06

Autorizando o acesso

Transcrição

Bem vindo de volta! Até agora nós já conseguimos fazer a verificação de login para nossa `ListaEmpresas` - ou seja, um usuário não consguirá acessar a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas) sem ter feito o login. Porém, as nossas outras ações não estão protegidas.

Como observamos na aula anterior, poderíamos simplesmente copiar o código abaixo e colar em todas as outras ações que gostaríamos de proteger:

```
HttpSession sessao = request.getSession();
        if(sessao.getAttribute("usuarioLogado") == null) {
            return "redirect:entrada?acao=LoginForm";
        }
```

COPIAR CÓDIGO

Porém, sabemos que copiar e colar é sempre uma má prática, já que, por exemplo, se quisermos alterar essa implementação no futuro, também será necessário alterar todas as outras ações.

Nosso objetivo é fazer essa verificação em um único lugar, e qual é o melhor lugar da nossa aplicação para isso, onde todas as requisições irão passar?

Se você pensou na `UnicaEntradaServlet`, acertou em cheio! Aqui temos todas as informações disponíveis para fazermos essa verificação. Vamos construir essa lógica logo no início do Servlet:

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


    protected void service(HttpServletRequest request, HttpSer

        HttpSession sessao = request.getSession();
        if(sessao.getAttribute("usuarioLogado") == null) {
            return "redirect:entrada?acao=LoginForm";
        }
```

COPIAR CÓDIGO

Como já sabemos, estamos pegando `sessao` a partir de `request.getSession()` e checando se o usuário está logado ou nao. Se o atributo `usuarioLogado` for nulo, o usuário não está logado e deve ser redirecionado. Mas repare que nesse caso não devemos usar `return` (que o Eclipse inclusive aponta como erro), mas sim `response.sendRedirect()`:

```
HttpSession sessao = request.getSession();
    if(sessao.getAttribute("usuarioLogado") == null) {
        response.sendRedirect("entrada?acao=LoginForm");
    }
```

COPIAR CÓDIGO

Vamos melhorar ainda mais esse código criando uma variável auxiliar (`usuarioNaoEstaLogado`) que deixa isso mais claro:

```
HttpSession sessao = request.getSession();
    boolean usuarioNaoEstaLogado = (sessao.getAttribute("usuar
    if(usuarioNaoEstaLogado) {
        response.sendRedirect("entrada?acao=LoginForm");
    }
```

COPIAR CÓDIGO

Dessa forma, se o usuário não estiver logado, ele será redirecionado para `LoginForm`. Faz sentido, não? Mas se testarmos isso no navegador, acessando uma URL da nossa aplicação (como `ListaEmpresas`), receberemos um "HTTP Status 500 - Internal Server Error".

Qual é o problema? Vamos verificar o console:

> GRAVE: Servlet.service() for servlet [br.com.alura.gerenciador.servlet.UnicaEntradaServlet] in context with path [/gerenciador] threw exception
>
> java.lang.IllegalStateException: Cannot call sendRedirect() after the response has been committed

Isso significa que o Servlet iria nos redirecionar para `LoginForm`, mas continuou executando e nos devolvendo outra resposta. Dessa forma nossa aplicação não irá funcionar, pois ela deve fazer uma coisa ou outra.

Portanto, queremos que, se o usuário não estiver logado, a execução da aplicação pare abruptamente. Vamos escrever isso. Como estamos trabalhando com o tipo `void`, basta usarmos `return`:

```
HttpSession sessao = request.getSession();
        boolean usuarioNaoEstaLogado = (sessao.getAttribute("us
        if(usuarioNaoEstaLogado) {
            response.sendRedirect("entrada?acao=LoginForm");
            return;
        }
```

COPIAR CÓDIGO

Dessa vez, se tentarmos acessar uma URL da aplicação, teremos:

> Esta página não está funcionando
>
> Redirecionamento em excesso por localhost
>
> Tente limpar os cookies.
>
> ERR_TOO_MANY_REDIRECTS

Além dessa diferença, podemos perceber que o navegador tentou chamar a URL http://localhost:8080/gerenciador/entrada?acao=LoginForm (http://localhost:8080/gerenciador/entrada?acao=LoginForm). Então qual o problema?

Repare que, na nossa aplicação, quando o usuário não está logado ele é redirecionado para `LoginForm`. Dessa forma, o navegador envia uma nova requisiçaõ. Porém, nessa nova requisição o usuário ainda não está logado e é redirecionado novamente para `LoginForm`, o que acontece sucessivamente em um ciclo.

O navegador detecta esse problema e nos retorna um erro. Então precisaremos quebrar esse ciclo, mas como?

Existem ações que não deveriam verificar se o usuário está logado ou não: a ação de `Login` e a ação que chama `LoginForm`, pois, como o usuário está tentando se identificar, elas devem ser de livre acesso, correto?

Para nos auxiliar nessa verificação, vamos criar uma nova variável `ehUmaAcaoProtegida`. No caso, se `ehUmaAcaoProtegida` for verdadeiro e o usuário não estiver logado, queremos que o navegador o redirecione para `LoginForm`.

```
boolean ehUmaAcaoProtegida = false;
        if(ehUmaAcaoProtegida & usuarioNaoEstaLogado) {
            response.sendRedirect("entrada?acao=LoginForm");
            return;
        }
```

COPIAR CÓDIGO

Agora precisaremos saber qual ação está sendo acessada nessa requisição, que é exatamente o que fazemos em `String paramAcao = request.getParameter("acao")`. Portanto, moveremos esse trecho para cima no nosso código, já que precisaremos dele para definirmos a expressão `ehUmaAcaoProtegida`.

Antes de definirmos todas as ações que são protegidas, faremos um teste com `ListaEmpresas`:

```
protected void service(HttpServletRequest request, HttpServletI


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

    HttpSession sessao = request.getSession();
    boolean usuarioNaoEstaLogado = (sessao.getAttribute("usuar
    boolean ehUmaAcaoProtegida = paramAcao.equals("ListaEmpresa

    if(ehUmaAcaoProtegida & usuarioNaoEstaLogado) {
        response.sendRedirect("entrada?acao=LoginForm");
        return;
    }
```

COPIAR CÓDIGO

Dessa vez, se tentarmos acessar a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas), seremos

corretamente redirecionados ao formulário de login. Porém, poderemos acessar normalmente as outras ações, como `MostraEmpresa`, já que ainda não as definimos como ações protegidas.

Entretanto, se repararmos bem, na verdade temos apenas duas ações que não precisam de autenticação: `Login` e `LoginForm`. Portanto, poderiamos fazer o inverso, definindo essas ações como não-protegidas por meio da negação `!()`:

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

COPIAR CÓDIGO

Feito isso, todas as ações que gostaríamos de proteger precisarão de uma autorização, verificando se o usuário fez login ou não. Quando o usuário tiver feito o login, ele conseguirá acessar essas ações normalmente.

Apesar de tudo estar funcionando como gostaríamos, nós colocamos o código de autenticação dentro do nosso *Controller*, o que não é muito elegante.

Quando usamos uma biblioteca de mais alto nível, como VRaptor, Spring MVC ou JSF, não somos nós que escrevemos o controlador. Nesses casos, teríamos que abrir a classe e recompilar o controlador para adicionarmos a autenticação e a verificação, o que não seria possível, já que não mexemos no código dos desenvolvedores desses controladores.

Nosso controlador serve para controlar o fluxo, e não para controlar o usuário. Portanto, precisamos colocar esse código em outro lugar, mas onde? Discutiremos isso no futuro. Até o próximo vídeo!