


Jakarta EE Web Total

Conversores e Validadores




Tópicos Abordados



- Processos de conversão e validação
- Conversores
 - Conversão automática
 - Mensagens de erro (padrão e customizadas)
 - Conversores customizados
- Validadores
 - Tags de validação
 - Mensagens de erro (padrão e customizadas)
 - Validadores customizados
 - Validando com métodos do bean
 - Ignorando a validação

Processos de Conversão e Validação



- Os processos de conversão e validação são necessários para garantir que os dados que chegam ao bean sejam consistentes

Título:

Data de publicação:

Número de páginas:

Preço:

Cadastrar

Strings

Diferentes tipos

```
public class LivroBean implements Serializable {  
    private String titulo;  
    private Date dataPublicacao;  
    private Integer numPaginas;  
    private Double preco;  
    ...  
}
```

Conversão Automática



- Algumas conversões são feitas de forma automática pelo JSF
 - Tipos de dados primitivos e classes wrappers correspondentes
 - Datas
 - Enums
- É possível especificar a formatação dos dados durante o processo de conversão



Formatação de Dados



- Datas e números podem ser formatados, a fim de indicar como a conversão será realizada

```
<h:inputText value="#{livro.dataPublicacao}">  
  <f:convertDateTime pattern="MM/yyyy" />  
</h:inputText>
```

```
<h:inputText value="#{livro.preco}">  
  <f:convertNumber minFractionDigits="2"  
    maxFractionDigits="2" groupingUsed="false" />  
</h:inputText>
```



Exibindo Mensagens de Erro



- Quando um erro de conversão ou validação ocorre, o JSF exibe a mesma tela novamente, com os dados do formulário já preenchidos
- A tag `h:message` é utilizada para exibir o erro ocorrido

```
<h:inputText id="numPaginas" value="#{livro.numPaginas}" />  
<h:message for="numPaginas" />
```

Referencia um componente

Título:	xxxxx	
Data de publicação:	xxxxx	j_idt6.dataPublicacao: não foi possível reconhecer 'xxxx' como uma data. Exemplo: 10/2011
Número de páginas:	xxxxx	j_idt6.numPaginas: 'xxxxx' deve ser um número entre -2147483648 e 2147483647 Exemplo: 9346
Preço:	xxxxx	j_idt6.preco: 'xxxxx' não é um número. Exemplo: 99



Exibindo Mensagens de Erro



- A tag `h:messages` pode ser utilizada para exibir todas as mensagens de erro
- Mais utilizada durante o desenvolvimento

• `j_idt6.dataPublicacao`: não foi possível reconhecer 'xxxx' como uma data.
• `j_idt6.numPaginas`: 'xxxx' deve ser um número formado por um ou mais dígitos.
• `j_idt6.preco`: 'xxxx' não é um número.

Título:	<input type="text" value="xxxx"/>
Data de publicação:	<input type="text" value="xxxx"/>
Número de páginas:	<input type="text" value="xxxx"/>
Preço:	<input type="text" value="xxxx"/>



Mensagens Resumidas e Detalhadas



- Uma mensagem de erro a ser exibida tem duas partes
 - *Summary*: mensagem resumida
 - *Detail*: mensagem detalhada
- É possível controlar qual parte exibir

```
<h:message for="comp" showSummary="true" showDetail="false" />
```

- Para `h:message`, o padrão é exibir o detail
- Para `h:messages`, o padrão é exibir o summary



Customizando Mensagens de Erro



- É necessário criar um arquivo de recursos na aplicação, que vai sobrescrever as mensagens do arquivo de recursos padrão do JSF

Deve ficar no classpath da aplicação

O arquivo deve ter a extensão `.properties`

MyMessages.properties

```
javax.faces.converter.IntegerConverter.INTEGER=0 dado ''{0}'' não é um número válido  
javax.faces.converter.IntegerConverter.INTEGER_detail=0 dado ''{0}'' não é um número válido
```

Devem ser definidas as mensagens *summary* e *detail*



Customizando Mensagens de Erro



- O próximo passo é configurar a aplicação para ler as informações deste arquivo
- Isto é feito referenciando o arquivo no *faces-config.xml*

```
<faces-config>
  <application>
    <message-bundle>app.MyMessages</message-bundle>
  </application>
</faces-config>
```

- O arquivo da aplicação será consultado primeiro, antes do arquivo de recursos do JSF



Customizando Mensagens de Erro



- Outra opção é fornecer a mensagem de erro de conversão diretamente na tag do componente

```
<h:inputText value="#{livro.preco}"
  converterMessage="Erro de conversão" />
```



Conversores Customizados



- Algumas vezes os conversores padrão do JSF não atendem as necessidades da aplicação
- É possível criar conversores customizados

```
@FacesConverter(forClass = Estado.class)
public class EstadoConverter implements Converter {

    public Object getAsObject(FacesContext context,
        UIComponent component, String value) {
        //...
    }

    public String getAsString(FacesContext context,
        UIComponent component, Object value) {
        //...
    }
}
```



Conversores Customizados



- No momento da conversão, um conversor customizado pode avisar sobre erros de conversão
- É preciso lançar uma `ConverterException`

```
public Object getAsObject(FacesContext context,
    UIComponent component, String value) {
    //...
    FacesMessage msg = new FacesMessage("Error!");
    throw new ConverterException(msg);
}
```



Validação de Dados



- O JSF possui mecanismos de validação de dados

Tag	O que valida
<code>f:validateRequired</code>	A presença de uma informação
<code>f:validateLength</code>	O tamanho do texto fornecido
<code>f:validateLongRange</code>	Se o número é do tipo <i>long</i> e está dentro de determinado intervalo
<code>f:validateDoubleRange</code>	Se o número é do tipo <i>double</i> e está dentro de determinado intervalo
<code>f:validateRegex</code>	Uma expressão regular



Validação de Dados



- Exemplos de validação

```
<h:inputText value="#{livro.titulo}">
  <f:validateRequired />
  <f:validateLength minimum="5" maximum="30" />
</h:inputText>
```

```
<h:inputText value="#{livro.titulo}" required="true" />
```

```
<h:inputText value="#{livro.numPaginas}">
  <f:validateLongRange minimum="10" maximum="9999" />
</h:inputText>
```



Exibindo Mensagens de Erro



- Os erros de validação são exibidos da mesma forma que os erros de conversão
 - Tags `h:message` e `h:messages`
- É possível mudar as mensagens de erro padrão do JSF através da definição de um arquivo de recursos na aplicação

MyMessages.properties

```
javax.faces.component.UIInput.REQUIRED=Forneça um valor  
javax.faces.validator.LengthValidator.MAXIMUM=O texto tem mais que {0} caracteres  
javax.faces.validator.LengthValidator.MINIMUM=O texto tem menos que {0} caracteres
```

- Também é possível utilizar o atributo `validatorMessage` no componente



Validadores Customizados



- Assim como nos conversores, é possível criar validadores customizados
- O primeiro passo é criar uma classe que implemente `Validator`

```
@FacesValidator("app.validator.Date")  
public class DateValidator implements Validator {  
    public void validate(FacesContext context, UIComponent  
        component, Object value) throws ValidatorException {  
        //...  
    }  
}
```



Validadores Customizados



- Depois de criado, o validador pode ser referenciado por componentes JSF

```
<h:inputText value="#{livro.dataPublicacao}">  
    <f:validator validatorId="app.validator.Date" />  
</h:inputText>
```



Validando com Métodos no Bean



- O JSF permite a implementação de métodos de validação, que podem ser referenciados por componentes

```
<h:inputText value="#{livro.preco}"  
  validator="#{livro.validarPreco}"  
</h:inputText>
```

Chama o método
validarPreco() no bean

```
public void validarPreco(FacesContext context, UIComponent  
  component, Object value) throws ValidatorException {  
  //...  
}
```



Ignorando a Validação



- Em algumas situações, ignorar a validação dos dados é necessário
- Para ignorar a validação, definir o atributo **immediate** do botão/link como true

```
<h:commandButton action="#{doc.processar}" value="Processar"  
  immediate="true" />
```

