02

Validação

Validação da entrada de dados

É comum que usuários preencham valores inválidos para nossa aplicação. Nada impede, por exemplo, o cadastro de um produto com preço negativo. É trabalho do desenvolvedor garantir que esses dados sejam validados antes de serem salvos no banco de dados. Mais ainda, o desenvolvedor deve exibir o erro para o usuário para que o mesmo possa consertá-lo depois. Fazer isso manualmente pode ser trabalhoso. O VRaptor provê uma excelente maneira de validar dados enviados pelo usuário.

Usando o Validator

A validação deve ser feita dentro do `ProdutoController` no método `adiciona(..)` antes de salvar o produto pelo *Dao*. Deve haver uma condição (*if*) que verifica, por exemplo, o preço do produto. Dentro desse *if* devemos criar alguma mensagem para o usuário final informando sobre o problema de validação.

Para facilitar o processo o VRaptor fornece um objeto especializado para a validação: a classe `Validator`. Para começarmos a utilizá-la, devemos recebê-la no construtor para que o VRaptor a injete:

```
import br.com.caelum.vraptor.Validator;

@Resource
public class ProdutoController {
    private Validator validator;

    public ProdutoController(Validator validator, ...) {
        this.validator = validator;
    }

}
```

Criando erros de validação e redirecionando para o formulário

Com uma instância de *validator* em mãos, basta o usuário adicionar qualquer erro existente nele. Ou seja, dentro do *if* chamamos o método `add(..)` para adicionar uma nova mensagem do tipo `ValidationMessage`. O primeiro parâmetro representa a mensagem de erro que será exibida ao usuário, e o segundo é a categoria a que esse erro pertence. Pode haver mais de um erro por categoria.

Além disso, é preciso configurar o local para onde o VRaptor deve redirecionar caso haja um problema de validação. O *validator* possui um método `onErrorUsePageOf(..)` para definir o controller e método. Em nosso caso, estamos dizendo ao *validator* que ele deve redirecionar para o `formulario()` do `ProdutoController`.

Observe o exemplo abaixo, no qual a ação `adicionar()` não permite que um produto com preço menor que R$ 0,10 seja adicionado:

```
public void adicionar(Produto produto) {
    if(produto.getPreco()<0.1) {
        validator.add(new ValidationMessage("O preço deve ser maior que R$ 0.1", "preco"));
    }
```

```
        validator.onErrorUsePageOf(ProdutoController.class).formulario();
    }
```

Ao testar no navegador e inserir um produto com o preço negativo, o VRaptor automaticamente volta para o formulário.

Mostrando dados e mensagens para o usuário

Podemos perceber que perdemos os dados já inseridos do produto quando existe um erro. Isso, com certeza, vai atrapalhar o usuário. Vamos garantir que o formulário continue preenchido, mesmo com erro de validação. No "formulario.jsp" precisamos completar cada input, de maneira que os dados do produto sejam renderizados. Colocamos nos inputs o atributo `value` com o produto na *expression languange*, ou seja, `${produto.nome}` , `${produto.descricao}` e `${produto.preco}` .

Já podemos testar isso no navegador. Ao preencher o formulário com um preço inválido voltamos para ele e os dados continuam na tela.

Ainda nos falta mostrar o erro de validação para o usuário. Para tal, o VRaptor automaticamente atribuirá uma variável chamada `errors` , que pode ser capturada pelo uso da *expression language* dentro do JSP. Vamos alterar o "formulario.jsp" e usar JSTL para iterar pelos erros. A tag `c:forEach` fará esse trabalho. Ao definir uma variável, podemos mostrar a categoria e a mensagem do erro.

O código abaixo mostra todos os erros que foram encontrados para o usuário:

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

Falta testar no navegador e tentar inserir um produto com preço negativo. Repare que, após termos submetido os dados, voltamos para o formulário e um erro é apresentado, acusando o preço do produto.

Estilos fluentes de validação

O VRaptor também provê outra maneira para chamar a API de validação. A ideia é tornar a leitura do código mais fluente, sem escrever muitos *if*s. O objeto *validator* possui um método `checking()` que recebe um objeto especializado que encapsula a regra de validação. Esse objeto é do tipo `Validations` .

Vamos instanciar a classe `Validations` e definir um bloco logo depois do construtor. Lá dentro usaremos o método `that` que recebe a condição de validação, a categoria e a mensagem. Mas, desta vez, será apenas a chave da mensagem. Para usar a variável "produto" dentro do bloco é preciso declará-la como `final` .

Repare que conseguimos ler o código. No caso, ele checa se o preço do produto é maior a 0.1; caso isso não seja verdade, ele cria um erro com a mensagem e categoria passados:

```
public void adicionar(final Produto produto) {

    validator.checking(new Validations(){{
        that(produto.getPreco() > 0.1,"erro", "produto.preco.invalido");
    }});

    validator.onErrorUsePageOf(ProdutoController.class).formulario();
}
```

Internacionalização das mensagens

Algo interessante de se notar é que não passamos a verdadeira mensagem dentro do código Java. Passamos apenas uma chave, no nosso caso estamos usando `produto.preco.invalido`. Essa chave deve ser configurada no arquivo `messages.properties`, o qual deve ser criado na pasta `src` do projeto.

Então, criemos esse arquivo pelo Eclipse e colocamos lá dentro a chave com a mensagem *O preço deve ser maior que R$ 0.1*. Lembrando que este tipo de arquivo usa uma estrutura parecida com os mapas do Java, nos quais é configurada uma chave associada a um valor. Segue abaixo um exemplo:

messages.properties

```
produto.preco.invalido = Preço deve ser maior que R$ 0.1
```

Vamos reiniciar o servidor e testar novamente o formulário. Ao validar o preço do produto, aparece a mensagem que foi carregada do arquivo `messages.properties`.

Para saber mais:

Podemos usar qualquer redirecionamento do VRaptor em caso de erro através da instrução `validator.onErrorUse()`. O código abaixo, por exemplo, redireciona para a ação `lista()` do `ProdutoController`:

```
validator.onErrorUse(logic()).redirectTo(ProdutoController.class).lista();
```

Encontramos mais informações sobre validações na documentação do VRaptor em (http://www.vraptor.org/pt/docs/validacao/)http://www.vraptor.org/pt/docs/validacao/ (http://www.vraptor.org/pt/docs/validacao/).