Modelagem de Hipermídia
Prof. Giangiacomo Ponzo Neto
Aula 5

Cookies

Os "*cookies*" são mecanismos fornecidos pelo HTTP que permitem gravar/recuperar pequenas quantidades de dados persistentes no navegador do usuário e podem ser configurados para persistir por vários dias na máquina cliente. Um *cookie* é composto por um nome e algum dado associado. Uma página JSP pode criar ou alterar *cookies* através do método addCookie() do objeto implícito "response" e recuperá-los através do método getCookies() do objeto implícito "request". Veja o exemplo abaixo, que altera o nosso programa anterior (Entrada.jsp) – não se esqueça de compilar depois (F9):

```
<% String usuario=request.getParameter("txtUsuario");
   String senha= request.getParameter("txtSenha");
%>
<html>
<body>
  <% if ( (usuario.equals("abc")) && (senha.equals("123")) ) {
       Cookie cookieUsuario = new Cookie("usuario",usuario);
       response.addCookie(cookieUsuario);
       response.sendRedirect("Outra.jsp");
     }
     else {
  %>
       Cai fora!
  <%
     }
  %>
</body>
</html>
```

O construtor da classe Cookie recebe duas variáveis do tipo String, que representam o nome e o valor do cookie. Alteramos também a página Outra.jsp para recuperar o cookie:

```
<%
  Cookie meusCookies[]=request.getCookies();
  String usuario="";
  for (int i=0;i<meusCookies.length;i++) {
    if (meusCookies[i].getName().equals("usuario"))
      usuario=meusCookies[i].getValue();
  }
%>
<html>
<body>
  Bem-vindo <%= usuario %>!
</body>
</html>
```

Compile (F9), volte ao login.html e execute-o (SHIFT F6).

Se o *cookie* já tiver expirado (ou não tiver sido gravado) ele retorna nulo. Adicionalmente, podemos trabalhar com outros métodos da classe Cookie. Alguns interessantes:

- setValue("string") - atribui um novo valor para o cookie.
- setMaxAge(inteiro) - define o tempo restante (em segundos) antes que o cookie expire.
- setComment("string") - insere um comentário para o cookie.

Depois de recuperar o *cookie* com o método getCookies( ), é necessário usar o método addCookie( ) do objeto response, para salvar as alterações feitas, se for o caso. Para apagar um *cookie*, usamos o método "setMaxAge(0)" com valor zero e depois mandamos gravar chamando o método "addCookie( )". Isso faz com que o *cookie* seja gravado e imediatamente (após zero segundos) expirado.

Apesar da facilidade de uso, os *cookies* têm alguns pontos contra:

- os *cookies* podem ser desabilitados no navegador cliente por questões de segurança ou privacidade.

- ? o tamanho dos dados armazenados é limitado.
- ? o navegador tem um limite de *cookies* por domínio ou no total.

Sessões

Uma sessão é um objeto temporário associado a cada usuário e sua manutenção é feita no servidor. Dados podem ser escritos e lidos na sessão a fim de manter o estado de transações e do sistema como um todo. Vamos reescrever o exemplo anterior (o login.html continua igual) com o propósito de salvar quem é o usuário em uma sessão. Altere "Entrada.jsp" conforme abaixo:

```
<% String usuario=request.getParameter("txtUsuario");
   String senha= request.getParameter("txtSenha");
   session.setAttribute("usuario",usuario );
%>
<html>
<body>
  <% if ( (usuario.equals("abc")) && (senha.equals("123")) ) {
       response.sendRedirect("Outra.jsp");
     }
     else {
  %>
       Cai fora!
  <%
     }
  %>
</body>
</html>
```

Compile o arquivo (F9). Repare que guardamos na sessão quem é o usuário (a senha não foi guardada, mas, se fosse necessário, era só proceder de maneira semelhante). Depois enviamos o usuário (caso tenha entrado corretamente) para a página Outra.jsp, que deve ser alterada conforme abaixo:

```
<html>
<body>
  Bem-vindo <%= session.getAttribute("usuario") %>!
</body>
</html>
```

Compile (F9), volte ao login.html e execute-o (SHIFT F6). Repare que, para encontrar o usuário, se procurou na sessão, direto e sem maiores complicações. Cada entrada no sistema de um mesmo navegador gera uma nova sessão. Se usarmos dois navegadores diferentes ao mesmo tempo (IE e Firefox, por exemplo) podemos entrar com dois usuários diferentes. O uso de sessões é simples e elegante, mas o problema é que elas só duram um certo tempo (dependendo de configuração no servidor usado), após o qual, se não houver atividade do usuário, são terminadas (por *timeout*). Se o usuário, após o *timeout*, tentar usar mais alguma coisa da sessão, é enviada uma mensagem de erro pelo servidor.

Beans

A última forma que veremos de manter dados globais é definir um "bean". Um *Java bean* é uma classe com atributos privados e seus métodos *get/set*. No nosso caso, esse *bean* será definido contendo um atributo correspondente a cada componente do *form* HTML associado (*text*, *password*, *checkbox*, etc.) e seus métodos *get*/*set*. Baseado no exemplo que estamos usando, repare o nome dos atributos no *bean* definido abaixo:

```
package beans;

public class LoginBean {
    String txtUsuario;
    String txtSenha;

    public void setTxtUsuario(String u) {
      txtUsuario=u;
    }

    public void setTxtSenha(String s) {
      txtSenha=s;
```

```
    }

    public String getTxtUsuario() {
      return txtUsuario;
    }

    public String getTxtSenha() {
      return txtSenha;
    }
}
```

Crie este *bean* (Arquivo-Novo Arquivo-Classes Java-Classe Java, nome “LoginBean”, pacote “beans”) e compile (F9). Repare a primeira linha: o *package* é necessário para indicar um direcionamento para o JSP encontrar o *bean* (se a aplicação estiver em um servidor externo, deve-se saber o *classpath* para colocar lá os *beans*, o que vai depender do servidor usado). Obs.: o nome do *package* pode ser qualquer nome válido – não precisa ser *beans*. Veja as alterações necessárias em "`Entrada.jsp`" para usar o *bean* acima:

```
<%@ page import="beans.LoginBean" %>
<jsp:useBean id="login" scope="session" class="beans.LoginBean" />
<jsp:setProperty name="login" property="*" />
<html>
<body>
  <% if ( (login.getTxtUsuario().equals("abc")) &&
                        (login.getTxtSenha().equals("123")) ) {
       response.sendRedirect("Outra.jsp");
     }
     else {
  %>
       Cai fora!
  <%
     }
  %>
</body>
</html>
```

Compile (F9). Verifique o uso do *import* (a classe LoginBean está no *package beans*, que é externo ao JSP, por isso precisa do *import*) e de duas novas *tags*: `jsp:useBean` e `jsp:setProperty`. A primeira procura pela classe LoginBean na sessão (para ser mais exato, o *bean* é um objeto - uma instância de LoginBean - mantido no servidor). A segunda *tag* é que faz todo o trabalho de pegar os dados de entrada do *form*, bater com os nomes da classe e disponibilizar os métodos públicos para a página JSP.

Finalmente, altere “Outra.jsp”:

```
<%@ page import="beans.LoginBean" %>
<jsp:useBean id="login" scope="session" class="beans.LoginBean" />
<jsp:setProperty name="login" property="*" />
<html>
<body>
  Bem-vindo <%=login.getTxtUsuario()%>!
</body>
</html>
```

Compile (F9), volte ao login.html e execute-o (SHIFT F6).

Conclusão

Vimos quatro formas de implementar algo semelhante: a manutenção de conteúdo durante a visita de um usuário ao “sistema”. Não existe uma forma “melhor”, todas têm seus pontos a favor e contra. Alguns julgam a última (usar *Java beans*) a mais elegante, pois, mesmo sendo mais trabalhosa, permite a sua reutilização em vários locais, seguindo o padrão OO.

Obs.: Repare que a passagem de conteúdo de login.html para Entrada.jsp foi sempre feita usando a primeira forma de transferência, pois o HTML não dispõe de outra maneira de passar valores para o JSP, senão via o *method* do *form*. Todas as 3 outras formas de manutenção de conteúdo só podem ser usadas internamente ao JSP.