CETEJ35 - Java Web - JAVA_XXX (2024_01)

Criando, Alterando e Excluindo

✓ **Feito:** Ver ✓ **Feito:** Gastar pelo menos 20 minutos na atividade ✓ **Feito:** Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Nesta aula nós finalizamos a implementação das quatro operações CRUD. Isso significa que nosso usuário será capaz de criar, alterar, excluir e listar as cidades em uma base de dados. Observe que ainda estamos usando uma base local, baseada em uma lista em memória. Nós vamos evoluir esse projeto até integrarmos essa base com um banco de dados.

ALTERANDO

Deixamos a alteração por último porque ela é a mais complexa das quatro operações. Primeiro, é necessário carregar os dados na tela. Segundo, a tela precisa ser adaptada para alterar em vez de criar. Em seguida, os dados precisam ser enviados para que o método no controlador encontre os dados na base, recupere esses dados, faça a atualização e recarregue a página com os novos dados. Vamos fazer um passo de cada vez nesta seção.

Expanda cada um dos itens abaixo para entender os procedimentos de implementação necessários.

▾ 1. Carregando dados na tela

Vamos começar inserindo comportamento no botão `ALTERAR`, na página `crud.ftl`. Para isso, siga o mesmo procedimento usado no botão `EXCLUIR`, conforme pode ser observado na linha 47, na Figura a seguir.

```
41              <#list listaCidades as cidade >
42                  <tr>
43                      <td>${cidade.nome}</td>
44                      <td>${cidade.estado}</td>
45                      <td>
46                          <div class="d-flex d-justify-content-center">
47                          <a href="/preparaAlterar?nome=${cidade.nome}&estado=${cidade.estado}" class="btn btn-warning mr-3">ALTERAR</a>
48                          <a href="/excluir?nome=${cidade.nome}&estado=${cidade.estado}" class="btn btn-danger">EXCLUIR</a>
49                          </div>
50                      </td>
51                  </tr>
52              </#list>
```

A propriedade `href` identifica para onde a solicitação será enviada. Nesse caso, estamos considerando uma URL `/preparaAlterar`. Essa mesma URL precisa ser usada quando fizermos o mapeamento do método no controlador. Assim como no botão `EXCLUIR`, precisamos enviar o nome e estado para que a cidade seja encontrada. Ambos valores são enviados como parâmetros. Esses valores são extraídos da lista de cidades usando a sintaxe de interpolação do Freemarker (`${}`).

Em seguida, precisamos criar o método que será responsável por tratar essa solicitação na classe `CidadeController`. Abra a classe e crie um método chamado `preparaAlterar()`. Assim como no método `excluir()`, o método preparaAlterar recebe dois parâmetros, representando o nome da cidade e do estado.

O método `preparaAlterar()` deve recuperar a cidade cujo nome e estado foram informados pelo parâmetro, colocar esses valores recuperados na memória da solicitação e recarregar a página. Para colocar os valores na memória da solicitação, vamos precisar de mais um parâmetro do tipo `org.springframework.ui.Model`. Veja como ficou a assinatura do método `preparaAlterar()`.

```
49      public String preparaAlterar(
50          @RequestParam String nome,
51          @RequestParam String estado,
52          Model memoria) {
```

Agora, o método precisa recuperar os dados e coloca-los na memoria. Vamos usar o atributo `cidades` para ter acesso à lista de cidades. Vamos filtrar apenas a cidade cujo nome e estado é igual aos valores passados como parâmetro. Como vamos ter apenas uma combinação de cidade/estado, então podemos usar o método `findAny()` para recuperar um objeto `java.util.Optional`, que encapsula a cidade buscada. O código da busca fica conforme a Figura abaixo.

> Nesse curso fazemos uso massivo dos recursos introduzidos desde o Java 8. Se você sente dificuldade em usar esses recursos, dê uma olhada **aqui**.

```
55              var cidadeAtual = cidades
56                      .stream()
57                      .filter(cidade ->
58                          cidade.getNome().equals(nome) &&
59                          cidade.getEstado().equals(estado))
60                      .findAny();
```

Se existir uma cidade com os valores buscados, ela será armazenada em um objeto do tipo `Optional`, acessível pela variável `cidadeAtual`. O último passo aqui é verificar se o `cidadeAtual` não está vazia. Nesse caso, precisamos adicionar a `cidadeAtual` na memória da solicitação.

> Teoricamente, sempre existirá uma cidade com os valores buscados porque a URL é montada de acordo com os dados na tabela. Contudo, alguém pode tentar acessar o aplicativo digitando a URL diretamente na barra de endereços do navegador. Para evitar problemas, faz sentido usar a validação.

```
55              var cidadeAtual = cidades
56                      .stream()
57                      .filter(cidade ->
58                          cidade.getNome().equals(nome) &&
59                          cidade.getEstado().equals(estado))
60                      .findAny();
61
62              if (cidadeAtual.isPresent()) {
63                  memoria.addAttribute("cidadeAtual", cidadeAtual.get());
64                  memoria.addAttribute("listaCidades", cidades);
65              }
```

Observe que a linha 64 também adiciona a lista de cidades na memória. Precisamos disso porque esse método não fará um redirecionamento. Esse método vai carregar a página diretamente. Por isso, se quisermos ver a lista de cidades, precisamos coloca-la na memoria da solicitação. O último passo no controlador é retornar o nome da página que será carregada e adicionar a anotação que faz o mapeamento com a URL de alteração. Veja como ficou o código completo.

```java
@GetMapping("/preparaAlterar")
public String preparaAlterar(
    @RequestParam String nome,
    @RequestParam String estado,
    Model memoria) {

        var cidadeAtual = cidades
                .stream()
                .filter(cidade ->
                    cidade.getNome().equals(nome) &&
                    cidade.getEstado().equals(estado))
                .findAny();

        if (cidadeAtual.isPresent()) {
            memoria.addAttribute("cidadeAtual", cidadeAtual.get());
            memoria.addAttribute("listaCidades", cidades);
        }

        return "/crud";
}
```

Por que não redirecionar? Por que cada solicitação tem um tempo de vida, lembra? Ela só existe em um caminho: o de ida, ou de volta. Nesse caso, se fizermos um redirecionamento, o valor de `cidadeAtual` será perdido.

O último passo para carregar os dados na tela é atualizar a página `crud.ftl` para que ela exiba os dados de cidadeAtual se ela existir na memória. Para isso, vamos usar o atributo `value` da `input` para definir os dados do campo. Também usamos uma sintaxe especial do Freemarker que só exibe valores caso eles existam. Veja como ficou o código dos nomes da cidade e do estado na página `crud.ftl`.

```html
<div class="form-group">
    <label for="nome">Cidade:</label>
    <input value="${(cidadeAtual.nome)!}" name="nome" type="text" class="form-control"
    placeholder="Informe o nome da cidade" id="nome">
</div>
<div class="form-group">
    <label for="estado">Estado:</label>
    <input value="${(cidadeAtual.estado)!}" name="estado" type="text" class="form-control"
    placeholder="Informe o estado ao qual a cidade pertence"
    id="estado">
</div>
```

Nesse ponto, você já pode executar o projeto, criar uma cidade e solicitar a alteração clicando no botão `ALTERAR`. Você verá a página inicial sendo recarregada com os valores da cidade que você escolheu. Agora, precisamos alterar e enviar os novos valores para alteração.

▾ 2. Adaptando tela para alterar em vez de criar

Se você tentou fazer um teste após a alterar os dados e clicou no botão `CRIAR`, vai perceber que uma nova cidade foi criada com os dados que você inseriu. Não é bem isso que esperamos. Na verdade, o que queremos é alterar os dados e não inserir uma nova cidade. Por isso, precisamos que, no momento em que os dados são carregados, o formulário também seja atualizado para alterar em vez de criar.

Para isso, vamos começar usando a diretiva `<#if>/<#else>` do Freemarker. Essas diretivas permitem tomar decisões com base em algum valor, da mesma forma que um `if/else` em Java.

Duas partes do formulário precisam ser alteradas. A primeira é a parte que define para onde os dados serão enviados. A segunda é o rótulo do botão. Para alterar a URL vamos criar uma estrutura condicional que verifica se a variável `cidadeAtual` existe na memória da solicitação. Se existir, então estamos *alterando* a cidade. Senão, significa que estamos *criando* uma nova cidade.

Contudo, para que alteração seja bem sucedida, também precisamos enviar os valores atuais. Os valores atuais são necessários para o controlador localize os dados na lista de cidades e faça a alteração. Nesse ponto, talvez você esteja pensando: "Já não fizemos isso?". Sim, fizemos, mas isso foi feito para recuperar os valores e mostra-los na página. Agora, vamos efetivamente realizar a alteração. Também, é importante lembrar que os dados que estão na memória da solicitação são resetados toda vez que uma nova solicitação é feita.

Para enviar os dados atuais, vamos usar um atributo escondido no formulário. Esse atributo usa os valores de `cidadeAtual` para identificar qual cidade deve ser alterada. Veja como ficou o código de envio do formulário com essas alterações.

```
20
21          <#if cidadeAtual??>
22              <form action="/alterar" method="POST">
23              <input type="hidden" name="nomeAtual" value="${(cidadeAtual.nome)!}"/>
24              <input type="hidden" name="estadoAtual" value="${(cidadeAtual.estado)!}"/>
25          <#else>
26              <form action="/criar" method="POST">
27          </#if>
28
```

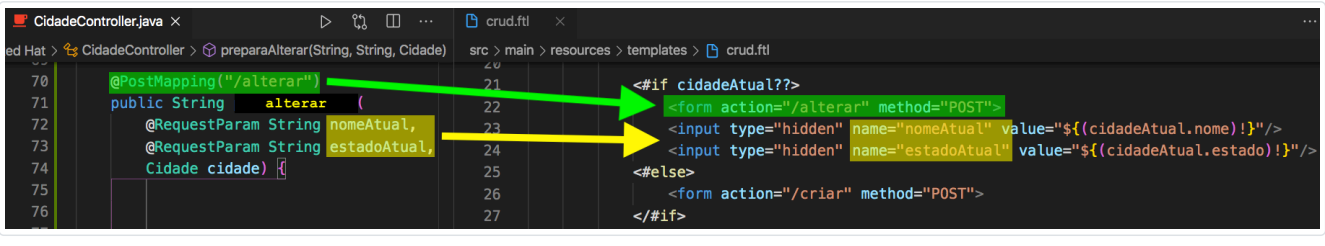
A segunda parte do formulário que precisa ser alterada é o botão que envia o formulário. Basicamente, o que muda é o rótulo. A estratégia adotada é a mesma usada para alterar o envio do formulário.

```
39          <#if cidadeAtual??>
40              <button type="submit" class="btn btn-warning">CONCLUIR ALTERAÇÃO</button>
41          <#else>
42              <button type="submit" class="btn btn-primary">CRIAR</button>
43          </#if>
44
```

▾ 3. Enviando dados para controlador

Esse procedimento envolve criar um novo método na classe `CidadeController`, mapear esse método com a URL usada na página `crud.ftl` para concluir a alteração, e alterar os dados na lista de cidades.

Vamos criar um método chamado `alterar()` na classe `CidadeController`. Esse método deve receber três parâmetros. Os dois primeiros representam os nomes atuais da cidade e do estado que devem ser alterados. O terceiro valor representa os novos valores, que são recebidos da mesma forma como acontece no método `criar()`. Também precisamos mapear o método com a URL de alteração enviada pelo `crud.ftl`.



O corpo do método é simples. Primeiro, precisamos remover a cidade atual da mesma forma que fizemos no método `excluir()`. Por que remover? Porque cada cidade é imutável, lembra? Não conseguimos alterar os dados de uma cidade existente. Por isso, vamos remover e inserir uma nova cidade com os dados novos.

Observe que podemos reutilizar o método `criar()`. Isso porque o método criar simplesmente insere novos valores e redireciona para a página inicial. Isso é exatamente o que queremos. Veja como ficou o código completo do método `alterar()`.

```
70      @PostMapping("/alterar")
71      public String alterar(
72          @RequestParam String nomeAtual,
73          @RequestParam String estadoAtual,
74          Cidade cidade) {
75
76              cidades.removeIf(cidadeAtual ->
77                      cidadeAtual.getNome().equals(nomeAtual) &&
78                      cidadeAtual.getEstado().equals(estadoAtual));
79
80              criar(cidade);
81
82              return "redirect:/";
83      }
```

Veja que o retorno do método `alterar()` nunca é usado. Isso porque o método `criar()` já retorna a solicitação. Ainda assim, o retorno é necessário para evitar erros de compilação.

O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana03-40-crud-alterar`.

Retroceder Avançar

◄ Atividade I WebConf

Seguir para...

Verificação de aprendizado - Criando, Alterando e Excluindo ►