Spring MVC I: Criando aplicações web
22%

AULA 05 Redirect com Escopo de Flash

ATIVIDADES
01 Redirect com Escopo de Flash 17min
02 Escopo de Flash
03 Always Redirect After Post
04 Exibindo a Listagem de Produtos
05 Usando o FlashScoped

01 Redirect com Escopo de Flash

PRÓXIMA ATIVIDADE

Transcrição

O cadastro de produtos da nossa aplicação já funciona, mas da mesma forma que melhoramos as rotas no capítulo passado, vamos melhorar outro ponto da aplicação neste momento.

É muito comum que após o cadastro de produtos, por exemplo, em vez de ser mostrada uma página com a mensagem de **cadastro realizado com sucesso**, sejamos levados novamente à **lista com todos os produtos.** Faremos essa modificação agora.

Para que após o cadastro de produtos, sejamos levados à página de listagem, devemos então modificar o método `gravar` no `ProdutosController`, para que este método chame o método `listar` em vez de retornar a *view* `ok.jsp`.

Nosso código atualmente está assim:

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

Vamos então, simplesmente mudar a rota do retorno do método para o endereço `/produtos`. Nosso código ficará assim:

```
@RequestMapping(method=RequestMethod.POST)
public String gravar(Produto produto){
    System.out.println(produto);
    produtoDao.gravar(produto);
    return "produtos";
}
```

Com essa mudança, quando cadastrarmos um novo produtos, o método `gravar` deve salvar o produto no banco de dados e então o **Spring** deve chamar o método `listar` que é responsável por atender o endereço `/produtos`. Vamos fazer um teste e realizar um cadastro em "novo produto".

```
![Erro 404 - página produtos.jsp não encontrada](https://s3.amazonaws.c
```

O que acontece? Vemos uma mensagem de **Erro 404.**

A página `produtos.jsp` não foi encontrada, porque realmente não temos uma página `produtos.jsp`, temos uma página `lista.jsp`.

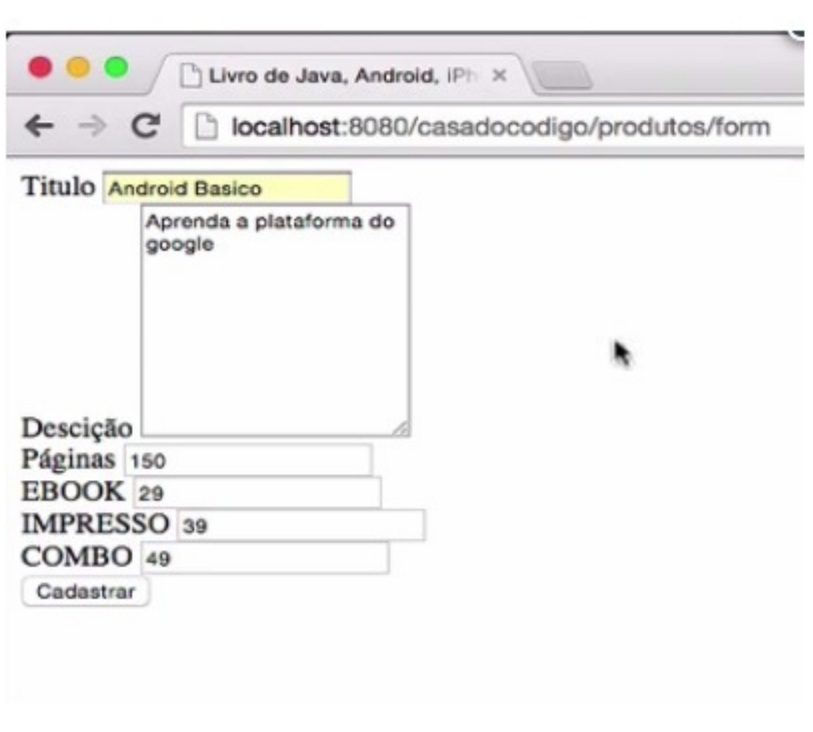
O **Spring**, quando retornamos uma **String** procura uma página com o mesmo nome da **String** que retornamos, por isso ele está procurando uma página chamada `produtos.jsp`.

Não era o que queríamos... Nós queremos ver a listagem de produtos que já está pronta na nossa página `lista.jsp`. Sabendo disso, vamos chamar o método `listar` do `ProdutosController` diretamente, isto fará com que o **Spring** carregue a listagem dos produtos e mostre a listagem no navegador.

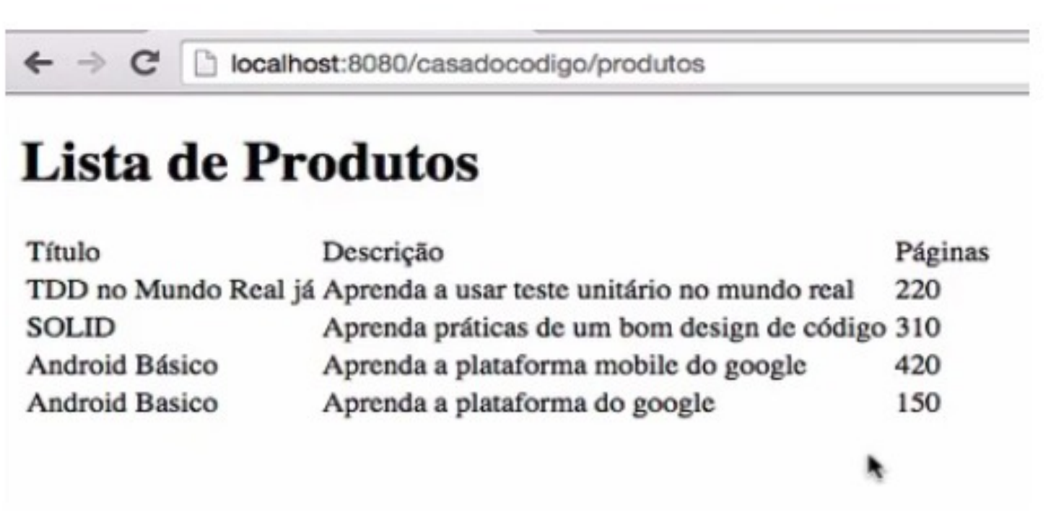
Logo, iremos mudar o retorno do nosso método `gravar` para que seja o mesmo do método `listar`. E depois, chamar o método `listar` com o `return`. Veja como ficou o código com as nossas modificações.

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView gravar(Produto produto){
    System.out.println(produto);
    produtoDao.gravar(produto);
    return listar();
}
```

Observe que o retorno do método muda, pois o método `listar` retorna um objeto do tipo `ModelAndView`. Vamos cadastrar um novo produto agora. Após o cadastro, devemos ver a listagem de produtos com o nosso novo produto.



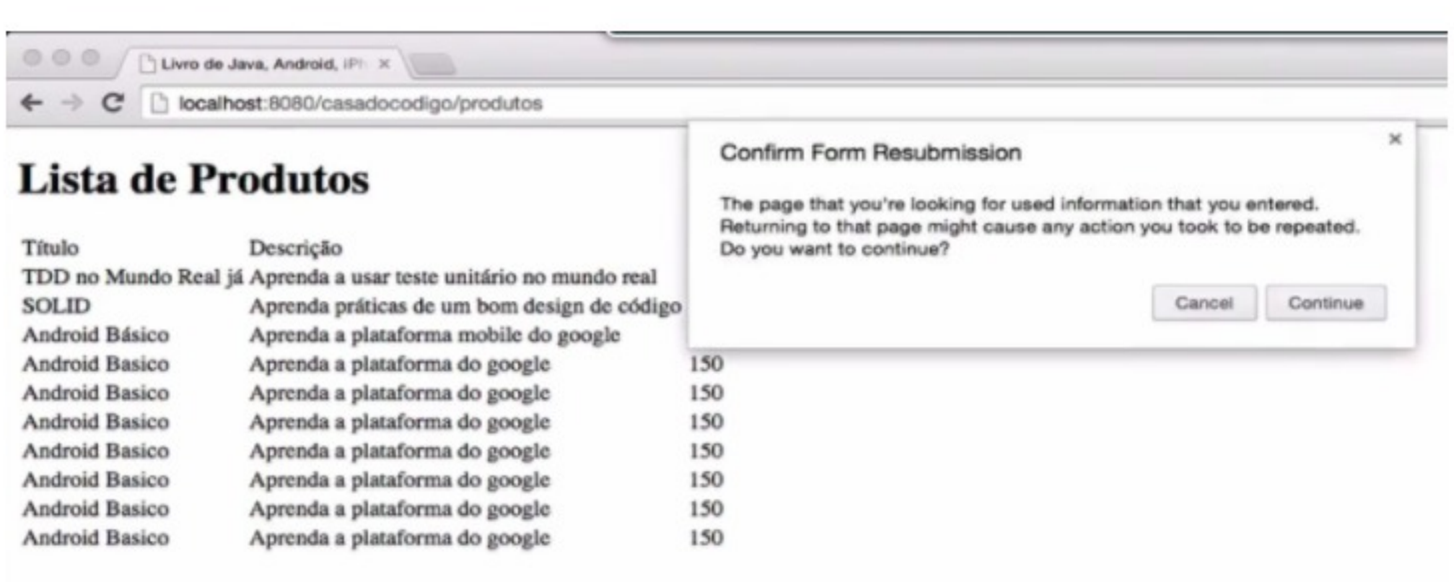
Titulo Android Basico
Aprenda a plataforma do google
Desciçâo
Páginas 150
EBOOK 29
IMPRESSO 39
COMBO 49



Lista de Produtos

Título	Descrição	Páginas
TDD no Mundo Real já	Aprenda a usar teste unitário no mundo real	220
SOLID	Aprenda práticas de um bom design de código	310
Android Básico	Aprenda a plataforma mobile do google	420
Android Basico	Aprenda a plataforma do google	150

Experimente apertar `F5` após o cadastro de algum produto.



Lista de Produtos

Título	Descrição	
TDD no Mundo Real já	Aprenda a usar teste unitário no mundo real	
SOLID	Aprenda práticas de um bom design de código	
Android Básico	Aprenda a plataforma mobile do google	
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150
Android Basico	Aprenda a plataforma do google	150

Confirm Form Resubmission

The page that you're looking for used information that you entered. Returning to that page might cause any action you took to be repeated. Do you want to continue?

Veja que o navegador nos questiona se queremos **resubmeter** o formulário. Isso quer dizer que se confirmarmos, ele vai enviar os dados do produto novamente e teremos o produto **recadastrado.** Duplicação de produtos não é bom!

Acontece que o navegador ainda está guardando os dados do post do formulário. Apesar de ser um problema real, não podemos culpar o navegador, pois este é o funcionamento normal no caso de posts de formulário. Modificaremos então este comportamento em nossa aplicação.

Resolveremos isto através de recursos do protocolo `HTTP`. Já usamos outros recursos do protocolo (GET e POST). Agora, usaremos um recurso chamado de `redirect`, que passa um `status` para o navegador carregar uma outra página e esquecer dos dados da requisição anterior. O `status` que o navegador recebe é um `302`.

Para isso devemos mudar a última linha do método `gravar`, que agora vai retornar um novo `ModelAndView` usando como rota o `redirect:produtos`.

Após as modificações, o método `gravar` ficará igual ao que está abaixo:

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView gravar(Produto produto){
    System.out.println(produto);
    produtoDao.gravar(produto);
    return new ModelAndView("redirect:produtos");
}
```

Teste cadastrar um novo produto e atualizar a página de listagem depois do cadastro. O navegador não nos pede mais confirmação de resubmissão do formulário e também não duplicará os produtos na listagem.

Nossa aplicação ainda precisa de mais um ajuste. Ela não mostra mais a mensagem de **produto cadastrado com sucesso** como tínhamos na *view* `ok.jsp`. Como é uma simples mensagem, usaremos um recurso do **Spring** que nos permite enviar informações entre **requisições.** Esse recurso é o `RedirectAttributtes`.

O método `gravar` agora deve receber um objeto do tipo `RedirectAttributes` fornecido pelo **Spring.** Usaremos então esse objeto para adicionar um atributo do tipo `Flash` (usando o método `addFlashAttribute` deste objeto), passando assim a uma chave `sucesso` e o valor dessa chave que é `Produto cadastrado com sucesso!`.

Veja como fica nosso método `gravar` com esta nova modificação:

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView gravar(Produto produto, RedirectAttributes redirect
    System.out.println(produto);
    produtoDao.gravar(produto);
    redirectAttributes.addFlashAttribute("sucesso","Produto cadastrado
    return new ModelAndView("redirect:produtos");
}
```

Observação: Atributos do tipo `Flash` têm uma particularidade que é interessante observar. Eles só duram até a próxima requisição, ou seja, transportam informações de uma requisição para a outra e então, deixam de existir.

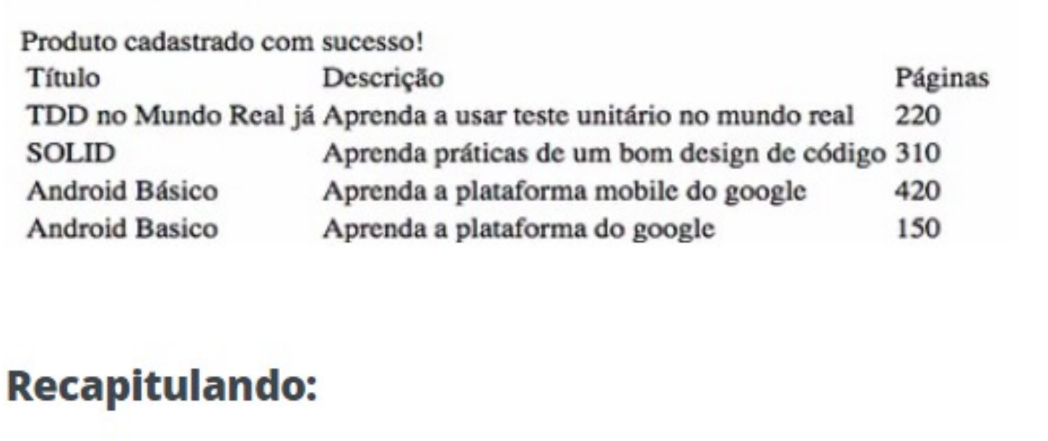
Note que usamos um `RedirectAttributes`. Isto faz muito sentido, já que após o post iremos redirecionar a página. A prática de fazer redirecionamentos após posts tem um nome bem conhecido, **Always redirect after post** (em português, significa **Sempre redirecione após post**).

Agora para exibirmos esta mensagem em nossa listagem, devemos modificar o `lista.jsp` para exibir o `RedirectAttributes`, que é acessado através da chave `message`, veja como fica nosso `lista.jsp`:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa d
</head>
<body>
    <h1>Lista de Produtos</h1>
    <p> ${sucesso} </p>
    <table>
        <tr>
            <td>Título</td>
            <td>Descrição</td>
            <td>Páginas</td>
        </tr>

        <c:forEach items="${produtos}" var="produto">
            <tr>
                <td>${produto.titulo}</td>
                <td>${produto.descricao}</td>
                <td>${produto.paginas}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Veja como ficou nossa mensagem de sucesso:



Lista de Produtos

Produto cadastrado com sucesso!

Título	Descrição	Páginas
TDD no Mundo Real já	Aprenda a usar teste unitário no mundo real	220
SOLID	Aprenda práticas de um bom design de código	310
Android Básico	Aprenda a plataforma mobile do google	420
Android Basico	Aprenda a plataforma do google	150

Recapitulando:

Até aqui fizemos nossa listagem de produtos, que usa o `DAO` para recuperar os produtos do banco de dados. Simplificamos nossas rotas assinando o `ProdutosController` com a anotação `RequestMapping('produtos')` e diferenciamos os métodos `listar` e `gravar` que mapeiam a mesma rota, através dos métodos usados pelo **protocolo HTTP** (`GET` e `POST`).

Também aprendemos a redirecionar de uma página para outra. Vimos o conceito de ***Always redirect after post*** (que significa, "Sempre redirecione depois de post"). Evitando que dados sejam reenviados para nossa aplicação, duplicando registros em nosso banco de dados.

Vimos também como podemos enviar mensagens de uma requisição para outra, havendo redirecionamento de páginas com o `Flash`, usando o `RedirectAttributes` e o método `addFlashAttribute`.

TIRAR DÚVIDA
PRÓXIMA ATIVIDADE