05

Ordem de execução

Transcrição

Terminamos o último vídeo com a ideia de que em algum momento gostaríamos de definir uma ordem para os nossos filtros. Nem sempre isso importa, já que os filtros podem ser independentes. Porém, pensando no caso do nosso controlador também ser um filtro, devemos garantir que ele seja o último elemento.

Primeiramente, vamos desabilitar `MonitoramentoFilter` e `AutorizacaoFilter` por meio das anotações. Dessa forma, eles não serão mais carregados pelo Tomcat, mas mapearemos esses filtros no `web.xml` .

Como já estabelecemos anteriormente, o filtro é muito parecido com um Servlet, tanto que o mapeamento é o mesmo. A diferença é que, no `web.xml` , onde estaria escrito `servlet` , escreveremos `filter` :

```
<filter>
    <filter-name>AF</filter-name>
    <filter-class>br.com.alura.gerenciador.servlet.AutorizacaoF
</filter>

<filter-mapping>
    <filter-name>AF</filter-name>
    <url-pattern>/entrada</url-pattern>
</filter-mapping>
```

COPIAR CÓDIGO

Nesse caso, definimos o nome concreto do filtro como `AF` (AutorizacaoFilter). A `filter-class` deve bater com o *full qualified name* da classe concreta. Isso tudo dentro de um elemento que se chama `filter`. Dessa forma, o Tomcat já sabe que existe um filtro nessa classe.

No mapeamento, devemos repetir o `filter-name` (`AF`) e estabelecer a nossa `url-pattern`, que é `/entrada`.

Agora faremos o mesmo para o `MonitoramentoFilter`:

```
<filter>
    <filter-name>MF</filter-name>
    <filter-class>br.com.alura.gerenciador.servlet.Monitoramen
</filter>

<filter-mapping>
    <filter-name>MF</filter-name>
    <url-pattern>/entrada</url-pattern>
</filter-mapping>
```

COPIAR CÓDIGO

Vamos colocar o `MonitoramentoFilter` primeiro no nosso `web.xml`. Dessa forma, a nossa primeira saída no console deveria sempre ser `MonitoramentoFilter`. Vamos reinicilizar o Tomcat e testar isso acessando a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas) no navegador.

Nós seremos redirecionados para `LoginForm`, tendo mandado duas requisições: uma para `ListaEmpresas`, que foi negada e retornou ao navegador, e uma para `LoginForm`.

O resultado no console será algo como:

MonitoramentoFilter

AutorizacaoFilter

Tempo de execução da ação ListaEmpresas -> 4

MonitoramentoFilter

AutorizacaoFilter

Tempo de execução da ação LoginForm -> 189

Vamos recapitular o que aconteceu na nossa aplicação durante esse teste:

- quando mandamos a requisição `ListaEmpresas`, tivemos `MonitoramentoFilter` na saída do console, pois ele foi chamado primeiro;
- `MonitoramentoFilter` começou a medir o tempo, pegou o parâmetro `acao` e chamou o método `dofilter()`, continuando para o próximo filtro;
- assim como definimos no `web.xml`, o próximo filtro é `AutorizacaoFilter`;
- como não estávamos logados, fomos redirecionados para `LoginForm` e saímos abruptamente desse método, sem passarmos pelo método `dofilter()` dessa classe;
- voltamos para o `MonitoramentoFilter`, por isso tivemos a saída "Tempo de execução da ação";
- o navegador enviou uma nova requisição, por isso recebemos novamente, na saída do console, `MonitoramentoFilter`, `AutorizacaoFilter` e "Tempo de execução da ação";
- repare que essa requisição demorou um pouco mais que a primeira, pois o HTML foi devolvido para renderizar nosso formulário.

Agora nossa motivação é transformarmos o `UnicaEntradaServlet` em um filtro. De início, vamos comentar a linha `@WebServlet(urlPatters="/entrada)` .

No pacote `servlet` , criaremos uma nova classe copiando e colando `AutorizacaoFilter` . Chamaremos essa classe de `ControladorFilter` . Deixaremos a linha `@WebFilter("/entrada")` comentada pois definiremos o mapeamento diretamente no `web.xml` .

Como esse filtro é o último na cadeia e não queremos que ele chame um próximo, apagaremos a linha `chain.doFilter(request, response)` . Além disso, apagaremos o código que construímos para verificação de login, mantendo somente as linhas em que é feito o *cast*. Assim, teremos:

```
public class ControladorFilter implements Filter {

    public void doFilter(ServletRequest servletRequest, Servle

        System.out.println("ControladorFilter");

        HttpServletRequest request = (HttpServletRequest) serv
        HttpServletResponse response = (HttpServletResponse) s

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


    }

}
```

COPIAR CÓDIGO

Vamos copiar e colar a lógica que construímos no `UnicaEntradaServlet` :

```
public class ControladorFilter implements Filter {
```

```
    public void doFilter(ServletRequest servletRequest, Servle

        System.out.println("ControladorFilter");

        HttpServletRequest request = (HttpServletRequest) servl
        HttpServletResponse response = (HttpServletResponse) s

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

        String nomeDaClasse = "br.com.alura.gerenciador.acao."

        String nome;
        try {
            Class classe = Class.forName(nomeDaClasse);//carreg
            Acao acao = (Acao) classe.newInstance();
            nome = acao.executa(request, response);
        } catch (ClassNotFoundException | InstantiationExceptio
            throw new ServletException(e);
        }


        String[] tipoEEndereco = nome.split(":");
        if(tipoEEndereco[0].equals("forward")) {
            RequestDispatcher rd = request.getRequestDispatcher
            rd.forward(request, response);
        } else {
            response.sendRedirect(tipoEEndereco[1]);

        }

    }

}
```

COPIAR CÓDIGO

Só falta fazermos o mapeamento. Para isso, no nosso `web.xml`, copiaremos um dos mapeamentos que fizemos anteriormente e colaremos após todos os

outros. Além disso, vamos editá-lo para que corresponda ao `ControladorFilter` :

```
<filter>
    <filter-name>CF</filter-name>
    <filter-class>br.com.alura.gerenciador.servlet.ControladorF
</filter>

<filter-mapping>
    <filter-name>CF</filter-name>
    <url-pattern>/entrada</url-pattern>
</filter-mapping>
```

COPIAR CÓDIGO

Agora basta salvar e reiniciar o Tomcat para testarmos as alterações acessando a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas) no navegador. Na saída do console teremos algo parecido com:

MonitoramentoFilter

AutorizacaoFilter

Tempo de execução da ação ListaEmpresas -> 11

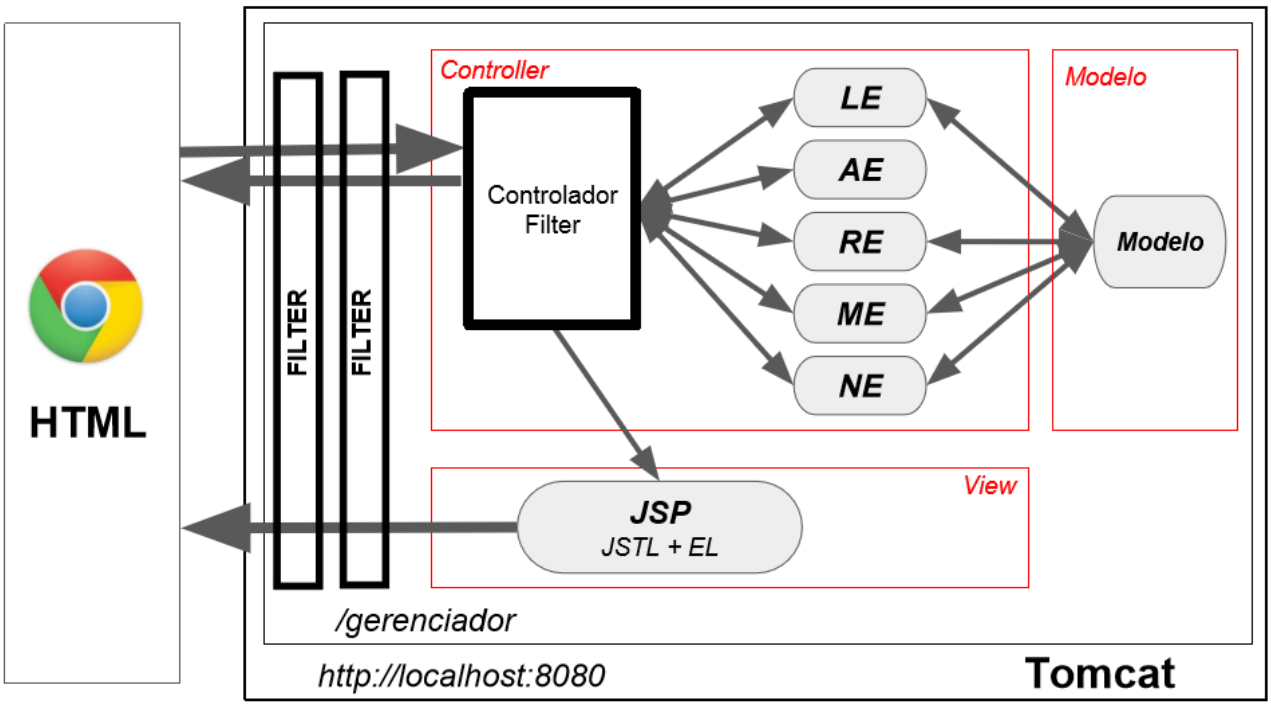
MonitoramentoFilter

AutorizacaoFilter

ControladorFilter

Tempo de execução da ação LoginForm -> 191

Aparentemente nosso `ControladorFilter` está funcionando como deveria. Ufa! Construímos uma bela cadeia de filtros.



Esse tópico de filtros é fundamental e deve ser entendido conceitualmente, mesmo que você não vá utilizar Servlets diretamente (por exemplo, por estar usando uma biblioteca de mais alto nível). Isso porque o Spring MVC, o EJB e o VRaptor também têm filtros, mas que são chamados de **interceptadores**.

Apesar dos filtros serem mais relacionados com o mundo web, enquanto os interceptadores estão mais relacionados com os *frameworks*, o conceito é o mesmo: eles sempre interceptam e filtram uma requisição, podem ser encadeados, possuem um objeto que é utilizado para enviar a requisição para o próximo passo e normalmente têm um objeto para termos acesso ao contexto, como `request e response .`

Entendendo esse conceito, você se sentirá mais confortável utilizando *frameworks* como o Spring MVC. Não é um tópico muito fácil, mas conseguimos implementar uma boa estrutura para resolver nossos problemas do dia-a-dia, de forma muito próxima ao que é feito no mundo real.

Confira os exercícios e nos encontramos na próxima aula. Até lá!