Transcrição

Agora faremos a delegação das outras duas ações que separamos no vídeo anterior (`RemoveEmpresa` e `MostraEmpresa`). Ainda faltarão outras ações, mas faremos a delegação delas com calma no futuro.

Criaremos uma classe para "RemoveEmpresa" clicando com o botão direito no pacote `acao` e em seguida em "New > Class". Lembre-se de que estamos criando uma classe comum, e não um Servlet, o que torna esse processo mais fácil (já que não precisaremos respeitar todos os detalhes que um Servlet requer).

Aqui criaremos o método `executa()` , que encapsula e executa o nosso código, e que recebe os mesmos parâmetros e exceções da classe `ListaEmpresas` . Portanto, vamos copiá-los aqui:

```
public class RemoveEmpresa {

    public void executa(HttpServletRequest request, HttpServletResponse response) throw

    }
}
```

Esse esboço de código é aplicável para todas as outras ações. Portanto, para simplificar, podemos copiar essa classe, colá-la no mesmo pacote e renomeá-la para `MostraEmpresa` .

No nosso `UnicaEntradaServlet` , vamos criar a ação `RemoveEmpresa` e intanciá-la, o que é muito simples, já que é um objeto que pode ser intanciado chamando o construtor sem argumentos. Para simplificar, seguiremos a mesma nomeclatura de `ListaEmpresas` , criando `acao.executa()` , e repetiremos o mesmo processo para `MostraEmpresa` . Assim, teremos:

```
if(paramAcao.equals("ListaEmpresas")) {

    ListaEmpresas acao = new ListaEmpresas();
    acao.executa(request, response);

} else if(paramAcao.equals("RemoveEmpresa")) {

    RemoveEmpresa acao = new RemoveEmpresa();
    acao.executa(request, response);

} else if(paramAcao.equals("MostraEmpresa")) {

    MostraEmpresa acao = new MostraEmpresa();
    acao.executa(request, response);
}
```

Nesse código, já retiramos as linhas `System.out.println("removendo empresa")` e `System.out.println("mostrando dados da empresa")` , e colocamos em suas respectivas classes, por exemplo:

```
public class MostraEmpresa {

    public void executa(HttpServletRequest request, HttpServletResponse response) throw

        System.out.println("mostrando dados da empresa");
    }
}
```

Agora, todo código específico para as nossas ações, como `RemoveEmpresa` e `MostraEmpresa` , está separado em outras classes. Vamos então preencher a classe `RemoveEmpresa` copiando o código do nosso `RemoveEmpresaServlet` e colando-o código aqui, já que a lógica continua a mesma - precisamos ler o parâmetro, fazer o parsing, usar o nosso modelo e enviar um redirecionamento.

```
public class RemoveEmpresa {

    public void executa(HttpServletRequest request, HttpServletResponse response) throw

        System.out.println("removendo empresa");

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

        System.out.println(id);

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

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

Essas ações já tinham sido implementadas, só estamos reorganizando nosso código. Se rodarmos o Tomcat, perceberemos que nosso código está funcionando corretamente.

Porém, se usarmos o link "remover" em http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas, ele continuará chamando o nosso Servlet, e não a nossa entrada única.

Portanto, precisaremos mexer no `.jps` para testarmos essa funcionalidade (ou digitar todas as URLs diretamente no navegador). Vamos abrir o `listaEmpresas.jsp` . A ideia é que qualquer link listado aqui chame o nosso `UnicaEntradaServlet` , enviando o `id` para que ele saiba qual empresa tem que remover ou mostrar:

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

            <li>
                ${empresa.nome } - <fmt:formatDate value="${empresa.dataAbertura }" pat
                <a href="/gerenciador/entrada?id=${empresa.id }">edita</a>
                <a href="/gerenciador/entrada?id=${empresa.id }">remove</a>
            </li>
        </c:forEach>
    </ul>
```

Porém, falta mais um parâmetro: precisamos falar para a nossa `UnicaEntradaServlet` qual é a ação. Portanto, além do `id` , vamos adicionar os parâmetros `acao=MostraEmpresa` e `acao=RemoveEmpresa` , respectivamente. Para separarmos os parâmetros, utilizamos `&` . Dessa forma, teremos:

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

            <li>
                ${empresa.nome } - <fmt:formatDate value="${empresa.dataAbertura }" pat
                <a href="/gerenciador/entrada?acao=MostraEmpresa&id=${empresa.id }">edi
                <a href="/gerenciador/entrada?acao=RemoveEmpresa&id=${empresa.id }">rem
            </li>
        </c:forEach>
    </ul>
```

> Dica: também é possível adicionar mais parâmetros separando-os com `&` .

Agora podemos consultar a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas para testarmos essas alterações. Como elas foram feitas no `.jsp` , não é necessário reiniciar o Tomcat. Iremos perceber que nossos links terão mudado e estarão sempre chamando `entrada` , usando o mesmo Servlet, e a ideia é exatamente essa.

Quando clicamos em "edita", por exemplo, somos encaminhados para a URL http://localhost:8080/gerenciador/entrada?acao=MostraEmpresa&id=1, então parece tudo certo.

Porém, temos um problema. Quando abrimos a página para editar as informações de uma empresa, como http://localhost:8080/gerenciador/entrada?acao=MostraEmpresa&id=1, e clicamos em enviar, o navegador volta a chamar a URL http://localhost:8080/gerenciador/listaEmpresas.

A mesma coisa acontece após removermos uma empresa. A ação `RemoveEmpresa` é executada corretamente, e isso inclusive pode ser conferido no console. Porém, ela continua chamando o nosso servlet na linha `response.sendRedirect("listaEmpresas")` .

Nossos Servlets continuam funcionando pois eles fazem parte do projeto, portanto precisaremos desabilitá-los. Nessa linha, queremos chamar nossa `entrada` , que precisa saber o que deve ser executado. Portanto, vamos definir o parâmetro que deve ser enviado utilizando `response.sendRedirect("entrada?acao=ListaEmpresas")` .

Dessa forma, o navegador irá enviar o usuário a `entrada` , já submetendo o parâmetro correto. Agora basta salvarmos e reiniciarmos o Tomcat para testarmos nosso código novamente.

Dessa vez, acessando a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas e clicando em "remover", conseguiremos chamar nossa ação e voltar à entrada corretamente.

Para garantirmos que `UnicaEntradaServlet` realmente seja nosso único Servlet, vamos comentar a linha `@WebServlet()` em `AlteraEmpresaServlet` , `ListaEmpresaServlet` , `MostraEmpresaServlet` `NovaEmpresaServlet` e `RemoveEmpresaServlet` , como no exemplo:

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

    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
//...
```

Após fazermos isso, a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas, a opção "remover" e a opção "editar" continuarão funcionando corretamente. Porém, se clicarmos em "Enviar" após fazermos alguma alteração... não irá funcionar. Isso porque comentamos `AlteraEmpresaServlet` , mas não criamos uma ação `AlteraEmpresa` . Mas isso é fácil de corrigir, certo?

Vamos copiar a classe `RemoveEmpresa` e colá-la no mesmo pacote, renomeando a nova classe criada como `AlteraEmpresa` . Agora copiaremos toda a implementação que construímos em `AlteraEmpresaServlet` e colaremos nessa classe. Assim, teremos:

```
public class AlteraEmpresa {

    public void executa(HttpServletRequest request, HttpServletResponse response) throw

        System.out.println("acao altera empresa" + id);

        String nomeEmpresa = request.getParameter("nome");
        String paramDataEmpresa = request.getParameter("data");
        String paramId = request.getParameter("id");
        Integer id = Integer.valueOf(paramId);

        Date dataAbertura = null;
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            dataAbertura = sdf.parse(paramDataEmpresa);
        } catch (ParseException e) {
            throw new ServletException(e);
        }

        System.out.println(id);

        Banco banco = new Banco();
        Empresa empresa = banco.buscaEmpresaPelaId(id);
        empresa.setNome(nomeEmpresa);
        empresa.setDataAbertura(dataAbertura);

        response.sendRedirect("entrada?acao=ListaEmpresas");
    }
}
```

Repare que concatenamos o `id` na linha `System.out.println("acao altera empresa")` para termos uma saída mais expressiva. Além disso, fazemos o redirecionamento em `System.out.println("acao altera empresa");` para o nosso `UnicaEntradaServlet` , ao invés do Servlet que não está mais mapeado, da mesma forma que no exemplo anterior.

Ainda precisaremos alterar nosso formulário `formAlteraEmpresa.jsp` . Na linha `<c:url value="/alteraEmpresa" var="LinkServletNovaEmpresa"/>` , vamos alterar o valor (`value`) para `/entrada` e a variável (`var`) para `linkEntradaServlet` . Também usaremos `linkEntradaServlet` para definir `action` .

Estamos enviando, através do formulário, os dados da empresa. Porém, também precisamos enviar para nosso `UnicaEntradaServlet` qual ação deve ser executada. Para isso, criaremos mais um campo escondido, cujo parâmetro se chama `acao` e cujo valor será `AlteraEmpresa` .

Com isso, teremos:

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


<c:url value="/entrada" var="linkEntradaServlet"/>

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

    <form action="${linkEntradaServlet }" method="post">

        Nome: <input type="text" name="nome" value="${empresa.nome }" />
        Data Abertura: <input type="text" name="data"  value="<fmt:formatDate value="${
        <input type="hidden" name="id" value="${empresa.id }">
        <input type="hidden" name="acao" value="AlteraEmpresa">
        <input type="submit" />
    </form>

</body>
</html>
```

Agora só falta criarmos mais um `else if` no nosso `UnicaEntradaServlet` ...

```
} else if (paramAcao.equals("AlteraEmpresa")) {
    AlteraEmpresa acao = new AlteraEmpresa();
    acao.executa(request, response);
}
```

...e importarmos a classe do nosso pacote `acao` com "Ctrl + Shift + O". Vamos aproveitar para já prepararmos o `else if` para `NovaEmpresa` , que é o único Servlet que ainda não reimplementamos:

```
} else if (paramAcao.equals("NovaEmpresa")) {
        NovaEmpresa acao = new NovaEmpresa();
        acao.executa(request, response);
}
```

Agora basta criarmos a classe `NovaEmpresa` , repetindo o processo que fizemos para `AlteraEmpresa` . Não podemos nos esquecer de mudar o redirecionamento na linha `response.sendRedirect("listaEmpresas");` para `response.sendRedirect("entrada?acao=ListaEmpresas");` :

```
public class NovaEmpresa {

    public void executa(HttpServletRequest request, HttpServletResponse response) throw

        System.out.println("Cadastrando nova empresa");

        String nomeEmpresa = request.getParameter("nome");
        String paramDataEmpresa = request.getParameter("data");

        Date dataAbertura = null;
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            dataAbertura = sdf.parse(paramDataEmpresa);
        } catch (ParseException e) {
            throw new ServletException(e);
        }

        Empresa empresa = new Empresa();
        empresa.setNome(nomeEmpresa);
        empresa.setDataAbertura(dataAbertura);

        Banco banco = new Banco();
        banco.adiciona(empresa);

        request.setAttribute("empresa", empresa.getNome());

        response.sendRedirect("entrada?acao=ListaEmpresas");
    }
}
```

Agora podemos importar essa classe no `UnicaEntradaServlet` . Apesar de ainda não termos testado, na teoria nosso código está funcionando como deveria (pelo menos por enquanto).

Nesse momento, temos um `else if` para cada ação do nosso código, cada uma em uma classe dedicada, e sempre passamos pelo `UnicaEntradaServlet` , já que não temos mais nenhum outro Servlet funcionando (exceto `OiMundo` , no qual não iremos mexer). Vamos testar?

Iniciando o servidor e acessando a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas, iremos verificar que as funções de remover empresa e editar empresa estão funcionando.

Por exemplo, se tentarmos editar a empresa **Alura**, seremos redirecionados para a página http://localhost:8080/gerenciador/entrada?acao=MostraEmpresa&id=1. Se exibirmos o código fonte dessa página, veremos que nossos dois parâmetros escondidos estão no código - ou seja, a ação que se chama `AlteraEmpresa` será executada.

Se fizermos alguma alteração e clicarmos em "Enviar", ela será salva e seremos redirecionados para `ListaEmpresas` , sempre usando nosso único Servlet.

Mas ainda temos um problema: se exibirmos o código fonte da página http://localhost:8080/gerenciador/formNovaEmpresa.jsp (que utilizamos para criar uma nova empresa), veremos que ele não irá funcionar, pois ainda está chamando nosso `NovaEmpresaServlet` . Portanto, precisamos ajustar o `formNovaEmpresa.jsp` .

Vamos alterar o campo `value` para `/entrada` e os campos `var` e `action=` para `linkEntradaServlet` . Além disso, teremos que enviar mais um parâmetro escondido. Faremos isso com `<input type="hidden" name="acao" value="NovaEmpresa" />` . Assim, teremos:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:url value="/entrada" var="linkEntradaServlet"/>

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

    <form action="${linkEntradaServlet }" method="post">

        Nome: <input type="text" name="nome"  />
        Data Abertura: <input type="text" name="data"  />

        <input type="hidden" name="acao" value="NovaEmpresa" />

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

</body>
</html>
```

Dessa vez, se usarmos a URL http://localhost:8080/gerenciador/formNovaEmpresa.jsp, conseguiremos criar uma nova empresa normalmente. Agora que temos todas as classes prontas, podemos até mesmo apagar os Servlets que não usaremos mais, e nossa aplicação continuara funcionando corretamente com as ações que criamos.

Até o momento, nós reorganizamos o código para deixá-lo mais simples. Porém, nosso Servlet ainda pode ser melhorado para ficar mais elegante e inteligente. Faremos isso nos próximos vídeos. Até lá!