Manual Básico de Struts

Por Welington B. Souza
javasouza@yahoo.com.br

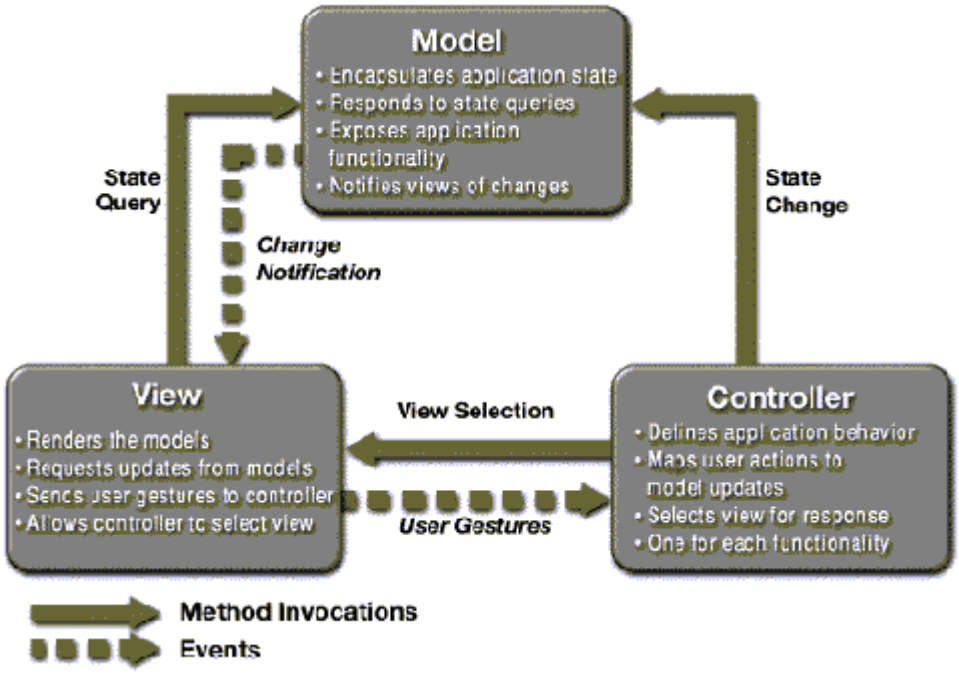
Introdução

O que é Struts?

É uma framework que implementa a arquitetura padrão MVC em Java.

Uma framework é uma extensão de uma linguagem mediante a uma ou mais hierarquias de classes que implementam uma funcionalidade e que (opcionalmente) podem ser estendidas. A framework pode envolver TagLibraries.

A arquitetura padrão MVC - Model-View-Controller (Modelo-Visualização-Controle) é um padrão que define a separação de maneira independente do Model (Modelo) que são os Objetos de Negócio, da View (Visualização) que compreende a interface com o usuário ou outro sistema e o Controller (Controle) que controla o fluxo da aplicação.



Como funciona isto em aplicações web?

O navegador gera uma solicitação que é atendida pelo Controller (um Servlet especializado). E o mesmo se encarrega de analisar a solicitação, seguir a configuração que está programada em seu XML e chamar a Action correspondente passando lhe os parâmetros enviados. A Action instanciará os objetos de negócio para completar a tarefa. De acordo com o resultado que retornar a Action, o Controller irá direcionar a geração de interface para uma ou mais JSPs, as quais poderão consultar os objetos do Modelo a fim de realizar a sua tarefa.

Pra que serve?

Evidentemente, como toda framework tenta, simplifica notavelmente a implementação de uma arquitetura de acordo com o padrão MVC. A mesma separa muito bem o fluxo da aplicação (Controller), do modelo de objetos de negócio (Model) e da geração da interface do usuário (View).

O controlador (Controller) já se encontra implementado por Struts, embora caso seja necessário podemos herdar e ampliar ou modificar, e o fluxo da aplicação podemos programar em um

arquivo XML as ações que serão executadas em cima do modelo de objetos de negócio, serão implementadas baseando se em classes pré-definidas pela framework e seguindo o padrão do modelo MVC.

E a geração da interface é feita através de um conjunto de Tags pré-definidas pela Struts cujo objetivo é evitar o uso de Scriptlets (os trechos de código Java entre "<%" e "%>"), o qual gera vantagens na manutenção e performance (pooling de Tags, caching, etc).

Logisticamente, separa claramente o desenvolvimento de interface do usuário (View) do fluxo da aplicação (Controller) e lógica de negócio (Model) permitindo desenvolver ambas em paralelo ou com pessoal especializado.

Também é evidente que possibilita a reutilização, suporte de múltiplas interfaces de usuário (Html, sHtml, Wml, Desktop applications, etc) e de múltiplos idiomas, localidades, etc.

Licença?

A Struts está disponível sobre a licença "free-to-use-license" da Apache Software Foundation (veja http://www.apache.org/LICENSE-1.1).

Para que serve este manual básico?

Para simplificar o "first touch" e para explicar Struts do ponto de vista do desenvolvimento de aplicações Web.

Onde encontrar mais informações?

http://jakarta.apache.org/struts
http://jakarta.apache.org/struts/userGuide
http://jakarta.apache.org/struts/api/index.html
"http://jguru.com/faq/Struts
http://jakarta.apache.org/struts/#Involved"
buscas em www.google.com com a primeira palavra Struts.

Modelo (Model)

Introdução

O modelo compreende todos os objetos de negócio onde se implementa a lógica de negócio (o "how it's done") e onde se deve suportar todos os requisitos e funcionalidades do sistema sem juntar com as partes correspondentes do fluxo da aplicação (o "what to do") que correspondem ao Controle (Controller).

Action Bean

Geralmente, os ActionBeans sempre realizam as seguintes ações:

Obter os valores necessários do Action Form, JavaBean, request, session ou de outro local.
Chamar os objetos de negócio do modelo.
Analisar os resultados, e de acordo com estes resultados, retornar o ActionForward correspondente.

Vamos ao exemplo de um Action Bean:

```
public final class LogonAction extends Action {
        public ActionForward perform(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response
```

```
        ) throws IOException, ServletException {

                // Obtendo os atributos
                Locale locale = getLocale(request);
                MessageResources messages = getResources();
                User user = null;

                // validando os parâmetros
                ActionErrors errors = new ActionErrors();
                String        username        =        ((LogonForm)
form).getUsername();
                String        password        =        ((LogonForm)
form).getPassword();
                try {
                        DataSource           dataSource           =
servlet.findDataSource(null);
                        Connection          myConnection          =
dataSource.getConnection();
                } catch (SQLException e) {
                        errors.add(ActionErrors.GLOBAL_ERROR,
new ActionError("error.database.missing"));
                        saveErrors(request, errors);
                        return                                    (new
ActionForward(mapping.getInput()));
                }
                UserFactory        userFactory        =        new
UserFactory(database);
                user = userFactory.getUser(username, password);
                if (user == null) {
                        errors.add(ActionErrors.GLOBAL_ERROR,
                                new
ActionError("error.password.mismatch"));
                        saveErrors(request, errors);
                        return                                    (new
ActionForward(mapping.getInput()));
                }

                // Guardando o usuário na sessão
                HttpSession session = request.getSession();
                session.setAttribute(Constants.USER_KEY, user);

                // Eliminando o form bean obsoleto
                if (mapping.getAttribute() != null) {
                        if ("request".equals(mapping.getScope())) {

        request.removeAttribute(mapping.getAttribute());
                        }
                }
                else {

        session.removeAttribute(mapping.getAttribute());
                }

                // Passando o controle para página seguinte
                return (mapping.findForward("success"));
        }
```

```
}
```

System State Beans

Os System State Beans são o conjunto de objetos de negócio que reapresentam o estado atual do sistema, por exemplo: o carrinho de compra que o usuário vai modificando ao longo de sua interação com a aplicação. Estes objetos de negócio serão tipicamente JavaBeans ou EJBs que terão suas referências guardadas na sessão do usuário, que serão modificadas no Action e serão consultadas nos JSPs para montar a camada de apresentação.

Esta classe de objetos não deverá ter nenhum conhecimento da Visualização (View)

Business Logic Beans

Os objetos de negócio implementam a lógica de negócio, é como fazer as coisas e sua própria persistência.

Estes objetos não devem ter nenhum conhecimento da View e do Controller de forma que deverão ser perfeitamente reutilizáveis para implementar o suporte às mais variadas interfaces e abertas o suficiente para serem incorporadas em novas aplicações.

Acessando o Banco de Dados

Com Struts pode-se definir um datasource para uma aplicação no arquivo struts-config.xml (mais informação na sessão "The Action Mappings Configuration File" do "Struts User Manual").

Este datasouce nos permite obter uma conexão de banco de dados de um Action e passar como parâmetro para o Model. Muito prático.
Este é um exemplo de acesso a uma conexão de um Action:

```
public ActionForward perform(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
{
        javax.sql.DataSource dataSource;
        java.sql.Connection myConnection;
        ...
        try {
                dataSource = servlet.findDataSource(null);
                myConnection = dataSource.getConnection();
                ...
        } catch (SQLException sqle) {
                getServlet().log("Connection.process", sqle);
        } finally {
                ...
                try {
                        myConnection.close();
                } catch (SQLException e) {
                        getServlet().log("Connection.close", e);
                }
        }

}
```

View- **Visualização**

Introdução

A visualização compreende os JSPs (principalmente) e os servlets envolvidos na geração da interface com o usuário ou com outros sistemas. Struts fornece suporte para construir aplicações multi-idiomas, interação com formulários e outras utilidades através de tags personalizadas (TagLibraries).

Internacionalização
Devemos seguir os seguintes passos:

Criar um arquivo texto (ex: MyApplication.properties) no diretório onde se encontram as classes da aplicação (ex: d:\com\empresa\application). Este arquivo deve conter as chaves e valores no seguinte formato: chave.subchave=texto que pertence ao idioma principal. Exemplo:

```
...
application.title=Demo de Aplicação com a Strutus  Framework
index.header=Bem vindo ao Demo de Aplicação com a Strutus Framework
...
```

Para cada idioma alternativo, deverá ser adicionado um novo arquivo no formato ?AppName_xx.properties?, onde xx é o código ISO do idioma. (MyApplication_en.properties).

No arquivo strutus-config.xml deve se configurar os parâmetros tag / servlet / init-param / param-name da aplicação e colocar o param-value na localização do arquivo com o idioma desejado

```
<servlet>
    <servlet-name>action</servlet-name>
    ...
    <init-param>
      <param-name>application</param-name>
      <param-value>com.empresa.aplicacao.MinhaAplicacao</param-
value>
    </init-param>
    ...
```

No arquivo web.xml devemos incluir:

```
<web-app>
  ...
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld
    <taglib-location>/WEB-INF/struts-bean.tld
  </taglib>
  ...
</web-app>
```

Nos JSPs onde utilizaremos a internacionalização deveremos incluir:

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

Para declarar que utilizaremos a TagLibrary struts-bean com o prefixo bean e definida no arquivo /WEB-INF/struts-bean.tld. E finalmente, utilizaremos a tag <bean:message key="chave.subchave"/> onde a chave e a subchave correspondem ao texto que será inserido de acordo com o idioma do usuário. Exemplo:

```
...
<TITLE><bean:message key="application.title"/></TITLE>
...
```

Por default Struts acessa o idioma principal da aplicação. Devemos utilizar a tag <html:html locale="true"> substituindo a tag <html>, assim, deveremos substituir também a tag <html> por </html:html> Deste modo, será usado preferencialmente o idioma principal que se encontra no header ?Accept-Language? enviado pelo navegador. Quando o usuário enviar uma requisição para escolher um novo idioma, baseado em uma lista de idiomas suportados, informados previamente, adicionaremos o trecho abaixo:

```
session.setAttribute(Action.LOCALE_KEY,
new Java.util.Locale(country, language));
```

onde country e language será a string do país e que será feita a tradução. Para mais informações a respeito do assunto, veja a documentação oficial da Sun disponível em http://java.sun.com/j2se/1.3/docs/guide/intl/.

Forms

Uma das tarefas que durante o desenvolvimento de uma aplicação consome muito trabalho (ainda que na realidade não mereça) é a interação com formulários, para se editar e obter nova informação. As validações, o tratamento de erros, a apresentação, e o mesmo a entrada de dados do form pelo usuário e mensagens de erros, são suportadas pela Struts, o que torna a vida um pouco mais fácil.

A idéia é a seguinte: todo trabalho de validação e geração de mensagens de erros serão implementados nos ActionForm e todo o trabalho de geração de interface no JSP.

Devemos seguir os seguintes passos:

Criar um ActionForm (veja a seção / Action Form Beans)
Criar a página JSP do formulário usando as tags da Struts. Exemplo:

```
...
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
...
<html:html>
...
   <html:form action="/logon" focus="username">
...
     <h1><bean:message key="logon.header"/></h1>
     <html:form action="/logon" focus="username">
     <h5><html:errors/></h5>
     <h3><bean:message key="logon.mainText"/></h3>
     <p>
         <bean:message key="logon.username"/>
         <html:text property="username" size="16" maxlength="16"/>
```

```
     </p>
     <p>
         <bean:message key="logon.password"/>
         <html:password       property="password"       size="16"
maxlength="16"
             redisplay=false/>
     </p>
     <p><html:submit property="submit" value="Submit"/></p>
     <p><html:reset/></p>
...
   </html:form>
...
</html:html>
```

Declarar o ActionForm no arquivo struts-config.xml, adicionando na seção /struts-config/form-beans a tag <form-bean name="formName" type="package.classe"/> e na declaração do Action (ver a seção Controller / Action Beans) adicionar os atributos name="formName", scope="(request ou session)", e input="paginaForm.jsp"

Exemplo:

```
<struts-config>
    ...
    <form-beans>
        ...
        <form-bean  name="logonForm"
                    type="com.empresa.aplicacao.LogonForm"/>
        ...
    </form-beans>
    ...
    <action-mappings>
        ...
        <action path="/logon"
                type="com.empresa.aplicacion.LogonAction"
                name="logonForm"                    scope="request"
input="/logon.jsp">
            ...
        </action>
        ...
    <action-mappings>
    ...
</struts-config>
```

Tags

Basicamente, uma Tag JSP consiste em uma tag no formato: <prefixoTagLib:nomeTag atributo=valor ... > que na compilação do JSP será substituído por uma chamada à classe TagHandler que se encarrega de resolver a sua funcionalidade.

A classe TagHandler estende BodyTagSupport o TagSupport (segundo se for uma tag que tem body), que implementa sua funcionalidade nos métodos doStartTag(), doEndTag(), doInitBody(), doAfterBody(). (apenas os dois últimos herdam (extends) de BodyTagSupport), além dos gets e sets métodos correspondentes a seus atributos. Esta classe define um Tag Library Definition (arquivo xml com extensão TLD onde se define o nome do tag, a classe TagHandler que atende a definição de seus atributos, se tiver body, etc) que por sua vez deve declarar no arquivo web.xml

(na seção <web-app>, devemos adicionar os tags <taglib> <taglib-uri> nomeTagLib </taglib-uri> <taglib-location> </taglib>) e no JSP onde será usado (<%@ taglib uri="nomeTagLib" prefix="prefixoTagLib" %>). Finalmente, o tag no JSP é usado na forma: <prefixo:nomeTag atributo=valor ...>. Para mais informações, veja na página oficial da Sun sobre TagLibraries em: http://java.sun.com/products/jsp/taglibraries.html.

A funcionalidade dos Tags da framework Struts se encontram documentados em:
html: http://jakarta.apache.org/struts/struts-html.html
bean: http://jakarta.apache.org/struts/struts-bean.html"
logic: http://jakarta.apache.org/struts/struts-logic.html
nested: "http://jakarta.apache.org/struts/struts-nested.html

As classes TagHandlers que implementam tais funcionalidades estão documentadas em:

html:
http://jakarta.apache.org/struts/api/org/apache/struts/taglib/html/package-summary.html
bean:
http://jakarta.apache.org/struts/api/org/apache/struts/taglib/bean/package-summary.html
logic:
http://jakarta.apache.org/struts/api/org/apache/struts/taglib/logic/package-summary.html
nested:
http://jakarta.apache.org/struts/api/org/apache/struts/taglib/nested/package-summary.html

---

Controller - Controle

Introdução

O Controller compreende a funcionalidade envolvida quando um usuário gera um estímulo (clique em um link, envio de um formulário, etc) para que se gere uma interface de resposta. Esta camada, chamará os objetos de negócio do Model para que resolvam com a sua própria funcionalidade e a lógica de negócio retornando um resultado que servirá de base para o Controller direcionar para o JSP que deverá gerar a interface resultante.

Struts já tem um servlet que a partir da configuração do arquivo struts-config.xml recebe as solicitações do usuário, chama o ActionBean correspondente, e de acordo com o resultado do ActionBean, executa um JSP.

Segue abaixo os passos a serem seguidos:

Criar uma classe Action que estende de org.apache.action.Action (Veja a seção "Controller Action Bean"). Configurar o arquivo strutus-config.xml incluindo o novo Action Mapping e seus possíveis forwards de saída.

Por exemplo:

```
        ...
        <action-mappings>
                ...
                <action                                 path="/logoff"
type="com.empresa.aplicacao.LogoffAction">
<forward name="success" path="/index.jsp"/>
                </action>
                ...
        </action-mappings>
        ...
</struts-config>
```

Neste caso, quando for solicitado "/logoff" o Controller chamará o LogoffAction e se este objeto retornar um ActionForward com o valor "sucess", O controller então encaminhará a solicitação para executar o arquivo "/index.jsp"

O que acontece se é uma ação associada a um formulário? A resposta é um pouco mais complexa: deve se definir um Form Bean, um Action Mapping com o Form Bean associado e/ou os fowards necessários.
Por exemplo:

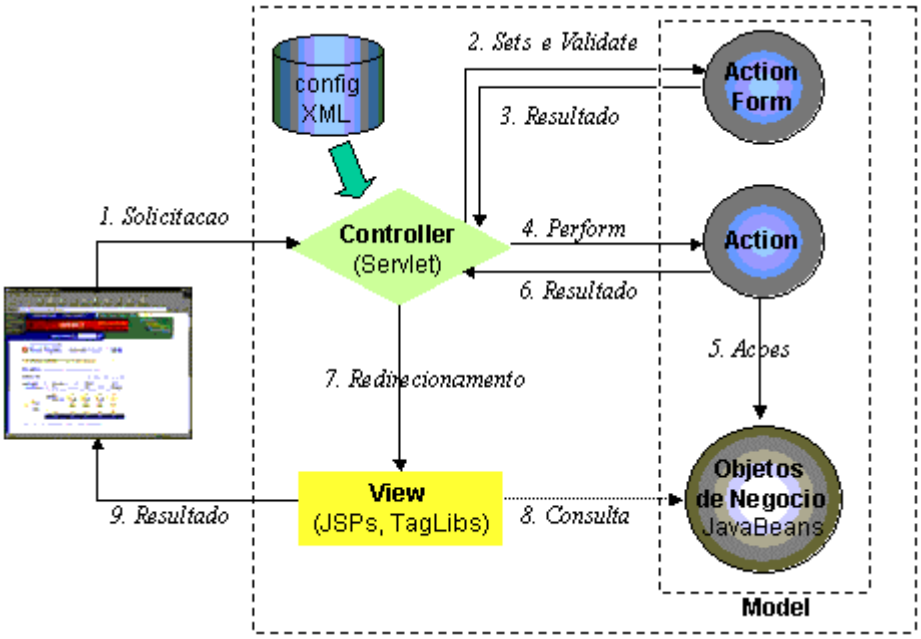
```
<struts-config>
        ...
        <form-beans>
                ...
                <form-bean      name="logonForm"
type="com.empresa.aplicacao.LogonForm"/>
                ...
        </form-beans>
        ...
        <global-forwards>
                ...
                <forward name="success" path="/mainMenu.do"/>
                ...
        </global-forwards>
        ...
        <action-mappings>
                ...
                <action path="/logon"

        type="com.empresa.aplicacao.LogonAction"
                                name="logonForm"
                                scope="request" input="/logon.jsp">
                </action>
                ...
        </action-mappings>
        ...
</struts-config>
```

Neste caso está definido um global-forward que, como seu nome indica, vem a ser um forward que se aplica a todos os action-mappings (exceto se houver o a redefinição para algum em particular).

Incluir os links (preferencialmente utilizando <html:link>) o form (necessariamente utilizando <html:form>) necessários nos JSPs correspondentes.

ActionForm Beans

Os ActionForm Beans são classes que estendem ActionForm e implementam os métodos get e set para cada imput do form de uma página, e seus métodos validate e reset.



Quando um usuário completa um formulário e enviá-lo, o Controller busca no escopo especificado o ActionForm Bean correspondente (de acordo com as configurações no arquivo struts-config.xml), e se não encontrar, o cria. Assim, realiza um set para cada input do form e finalmente chama o método "validate". Se o método validate retornar um ou mais erros, o Controller chama o JSP do formulário para que o usuário (com os valores preenchidos inicialmente pelo usuário) com a(s) mensagem(s) de erro correspondente(s). Se tudo correr bem, chamará o método "perform" do Action (também configurado no arquivo struts-config.xml) passando o ActionForm Bean como parâmetro para que seja utilizado para obter os valores dos dados.

Apesar do ActionForm ter características que correspondem ao Model, o ActionForm pertence ao View. Justamente um destes pontos comuns é á validação de dados e a fim de evitar a duplicação de funcionalidade,

Se um ActionForm deve realizar controles de validação que se é implementado em um objeto de negócio então devemos utilizar uma instância deste objeto para efetuar as validações.

```
public final class ClienteForm extends ActionForm {

        private String nome = null;

        ActionErrors errors = null;
        Cliente cliente = null;
        ...
        public ClienteForm() {
                ...
                // Criar os objetos ...
                cliente = new Cliente();
                errors = new ActionErrors;
                ...
        }

        public String getNome() {
                return (this.nome);
```

```
	}

	public void setNome(String nome) {
		try {
			cliente.setNome(nome);
		} catch (Exception e) {
			errors.add("nome",                    new
ActionError("error.nome"));
		}
		this.nome = nome;
	}

	public void reset(ActionMapping mapping,
HttpServletRequest request)
{
		this.nome = null;
	}

	public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request)
{
		...
		return errors;
	}
}
```

Quando escrever um ActionForm devemos ter em mente os seguintes princípios:
Não deve ter nada que corresponda com a lógica de negócio.
Não deveria ter mais implementações de getters e setters (obrigatoriamente um para para cada input do form; se o input se chama nome então devemos ter getNome() e setNome(String nome)), e os métodos reset e validate.
Deve ser um Firewall entre o usuário e o Action que detém todos os tipos de erros de campos sem preenchimento ou inconsistência.
Se o formulário se desenrola em várias páginas (por exemplo interfaces do tipo "Wizard/Assistentes") o ActionForm e o Action deverão ser os mesmos, o que permitirá, entre outras coisas, que os inputs podem-se estender em páginas diferentes sem trocar o ActionForm e o Action.

Como colocar a Struts em funcionamento ?

Como toda framework, Struts consiste em uma biblioteca de classes e uma série de configurações para a sua instalação bastante similar com os servidores de aplicações que fornecem (ou podem agregar) suporte para:
Java Development Kit (version 1.2 ou posterior)
Servlet Container (version 2.2 ou posterior)
JavaServer Pages (JSP) (version 1.1 ou posterior)
XML Parser compatible with Java API for XML Parsing (JAXP) specification, 1.1 ou posterior
JDBC 2.0 Optional Package Classes

A instalação consiste em:
Baixar os binários em: http://jakarta.apache.org/struts/index.html#Acquiring e descompactar;
Copiar lib/commons-*.jar (classes comuns da Jakarta), /lib/struts.jar (JAR das classes da Struts) e /lib/struts*.tld (Tag Library Descriptors das Tags da Struts) no diretório WEB-INF/lib da web application;
Modificar o arquivo WEB-INF/web.xml da web application e adicionar o elemento "<servlet>" que define o Servlet do Controller e um "<servlet-mapping>" que atende as

solicitações. Por exemplo:

```
<web-app>
        ...
        <servlet>
                <servlet-name>action</servlet-name>
                <servlet-class>
org.apache.struts.action.ActionServlet
</servlet-class>
                <init-param>
                        <param-name>application</param-name>
                        <param-value>
com.empresa.aplicacao.ApplicationResources
</param-value>
                </init-param>
                <init-param>
                        <param-name>config</param-name>
                        <param-value>/WEB-INF/struts-
config.xml</param-value>
                </init-param>
                <init-param>
                        <param-name>debug</param-name>
                        <param-value>2</param-value>
                </init-param>
                <init-param>
                        <param-name>detail</param-name>
                        <param-value>2</param-value>
                </init-param>
                <init-param>
                        <param-name>validate</param-name>
                        <param-value>true</param-value>
                </init-param>
                <load-on-startup>2</load-on-startup>
        </servlet>
        ...
        <servlet-mapping>
                <servlet-name>action</servlet-name>
                <url-pattern>*.do</url-pattern>
        </servlet-mapping>
        ...
</web-app>
```

Observação: com.empresa.aplicacao.ApplicationResources é a localização do arquivo ".properties" do idioma principal da aplicação.
Modificar o arquivo WEB-INF/web.xml da aplicação para incluir a definição das TagLibraries. Por exemplo:

```
<web-app>
        ...
        <taglib>
                <taglib-uri>strutsBean</taglib-uri>
                <taglib-location>/WEB-INF/struts-bean.tld</taglib-
location>
        </taglib>

        <taglib>
                <taglib-uri>strutsHtml</taglib-uri>
```

```
                <taglib-location>/WEB-INF/struts-html.tld</taglib-
location>
        </taglib>

        <taglib>
                <taglib-uri>strutsLogic </taglib-uri>
                <taglib-location>/WEB-INF/struts-
logic.tld</taglib-location>
        </taglib>

        <display-name></display-name>
        <description></description>
        ...
</web-app>
```

Adicionar a definição das TagLibraries nos JSPs que as utilizam. Exemplo:

```
<%@       taglib       uri="strutsBean"       prefix="bean"       %>

<%@       taglib       uri="strutsHtml"       prefix="html"       %>

<%@      taglib      uri="strutsLogic"      prefix="logic"      %>
```

Criar um arquivo struts-config.xml definindo as configurações do Controller de acordo com os passos anteriores.

Atualmente está começando a surgir aplicações que administram a configuração da Struts de forma visual (Camino, StrutsConsole, etc). É muito recomendável familiarizar-se primeiro com a configuração manual antes de utilizar estas ferramentas, pois caso ocorra(m) problema(s) imprevisto(s) pela ferramenta o(s) mesmo(s) pode(m) ser corrigido(s) manualmente.