02

Como funcionam os ajudantes de formulário

Uma das grandes facilidades do Play! são as views compiladas. Toda view (arquivo com extensão `.scala.html` na pasta `app/views/`) é compilada, tenha ela código dinâmico ou não. Então vamos utilizar esse poder para garantir que não ocorram erros e facilitar nosso trabalho na hora de criar os formulários. Vamos começar?

Temos um formulário de produto com a seguinte estrutura:

```
<form action="/produto/novo" method="post">
    <!-- título e outros inputs -->
</form>
```

O primeiro passo é substituir a tag `<form>` pelo ajudante. Para utilizar o ajudante de formulário, deve-se utilizar o `@` , que declara que o conteúdo a seguir será código compilável, seguido do pacote `helper` . Utiliza-se então o método `form` , que recebe uma rota já previamente configurada como parâmetro.

Portanto, a sintaxe correta é a seguinte, adequando o parâmetro da rota caso tenha criado uma diferente:

```
@helper.form(routes.ProdutoController.salvaNovoProduto) {
    <!-- título e outros inputs -->
}
```

A alternativa a seguir não pode ser utilizada, apesar de ser intuitiva:

```
@helper.form("/produto/novo", "post") {
    <!-- título e outros inputs -->
}
```

Isso acontece pois o ajudante de formulário precisa receber um objeto do tipo `play.api.mvc.Call` e se você tentar utilizá-la, vai receber o seguinte erro que deixa isso claro:

```
type mismatch;
[error]  found   : String("/produto/novo")
[error]  required: play.api.mvc.Call
```

Apesar disso, existe uma outra alternativa que apesar de não ter sido abordada em aula, é válida também! Ela usa a seguinte sintaxe:

```
@helper.form(play.api.mvc.Call.apply("post", "/produto/novo", null)) {
    <!-- título e outros inputs -->
}
```

Isso acontece pois estamos instanciando um objeto adequado para o ajudante de formulário. Essa alternativa pode ser útil caso você queira criar um formulário que será enviado para alguma rota externa ao seu sistema.

Os ajudantes são úteis impedindo erros na hora de digitar a url e método do seu formulário, mostrando erros de compilação caso a rota especificada não exista. Agora podemos simplificar um pouco mais utilizando o ajudante de formulários para gerar os campos do formulário e seus rótulos, ou labels em inglês. Vamos começar com o título?

Vamos usar o ajudante de campo de texto! Ele precisa de um objeto que representa um campo de formulário, um `Field`. Substitua o input e sua label do campo de título pelo seguinte:

```
@helper.inputText(formulario("titulo"))
```

Vai ocorrer um erro, pois não temos essa variável `formulário` disponível para nós. E como podemos passar essa variável para a view? Bom, primeiro a gente tem que declarar que a view recebe um parâmetro. Na primeira linha, altere o parâmetro message para `@(formulario DynamicForm)`. Este erro se resolve, mas surge outro. Não passamos o parâmetro quando utilizamos a view lá no controller. O que precisamos fazer é gerar um objeto de formulário e passá-lo para a view como parâmetro do método `render()`, e para isso usamos a fábrica de formulários:

```
public Result formularioDeNovoProduto() {
    DynamicForm formulario = formularios.form();
    return ok(formularioDeNovoProduto.render(formulario));
}
```

Agora os erros todos somem e a gente pode abrir a página do formulário. E olha só, ele já vem com uma label! Bacana, vamos substituir os outros campos também.

```
@helper.inputText(formulario("titulo"))
@helper.inputText(formulario("codigo"))
@helper.inputText(formulario("tipo"))
@helper.textarea(formulario("descricao"))
@helper.inputText(formulario("preco"), 'type -> "number")
<input type="submit" value="Cadastrar">
```

Repare que o preço não é um `inputNumber` mas um `inputText` com um atributo `number`. Essa notação com a apóstrofe serve pra indicar que você quer alterar um atributo do seu campo.

Mas e se a gente quiser passar um valor padrão para o formulário? O Play! nos fornece dois jeitos interessantes de fazer isso: um deles é preencher o formulário com um mapa, antes de passar para a view.

```
public Result formularioDeNovoProduto() {
    Map<String, String> params = new HashMap<String, String>();
    params.put("tipo", "e-book");
    DynamicForm formulario = formularios.form().fill(params);
    return ok(formularioDeNovoProduto.render(formulario));
}
```

Mas existe um jeito mais prático de fazer isso. Sabemos que o formulário é de um produto, certo? Então podemos criar um formulário preenchido a partir de um objeto Produto existente! Olha só:

```
public Result formularioDeNovoProduto() {
    Produto produto = new Produto();
    produto.setTipo("e-book");
```

```
        Form<Produto> formulario = formularios.form(Produto.class).fill(produto);
        return ok(formularioDeNovoProduto.render(formulario));
    }
```

Só que agora o formulário é de um tipo específico, não dinâmico, então ocorre um erro de compilação. Precisamos refletir a mudança na view, alterando o tipo da variável `formulario` :

```
@(formulario: Form[Produto])
```

Os formulários Play! refletem os nomes e valores do objeto passado para os campos do formulário, permitindo que utilizemos um objeto java para preenchê-lo de modo automático.