02

Autenticação de usuários

É comum hoje em dia acessarmos algum site que peça para fazermos nosso login para podermos ter acesso a funcionalidades da aplicação. Esse processo também é conhecido como autenticação. Mas, como o site sabe, nas requisições seguintes que fazemos, quem somos nós?

Cookie

O protocolo HTTP utilizado para o acesso às páginas é limitado por não manter detalhes como quem é quem entre uma conexão e outra. Para resolver isso, foi inventado um sistema para facilitar a vida dos programadores.

Um cookie é normalmente um par de strings guardado no cliente, assim como um mapa de strings. Esse par de strings possui diversas limitações que variam de acordo com o cliente utilizado, o que torna a técnica de utilizá-los algo do qual não se deva confiar muito. Já que as informações do cookie são armazenadas no cliente, o mesmo pode alterá-la de alguma maneira, sendo inviável, por exemplo, guardar o nome do usuário logado.

Quando um cookie é salvo no cliente, ele é enviado de volta ao servidor toda vez que o cliente efetuar uma nova requisição. Desta forma, o servidor consegue identificar aquele cliente sempre com os dados que o cookie enviar. Um exemplo de bom uso de cookies é na tarefa de lembrar o nome de usuário na próxima vez que ele quiser se logar, para que não tenha que redigitar o mesmo. Cada cookie só é armazenado para um website. Cada website possui seus próprios cookies e estes não são vistos em outra página.

Sessão

Usar Cookies parece facilitar muito a vida, mas através de um cookie não é possível marcar um cliente com um objeto, somente com Strings. Imagine gravar os dados do usuário logado através de cookies. Seria necessário um cookie para cada atributo: usuário, senha, id, data de inscrição, etc. Sem contar a falta de segurança. O Cookie também pode estar desabilitado no cliente, sendo que não será possível lembrar nada que o usuário fez. Uma sessão facilita a vida de todos por permitir atrelar objetos de qualquer tipo a um cliente, não sendo limitada somente às strings e é independente de cliente.

A sessão nada mais é que um tempo que o usuário permanece ativo no sistema. A cada página visitada, o tempo de sessão é zerado. Quando o tempo ultrapassa um limite demarcado no arquivo `web.xml`, o cliente perde sua sessão.

Tela de login

Vamos começar então a fazer uso de tudo que aprendemos acima e fazer o login da nossa aplicação. O primeiro passo será o formulário de login, onde o usuário digitará um login e senha e, caso ele tenha acertado, guardaremos na sessão que ele está logado, junto com seu nome de usuário.

O JSP de login não tem segredo. Veja que estamos postando um usuário:

```
<html>
<body>
<h2>Página de Login das Contas</h2>
<form action="efetuaLogin" method="post">
Login: <input type="text" name="login" /> <br />
```

```
Senha: <input type="password" name="senha" /> <br />
<input type="submit" value="Entrar nas contas" />
</form>
</body>
</html>
```

Vamos agora criar o `LoginController` com dois métodos: um que só retorna o formulário, outro que fará o login. Para lidar com a sessão, basta receber `HttpSession` na action. Essa classe possui um método chamado `setAttribute()` que nos permite guardar um objeto e dar um nome qualquer a ele.

```
@Controller
public class LoginController{
  @RequestMapping("/loginForm")
  public String loginForm() {
    return "formulario-login";
  }

  @RequestMapping("/efetuaLogin")
  public String efetuaLogin(Usuario usuario, HttpSession session) {
    if(new UsuarioDAO().existeUsuario(usuario)) {
      // usuario existe, guardaremos ele na session
      session.setAttribute("usuarioLogado", usuario);
      return "menu";
    }
    // ele errou a senha, voltou para o formulario
    return "formulario-login";
  }
}
```

Ao olhar o código acima atentamente, verá que se tudo der certo, redirecionamos para "menu". Vamos criar essa JSP então:

```
<html>
<body>
<h2>Página inicial da Lista de Contas</h2>
<p>Bem vindo, ${usuarioLogado.login}</p>
<a href="listaContas">Clique aqui</a> para acessar a lista de contas
</body>
</html>
```

Veja que fazemos referência a `usuarioLogado` . As variáveis guardadas na sessão ficam também disponíveis em nossa JSP (graças, claro, ao Spring MVC).

Bloqueando usuários não logados

Não podemos permitir que nenhum usuário acesse as contas sem que ele esteja logado na aplicação, pois essa não é uma informação pública. Precisamos, portanto, garantir que antes de executarmos qualquer ação o usuário esteja autenticado, ou seja, armazenado na sessão.

Utilizando o Spring MVC, podemos utilizar o conceito dos **Interceptadores**, que funcionam como Filtros, ou seja, toda requisição antes de ser executada passará por ele. Nele, podemos por exemplo, impedir que a requisição continue se o

usuário não estiver logado.

Para criarmos um Interceptador basta criarmos uma classe que implemente a interface `org.springframework.web.servlet.HandlerInterceptor`. Ao implementar essa interface, precisamos implementar 3 métodos: `preHandle`, `postHandle` e `afterCompletion`. Os métodos `preHandle` e `postHandle` serão executados antes e depois, respectivamente, da ação. Enquanto o método `afterCompletion` é chamado no final da requisição, ou seja, após ter renderizado o JSP.

Para nossa aplicação queremos verificar o acesso antes de executar uma ação, por isso vamos usar apenas o método preHandle da interface `HandlerInterceptor`. Porém, usando a interface `HandlerInterceptor` somos obrigados implementar todos os métodos definidos na interface. Queremos usar apenas o `preHandle`. Por isso o Spring MVC oferece uma classe auxiliar (`HandlerInterceptorAdapter`) que já vem com uma implementação padrão para cada método da interface. Então para facilitar o trabalho vamos estender essa classe e sobrescrever apenas o método que é do nosso interesse.

Nesse interceptor, verificaremos se existe a variável `usuarioLogado` na sessão. Em caso positivo, deixamos a requisição continuar (basta fazer com que o método devolva `true`). Caso contrário, devolvemos `false` e redirecionamos para a página de login. Repare que pegamos a sessão através do objeto request, que vem como parâmetro do método `preHandle`.

Veja o código abaixo:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {
  @Override
  public boolean preHandle(HttpServletRequest request,HttpServletResponse response,
    Object controller) throws Exception {

    if(request.getSession().getAttribute("usuarioLogado") != null) {
      return true;
    }
    response.sendRedirect("loginForm");
    return false;
  }
}
```

Falta só mais uma verificação. Existem duas ações na nossa aplicação que não necessitam de autorização. São as ações do `LoginController`, necessárias para permitir a autenticação do usuário. Além disso, vamos garantir também que a pasta de resources pode ser acessada mesmo sem login. Essa pasta possui as imagens, css e arquivos JavaScript. No entanto a classe `AutorizadorInterceptor` fica:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {
  @Override
  public boolean preHandle(HttpServletRequest request,HttpServletResponse response,
    Object controller) throws Exception {

    // se for a pagina de login ou resources, deixa passar
    String uri = request.getRequestURI();
    if(uri.endsWith("loginForm") || uri.endsWith("efetuaLogin") || uri.contains("resources")){
      return true;
    }

    if(request.getSession().getAttribute("usuarioLogado") != null) {
```

```
        return true;
      }
      response.sendRedirect("loginForm");
      return false;
    }
  }
```

Ainda precisamos registrar o nosso novo interceptador. Mas o Spring MVC não permite que façamos isso via anotações, então usaremos a configuração via XML nesse caso. Já vimos o arquivo `spring-context.xml`, nele vamos usar o tag `mvc:interceptors` para cadastrar a classe `AutorizadorInterceptor`:

```
<mvc:interceptors>
  <bean class="br.com.caelum.contas.interceptor.AutorizadorInterceptor" />
</mvc:interceptors>
```

Pronto. Agora nosso sistema está bloqueado! Só usuários autenticados conseguem acessar. Entenda o fluxo: quando uma requisição chega, ele cai automaticamente em nosso interceptor de autenticação. O interceptor descobre se o usuário está logado; se ele está, ele deixa a requisição continuar como se nada tivesse acontecido. Mas, caso o usuário não esteja logado (ou seja, não existe nada na Sessão), ele redireciona para a página de login!

Interceptors são bastante úteis quando temos tarefas a serem executadas para todas as requisições. Acostume-se com eles!