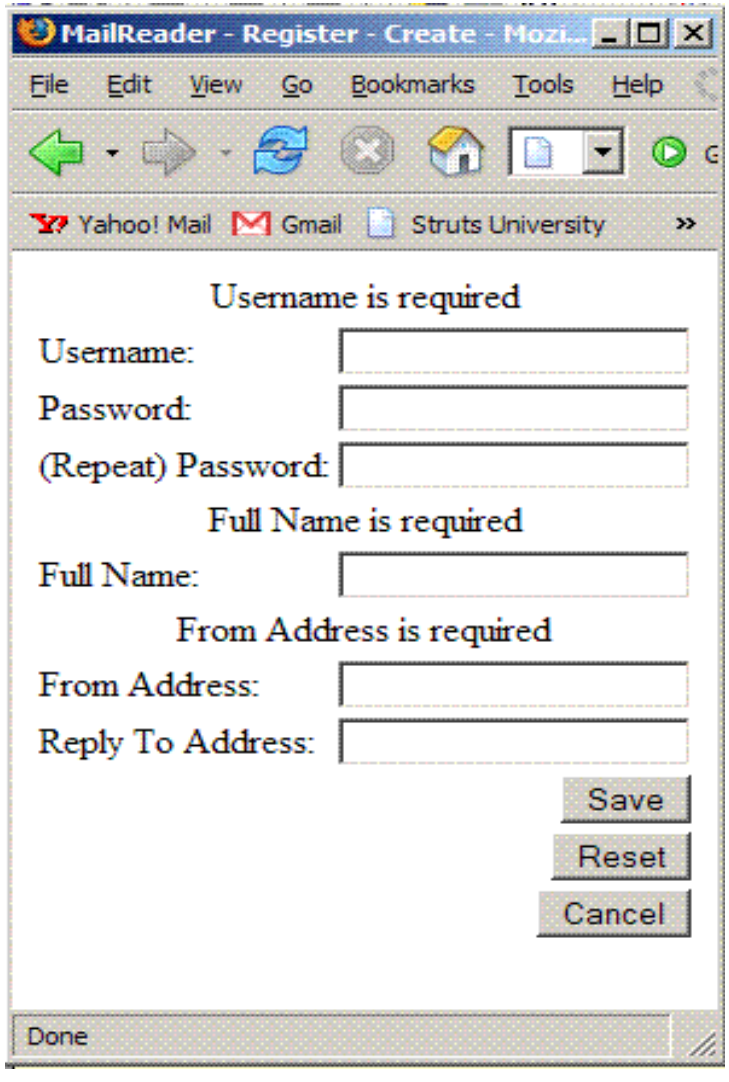
source: Apache Struts 2 documentation

Struts 2 Tags

- The Struts Tags help you create rich web applications with a minimum of coding.

source: Apache Struts 2 documentation

For example page like following...



source: Apache Struts 2 documentation

Without Struts Tags (a partial form)

```
<% User user = ActionContext.getContext() %>
<form action="Profile_update.action" method="post">
 <table>
  <tr>
    <td> align="right"<label>First name:</label></td>
    <td><input type="text" name="user.firstname"
       value="<%=user.getFirstname() %> /></td>
  </tr>
  <tr>
   <td>
    <input type="radio" name="user.gender" value="0"
      id="user.gender0"
     <% if (user.getGender()==0) { %>
     checked="checked" %> } %> />
     <label for="user.gender0">Female</label>
...
```

source: Apache Struts 2 documentation

After Struts Tags (a complete form)

```
<s:actionerror/>
<s:form action="Profile_update" validate="true">
  <s:textfield label="Username" name="username"/>
  <s:password label="Password" name="password"/>
  <s:password label="(Repeat) Password" name="password2"/>
  <s:textfield label="Full Name" name="fullName"/>
  <s:textfield label="From Address" name="fromAddress"/>
  <s:textfield label="Reply To Address"
               name="replyToAddress"/>
  <s:submit value="Save" name="Save"/>
  <s:submit action="Register_cancel" value="Cancel"
            name="Cancel"
            onclick="form.onsubmit=null"/>
</s:form>
```

source: Apache Struts 2 documentation

Thank you!

Sang Shin
www.JavaPassion.com
"Learning is fun!"