Desenvolvimento Web I

Aula 04 - Trabalhando com sessões

Apresentação

Olá, caro aluno! Você está gostando do assunto? Espero que sim, e ainda estamos nos primeiros passos da programação para web! Lembre-se, qualquer dúvida, não hesite em reler as aulas anteriores e utilizar outros recursos, como os meios de comunicação com os tutores.

Muito bem, vamos agora avançar no assunto, dando continuidade à aula anterior. Nela, você aprendeu a receber e repassar os valores de parâmetros de uma tela para outra, não foi? Apesar de essa abordagem ser útil em algumas situações, você viu que o repasse de parâmetros de requisições web possui limitações, como o trabalho extra de codificação, o perigo de transitar entre as requisições de dados sensíveis (senhas, dados bancários etc.) e a perda das informações, caso a máquina do cliente trave.

Além disso, existem situações nas quais você irá querer identificar o usuário que está conectado no sistema, mesmo quando não há informações a serem trafegadas entre páginas, como foi o caso do exemplo mostrado na aula anterior. Lembre-se de que estamos trabalhando com sistemas web, ou seja, sistemas que podem estar sendo acessados por dezenas ou até milhares de usuários ao mesmo tempo. Imagine quantos usuários estão acessando um dos serviços do Google – correio eletrônico, sistema de busca, Google+, entre outros, neste exato momento, ao mesmo tempo?

Nesta aula, você aprenderá um recurso elaborado e muito útil, chamado de sessão, o qual pode evitar as limitações da abordagem de repasse de parâmetros vista na aula anterior. Mas, lembre-se, apesar do recurso que você irá aprender agora ser o mais interessante na maioria das situações, o anterior continua válido para situações simples, em que a passagem de parâmetros de uma tela para outra se mostra adequada.

Vídeo 01 - Apresentação

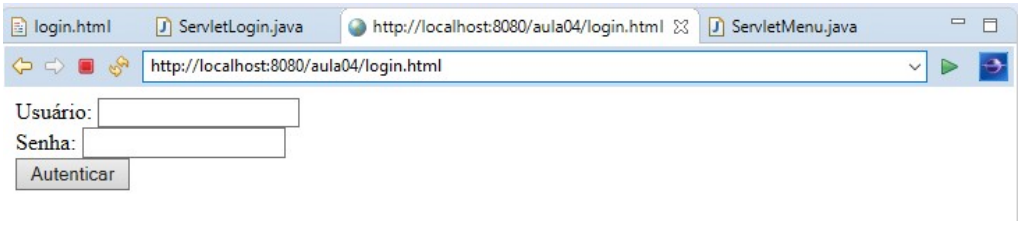
Objetivos

- Entender o funcionamento das sessões em aplicações web.
- Utilizar esse recurso na programação web utilizando a linguagem Java.

Um sistema de autenticação

Para apresentar os conceitos da aula de hoje, faremos uso de um sistema de autenticação, além do próprio sistema de cadastro de dados pessoais e profissionais visto na aula passada. Sobre a parte de autenticação, a ideia é que o usuário precise entrar com sua credencial (combinação de nome de usuário e senha válidos) antes de poder realizar operações no sistema de cadastro. A tela inicial desse sistema de autenticação pode ser vista na Figura 1.

Figura 01 - Tela inicial do sistema de autenticação



Após entrar com seu nome de usuário e senha, o sistema de autenticação irá verificar se a combinação de nome de usuário e senha é válida. Caso a combinação seja válida, o usuário visualizará uma página de entrada com uma mensagem de saudação personalizada (ver Figura 2). Se a combinação não for válida, uma mensagem de erro será apresentada (Figura 3).

Figura 02 - Autenticação do usuário Jose

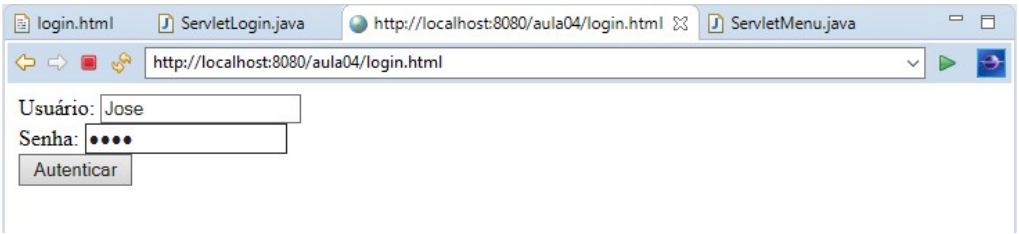
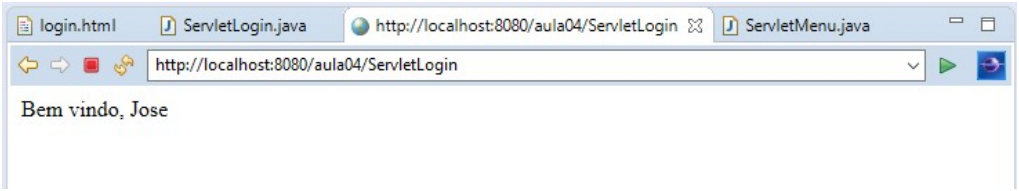


Figura 03 - Tela do sistema quando a combinação de nome de usuário e senha não é válida



Sessões de usuário

Para implementar o sistema de autenticação que lhe apresentamos, faremos uso do conceito de sessão do usuário. A ideia de sessão é basicamente a de manter o estado do cliente (ou melhor, das informações sobre o cliente) no próprio servidor. Lembre-se de que na aula anterior o estado do cliente (dados pessoais, profissionais e bancários) ficava no lado cliente, ou seja, no navegador web. Usamos os campos ocultos para isso, não foi? Caso o navegador do cliente fosse fechado, perderíamos todos os dados.

Pois bem, para manter o estado do cliente no lado do servidor, precisamos de uma classe Java para representar esse estado. O nome dessa classe (na verdade, interface) é javax.servlet.http.HttpSession. Cada usuário deverá ter um e apenas um objeto do tipo HttpSession para representar o seu estado (informações). Um usuário, para nós, é um programa que está acessando o servidor, ou seja, geralmente um navegador web (cliente do servidor) que está sendo operado por um ser humano (o cliente do negócio, consumidor de uma loja virtual, etc.).

Para criar um objeto sessão para representar um cliente, você deve fazer uso do método getSession() da classe HttpServletRequest. Esse método tem o seguinte comportamento:

Vídeo 02 - Introdução aos Cookies

- se não existe nenhum objeto sessão para representar o cliente web em atendimento, esse método cria um objeto do tipo HttpSession e relaciona esse objeto ao cliente web em questão;
- caso já exista um objeto sessão relacionado ao cliente web, esse método simplesmente retorna o objeto.

Você pode estar se perguntando agora como é que se relaciona um objeto a um cliente web. Vamos pensar bem, se tivermos 30 usuários acessando o sistema ao mesmo tempo, quando chega uma requisição web, como podemos identificar um determinado usuário? Uma primeira solução seria observar o endereço IP da máquina do cliente (forma de endereçamento dos computadores na Internet). Isso pode ser visto no código mostrado na Listagem 1. A classe HttpServletRequest possui vários métodos para acessarmos informações sobre a requisição. Uma dessas informações é o endereço IP ou nome do host de onde vem a requisição (ver métodos getRemoteAddr() e getRemoteHost() da Listagem 1).

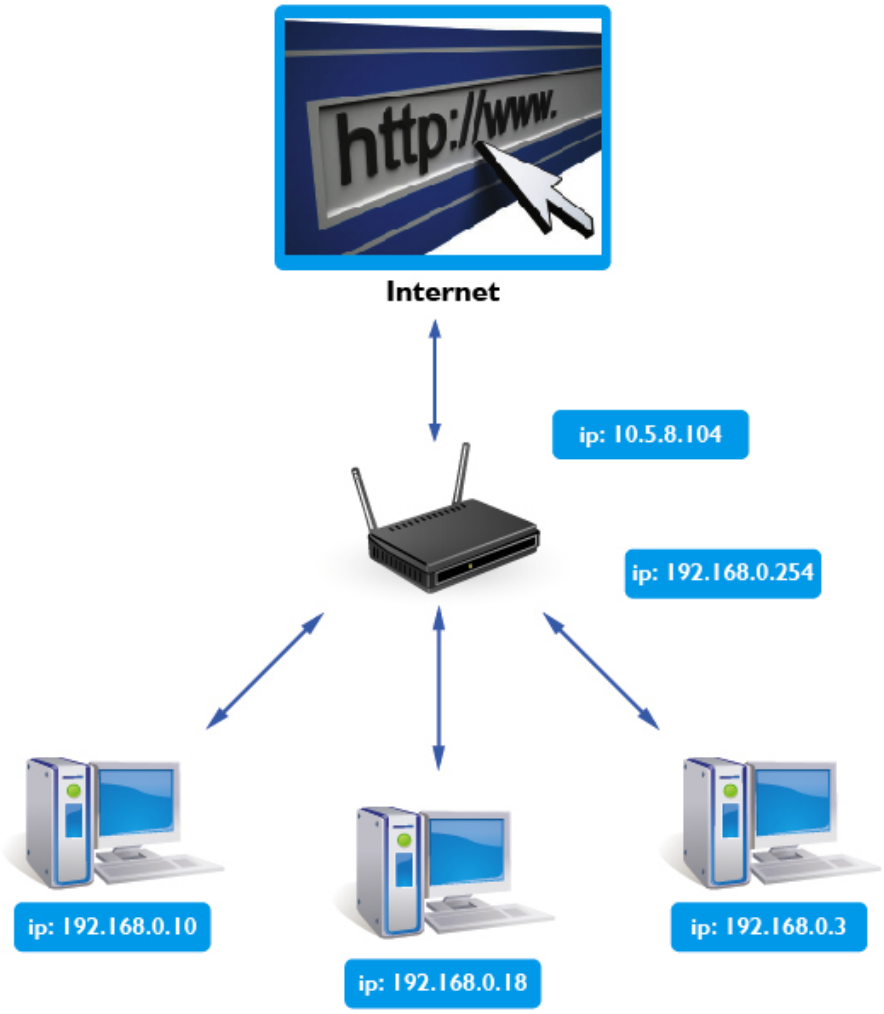
Vídeo 03 - Criando os Cookies

```
1  package aula04;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.annotation.WebServlet;
8  import javax.servlet.http.HttpServlet;
9  import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 @WebServlet("/ServletMostraIp")
13 public class ServletMostraIp extends HttpServlet {
14    private static final long serialVersionUID = 1L;
15
16    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servlet
17       doPost(request, response);
18    }
19
20    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
21       PrintWriter resposta = response.getWriter();
22       resposta.write("IP do cliente: ");
23       resposta.write(request.getRemoteAddr());
24       resposta.write("\n Host do cliente: ");
25       resposta.write(request.getRemoteHost());
26    }
27 }
```

Listagem 1 - Servlet que mostra informações sobre máquina do cliente

Porém, esses métodos getRemoteAddr() e getRemoteHost() só serão úteis para identificar os clientes web em alguns casos, quando seu Servlet estiver rodando em um servidor web e os clientes web rodando em máquinas na Internet com IP real. Por exemplo, se várias máquinas estão conectadas a um roteador e esse roteador está conectado à internet, os acessos de todas essas máquinas irão aparecer com o IP do roteador diretamente ligado à internet, pois apenas o seu IP é que é “real” (visível por outras máquinas da internet). Isto pode ser visto através da Figura 4, que mostra o IP de um roteador que liga 3 computadores à internet. Apenas o IP externo/real (no exemplo, o IP 10.5.8.104) do roteador é visível. Nem o IP interno do roteador nem o dos computadores são visíveis pelos servidores web que rodam na Internet.

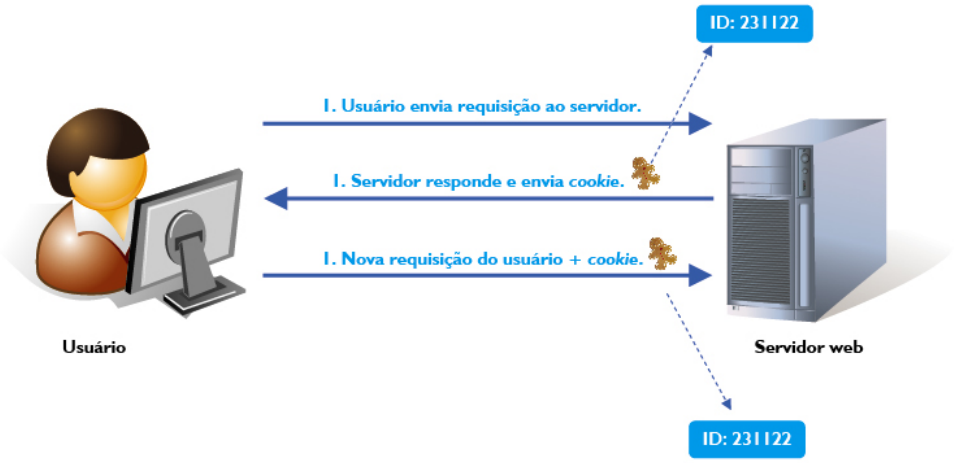
Figura 04 - IP real e virtual



Fonte: autoria própria.

A saída encontrada para o problema de identificação do cliente web é dada pelos chamados *cookies* (biscoitos, em inglês). Os cookies HTTP são pequenos textos armazenados no navegador web. O texto de um cookie consiste em um ou mais pares de identificador/valor. Um exemplo seria guardar em um cookie do navegador o texto ID=231122, onde o valor 231122 foi gerado aleatoriamente pelo servidor para representar aquele cliente específico. Uma vez criado o cookie no navegador, ele será enviado para o servidor durante toda a comunicação a ser realizada com ele. Dessa forma, ao receber uma requisição, o servidor poderá identificar o cliente pelo ID armazenado no cookie do cliente. Visualize este processo através da Figura 5.

Figura 05 - Uso de cookies para identificar usuário ao longo da comunicação web



De fato, cada cookie está associado a um determinado sítio na internet. Sendo assim, o sistema do Banco do Brasil, por exemplo, pode gerar um número aleatório para um usuário após ele ter sido autenticado e mandar esse identificador para ser armazenado no navegador web. Reforçando, em todo acesso que for feito dali em diante, o navegador web irá enviar esse identificador para o servidor. Dessa forma, o servidor poderá saber qual usuário está acessando o sistema naquele momento. Ficou claro? Esse número aleatório que é gerado para cada usuário é, basicamente, o identificador único da sessão do usuário. Por questões de segurança, cada servidor só poderá ler os cookies associados ao seu domínio.

Vamos voltar novamente para a criação dos objetos HttpSession. Ao ser executado, o método getSession() irá, de forma automática (ou seja, é tudo transparente para o programador), enviar na resposta do Servlet uma instrução indicando um cookie que deverá ser criado, cujo valor é um identificador único para o cliente e o qual será utilizado nas demais requisições para fazer a identificação desse cliente. Dessa forma, quando usamos sessões, estamos dizendo para o servidor e o navegador web criarem um vínculo, e isso tudo é feito de forma transparente e automática para nós!

Vamos ver agora esse assunto em prática. Vejamos como pode ser implementada a funcionalidade de autenticação (login) do usuário no exemplo a seguir.

Exemplo 1

Em primeiro lugar, temos na Listagem 2 o código fonte da página HTML de autenticação (login.html) ilustrada anteriormente pela Figura 1.

```
1  <html>
2    <body>
3      <form action="ServletLogin" method="post">
4        Usuário: <input type="text" name="usuario" /><br>
5        Senha: <input type="password" name="senha" /><br>
6        <input type="submit" value="Autenticar" />
7      </form>
8    </body>
9  </html>
```

Listagem 2 - Código HTML da página de autenticação do usuário

O formulário de autenticação de usuário está direcionado para um Servet de nome ServletLogin, cujo código fonte é visto na Listagem 3. Observe que os comandos das linhas 26 e 27 são responsáveis por pegar o nome de usuário e senha informados pelo cliente. Esses dados são então validados pelo método autenticar (linhas 39 a 44). Por estar fora do assunto desta aula, essa validação apenas verifica se o nome de usuário e senha não são vazios, nem nulos. Em um sistema real, esse método iria verificar, por exemplo, se os dados enviados pelo usuário estão de acordo com os dados cadastrados em uma base de dados.

Caso o método autenticar() retorne o valor true, isso quer dizer que o nome de usuário e senha são válidos. Nesse caso, a sessão do usuário (linha 30) é criada, caso ainda não exista, e a informação do nome do usuário é armazenada nela, através do método setAttribute() (linha 31). O método setAttribute() recebe dois parâmetros, o nome do atributo e o seu valor. Já havíamos falado que a sessão guarda o estado do cliente, não é? Muito bem, o estado nada mais é do que esse mapeamento de “atributos” do cliente nos seus respectivos valores. No caso, cada sessão vai mapear o atributo de nome “usuario” (note o uso da constante USUARIO) para o nome de usuário utilizado pelo cliente para se autenticar no sistema. Ao autenticar o usuário, uma mensagem de boas- vindas será apresentada (linha 32). Caso contrário, uma mensagem de nome de usuário e senha inválidos é mostrada (linha 34).

```java
package aula04;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

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

    public static final String USUARIO = "usuario";

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
        PrintWriter resposta = response.getWriter();
        resposta.write("<html><body>");
        String nomeUsuario = request.getParameter(USUARIO);
        String senhaUsuario = request.getParameter("senha");
        // Login e senha corretos
        if (autenticar(nomeUsuario, senhaUsuario)) {
            HttpSession sessao = request.getSession();
            sessao.setAttribute(USUARIO, nomeUsuario);
            resposta.write("Bem vindo, " + nomeUsuario);
        } else {
            resposta.write("Usuário e senha inválidos");
        }
        resposta.write("</body></html>");
    }

    private boolean autenticar(String nomeUsuario, String senhaUsuario) {
        // Aqui entraria o código de autenticação.
        // Está verificando apenas se é diferente de nulo ou vazio
        return !("".equals(nomeUsuario) || "".equals(senhaUsuario)
                    || nomeUsuario == null || senhaUsuario == null);
    }
}
```

Listagem 3 - Código do Servlet de autenticação

Até agora, tudo bem? Faça as atividades e reveja o conteúdo visto até agora, se necessário.

Atividade 01

1. Implemente a página HTML de autenticação e o ServletLogin, de acordo como mostrado no exemplo que resolvemos juntos. Novamente, tente fazer sem olhar as listagens apresentadas, tirando dúvidas somente quando necessário. Execute o sistema de autenticação implementado, testando ambos os casos de usuário e senha inválidos e de usuário autenticado.
2. Altere o ServletLogin para autenticar apenas o usuário de nome “jose” e senha “12345”. Execute novamente o sistema e certifique-se de que apenas esse usuário está tendo acesso ao sistema. Dica: compare os parâmetros recebidos com as Strings “jose” e “12345”.

Usando a sessão em outras partes do sistema

Ok, vamos em frente ao assunto. Vejamos agora como o objeto sessão pode ser utilizado pelas demais páginas do sistema. Uma vez autenticado, o cliente pode agora realizar as atividades disponibilizadas pelo sistema. Para isso, vamos fazer uma alteração no ServletLogin. Vamos trocar o código após a linha 34 da Listagem 3 (mensagem de boas-vindas dentro do condicional de sucesso na autenticação) por um comando de redirecionamento:

```
request.getRequestDispatcher(“ServletMenu”).forward(request, response);
```

O comando apresentado faz com que a responsabilidade de resposta para a requisição sendo atualmente processada seja transferida (método forward()) para o ServletMenu (método getRequestDispatcher()). Ou seja, tudo o que for escrito na resposta (PrintWriter de resposta) será descartado e a resposta final para o usuário será aquela apresentada pelo Servlet cujo nome é indicado no parâmetro do

método getRequestDispatcher(). E qual a ideia de transferir a responsabilidade de montar a resposta para outro Servlet? Uma vez o usuário autenticado, vamos agora apresentar as funções que ele pode executar no sistema. Isso poderia ser feito pelo próprio ServletLogin, mas teríamos uma mistura de funcionalidades (autenticação de usuário e apresentação de menu de funcionalidades). Sendo assim, é interessante optar por separar esse código todo em dois Servlets, ok?

Vamos então considerar a funcionalidade do sistema visto na aula anterior: aquele cadastro de dados pessoais e financeiros realizados em duas telas de cadastro, está lembrado? Releia a aula anterior, se necessário. Um ServletMenu pode ser implementado como mostra a Listagem 4. Observe que na linha 22 nós tentamos recuperar a sessão do usuário. Opa, mas agora o método getSession() de HttpServletRequest está recebendo um boleano como parâmetro! E é isso mesmo! Existem dois métodos getSession() na classe HttpServletRequest. O que não recebe parâmetro você já conhece, já o que recebe um boleano tem o funcionamento descrito a seguir.

- Caso o valor passado para o método seja o valor true, o método irá se comportar da mesma forma que o método getSession() que não recebe parâmetros.

- Por outro lado, se o valor passado for false (como no caso do ServletMenu), o método irá retornar à sessão associada ao cliente, se ela já tiver sido criada, ou o valor null, caso contrário. Em outras palavras, o método getSession(false) nunca irá criar uma sessão, apenas retornar uma criada previamente, se a mesma existir.

Atenção!

Se não ficou claro a importância de usar o método getSession(false), considere que um usuário mal intencionado descubra o link direto para o ServletMenu. Ele então pode pular a etapa de autenticação e executar diretamente o ServletMenu, tendo acesso ao conteúdo que pode não ter sido criado para que ele tenha acesso. Dessa forma, o ServletMenu é também responsável por verificar se o cliente que o está acessando está realmente autenticado ou não. Se não houver uma sessão criada previamente, é porque ele provavelmente não foi autenticado! Se não for isso, então o servidor foi reiniciado (os objetos sessão são perdidos) ou a sessão expirou (ainda vamos falar sobre isso). Vale lembrar que esse método de proteção não é uma solução final para uma segurança ideal. Vários outras técnicas, como criptografia SSL, por exemplo, podem ser utilizadas para que seu sistema esteja totalmente protegido. Para efeitos práticos de aprendizagem não estaremos cobrindo amplamente o assunto de criptografia, mas você pode se certificar se a empresa que hospedará seus sistemas suporta essas tecnologias.

Recapitulando, se você precisa que um usuário se autentique em um sistema, você pode fazer com que somente o Servlet responsável pela autenticação possa criar uma sessão para o cliente. Outra opção, caso você queira também manter o estado de clientes ainda não autenticados, é deixar qualquer Servlet criar sessões para usuários, mas deixar apenas o Servlet de autenticação como responsável por criar o atributo “usuario”, indicando que ele já foi autenticado. Note que estamos lhe mostrando agora duas verificações que precisamos fazer para identificar se um usuário está realmente autenticado: (i) seu objeto de sessão deve estar criado e (ii) o atributo “usuario” deve estar preenchido (linha 23). Ser rigoroso com segurança é sempre bom!

Vídeo 04 - Visualizar Cookies

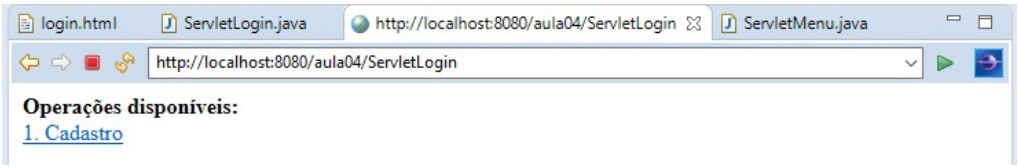
```
1  package aula04;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.annotation.WebServlet;
8  import javax.servlet.http.HttpServlet;
9  import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11 import javax.servlet.http.HttpSession;
12
13 @WebServlet("/ServletMenu")
14 public class ServletMenu extends HttpServlet {
15    private static final long serialVersionUID = 1L;
16
17    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servle
18       doPost(request, response);
19    }
20
21    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
22       PrintWriter resposta = response.getWriter();
23       resposta.write("<html><body>");
24       HttpSession sessao = request.getSession(false);
25       if (sessao == null || sessao.getAttribute(ServletLogin.USUARIO) == null) {
26          resposta.write("<a href=\"login.html\">Faça primeiro o seu login</a><BR>");
27       } else {
28          resposta.write("<b>Operações disponíveis:</b><BR>");
29          resposta.write("<a href=\"cadastro.html\">1. Cadastro</a><BR>");
30          // Novas opções de funcionalidades entram aqui!
31       }
32       resposta.write("</body></html>");
33    }
34
35 }
```

Listagem 4 - Código do ServletMenu, o qual mostra as funcionalidades disponíveis para o usuário

Pois bem, se o usuário não estiver autenticado, então, uma mensagem é apresentada para o usuário (linha 26) indicando que o mesmo deve se autenticar. Caso contrário, um menu de funcionalidades é apresentado (linhas 28 a 29). Note que novas funcionalidades podem ser adicionadas na linha 30, adicionando-se código similar ao da linha 29. No caso, a página gerada pelo ServletMenu aponta para a página inicial do sistema de cadastro desenvolvido na última aula (linha 29), então, o cadastro estará funcionando usando a abordagem antiga (passagem de parâmetros).

A tela resultante com o menu de opções é vista na Figura 6.

Figura 06 - Tela de funcionalidades disponíveis para o usuário



Atividade 02

1. Altere a classe ServletLogin e implemente a classe ServletMenu conforme apresentamos para você, tentando não olhar para o código mostrado, a não ser em caso de dúvidas. Nesse caso, releia o assunto, não só o código, ok? Execute o sistema e verifique se seu comportamento está funcionando de acordo com o que deveria.

2. Altere a programação visual das páginas do sistema, dando um caráter personalizado. As telas mostradas durante as aulas são bem simples, então, tenho certeza que você conseguirá torná-las mais agradáveis! Explore o seu conhecimento obtido na disciplina de autoria web, relendo aquele material sempre que necessário.

3. Altere o ServletMenu para dar uma mensagem de boas-vindas no caso de um usuário já autenticado.

Usando a sessão em outras partes do sistema II

Vamos agora alterar o sistema de cadastro desenvolvido na aula passada, na qual tínhamos usado a abordagem de repasse de parâmetros, para considerar sua implementação através do uso de sessões. Pois bem, para você não apagar o código já criado, ou seja, mantenha o projeto aula03 como está e simplesmente copie os códigos necessários assim que solicitado. Vamos criar um novo Servlet no projeto atual (da aula 04) e chamá-lo de ServletTela1CadastroSessao. Copie o arquivo cadastro.html da aula passada para o projeto dessa aula e altere o formulário web da página cadastro.html para apontar para esse novo Servlet:

```
<form action="ServletTela1CadastroSessao">
```

Depois, copie a implementação dos métodos doGet() e doPost() do ServletTela1Cadastro (da aula passada) para o ServletTela1CadastroSessao (dessa aula). A página gerada pelo ServletTela1CadastroSessao precisa ter um formulário que aponte para ServletTela2CadastroSessao, um novo Servlet que também precisa ser criado. Além disso, precisaremos receber os dados que vêm da primeira tela (dados pessoais) e colocá-los em algum lugar temporário no servidor. E qual é esse lugar? O objeto sessão!

Porém, para estruturar melhor os dados a serem recebidos, é melhor criar uma classe para agrupá-los, como mostrado a seguir, na Listagem 5.

```java
package aula04;

public class DadosPessoais {

   private String nome;
   private String sobrenome;
   private String rua;
   private String complemento;
   private String cidade;
   private String cep;
   private String estado;
   public String getNome() {
         return nome;
   }
   public void setNome(String nome) {
         this.nome = nome;
   }
   public String getSobrenome() {
         return sobrenome;
   }
   public void setSobrenome(String sobrenome) {
         this.sobrenome = sobrenome;
   }
   public String getRua() {
         return rua;
   }
   public void setRua(String rua) {
         this.rua = rua;
   }
   public String getComplemento() {
         return complemento;
   }
   public void setComplemento(String complemento) {
         this.complemento = complemento;
   }
   public String getCidade() {
         return cidade;
   }
   public void setCidade(String cidade) {
         this.cidade = cidade;
   }
   public String getCep() {
         return cep;
   }
   public void setCep(String cep) {
         this.cep = cep;
   }
   public String getEstado() {
         return estado;
   }
   public void setEstado(String estado) {
```

```
52          this.estado = estado;
53      }
54
55  }
```

Listagem 5 - Código-fonte da classe DadosPessoais

Cada parâmetro da requisição será armazenado em um atributo da classe DadosPessoais. Por exemplo, o nome do usuário será armazenado no atributo nome da classe DadosPessoais. A mesma coisa acontece para os demais parâmetros da requisição a serem armazenados na sessão do usuário.

Sendo assim, o nosso ServletTela1CadastroSessao será implementado de acordo com o código mostrado na Listagem 6. Note na linha 45 que o formulário agora referencia o ServletTela2CadastroSessao ao invés do antigo ServletTela1Cadastro. Note também a verificação de que o usuário está autenticado no sistema (linhas 31 e 32). Essa verificação deve ser feita por todos os Servlets que implementem operações as quais exijam que o usuário esteja autenticado.

```java
package aula04;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/ServletTela1CadastroSessao")
public class ServletTela1CadastroSessao extends HttpServlet {
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servle
      doPost(request, response);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
      PrintWriter resposta = response.getWriter();
      resposta.write("<html>");
      resposta.write("<head>");
      resposta.write("<title>Dados Profissionais</title>");
      resposta.write("</head>");
      resposta.write("<body>");
      HttpSession sessao = request.getSession(false);
      if (sessao == null
        || sessao.getAttribute(ServletLogin.USUARIO) == null) {
           resposta.write(
           "<a href=\"login.html\">Faça primeiro o seu login</a><BR>");
      } else {
         DadosPessoais dados = new DadosPessoais();
         dados.setNome(request.getParameter("nome"));
         dados.setSobrenome(request.getParameter("sobrenome"));
         dados.setRua(request.getParameter("rua"));
         dados.setComplemento(request.getParameter("complemento"));
         dados.setCidade(request.getParameter("cidade"));
         dados.setCep(request.getParameter("cep"));
         dados.setEstado(request.getParameter("estado"));
         sessao.setAttribute("dadosPessoais", dados);

         resposta.write("Preencha seus dados profissionais:");
         resposta.write("<form action=\"ServletTela2CadastroSessao\" method=\"POST\">");
         resposta.write("Empresa:<input type=\"text\" name=\"empresa\"> <BR>");
         resposta.write("Endereço profissional:<BR>");
         resposta.write("Rua: <input type=\"text\" name=\"ruaEmpresa\">");
         resposta.write("Complemento:<input type=\"text\" name=\"complementoEmpresa\"><BR>")
         resposta.write("Cidade:<input type=\"text\" name=\"cidadeEmpresa\">");
         resposta.write("CEP:<input type=\"text\" name=\"cepEmpresa\">");
```

```
52          resposta.write("Estado:<input type=\"text\" name=\"estadoEmpresa\"><BR>");
53          resposta.write("<input type=\"submit\" value=\"Confirmar\"><BR>");
54          resposta.write("</form>");
55          resposta.write("</body></html>");
56      }
57  }
58  }
```

Listagem 6 - Código-fonte do ServletTela1CadastroSessao

Finalmente, o código da linha 34 a 42 é responsável por pegar os dados da requisição, agrupá-los no objeto do tipo DadosPessoais e armazená-lo na sessão do usuário. Basicamente, isso é feito por comandos como o da linha 36, onde o valor do parâmetro da requisição (retorno do método getParameter("sobrenome")) é utilizado para setar o atributo do objeto DadosPessoais (método setSobrenome()). No final de tudo, é necessário armazenar o objeto DadosPessoais na sessão (linha 42).

Para o ServletTela2CadastroSessao (sucessor do antigo ServletTela2Cadastro), precisaremos criá-lo para receber os dados da segunda tela de cadastro (dados profissionais), recuperar os dados pessoas da sessão do usuário e, finalmente, montar a tela de confirmação de cadastro. Vamos ver como fica seu código? Ele está mostrado na Listagem 7.

Note o uso do bloco de código das linhas 30 a 32, responsáveis por verificar se o usuário está autenticado no sistema, bem como o código da linha 36 a 38, responsável por verificar se o objeto DadosPessoais foi criado e colocado na sessão do usuário. Se o objeto não tiver sido criado e colocado na sessão, o que isso quer dizer? Bom, muito provavelmente o usuário se autenticou no sistema, mas digitou diretamente a URL da segunda tela, pulando o preenchimento da primeira tela.

Por fim, note que o bloco de código da linha 39 a 51 pegam os dados do objeto DadosPessoais, enquanto o bloco de código da linha 52 a 63 pegam dados do objeto request, ou seja, digitados e enviados pela tela anterior do sistema.

```java
package aula04;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ServletTela2CadastroSessao extends HttpServlet {

	protected void doGet(HttpServletRequest request,
			HttpServletResponse response)
				throws ServletException, IOException {
		doPost(request, response);
	}

	protected void doPost(HttpServletRequest request,
			HttpServletResponse response)
				throws ServletException, IOException {
		PrintWriter resposta = response.getWriter();
		resposta.write("<html>");
		resposta.write("<head>");
		resposta.write("<title>Confirmação de registro</title>");
		resposta.write("</head>");
		resposta.write("<body>");
		HttpSession sessao = request.getSession(false);
		if (sessao == null
		  || sessao.getAttribute(ServletLogin.USUARIO) == null) {
			resposta.write(
			  "<a href=\"login.html\">Faça primeiro o seu login</a>");
		} else {
			resposta.write(
				"<h3>Registro realizado com sucesso!</h3><BR><BR>");
			resposta.write("<b>Seus dados pessoais:</b><BR>");
			DadosPessoais dados = (DadosPessoais) sessão
				.getAttribute("dadosPessoais");
			if (dados == null) {
			resposta.write(
"<a href=\"ServletLogin\">Dados incompletos. Inicie o sistema novamente.</a>");
			} else {
				resposta.write(dados.getNome() + " "
				+ dados.getSobrenome());
				resposta.write("<BR>");
				resposta.write(dados.getRua());
				resposta.write("<BR>");
				resposta.write(dados.getComplemento());
				resposta.write("<BR>");
				resposta.write(dados.getCidade());
```

```
52                      resposta.write(", ");
53                      resposta.write(dados.getCep());
54                      resposta.write(", ");
55                      resposta.write(dados.getEstado());
56                      resposta.write("<BR>");
57                      resposta.write(
58                        "<b>Seus dados profissionais:</b><BR>");
59                      resposta.write(request.getParameter("empresa"));
60                      resposta.write("<BR>");
61                      resposta.write(request.getParameter("ruaEmpresa"));
62                      resposta.write("<BR>");
63                      resposta.write(request.getParameter(
64                      "complementoEmpresa"));
65                      resposta.write("<BR>");
66                      resposta.write(request.getParameter("cidadeEmpresa"));
67                      resposta.write(", ");
68                      resposta.write(request.getParameter("cepEmpresa"));
69                      resposta.write(", ");
70                      resposta.write(request.getParameter("estadoEmpresa"));
71                  }
72              }
73              resposta.write("</body></html>");
74      }
75
76  }
```

Listagem 7 - Servlet responsável por montar a segunda tela de cadastro

Atividade 03

1. Implemente o novo sistema de cadastro, com tela de login, conforme mostrado, usando sessões ao invés de repasse de parâmetros. Lembre-se de tentar não olhar para o código mostrado, assim você vai poder comprovar pra você mesmo que você entendeu bem os conceitos. Execute o sistema implementado e verifique que tudo está funcionando corretamente. Corrija quaisquer erros de implementação que você pode ter feito.

2. O acesso ao arquivo cadastro.html não tem verificação de autenticação de usuário, ou seja, qualquer usuário, autenticado ou não, pode ver seu conteúdo. Realize o acesso a esse arquivo sem que você esteja autenticado. Se necessário, reinicie o servidor web para que você não esteja mais autenticado.

3. Crie um ServletCadastro cujo papel é gerar o conteúdo da página cadastro.html. Faça nesse Servlet o controle de autenticação de usuário,

conforme foi feito com o ServletTela1CadastroSessao. Execute o Servlet e confirme que apenas usuários autenticados têm acesso à página inicial de cadastro gerada pelo Servlet. Por fim, altere o ServletMenu para que a página de menu gerada referencie o ServletCadastro, e não mais o arquivo cadastro.html. Renomeie o arquivo cadastro.html para cadastro_antigo.html e verifique que o sistema, quando acessado pelo ServletLogin, continua funcionando.

Finalizando sessões

Uma vez que uma sessão é aberta, existem algumas formas de fazer com que a mesma seja finalizada. A primeira forma de se fazer isso é reiniciando o servidor. Caso isso ocorra, todos os objetos sessão serão perdidos. A reinicialização do sistema não é frequente, na prática, visto que reiniciá-los quer dizer deixar os usuários sem poder usar o sistema durante um certo período de tempo (tempo de reinicialização).

Uma segunda forma de se finalizar uma sessão é através do método invalidate() da classe HttpSession. Esse método é utilizado para liberar o objeto sessão do servidor (encerrar a sessão) e descartar todo o seu estado (atributos). Essa forma de se liberar a sessão é geralmente utilizada quando se tem sistemas de autenticação, no qual o usuário realiza uma ação solicitando essa ação (exemplo: logout do usuário). Em um sistema bancário, por exemplo, ao terminar suas operações, um usuário deve clicar na opção de sair do sistema, encerrando assim sua sessão. Uma outra opção é remover o atributo USUARIO da sessão usando o método removeAttribute() de HttpSession, mas o invalidate() é mais indicado (por questões de segurança) se você não quer mais guardar informações sobre o usuário, já que ele remove o objeto sessão todo, e não apenas um de seus atributos.

Por fim, a terceira forma de encerrar uma sessão é o chamado timeout de sessão. O termo timeout é utilizado para indicar que um determinado tempo de sessão inativa foi alcançado. Por exemplo, um timeout de 20 minutos quer dizer que, se o usuário relativo a essa sessão não acessar o sistema por mais de 20 minutos, a sessão será descartada automaticamente pelo servidor. Esse recurso é importante, já que às vezes acessamos um sistema web através de computadores compartilhados. Usuários que acessam contas bancárias, por exemplo, devem ter sua sessão automaticamente expirada (liberada) em torno de 7 minutos de

inatividade. Dessa forma, se o usuário for embora e esquecer o sistema aberto em um computador compartilhado, uma pessoa mal-intencionada que chegue 10 minutos depois não vai poder acessar a conta do usuário do banco, pois a sessão já vai ter expirado. O valor do timeout de uma sessão pode ser configurado pelo método setMaxInactiveInterval(), o qual recebe um valor inteiro representando o tempo de timeout em segundos.

Atividade 04

1. Implemente um Servlet chamado ServletSair, o qual deve invalidar a sessão atual do usuário e redirecionar a requisição para o Servlet de autenticação. Coloque um link de nome sair dentre as opções geradas pelo ServletMenu, o qual deve apontar para o ServletSair. Execute o sistema e verifique se o ServletSair está funcionando de acordo.

Conclusão

Muito bem, é isso, chegamos ao fim da nossa aula! Faça sua autoavaliação e, qualquer dúvida, releia o material. Na próxima aula, iremos dar continuidade na programação web, tratando de uma nova tecnologia, a qual irá lhe ajudar a desenvolver código de forma mais legível e elegante. Até lá!

Leitura Complementar

Para complementar seu entendimento sobre o uso de sessões, analise os métodos disponibilizados pelas classes envolvidas. Isso pode ser feito observando-se a API do Java:

Java(TM) EE 8 Specification APIs

- <https://javaee.github.io/javaee-spec/javadocs/>

Para os mais curiosos, informações extras sobre cookies podem ser vistas na classe Cookie nesse mesmo pacote.

Resumo

Nesta aula, você aprendeu a trabalhar com sessões de usuário. Você viu que uma sessão pode ser aberta para um determinado usuário com o objetivo de manter no servidor o estado (informações) do cliente. Um dos usos mais comuns de sessão é para autenticação de usuários. Você viu que o método getSession() pode receber um boleano como parâmetro, modificando um pouco o seu comportamento. No caso, se o boleano for false, o método não deve criar uma sessão para o usuário, mas retornar o valor null. Você também viu que existem várias formas de uma sessão ser finalizada (timeout, reinicialização do servidor ou uso do método invalidate()).

Autoavaliação

1. Descreva o funcionamento das sessões dos usuários com relação aos sistemas web.

2. Liste as principais classes e métodos utilizados na programação Java para web que utilizados para manipular sessões de usuários. Indique quando um Servlet deve ou não criar uma sessão e como eles podem fazer isso.

Referências

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com J2EE J2EE e UML.** Rio de Janeiro: Ciência Moderna, 2003. 324 p.

CATTELL, Rick; INSCORE, Jim. **J2EE:** criando aplicações comerciais. Rio de Janeiro: Editora Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), 2001. 752 p. Disponível em: <http://pdf.moreservlets.com/>. Acesso em: 11 maio 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP).** 2nd ed. New Jersey: Prentice Hall PTR (InformIT), 2003. 736p. (Core Technologies, 1). Disponível em: <http://pdf.coreservlets.com/>. Acesso em: 11 maio 2012.

HYPERTEXT Transfer Protocol: HTTP/1.1 – Methods Definition. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Acesso em: 11 maio 2012.

J2EE Tutorial. Disponível em: <http://java.sun.com/javaee/reference/tutorials/>. Acesso em: 11 maio 2012.