06

Formulário de alteração

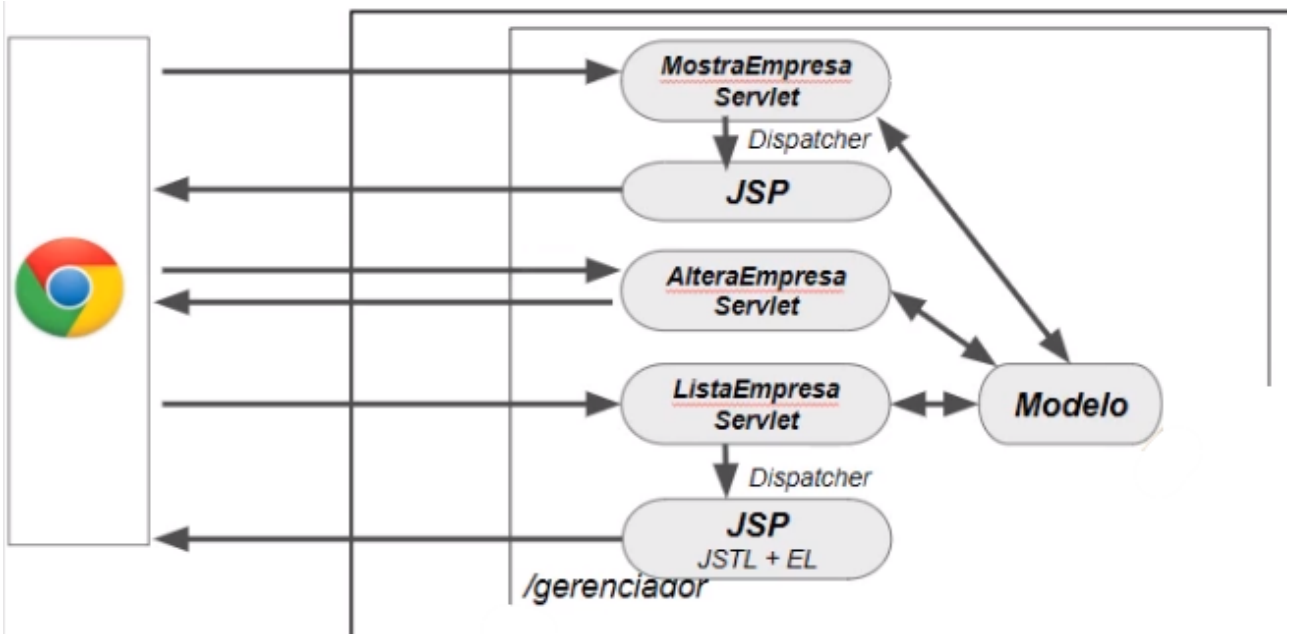
Transcrição

Ao acessarmos http://localhost:8080/gerenciador/listaEmpresas (http://localhost:8080/gerenciador/listaEmpresas), teremos as duas empresas, **Alura** e **Caelum**, cadastradas automaticamente. Ao lado de cada uma, é exibido o link "remove". A ideia agora é criamos um segundo link, chamado "editar".

Antes de trabalharmos diretamente nessa implementação, vamos nos atentar para um detalhe de fluxo.

Precisamos criar um Servlet para alterar os dados de uma empresa, como o nome ou a data de criação. Porém, antes disso, precisaremos de outro Servlet. Isso porque, quando clicarmos sobre o link "edita", a ideia é que sejamos redirecionados para outra página que exibe os dados dessa empresa.

Clicando no link "edita", o navegador chamará um Servlet que intitularemos de `MostraEmpresaServlet`, cuja tarefa consiste em carregar os dados dessa empresa. Por meio do JSP, popularemos um formulário que é devolvido para o navegador, para que este formulário seja preenchido e eventualmente modificado pelo usuário. No caso de modificação, será chamado o `AlteraEmpresaServlet`, que modificará o modelo.



Em `listaEmpresas`, criaremos um link que chama o Servlet `mostraEmpresa`, enviando o `id` da empresa, de forma que fique explícito com qual empresa queremos trabalhar.

```
<li>
    ${Empresa.nome} - <fmt>formatDate value="${empresa.dataAbe
    <a href="/gerenciador/mostraEmpresa?id=${empresa.id}">edit
    <a href="/gerenciador/removeEmpresa?id=${empresa.id}">remo
</li>
```

COPIAR CÓDIGO

Em seguida, precisaremos criar o Servlet `mostraEmpresaServlet`, cujo mapeamento será `/mostraEmpresa`, e ele incluirá somente o método `doGet()`, sem construtor. Limparemos um pouco o código removendo os comentários:

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

import java.io.IOException;

@WebServlet("/mostraEmpresa")
public class MostraEmpresaServlet extends HttpServlet {
    private static final serialVersionUID = 1L;
```

```
    protected void doGet(HttpServletRequest request, HttpServle

        response.getWriter().append("Served at: ").append(requ
    }
}
```

COPIAR CÓDIGO

Faremos um procedimento similar ao que realizamos em `removeEmpresaServlet`, isto é, o *parsing* do parâmetro.

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

import java.io.IOException;

@WebServlet("/mostraEmpresa")
public class MostraEmpresaServlet extends HttpServlet {
    private static final serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServle

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

        response.getWriter().append(Served at: ").append(reques
    }
}
```

COPIAR CÓDIGO

A partir de agora, precisamos buscar as informações da empresa no banco de dados. Iremos à classe auxiliar `Banco`, e usaremos o método `buscaEmpresaPelaId()`, ainda não criado, mas sabemos que essa será sua estrutura e que ele deverá nos devolver uma `Empresa`.

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

import java.io.IOException;

@WebServlet("/mostraEmpresa")
public class MostraEmpresaServlet extends HttpServlet {
    private static final serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServl

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

        Banco banco = new Banco();

        Empresa empresa = banco.buscaEmpresaPelaId(id);

        response.getWriter().append(Served at: ").append(reque
    }
}
```

COPIAR CÓDIGO

Assim feito, o próximo passo é criar o método `buscaEmpresaPelaId()` em nossa classe auxiliar `Banco.java`. Dentro do laço criaremos a condicional `if`, e por fim devolvemos a empresa por meio de `return empresa`.

```
public Empresa buscaEmpresaPelaId(Integer id) {
    for (Empresa empresa: lista) {
        if(empresa.getId() == id)  {
            return empresa;
        }

    }
    return null;
}
```

COPIAR CÓDIGO

Manteremos o retorno nulo neste caso, mas existem duas formas de se escrever um método dessa natureza. No primeiro, se recebemos um valor nulo do Servlet, sabemos que a empresa é inexistente. Em outro caso, se a `id` passada não existe, o método retorna uma exceção.

Pensando na boa prática de sempre executar o processo passo a passo, testaremos as modificações imprimindo o nome da empresa por meio do `System.out.println(empresa.getNome)`

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

import java.io.IOException;

@WebServlet("/mostraEmpresa")
public class MostraEmpresaServlet extends HttpServlet {
    private static final serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServle

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

        Banco banco = new Banco();

        Empresa empresa = banco.buscaEmpresaPelaId(id);

        System.out.println(empresa.getNome());

        response.getWriter().append("Served at: ").append(requ
    }
}
```

COPIAR CÓDIGO

Reiniciaremos o Tomcat, e, acessando a URL http://localhost:8080/gerenciador/listaEmpresas (http://localhost:8080/gerenciador/listaEmpresas), veremos o novo link "edita", que nos direciona para http://localhost:8080/gerenciador/mostraEmpresa (http://localhost:8080/gerenciador/mostraEmpresa).

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

O Servlet foi chamado, portanto nosso código está funcional. Contudo, ainda nos é devolvida uma mensagem inútil:

> Served at: /gerenciador.

Enquanto isso, no console do Eclipse a empresa **Alura** é impressa, o que revela que nosso método de `getNome()` funciona.

A próxima etapa, como vimos no fluxo no começo desse capítulo, é popularmos o JSP com um formulário a partir do despachador. Criaremos esse JSP, cujo nome será `formAlteraEmpresa.jsp`.

Mas antes de trabalharmos no JSP, adicionaremos o `RequestDispatcher` em `MostraEmpresaServlet`. No despachador, indicaremos para onde a requisição deve ser enviada - neste caso, para `/formAlteraEmpresa.jsp`. Adicionaremos o `rd.forward(request, response)` para que a requisição e resposta sejam de fato enviadas.

Por fim, usaremos o termo `request.setAttribute()` para que outas informações sejam levadas com a requisição, no caso, `empresa` e sua

referência.

```
    protected void doGet(HttpServletRequest request, HttpServle

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

        Banco banco = new Banco();

        Empresa empresa = banco.buscaEmpresaPelaId(id);

        System.out.println(empresa.getNome());

        request.setAttribute("empresa", empresa);

        RequestDispatcher rd = request.getRequestDispatcher("/
        rd.forward(request, response);
    }
}
```

COPIAR CÓDIGO

Testaremos as modificações o navegador. Abriremos a página de lista de empresas, e clicaremos no link "edita" para a empresa **Alura**. Seremos direcionados para um formulário não populado. Ainda precisamos resolver esse problema.

Em `formAlteraEmpresa.jsp`, temos o `<input>`:

```
<form action="${linkServletNovaEmpresa }" method="post">
    Nome: <input type="text" name="nome" />
    Data Abertura: <input type="text" name="data" />
```

COPIAR CÓDIGO

O que queremos é que esse `<input>` já possua um valor inicial (`value`). Para imprimirmos o nome da empresa dentro do JSP, usaremos a *expression language*, e chamaremos o atributo `empresa.nome` . Em `Data Abertura` , faremos o mesmo procedimento, mas alterando o parâmetro para `empresa.dataAbertura`

```
<form action="${linkServletNovaEmpresa }" method="post">

    Nome: <input type="text" name="nome" value="${empresa.nome
    Data Abertura: <input type="text" name="data" value="${empr
```

COPIAR CÓDIGO

Ao acessarmos novamente o formulário, teremos os campos "Nome" e "Data Abertura" povoados, mas a data exibida no segundo campo não estará formatada. Já sabemos corrigir esse problema: acessaremos o `listaEmpresas` para reaproveitarmos o código correspondente à formatação de datas, isto é, o `fmt` .

```
<li>
    ${empresa.nome} - <fmt:formatDate value="${empresa.dataAber
    <a href="/gerenciador/mostraEmpresa?id=${empresa.id }">edi
    <a href="/gerenciador/removeEmpresa?id=${empresa.id }">rem
</li>
```

COPIAR CÓDIGO

Inseriremos o `fmt` dentro das aspas duplas de `value` em `formAlteraEmpresa.jsp` .

```
<form action="${linkServletNovaEmpresa }" method="post">
```

```
Nome: <input type="text" name="nome" value="${empresa.nome
Data Abertura: <input type="text" name="data" value="<fmt:
```

COPIAR CÓDIGO

Por fim, precisaremos importar a biblioteca `fmt` :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"
```

COPIAR CÓDIGO

Ao acessarmos novamente o formulário pelo navegador, a data já estará com a formatação correta:

Nome: Alura Data Abertura: 31/08/2018

Ao analisarmos o código fonte da página, confirmaremos que a formatação da data (o fmt) já foi interpretado no servidor. Lembrando que o navegador não entende o fmt, por isso essa leitura é feita pelo Tomcat.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

        <form action="/gerenciador/novaEmpresa" method="post">

            Nome: <input type="text" name="nome" value"Caelu "
            Data Abertura: <input type="text" name="data" va
```

```
            <input type="submit" />
        </form>

    </body>
    </html>
```

COPIAR CÓDIGO

Ainda temos um pequeno problema: não deveríamos chamar `/gerenciador/novaEmpresa`, mas sim `/gerenciador/AlteraEmpresa`, como vimos na apresentação no começo deste capítulo. Isto é, ainda precisamos criar o Servlet e a URL `alteraEmpresa`.

```
<c:url value="/alteraEmpresa" var="linkServletNovaEmpresa"/>
```

COPIAR CÓDIGO

Finalizamos nosso JSP, os dados estão sendo populados e nas próximas etapas aprenderemos como fazer a alteração desses dados.