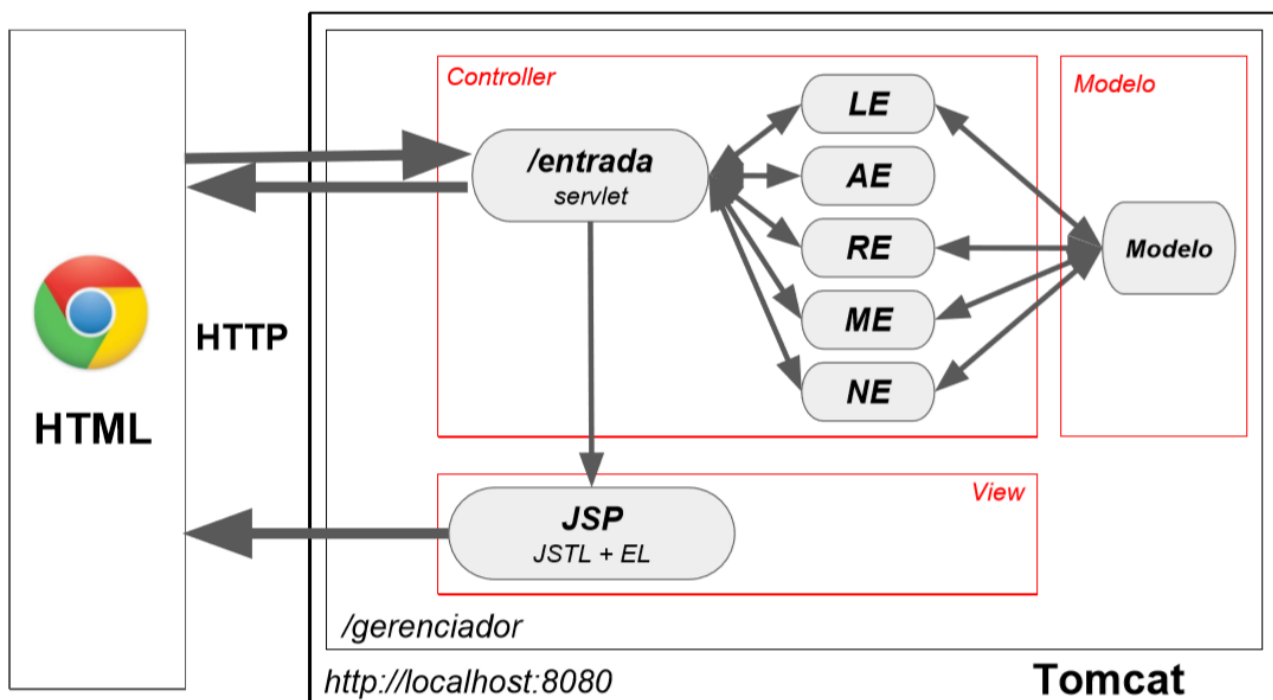




Centralizando o redirecionamento

Transcrição

Olá, bem vindo de volta. Vamos continuar a construção do nosso próprio controlador. Na última aula, criamos um Servlet único que serve como controlador (/entrada) e que vai delegar as chamadas para as nossas ações.



Porém, nessa implementação, são as ações que chamam o `.jsp`, e não a `/entrada`. Agora corrigiremos isso, e quem chamará o `.jsp` será o nosso controlador, pois é ele quem controla o fluxo.

Começaremos a trabalhar com `ListaEmpresas.java`:

```
public class ListaEmpresas {

    public void executa(HttpServletRequest request, HttpServletResponse response) {
```

```
System.out.println("listando empresas");

Banco banco = new Banco();
List<Empresa> lista = banco.getEmpresas();

request.setAttribute("empresas", lista);

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

}
```

[COPIAR CÓDIGO](#)

A nossa ação está usando um `RequestDispatcher`. Vamos recortar esse código ("Ctrl + X") e colá-lo no nosso `UnicaEntradaServlet`. Poderíamos fazer isso dessa forma:

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

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

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

[COPIAR CÓDIGO](#)

Porém, trabalharemos de outra maneira, colando esse trecho ao final do código:

```
protected void service(HttpServletRequest request, HttpServlet ...
```

```
String paramAcao = request.getParameter("acao");

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);

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

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

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

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

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

[COPIAR CÓDIGO](#)

Repare que nosso `RequestDispatcher` está chamando `/listaEmpresas.jsp`, e só deveria ser utilizado para a ação `ListaEmpresas`. Em outras ações, como `MostraEmpresa`, deveríamos executar outro `.jsp`. Portanto, o nome do `.jsp` varia dependendo da ação que estamos executando, certo?

Por isso, não devemos deixar esse parâmetro fixo no nosso código. Precisamos sinalizar, de alguma forma, o que nosso Servlet deve fazer. Repare que, no nosso projeto, o objetivo é que o controlador `/entrada` passe os dados (a requisição) para nossa classe `ListaEmpresas`, que deve devolver o nome do nosso `.jsp`. Dessa forma, o Servlet saberá o que deve ser feito.

Para tanto, não teremos um `void` na classe `ListaEmpresas`, mas sim uma `String`. Essa `String` deve retornar `"listaEmpresas.jsp"`:

```
public class ListaEmpresas {  
  
    public String executa(HttpServletRequest request, HttpServletResponse response)  
    {  
  
        System.out.println("listando empresas");  
  
        Banco banco = new Banco();  
        List<Empresa> lista = banco.getEmpresas();  
  
        request.setAttribute("empresas", lista);  
  
        return "listaEmpresas.jsp";  
    }  
}
```

[COPIAR CÓDIGO](#)

Ou seja, nosso método devolve uma `String`, e essa `String` devolve o nome do `.jsp`. Agora, no `UnicaEntradaServlet`, a instrução `if` deve receber o nome do `.jsp`:

```
if(paramAcao.equals("ListaEmpresas")) {  
  
    ListaEmpresas acao = new ListaEmpresas();  
    String nome = acao.executa(request, response);  
}
```

[COPIAR CÓDIGO](#)

Ao invés de usarmos o `RequestDispatcher`, vamos colocar o `nome` como argumento:

```
RequestDispatcher rd = request.getRequestDispatcher(nome);  
rd.forward(request, response);
```

[COPIAR CÓDIGO](#)

O código não irá funcionar, pois a variável/String "nome" só é visível dentro dos colchetes da instrução `if`. Portanto, precisaremos definir essa variável antes da instrução:

```
String nome = null  
if(paramAcao.equals("ListaEmpresas")) {  
  
    ListaEmpresas acao = new ListaEmpresas();  
    nome = acao.executa(request, response);  
}
```

[COPIAR CÓDIGO](#)

Agora a instrução `if` executa a ação, leva a `request` e `response` e nos devolve o nome do `.jsp`. Esse código já deve funcionar para `ListaEmpresas`.

Testaremos isso acessando a URL <http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas> (<http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas>).

Nossa `ListaEmpresas` está funcionando, mas as outras ações ainda não. Agora trabalharemos com `RemoveEmpresa`:

```
public class RemoveEmpresa {
```

```
public void executa(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
    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("entrada?acao=ListaEmpresas");  
}  
}
```

[COPIAR CÓDIGO](#)

Aqui temos outros problemas. Repare que `RemoveEmpresa` não usa o `RequestDispatcher`, mas sim `response.sendRedirect`. Nesse caso, ou chamamos o `.jsp` dentro da mesma requisição e usamos o `Dispatcher` da mesma requisição, ou voltamos para o navegador e o instruímos a enviar uma nova requisição (`response.sendRedirect()`). Nosso controlador tem que atender as duas situações.

Vamos recortar a linha `response.sendRedirect("entrada?acao=ListaEmpresas");` e criar um `if()` no nosso controlador:

```
if() {  
    RequestDispatcher rd = request.getRequestDispatcher(nome);  
    rd.forward(request, response);  
} else {  
    response.sendRedirect("entrada?acao=ListaEmpresas");  
}
```

[COPIAR CÓDIGO](#)

Dessa forma, ou usaremos `RequestDispatcher` ou `response.sendRedirect()`.

Na classe `ListaEmpresas`, incluiremos o prefixo `forward` para nosso `.jsp`.

```
return "forward:listaEmpresas.jsp";
```

[COPIAR CÓDIGO](#)

Já a classe `RemoveEmpresa` não deve mais receber um `void`, mas sim uma `String`. Nela, devemos retornar um `response.sendRedirect()` para entrada?acao=ListaEmpresas:

```
public class RemoveEmpresa {  
  
    public String executa(HttpServletRequest request, HttpServ.  
  
        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);  
  
        return "redirect:entrada?acao=ListaEmpresas";  
    }  
}
```

[COPIAR CÓDIGO](#)

Através do prefixo `forward`, definimos que nossas ações não usarão mais `response.sendRedirect()` nem o `RequestDispatcher`. Quem fará isso será nosso controlador `UnicaEntradaServlet`. Vamos implementar:

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

[COPIAR CÓDIGO](#)

A string `nome` agora tem sempre um prefixo. Então, precisaremos separá-la em duas partes. Faremos isso utilizando `split()`, um método prático da classe `String`.

Queremos que o método separe essa string baseando-se em um caractere de separação - no caso, `:`. O método `split()` devolve um array, com uma variável que contém duas posições - o tipo e o endereço. Então, criaremos uma variável chamada `tipoEEndereco` (mas você pode usar outro nome).

Se a primeira posição desse array (`[0]`) tem uma string que se chama `forward`, queremos usar o `Dispatcher`. Assim teremos `if(tipoEEndereco[0].equals("forward"))`. Como parâmetro de `request.getRequestDispatcher()`, ao invés de `nome`, usaremos o valor na segunda posição (`[1]`) do nosso array `tipoEEndereco`.

Dessa forma, se tivermos `forward` na primeira posição do array, usaremos o valor da segunda posição para recebermos `listaEmpresas.jsp`. Se não (`else`), faremos um `sendRedirect()` para `tipoEEndereco` na segunda posição (`[1]`).

Assim, teremos:

```
String[] tipoEEndereco = nome.split(":");  
if(tipoEEndereco[0].equals("forward")) {
```



```
RequestDispatcher rd = request.getRequestDispatcher(tipoEEndereco[0]);  
rd.forward(request, response);  
} else {  
    response.sendRedirect(tipoEEndereco[1]);  
}
```

[COPIAR CÓDIGO](#)

Agora podemos testar nossas ações `ListaEmpresas` e `RemoveEmpresa`.

Acessando a URL [http://localhost:8080/gerenciador/entrada?](http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas)

[acao=ListaEmpresas](http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas) ([http://localhost:8080/gerenciador/entrada?](http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas)

[acao=ListaEmpresas](http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas)), veremos que as duas ações continuam funcionando corretamente, o que é um bom sinal. Agora precisaremos ajustar as outras três ações com base nesse padrão que criamos.

Faremos isso no próximo vídeo, mas sintá-se à vontade para tentar sozinho.

Até lá!