PROJETO FRAMEWORK - CELEPAR

VALIDAÇÃO DE DADOS COM O STRUTS VALIDATOR

Março – 2006

Sumário de Informações do Documento

Tipo do Documento: Definição

Título do Documento: Validação de dados com o Struts Validator

Estado do Documento: EB (Elaboração)

Responsáveis: Grupo Framework

Palavras-Chaves: validação, java, struts, validator

Resumo: Padronizar a forma de validação no lado servidor

Número de páginas: 14

Software utilizados:

Versão	Data	Mudanças
1.0	10/02/05	Criação do documento
1.1	30/09/05	Propósito e exemplificação da utilização do método validaForm() da Classe BaseDispatchAction.
1,2	02/03/06	Adicionado tópico de considerações finais – revisado por Elisabeth Hoffmann

SUMÁRIO

1 VALIDAÇÃO NA CAMADA DE CONTROLE

1.1 Introdução

Para realizar a validação de dados no lado servidor foi padronizada a utilização do mecanismo de validação (Validator) disponível no framework Struts.

O Struts Validator visa centralizar o processo de validação de dados, reduzir a redundância de código e desacoplar o código de validação, além de permitir que a validação de formulários seja realizada tanto do lado servidor (no container web), quanto no lado do cliente (no navegador).

1.2 Configuração do Struts Validator

O Struts Validator é um plug-in do Struts e deve ser registrado no arquivo struts-config.xml:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
      value="/WEB-INF/validation.xml", /WEB-INF/validation-rules.xml />
   <set-property property="stopOnFirstError" value="false"/>
</plug-in>
```

1.3 Mecanismos de validação

A validação de formulários no Struts era feita no método validate dos ActionForms, chamado quando o formulário é submetido. Utilizando o Struts Validator, as validações são especificadas em arquivos de configuração e classes externas. Em vez de herdar da Classe ActionForm, herda-se da subclasse ValidatorForm. Esta classe contém uma implementação padrão do método validate, que lê os arquivos de configuração do Struts Validator ativando as validações especificadas. A classe a seguir mostra um exemplo de formulário validado (example.form.CadastroUsuarioForm). Repare que nenhum código de validação é inserido na classe:

```
package example.form;
public class CadastroUsuarioForm extends ValidatorForm {
      private String nome;
      private String endereco;
      private String telefone;
      private String login;
      private String senha;

      public String getNome() {
            return nome;
      }
      public void setNome(String nome) {
            this.nome = nome;
      }

      //Os demais métodos get/set para os atributos
}
```

O formulário deve ainda ser registrado no arquivo struts-config.xml:

```
<struts-config>
   ...
   <form-beans>
      <form-bean name="cadastroUsuarioForm"
         type="example.form.CadastroUsuarioForm" />
      </form-beans>
   </form-beans>
   ...
</struts-config>
```

1.4 Validadores do Struts

O Struts 1.2.4 utilizado aqui, possui um conjunto de validadores pré definidos, para validações de tipos primitivos, datas, intervalos e expressões regulares, entre outros. Veja a listagem:

Validator	Descrição
required	Determina que o campo é de preenchimento obrigatório
validwhen	Determina o preenchimento obrigatório de um campo, se determinado campo estiver preenchido
minlength, maxlength	especifica o tamanho mínimo/máximo do conteúdo de um

Validator	Descrição
	campo
mask	Define um “máscara” (expressão regular) para formatação do campo. É utilizado o pacote Jakarta Regexp para a validação
byte, short, integer, long, float, double	verifica se o valor de um campo pode ser convertido para o tipo Byte, Short, Integer, Long, Float, Double de acordo com o validador
date	Verifica se o valor do campo pode ser convertido para um Date. este validador utiliza a classe java.text.SimpleDateFormat para fazer a conversão
intRange, floatRange, doubleRange	Verifica se o valor do campo é um int, float ou double e se o mesmo está dentro de uma faixa de valores especificada
creditCard	Valida números de cartão de crédito
email	Valida endereços de e-mail
url	Valida endereços url

Todos os validadores padrões do Struts são especificados no arquivo validation-rules.xml. A aplicação struts-blank.war incluída na distribuição do Struts já vem configurada para utilização do plug-in de validação incluindo o validation-rules.xml com as configurações dos validadores padrões, o qual pode ser utilizado para uma nova aplicação.

O código a seguir, retirado do arquivo validation-rules.xml, mostra a declaração do validador intRange, que verifica se o valor de um campo está entre dois limites inteiros.

```
<form-validation>
...
      <validator name="intRange"
            classname="org.apache.struts.validator.FieldChecks"
               method="validateIntRange"
         methodParams="java.lang.Object,
                       org.apache.commons.validator.ValidatorAction,
                       org.apache.commons.validator.Field,
                       org.apache.struts.action.ActionMessages,
                       javax.servlet.http.HttpServletRequest"
              depends="integer"
                  msg="errors.range"/>
...
</form-validation>
```

Como mostrado, os validadores são definidos usando o elemento <validator>, que define atributos especificando cada detalhe da validação. No exemplo intRange é definido pelo método validateIntRange da classe org.apache.struts.validator. FieldChecks.

A definição de um validador inclui ainda outras informações importantes:

- O atributo *depends* determinada que o validador somente será executado após todos os outros validadores listados como dependentes. O intRange, por exemplo, é executado somente quando um campo contiver um valor inteiro, assim é definido como dependente do validador predefinido integer (não mostrado no código acima);

- *msg* define o identificador da mensagem exibida se a validação gerar um erro; a mensagem dever ser especificada em um dos arquivos de recursos do Struts. O código a seguir mostra o arquivo ApplicationResources.properties usado neste exemplo com mensagens traduzidas do arquivo original retirado do struts-blank.war.

```
errors.required={0} obrigatório.
errors.invalid={0} inválido.

errors.byte={0} precisa ser um byte.
errors.short={0} precisa ser um inteiro (short).
errors.integer={0} precisa ser um inteiro.
errors.long={0} precisa ser um inteiro longo.
errors.float={0} precisa ser um float.
errors.double={0} precisa ser um double.

errors.date={0} não é uma data.
errors.range={0} não está no intervalo {1} a {2}.
errors.email={0} não é um endereço de e-mail válido.
```

1.5 Validando um formulário

Após feita a configuração dos validadores e criação de um formulário é necessário definir a validação aplicada a cada campo. Para isso é necessário criar um elemento <formset> no arquivo validation.xml, definindo assim grupos de validações para formulários distintos. Os formulários são definidos em elementos <form>, contendo elementos <field> para cada campo a ser validado. A seguir a configuração para a validação do formulário criado anteriormente como exemplo:

```
<form-validation>
      <formset>
            <form name="cadastroUsuarioForm">
                  <field property="nome" depends="required">
                        <msg name="required"
                              key="cadastroUsuarioForm.erro.nome" />
                        <arg0 key="cadastroUsuarioForm.nome" />
                  </field>
                  <field property="email" depends="required,email">
                        <arg0 key="cadastroUsuarioForm.email" />
                  </field>
                  <field property="endereco" depends="mask">
                        <arg0 key="cadastroUsuarioForm.endereco" />
                        <var>
                              <var-name>mask</var-name>
                              <var-value>^[a-zA-Z ]*$</var-value>
                        </var>
                  </field>
                  <field property="telefone" depends="required,mask">
                        <arg0 key="cadastroUsuarioForm.telefone" />
                        <var>
                              <var-name>mask</var-name>
                              <var-value>^\d{3,4}-\d{4}$</var-value>
                        </var>
                  </field>
                  <field property="login" depends="required,mask">
                        <arg0 key="cadastroUsuarioForm.login" />
                        <var>
                              <var-name>mask</var-name>
                              <var-value>^[a-zA-Z]*$</var-value>
                        </var>
                  </field>
                   <field property="senha"
                        depends="required">
                        <arg0 key="cadastroUsuarioForm.senha" />
                  </field>
            </form>
      </formset>
</form-validation>
```

A passagem de argumentos para os métodos de validação é feita usando elementos <arg0>, <arg1>, <arg2> e <arg3>, que tanto podem referenciar variáveis definidas no ApplicationResources.properties, como especificar valores explicitamente. Além desses atributos, <field> pode conter elementos <var>, declarando variáveis que também podem ser utilizados pelo validador, como no validador mask mostrado no código anterior, onde o elemento <var> define a máscara utilizada pelo validador.

Um campo pode também especificar uma mensagem de validação personalizada para usar no lugar da default do validador. No código anterior isso foi feito para o campo nome onde a chave para a mensagem , definida em um elemento <msg> , onde o atributo name referencia o validador, e o atributo key passou a ser cadastroUsuarioForm.erro.nome, o qual precisa ser incluído no arquivo

ApplicationResources.properties. De acordo com as definições de validações feita é necessário adicionar o seguinte conteúdo no arquivo ApplicationResources.properties.

```
cadastroUsuarioForm.nome=Nome do usuário
cadastroUsuarioForm.endereco=Endereço
cadastroUsuarioForm.telefone=Telefone
cadastroUsuarioForm.login=Login no sistema
cadastroUsuarioForm.senha=Senha de acesso
cadastroUsuarioForm.confirmSenha=Confirmação da Senha de acesso
cadastroUsuarioForm.email=E-mail do usuário
cadastroUsuarioForm.erro.nome=O campo Nome do usuário é de preenchimento obrigatório.
```

Já na configuração da Action se o atributo validate estiver configurado com o valor “true” o Struts em qualquer chamada que seja feita à Action em questão invocará a validação do Form antes de passar o controle para a Action e caso a validação apresente algum erro retorna para a página definida na configuração do atributo “input”, o qual passa a ser obrigatório quando validate=”true” como segue:

```
<action path="/cadastroUsuario"
            type="example.action.CadastroUsuarioAction"
            name="cadastroUsuarioForm"
            scope="request"
            validate="true"
            parameter="action"
            input="editUsuarioDef">
        <forward name="edit" path="editUsuarioDef" />
        <forward name="list"   path="listUsuarioDef" />
</action>
```

Tal tipo de configuração pode apresentar situações inesperadas já que as classes Actions a serem criadas acabam extendendo da classe BaseDispatchAction onde cada método da classe passa a ser responsável por uma operação (passo) do proceso e a cada invocação feita a qualquer dos métodos da Action o Struts vai tentar validar o Form antes de passar o controle para a classe Action em questão, situação que pode ser indesejada dependendo do caso.

Por exemplo, é feita uma invocação a Action onde o resultado esperado é a apresentação de uma página com um formulário para informação dos dados, mas como o Form em questão não foi ainda preenchido, a validação do mesmo falha e o Struts acaba acusando erro antes mesmo de a

Action poder redirecionar para a página de informação dos dados.

Para resolver tais problemas foi criado um método validaForm() na classe BaseDispatchAction onde a finalidade é permitir à classe Action de decidir em quais de seus métodos validar o Form. A solução baseia-se em alterar a configuração da Action mudando o valor do atributo validate para "false" e removendo o atributo input como segue:

```
<action path="/cadastroUsuario"
            type="example.action.CadastroUsuarioAction"
            name="cadastroUsuarioForm"
            scope="request"
            validate="false"
            parameter="action">
        <forward name="edit" path="editUsuarioDef" />
        <forward name="list"   path="listUsuarioDef" />
</action>
```

Feito isso o Struts não irá mais validar o Form antes de passar o controle para a classe Action, a responsabilidade de validar o Form (através do método validaForm()) passa a ser da Action em cada método onde se espera um form com dados válidos, sendo necessário então codificar como segue no inicio do método a ser invocado na Action:

```
if (!validaForm(mapping, form, request)) {
      return mapping.findForward("edit");
}
```

O método validaForm() então realiza a validação do Form de acordo com as configurações de validação e retorna um boolean como resultado. Em caso de falha (false) resta ao desenvolvedor retornar o controle para alguma das definições de "forward" possíveis para a Action. Caso contrário seguir normalmente a execução do método agora com todos os dados válidos.

1.6 Mostrando o resultado da validação

O Struts Validator apenas armazena as mensagens a serem exibidas quando alguma validação acusar um erro. A exibição destas mensagens não é propósito deste documento, as quais serão exibidas de acordo com o padrão proposto para tratamento de Exceções.

1.7 Estendendo as validações

O validador mask possibilita a criação de diversas validações já que é baseado na verificação de uma expressão regular. O pacote Jakarta RegExp (http://jakarta.apache.org/regexp) é usado para analisar a expressão. Se uma expressão precisar ser usada por mais de um campo, poderá também ser definida como uma constante no arquivo validation.xml. Seguem alguns exemplos de expressões de validação que podem se utilizar de tal recurso:

Expressão	***Descrição***
^[a-zA-Z]*$	String com caracteres de "a" à "z", tanto maiúsculas quanto minúsculas
^[a-zA-Z]*$	String com caracteres de "a" à "z" e "espaço", tanto maiúsculas quanto minúsculas
(^\d{3,4}-\d{4}$)\|(^\d{7,8}$)	Telefone com 7 ou 8 digítos formatado ou não com o separador
^\d{5}-\d{3}$	CEP (5 números + "-" + 3 números)

O framework Struts disponibiliza vários pontos de extensão (Extension-Points), sendo que o Struts Validator é um deles. Para os casos onde os validadores padrões do Struts Validator não forem suficientes a extensão é a solução, onde os desenvolvedores podem criar seus próprios validadores para atuarem em conjunto com o Struts Validator. Por exemplo para validação de CPF, CNPJ, entre outros. Para isso são necessárias duas etapas:

- Criar um método em uma classe Java para lidar com as validações no lado do servidor;
- Atualizar o arquivo validator-rules.xml com um novo elemento para o validador criado.

Para exemplificar a criação de um novo validador imagine a seguinte situação. Em uma tela para alteração de senha é comum fazer com que o usuário forneça a nova senha duas vezes, para garantir que tenha digitado corretamente. Outros campos de um formulário também poderiam ser correlacionados de alguma maneira. Então o validador a ser criado para resolver a situação anterior além de mostrar como criar um novo validador servirá para exemplificar o correlacionamento de campos e validadores.

Qualquer classe pode ser usada para armazenar o método validate. A assinatura básica do

método é a seguinte:

```
public static boolean validateXxx(
    Object bean,
    ValidatorAction va,
    Field field,
    ActionMessages errors,
    HttpServletRequest request);
```

Onde Xxx fica a critério do desenvolvedor para nomear o seu método validate.

As propriedades do método validate.

Propriedade	**Descrição**
Object bean	O componente no qual a validação está sendo executada
Validator va	É um JavaBean que representa o elemento validator para esse validador a partir de validation-rules.xml.
Field field	Esse JavaBean representa o elemento para o campo que vai ser validado.
Actionmessages errors	Um objeto ActionMessages que recerá qualquer erro de validação que possa ocorrer
HttpServletRequest request	O objeto de solicitação atual sob esta operação

A partir das informações anteriores o validador utilizado para comparar o valor de dois campos, no caso as senhas, teria o seguinte código:

```
package example.validator;

//imports necessários

public class StrutsValidator {

      public static boolean validateIdentico(Object bean,
                                    ValidatorAction va, Field field,
                                    ActionMessages errors,
                                    HttpServletRequest request) {

            String value = ValidatorUtils.getValueAsString(bean,
                  field.getProperty());
            String sProperty2 = field.getVarValue("secondProperty");
            String value2 = ValidatorUtils.getValueAsString(bean,
                  sProperty2);

            if (!GenericValidator.isBlankOrNull(value)) {
                  try {
                        if (!value.equals(value2)) {
                              errors.add(field.getKey(),
                                    Resources.getActionMessage(request,
                                          va,
                                          field));

                              return false;
                        }
                  } catch (Exception e) {
                        errors.add(field.getKey(),
                              Resources.getActionMessage(request,va,
                                    field));
                        return false;
                  }
            }
            return true;
      }
}
```

E a definição do novo validador no validation-rules.xml é a seguinte:

```
<validator name="identico"
           classname="gov.pr.celepar.validator.StrutsValidator"
              method="validateIdentico"
       methodParams="java.lang.Object,
                     org.apache.commons.validator.ValidatorAction,
                     org.apache.commons.validator.Field,
                     org.apache.struts.action.ActionMessages,
                     javax.servlet.http.HttpServletRequest"
             depends=""
                 msg="errors.identico"/>
```

Feito isso o novo validador chamado “idêntico” está pronto para ser utilizado.

Então no formulário deste exemplo, cadastroUsuarioForm, precisa ser adicionada a propriedade senha2, do tipo String, e seus métodos get/set para armazenar o valor do senha2 digitado em tela pelo usuário.

É preciso ainda redefinir a validação do campo senha no arquivo validation.xml, como segue:

```
<field property="senha"
      depends="required,identico">
      <arg0 key="cadastroUsuarioForm.senha" />
      <arg1 key="cadastroUsuarioForm.confirmSenha" />
      <var>
            <var-name>secondProperty</var-name>
            <var-value>senha2</var-value>
      </var>
</field>
```

Repare que na nova configuração o validador passou a receber dois argumentos (arg0 e arg1), que serão utilizados para a correta exibição da mensagem caso a validação não tenha sucesso.

Na configuração do novo validador “identico”, a mensagem padrão foi definida como errors.identico, então é necessário ainda adicionar a nova mensagem no arquivo ApplicationResources.properties.

```
errors.identico = {0} e {1} não são iguais.
```

1.8 Considerações finais

Para evitar a criação de muitas validações que não serão utilizadas, a validação do javascript do cliente não deve existir. Caso a validação seja requerida pelo analista, a validação do javascript do cliente poderá existir, como por exemplo a validação de um CEP no evento de saída do campo.