Desenvolvimento WEB II

Struts 2

Sérgio Cerqueira
sergioruivace@les.inf.puc-rio.br

Agenda

- Introdução
- Struts e a Web
- Contextualização
- Arquitetura
- Configuração
- Action
- Result Type
- Interceptor
- Value Stack / OGNL
- Tag Library
- Request e dados do formulário
- Acessando a Camada de Négocio
- Validação
- Internacionalização

Introdução

- Struts
  - Lançado em maio de 2000
  - Grande quantidade de projetos em fase de manutenção
  - "dot com boom"
  - Solução bem vinda
- Struts 2
  - Difícil evolução a partir do struts
  - WebWork

Introdução

- Objetivo de promover um desenvolvimento Web mais fácil
- Características
  - Redução de configuração de XML
  - Anotações
  - “Codding by Convension” – Programação por Convenção
  - Fusão entre actions e forms
  - Actions são POJOs
  - Criação dos Interceptors

Introdução

- Características – Continuação
  - Opções de plugins
  - Opções de "Result Types"
    - JSPs
    - Velocity
    - Outros
  - "Dependency Injection" – Injeção de Dependencia

Struts e a Web

- Frameworks para Web

- Motivos para utilizar o struts
  - Framework Baseado em ações
  - Maduro com comunidade vibrante
  - Opção de utilizar configuração via XML e anotações
  - Ação baseada em POJO são fáceis para testar
  - Integração com Spring, SiteMesh e Tiles
  - Integração com OGNL
  - Bibliotecas baseadas em Temas e tags em AJAX
  - Múltiplas opções de visualização (JSP, Velocity, ...)
  - Plugins para estender e modificar as características do Struts

Contextualização

- Servlets
  - Oferecem um caminho para mapear uma URL para uma classe específica
- JSP e desenvolvimento em Scriplet
  - Provê a lógica para o processamento do request e a lógica de apresentação
  - "Cut-and-paste code" e manutenção
  - Objetos comuns de formatação
- TagLibs
  - Facade
  - Objeto comum de formatação
  - Criação de taglibs próprias

Contextualização

- MVC e MVC2
  - MVC

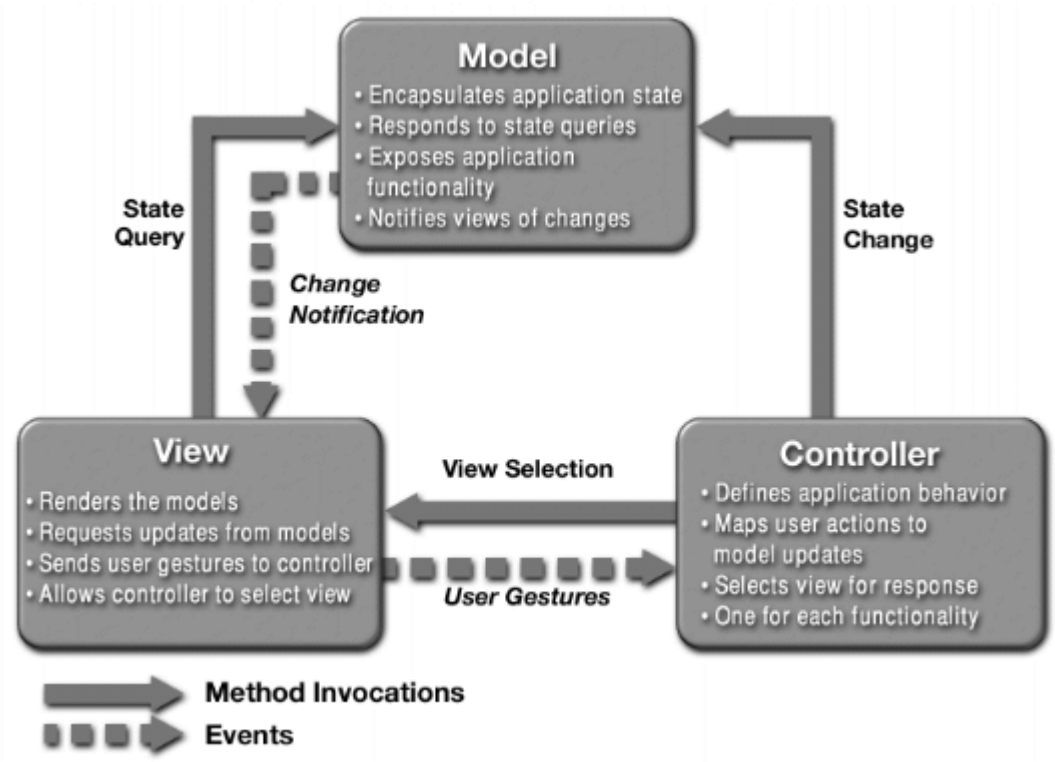
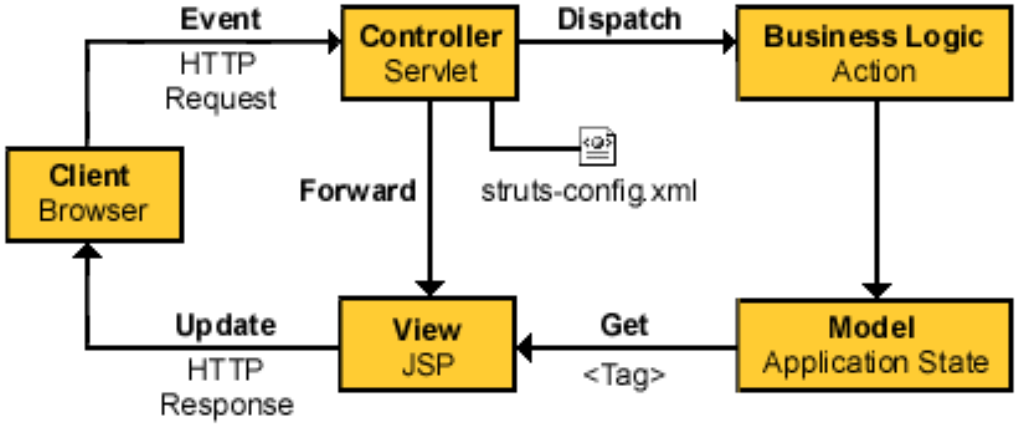


Figura: http://java.sun.com/developer/technicalArticles/javase/mvc/index.html

Contextualização

- MVC2 no Struts



- Framework Baseado em Ação(Spring, Struts, ...)
  - A açãoExecuta uma funcionalidade específica para uma dada URL
  - Os JSPs tratam da apresentação
  - Deixa cada parte fazer o que fazem de melhor

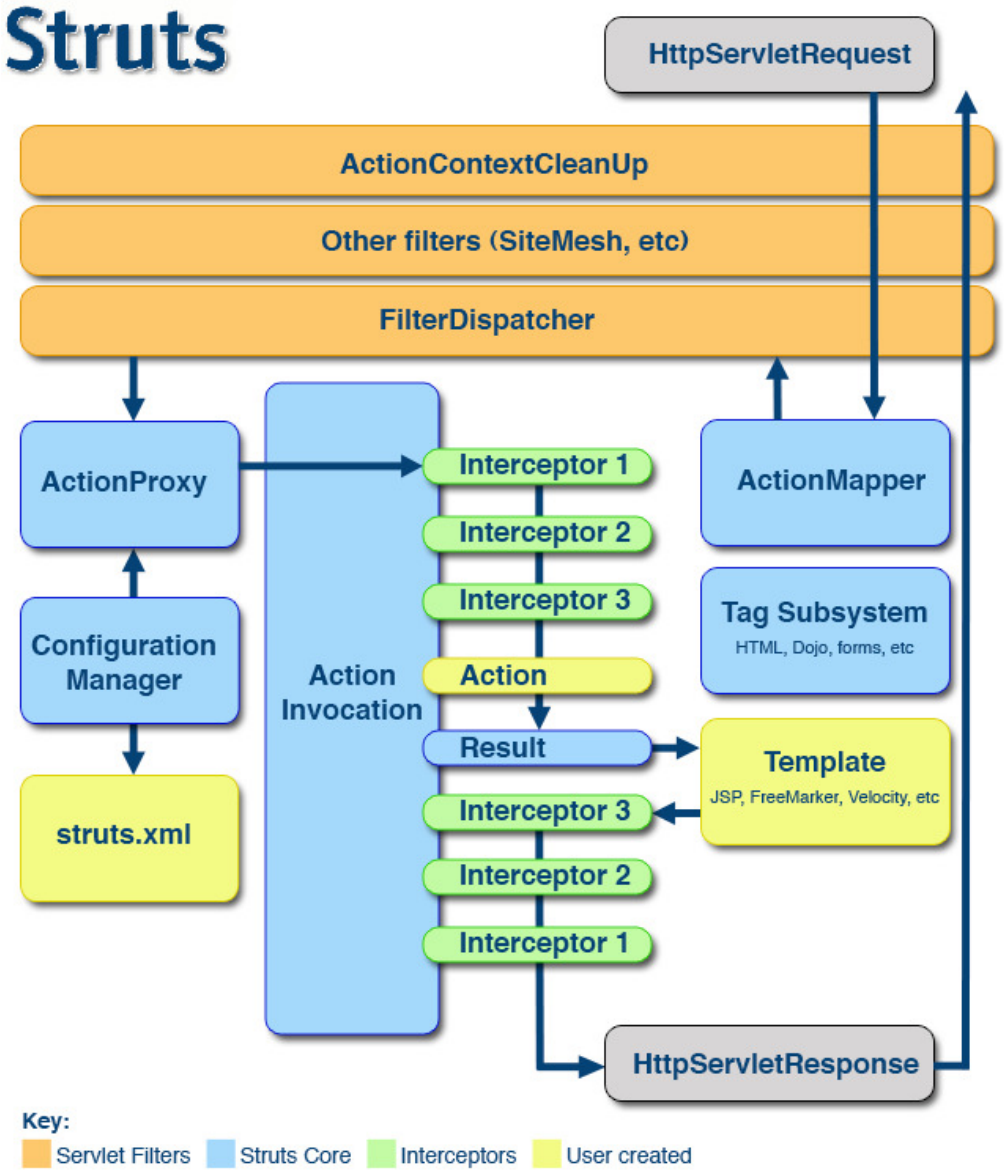
Figura: http://www.ibm.com/developerworks/library/j-struts/

Contextualização

- Frameworks Baseados em Componentes(JSF, Tapestry,...)
  - Complexidade das aplicações e das paginas
  - Componentes de Interface e Classes
  - Reuso

- AJAX(Asynchronous Javascript and XML)
  - Requisições em pequenas quantidades
  - Requisição e resposta via XML
  - Dados "setados" via Javascript

Arquitetura



- No diagrama, um primeiro pedido vai para o Servlet container que é transmitido através de uma cadeia de filtros padrões. A cadeia inclui o (opcional) filtro ActionContextCleanUp.
- Em seguida, o obrigatório FilterDispatcher é chamado, que por sua vez consultará o ActionMapper para determinar se o pedido deve invocar uma ação.
- Se o ActionMapper determina que uma ação deve ser invocada, o FilterDispatcher delega o controle ao ActionProxy.

Figura: http://struts.apache.org/2.x/docs/big-picture.html

Arquitetura

- O ActionProxy consulta o framework Configuration Manager (inicializado a partir do arquivo struts.xml).
- Em seguida, o ActionProxy cria um ActionInvocation, que é responsável pela execução do "command pattern". Isso inclui invocar qualquer Interceptor e mais a frente invocando a Action em si.
- Depois do retorno da Action, o ActionInvocation é responsável por analisar-se há um result associado à "Action result code" mapeados no struts.xml.
- O resultado é então executado, que muitas vezes (mas nem sempre, como é o caso de Action Chaining) envolve um Template escrito em JSP ou FreeMarker a serem renderizados.
- Enquanto está renderizando, os Templates podem usar o Struts Tags fornecidos pelo framework. Alguns desses componentes irão trabalhar com o ActionMapper a tornar adequada URLs de requisições adicionais.

Arquitetura

- Os Interceptors são executadas novamente (em ordem inversa).
- Finalmente, a resposta retorna através dos filtros configurados no web.xml. Se o filtro ActionContextCleanUp está presente, o FilterDispatcher não irá limpar a Thread local ActionContext. Se o filtro ActionContextCleanUp não está presente, o FilterDispatcher irá limpar todas as Threads locais.
- Importante
  - Todos os objetos nesta arquitetura (Actions, Results, Interceptors, e assim por diante) são criados por um **ObjectFactory**.
  - Este **ObjectFactory** é "plugável".
  - Nós podemos fornecer os nossos próprios **ObjectFactory** por qualquer motivo. A única exigência é saber quando objetos no framework são criados.
  - Uma implementação popular do **ObjectFactory** é o Spring fornecido pelo Spring Plugin.

Configuração

- Localização dos arquivos

Arquivo	Opcional	Localização (Relativa à webapp)	Localização (Relativa ao projeto)	Propósito
web.xml	Não	/WEB-INF/	\WebContent\WEB-INF	Descritor de implantação WEB para incluir todos os componentes necessários no framework
struts.xml	Sim	/WEB-INF/classes/	\src	Configuração principal, contém result/view types, action mappings, interceptors, e outros
struts.properties	Sim	/WEB-INF/classes/	\src	Propriedades do framework
applicationContext.xml	Sim	/WEB-INF/	\WebContent\WEB-INF	Configuração do spring

Configuração

- Documentação sobre configurações do Struts2 pode ser vista em: http://struts.apache.org/2.x/docs/configuration-files.html

- Adicionando o framework ao projeto
  - Colocar o Jar "struts2-core.jar"
  - Ou configurar o pom.xml no projeto maven

```
<dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-core</artifactId>
        <version>2.0.11</version>
</dependency>
```

- Configurando o web.xml

Configuração

```
<filter>
        <filter-name>action2</filter-name>
        <filter-class>
                org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
</filter>
<filter-mapping>
        <filter-name>action2</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

- O arquivo struts.properties
  - Pode mudar as configurações defaut do framework
  - Sobrescreve o default.properties

Configuração

struts.devMode = **true** – Habilita o modo de desenvolvimento, útil para fazer debug
struts.url.http.port = **8080** – diz a porta em que o servidor está rodando (para a geração das URLs corretamente)
struts.objectFactory = **spring** – diz qual é a fabrica dos objetos
struts.action.extension = **html** – diz quais extensões que o ActionMapper irá aceitar (aceita vários separados por vírgula)

- O arquivo struts.xml
  - Contêm informações referentes às Actions

```
<?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>
        <package name="struts2" extends="struts-default" namespace="/">
                ...
        </package>
</struts>
```

Configuração

1. A Tag Include
   - Utilizada para deixar o projeto modular
   - Somente tem um atributo “file”
   - A ordem da inclusão é importante (referencia)

```
<struts>
        <include file=“colaborador.xml" />
        <include file=“projeto.xml" />
        <include file=“publicacao.xml" />
</struts>
```

2. A Tag Package
   - Usado para agrupar elementos com atributos comuns
   - Os atributos desta tag são:
   1. name – nome útino para o pacote
   2. extends – o nome do pacote que será estendido
   3. namespace – prove um mapeamento da URL para o pacote
   4. abstract – desabilita o acesso às Actions

Action

- Mais básica unidade de trabalho que pode ser associada a uma HTTP request.
- A Action não precisa estender outra classe e não é necessário implementar nenhuma interface.
- A classe é um simples POJO
- Porem há a possibilidade de estender classes e de implementar interfaces e o framework e os plugins oferecem várias.
- A Action pode ter um Single Result, um Multiple Result e usar Wildcards

Action

1. Single Result

```
class ProjetoAction {
        public void String execute() throws Exception {
        return " sucesso";
        }
}
```

- A classe tem um único método chamado "execute", que pode ser alterado
- A String resultado do método deverá bater com a configuração da Action no struts.xml, para um Result específico ser renderizado para o usuário

```
<action name="projeto" class="br.puc.rio.inf.les.prds.sgpa.action.Projeto">
        <result>projeto.jsp</result>
</action>
```

- O atributo "class" fornece o caminho completo para a classe da Action
- O atributo "name" fornece a informação da URL para executar a Action

```
http://localhost:8080/SGPA/projeto.action
```

Action

2. Multiple Results

```
class ProjetoAction {
        public void String execute() throws Exception {
                if( minhaLogicaFuncionou() ) {
                        return "sucesso";
                } else {
                        return “erro";
                }
        }
}
```

- Como há dois possíveis resultados, eles devem ser mapeados na configuração da Action

```
<action name=“projeto" class=" br.puc.rio.inf.les.prds.sgpa.action.Projeto">
        <result>projeto.jsp</result>
        <result name=“erro">erro.jsp</result>
</action>
```

3. Wildcards
   - Com o crescimento da aplicação, o numero de mapeamentos cresce extremamente

Action

- Utiliza o "Pattern Matching", Casamento de Padrões para fazer o mapeamento
- Se a aplicação utiliza o padrão "/{entidade}/{ação}.action", as URLs seriam "/Projeto/listar.action" ou "/Aluno/adicionar.action"

```
<action name="/*/*" method="{2}" class="br.puc.rio.inf.les.prds.sgpa.action.{1}Action">
        <result name="adicionar">/{1}/adicionar.jsp</result>
        <result name="listar">/{1}/listar.jsp</result>
</action>
```

- Cada asterisco se chama wildcard
- Deve habilitar a opção no struts.properties

```
struts.enable.SlashesInActionNames = true
```

- Ser a URL "http://localhost:8080/SGPA/Projeto/listar.action" fosse passada as wildcards seriam substituídas por:

```
<action name="/Projeto/listar" method="listar" class=" br.puc.rio.inf.les.prds.sgpa.action.ProjetoAction">
        <result name="adicionar">/Projeto/adicionar.jsp</result>
        <result name="listar">/Projeto/listar.jsp</result>
</action>
```

Result Type

- Um resultado gerado pela Action não precisa necessariamente ser um JSP
- Para informar o tipo do resultado, é usado o atributo type na tag result
- O valor default é "dispatcher" que renderiza JSP

```
<action name="projeto" class="br.puc.rio.inf.les.prds.sgpa.action.Projeto">
        <result type="dispatcher">projeto.jsp</result>
</action>
```

- O framework oferece várias implementações da Interface Result
  1. Chain Result
  2. Dispatcher Result
  3. FreeMarker Result

Result Type

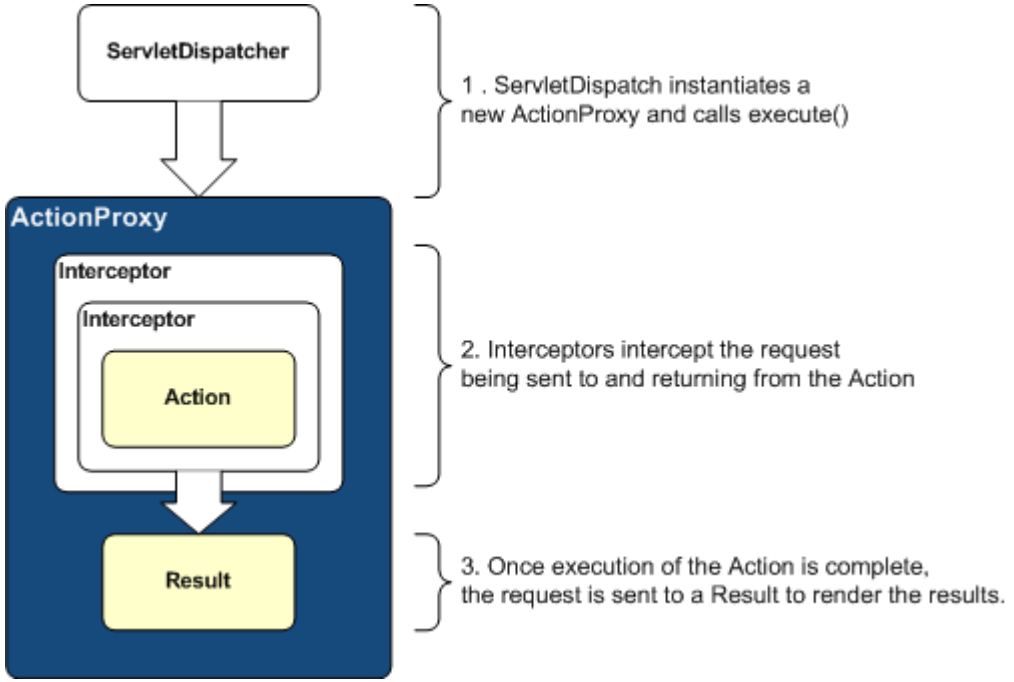
4. HttpHeader Result
5. Redirect Result
6. Redirect Action Result
7. Stream Result
8. Velocity Result
9. XSL Result
10. PlainText Result
11. S2PLUGINS: Tiles Result

- É possível implementar o próprio Result Type
- É possível alterar o Result Type default

```
<result-types>
        <result-type name="dispatcher“
        class="org.apache.struts2.dispatcher.ServletDispatcherResult“/>
        <result-type name=“meuresultado“
        class="br.puc.rio.inf.les.prds.sgpa.ServletMeuResultado"  default="true"/>
</result-types>
```

Interceptor

- Os interceptors são executados antes e depois de uma Action ser chamada
- São "plugaveis"
- Podem ser ordenados
- Eles têm acesso à Action em execução



Interceptor

- São úteis para:
  1. Proteção contra submits duplos
  2. Conversão de tipos
  3. Criação e inserção de dados em objetos
  4. Validação
  5. Upload de arquivo
  6. Preparação de página
  7. Outros

- Há a possibilidade da criação de novos Interceptors

- Existe uma ampla variedade de Interceptors com finalidades diversas, já fornecidos pelo framework

Interceptor

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.

Interceptor

Interceptor	Name	Description
Logger Interceptor	logger	Outputs the name of the Action.
Message Store Interceptor	store	Store and retrieve action messages / errors / field errors for action that implements ValidationAware interface into session.
Model Driven Interceptor	model-driven	If the Action implements ModelDriven, pushes the getModel Result onto the Value Stack.
Scoped Model Driven Interceptor	scoped-model-driven	If the Action implements ScopedModelDriven, the interceptor retrieves and stores the model from a scope and sets it on the action calling setModel.
Parameters Interceptor	params	Sets the request parameters onto the Action.
Prepare Interceptor	prepare	If the Action implements Preparable, calls its prepare method.
Scope Interceptor	scope	Simple mechanism for storing Action state in the session or application scope.
Servlet Config Interceptor	servletConfig	Provide access to Maps representing HttpServletRequest and HttpServletResponse.
Static Parameters Interceptor	staticParams	Sets the struts.xml defined parameters onto the action. These are the <param> tags that are direct children of the <action> tag.
Roles Interceptor	roles	Action will only be executed if the user has the correct JAAS role.
Timer Interceptor	timer	Outputs how long the Action takes to execute (including nested Interceptors and View)
Token Interceptor	token	Checks for valid token presence in Action, prevents duplicate form submission.

Interceptor

Interceptor	Name	Description
Token Session Interceptor	tokenSession	Same as Token Interceptor, but stores the submitted data in session when handed an invalid token
Validation Interceptor	validation	Performs validation using the validators defined in *action*-validation.xml
Workflow Interceptor	workflow	Calls the validate method in your Action class. If Action errors are created then it returns the INPUT view.
Parameter Filter Interceptor	N/A	Removes parameters from the list of those available to Actions
Profiling Interceptor	profiling	Activate profiling through parameter

A tabela se encontra em: http://struts.apache.org/2.x/docs/interceptors.html

Interceptor

- Configurando
  - A tag <package> deve estender o “struts-default”
  - Os interceptors devem ser definidos na tag <interceptors> dentro da tag <package>
  - Com os interceptors definidos basta referenciá-los na Action

```
<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>
```

  - É possível referenciar UM interceptor para todas as Actions do pacote

```
<default-interceptor-ref name=" logger"/>
```

Interceptor

- Mas geralmente uma Action necessita de vários Interceptors
- Para resolver essa limitação foram criadas as pilhas de Interceptors "Interceptors staks"

```
<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>
```

- É possível referenciar UMA pilha de interceptors para todas as Actions do pacote

```
<default-interceptor-ref name=" myStack"/>
```

Value Stack / OGNL

- Value stack é exatamente o que diz uma pilha de objetos
- Object Graph Navigation Language(OGNL) oferece um meio unificado para acessar objetos dentro da value stack
- O Value stack oferece os seguintes objetos:
  1. Objetos Temporais – durante a execução esses objetos são colocados na value stack; um exemplo seria o valor atual de uma iteração em um JSP
  2. O objeto do modelo – se objetos do modelo estão sendo usados, o objeto atual do modelo é colocado antes da Action
  3. O objeto da Action – a Action que está sendo executada
  4. Named Objects – esses objetos incluem #aplication, #session, #request, #attr e #parameters

Value Stack / OGNL

- O meio mais fácil de acessar a Value Stack é a travez de tags pelo JSP, Velocity ou Freemarker
- HTML tags são geralmente utilizadas para acessar propriedades de objetos
- Tags de controle são usadas com expressões
- Tags de dados são utilizadas para manipular a pilha(set, push)
- Não é necessário saber em qual escopo o objeto está
- Se houver dois atributos com o mesmo identificador, sempre será retornado o primeiro que for encontrado
- Com OGNL se souber a posição do objeto, o mesmo pode ser acessado usando "[2].id"

Tag Library

- Antes da tag ser acessada, uma tag library deve ser definida no JSP

```
<%@taglib prefix="s" uri="/struts-tags" %>
```

```
<s:textfield label="Name" name="person.name"/>
```

- O framework oferece vária tag libraries acessíveis em JSP, Freemarker e Velocity
- Por default o atributo “value” das form tags e muitos outros atributos de outras tags aceitam OGNL
- Se o atributo não tiver aceitar OGNL como defautl(como o atributo label), basta colocar a expressão dentro “%{” e “}”
- As tags de ajax podem operar tanto no modo de ajax ou no de sem ajax

Tag Library

Form Tags	Non-Form UI Tags	Ajax Tags
• checkbox • checkboxlist • combobox • doubleselect • head • file • form • hidden • label • optiontransferselect • optgroup • password • radio • reset • select • submit • textarea • textfield • token • updownselect	• actionerror • actionmessage • component • div • fielderror	• a • autocompleter • bind • datetimepicker • div • head • submit • tabbedpanel • textarea • tree • treenode

Figura1: http://struts.apache.org/2.x/docs/ui-tag-reference.html

Control Tags	Data Tags
• if • elseIf • else • append • generator • iterator • merge • sort • subset	• a • action • bean • date • debug • i18n • include • param • property • push • set • text • url

Figura2: http://struts.apache.org/2.x/docs/generic-tag-reference.html

Request e dados do Formulário

- Para acessar um dado, é necessário criar um getter e/ou setter para o mesmo
- Cada string de request e valores de um formulário são um simples par nome/valor

```
"/aluno.action?nome=doug&idade=15"
```

- A Action deve ter um setNome(String nome) e um setIdade(int idade)
- O framework converte automaticamente para o tipo no método da Action, desde que ele seja de um tipo primitivo ou objetos básicos.
- Pode ser criado um conversor para tipos complexos
- Struts2 ajuda na navegação por objetos mais complexos

```
"aluno.endereco.cep=00000000"
```

Request e dados do Formulário

- Seria equivalente a:

```
getAluno().getEndereco.setCep(00000000)
```

- A Action seria parecida com essa

```
public class AlunoAction
{
        private Aluno aluno;

        public AlunoAction()
        {}
        public String execute() {
                return "sucesso";
        }
        public Aluno getAluno() {
                return aluno;
        }
        public void setAluno(Aluno aluno) {
                this.aluno = aluno;
        }
}
```

- Para acessar o nome do aluno em um JSP basta utilizar uma tag

```
<s:property value="aluno.nome"/>
```

Acessando a camada de Negócio

- Para fornecer um baixo acoplamento, o Struts 2 usa uma técnica chamada Injeção de Dependência (DI), ou Inversão de controle (IoC).
- O Spring Framework é o plug-in preferencial para a injeção de dependência
- Há outras opções disponíveis
  - The Plexus
  - EJB3
- Para utilizar o Spring é necessário:
  1. Baixar struts2-spring-plugin.jar e colocar na pasta lib
  2. Registrar o listener do Spring no web.xml

```
<listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

Acessando a camada de Negócio

3. Falar qual é o construtor que o Struts irá utilizar no struts.properties

```
struts.objectFactory=spring
```

4. Deve ser criado um arquivo com o nome applicationContext.xml, que irá conter a configuração do Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
        <bean id="projetoDAO"class="..." />
        <bean id="colaboradorDAO"class="..." />
        <bean id="projetoFacade"class="..."  autowire="constructor" />
        <bean id="projetoAction"class="..." autowire="constructor" />
</beans>
```

- A configuração do Spring é baseada em beans

Acessando a camada de Negócio

- Com isso o Spring fica responsável por criar a sua classe e injetá-la de acordo com a configuração
- Uma característica importante é o modo de como ocorre a injeção
  - Name
  - Type
  - Constructor
  - Auto
- É possivel quebrar o arquivo applicationContext.xml em outros menores, basta configurar no web.xml:

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                classpath*:applicationContext*.xml
        </param-value>
</context-param>
```

Acessando a camada de Negócio

- Uma classe ficaria assim:

```
public class ProjetoAction {
        private ProjetoFacade projetoFacade;
        public ProjetoAction(ProjetoFacade projetoFacade) {
                this.projetoFacade_= projetoFacade;
        }
        public String execute() {
        …
        }
}
```

Validação

- Há várias formas de fazer validação no framework
  - Básica
  - No cliente
  - Via ajax
  - Por anotação
  - Outras
- Para uma validação mais complexa o ideal é implementar o método validate() na action e estender a classe ActionSupport
- Para uma validação mais simples o ideal é:
  1. criar um arquivo xml no mesmo pacote da Action com o nome "MinhaAction-validation.xml"
  2. Incluir a pilha de Interceptors "defautlStack" na Action, pois ela contem os Interceptors necessários para a validação

Validação

- O arquivo xml é simples e auto-explicativo

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
        <field name="nome">
                <field-validator type="requiredstring">
                        <message>Você deve colocar um nome</message>
                </field-validator>
        </field>
        <field name="age">
                <field-validator type="int">
                        <param name="min">13</param>
                        <param name="max">19</param>
                        <message>Formulário para pessoas entre 13 e 19 anos</message>
                </field-validator>
        </field>
</validators>
```

- Cada campo pode ter um ou mais nós "field-validator"
- O nó message pode conter o atributo "key" útil para a internacionalização

Validação

Nome	Descrição
Conversion	Verifica quando existe um erro de conversão
Date	Checa se a data está em um intervalo específico
Double	Checa se o double está em um intervalo específico
Email	Verifica se o e-mail é válido
Expression	Avalia uma expressão OGNL
Fieldexpression	Valida um campo utilizando OGNL
Int	Checa se o int está em um intervalo específico
Regex	Checa se o valor de uma propriedade bate com uma expressão regular
Required	Verifica se uma propriedade não é nula
Requiredstring	Verifica se uma propriedade não é nula e não é vazia
Stringlength	Checa se o tamanho da string está em um intervalo específico
Url	Verifica se a propriedade é uma URL válida
Visitor	Permite redirecionar a valildação para as propriedades do objeto na Action

Internacionalização

- Struts2 oferece suporte para internacionalização via resource bundles, interceptors e tag libraries
- Resource Bundles
  - Pode ser feito com um arquivo contendo todo o texto ou com vários arquivos
  - Os arquivos podem ser nomeados e divididos em diversas formas
  - Há uma ordem de busca até uma chave ser encontrada
    1. Um properties para cada Action (MinhaAction.properties)
    2. Um properties para cada classe base na hierarquia da Action
    3. Um properties para cada interface e sub-interface
    4. Um properties para cada classe do modelo (se a Action é model-driven)
    5. Um properties chamado package.properties em cada pacote
    6. Arquivos de propriedades configurados no "struts.properties"

```
struts.custom.i18n.resources=testmessages,testmessages2
```

Internacionalização

```
br/
  puc/
    rio/
      inf/
        les/
          prds/
            sgpa/
              package.properties
              action/
                package.properties
                ProjetoAction.java
                ProjetoAction.properties
```

Se não existir o arquivo **ProjetoAction.properties,** o framework irá procurar pelo **package.properties** dentro do **.../prds/sgpa/action.**

Se não houver ele irá prcurar no **.../prds/sgpa** e assim sucessivamente

- O arquivo de propriedades consiste no par chave=texto
- É possível utilizar tokens

```
# Comentário
erro.salvar.arquivo                 = Erro ao salvar um arquivo.
erro.salvar.arquivo.nao.enviado     = Arquivo não informado.
label.nome.pessoa                   = Nome da  Pessoa:
label.nome.empresa                  = Nome da Empresa:
label.saudacao                      = Olá!
label.saudacao.nome                 = Olá {0}!
```

- Interceptors
  - Por default o framework colocar o locale do usuário na sessão, vindo do browser

Internacionalização

- Quando é necessário suporte para múltiplas línguas sem depender do browser, é possível utilizar o "i18n" interceptor
- O interceptor checa o parâmetro "request_locale" e salva as informações até o parâmetro ser alterado novamente

- Tag Libraries
  - Alem da internacionalização utiliza locales para formatação de dados como as datas
  - É necessário extender a classe ActionSupport
  - Um meio de utilizar a internacionalização é usando a tag "text"

```
<s:text name="label.saudacao"/>
<s:text name="label.saudacao.nome">
      <s:param >Mr Smith</s:param>
</s:text>
```

Internacionalização

- Outro meio é utilizando métodos da OGNL e a tag “property”

```
<s:property value="getText(‘label.saudacao.nome’)"/>
<s:property value="getText(‘l label.saudacao.nome’)">
    <s:param >Mr Smith</s:param>
</s:text>
```

- As tags que não aceitam OGNL por default pode usar “%{}”

```
<s:textfield label="%{getText(‘label.saudacao’)}"/>
```

- O atributo key na maioria das tags pode ser utilizada para recuperar uma mensagem do resource bundle

```
<s:textfield key="label.saudacao" name="textfieldName"/>
```

- Existem outras formas de utilizar internacionalização que não serão tratadas

Bibliografia

- **Struts, an open-source MVC implementation,** Malcolm G. Davis :http://www.ibm.com/developerworks/library/j-struts/
- **Designing Enterprise Applications with the J2EETM Platform, Second Edition**, Sun: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html#1080752
- **Starting with Struts 2**, Ian Roughley: http://www.infoq.com/minibooks/starting-struts2
- **Choosing a Java Web Framework,** Matt Raible: http://www.urlfan.com/local/choosing_a_java_web_framework/35225083.html
- **Apache Struts 2,** Apache**:** http://struts.apache.org/2.x/
- **Apache Struts 2 Documentation**, Apache**:** http://struts.apache.org/2.x/docs/home.html