02

Finalizando o carrinho

Transcrição

Nosso carrinho de compras já funciona. Além de listar os produtos escolhidos, ele mostrará o total da soma dos preços dos produtos. O que falta fazer para que a aplicação fique completa é que ao clicar no botão de finalização da compra, a compra seja registrada.

Já que a compra é algo especifico e diferente da lógica do carrinho, criaremos um novo *controller* para podermos implementar tal funcionalidade. Crie uma nova classe chamada `PagamentoController` e a defina como *controller* assinando a mesma com a anotação `@Controller`. Defina também o caminho que o *controller* irá atender através da anotação `@RequestMapping` com o valor `/pagamento`.

```
@RequestMapping("/pagamento")
@Controller
public class PagamentoController {

}
```

Neste *controller* criaremos um método responsável por finalizar as compras dos usuários. Chamaremos este método de `finalizar` e ele retornará um objeto do tipo `ModelAndView` redirecionando o usuário para a listagem dos produtos ou seja `redirect:/produtos`. Este método também será anotado com a anotação `@RequestMapping` usando o valor `/finalizar`.

O método `finalizar` não precisa receber nenhum parâmetro, mas ele precisa do carrinho de compras. O carrinho de compras já está no escopo de sessão e para que o tenhamos em mãos, precisamos apenas criar um novo atributo na classe `PagamentoController` do tipo `CarrinhoCompras` anotado com `@Autowired`. Desta forma o Spring já busca a atribui o carrinho de compra em nosso *controller*. Por hora, vamos apenas imprimir o total da compra e redirecionar. Até aqui teremos:

```
@RequestMapping("/pagamento")
@Controller
public class PagamentoController {

    @Autowired
    CarrinhoCompras carrinho;
    @RequestMapping("/finalizar")
    public ModelAndView finalizar(){
        System.out.println(carrinho.getTotal());
        return new ModelAndView("redirect:/produtos");
    }
}
```

Parece estranho redirecionar o usuário e não informá-lo se o processo funcionou bem. Pensando nisto, vamos fazer uso da classe `RedirectAttributes` que chamaremos de `model` e do método `addFlashAttribute` para exibirmos uma mensagem amigável para o usuário. Nos campos de chave e valor do método `addFlashAttributes` usaremos `sucesso` e `Pagamento Realizado com Sucesso` respectivamente.

```
model.addFlashAttribute("sucesso", "Pagamento Realizado com Sucesso");
```

Com as alterações, o nosso código ficará assim:

```
@RequestMapping("/pagamento")
@Controller
public class PagamentoController {

    @Autowired
    CarrinhoCompras carrinho;

    @RequestMapping(value="/finalizar", method=RequestMethod.POST)
    public ModelAndView finalizar(RedirectAttributes model){
        System.out.println(carrinho.getTotal());
        model.addFlashAttribute("sucesso", "Pagamento Realizado com Sucesso");

        return new ModelAndView("redirect:/produtos");
    }
}
```

Usamos como chave da mensagem o valor `sucesso`, porque na página de listagem dos produtos em `lista.jsp` já há um código responsável por exibir mensagens que venham com esta mesma chave. Lá deve haver uma linha código como esta:

```
<p> ${sucesso} </p>
```

Agora queremos que ao clicar no botão de finalizar a compra, a requisição seja enviada para o `PagamentoController` e que o método `finaliza` seja chamado. Veja como está o botão de finalizar compra atualmente:

```
<td colspan="3">
    <input type="submit" class="checkout" name="checkout" value="Finalizar compra" />
</td>
```

O botão é um `input`. Por questões de prática, vamos envolver este `input` com a tag `form`. Vamos definir o `action` deste formulário com a ajuda das tags do **Spring** (`s:mvcUrl`) e ajustar o atributo `method` para que seja post. Desta forma teremos:

```
<form action="${s:mvcUrl('PC#finalizar').build()}" method="post">
    <input type="submit" class="checkout" name="checkout" value="Finalizar compra" />
</form>
```

Já que estamos usando `post` como médoto no formulário, vamos fazer com que o método `finalizar` receba apenas requisições `post`, pois não faz muito sentido recebermos uma requisição do tipo `get` neste contexto. Ajuste a assinatuda do médoto `finalizar` para que esta fique da seguinte forma:

```
@RequestMapping(value="/finalizar", method=RequestMethod.POST)
public ModelAndView finalizar(RedirectAttributes model){
    System.out.println(carrinho.getTotal());
```

```
        model.addFlashAttribute("sucesso", "Pagamento Realizado com Sucesso");

        return new ModelAndView("redirect:/produtos");
    }
```

Assim já poderemos fazer os testes necessários para saber se tudo que fizemos até o momento funciona. Inicie então o servidor e tente efetuar uma compra. Adicione alguns produtos ao carrinho e clique em `Finalizar Compra`.

Ao iniciar nossa aplicação, teremos uma mensagem de erro:

```
Error creating bean with name 'carrinhoCompras': Scope 'session' is not active for the current t
```

O **Spring** não conseguiu atribuir o carrinho de compras da sessão ao `PagamentoController`. Já vimos este problema antes na classe `CarrinhoComprasController` e resolvemos usando a anotação que muda o escopo do ***controller***. Desta vez, resolveremos este problema de uma outra forma.

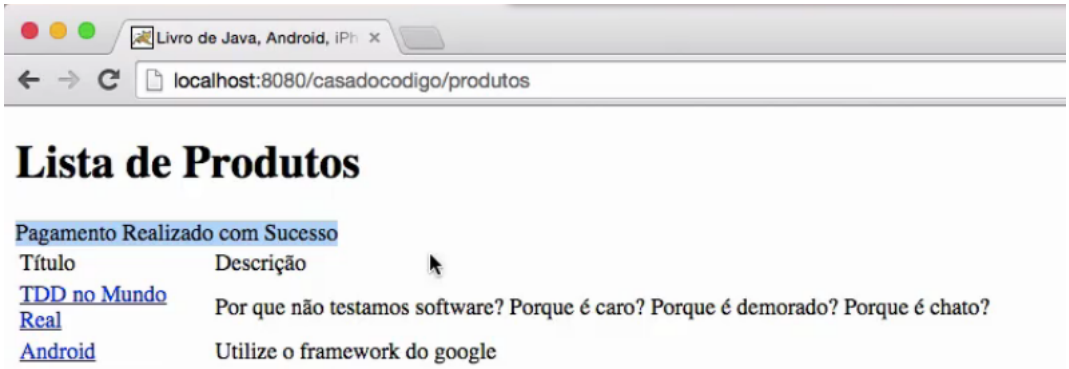
Na classe `CarrinhoCompras`, usamos a anotação:

```
@Scope(value=WebApplicationContext.SCOPE_SESSION)
```

Passaremos para esta mesma anotação uma segunda informação definindo que o **Spring** criará um ***proxy*** envolvendo o objeto alvo (`TARGET_CLASS`) afim de possibilitar que as informações possam ser transitadas de um escopo para o outro. A anotação com a modificação proposta fica da seguinte forma:

```
@Scope(value=WebApplicationContext.SCOPE_SESSION, proxyMode=ScopedProxyMode.TARGET_CLASS).
```

Assim conseguiremos fazer com que o carrinho de compras seja acessivel dentro dos ***controllers*** sem ter que mudar o escopo dos ***controllers*** para isso. Reinicie então o servidor e experimente fazer algumas compras.



Nosso próximo passo será fazer com que o link de remoção de produtos do carrinho funcione. Começaremos adicionando o `action` correto para esta finalidade na página `itens.jsp`. Nesta página procure pelo link de remoção que deve estar da seguinte forma:

```
<td class="remove-item">
    <form action="" method="post">
        <input type="image" src="/excluir.png" alt="Excluir" title="Excluir" />
```

```
    </form>
</td>
```

Usaremos novamente as tags do **Spring** para esta tarefa. No `mvcUrl` criaremos uma url que aponte para o `CarrinhoComprasController` e chame o método `remover`. Este método precisará do `id` e o tipo de preço do produto, então o passaremos através do `arg` da seguinte forma:

```
<td class="remove-item">
    <form action="${s:mvcUrl('CCC#remover').arg(0, item.produto.id).arg(1,item.tipoPreco).build
        <input type="image" src="/excluir.png" alt="Excluir" title="Excluir" />
    </form>
</td>
```

Na classe `CarrinhoComprasController` precisamos adicionar o método `remover` que fará simplesmente uma chamada ao método `remover` da classe `CarrinhoCompras` passando o `id` do produto recebido e o tipo de preço do produto. Após isso, o método `remover` da classe `CarrinhoComprasController` criaremos um objeto do tipo `ModelAndView` para redirecionar o usuário para seu carrinho de compras. Em código teremos:

```
@RequestMapping("/remover")
public ModelAndView remover(Integer produtoId, TipoPreco tipoPreco){
    carrinho.remover(produtoId, tipoPreco);
    return new ModelAndView("redirect:/carrinho");
}
```

Observação: Note que estamos fazendo o mapeamento do método com a anotação `@RequestMapping` com o valor `/remover`.

Para finalizar esta funcionalidade, precisamos fazer com que a classe `CarrinhoCompras` remova o produto da lista de itens. Lembre-se que os itens desta lista são objetos da classe `CarrinhoItem`. Sendo assim, precisamos criar um objeto `CarrinhoItem` que precisa de um objeto `Produto` e em `TipoPreco`.

O objeto `TipoPreco` já esta sendo passado por parâmetro, mas o produto não. Estamos recebendo apenas o `id` do produto. Como fazer então? Poderíamos buscar este produto no banco de dados, mas na verdade, neste ponto, não precisamos disso. Neste caso em especifico, podemos criar um novo objeto `produto` e usar o método `setId` para definir o `id` do produto com o `id` recebido por parâmetro. Desta forma criamos um `carrinhoItem` e usamos o método `remove` do objeto `itens` na classe `CarrinhoCompras` passando este `carrinhoItem` criado. Vejamos como fica em código:

```
public void remover(Integer produtoId, TipoPreco tipoPreco) {
    Produto produto = new Produto();
    produto.setId(produtoId);
    itens.remove(new CarrinhoItem(produto, tipoPreco));
}
```

Observação: Lembre-se de que este método `remover` é para ser escrito na classe `CarrinhoCompras`.

Faça alguns testes para verificar que tudo está funcionando. Reinicie o servidor, vá até lista de produtos, adicione alguns no carrinho de compras, remova alguns e finalize a compra. Tudo deve estar funcionando sem problemas.