04

Filtro de autorização

Transcrição

Bem vindo de volta! No último vídeo implementamos nosso primeiro filtro - o que não é algo muito simples de entender e pode ser que você tenha dúvidas.

Mesmo com dúvidas, você deve ter reparado que nós adicionamos uma nova funcionalidade na nossa aplicação sem mexermos no controlador, nas nossas ações ou no modelo. Portanto, um filtro é muito fácil de ser "plugado" e "desplugado".

Agora nosso objetivo é criarmos mais um filtro. Dessa forma, teremos uma cadeia na qual um filtro é executado após o outro. Nós poderíamos inclusive remover o `UnicaEntradaServlet` e implementá-lo através de um filtro.

Anteriormente nós construímos a verificação de login no nosso `UnicaEntradaServlet`. Nossa motivação atual é colocarmos essa lógica em um filtro. Assim, será possível termos essa funcionalidade mesmo que estejamos utilizando um controlador do mercado, como o Spring MVC ou o VRaptor.

Dessa vez, clicando com o botão direito no pacote `servlet`, criaremos o filtro através do diálogo no menu "New > Filter". Chamaremos esse filtro de "AutorizacaoFilter". Na caixa "Filter mappings", editaremos o objeto para `/entrada` (o mesmo do nosso Servlet). Clicando em "Next", o Eclipse exibirá uma janela para implementação de interface já com `javax.servlet.Filter` adicionado, portanto basta clicarmos em "Finish".

O Eclipse então criará o filtro com os seguintes métodos:

```
@WebFilter("/entrada")
public class AutorizacaoFilter implements Filter {
    /**
     * @see Filter#destroy()
     */
    public void destroy() {
        // TODO Auto-generated method stub
    }

    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, F
     */
    public void doFilter(ServletRequest request, ServletRespon
        // TODO Auto-generated method stub
        // place your code here

        // pass the request along the filter chain
        chain.doFilter(request, response);
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletExcep
        // TODO Auto-generated method stub
    }

}
```

COPIAR CÓDIGO

O método `init()` é chamado automaticamente quando o Servlet é iniciado, e quando o Tomcat baixa a aplicação ele também pode chamar o método `destroy()`. Porém, como vimos anteriormente, não precisaremos usar esses métodos e portanto podemos apagá-los.

```
@WebFilter("/entrada")
public class AutorizacaoFilter implements Filter {

    public void doFilter(ServletRequest request, ServletRespons

        chain.doFilter(request, response);
    }

}
```

COPIAR CÓDIGO

Agora que temos o real esboço do nosso filtro, copiaremos o código que criamos para autorização na `UnicaEntradaServlet` e colaremos antes do método `chain.doFilter()`:

```
@WebFilter("/entrada")
public class AutorizacaoFilter implements Filter {

    public void doFilter(ServletRequest request, ServletRespons

        HttpSession sessao = request.getSession();
        boolean usuarioNaoEstaLogado = (sessao.getAttribute("us
        boolean ehUmaAcaoProtegida = !(paramAcao.equals("Login'
        if(ehUmaAcaoProtegida & usuarioNaoEstaLogado) {
            response.sendRedirect("entrada?acao=LoginForm");
            return;
        }

        chain.doFilter(request, response);

    }

}
```

COPIAR CÓDIGO

Não se esqueça de comentar as linhas copiadas em `UnicaEntradaServlet`.

Repare que nosso `request` não possui um método `getSession()`, pois, mesmo que o filtro receba `request` e `response`, os tipo não são `HttpServletRequest` e `HttpServletResponse`, apenas `ServletRequest` e `ServletResponse` - ou seja, é uma interface um pouco mais simples do que as classes que recebemos em `UnicaEntradaServlet`.

Portanto, precisaremos fazer um cast para obtermos uma referência mais específica (`HttpServletRequest` e `HttpServletResponse`) a partir de uma referência genérica. Vamos aproveitar para renomear nossas variáveis para `servletRequest` e `servletResponse`.

Também precisaremos pegar o parâmetro com o nome `acao` a partir da requisição, o que fazemos com a linha `String paramAcao = request.getParameter("acao")`.

```
@WebFilter("/entrada")
public class AutorizacaoFilter implements Filter {

    public void doFilter(ServletRequest servletRequest, Servle

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

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

        HttpSession sessao = request.getSession();
        boolean usuarioNaoEstaLogado = (sessao.getAttribute("us
        boolean ehUmaAcaoProtegida = !(paramAcao.equals("Login'
        if(ehUmaAcaoProtegida & usuarioNaoEstaLogado) {
            response.sendRedirect("entrada?acao=LoginForm");
            return;
        }
```

```
            chain.doFilter(request, response);

        }

    }
```

COPIAR CÓDIGO

Em teoria nosso filtro de autorização já deveria estar funcionando. Vamos testar? Se tentarmos chamar a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas), seremos redirecionados para `LoginForm` e conseguiremos fazer o login e o logout normalmente.

Além disso, nosso `MonitoramentoFilter` continua funcionando e mostrando as saídas no console:

> Logando nico
>
> Usuario existe
>
> Tempo de execução da ação Login -> 37
>
> listando empresas
>
> Tempo de execução da ação ListaEmpresas -> 308

Portanto, os dois filtros agora funcionam em conjunto. Mas a dúvida é: qual dos dois foi executado primeiro? Vamos testar isso escrevendo um `system.out.println()` para cada um dos filtros, bem no início da classe. Dessa forma, assim que a requisição entrar no filtro iremos imprimir o nome da classe.

```
System.out.println("MonitoramentoFilter");
```

COPIAR CÓDIGO

E também:

```
System.out.println("AutorizacaoFilter");
```

COPIAR CÓDIGO

Quando testamos isso acessando uma URL da aplicação (`ListaEmpresas` , por exemplo), o resultado é algo como:

AutorizacaoFilter

AutorizacaoFilter

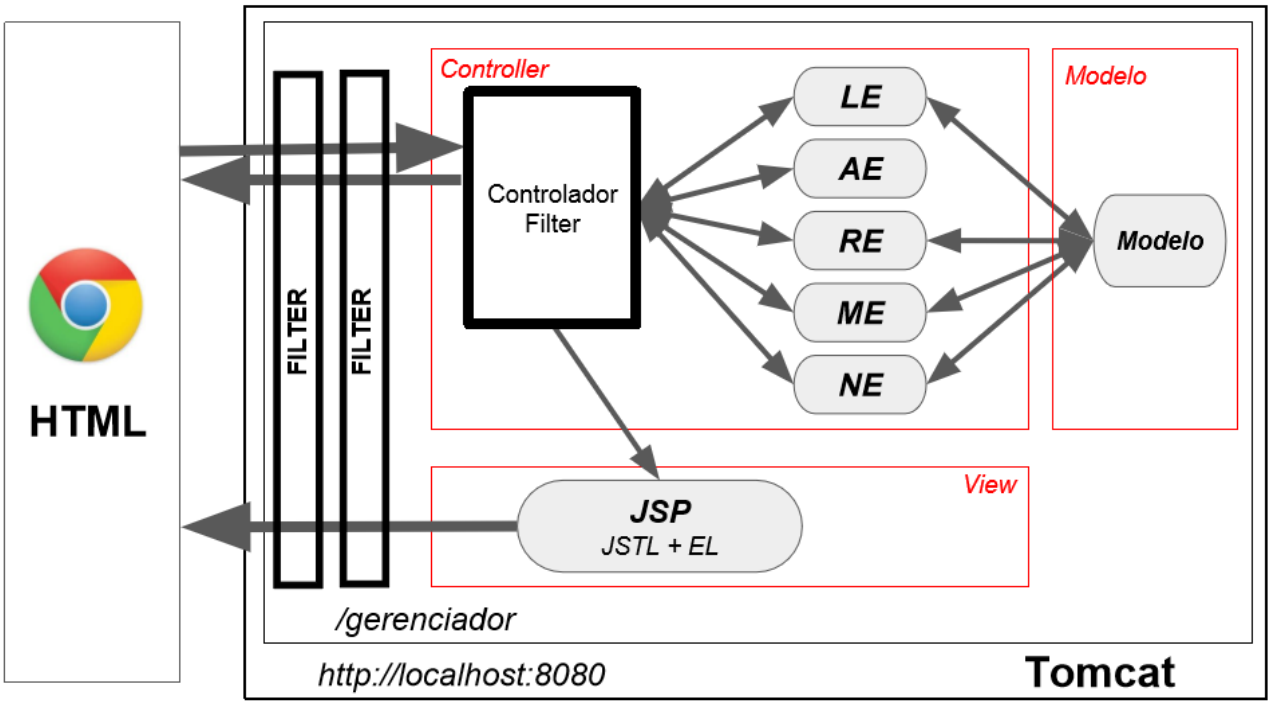
MonitoramentoFilter

Tempo de execução da ação LoginForm -> 270

Ou seja, a requisição passou primeiro pelo `AutorizacaoFilter` . Como não estávamos logados, fomos redirecionados para `LoginForm` . Esse redirecionamento voltou para o navegador e enviou uma nova requisição, que caiu novamente no `AutorizacaoFilter` . No entanto, como a ação agora é `LoginForm` , nós conseguimos passar pela condição `if()` e consequentemente o `MonitoramentoFilter` foi executado. Por isso `AutorizacaoFilter` foi executado duas vezes.

Mas será que poderíamos mudar a ordem dos filtros? Quando usamos anotações, não temos controle sobre a ordem dos filtros. Se queremos uma garantia da ordem de execução, teremos que usar `web.xml` , pois, assim como um Servlet, cada filtro pode ser definido dentro do `web.xml` .

Isso seria necessário, por exemplo, se transformássemos nosso controlador `UnicaEntradaServlet` em um filtro, já que precisaríamos que esse filtro fosse o último na cadeia.



Vamos testar isso no próximo vídeo. Não se esqueça de fazer os exercícios e até lá!