Módulo 6

Programação WEB

Lição 7

Introdução a MVC e ao *Framework* Struts

Versão 1.0 - Nov/2007

Autor
Daniel Villafuerte

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Angelo de Oliveira
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

A arquitetura *Model-View-Controller* (MVC) é um padrão arquitetural comprovadamente eficaz em projetos de desenvolvimento. São definidos três componentes separados – *Model* (modelo), *View* (visão) e *Controller* (controle) – e dividi-se o projeto nestes componentes.

Ao final desta lição, o estudante será capaz de:

- Compreender o funcionamento da arquitetura MVC
- Utilizar o *framework Struts* no desenvolvimento de aplicações

2. Introdução à arquitetura Model-View-Controller

2.1. Motivação

Em toda aplicação, a parte do código mais sujeita a mudança é a porção da interface de usuário. A interface é o aspecto mais visível ao usuário e com o qual o usuário interage. Sendo assim, é o alvo mais provável de pedidos de mudança ou de melhorias da usabilidade.

Ter sua lógica de negócio firmemente acoplada com a interface de usuário leva a processos de alterações da interface mais complexos e sujeitos os erros. As mudanças a uma parte têm o potencial de trazer conseqüências em cascata no restante da aplicação.

2.2. Solução

O padrão MVC fornece uma solução para este problema dividindo a aplicação nos componentes *Model*, *View* e *Controller*, desacoplando estes de quaisquer outros ao fornecer um conjunto de recomendações sobre suas interações.

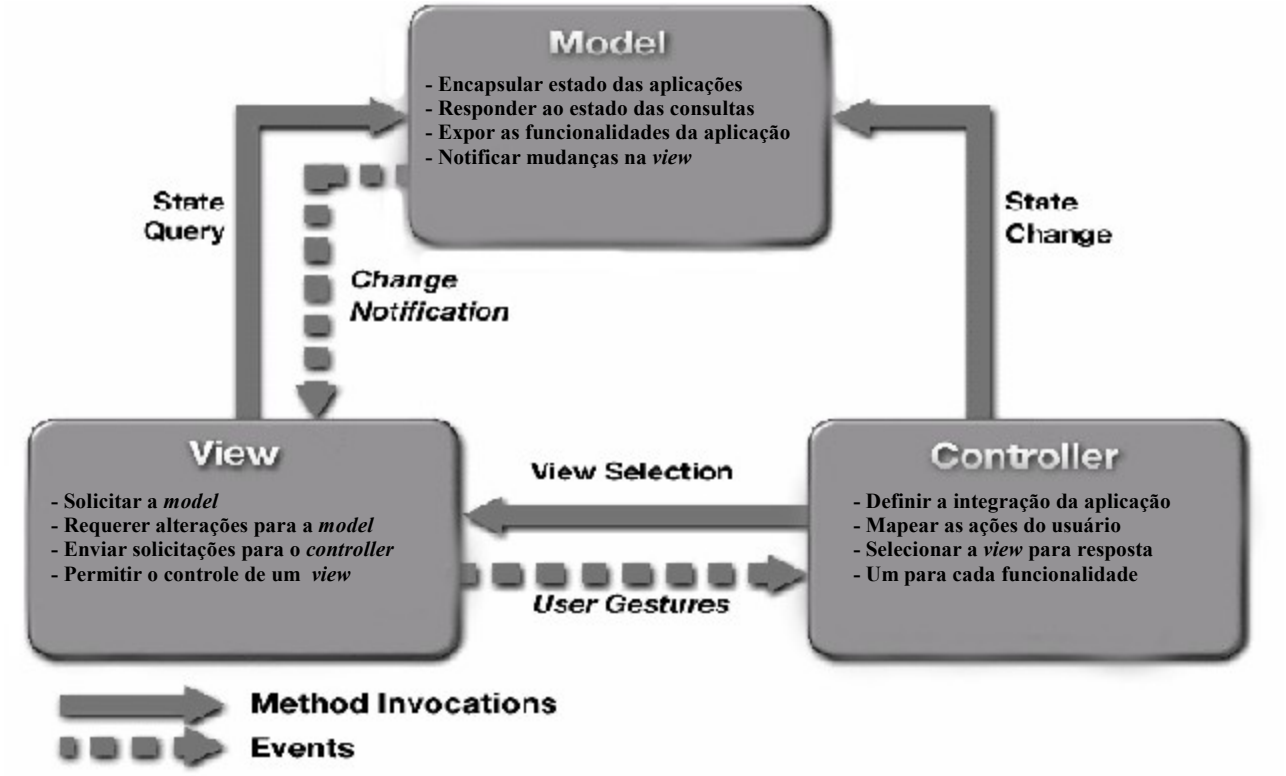

Figura 1: O padrão MVC

O diagrama acima mostra os três componentes definidos pelo padrão MVC assim como as suas interações previstas. Vamos analisar parte por parte.

2.3. Model

O padrão MVC define uma camada chamada *Model* que representa os dados usados pela aplicação, assim como as operações de negócio associadas a eles. Definindo a *Model* como uma camada separada, detalhes como recuperação, persistência e manipulação dos dados são abstraídas do restante da aplicação.

Há diversos benefícios com este tipo de abordagem. Primeiramente, isto assegura que os detalhes dos dados e das operações nos dados podem ser encontrados em uma área bem definida (a *Model*) em vez de estarem dispersos na aplicação. Isto prova-se benéfico durante a fase de manutenção da aplicação. Em segundo lugar, tendo-se os detalhes dos dados totalmente

separados de qualquer implementação de interface, os componentes da *Model* podem ser reaproveitados mais facilmente em outras aplicações que necessitam de uma funcionalidade similar.

2.4. View

Esta camada compreende todos os detalhes da implementação da interface de usuário. Aqui os componentes gráficos fornecem as representações do estado interno da aplicação e oferecem aos usuários as formas de interagir com a aplicação. Nenhuma outra camada interage com o usuário, somente a *View*.

Ter uma camada *View* separada, fornece diversos benefícios. Por exemplo, é mais fácil incluir a presença de um grupo de design separado na equipe de desenvolvimento. Este grupo de *design* pode se concentrar completamente no estilo, *look & feel* da aplicação sem ter que se preocupar a respeito de outros detalhes.

Além disso, ter uma camada *View* separada, torna possível fornecer múltiplas interfaces à aplicação. Considerando que a funcionalidade do núcleo da aplicação encontra-se em algum outro lugar (na *Model*), múltiplas interfaces podem ser criadas (baseadas no *Swing*, baseadas na WEB, baseadas na console), todas podem ter diferentes *look & feel* e todas podem simplesmente utilizar os componentes da *Model* com suas funcionalidades.

2.5. Controller

Por último, a arquitetura MVC inclui a camada do componente *Controller*. Esta camada contém detalhes sobre o fluxo de programa/transição da tela e é também responsável por capturar os eventos gerados pelo usuário na camada *View* e, possivelmente, atualizar os componentes da *Model* usando dados fornecidos pelo usuário.

Há diversos benefícios em se ter uma camada separada para a *Controller*. Primeiro, tendo um componente da aplicação separado para conter os detalhes da transição de tela, componentes da *View* podem ser projetados de maneira que não necessitem estar cientes um do outro. Isto facilita a múltiplas equipes independentes de desenvolvimento que trabalham simultaneamente. As interações entre os componentes da *View* são abstraídas na *Controller*.

Segundo, tendo uma camada separada que atualize os componentes da *Model*, detalhes são removidos da camada de apresentação. A camada de apresentação pode se especializar em sua finalidade preliminar de apresentar dados ao usuário. Os detalhes de como os dados do usuário mudam o estado da aplicação são escondidos dentro do componente da *Controller*. Isto fornece uma separação limpa entre a lógica de apresentação e a lógica de negócio.

Não podemos afirmar que o padrão MVC possua somente benefícios e nenhum efeito colateral. Dividir a aplicação em três componentes separados resulta em aumento de complexidade. Para pequenas aplicações que não se beneficiam do acoplamento fraco da *Model*, pode ser excessivo o uso deste padrão. Entretanto, é melhor lembrar que as aplicações freqüentemente começam pequenas e tornam-se sistemas complexos. Assim, deve-se sempre buscar o acoplamento fraco.

3. Arquitetura MVC para a WEB: A Arquitetura *Model 2*

A arquitetura MVC foi modificada ligeiramente e adaptada para o uso em aplicações WEB. A arquitetura resultante foi chamada, então, de arquitetura *Model* 2.

As aplicações *Model* 2 têm tipicamente o seguinte:

- Uma servlet *Controller* que fornece um ponto de acesso único ao restante da aplicação. Este *Controller* é responsável por fornecer a gerência central do fluxo da aplicação e dos serviços como a manipulação da segurança e a gerência do usuário. Este tipo de controlador é frequentemente chamado de *Front Controller*.
- A servlet *Controller* usa tipicamente configurações XML para determinar o fluxo da aplicação e o processamento do comando. Também emprega, geralmente, os componentes de ajuda que servem como objetos *Command*. Isto significa que tais componentes de ajuda estão associados com às ações do usuário e são criados/chamados para gerenciar aquelas ações enquanto ocorrem, chamando os componentes da *Model* quando necessário. Isto serve para desacoplar a servlet *Controller* da *Model*.

Usar tal arquitetura foi provado ser vantajoso para nossas aplicações. Implementá-la pode ser feito mais facilmente mediante o uso de *frameworks* existentes. Estes *frameworks* fornecem muitos dos detalhes de configuração de modo que possamos concentrar nossa atenção em tarefas mais importantes. Fornecem também úteis funcionalidades adicionais úteis.

Neste módulo, trataremos dos dois *frameworks* mais populares: *Struts* e *JavaServer Faces* (JSF). Debateremos primeiramente sobre o *Struts* em seguida a JSF.

4. STRUTS

Struts é um *framework* de código aberto que é disponibilizado e gerenciado pela *Apache Software Foundation*. Temos abaixo uma representação de como o *Struts* gerencia a arquitetura *Model* 2:

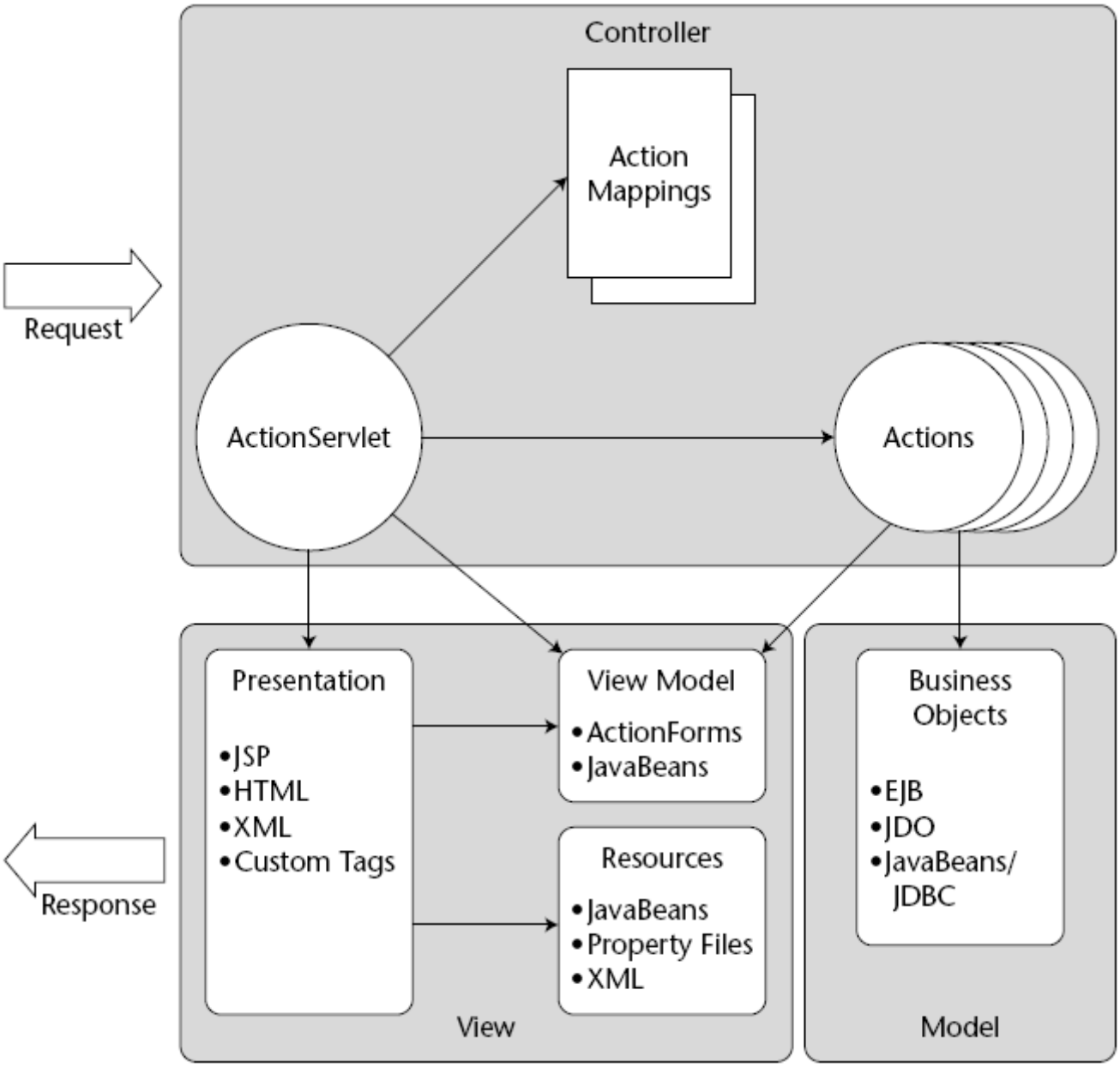

Figura 2: Struts e a arquitetura Model 2

Vamos examinar os objetos fornecidos pelo *framework* para cada um dos componentes *Model*, *View* e *Controller*.

4.1. Controller

4.1.1. ActionServlet

No centro da implementação do *Controller* do *framework Struts* encontra-se a ***ActionServlet.*** Esta serve como uma *servlet Front Controller* e fornece um único ponto de acesso ao restante da aplicação. Contém também a lógica de manipulação da requisição do cliente – através da requisição HTTP do cliente e, baseado na requisição, ou redireciona o usuário diretamente à página WEB ou despacha a requisição ao objeto gerenciador chamado ***Actions*** que será, então, responsável por determinar o resultado da resposta.

A *ActionServlet* conhece todos estes detalhes – qual *Action* chamar para gerenciar determinada requisição, qual componente de *View* deve ser chamado em seguida – lendo esta informação de um arquivo de configuração XML, geralmente nomeado *struts-config.xml*.

Esta servlet é fornecida pelo *framework Struts*. Tudo o que é necessário para incluí-la em nossa aplicação é configurá-la corretamente no descritor de implementação da aplicação.

Abaixo está um trecho de web.xml exibindo como configurar o *ActionServlet* para o uso:

```
...
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>
org.apache.struts.action.ActionServlet
</servlet-class>
<init-param>
<param-name>application</param-name>
<param-value>ApplicationResources</param-value>
</init-param>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
</servlet>
...
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

4.1.2. Action

Como mencionamos antes, algumas requisições do cliente são delegadas às instâncias de objetos da *Action* por nossa classe *servlet Front Controller*. Todos os objetos *Action* definem um método chamado execute() e é este o método que é chamado pela *ActionServlet* para gerenciar a requisição.

O *framework Struts* fornece aos desenvolvedores somente a classe base *Action*. Para incluir objetos *Action* como gerenciadores de requisições em sua aplicação, os desenvolvedores devem estender esta classe base e fornecer uma implementação para o método *execute()*.

Uma atividade comum em aplicações WEB é o início de uma sessão do usuário. Abaixo é mostrada uma implementação da classe *LoginAction* que poderia ser utilizada para gerenciar tais requisições.

```
package actions;

import forms.LoginForm;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginAction extends Action {

  public ActionForward execute(ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception {
    // faz o cast do objeto genérico ActionForm
    // para a implementação específica ActionForm
    // configurada para esta Action
    LoginForm loginForm = (LoginForm) form;

    // Recupera os dados especificados pelo usuário.
    String loginName = loginForm.getLoginName();
    String password = loginForm.getPassword();
```

```
      // Verifica se é o usuário correto
      if (!(loginName.equals("JEDI") && password.equals("JEDI")))
        return mapping.findForward("failure");

      // Armazena o resultado no session scope para uso no restante da aplicação
      HttpSession session = request.getSession();
      session.setAttribute("USER", loginName);

      // o usuário efetuou o login com sucesso. Despacha o usuário para
      // o restante da aplicação.
      return mapping.findForward("success");
    }
  }
```

Observe que a implementação acima emprega o uso de um objeto de negócio, chamado *UserService*, para a autenticação do usuário e não fornece diretamente sua própria implementação no método *execute()*. Instâncias de *Action* devem ser criadas desta maneira – a funcionalidade central deve ser delegada aos objetos de negócio (que podem ser considerados parte da *Model*), não implementada na própria *Action*. As únicas atividades que uma *Action* deve executar são:

- Recuperar as informações fornecidas pelo *JavaBean* de usuário do *ActionForm* associado.
- Traduzir dados do formulário em parâmetros requeridos pelos objetos de negócio que implementam a funcionalidade.
- Recuperar o resultado da operação do objeto de negócio e determinar a *View* seguinte para onde o usuário deve ser encaminhado.
- Opcionalmente, armazenar os resultados dos dados da operação de negócio na sessão ou solicitar objetos que serão utilizados pelo restante da aplicação.

Convém lembrar que ao codificar os exemplos dos objetos *Action*, que o *framework* irá criar uma única cópia do objeto e usá-lo para facilitar todas as requisições. Isto significa que devemos sempre codificar a *Action* para ser *thread-safe* e certificar em utilizar sempre variáveis locais e não variáveis de classe.

Instâncias de *Action* são capazes de instruir a *ActionServlet* para qual componente de *View* delegar a resposta retornando instâncias de objetos ***ActionForward***. *Actions* têm o acesso a estes objetos de *ActionForward* com o uso de um objeto ***ActionMapping***, que encapsula os dados de mapeamentos de caminhos lógicos para cada *Action*. Estes mapeamentos são lidos do arquivo de configuração pela *ActionServlet*, que é responsável por enviar a *ActionMapping* necessária à *Action*. Deste modo, para instruir a *ActionServlet* a passar o controle para um caminho lógico chamado *success*, nossa *Action* executa a seguinte instrução:

```
return mapping.findForward("success");
```

4.1.3. ActionForm

O *framework Struts* fornece uma classe chamada *ActionForm*. Instâncias desta classe são usadas para facilitar a recuperação dos dados dos formulários preenchidos pelo usuário através das instâncias de *Action* que gerenciam os eventos de formulário.

Cada instância de *ActionForm* representa um formulário ou um conjunto de formulários, define as propriedades que correspondem aos elementos do(s) formulário(s) que representam, e as expõem usando métodos *setters* e *getters* publicamente acessíveis. *Actions* que necessitam dos dados dos formulários, simplesmente chamam os métodos *getters* da instância de *ActionForm*.

Struts fornece a definição base da classe; os desenvolvedores têm a responsabilidade de criar suas próprias implementações.

Abaixo é listado o *ActionForm* usado no exemplo acima:

```
import org.apache.struts.action.*;

public class LoginForm extends ActionForm {
  private String loginName;
```

```
    private String password;

    public String getLoginName() {
      return loginName;
    }
    public void setLoginName(String loginName) {
      this.loginName = loginName;
    }
    public String getPassword() {
      return password;
    }
    public void setPassword(String password) {
      this.password = password;
    }
  }
```

Ao codificar *ActionForms*, devemos lembrar de:

- Definir propriedades (com métodos *get* e *set*) para cada elemento representado no formulário.
- NÃO colocar nenhuma lógica de negócio no *ActionForm*. São concebidos meramente para transferir dados entre componentes da *View* e do *Controller* e por isso não são utilizados pela lógica de negócio.
- Opcionalmente, incluir um método de validação dos dados antes que o controle passe para a *Action*.

4.1.4. Arquivo struts-config.xml

Atua como arquivo de configuração para os componentes do *framework Struts*. Podemos definir qual *Action* é chamada para cada requisição, que componente de formulário usar para cada *Action* e o mapeamento de nomes lógicos para caminhos reais, entre outros. Abaixo, temos uma cópia do arquivo *struts-config.xml* usado para o exemplo acima:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
 <form-beans>
  <form-bean name="loginForm" type="login.LoginForm"/>
 </form-beans>
 <action-mappings>
  <action name="loginForm"
   path="/login"
   scope="request"
   type="login.LoginAction">
   <forward name="success" path="/success.jsp"/>
   <forward name="failure" path="/failure.jsp"/>
  </action>
 </action-mappings>
</struts-config>
```

Descreveremos, a seguir, cada um desses elementos.

<!DOCTYPE ...>

Define o arquivo XML como sendo um arquivo de configuração para utilização pelo *framework Struts*. Excluir esta linha, ou mesmo digitá-la errado, resultará em erros quando a aplicação carregar.

<struts-config>

Elemento raiz do arquivo de configuração. Todos os outros elementos são filhos desse elemento.

<form-beans>

Marca o início e o fim das definições das instâncias de uma classe *ActionForms*. Elementos <form-beans> DEVEM ser colocados como filhos deste elemento.

<form-bean>

Define uma instância de *ActionForm* que pode ser utilizada pela aplicação. Tem dois atributos:

- **name** – o nome lógico a ser associado com a classe *ActionForm*
- **type** – o nome completo da classe *ActionForm*.

<action-mappings>

Marca o início e o fim das definições de ações e seus mapeamentos. Todos os elementos <action> DEVEM ser colocados como filhos deste elemento.

<action>

Define uma instância de um objeto *Action* para utilização pela aplicação. A maior parte dos elementos de ação implementa os seguintes atributos:

- **name** – o nome do elemento <form-bean> a ser utilizado nesta ação.
- **path** – o caminho relativo ao contexto a ser utilizado por esta *Action*. Qualquer requisição a este caminho resulta na chamada da *Action* definida.
- **scope** – contexto do escopo onde nossa *ActionForm* pode ser acessada. Isto informa onde a *ActionServlet* deverá armazenar a instância da classe *ActionForm*.
- **type** – o nome de classe *Action*.

<forward>

Ações podem ter nenhum ou muitos elementos de redireção. Este elemento define o mapeamento lógico entre um nome e um caminho na nossa aplicação.

Tem os seguintes atributos:

- **name** – o nome lógico do elemento de redireção que pode ser utilizado pela instância de *Action*.
- **path** – o caminho para o componente de visualização associado a este redirecionador.

4.1.5. Resumo do que se deve fazer para a camada *Controller*:

Executar uma única vez:

- Configure o *ActionServlet* no descritor de instalação da nossa aplicação

Para cada processador de formulário adicionado à aplicação:

- Crie um objeto *ActionForm* que representará todos os dados obtidos do formulário.
- Crie um objeto *Action* que, no seu método de execução, defina como o formulário será processado.
- Crie uma entrada de configuração para o objeto *ActionForm* em *struts-config.xml*, dentro da seção <form-beans>.
- Crie uma entrada de configuração para o objeto *Action* em *struts-config.xml*, dentro da seção <action-mappings>.
- Configure todos os redirecionadores usados pela *Action*, colocando elementos <forward> dentro do corpo de definição <action> da ação.

4.2. Model (Modelo)

O *framework Struts* não fornece explicitamente nenhum componente dentro de *Model*. Quais objetos utilizar como componentes *Model* é deixado a critério do desenvolvedor, apesar de serem normalmente JavaBeans ou, eventualmente, *Entreprise JavaBeans* (EJB).

4.3. View (Visualização)

Struts pode utilizar qualquer tecnologia da camada de apresentação, apesar de, na maioria dos casos, utilizar JSP e/ou HTML. O que o *Struts* fornece para esta camada é um conjunto de bibliotecas de *tags* que permite utilizar as facilidades do *Struts* para popular e validar automaticamente os formulários.

4.3.1. struts-html

Struts fornece uma biblioteca de *tags* chamada *struts-html* que imita muitas das funcionalidades das *tags* HTML padrão, mas traz também novas funcionalidades.

Esta biblioteca de *tags* é mais utilizada na criação de formulários da aplicação. Observaremos o exemplo do formulário descrito a seguir.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>

<html>
 <head>
  <title>Login Page</title>
 </head>
 <body>
  <h1> Login Page </h1>
  <br/>
  <html:form action="/login">
   User Name : <html:text property="loginName"/> <br/>
   Password : <html:password property="password"/> <br/>
   <html:submit/>
  </html:form>
 </body>
</html>
```

Este e o formulário utilizado no exemplo anterior. Observe como os elementos HTML padronizados como <form>, <input type="text">, <input type="password"> e <input type="submit"> foram substituídos pela biblioteca de *tags* struts-html. Iremos, a seguir, descrever estas tags.

<html:form>

A *tag* <html:form> renderiza um formulário em HTML. Cada *tag* do formulário é associada a um mapeamento de ações definidas pelo atributo *action*. O valor deste atributo *action* especifica o caminho do mapeamento da ação.

Revendo o arquivo de configuração utilizado no exemplo anterior,

```
...
<action name=”loginForm”
path=”/login”
scope=”request”
type=”login.LoginAction”>
...
```

Notamos que o valor do atributo *path* coincide com o valor no atributo *action* na *tag* <html:form>. Isso indica que este formulário, quando submetido, será enviado à classe *LoginAction* para seu processamento.

Uma das condições para termos uma *tag* <html:form> válida é que cada um dos campos que ela contenha deve ser definido no *ActionForm* especificado para o mapeamento da ação particular.

Outros atributos possíveis:

- **method** - pode ser tanto POST como GET. Define o método HTTP a ser utilizado ao submeter o formulário

<html:text>

Esta tag renderiza o campo padrão de entrada de texto do HTML. O atributo *property* especifica

qual propriedade no *ActionForm* está ligada a ele. Uma vez que o valor do atributo no exemplo acima é "loginName", este campo de texto é vinculado à propriedade *loginName* do formulário *LoginForm*.

Outros atributos disponíveis para esta tag:

- **size** – define o tamanho do campo de texto a ser mostrado
- **maxlength** – define o comprimento máximo do valor informado pelo usuário

<html:password>

Esta *tag* renderiza um campo de senha padrão do HTML. O atributo *property* especifica a qual propriedade do *ActionForm* está ligada. No nosso exemplo, este campo está ligado à propriedade *password* do *LoginForm*.

Em nossas discussões anteriores sobre as tags <*html:text*> e <*html:password*>, foi mencionado que eles estão ligados a propriedades da instância da classe *ActionForm* associadas ao formulário. Isto significa, em termos práticos, que os valores informados nestes campos serão automaticamente atribuídos às propriedades correspondentes ao objeto *ActionForm*. Além disso, se houvessem valores prévios no objeto *ActionForm* (o formulário já ter sido acessado), os campos do formulário seriam automaticamente preenchidos com valores do objeto *ActionForm*.

Outras tags na biblioteca de *tags struts-html*:

<html:hidden>

Renderiza um campo de formulário HTML oculto.

Exemplo de utilização:

```
<html:hidden property="hiddenField"/>
```

<html:radio>

Renderiza um controle HTML do tipo *radio button*.

Exemplo de utilização:

```
<html:radio property="radioField"/>
```

<html:select>, <html:option>

A *tag* <*html:select*> é utilizada para renderizar uma lista de seleção. As opções para esta lista de seleção são especificadas usando as tags <*html:option*>.

Exemplo de utilização:

```
<html:form action="/sampleFormAction">
 <html:select property="selectField" size="5">
  <html:option value="0 value">0 Label</html:option>
  <html:option value="1 value">1 Label</html:option>
  <html:option value="2 value">2 Label</html:option>
 </html:select>
</html:form>
```

O exemplo acima gera uma lista de seleção com tamanho de 5 itens (size="5"). Observe que o corpo da tag <html:option> atua como um rótulo para este item da lista, enquanto o atributo *value* especifica o valor que será repassado à ação quando ocorrer sua submissão.

<html:checkbox>, <html:textarea>

São utilizados para renderizar campos de caixa de seleção e de área de texto, respectivamente.

4.3.2. Resumo de coisas a fazer para a camada *View*:

Executar somente uma vez:

- Configure as bibliotecas de *tags* para utilização no *deployment descriptor* da aplicação.
- Coloque os arquivos JAR contendo a implementação das bibliotecas de *tags* no diretório

WEB-INF/lib da aplicação.

Para cada formulário a ser criado:

- Adicione a diretiva *taglib* apropriada à página JSP para permitir a utilização da biblioteca de *tags* struts-html.
- No lugar da *tag* <*form*>, utilizar a *tag* <*html:form*>. Especifique no seu atributo *Action* e o caminho da *Action* que processará o formulário.
- No lugar das *tags* de campo HTML (por exemplo a *tag* <*input type="text"*>), utilize as *tags* incluídas na biblioteca de *tags struts-html* que desempenham funcionalidade semelhante (por exemplo a *tag* <*html:text*>).
- Assegure-se que todos os campos de entrada presentes na definição do formulário estejam presentes como propriedades no objeto *ActionForm* associado com esta requisição. NÃO é necessário que todas as propriedades no objeto sejam mostradas como campos, mas requer que todos os campos estejam presentes como propriedades.
- Lembre-se de fechar a *tag* <html:form>.

4.4. Entendendo o Struts como um todo

Para compreender como o *framework Struts* funciona como um todo, vamos tomar como exemplo o cenário visto acima: a entrada de um usuário no sistema.

Antes mesmo de o usuário entrar no sítio, o *ActionServlet* carrega o arquivo de configuração e lê os detalhes. Assim, quando o usuário acessar o formulário de *login*, o *framework* já sabe qual *ActionForm* associada que armazenará seus detalhes e qual *Action* deve ser processada na submissão do formulário.

Quando a página de *login* carregar, as *tags* struts-html que utilizamos tentam renderizar os campos HTML. Se o *ActionForm* para este formulário não existir, a página não será mostrada. Se houver mais campos no formulário do que propriedades no *ActionForm* para lhes dar suporte, a página também não será mostrada. Se o *ActionForm* existir, as *tags* verão se há algum valor armazenado no *ActionForm*. Se houver, os campos do formulário são prenchidos com esses dados. Se não, os campos do formulário serão deixados vazios e o usuário verá um formulário em branco.

Quando um formulário é submetido, os valores nos campos do formulário são automaticamente atribuídos ao objeto *ActionForm* pelo *framework Struts*. Este objeto é, então, passado ao *Action handler* apropriado, juntamente com o objeto *ActionMapping* que traz detalhes dos mapeamentos, como especificados no arquivo de configuração.

O objeto *Action* executa o seu processamento e, então, avisa ao *ActionServlet* para onde ir em seguida, especificando um dos redirecionamentos configurados no mapeamento. O *ActionServlet*, então, redireciona o usuário para aquela página.

5. Exercícios

5.1. Material Educacional

Crie uma aplicação WEB baseada em *Struts*. A aplicação será usada para gerenciar e administrar um endereço WEB que permite o *download* de material educacional. Há dois níveis de usuário para esta aplicação: usuário comum e usuário administrador.

As atividades disponíveis para os administradores são:

a) Gerenciamento do *download* de material – inclusão, alteração e exclusão de *downloads*

b) Gerenciamento dos usuários – inclusão, alteração e exclusão de usuários

c) Relatório de atividade dos usuários – os administradores têm acesso a uma página que mostra uma lista de usuários classificados pela quantidade de *downloads* em ordem decrescente. Ao selecionar um usuário nesta lista, será mostrado um histórico detalhado dos *downloads* deste usuário

As atividades disponíveis para os usuário comuns são:

a) Navegação pelos materiais para *download*

b) Seleção de material para baixar – sua implementação NÃO necessita realmente prover material para *download*. A aplicação deve ter um *link* que, ao ser selecionado, simulará um *download*.

Ambos tipos terão um ponto comum de início para a aplicação: uma página de entrada que receberá o nome do usuário e sua senha pessoal. A aplicação terá, então, que determinar o nível de autorização deste usuário e redirecioná-lo à página apropriada.

Descrição das Telas

Além da página de entrada, que pode ser implementada simplesmente por dois elementos de entrada de texto e um botão de submissão, há diversas outras telas que compõe esta aplicação.

Para os **administradores**, a seqüencia de telas pode ser descrita pelo seguinte mini-roteiro:

- Página de Conteúdo Principal – apresenta três *links*, cada um correspondendo a um grande grupo de atividades permitidas a um administrador
- Página de Gerenciamento de *Downloads* – apresenta ao administrador opções para adicionar, atualizar ou apagar material de *download*
- Página para Adicionar Material para *Download* – contém um formulário com os elementos de entrada necessários para inserir um novo material para *download*
- Página para Atualizar Material para *Download* – contém um formulário com os elementos de entrada necessários para atualizar um material existente para *download*
- Página para Apagar Material para *Download* – contém um formulário com um campo de texto que receberá o identificador do material como entrada para exclusão
- Página de Gerenciamento de Usuário – apresenta ao administrador opções para adicionar, atualizar ou apagar um usuário
- Página para Adicionar Usuário – contém um formulário com os elementos de entrada necessários para adicionar um novo usuário
- Página para Atualizar Usuário – contém um formulário com os elementos de entrada necessários para atualizar um usuário existente
- Página para Apagar Usuário – contém um formulário com um campo de texto que receberá como entrada o identificador do usuário para exclusão
- Página de Atividade dos Usuários – lista de usuários em ordem decrescente de acordo com a quantidade de downloads. Clicar num usuário da lista levará o administrador a uma

página detalhada de *downloads* do usuário

- Página Detalhada de *Downloads* do Usuário – apresenta um histórico detalhado para um usuário em particular
- Página de Erro do Administrador – página apresentada em caso de qualquer erro na aplicação. Mostra a natureza do erro ao administrador

O lado do **usuário** será mais simples, contendo as seguintes páginas:

- Página de Navegação pela Lista de Material de *Download* – a lista contém somente o nome do *download*. Clicar em um determinado nome envia o usuário a uma página mais detalhada
- Página Detalhada do Material para Download – contém detalhes completos sobre o item de *download* selecionado. Esta página contém um *link* que pode ser utilizado para realizar o download do material
- Página de Erro – página apresentada em case de qualquer erro na aplicação. Mostra a natureza do erro do usuário

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.