



Lightweight J2EE Framework

Struts, spring, hibernate

Software System Design
Zhu Hongjun



Session 2: Struts MVC

- Key Core Features
- Request Processing Life-cycle
- **Action & Result & Result type**
- **Interceptor**
- **Validation**
- Configuration Files & Packages



轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Are annotations better than XML

- The introduction of annotation-based configurations raised the question of whether this approach is 'better' than XML
- The short answer is it depends
- The long answer is that each approach has its pros and cons, and usually it is up to the developer to decide which strategy suits them better
 - Due to the way they are defined, annotations provide a lot of context in their declaration, leading to shorter and more concise configuration.
 - However, XML excels at wiring up components without touching their source code or recompiling them.
 - Some developers prefer having the wiring close to the source while others argue that annotated classes are no longer POJOs and, furthermore, that the configuration becomes decentralized and harder to control



Key Core Features

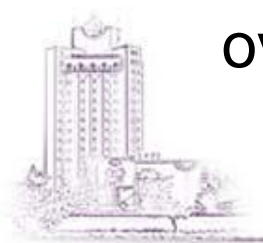
- Key core features of struts
 - Pluggable framework architecture that allows request lifecycles to be customized for each action
 - Flexible validation framework that allows validation rules to be decoupled from action code
 - Hierarchical approach to internationalization that simplifies localizing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



Key core features

■ Key core features of struts (cont.)

- 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.)



轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Key core features

■ Struts 1 Vs Struts 2

■ Improvements made in Struts 2

■ Simplified Design

- Programming the abstract classes instead of interfaces is one of design problem of struts1 framework that has been resolved in the struts 2 framework
- Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent

■ Intelligent Defaults

- Most configuration elements have a default value which can be set according to the need



Key core features

■ Struts 1 Vs Struts 2

■ Improvements made in Struts 2 (cont.)

■ Improved Results

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

■ POJO Action's

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



Key core features

■ Struts 1 Vs Struts 2

■ Improvements made in Struts 2 (cont.)

- No More ActionForms
- Enhanced Testability
- Better Tag Features
- Annotation Support
- Stateful Checkboxes
 - Struts 2 checkboxes do not require special handling for false values



Key core features

■ Struts 1 Vs Struts 2

■ Improvements made in Struts 2 (cont.)

■ Quick Start

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

■ Customizable Controller

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



Key core features

■ Struts 1 Vs Struts 2

■ Improvements made in Struts 2 (cont.)

- Easy Spring Integration

- Easy Plug-in's

- Struts 2 extensions can be added by dropping in a JAR

■ 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, etc.



Request processing life-cycle

■ How does the code works

- The web browser requests a resource “hello.action” (/mypage.action, /reports/myreport.pdf, etc.)
 - According to the settings loaded from the web.xml, the container finds that all requests are being routed to `org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter`, including the *.action requests



```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Basic Struts 2 Application - Welcome</title>
</head>
<body>
<h1>Welcome To Struts 2!</h1>
<p><a href="<s:url action='hello' />">Hello World</a></p>
</body>
</html>

```

Client request form



轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>JeeProject</display-name>

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

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>water.servlet.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>

```

web.xml demo

轻量级J2EE框架

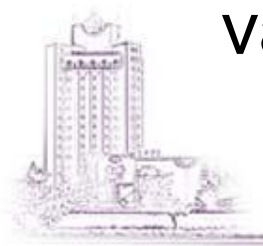
朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Request processing life-cycle

- How does the code works (cont.)
 - The Filter Dispatcher looks at the request and determines the appropriate Action
 - The framework looks for an action mapping named "hello", and it finds that this mapping corresponds to the class "HelloWorldAction"
 - The Interceptors automatically apply common functionality to the request, like workflow, validation, and file upload handling



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd
```

Actions demo

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Request processing life-cycle

- How does the code works (cont.)
 - The framework instantiates the Action and calls the Action's execute method
 - The execute method creates the MessageStore object and returns a result string
 - The Result renders the output to the browser, be it HTML, images, PDF, or something else
 - The framework checks the action mapping to see what page to load if SUCCESS is returned. The framework tells the container to render as the response to the request, the resource HelloWorld.jsp




```
package org.apache.struts.helloworld.action;

import org.apache.struts.helloworld.model.MessageStore;
import com.opensymphony.xwork2.ActionSupport;

public class HelloWorldAction extends ActionSupport {

    private static final long serialVersionUID = 1L;

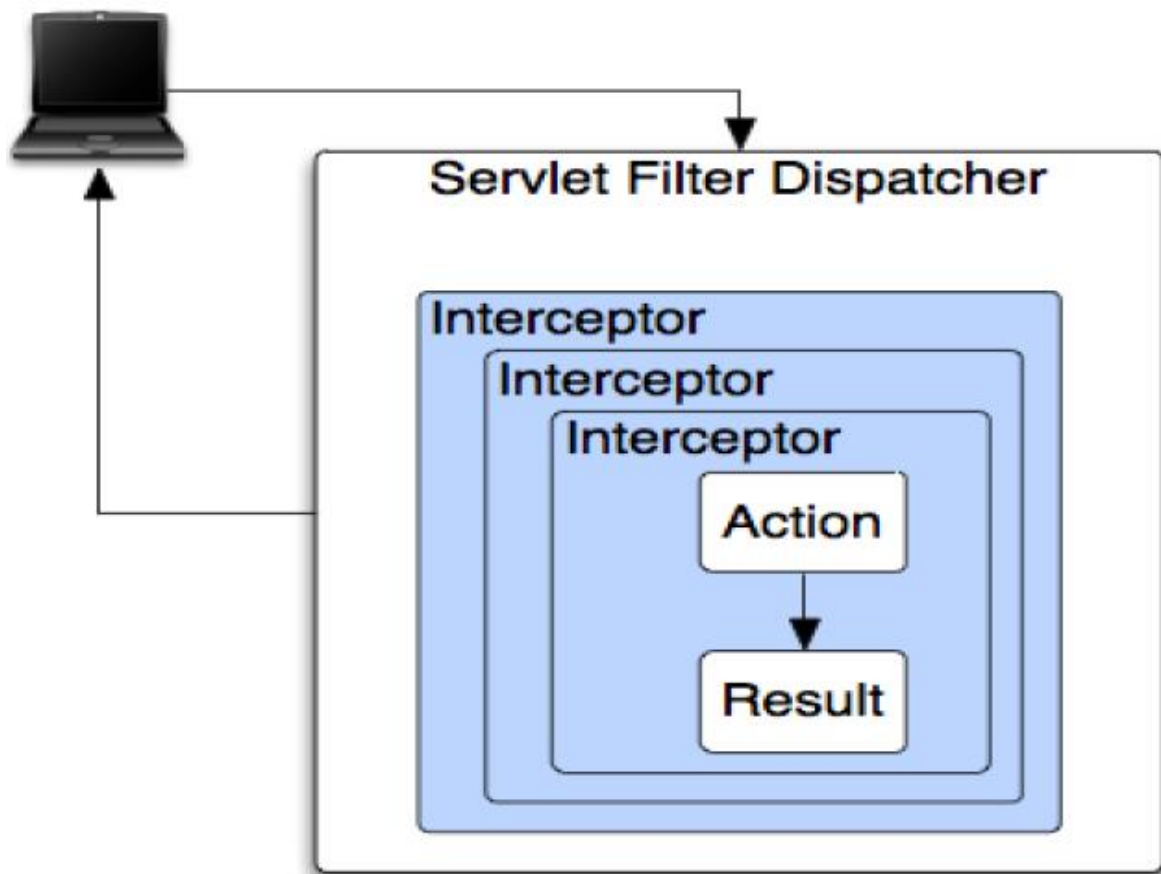
    private MessageStore messageStore;

    public String execute() throws Exception {

        messageStore = new MessageStore() ;
        return SUCCESS;
    }

    public MessageStore getMessageStore() {
        return messageStore;
    }

    public void setMessageStore(MessageStore messageStore) {
        this.messageStore = messageStore;
    }
}
```



Struts 2 Architecture

轻量级J2EE框架

朱洪军

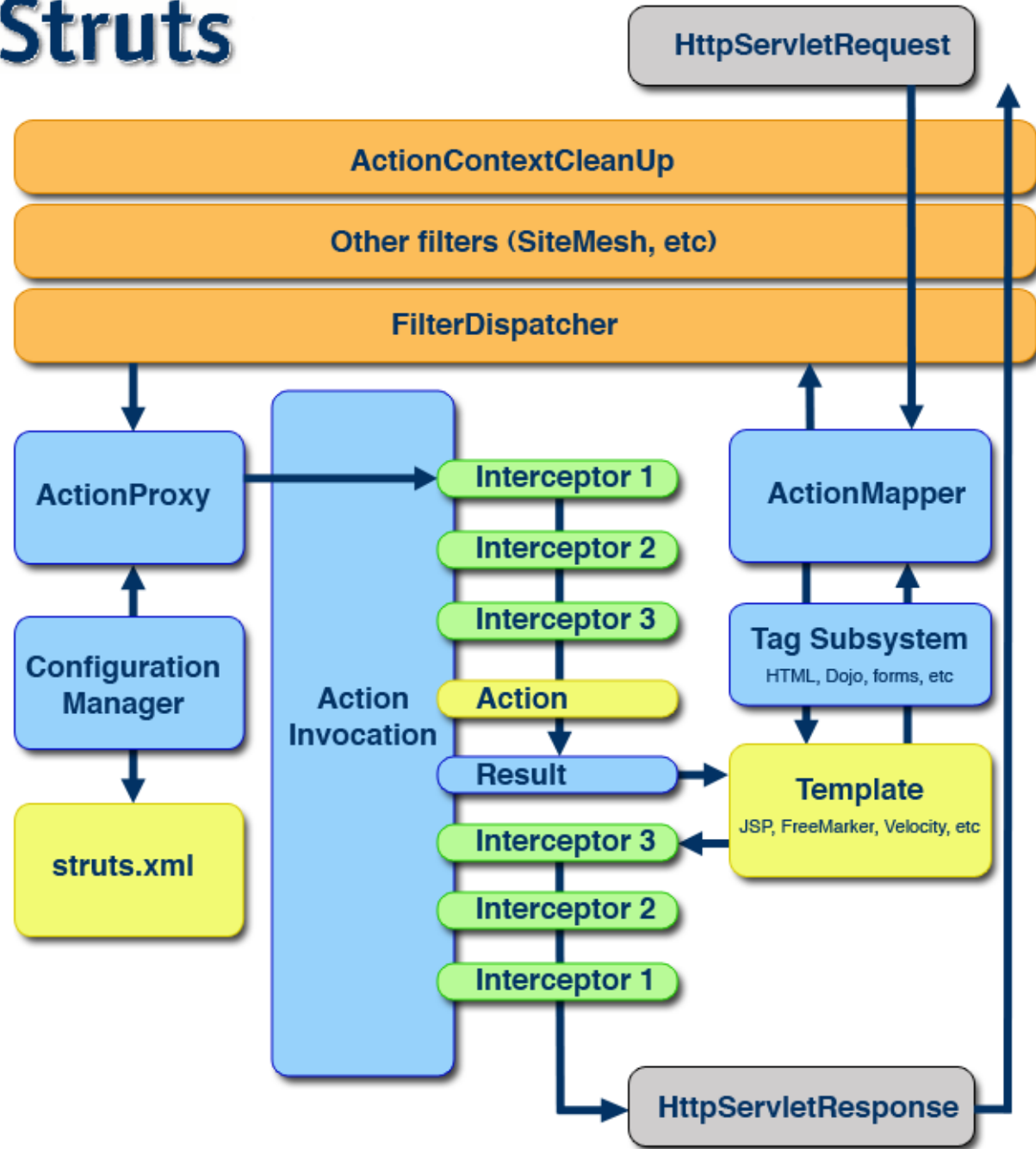
<http://staff.ustc.edu.cn/~waterzhj>



Software Engineering

Struts

The Struts 2 Request Flow



Key:

Servlet Filters

Struts Core

Interceptors

User created

轻量级J2E

The Struts 2 Profiling Aspects

Struts2 profiling aspects involves the following:

- ActionContextCleanUp
- FreemarkerPageFilter
- DispatcherFilter
 - Dispatcher
 - creation of DefaultActionProxy
 - creation of DefaultActionInvocation
 - creation of Action
 - execution of DefaultActionProxy
 - invocation of DefaultActionInvocation
 - invocation of Interceptors
 - invocation of Action
 - invocation of PreResultListener
 - invocation of Result

execution sequence

Action & Result & Result type

■ Action

- The action mappings are the basic "unit-of-work" in the framework
- Essentially, the action maps an identifier to a handler class
- When a request matches the action's name, the framework uses the mapping to determine how to process the request



Action & Result & Result type

■ Action (cont.)

■ Action Mapping

- The action mapping can specify a set of
 - result types
 - exception handlers
 - an interceptor stack
- Only the name attribute is required. The other attributes can also be provided at package scope



```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="basicstruts2" extends="struts-default">
        <action name="index">
            <result>/jsp_files/index.jsp</result>
        </action>
        <action name="hello">
            <result>/jsp_files/hello.jsp</result>
        </action>
    </package>
</struts>

```

```

<action name="edit-*" class="org.apache.struts2.showcase.action.EmployeeAction">
    <param name="empId">{1}</param>
    <result>/empmanager/editEmployee.jsp</result>
    <interceptor-ref name="crudStack">
        <param name="validation.excludeMethods">execute</param>
    </interceptor-ref>
</action>

```

Action Mapping demo

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Action & Result & Result type

■ Action (cont.)

■ Action Names

- In a web application, the name attribute is matched as 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: the action name
- A request for “`http://www.planetstruts.org/struts2-mailreader/Welcome.do`” will map to the Welcome action

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Action & Result & Result type

■ Action (cont.)

■ Action Names (cont.)

- Within an application a 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. Forms may also submit directly to a Struts Action name (rather than a "raw" URI)



```
<action name="hello">
    <result>/jsp_files/hello.jsp</result>
</action>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <p>
        <p><a href="<s:url action='hello'"/>">Hello World</a></p>
    </p>
</body>
</html>
```

Action Name demo

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Action & Result & Result type

■ Action (cont.)

■ Action Methods

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

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



Action & Result & Result type

■ Action (cont.)

■ Action Methods (cont.)

- Implementing the Action interface is optional. If Action is not implemented, the framework will use reflection to look for an execute method
- Sometimes, developers like to create more than one entry point to an Action. A different entry point can be specified by the method attribute

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

Action & Result & Result type

■ Action (cont.)

■ Wildcard Method

- Many times, a set of action mappings will share a common pattern
 - For example, all your edit actions might start with the word "edit", and the same for delete actions
- Rather than code a separate mapping for each action class that uses this pattern, you can write it once as a wildcard mapping



Wildcard Method Demo

```
public String input() throws Exception {  
    // TODO Auto-generated method stub  
    return "success";  
}
```

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

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

在里级JZEE性宋 木洪平 <http://staff.ustc.edu.cn/~waterzhj>



Action & Result & Result type

■ Action (cont.)

■ ActionSupport Default

- If the class attribute in an action mapping is left blank, the ActionSupport class is used as a default
 - The ActionSupport class has an execute method that returns "success" and an input method that returns "input"
- To specify a different class as the default Action class, set the default-class-ref package attribute

轻量级J2EE

```
<action name="Hello">  
    // ...  
</action>
```

du.cn/~waterzhj



Action & Result & Result type

■ Action (cont.)

■ Action Default

- Usually, if an action is requested, and the framework can't map the request to an action name, the result will be the usual "404 - Page not found" error
- But, 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



Action Default Demo

There are no special requirements for the default action. Each package can have its own default action, but there should only be one default action per namespace

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

    <default-action-ref name="UnderConstruction"/>

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

    ...

```

Action & Result & Result type

■ Action (cont.)

■ 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>
```



Action & Result & Result type

■ Action (cont.)

■ 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



Model Driven Demo

```
public class LoginAction extends ActionSupport implements ModelDriven<User> {  
  
    private User user = new User();  
  
    @Override  
    public String execute() throws Exception {  
  
        return "success";  
    }  
  
    @Override  
    public User getModel() {  
        // TODO Auto-generated method stub  
        return user;  
    }  
}
```

```
public class User {  
  
    private String name;  
    private String password;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getPassword() {  
        return password;  
    }  
  
    public void setPassword(String password) {  
        this.password = password;  
    }  
}
```

```
<s:form action="login">  
    <s:textfield name="name" label="用户名"></s:textfield>  
    <s:textfield name="password" label="密码"></s:textfield>  
    <s:submit value="登录" method="login"></s:submit>  
    <s:submit value="注册" method="register"></s:submit>  
</s:form>
```

轻量级J2EE框架 朱洪军 <http://www.it-ebooks.info>

Action & Result & Result type

■ Action (cont.)

■ DMI (Dynamic Method Invocation)

- Dynamic Method Invocation (DMI) will use the string following a "!" character in an action name as the name of a method to invoke (instead of execute)

■ Problems

- First, DMI can cause security issues if POJO actions are used.
- Second, DMI overlaps with the Wildcard Method feature that we brought over from Struts 1



DMI Demo1

```
public String login() {  
    .....  
    return "success";  
}  
  
public String clear() {  
    .....  
    return "clear";  
}
```

```
<s:form action="login!login">
```

```
    <s:textfield name="loginName" label="用户名"></s:textfield>
```

```
    <s:textfield name="loginPassword" label="密码"></s:textfield>
```

```
    <s:submit value="登录"></s:submit>
```

```
    <s:reset value="清空"></s:reset>
```

```
</s:form>
```

```
<action name="login" class="water.action.LoginAction">  
    <result name="input">/jsp_files/index.jsp</result>  
    <result name="success">/jsp_files/hello.jsp</result>  
    <result name="clear">/jsp_files/index.jsp</result>  
</action>
```

轻量

DMI Demo 2

```
<action name="login" class="water.action.LoginAction">
  <result name="input">/jsp_files/index.jsp</result>
  <result name="login">/jsp_files/hello.jsp</result>
  <result name="register">/jsp_files/register.jsp</result>
</action>
```

```
<s:form action="login">
```

```
  <s:textfield name="loginName" label="用户名"></s:textfield>
  <s:textfield name="loginPassword" label="密码"></s:textfield>
  <s:submit value="登录" method="login"></s:submit>
  <s:submit value="注册" method="register"></s:submit>
```

```
</s:form>
```

```
public String login() {
    return "login";
}

public String register() {
    return "register";
}
```

轻量级J2EE框架 朱洪军 <http://st>



Action & Result & Result type

■ Action (cont.)

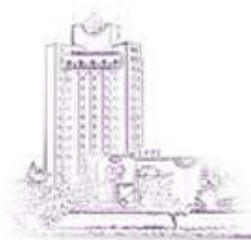
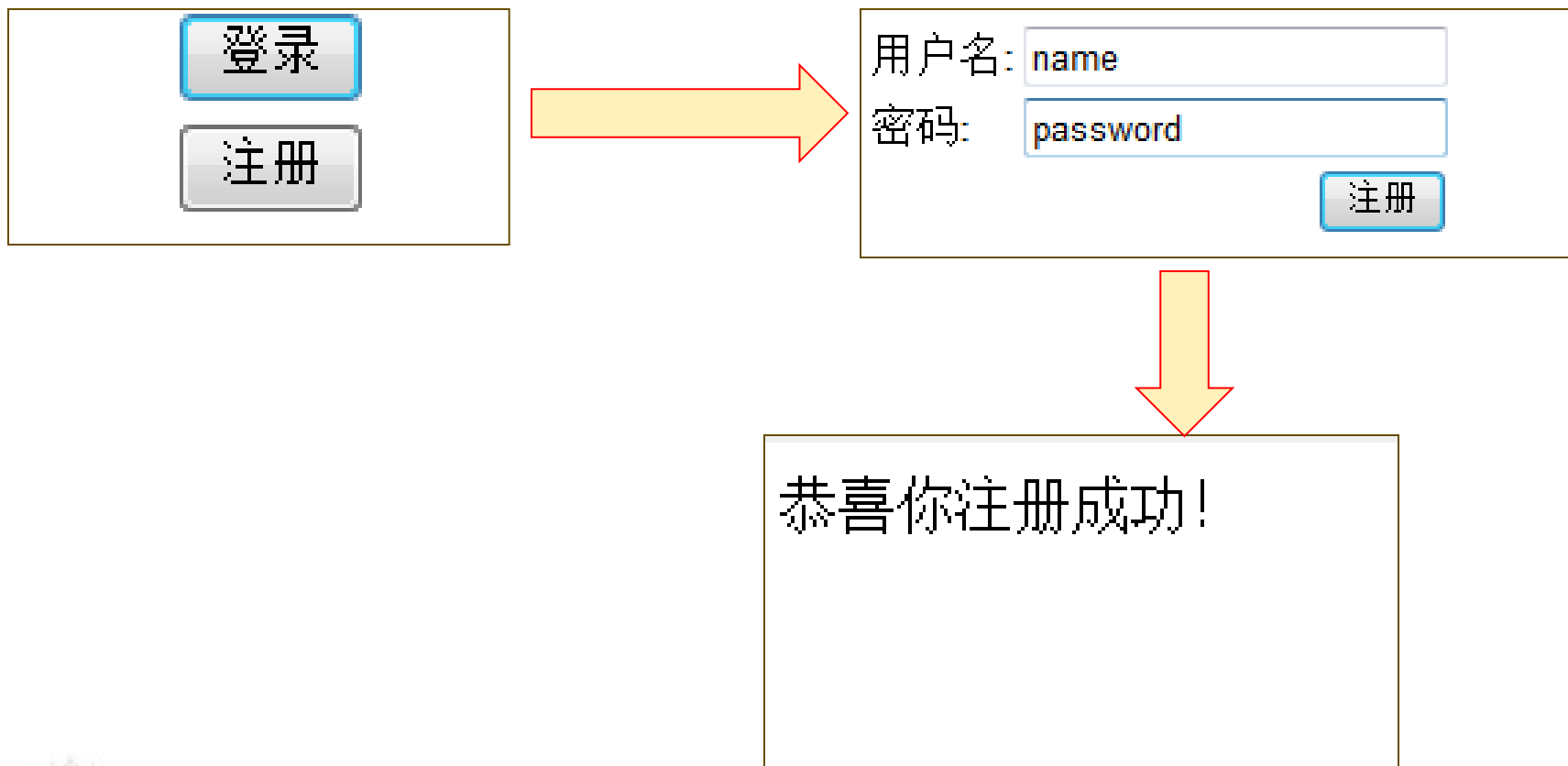
■ Post-Back Default

- A good practice is to link to actions rather than pages
- Another common workflow strategy is to first render a page using an alternate method, like input and then have it submit back to the default execute method
- Using these two strategies together creates an opportunity to use a "post-back" form that doesn't specify an action

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Post-Back Scenario



Post-Back Demo

```
<s:form action="login">
  <s:textfield name="loginName" label="用户名"></s:textfield>
  <s:textfield name="loginPassword" label="密码"></s:textfield>
  <s:submit value="登录" method="login"></s:submit>
  <s:submit value="注册" method="register"></s:submit>
</s:form>
```



```
<s:form>
  <s:textfield name="loginName" label="用户名"></s:textfield>
  <s:textfield name="loginPassword" label="密码"></s:textfield>
  <s:submit value="ok"></s:submit>
</s:form>
```

```
<action name="login" class="water.action.LoginAction">
  <result name="login">/jsp_files/hello.jsp</result>
  <result name="register">/jsp_files/register.jsp</result>
  <result name="success">/jsp_files/success.jsp</result>
  <result name="input">/jsp_files/index.jsp</result>
</action>
```

Action & Result & Result type

■ 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. A standard set of result tokens are defined by the ActionSupport base class
- Applications can define other result tokens to match specific cases

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Predefined results by the ActionSupport

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

Returning ActionSupport
.NONE (or null) from
an action class method
causes the results
processing to be skipped



Action & Result & Result type

■ Result (cont.)

■ Result Element

■ The result element has two jobs

- First, it provides a logical name
- Second, the result element provides a result type

■ Intelligent Defaults

- Each package may set a default result type to be used if none is specified in a result element



Action & Result & Result type

■ Result (cont.)

■ Result Element (cont.)

■ Intelligent Defaults (cont.)

- If the name attribute is not specified, the framework will give it the name "success"
- A special 'other' result can be configured by adding a result with name="*"
 - This result will only be selected if no result is found with a matching name
- **In most cases if an action returns an unrecognized result name this would be a programming error and should be fixed**



Use Default Values in Result Elements

the framework will give it the name "success"

the framework will use the default dispatcher type

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

```
<action name="Hello">
  <del>result</del>/hello/Result.jsp</del>
  <result name="error">/hello/Error.jsp</result>
  <result name="input">/hello/Input.jsp</result>
  <result name="*">/hello/Other.jsp</result>
</action>
```

The name="*" is **not** a wildcard pattern, it is a special name that is only selected if an exact match is not found

Action & Result & Result type

■ Result (cont.)

■ Global Results

- Most often, results are nested with the action element. But some results apply to multiple actions
- 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



Global Result Demo

```
public String hello() throws Exception {  
    // TODO Auto-generated method stub  
    return "global";  
}
```

```
<global-results>  
    <result name="global">/jsp_files/global.jsp</result>  
</global-results>  
  
<action name="hello" class="water.action.HelloAction" method="hello">  
    <result>/jsp_files/hello.jsp</result>  
    <result name="failed">/jsp_files/failed.jsp</result>  
</action>
```

Action & Result & Result type

■ Result (cont.)

■ Dynamic Results

- A result may not be known until execution time
- Sometimes there is a need to redirect from one action to another, but you do not know the exact URL or that the destination URL requires parameters that are only known at run-time
- Result values may be retrieved from its corresponding Action implementation by using EL expressions that access the Action's properties



Dynamic Result Demo

```
private String nextAction;  
  
@Override  
public String execute() throws Exception {  
    setNextAction("hi");  
    return "global";  
}  
  
public void setNextAction(String nextAction) {  
    this.nextAction = nextAction;  
}  
public String getNextAction() {  
    return nextAction;  
}
```

```
<action name="hello" class="water.action.HelloAction">  
    <result>/jsp_files/hello.jsp</result>  
    <result name="global" type="redirectAction">`${nextAction}`</result>  
</action>
```

```
<action name="hi" class="water.action.MyDefaultAction">  
    <result>/jsp_files/failed.jsp</result>  
</action>
```

Action & Result & Result type

■ Result Type

- Most use cases can be divided into two phases. First, we need to change or query the application's state, and then we need to present an updated view of the application
- The Action class manages the application's state, and the Result Type manages the view
- The framework provides several predefined result types



Some predefined result types

Chain Result	Used for Action Chaining
Dispatcher Result	Used for web resource integration, including JSP integration
FreeMarker Result	Used for FreeMarker integration
HttpHeader Result	Used to control special HTTP behaviors
Redirect Result	Used to redirect to another URL (web resource)
Redirect Action Result	Used to redirect to another action mapping
Stream Result	Used to stream an InputStream back to the browser (usually for file downloads)
Velocity Result	Used for Velocity integration
XSL Result	Used for XML/XSLT integration
PlainText Result	Used to display the raw content of a particular page (i.e jsp, HTML)
Tiles Result	Used to provide Tiles integration



Action & Result & Result type

■ Result Type (cont.)

■ Dispatcher Result

- Includes or forwards to a view (usually a jsp). Behind the scenes Struts will use a RequestDispatcher, where the target servlet/JSP receives the same request/response objects as the original servlet/JSP

■ Parameters

- Location (default)
- parse

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



Action & Result & Result type

■ Result Type (cont.)

■ Chain Result

- This result invokes an entire other action, complete with it's own interceptor stack and result

■ Parameters

- actionName (default)
- namespace
- method
- skipActions (optional)



Chain Result Demo

```
<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>
```


Action & Result & Result type

■ Result Type (cont.)

■ FreeMarker Result

- Renders a view using the Freemarker template

- Parameters

- Location (default)
- parse
- contentType
- WriteIfCompleted

```
<result name="success" type="freemarker">foo.ftl</result>
```

Action & Result & Result type

■ Result Type (cont.)

■ HttpHeaders Result

- For setting HTTP headers and status by optionally evaluating against the ValueStack. This result can also be used to send an error to the client

■ Parameters

- status
- parse
- headers
- error
- errorMessage

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



HttpHeader Result Demo

```
<action name="error">
```

```
  <result name="internalError" type="httpheader">
```

```
    <param name="status">500</param>
```

```
    <param name="header.a">a custom header value</param>
```

```
    <param name="header.b">another custom header value</param>
```

```
  </result>
```

```
  <result name="proxyRequired" type="httpheader">
```

```
    <param name="error">305</param>
```

```
    <param name="errorMessage">this action must be accessed through a proxy</param>
```

```
  </result>
```

```
</action>
```

http://localhost:8080/Struts2Project/abc/s

HTTP Status 500 -

type Status report

message

description The server encountered an internal error () tha

Apache Tomcat/7.0.22

Action & Result & Result type

■ Result Type (cont.)

■ Redirect Result

- 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

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Action & Result & Result type

■ Result Type (cont.)

■ Redirect Result (cont.)

■ 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



Redirect Result Demo

```
<action name="say">  
  <result type="redirect">  
    <param name="location">/jsp_files/failed.jsp</param>  
    <param name="parse">false</param>  
    <param name="anchor">Nothing</param>  
  </result>  
</action>
```

http://localhost:8080/Struts2Project/jsp_files/failed.jsp#Nothing

this is failed page!

Action & Result & Result type

■ Result Type (cont.)

■ 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
- It is strongly recommended that if you are redirecting to another action, you use this result rather than the standard redirect result



Action & Result & Result type

■ Result Type (cont.)

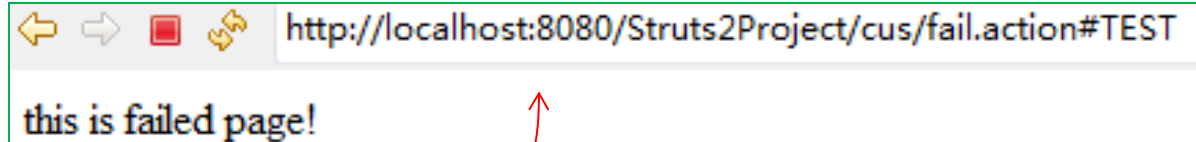
■ Redirect Action Result (cont.)

■ Parameters

- actionName (default)
- Namespace
- suppressEmptyParameters
- Parse
- anchor



Redirect Action Result



http://localhost:8080/Struts2Project/cus/fail.action#TEST

this is failed page!

```
<action name="say">  
  <result type="redirectAction">  
    <param name="actionName">fail</param>  
    <param name="namespace">/cus</param>  
    <param name="anchor">TEST</param>  
  </result>  
</action>
```

```
<package name="custom" extends="struts-default" namespace="/cus">  
  <action name="fail">  
    <result>/jsp_files/failed.jsp</result>  
  </action>  
</package>
```

轻里级J2EE技术 第六讲 http://staff.ustc.edu.cn/~wxc1213/

Software Engineering

Action & Result & Result type

■ Result Type (cont.)

■ Stream Result

- For sending raw data (via an `InputStream`) directly to the `HttpServletResponse`. Very useful for allowing users to download content

■ Parameters

- `contentType`
- `contentLength`
- `contentDisposition`
- `inputName`
- `bufferSize`
- `allowCaching`
- `contentCharSet`

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



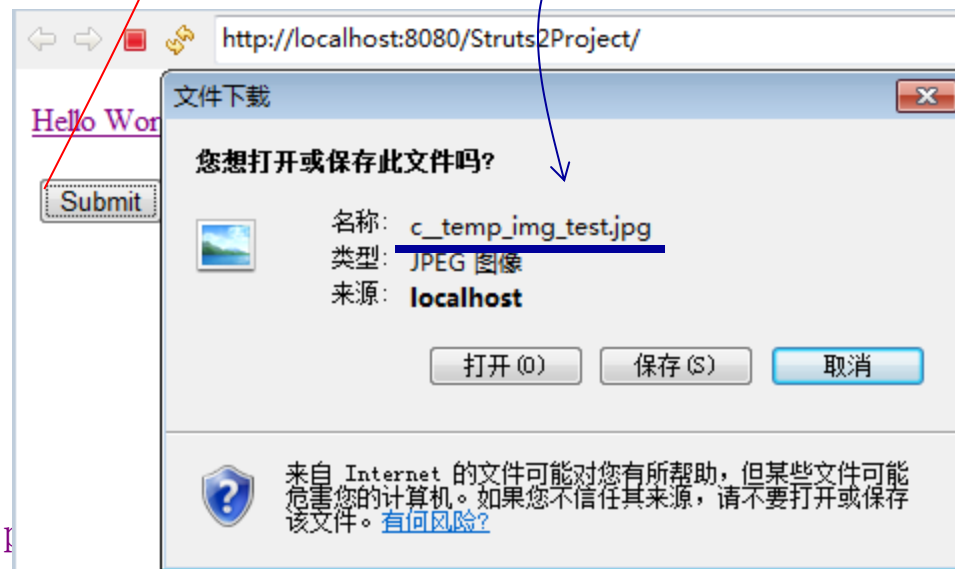
Stream Result Demo

```
<action name="say" class="water.action.HelloAction" method="say">

    <result name="success" type="stream">
        <param name="contentType">image/jpeg</param>
        <param name="inputName">imageStream</param>
        <param name="contentDisposition">attachment;filename="c:/temp/img_test.jpg"</param>
        <param name="bufferSize">20</param>
        <param name="contentLength">878539</param>
    </result>
</action>
```

```
private InputStream imageStream;
public InputStream getImageStream() {
    return imageStream;
}

File f=new File("C:/temp/resource/img_test.jpg");
public String say() {
    try {
        imageStream=new FileInputStream(f);
    } catch (FileNotFoundException e) {
        imageStream=null;
    }
    return "success";
}
```



Action & Result & Result type

■ Result Type (cont.)

■ Plain Text Result

- A result that send the content out as plain text
- Useful typically when needed to display the raw content of a JSP or Html file for example
- Parameters
 - Location (default)
 - Charset (optional)



Plain Text Result Demo

```
http://localhost:8080/Struts2Project/abc/hi?sessionId=2D1C00931EC11F6A5C3831F08771

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <p>this is failed page!</p>
    <s:form action="/abc/hi">
        <s:submit></s:submit>
    </s:form>

</body>
</html>
```

```
<action name="hi" class="water.action.MyDefaultAction">
    <result type="plainText">/jsp_files/failed.jsp</result>
</action>
```

Action & Result & Result type

■ Result Type (cont.)

■ Velocity Result

- Used for Velocity integration

■ XSL Result

- Used for XML/XSLT integration

■ Tiles Result

- Used to provide Tiles integration



Interceptor

■ Why Interceptor

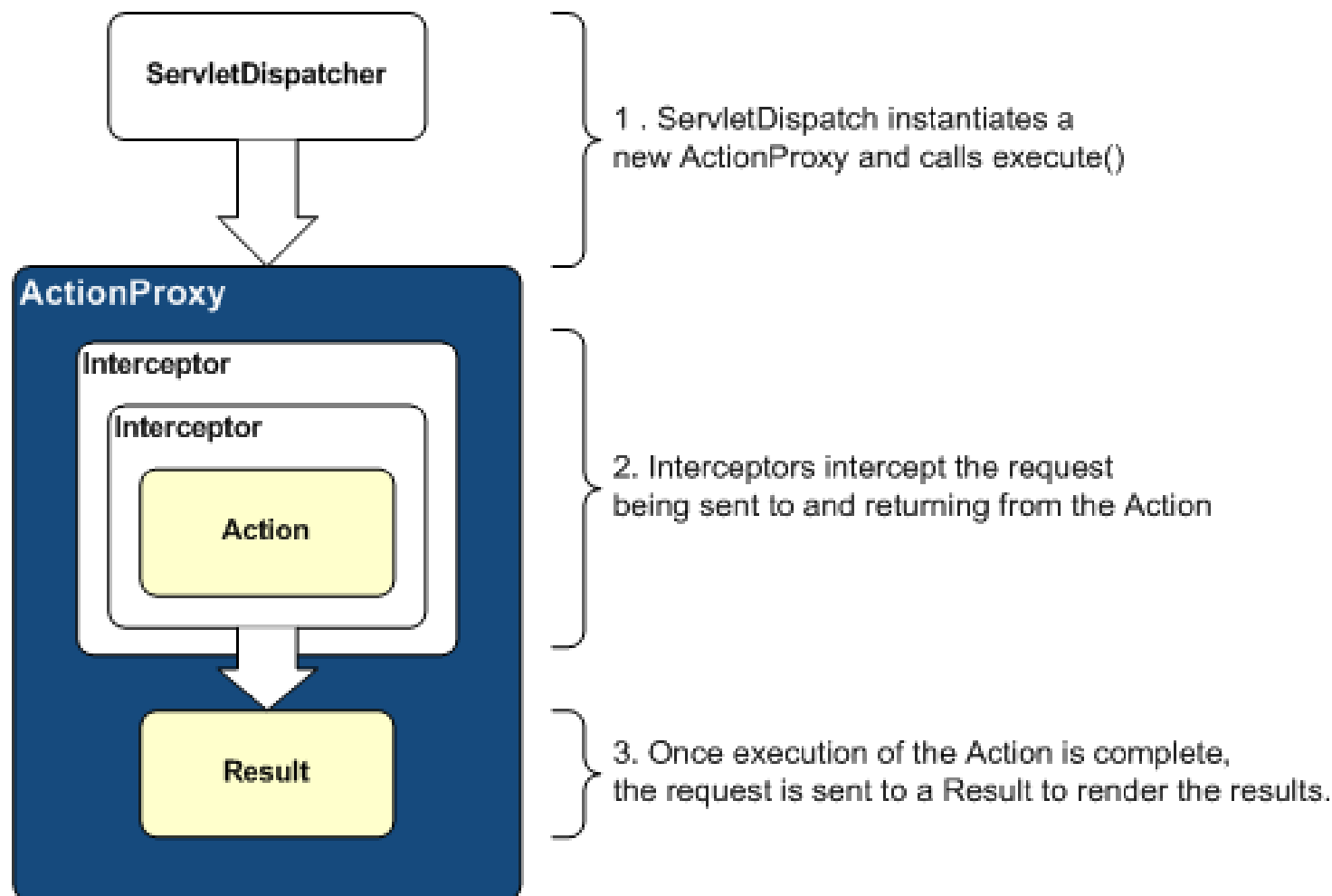
■ Common Concerns

- input validated
- protection from a double submit
- Etc.

- The framework makes it easy to share solutions to these concerns using an "Interceptor" strategy



Action Lifecycle




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

Interceptor execute sequence demo



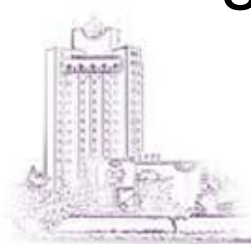
轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



Interceptor

■ Understanding Interceptors

- Interceptors can execute code before and after an Action is invoked
- Each and every Interceptor is pluggable, so you can decide exactly which features an Action needs to support
- The Interceptors are defined in a stack that specifies the execution order



Interceptor

- Configuring and Stacking 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
 - We can bundle Interceptors together using an Interceptor Stack



Configuring 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="redirectAction">/secure/home</result>
  </action>
</package>
```



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="tutorial.Login">
    <interceptor-ref name="myStack" />
    <result name="input">login.jsp</result>
    <result name="success"
      type="redirectAction">/secure/home</result>
  </action>
</package>
```



Interceptor

■ Writing Interceptors

- Interceptors must implement the `com.opensymphony.xwork2.interceptor.Interceptor` interface
 - Init method
 - Be called the after interceptor is instantiated
 - Intercept method
 - Be where the interceptor code is written
 - Destroy method
 - To release resources on application shutdown



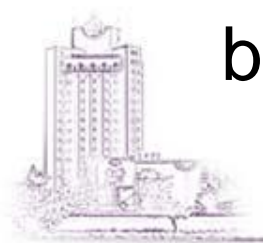
Interceptor Interface

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



Interceptor

- Writing Interceptors (cont.)
 - Interceptors are declared using the *interceptor* element, nested inside the *interceptors* element in `struts.xml` or `struts-default.xml`
 - The `AbstractInterceptor` class provides an empty implementation of *init* and *destroy*, and can be used if these methods are not going to be implemented



Interceptor Interface Demo

```
<interceptors>
  <interceptor name="myi" class="water.interceptor.MyInterceptorTest"></interceptor>
</interceptors>

<action name="say" class="water.action.HelloAction" method="say">
  <interceptor-ref name="myi"></interceptor-ref>
  <result name="success" type="stream">
    .....
  </result>
</action>
```

```
public class MyInterceptorTest implements Interceptor {

    private static final long serialVersionUID = -1755946948307919602L;

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public void init() {
        // TODO Auto-generated method stub
    }

    @Override
    public String intercept(ActionInvocation ai) throws Exception {
        System.out.println("StartIntercept.....");
        HelloAction ha = (HelloAction) ai.getAction();
        ha.setResultString(null);
        ai.invoke();
        System.out.println("EndIntercept.....");
        return "testResult";
    }
}
```

轻量级J2EE框架 朱洪军 <http://staff.ustc.edu.cn/~waterzhj>



AbstractInterceptor Demo

```
<interceptors>
  <interceptor name="log1" class="water.interceptor.Log1"></interceptor>
  <interceptor name="log2" class="water.interceptor.Log2"></interceptor>
  <interceptor name="log3" class="water.interceptor.Log3"></interceptor>
</interceptors>
<action name="index">
  <result>/index.jsp</result>
</action>
<action name="login">
  <interceptor-ref name="log1"></interceptor-ref>
  <interceptor-ref name="log2"></interceptor-ref>
  <interceptor-ref name="log3"></interceptor-ref>
  <result>hello.jsp</result>
</action>
```

```
public class Log1 extends AbstractInterceptor {
```

```
    @Override
```

```
    public String intercept(ActionInvocation arg0) throws Exception {
```

```
        // TODO Auto-generated method stub
```

```
        System.out.println("start intercept log1");
```

```
        String result = arg0.invoke();
```

```
        System.out.println("end intercept log1");
```

```
        return result;
```

```
    }
```

```
}
```

Interceptor

■ Framework Interceptors

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



Framework Interceptors

Interceptor	Name	Description
Alias Interceptor	alias	Converts similar parameters that may be named differently between requests.
Chaining Interceptor	chain	Makes the previous Action's properties available to the current Action. Commonly used together with <result type="chain"> (in the previous Action).
Checkbox Interceptor	checkbox	Adds automatic checkbox handling code that detect an unchecked checkbox and add it as a parameter with a default (usually 'false') value. Uses a specially named hidden field to detect unsubmitted checkboxes. The default unchecked value is overridable for non-boolean value'd checkboxes.
Cookie Interceptor	cookie	Inject cookie with a certain configurable name / value into action. (Since 2.0.7.)
Conversion Error Interceptor	conversionError	Adds conversion errors from the ActionContext to the Action's field errors
Create Session Interceptor	createSession	Create an HttpSession automatically, useful with certain Interceptors that require a HttpSession to work properly (like the TokenInterceptor)
DebuggingInterceptor	debugging	Provides several different debugging screens to provide insight into the data behind the page.
Execute and Wait Interceptor	execAndWait	Executes the Action in the background and then sends the user off to an intermediate waiting page.
Exception Interceptor	exception	Maps exceptions to a result.
File Upload Interceptor	fileUpload	An Interceptor that adds easy access to file upload support.
i18n Interceptor	i18n	Remembers the locale selected for a user's session

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Interceptor

■ File Upload Interceptor

- The Struts 2 framework provides built-in support for processing file uploads
- When correctly configured the framework will pass uploaded file(s) into your Action class
- Support for individual and multiple file uploads are provided



Interceptor

■ File Upload Interceptor (cont.)

■ Dependencies

- The Struts 2 framework leverages add-on libraries to handle the parsing of uploaded files. These libraries are not included in the Struts distribution, you must add them into your project
- Commons-fileupload.jar
- Commons-io.jar



Interceptor

■ File Upload Interceptor (cont.)

■ Basic Usage

- Configure an Action mapping for your Action class as you typically would
- A form must be create with **a form field of type file** and The form used to upload the file must have its **encoding type set to multipart/form-data**
- The fileUpload interceptor will use setter injection to insert the uploaded file and related data into your Action class



File Upload Interceptor Demo

```
<action name="upl" class="water.action.FileAction">
  <interceptor-ref name="fileUpload"></interceptor-ref>
  <interceptor-ref name="basicStack"></interceptor-ref>
  <result name="success">/jsp_files/hello.jsp</result>
</action>
```

```
<s:form action="/abc/upl" method="post" enctype="multipart/form-data">
  <s:file name="myFile" label="File"></s:file>
  <s:submit></s:submit>
</s:form>
```

```
private File myFile;
private String myFileFileName;
private String myFileContentType;

@Override
public String execute() throws Exception {
    if (myFile != null) {
        uploadFile(myFile);
    }
    return ActionSupport.SUCCESS;
}
```

轻量级J2E

waterzhj



Interceptor

■ Validation Interceptor

- This interceptor runs the action through the standard validation framework, which in turn checks the action against any validation rules (found in files such as *ActionClass-validation.xml*)
- This interceptor does nothing if the name of the method being invoked is specified in the **excludeMethods** parameter



Validation Interceptor Demo

```
<action name="login" class="water.action.LoginAction">
  <result name="input">/jsp_files/index.jsp</result>
  <result>/jsp_files/hello.jsp</result>
</action>
```

```
<s:form action="/abc/login">
  <s:textfield name="loginName" label="用户名"></s:textfield>
  <s:textfield name="loginPassword" label="密码"></s:textfield>
  <s:submit value="登录"></s:submit>
</s:form>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//Apache Struts//XWork Validator 1.0.3//EN"
"http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
<validators>
  <field name="loginName">
    <field-validator type="requiredstring">
      <message>you must input a login name!</message>
    </field-validator>
  </field>
  <field name="loginPassword">
    <field-validator type="requiredstring">
      <message>you must input a login password!</message>
    </field-validator>
  </field>
</validators>
```

you must input a login name!

用户名:

you must input a login password!

密码:

登录

```
private String loginPassword;
private String loginName;

@Override
public String execute() throws Exception {
  // TODO Auto-generated method stub
  return ActionSupport.SUCCESS;
}
```

轻量级J2EE框架 朱洪军 <http://www.it-ebooks.info>

Validation

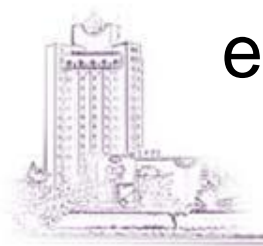
- Struts 2 validation is configured via XML or annotations. Manual validation in the action is also possible
- and may be combined with XML and annotation-driven validation
- Validation also depends on both the validation and workflow interceptors



Validation

■ Manual Validation

- The straightforward way to validate user input is to write Java code to check the request parameters
- To enable the Struts 2 Action class to validate a user's input on a Struts 2 form, you can **define a validate method** in your Action class OR **override validate method** inherited by extending `ActionSupport` class



Manual Validation Dmeo

required a name!

用户名:

密码:

```
public void validate() {  
    if (user.getName() == null || user.getName().equals("")) {  
        addFieldError("name", "required a name!");  
    }  
}
```



Validation

■ Use Built-in Validators

- Using built-in validation is a two-step process
 - You must tell Struts the action that you want to validate
 - Accomplished via interceptors
 - As default, "defaultStack" already has validation turned on
 - You must tell Struts what the validation rules are
 - Accomplished by specifying rules in an XML file



Use Built-in Validators Demo

```
<validators>
  <field name="name">
    <field-validator type="requiredstring">
      <message>name is required!</message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="requiredstring">
      <message>password is required!</message>
    </field-validator>
  </field>
</validators>
```

Validation rules defined in
“LoginAction-validation.xml”

```
<action name="login" class="water.action.LoginAction">
  <result name="login">/jsp_files/index.jsp</result>
  <result name="register">/jsp_files/register.jsp</result>
  <result name="success">/jsp_files/success.jsp</result>
</action>
```

Action struts use built-in
validation to validate

name is required!

用户名:

password is required!

密码:

```
private String name, password;
```

```
public String getName() {
    return name;
}
```

```
public void setName(String name) {
    this.name = name;
}
```

```
public String getPassword() {
    return password;
}
```

```
public void setPassword(String password) {
    this.password = password;
}
```

LoginAction.java

<http://sta>

Validation

■ Custom Validators

- When none of WebWork's built-in validators fit your business needs, you can quite easily declare a custom validator and have it behave the way you want.
- It can be invoked and used just like the built-in validators



Validation

■ Custom Validators

- To create a custom validator, you must define a class that implements the Validator interface or extends ValidatorSupport class
- And register it by adding an entry for it in the *validators.xml* file



Custom Validators Demo

```
public class NameValidator extends ValidatorSupport {  
  
    @Override  
    public void validate(Object o) throws ValidationException {  
        String name = (String) this.getFieldValue("name", o);  
        if (name.equals("") || name == null) {  
            ValidatorContext vc = getValidatorContext();  
            vc.addFieldError("name",  
                "you must supply a non-null value for name!");  
        }  
    }  
}
```

NameValidator.java

Validators.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE validators PUBLIC  
    "-//OpenSymphony Group//XWork Validator Config 1.0//EN"  
    "http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">  
  
<validators>  
    <validator name="name-validation" class="water.validation.NameValidator"></validator>  
</validators>
```

you must supply a non-null value for name!

用户名:

password is required!

密码:

登录

注册

LoginAction-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE validators PUBLIC  
    "-//Apache Struts//XWork Validator Config 1.0//EN"  
    "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">  
  
<validators>  
    <field name="name">  
        <field-validator type="name-validation">  
            <message></message>  
        </field-validator>  
    </field>  
    <field name="password">  
        <field-validator type="requiredstring">  
            <message>password is required!</message>  
        </field-validator>  
    </field>  
</validators>
```

Validation

■ Built-in Validators

■ Conversion Validator

- Field Validator that checks if a conversion error occurred for this field

```
<field name="myField">  
  <field-validator type="conversion">  
    <message>Conversion Error Occurred</message>  
  </field-validator>  
</field>
```



Validation

■ Built-in Validators (cont.)

■ Date Validator

- Field Validator that checks if the date supplied is within a specific range

```
<field name="birthday">  
  <field-validator type="date">  
    <param name="min">01/01/1990</param>  
    <param name="max">01/01/2000</param>  
    <message>Birthday must be within ${min} and ${max}</message>  
  </field>  
</field>
```



Validation

■ Built-in Validators (cont.)

■ Email Validator

- EmailValidator checks that a given String field, if not empty, is a valid email address

```
<field name="myEmail">  
  <field-validator type="email">  
    <message>Must provide a valid email</message>  
  </field-validator>  
</field>
```



Validation

■ Built-in Validators (cont.)

■ FieldExpression Validator

- Validates a field using an OGNL expression

```
<field name="myField">
  <field-validator type="fieldexpression">
    <param name="expression"><![CDATA[#myCreditLimit > #myGirlfriendCreditLimit]]></param>
    <message>My credit limit should be MORE than my girlfriend</message>
  </field-validator>
</field>
```



Validation

■ Built-in Validators (cont.)

■ Regex Validator

- Validates a string field using a regular expression

```
<field name="myStrangePostcode">  
  <field-validator type="regex">  
    <param name="expression"><![CDATA[([aAbBcCdD][123][eEfFgG][456])]]></param>  
  </field-validator>  
</field>
```



Validation

■ Built-in Validators (cont.)

■ Visitor Validator

- The VisitorFieldValidator allows you to forward validation to object properties of your action using the object's own validation files
- This allows you to use the ModelDriven development pattern and manage your validations for your models in one place, where they belong, next to your model classes



Visitor Validator Demo

```
<action name="login" class="water.action.LoginAction">
  <result name="login">/jsp_files/index.jsp</result>
  <result name="register">/jsp_files/register.jsp</result>
  <result name="success">/jsp_files/success.jsp</result>
  <result name="input">/jsp_files/index.jsp</result>
</action>
```

JSP view

```
<s:form action="login">
  <s:textfield name="user.name" label="用户名"></s:textfield>
  <s:textfield name="user.password" label="密码"></s:textfield>
  <s:textfield name="user.age" label="年龄"></s:textfield>
  <s:textfield name="user.address" label="地址"></s:textfield>
  <s:submit value="登录" method="login"></s:submit>
  <s:submit value="注册" method="register"></s:submit>
</s:form>
```

User-test-validation.xml

```
<validators>
  <field name="name">
    <field-validator type="name-validation">
      <message></message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="requiredstring">
      <message>password is required!</message>
    </field-validator>
  </field>
  .....
</validators>
```

LoginAction-validation.xml

```
<validators>
  <field name="user">
    <field-validator type="visitor">
      <param name="context">test</param>
      <message>User:</message>
    </field-validator>
  </field>
</validators>
```

```
private User user = new User();
```

```
public User getUser() {
  return user;
}
```

LoginAction.java

```
public void setUser(User user) {
  this.user = user;
}
```

```
private String name, password, address;
private int age;
```

```
public String getAddress() {
  return address;
}
```

User.java

```
public void setAddress(String address) {
  this.address = address;
}
```

af

```
.....|
```

Validation

■ Client Validation

- Create a Client-Side validation workflow by three steps
 - Set the **validate** attribute of action-form to be true
 - A correct action and namespace attributes must be provided to the `<s:form>` tag
 - Use the **default xhtml theme** for action-form
 - Create the action validation xml to configure the validators to be used



Client Validation Demo

```
<form id="login" name="login" onsubmit="return validateForm_login();"
action="/Struts2Project/login.action" method="post" mehtod="post"
onreset="clearErrorMessages(this);clearErrorLabels(this);"
>
```

```
<s:form action="login" mehtod="post" validate="true">
  <s:textfield name="name" label="用户名"></s:textfield>
  <s:textfield name="password" label="密码"></s:textfield>
  <s:textfield name="age" label="年龄"></s:textfield>
  <s:textfield name="address" label="地址"></s:textfield>
  <s:submit value="登录" method="login"></s:submit>
  <s:submit value="注册" method="register"></s:submit>
</s:form>
```

```
<validators>
  <field name="name">
    <field-validator type="requiredstring">
      <message>need a value</message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="requiredstring">
      <message>password is required!</message>
    </field-validator>
  </field>
  .....
</validators>
```

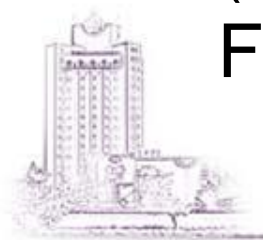
```
<script type="text/javascript">
  function validateForm_login() {
    form = document.getElementById("login");
    clearErrorMessages(form);
    clearErrorLabels(form);

    var errors = false;
    var continueValidation = true;
    // field name: name
    // validator name: requiredstring
    if (form.elements['name']) {
      field = form.elements['name'];
      var error = "need a value";
      if (continueValidation && field.value != null &&
        (field.value == "" || field.value.replace(/\s+|\s$/g, "").length ==
        0)) {
        addError(field, error);
        errors = true;
      }
    }
  }
}
```

Validation

■ Field and Non Field Validation

- There are two ways you can define validators in your -validation.xml file
 - <validator>
 - <field-validator>
- Non-Field-Validator: The <validator> element allows you to declare both types of validators (either a plain Validator or a field-specific FieldValidator)



Validation

- Field and Non Field Validation (cont.)
 - FieldValidators defined within a <field-validator> element will have their fieldName automatically filled with the value of the parent <field> element's fieldName attribute
 - It is always better to use field validator than non field validator
 - Plain validator takes precedence over field-validator



Field Validator VS Non Field Validator

```
<field name="email_address">
  <field-validator type="required">
    <message>You cannot leave the email address field empty. </message>
  </field-validator>
  <field-validator type="email">
    <message>The email address you entered is not valid. </message>
  </field-validator>
</field>

<validator type="required">
  <param name="fieldName">email_address</param>
  <message>You cannot leave the email address field empty. </message>
</validator>
<validator type="email">
  <param name="fieldName">email_address</param>
  <message>The email address you entered is not valid. </message>
</validator>
```

Validation

■ Short-Circuiting Validator

- Plain validator get validated first in the order they are defined and then the field-validator in the order they are defined
- Failure of a particular validator marked as short-circuit will prevent the evaluation of subsequent validators
- A FieldValidator that gets short-circuited will only prevent other FieldValidators for the same field from being evaluated

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Short-Circuiting Validator Demo

Since Plain Validator 2 is short-circuited, if its validation failed, it will causes Field validators for email field and Field validators for email2 field to not be validated as well

轻量级

```
<validators>
  <!-- Field Validators for email field -->
  <field name="email">
    <field-validator type="required" short-circuit="true">
      <message>You must enter a value for email. </message>
    </field-validator>
    <field-validator type="email" short-circuit="true">
      <message>Not a valid e-mail. </message>
    </field-validator>
  </field>
  <!-- Field Validators for email2 field -->
  <field name="email2">
    <field-validator type="required">
      <message>You must enter a value for email2. </message>
    </field-validator>
    <field-validator type="email">
      <message>Not a valid e-mail2. </message>
    </field-validator>
  </field>
  <!-- Plain Validator 1 -->
  <validator type="expression">
    <param name="expression">email.equals(email2)</param>
    <message>Email not the same as email2</message>
  </validator>
  <!-- Plain Validator 2 -->
  <validator type="expression" short-circuit="true">
    <param name="expression">email.startsWith('mark')</param>
    <message>Email does not start with mark</message>
  </validator>
</validators>
```


Configuration Files & Packages

■ Configuration Files

- From a Struts developer point of view, the one required configuration file used by the framework is web.xml
- By default, Struts will load a set of internal configuration files to configure itself
 - struts.xml
 - struts.properties
 - struts-default.xml
 - Etc.

轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Configuration Files & Packages

■ Configuration Files (cont.)

■ web.xml

- The web.xml web application descriptor file represents the core of the Java web application, so it is appropriate that it is also part of the core of the Struts framework
- In the web.xml file, Struts defines its FilterDispatcher, the Servlet Filter class that initializes the Struts framework and handles all requests



web.xml Demo

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <display-name>Struts2Project</display-name>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <welcome-file-list>
    <welcome-file>/jsp_files/register.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Handles both the preparation and execution phases of the Struts dispatching process

Configuration Files & Packages

■ Configuration Files (cont.)

■ struts.xml

- The core configuration file for the framework is the default (struts.xml) file and should reside on the classpath of the webapp (generally /WEB-INF/classes)
- We can break up a large struts.xml file into smaller pieces
 - By include
 - By jar



Struts Include Demo

```
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <include file="Home.xml"/>
  <include file="Hello.xml"/>
  <include file="Simple.xml"/>
  <include file="/util/POJO.xml"/>
  <include file="/com/initech/admin/admin-struts.xml"/>
</struts>
```



Configuration Files & Packages

■ Configuration Files (cont.)

■ struts.xml (cont.)

- The framework uses struts.xml to initialize its own resources. These resources include
 - Administrative Elements
 - package, bean, include, constant, etc.
 - Request Handling Elements
 - action, result, interceptor, etc.
 - Error Handling Elements
 - exception



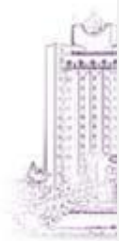
struts.xml Demo

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="basicstruts2" extends="struts-default">

        <action name="login" class="water.action.LoginAction">
            <result name="login">/jsp_files/index.jsp</result>
            <result name="register">/jsp_files/register.jsp</result>
            <result name="success">/jsp_files/success.jsp</result>
            <result name="input">/jsp_files/index.jsp</result>
        </action>

    </package>

    <package name="custom" extends="struts-default" namespace="/cus">
        <action name="fail">
            <result>/jsp_files/failed.jsp</result>
        </action>
    </package>
</struts>
```



exception configuration Demo

```
<package name="default">
  ...
  <global-results>
    <result name="login" type="redirect">/Login.action</result>
    <result name="Exception">/Exception.jsp</result>
  </global-results>

  <global-exception-mappings>
    <exception-mapping exception="java.sql.SQLException" result="SQLException"/>
    <exception-mapping exception="java.lang.Exception" result="Exception"/>
  </global-exception-mappings>

  ...
  <action name="DataAccess" class="com.company.DataAccess">
    <exception-mapping exception="com.company.SecurityException" result="login"/>
    <result name="SQLException" type="chain">SQLExceptionAction</result>
    <result>/DataAccess.jsp</result>
  </action>
  ...
</package>
```



Configuration Files & Packages

■ Configuration Files (cont.)

■ struts.properties

- The framework uses a number of properties that can be changed to fit your needs
- To change any of these properties, specify the property key and value in an struts.properties file
- The properties file can be locate anywhere on the classpath, but it is typically found under /WEB-INF/classes



struts.properties

Demo

```
struts.devMode = false

### when set to true, resource bundles will be reloaded on _every_ request.
### this is good during development, but should never be used in production
struts.i18n.reload=false

### Standard UI theme
### Change this to reflect which path should be used for JSP control tag templates by default
struts.ui.theme=xhtml
struts.ui.templateDir=template
#sets the default template type. Either ftl, vm, or jsp
struts.ui.templateSuffix=ftl

### Configuration reloading
### This will cause the configuration to reload struts.xml when it is changed
struts.configuration.xml.reload=false

### Location of velocity.properties file. defaults to velocity.properties
struts.velocity.configfile = velocity.properties
```



轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Configuration Files & Packages

■ Configuration Files (cont.)

■ struts-default.xml

- This file 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
- To exclude the struts-default.xml or to provide your own version, use struts.properties



struts-default.xml

Demo

```
<!-- Silly workarounds for OGNL since there is currently no way to flush its internal caches -->
<bean type="ognl.PropertyAccessor" name="java.util.ArrayList" class="com.opensymphony.xwork2.ognl.accessor.XWorkListPropertyAccessor" />
<bean type="ognl.PropertyAccessor" name="java.util.HashSet" class="com.opensymphony.xwork2.ognl.accessor.XWorkCollectionPropertyAccessor" />
<bean type="ognl.PropertyAccessor" name="java.util.HashMap" class="com.opensymphony.xwork2.ognl.accessor.XWorkMapPropertyAccessor" />

<package name="struts-default" abstract="true">
  <result-types>
    <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/>
    <result-type name="dispatcher" class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/>
    <result-type name="freemarker" class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
    <result-type name="httpheader" class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
    <result-type name="redirect" class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
    <result-type name="redirectAction" class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
    <result-type name="stream" class="org.apache.struts2.dispatcher.StreamResult"/>
    <result-type name="velocity" class="org.apache.struts2.dispatcher.VelocityResult"/>
    <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/>
    <result-type name="plainText" class="org.apache.struts2.dispatcher.PlainTextResult" />
  </result-types>

  <interceptors>
    <interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
    <interceptor name="autowiring" class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
  </interceptors>
</package>
```



轻量级J2EE框架

朱洪军

<http://staff.ustc.edu.cn/~waterzhj>



Configuration Files & Packages

■ Packages

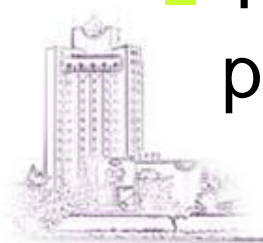
- Packages are a way to group actions, results, result types, interceptors, and interceptor-stacks into a logical configuration unit
- Conceptually, packages are similar to objects in that they can be extended and have individual parts that can be overridden by "sub" packages



Configuration Files & Packages

■ Packages

- The package element has one required attribute, name, which acts as the key for later reference to the package
- The extends attribute is optional and allows one package to inherit the configuration of one or more previous packages
- The optional abstract attribute creates a base package that can omit the action configuration



package "extends" demo

```
<struts>
  <package name="employee" extends="struts-default, json-default" namespace="/employee">

    <action name="list" method="list" class="org.apache.struts2.showcase.action.EmployeeAction" >
      <result>/empmanager/listEmployees.jsp</result>
      <result type="json">
        <param name="root">employees</param>
      </result>
    </action>

  </package>
</struts>
```



Configuration Files & Packages

■ Packages

- The namespace attribute subdivides action configurations into logical modules, each with its own identifying prefix
 - Default Namespace
 - The default namespace is "" - an empty string
 - If an action configuration is not found in a specified namespace, the default namespace is also be searched
 - Root Namespace
 - A root namespace ("/") is also supported
 - The root is the namespace when a request directly under the context path is received



Namespace Demo

```
<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>
```

Conclusions

- Key Core Features
- Request Processing Life-cycle
- **Action & Result & Result type**
- **Interceptor**
- **Validation**
- Configuration Files & Packages

