02

Validação e integração com Bean Validation

Aplicando regras de validação em nosso formulário

Por enquanto nosso formulário não está validando nenhuma entrada de dados, por exemplo, tente adicionar um produto sem nome, quantidade ou com algum valor negativo. Todas essas operações funcionam, mas não deveriam. Faz parte de nosso trabalho validar esses valores e enviar uma mensagem ao usuário caso alguma de nossas regras não tenha sido atendida.

Mas como fazer a validação em nosso `Controller`? Isso é muito simples, poderiamos fazer um simples `if`, usar o `Result` para incluir uma mensagem na view e redirecionar o fluxo de volta para o formulário! Algo como:

```
@Post
public void adiciona(Produto produto) {
    if (produto.getQuantidade() < 0) {
        result.include("mensagem", "Não pode cadastrar
            um produto com quantidade negativa!");

        result.forwardTo(this).formulario();
        return;
    }
    dao.adiciona(produto);
    result.include("mensagem", "Produto adicionado com sucesso!");
    result.redirectTo(this).lista();
}
```

Mas repare que isso é muito trabalhoso! Se eu for validar mais campos, preciso repetir esse `if` varias vezes. Cada vez mais aumentamos nosso código e sua complexidade. O VRaptor facilita muito esse processo de validação com o uso de um objeto especializado, o `Validator`.

Validando com VRaptor Validator

Como criar um `Validator`? Lembre-se do conceito de `injeção de dependências`, nós não queremos **criar**, e sim **usar** esse objeto. Vamos pedir para que o VRaptor nos forneça! Basta adicionar esse parâmetro no construtor:

```
@Controller
public class ProdutoController {

    private final Result result;
    private final ProdutoDao dao;
    private final Validator validator;

    @Inject
    public ProdutoController(Result result, ProdutoDao dao, Validator validator) {
        this.result = result;
        this.dao = dao;
        this.validator = validator;
    }
```

```
    @Deprecated
    ProdutoController() {
        this(null, null, null); // apenas para uso do CDI!
    }

    // restante do código omitido
}
```

Para aplicar a mesma validação que fizemos à pouco, mas agora com o `Validator`, podemos utilizar seu método `check` passando uma condição `booleana` e um objeto que representa uma mensagem de validação, o `SimpleMessage`. Caso seu resultado seja `false`, essa mensagem é adicionada numa lista de erros de validação para que posteriormente seja apresentada ao usuário na `view`. Algo como:

```
validator.check(produto.getQuantidade() > 0,
    new SimpleMessage("erro", "Não pode cadastrar um produto com quantidade negativa!"));
```

Repare que invertemos a condição para `> 0`, afinal é isso que queremos garantir.

A classe `SimpleMessage` recebe dois argumentos, o primeiro é uma `categoria` e o segundo a mensagem em si. Essa categoria ajuda quando queremos agrupar mensagens para apresentar na view. Veremos mais sobre como usá-la a seguir.

Redirecionando o fluxo em caso de erros de validação

Nosso próximo passo é configurar para qual página queremos redirecionar o cliente caso exista algum erro de validação. Para fazer isso, podemos usar o método `onErrorUsePageOf()`:

```
validator.onErrorUsePageOf(this).formulario();
```

Nosso método adiciona agora pode validar os casos em que o usuário tentar cadastrar um produto com quantidade negativa:

```
@Post
public void adiciona(Produto produto) {

    validator.check(produto.getQuantidade() > 0,
        new SimpleMessage("erro", "Não pode cadastrar um produto com quantidade negativa!"));

    validator.onErrorUsePageOf(this).formulario();

    dao.adiciona(produto);
    result.include("mensagem", "Produto adicionado com sucesso!");
    result.redirectTo(this).lista();
}
```

Podemos agora criar novas validações adicionando novas chamadas ao método `check`.

Isolando mensagens e internacionalização

Mas repare que a mensagem de validação foi inserida diretamente em nosso código, dessa forma nossos `Controller`s ficam cheios de mensagens e fica dificil internacionalizar seus valores. Essa não é uma boa prática.

Além da `SimpleMessage` existe uma outra opção de mensagem, a `I18nMessage`! No lugar de receber diretamente a mensagem, essa classe recebe uma **chave** para a verdadeira mensagem, que estará configurada em um arquivo de configuração de nosso projeto. Vamos mudar nosso código para utilizar essa implementação.

```
@Post
public void adiciona(Produto produto) {

    validator.check(produto.getQuantidade() > 0,
        new I18nMessage("erro", "produto.quantidade.negativa"));

    validator.onErrorUsePageOf(this).formulario();

    dao.adiciona(produto);
    result.include("mensagem", "Produto adicionado com sucesso!");
    result.redirectTo(this).lista();
}
```

Agora basta adicionar o arquivo de configuração dessas mensagens em nosso `src`, o arquivo deve se chamar `messages.properties`.

Dentro do arquivo, nós colocamos a **chave** e o **valor** (mensagem) correspondente ao erro:

```
produto.quantidade.negativa = Não pode cadastrar um produto com quantidade negativa!
```

Após aplicar essa configuração, vamos reiniciar o tomcat e tentar adicionar um produto com quantidade negativa. Repare, ao validar ele redirecionou para o formulario (antes de adicionar o produto no banco de dados) e nos mostrou a mensagem de validação que escrevemos no arquivo **messages.properties**.

Exibindo os erros em nossa View

Repare que quando existe um erro de validação, nós estamos perdendo as informações que digitamos anteriomente. Se for um formulário com vários campos, com certeza vai incomodar bastante o usuário. Para manter esses dados em caso de erro de validação, basta adicionar o atributo value nos `input`s com o atributo do produto na `Expression Language`:

```
Nome: <input type="text" name="produto.nome" value="${produto.nome}"/>
Valor: <input type="text" name="produto.valor" value="${produto.valor}"/>
Quantidade: <input type="text" name="produto.quantidade" value="${produto.quantidade}"/>
```

Apenas com essa mudança já podemos testar que ao preencher um produto com quantidade negativa e submeter o formulario, o VRaptor vai redirecionar para o formulário com os campos populados.

O último passo é mostrar para o usuário a mensagem de validação. O Validator do VRaptor automaticamente inclui na view uma variável chamada `errors`. Podemos recuperar essa variável com o uso de `EL`, e iterar sobre seus valores:

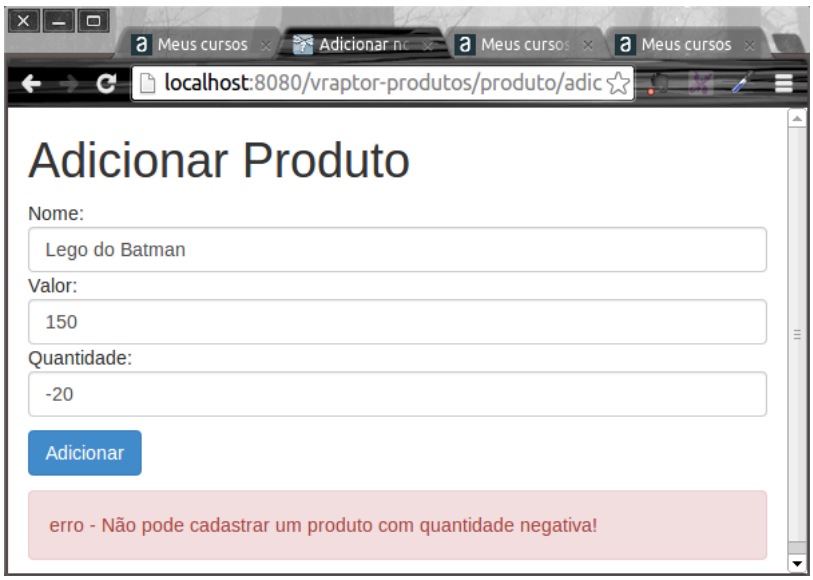
```
<c:forEach var="error" items="${errors}">
    ${error.category} - ${error.message}<br/>
```

```
</c:forEach>
```

Vamos adicionar esse trecho de código em nosso `formulario.jsp`, logo depois da tag `form`:

```
<!-- inicio do arquivo omitido -->
    </form>
       <c:if test="${not empty errors}">
           <div class="alert alert-danger">
               <c:forEach var="error" items="${errors}">
                   ${error.category} - ${error.message}<br />
               </c:forEach>
           </div>
       </c:if>
   </div>
  </body>
</html>
```

Pronto, agora basta restartar o servidor e tentar cadastrar um produto com quantidade negativa novamente. Repare que agora a mensagem de validação é apresentada para o usuário.



Utilizando as anotações do Bean Validation

O VRaptor possui integração com o `Bean Validation`, dessa forma conseguimos utilizar suas anotações para validar os campos dos nossos modelos. Vamos fazer a mesma validação da quantidade, agora anotando esse atributo da classe `Produto` com `@Min(value=0)`

```
@Entity
public class Produto {

    @GeneratedValue @Id
    private Long id;

    private String nome;

    private Double valor;
```

```
    @Min(value=0)
    private Integer quantidade;

    // continuação do código
}
```

Para que a validação aconteça, precisamos adicionar a anotação `@Valid` no parametro `Produto` do nosso método `adiciona`:

```
@Post
public void adiciona(@Valid Produto produto) {
        validator.onErrorUsePageOf(this).formulario();
    dao.adiciona(produto);
    result.include("mensagem", "Produto adicionado com sucesso!");
    result.redirectTo(this).lista();
}
```

Vamos testar! Basta restartar o servidor e tentar cadastrar um produto com valor negativo.

Excelente, ele está validando! Mas note que a mensagem foi:

```
adiciona.produto.quantidade - must be greater than or equal to 0
```

Podemos modificar essa mensagem adicionando o atributo message nas anotações de validação, neste caso na `@Min`:

```
@Min(value=0, message="Minha mensagem")
private Integer quantidade;
```

Mas para evitar inserir varias mensagens de validação em nosso código, vamos isolar no arquivo de internacionalização. Basta adicionar a chave(`key`) da sua mensagem entre `{}`:

```
@Min(value=0, message="{produto.quantidade.negativa}")
private Integer quantidade;
```

E agora adicionar a chave e valor no arquivo `.properties`. Uma diferença importante aqui, é que o Bean Validation exige que você chame o arquivo de `ValidationMessages.properties`, e não de `messages.properties` como estavamos fazendo. Vamos acessar o arquivo `ValidationMessages.properties` que já está dentro da pasta `src/main/resources`, e adicionar o conteúdo:

```
produto.quantidade.negativa = Não pode cadastrar um produto com quantidade negativa!
```

Pronto! Vamos testar mais uma vez agora, tudo o que precisamos fazer é restartar o servidor e tentar adicionar um novo produto com quantidade negativa.

Vamos adicionar mais algumas regras de validação! Outros casos bem comuns que costumamos validar no dia a dia é por exemplo o tamanho do texto do nome do produto. E também que o campo nome do produto não pode ser vazio. Podemos adicionar as anotações `@Size` e `@NotEmpty` para garantir isso:

```
@Entity
public class Produto {

    @GeneratedValue @Id
    private Long id;

    @NotEmpty @Size(min=0)
    private String nome;

    @Min(value=0)
    private Double valor;

    @Min(value=0)
    private Integer quantidade;

    // continuação do código
```