02

Preparando ProdutosController

Transcrição

Nossa aplicação está começando a tomar forma. Ela já lista os produtos em nossa **home**. Agora começaremos a criar o cadastro de produtos, para assim podermos cadastrar livros da Casa do Código.

Para o cadastro de produtos, precisaremos de um formulário. Sendo assim, crie um novo arquivo **JSP** chamado **form.jsp** dentro da pasta `WEB-INF/views/produtos/` . A pasta `produtos` ainda não existe, teremos que criá-la também. faremos alterações para deixar no padrão HTML 5. O arquivo `form.jsp` inicial deve estar parecido com esse:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa do Código</title>
</head>
<body>

</body>
</html>
```

O próximo passo é criar o **html** referente ao formulário (`form`) de cadastro dos livros. Eles terão inicialmente os seguintes atributos: **Título**, **Descrição** e **Número de Páginas**, todos do tipo texto. Vamos então criar o formulário com estes campos. O formulário deve ficar parecido com este:

```
<form action="/produtos" method="post">
    <div>
        <label>Título</label>
        <input type="text" name="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <textarea rows="10" cols="20" name="descricao"></textarea>
    </div>
    <div>
        <label>Páginas</label>
        <input type="text" name="paginas" />
    </div>
    <button type="submit">Cadastrar</button>
</form>
```

Note que estamos fazendo o formulário enviar seus dados para o **path** `/produtos` e que o estamos enviando via `post` no método do formulário.

Nosso `form.jsp` deve ser acessado na url `localhost:8080/casadocodigo/produtos/form` . Se acessarmos agora, veremos uma página de `erro 404` , pois as *views* não podem ser acessadas diretamente. Lembra? Como resolvemos isso? Criando um `Controller` !

Crie então o `ProdutosController` dentro do pacote `br.com.casadocodigo.loja.controllers`. Neste `Controller` crie o método `form` que retorna a *view* com o formulário. Mapeie o **path** que este método vai atender com a anotação `@RequestMapping`. O `ProdutosController` deve ficar parecido com este:

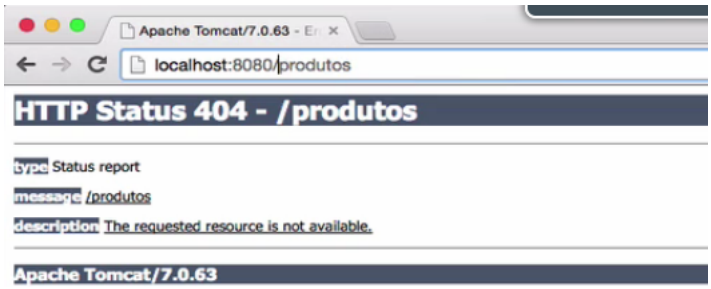
```
@Controller
public class ProdutosController {

    @RequestMapping("/produtos/form")
    public String form(){
        return "produtos/form";
    }

}
```

Se acessarmos agora o nosso formulário em `localhost:8080/casadocodigo/produtos/form`, devemos vê-lo sem nenhum problema. Não esqueça de reiniciar o servidor! Teste o formulário, tente cadastrar um livro!

Quando enviamos o formulário, recebemos um `erro 404`:



Mas tem algo estranho neste erro 404, note a **url**. O caminho está sem o `/casadocodigo/`. Isto acontece porque o nosso formulário não aponta `/casadocodigo/` em sua action. Vamos fazê-lo apontar, então.

```
<form action="/casadocodigo/produtos" method="post">
    [...]
</form>
```

Este é o primeiro passo para resolver nosso problema. Agora precisamos mapear o `path` `/produtos` para um método no `ProdutosController`. Algo parecido com:

```
@RequestMapping("/produtos")
public String gravar(String titulo, String descricao, int paginas){
    System.out.println(titulo);
    System.out.println(descricao);
    System.out.println(paginas);

    return "ok";
}
```

Nosso método atende ao *path* `/produtos` e se chama `gravar`. Este irá gravar os produtos em um banco de dados posteriormente. Ele recebe os dados do produto e imprime no terminal, por fim, retorna "ok" para indicar que tudo ocorreu bem.

Faça alguns testes! Teremos um erro 404 indicando que a *view* `ok` não foi encontrada, mas não deixe de verificar se os dados do formulário foram impressos no console. Funciona! O SpringMVC sozinho verifica a assinatura do nosso método e faz um *bind* dos parâmetros do método com os *names* do formulário.

Nossa aplicação já funciona, mas antes de continuarmos, vamos melhorar um ponto e corrigir outro.

Primeiro ponto, imagine que o formulário de produtos terá 30 campos. A assinatura do nosso método ficará enorme! Vamos mudar isso, o método `gravar` requer um **produto**. Vamos criar um Produto então.

Crie uma classe chamada `Produto` com os mesmos atributos do formulário e os defina como `private`. Use os atalhos do Eclipse e gere também os *getters and setters*. Gere também o `toString` na classe `Produto` para que deixemos de imprimir aquela mensagem padrão estranha e possamos imprimir o objeto diretamente de forma amigável.A classe `Produto` deve estar no pacote `br.com.casadocodigo.loja.models`.Esta classe representa uma entidade no nosso sistema.

```
package br.com.casadocodigo.loja.models;

public class Produto {
    private String titulo;
    private String descricao;
    private int paginas;


    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public String getDescricao() {
        return descricao;
    }
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }
    public int getPaginas() {
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }

    @Override
    public String toString() {
        return "Produto [titulo=" + titulo + ", descricao=" + descricao + ", paginas=" + paginas
    }

}
```

Desta forma isolamos todo o comportamento e dados dos produtos em uma classe. Podemos então em nosso **Controller** receber um **Produto** em vez de seus dados separadamente.O SpringMVC fará o `bind` dos *names* em nosso formulário com os atributos do **Produto** de agora em diante. Sendo assim, vamos modificar o `ProdutosController` para recebermos um objeto `produto` agora.

O método `gravar` deve ficar assim:

```
@RequestMapping("/produtos")
public String gravar(Produto produto){
    System.out.println(produto);
    return "/produtos/ok";
}
```

Agora recebemos um objeto do tipo produto, imprimimos o produto no console e retornamos a *view* "ok", que deve estar dentro da pasta produtos (perceba que mudamos também o caminho de retorno da view de `/ok` em `/views/` para `/produtos/ok` em `/views/produtos/ok`). Crie a *view* `ok.jsp` na pasta `WEB-INF/views/produtos` com uma mensagem de sucesso. Algo como:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
        <html>
<head>
<meta charset="UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa do Código</title>
</head>
<body>
 <h1>Produto cadastrado com sucesso!</h1>
</body>
</html>
```

Agora, se acessarmos nosso formulário, preenchermos os campos e tentarmos cadastrar um produto (um livro), teremos a mensagem de sucesso e os dados do nosso livro serão mostrados no console. Faça o teste!