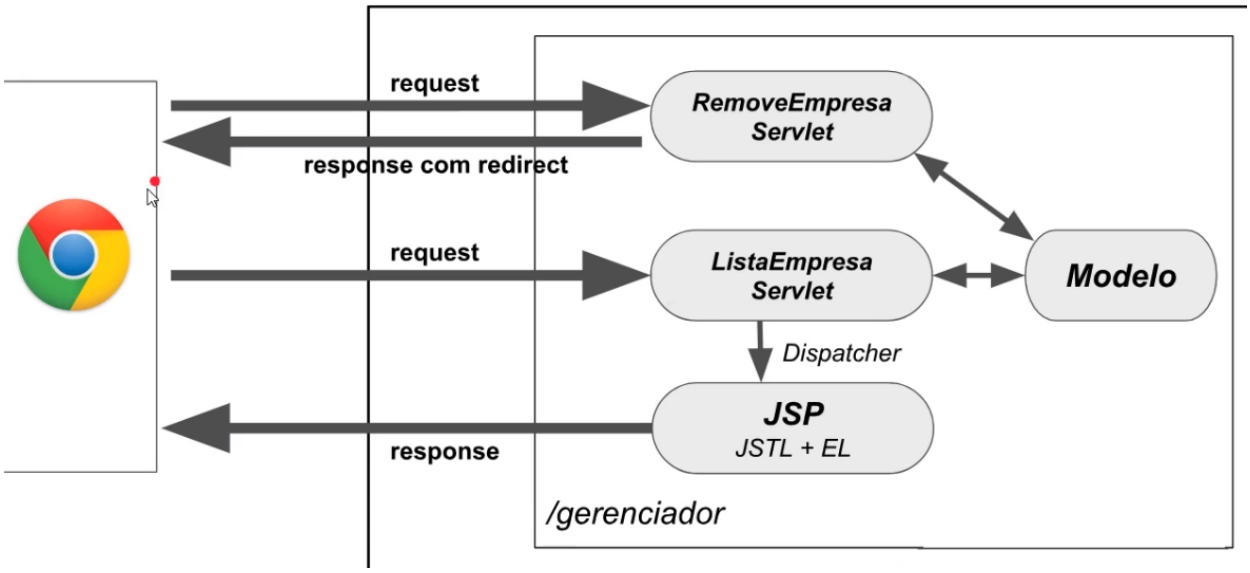
04

Removendo empresa

Transcrição

Nesta aula focaremos na função de remover uma empresa da lista de cadastro. De acordo com o fluxo do projeto, deveríamos ver no navegador a lista das empresas com as respectivas datas e um link de remoção que enviará uma requisição para o Servlet, que ainda precisa ser criado.

Esse Servlet atualizará o modelo e removerá a empresa em questão, enviando uma resposta de redirecionamento para o navegador, que automaticamente enviará uma nova requisição para listar todas as empresas novamente.



Começaremos a trabalhar em `listaEmpresas.jsp` , no qual teremos o laço que renderiza o nome e data da empresa.

```
Lista de empresas: <br />

<ul>
```

```
        <c:forEach items="${empresas}" var="empresa">

            <li>
                ${empresa.nome } - <fmt:formatDate value="${em

            </li>
        </c:forEach>
    </ul>

</body>
</html>
```

COPIAR CÓDIGO

Para criarmos um link no HTML, utilizamos o elemento `<a href=>` , e definiremos para onde realizaremos a chamada. Nesse caso, será para `/gerenciador/removeEmpresa` , sendo `removeEmpresa` como chamaremos o Servlet no mapeamento da URL. O nome do link que veremos no navegador será `remove` .

```
    Lista de empresas: <br />

    <ul>
        <c:forEach items="${empresas}" var="empresa">

            <li>
                ${empresa.nome } - <fmt:formatDate value="${em
                <a href="/gerenciador/removeEmpresa">remove</a
            </li>
        </c:forEach>
    </ul>

</body>
</html>
```

COPIAR CÓDIGO

Com isso, ao acessarmos a URL http://localhost:8080/gerenicador/listaEmpresas (http://localhost:8080/gerenicador/listaEmpresas), o link "remove" já estará visível:

> Lista de empresas:
>
> Alura - 31/08/2018 remove
>
> Caelum - 31/08/2018 remove

Clicando nesse link, receberemos uma mensagem de **erro 404**, o que faz todo sentido, pois nosso Servlet ainda não foi criado. É exatamente o que faremos agora.

Na pasta `src`, criaremos um Servlet chamado `RemoveEmpresaServlet`. Nas configurações de criação do novo Servlet, a URL Mappings será `removeEmpresa` como já havíamos definido informalmente. Usaremos, também, apenas o método `doGet`, sem o construtor.

```
package br.com.alura.gerenciador.servlet;

import java.io.IOException;

@WebServlet("/removeEmpresa")
public class RemoveEmpresaServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServle
        //TODO Auto-genarerated method stub
        repsonse.getWriter().apeend("Served at: ").apeend(requ
```

COPIAR CÓDIGO

Com essa implementação, já podemos testar o funcionamento da aplicação. É importante não esperar grandes modificações para efetuar os testes. Assim, mantemos a segurança de que cada etapa está funcionando corretamente.

De volta ao navegador, ao clicarmos no link "remove" teremos a seguinte mensagem na tela:

> Served at: /gerenciador

Já estamos sendo encaminhados para algum lugar, isto é, nosso Servlet está funcionado, ainda que não exiba nenhuma informação útil por enquanto.

Em nosso Servlet, precisamos criar uma forma de apagar uma empresa específica, ou seja, o navegador precisa nos enviar como parâmetro a informação sobre a empresa a ser removida.

Analisando `Empresa.java`, encontraremos as informações de identificação da empresa: `nome` e `id`.

```
package br.com.alura.gerenciador.servlet;

import java.util.Date;

public class Empresa {

    private Integer id;
    private String nome;
    private Date dataAbertura = new Date();
```

COPIAR CÓDIGO

Não é provável, mas é possível que duas empresas possuam o mesmo nome. Portanto, não é interessante utilizarmos o nome como diretriz de identificação para removermos uma empresa no Servlet. Normalmente, utilizamos a chav

primária, um ID, pois cada cadastro no banco de dados possui seu número sequencial ou algo do gênero.

No nosso projeto, usaremos a identificação via `id`. Esse é um procedimento muito comum na web para identificar algum objeto ou registro, de forma que o navegador informa esse id.

Ao analisarmos `Banco.java`, verificaremos que o `id` não é cadastrado automaticamente, e isso precisa ser modificado.

```
public class Banco {

    private static List<Empresa> lista = new ArrayList<>();

    static {
        Empresa empresa = new Empresa();
        empresa.setNome("Alura");
        Empresa empresa2 = new Empresa();
        empresa2.setNome("Caelum");
        lista.add(empresa);
        lista.add(empresa2);
    }

    public void adiciona(Empresa empresa) {
        Banco.lista.add(empresa);
    }

    public List<Empresa> getEmpresas(){
        return Banco.lista;
    }

}
```

COPIAR CÓDIGO

Vamos adicionar mais um atributo estático. O que faremos é simular uma chave sequencial no banco de dados, que chamaremos de `chaveSequencial`, e

que começará com `1`. Fixaremos que a empresa também deve possuir um `id`, passaremos a `chaveSequencial`, e incrementaremos essa chave utilizando a operação *post increment* `++`. Faremos esse processo para `empresa` e `empresa2`.

Por fim, a chave sequencial também deve ser mencionada no método `adiciona()`, pois quando inserimos uma nova empresa uma chave deve existir. Incluiremos o nome `Banco` da classe, para que fique bem claro que se trata de um atributo estático.

```
public class Banco {

    private static List<Empresa> lista = new ArrayList<>();
    private static Integer chaveSequencial = 1;

    static {
        Empresa empresa = new Empresa();
        empresa.setId(chaveSequencial++);
        empresa.setNome("Alura");
        Empresa empresa2 = new Empresa();
        empresa2.setId(chaveSequencial++);
        empresa2.setNome("Caelum");
        lista.add(empresa);
        lista.add(empresa2);
    }

        public void adiciona(Empresa empresa) {
        empresa.setId(Banco.chaveSequencial++);
        Banco.lista.add(empresa);
    }
```

COPIAR CÓDIGO

Dessa forma, estamos simulando que a empresa possui uma chave. Isso é muito importante, pois junto com a requisição deve existir a informação sobre a *id*. Isso é feito por meio de um parâmetro da requisição.

Já aprendemos que existem duas formas de enviar um parâmetro: dentro de uma requisição do método POST, quando possuímos um formulário, ou dentro da URL visível no navegador.

O que faremos é incluir essa informação na URL. De volta para `listaEmpresas`, inseriremos o parâmetro na URL, utilizando o `?` e logo em seguida inseriremos o parâmetro `id`, para cada empresa (`${empresa.id}`).

```
    Lista de empresas: <br />

    <ul>
        <c:forEach items="${empresas}" var="empresa">

            <li>
                ${empresa.nome } - <fmt:formatDate value="${em
                <a href="/gerenciador/removeEmpresa?id=${empre
            </li>
        </c:forEach>
    </ul>

</body>
</html>
```

COPIAR CÓDIGO

Essa é a metodologia utilizada em um projeto real. Vamos visualizar essas modificações no navegador.

A renderização não mudou nada, o link "remove" continua visível, mas agora os links estão associados a *ids* diferentes (`id=1` e `id=2`), de forma que cada empresa possui o seu *id*. Contudo, o Servlet ainda não realiza nada com esta informação, pois ainda precisamos ler o parâmetro e passar para o banco de dados o comando de remoção.

De volta ao Servlet, usaremos o `System.out.println()` para imprimir o `id` - lembrando que essa informação será impressa no lado do servidor.

```
String paramId = request getParameter("id");
Integer id = Integer.valueof(paramId);

System.out.println(id)
```

COPIAR CÓDIGO

Voltaremos ao navegador, e na lista de empresas clicaremos sobre o link "remove" da segunda empresa (com `id=2`). O parâmetro será enviado e recebido pelo Servlet, e veremos o valor `2` impresso no console.

Com esse *id* em mãos, queremos deletar a empresa associada a essa identificação. Normalmente neste ponto utilizaríamos um SQL para realizar um *delete* no banco de dados. Como estamos simulando um banco, essa medida não será necessária.

Criaremos nosso `Banco` um objeto `banco` dessa classe. Lembre-se que existe apenas uma lista de empresas, trata-se de um elemento estático da classe e não do objeto, portanto não teremos mais ou menos empresas com a criação de um novo banco.

Em seguida, acionaremos o método `removeEmpresa()`, que receberá como parâmetro o `id`. Criaremos rapidamente esse método em `Banco.java`, adicionando um laço `foreach`. Em cada interação, verificaremos o `id` da empresa, e caso esse `id` seja igual àquele especificado no parâmetro, a empresa será removida.

```
public List<Empresa> getEmpresas(){
    return Banco.lista;
}
```

```
public void removeEmpresa(Integer id) {
    for (Empresa empresa: lista) {
        if(empresa.getId() == id ) {
            lista.remove(empresa);
        }
    }
}
```

COPIAR CÓDIGO

Para testarmos a modificação no navegador, clicaremos sobre o link "remove" relacionado à empresa **Alura**. Teremos um comportamento que indica que a empresa foi removida, contudo ainda resta elaborarmos uma resposta para esta ação.

Ao atualizarmos a lista, restará a empresa **Caelum**, que também tentaremos remover. Mas ao tentarmos finalizar a ação de deletar a empresa, receberemos a mensagem de erro:

> HTTP Status 500 - Internal Server Error
>
> Type Exception Report
>
> Description The server encountered an unexpected condition that prevent it from fulfilling the request.
>
> Exception
>
> java.util.ConcurrentModificationException

O **erro 500**, como já sabemos, significa um problema do lado do servidor. Mesmo a exceção sendo indicada, é uma boa prática como desenvolvedor analisar, no Eclipse, o console do Tomcat, onde teremos mais informações

acerca do problema, uma vez que as mensagens de erro exibidas no navegador normalmente são incompletas.

Nos chama atenção a seguinte mensagem no console do Tomcat:

> at
> br.com.alura.gerenciador.servlet.Banco.removeEmpresas(Banco.java:32)

Ao clicarmos sobre a mensagem, seremos direcionados para `Banco.java`, mais especificamente no laço que criamos no método `removeEmpresa()`. O problema ocorreu porque não podemos fazer um laço e modificar a lista ao mesmo tempo, pelo menos não na implementação `Arraylist` que fizemos.

```
public void removeEmpresa(Integer id) {
    for (Empresa empresa: lista) {
        if(empresa.getId() == id ) {
            lista.remove(empresa);
        }
    }
}
```

COPIAR CÓDIGO

A forma tradicional de resolvermos esse problema foi introduzida no Java 1.5. Quando realizamos um laço, cada coleção (lista) existente no projeto possui um objeto específico para fazer a iteração, chamado `Iterator`. Esse objeto ainda existe, e é proveniente de `lista`, que também possui método `iterator()`.

Realizaremos a importação da classe `Iterator (java.util)`. Introduziremos também `<Empresa>`, para que a interação seja feita com os elementos desse tipo.

```
public void removeEmpresa(Integer id) {

    Iterator<Empresa> it = lista.iterator();
```

```
public void removeEmpresa(Integer id) {
    for (Empresa empresa: lista) {
        if(empresa.getId() == id ) {
            lista.remove(empresa);
        }
    }
}
```

COPIAR CÓDIGO

Para acessarmos coleção a partir do `Iterator` usaremos o método `hasNext()` e verificaremos se há um próximo elemento na coleção. Enquanto (`while`) existir um próximo elemento a ser acessado, o laço continuará.

Como sabemos que de fato há um próximo elemento, ele será acessado. Portanto escreveremos `Empresa emp = it.next()` . Em seguida, faremos o laço `if` , e utilizaremos o `Iterator` para remover o elemento: se o `id` da empresa for igual ao do parâmetro, a empresa será removida.

```
public void removeEmpresa(Integer id) {

    Iterator<Empresa> it = lista.iterator();

    while(it.hasNext()) {
        Empresa emp = it.next();

        if(emp.getId() == id ) {
            it.remove();
        }
    }
}
```

COPIAR CÓDIGO

Dessa forma, não usaremos a lista diretamente, mas tomaremos ações a partir de objetos específicos que sabem acessar e remover elementos de uma determinada coleção.

Finalizadas as modificações, podemos testá-las no navegador. Na lista de empresas, removeremos a **Alura** e **Caelum**, clicando sobre os links "remove" ao lado de cada uma delas. Ao atualizarmos a lista de empresas, veremos que esta está vazia, portanto, de fato a remoção foi efetivada.

Aparentemente tudo funciona, mas resta construímos uma resposta à essa ação, de forma que o usuário seja mais bem orientado sobre o que está acontecendo na aplicação.

Em `RemoveEmpresaServlet` , adicionaremos o `reponse.sendRedirect("listaEmpresas")` .

```
String paramId = request.getParameter("id");
Integer id = Integer.valueOf(paramId);

System.out.println(id);

Banco banco = new Banco();
banco.removeEmpresa(id);

response.sendRedirect("listaEmpresas");
```

COPIAR CÓDIGO

Dessa forma, toda a vez que removermos um item da lista, seremos redirecionados para a lista de cadastramento atualizada como resposta.