03

URLs amigáveis

Transcrição

Nesta aula, criaremos a página de "detalhes" dos nossos livros. Se entrarmos no site da Casa do Código (http://www.casadocodigo.com.br) e clicarmos em qualquer um dos livros, veremos outra página com todos os detalhes do produto, como por exemplo: a descrição, a quantidade de páginas, sumário etc.

Diversas dessas informações estão sendo gravadas no banco de dados durante o cadastro do produto. Então, vamos utilizar a página da Casa do Código (http://www.casadocodigo.com.br) como modelo e criar nossa página de detalhes.

Observação: Os arquivos da página de modelo que usaremos estão disponíveis para download (https://s3.amazonaws.com/caelum-online-public/spring-mvc-1-criando-aplicacoes-web/detalhe-com-css.zip)*nos exercício. Usaremos o código disponibilizada como referência nesta aula.

Faremos na página de "detalhes" uma série de modificações para que esta passe a exibir os detalhes dos nossos produtos. A página se encontrará no arquivo `detalhe.jsp`. A primeira mudança que faremos neste arquivo é referente ao título da páginas. Veja como está agora:

```
<title>Livros de Java, SOA, Android, iPhone, Ruby on Rails e muito mais - Casa do Código</title>
```

Como a página deve exibir os detalhes de um determinado produto. Mudaremos o título de acordo com o livro escolhido. Substituiremos então a parte do título da página que tem o seguinte texto: `Livros de Java, SOA, Android, iPhone, Ruby on Rails e muito mais` para `${produto.titulo}`.

```
<title>${produto.titulo} - Casa do Código</title>
```

Na tag `body` não há nenhuma classe de estilo. Vamos usar uma chamada `produto` para esclarecer que se refere à página de um produto especifico. Assim também poderemos aproveitar os `CSS` da Casa do Código (http://www.casadocodigo.com.br). A tag `body` ficará dessa forma: `<body class="produto">`.

Note que o código fonte da página está todo em Inglês. Por questões de praticidade, modificaremos algumas partes para usarmos o idioma Português. Busque a parte do código referente ao carrinho de compras, que estará dentro da tag `header`:

```
<header id="layout-header">
       <div class="clearfix container">
           <a href="/" id="logo">
           </a>
           <div id="header-content">
               <nav id="main-nav">

                   <ul class="clearfix">
                       <li><a href="/cart" rel="nofollow">Carrinho</a></li>

                       <li><a href="/pages/sobre-a-casa-do-codigo" rel="nofollow">Sobre Nós</a>
```

```
                        <li><a href="/pages/perguntas-frequentes" rel="nofollow">Perguntas Frequ
                    </ul>
                </nav>
            </div>
        </div>
    </header>
```

Nós iremos trabalhar com a seguinte `li` :

```
<li><a href="/cart" rel="nofollow">Carrinho</a></li>.
```

Vamos mudar o `/cart` para `/carrinho` . O código resultante será:

```
<li><a href="/carrinho" rel="nofollow">Carrinho</a></li>
```

Não vamos nos preocupar com as `urls` neste momento, faremos elas funcionarem logo em seguida.

A tag `article` tem o atributo `id` , gerado dinamicamente pelo sistema da Casa do Código (http://www.casadocodigo.com.br).

```
<article id="livro-java8" itemscope intemtype="http://schema.org/Book">
          <header id="product-highlight" class="clearfix">
            <div id="product-overview" class="container">
              <img itemprop="image" width="280px" height="395px" src="http://cdn.shopify.com/s/f
              <h1 class="product-title" itemprop="name">Java 8 Prático: Lambdas, Streams e os no

              \...
```

Vamos fazer uma adaptação para que seja usado o `id` do produto exibido.

Iremos apagar o seguinte trecho na primeira linha

```
itemscope intemtype="http://schema.org/Book
```

E da mesma forma que fizemos no título da página, faremos o seguinte: `${produto.id}` .

A tag `article` ficará assim:

```
<article id="${produto.id}">
```

No arquivo disponibilizado da Casa do Código, na tag `h1` dentro da tag `article` faremos alterações para o texto `AQUI COLOQUE O TÍTULO` .

```
\...
  <h1 class="product-title">AQUI COLOQUE O TÍTULO</h1>
              <p class="product-author">
```

```
                <span class="product-author-link">

                </span>
              </p>
```

No lugar, iremo substituir pelo nome do produto. Já sabemos como fazer, certo? Podemos usar `${produto.titulo}`. A tag `h1` ficará assim:

```
<h1 class="product-title">${produto.titulo}</h1>
```

Onde temos o seguinte código:

```
<p class="book-description">
    AQUI COLOQUE A DESCRIÇÃO
</p>
```

Vamos trocar o texto `AQUI COLOQUE A DESCRIÇÃO` pela real descrição do livro através do produto: `${produto.descricao}`.

```
<p class="book-description">
    ${produto.descricao}
</p>
```

O código ficará assim:

```
<header id="product-highlight" class="clearfix">
   <div id="product-overview" class="container">
    <img itemprop="image" width="280px" height="395px" src="http://cdn.shopify.com/s/files/1/01

    <h1 class="product-title" itemprop="name">
      ${produto.titulo}
    </h1>
    <p class="product-author">
       <span class="product-author-link">

   </span>
</p>

<p itemprop="description" class="book-description">${produto.descricao }</p>

    </div>
</header>
```

Um pouco mais abaixo deste código encontraremos a parte do código `HTML` referente a compra dos livros. A parte do `HTML` que exibe os preços dos livros. As opções de preço são envolvidas pela tag `li`. Veja o código atual:

```
<li class="buy-option">
    <input type="radio" name="id" class="variant-radio" id="product-variant-9720393823" value="
    <label  class="variant-label" for="product-variant-9720393823">
      E-book + Impresso
```

```
    </label>
    <small class="compare-at-price">R$ 39,90</small>
    <p class="variant-price">R$ 29,90</p>
</li>
```

No cadastro de produtos, deixamos disponível três opções de preços: Ebook, Impresso e Combo. Vamos aproveitar as facilidades da **Expression Language** e fazer um `forEach` para repetir o código de opção de compra para as três opções que temos. Faremos uso da tag `forEach` da **JSTL**. O código deste `forEach` ficará dessa forma:

```
<c:forEach items="" var="preco">
    <li class="buy-option">
        <input type="radio" name="id" class="variant-radio" id="product-variant-9720393823" valu
        <label  class="variant-label" for="product-variant-9720393823">
          E-book + Impresso
        </label>
        <small class="compare-at-price">R$ 39,90</small>
        <p class="variant-price">R$ 29,90</p>
    </li>
</c:forEach>
```

Note que a tag `li` que estamos modificando está dentro de um formulário. Ou seja, envolvido por uma tag `form`. Cada `li` no código da se refere a uma variante do produto, que podem ser: Ebook, Impresso ou Combo. Na Casa do Código (http://www.casadocodigo.com.br) cada uma dessas variações tem um `id` próprio, mas em nosso caso não. Temos variações de preços, mas não de produtos.

Sendo assim, iremos remover o valor do atributo `id` do `input type="radio"` fazendo com que o código fique desta forma:

```
<c:forEach items="" var="preco">
    <li class="buy-option">
        [...]
        <input type="radio" name="id" class="variant-radio" id="" value="9720393823"  checked="
        [...]
    </li>
</c:forEach>
```

Dessa forma, perdemos a referência para o produto a ser adicionado ao carrinho de compras do nosso sistema. Este formulário que estamos modificando é o que adiciona produtos no carrinho. Precisamos de alguma referência do produto a ser adicionado no carrinho. Vamos criar um novo `input`, mas desta vez sendo do tipo `hidden`, configurando o valor desse `input` para ser o `id` do produto e o `name` como sendo `produtoId`. Este código deve vir antes do `forEach`.

```
<input type="hidden" value="${produto.id}" value="produtoId" />
```

Agora que temos uma referência ao produto, precisamos deixar disponível também uma referência ao tipo de preço a ser escolhido pelo usuário na hora da compra. Modificamos anteriormente o `input type="radio"` no qual o usuário selecionava uma variante do produto. No nosso caso, temos variantes de preço. Vamos adaptar isso também.

O `input type="radio"` agora deve referenciar os tipos de preços que temos em nosso sistema. seu atributo `name` então deve ter valor definido como `tipo` assim como seu `id` . E o atributo `value` deve ter seu valor definido como o tipo de preço do produto. Algo como: `${preco.tipo}` .

No `label` que se encotra logo abaixo do `input type="radio"` deve ser exibido o tipo de preço. Então em vez de:

```
<label  class="variant-label" for="product-variant-9720393823">
  E-book + Impresso
</label>
```

Teremos:

```
<label  class="variant-label">
    ${preco.tipo}
</label>
```

Removemos o atributo `for` porque não precisamos dele e também para deixar o código um pouco mais simples.A tag `p` abaixo do `label` exibe o valor do produto, o preco! Até aqui só exibimos informações referentes ao tipo do preço. Não podemos esquecer do valor. Exibiremos então o valor do preço atravéz do código: `${preco.valor}` . A tag `p` que se encontra assim:

```
<p class="variant-price">R$ 29,90</p>
```

Ficará assim:

```
<p class="variant-price">${preco.valor}</p>
```

A última mudança que precisamos fazer no formulário de adição produtos no carrinho é referente ao atributo `title` do botão no final do formulário. Este botão é do tipo `submit` , responsável por fazer o envio dos dados do formulário. O botão contém o atributo `title` da seguinte forma: `title="Compre Agora"` .

```
<button type="submit" class="submit-image icon-basket-alt" alt="Compre Agora" title="Compre Ag
```

Vamos compor este título com o nome do livro, para ficar mais intuitivo. Desta forma o atributo do botão ficará da seguinte forma:

```
title="Compre Agora ${produto.titulo}!"
```

Observação: Não esqueça de atualizar os outros pontos do código que se referem ao carrinho de compras. Modificamos o `/cart` para `/carrinho` lembra? Essa mudança também ocorre na `action` do formulário que estava assim:

```
<form action="/cart/add" method="post" class="container">
```

E ficou assim:

```
<form action="/carrinho/add" method="post" class="container">
```

O código completo do formulário com todas as modificações que fizemos até aqui ficará da seguinte forma:

```
<form action="/carrinho/add" method="post" class="container">
   <ul id="variants" class="clearfix">
       <input type="hidden" name="produtoId" value="${produto.id}" />
       <c:forEach items="" var="preco">
         <li class="buy-option">
           <input type="radio" name="tipo" class="variant-radio" id="tipo" value="${preco.tipo
           <label  class="variant-label">
             ${preco.tipo}
           </label>
           <small class="compare-at-price">R$ 39,90</small>
           <p class="variant-price">${produto.titulo}</p>
         </li>
       </c:forEach>
   </ul>
   <button type="submit" class="submit-image icon-basket-alt" alt="Compre Agora" title="Compre
 </form>
```

Não se preocupe com as partes que não modificamos. Estamos adaptando o código as nossas necessidades e ainda vamos continuar modificando nos próximos passos.

Antes do final da tag `article` existe uma tag `section` que deve exibir algumas informações que também são importantes de serem mostradas. Como o número de páginas e a data de publicação. Veja o código dessa seção abaixo:

```
<section class="data product-detail">
    <h2 class="section-title">Dados do livro:</h2>
    <p>Número de páginas: <span>AQUI O NÚMERO DE PÁGINAS</span></p>
    <p></p>
    <p>Data de publicação: AQUI A DATA DE PUBLICAÇÃO </p>
    <p>Encontrou um erro? <a href='/submissao-errata' target='_blank'>Submeta uma errata</a></p
</section>
```

Vamos exibir o número de páginas através de: `${produto.paginas}` e a data de lançamento da seguinte forma: `${produto.dataLancamento}` . Depois dessas mudanças o código da seção ficará assim:

```
<section class="data product-detail">
    <h2 class="section-title">Dados do livro:</h2>
    <p>Número de páginas: <span>${produto.paginas}</span></p>
    <p></p>
    <p>Data de publicação: ${produto.dataLancamento}</p>
    <p>Encontrou um erro? <a href='/submissao-errata' target='_blank'>Submeta uma errata</a></p
</section>
```

Nossa página de detalhe com todas as modificações que fizemos até aqui está pronta. Precisamos agora criar um novo método na classe `ProdutosController` que faça a seguinte tarefa: Buscar um produto usando o `id` e deixar este produto disponível na `view` de detalhes, ou seja, no arquivo `detalhe.jsp`.

Chamaremos este novo método de `detalhe`. Ele precisa fazer o mapeamento para a url `/detalhe` através da anotação `@RequestMapping` e receber um `id` do tipo `int`. Após isto, precisaremos criar um objeto do tipo `ModelAndView` passando para o construtor da classe do `ModelAndView` o caminho da `view` a ser utilizada. A view a ser utilizada será `detalhe.jsp`, que acabamos de criar.

No próximo passo, usaremos o objeto `produtoDao` que temos no `ProdutosController` para buscar os livros pelo `id` e atribui-lo a um outro objeto da classe `Produto`. O método que faz a busca se chamará `find`. Ainda não temos o método da busca, mas vamos deixar a parte do controller pronta. O código ficará assim:

```
@RequestMapping("/detalhe")
public ModelAndView detalhe(int id){
    ModelAndView modelAndView = new ModelAndView("/produtos/detalhe");
    Produto produto = produtoDao.find(id);
    modelAndView.addObject("produto", produto);
    return modelAndView;
}
```

Em seguida, iremos criar o método `find` na classe `ProdutoDAO` para que receba um `id` e retorne um objeto da classe `Produto`. Para esta tarefa utilizaremos o objeto `manager` da classe `ProdutoDAO` para buscar o produto através do método `find`, que requer dois parâmetros. O primeiro se refere à classe representante da entidade na qual o produto será buscado. O segundo será o próprio `id`. O valor do primeiro parâmetro será `Produto.class`. No fim, retornaremos o resultado dessa busca. O código ficará dessa forma:

```
public Produto find(int id){
    return manager.find(Produto.class, id);
}
```

A página de detalhes deve funcionar e já estamos fazendo a busca pelo produto no banco de dados. Também implementamos a lógica que busca o produto e envia o produto para a `view` no `ProdutosController`. Logo, poderemos exibir os dados do produto no arquivo `detalhe.jsp` que é a nossa `view`.

Mas como faremos para acessar a páginas de detalhes de um produto? Vamos precisar passar o `id` para que o produto possa ser buscado e exibido. Podemos tentar acessar diretamente pela `url`. O `id` pode ser passado na `url` da seguinte forma: `detalhe?id=4`. Sendo assim, podemos montar a seguinte `url` para exibir os detalhes de um determinado livro:

```
localhost:8080/casadocodigo/produtos/detalhe?id=4
```

Agora, é possível ver a página de detalhes de um livro. Alguns detalhes poderão não ser exibidos corretamente - isso é perfeitamente normal -, porque não fizemos todas as adaptações necessárias ainda. Os preços são uma das informações que não serão exibidas corretamente, considerando que não o escrevemos da forma correta. Veremos como ele esta agora:

```
<c:forEach items="" var="preco">
  <li class="buy-option">
    <input type="radio" name="tipo" class="variant-radio" id="tipo" value="${preco.tipo}"  checl
    <label  class="variant-label">
      ${preco.tipo}
    </label>
    <small class="compare-at-price">R$ 39,90</small>
    <p class="variant-price">${preco.valor}</p>
  </li>
</c:forEach>
```

Perceba que não preenchemos o atributo `items` do `forEach`. Devemos passar, para este atributo, a lista de preços presente no produto dessa forma:

```
<c:forEach items="${produto.precos}" var="preco">
  <li class="buy-option">
    <input type="radio" name="tipo" class="variant-radio" id="tipo" value="${preco.tipo}"  checl
    <label  class="variant-label">
      ${preco.tipo}
    </label>
    <small class="compare-at-price">R$ 39,90</small>
    <p class="variant-price">${preco.valor}</p>
  </li>
</c:forEach>
```

Se atualizamos a página agora, teremos um erro:

```
org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: br.com.casadocodigo.loja.models.Produto.precos, could not initialize proxy
        org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:572)
        org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:212)
        org.hibernate.collection.internal.AbstractPersistentCollection.initialize(AbstractPersistentCollection.java:551)
        org.hibernate.collection.internal.AbstractPersistentCollection.read(AbstractPersistentCollection.java:140)
```

Este erro de `LazyInitializationException` indica que ao carregar o produto, os preços não foram carregados juntos. Ou seja, tentamos exibir os preços sem carregá-los do banco de dados. Isto acontece porque nosso `ProdutoDAO` no método `find` só busca o produto, sem se relacionar com seus preços.

Precisamos então fazer com que o **Hibernate** relacione os produtos com seus preços. Faremos isto através de uma *query* personalizada:

```
select distinct(p) from Produto p join fetch p.precos precos where p.id = :id
```

Observe que estamos fazendo um `select` comum, mas estamos usando o `distinct` para que o **Hibernate** nos retorne apenas resultados diferentes. Estamos também fazendo um `join fetch` com a tabela `Precos` usando como relação o `id` do produto presente na tabela de preços.

Usaremos o `sql` através do método `createQuery` e passaremos o `:id` através do método `setParameter`. Queremos retornar apenas um resultado desta *query* através do método `getSingleResult`. Nosso método `find` ficará assim:

```
return manager.createQuery("select distinct(p) from Produto p join fetch p.precos precos where
```

Para que não precisemos ficar chutando `ids` diretamente na `url`. Criaremos links na listagem e assim poderemos clicar em um determinado produto e já visualizar a página de detalhes do produto. Faremos isso no arquivo `lista.jsp` usando o construtor de `urls` do **Spring**. Lembra do `mvcUrl`? Iremos usa-lo novamente aqui.

Passaremos o `id` para o `mvcUrl` através do método `arg` que recebe dois parâmetros: o primeiro será a posição do parametro, que terá valor `0`. O segundo será o valor do parâmetro, o `produto.id`.

```
<a href="${s:mvcUrl('PC#detalhe').arg(0, produto.id).build()}">${produto.titulo}</a>
```

Observação: Estamos construindo o link para a página de detalhes do produto na listagem dos produto. Este link deve ser criado de forma a envolver o título do livro na listagem. Lembre-se de importar a `taglib` do **Spring** para poder usar o `mvcUrl` da mesma forma que fizemos no `form.jsp` usando o ***import***: `<%@ taglib uri="http://www.springframework.org/tags" prefix="s" %>`.

Mas veja como está ficando nossa `url`: `http://localhost:8080/casadocodigo/produtos/detalhe?id=2`.

Observe como são as `urls` do site da Casa do Código (http://www.casadocodigo.com.br):

```
https://www.casadocodigo.com.br/products/livro-code-igniter
```

Elas parecem mais interessantes, porque mostram o nome do produto. Estas `urls` são chamadas de **URLs Amigaveis**.

Nós não mudaremos nossas `urls` para ficarem amigaveis desta forma, mas faremos com que o parâmetro seja passado de forma diferente. Veja como nossa `url` parece mais simples usando o seguinte formato:

```
http://localhost:8080/casadocodigo/produtos/detalhe/2
```

Eliminaremos o `?id=` e deixaremos nossas `urls` mais simples.

Para isto, precisamos mudar a assinatura do método `detalhe` em nosso `ProdutosController`. Ela precisa receber o parâmetro separado pela barra (`/`). A anotação `@RequestMapping` permite que façamos isso da seguinte forma:

```
@RequestMapping("/detalhe/{id}")
```

Mas apenas isto não é o suficiente. Precisaremos indicar para o método `detalhe` que o parâmetro `id` será recuperado do caminho da `url`. Então, usaremos uma nova anotação: `@PathVariable` passando o `id` desta forma: `@PathVariable("id")`.

Com as alterações o método `detalhe` da classe `ProdutosController` ficará assim:

```
@RequestMapping("/detalhe/{id}")
public ModelAndView detalhe(@PathVariable("id") Integer id){
    ModelAndView modelAndView = new ModelAndView("/produtos/detalhe");
    Produto produto = produtoDao.find(id);
    modelAndView.addObject("produto", produto);
    return modelAndView;
}
```

Algumas coisas ainda não estão prontas - nossas datas ainda não são exibidas corretamente. Mas corrigiremos isso em breve.

Por enquanto, faremos alguns exercícios para fixar o que vimos até aqui. Mas já aprendemos como relacionar os produtos com seus preços, criar a página de detalhes do produto e configurar as `urls` para ficarem amigáveis.