09

Usando a Bean Validation

Transcrição

Mas vamos pensar com calma. O que realmente queremos validar no título, só queremos validar se ele é obrigatório? Imagine outro cenário em que o usuário só preenche um espaço para driblar nossa validação. Como resolvemos isso?

Faremos então uma validação especifica que o Java EE já possui e que é feita diretamente na classe Java, esta validação é chamada de **Bean Validation**. A `Bean Validation` é uma especificação do JavaEE e toda especificação precisa ter uma implementação, o **WildFly** já tem um framework que implementa esta especificação internamente no próprio servidor, que é o **Hibernate Validator**.

Usando o **Hibernate Validator**, quando queremos dizer que o Título não pode ser nulo, nem vazio e nem possuir espaços em branco, usamos a annotation `@NotBlank` em cima do atributo.

Porém, existem outras coisas que queremos validar, por exemplo, o valor mínimo aceito pelo preço e número de páginas, o tamanho mínimo preenchido para o campo descrição, e por aí vai. Para vários desses casos, temos annotations específicas da **Bean Validation**, e para os casos não cobertos pela **Bean Validation** o **Hibernate Validator** veio para cobrir.

Vamos modificar a nossa classe para usar as anotações de validação:

```
@Entity
public class Livro {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @NotBlank // Valida já vazio e espeços em branco
    private String titulo;

    @Lob
    @Length(min=10) // Número mínimo de caracteres que o campo pode ter
    @NotBlank
    private String descricao;

    @DecimalMin("20") // Valor decimal mínimo
    private BigDecimal preco;

    @Min(50) // Valor inteiro mínimo
    private Integer numeroPaginas;

    @ManyToMany
    @Size(min=1) // número mínimo de elementos na lista
    @NotNull // A lista não pode ser nula
    private List<Autor> autores = new ArrayList<>();

    // getters e setters abaixo

}
```

Faça *Full Publish* e teste nossa aplicação. Observe que as mensagens já estão em português porque o próprio **Hibernate Validator** já consegue tranformar. Tente cadastrar também espaço em branco e veja como o formulário se comporta.

Um problema que encontramos agora é que a mensagem não está informando o campo a que ela se refere. Vamos melhorar isso. Abra o arquivo `jsf_messages.properties` que criamos e vamos adicionar mais uma linha:

```
# Mantenha as demais mensagens acima.
javax.faces.validator.BeanValidator.MESSAGE={0}
```

Ao informar `{0}` estamos informando que só queremos exibir a mensagem. Mas ainda assim teríamos o problema de ter a mensagem e não sabermos o campo. Para resolver este problema, uma prática muito usada no mercado é exibir a mensagem na frente do próprio campo, assim fica mais claro para o usuário o que aconteceu no campo que ele preencheu. Vamos voltar ao nosso formulário e adicionar um `<message>` logo após o `input` .

```
<div>
    <h:outputLabel value="Título" />
    <h:inputText value="#{adminLivrosBean.livro.titulo}"
            required="true" />
    <h:message for="titulo" />
</div>
```

Vamos tratar primeiro o campo título. Adicionamos um componente `<h:message />` para exibir somente uma mensagem de erro, que é diferente do componente `<h:messages />` (plural) que exibe uma lista de erros. A outra diferença é no atributo `for` do componente `<h:message>` , onde vinculamos a mensagem ao campo a que ele se refere. Mas falta uma coisa: quem é título?

Perceba que no `input` não dizemos o nome dele, e o JSF precisa identificar o input para colocar a mensagem correta no campo. Obtemos esse resultado com o atributo `id` . Praticamente toda tag do JSF possui o atributo `id` , e no caso dos inputs, também servem para referenciar a mensagem com o `input` dela. Assim, o código completo do campo Título será:

```
<div>
    <h:outputLabel value="Título" />
    <h:inputText value="#{adminLivrosBean.livro.titulo}"
            required="true" id="titulo" />
    <h:message for="titulo" />
</div>
```

Faça o mesmo para os demais campos do nosso formulário.

Agora já não precisamos mais do componente `<h:messages />` que ficava no início do formulário, já que estamos apresentando a mensagem campo a campo, e não precisamos mais das validações específicas para número de páginas e preço, já que a **Bean Validation** está cuidando disso.

Se você testar a aplicação nesse momento, perceberá que todas as validações funcionam, mas quando não selecionamos nenhum autor, recebemos um erro `ConstraintValidationException` . O que está acontecendo é que a nossa classe Livro não está vinculada diretamente com o Autor e sim com uma lista de Integers, que são os ids dos autores. Apenas no momento de salvar de fato no banco de dados, é que ocorre a validação e por isso recebemos um

erro tão feio, ao invés das validações elegantes que já temos nos demais campos. Mas vamos tratar deste caso mais a frente em nosso curso.