Unidade 11: Sessão e Login

Prof. Daniel Caetano

Objetivo: Permitir compartilhamento de dados temporário entre diferentes Servlets que compõem uma aplicação.

INTRODUÇÃO

Até agora, sempre que precisamos transferir um dado entre um servlet para outro nos bastava inseri-lo em uma requisição e transferi-la. Entretanto, isso funciona apenas em aplicações mais simples. Quando se deseja construir uma aplicação profissional, é frequente precisar armazenar algumas informações de maneira que vários servlets possam compartilhá-la.

Na aula de hoje aprenderemos como fazer isso por meio das variáveis de sessão, que será usada, nessa aula, para implementar o controle de login.

1. ESTADOS E O COMPORTAMENTO DOS COMPONENTES

Antes de mais nada, precisamos compreender o que significa "login".

Vamos começar entendendo os dois tipos de componente de um sistema, quando classificados de acordo com seu comportamento: **existem os componentes que executam sempre a mesma tarefa**... e **existem os componentes que, dependendo da situação, executam tarefas diferentes**.

No primeiro caso, isto é, em componentes que fazem sempre a mesma tarefa, para um dado conjunto de parâmetros, o resultado sempre o mesmo. Por exemplo: se criarmos um componente que realiza a soma de dois números, sempre que passarmos os números 2 e 3 como parâmetros, o componente nos responderá o resultado 5.

No segundo caso, o componente não age sempre da mesma forma. Digamos que criemos um componente que, dependendo da situação, ele calcula a soma ou a multiplicação. Neste caso, se passarmos os valores 2 e 3 para este componente poderemos obter como resposta os valores 5 (resultado da soma) ou 6 (resultado da multiplicação).

Neste segundo caso, dizemos que a resposta do componente depende do **estado** deste componente. Para entender o conceito, vamos a um exemplo mais concreto. Quando somos jovens, eventualmente precisamos pedir dinheiro para nossos pais para, por exemplo, pagar o ônibus até a escola. Neste caso, nosso pai pode responder fornecendo o dinheiro ou pode

responder que não tem dinheiro. A resposta, depende, portanto, do **estado** financeiro de nosso pai, isto é, se ele tem ou não o dinheiro.

Por essa razão, os componentes que sempre se comportam da mesma maneira, considerando apenas os parâmetros, chamamos de "**componentes sem estado**" ou, usando o jargão da área, são componentes *stateless*. Já os componentes que podem se comportar de maneira diferente, isto é, fornecer respostas diferentes para uma mesma requisição, são chamados de "**componentes com estado**" ou, usando o jargão da área, são componentes *statefull*.

Quando desenvolvemos um sistema que envolve componentes com estado é importante identificar 3 elementos:

a) Quais são esses estados?
b) Como ocorre a mudança de um estado para outro?
c) Qual o comportamento do componente em cada um destes estados?

As perguntas (a) e (b) são respondidas por um diagrama chamado "Diagrama de Estados", como o representado abaixo:

A bolinha preta indica o estado inicial do sistema (no caso, "Usuário não Autenticado") e as setas indicam as ações que levam à transição entre tais estados - A autenticação no Banco de Dados muda o estado para "Usuário Autenticado", por exemplo.

A pergunta (c) é, normalmente, respondida pela descrição dos Casos de Uso: uma das seções desta descrição é chamada de "Pré-Condições". Nessa seção deve ser descrito, por exemplo: "Para esse caso de uso ocrrer, o sistema deve estar no estado "Usuário Autenticado".

2. O LOGIN E OS ESTADOS RELACIONADOS

Sempre que um sistema possui login, praticamente todos os seus componentes passam a ter **dois** estados distintos:

1. **Logado**: atua normalmente
2. **Não Logado**: redireciona a execução para a tela de login

A transição entre estes estados se dá nos seguintes casos:

Não Logado para Logado:

1. Quando o usuário realiza o Login

Logado para Não Logado:

1. Quando o usuário seleciona a opção "sair"
2. Quando passou tempo demais sem que o usuário fizesse nada.

Para que os componentes de um sistema (Servlets ou JSPs) possam reagir adequadamente em cada caso, eles precisam ter acesso a uma variável que lhes indique se um determinado usuário **já fez o login** ou se **não fez o login**.

Para armazenar essa informação usaremos um recurso adicional do Java EE chamado Sessão.

3. A SESSÃO

De maneira direta, a Sessão é um objeto do Java EE que permite o compartilhamento de dados/objetos entre os vários Servlets/JSPs que compõem uma Aplicação Web.

A sessão pode ser encarada como uma grande caixa de "achados e perdidos" aos quais todos os Servlets/JSPs têm acesso, pois existe uma referência para ela dentro da **request**. Para "descobrir" quem é a sessão do Servlet/JSP atual, usa-se o seguinte comando:

```
request.getSession()
```

Frequentemente ele aparece no código associado a uma variável de objetos do tipo HttpSession, com o seguinte formato:

```
HttpSession session = request.getSession();
```

> **NOTA**: O passo de pegar uma referência para a sessão é desnecessário nos JSPs, já que o objeto **session** já é previamente criado. Basta usá-lo.

O armazenamento de objetos na sessão pode ser feito **exatamente** da mesma forma que na requisição:

```
session.setAttribute("etiqueta", objeto);
```

Recuperar um objeto da sessão também ocorre da mesma maneira que com a requisição:

```
Object objeto = (Object)session.getAttribute("etiqueta");
```

A diferença é que tudo que for colocado na sessão permanecerá lá até que seja retirado ou até que a sessão seja encerrada.

Para tirar um objeto da sessão, basta removê-lo com o comando apropriado:

```
/* Coloca o Objeto */
session.setAttribute("etiqueta", objeto);

/* Remove o Objeto */
session.removeAttribute("etiqueta");
```

3.1. Ciclo de Vida da Sessão

A sessão é criada pelo Java automaticamente, assim que é solicitada à **request** pela primeira vez . Depois disso, há duas formas de ela ser destruída:

a) Por tempo

De acordo com o tempo especificado no arquivo **web.xml**, chamdo ***timeout***

b) Por solicitação do usuário

Pelo método **invalidate**: **session.invalidate();**

Um Servlet/JSP pode alterar o ***timeout*** de uma sessão através do seguinte comando:

```
session.setMaxInactiveInterval( tempo );
```

Por exemplo, para mudar o tempo máximo de inatividade para 5 minutos (300 segundos), basta dar o comando:

```
session.setMaxInactiveInterval( 300 );
```

4. O COMPONENTE DE LOGON

Uma vez que nenhum sistema comercial profissional estará completo sem um sistema de logon apropriado, é interessante criá-lo da maneira mais reutilizável possível... e é exatamente isso que veremos nessa aula.

Vamos começar criando uma nova Aplicação Web, **WProjeto8**. Vamos fazer algo realmente simples para testar:

PASSO A. Crie um Pacote Java chamado **teste**
PASSO B. Crie um Servlet chamado **Teste**
PASSO C. Faça este servlet inserir uma String na requisição:

Etiqueta	String
msg	"Olá, você obteve acesso!"

Teste.java (método processRequest)

```
protected void processRequest(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

    RequestDispatcher rd;

    try {
        request.setAttribute("msg","Olá, você obteve acesso!");
        rd = request.getRequestDispatcher("/Teste.jsp");
        rd.forward(request, resposnse);
        return;
    } finally {
    }
}
```

PASSO D. Crie, agora, um JSP chamado **TesteView**
PASSO E. Modifique o JSP para que imprima a mensagem **msg** da requisição.
PASSO F. Teste o seu servlet, executando:

http://localhost:8080/WProjeto8/Teste

Observe que você obteve acesso.

Vamos, agora, proteger este "sistema" com um logon.

4.1. Criando a Base do Componente de Logon

Como pretendemos criar um módulo de login reaproveitável, vamos criá-lo como um pacote separado. Com o projeto **WProjeto8** aberto...

PASSO 1: Clique com o botão direito na pasta "Pacotes de Código Fonte" e selecione **Novo > Pacote Java**. Indique o nome **Logon** para este pacote e clique, depois em Finalizar.

PASSO 2: Clique com o botão direito no pacote "Logon" e seleciona **Novo > Classe Java**. Indique o nome **Logon** para essa classe.

PASSO 3: Agora precisamos criar o servlet **WLogon**, que irá receber as informações de um formulário e verificá-la. Clique com o botão direito no pacote Logon e selecione **Novo > Servlet**. Dê o nome de **WProcessaLogon** ao servlet e não esqueça de marcar a caixa que adiciona as informações do servlet no descritor. Esse é um servlet de processamento, que deve ter o seguinte código de processRequest:

WLogon (método processRequest)

```
    protected void processRequest(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

        try {
            // Recupera parâmetros
            String nome = request.getParameter("user");
            String pass = request.getParameter("pass");
            String sucesso = request.getParameter("sucesso");
            String fracasso = request.getParameter("fracasso");
            // Pega referência para a Sessão
            HttpSession sessao = request.getSession();
            // Define referência para o request dispatcher
            RequestDispatcher rd;


            // Verifica logon...
            boolean logon = false;
            if (nome.compareTo("admin")==0 && pass.compareTo("12345")==0) {
                logon = true;
            }

            // Se logon passou...
            if (logon == true) {
                // Coloca nome do usuário na sessão, com etiqueta
                // Igual a "user"
                sessao.setAttribute("user",nome);
                // Redireciona para destino em caso de sucesso.
                rd = request.getRequestDispatcher(sucesso);
                rd.forward(request,response);
                return;
            }
            // SE logon NÃO passou...
            else {
                // Remove usuário da sessão
                sessao.removeAttribute("user");
                // Redireciona para destino em caso de fracasso
                rd = request.getRequestDispatcher(fracasso);
                rd.forward(request,response);
                return;
            }
        }
    }
```

PASSO 4: Em caso de fracasso, podemos adicionar um código de erro também:

WLogon (método processRequest)

```
    protected void processRequest(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

        try {
            // Recupera parâmetros
            String nome = request.getParameter("user");
            String pass = request.getParameter("pass");
```

```
            String sucesso = request.getParameter("sucesso");
            String fracasso = request.getParameter("fracasso");
            // Pega referência para a Sessão
            HttpSession sessao = request.getSession();
            // Define referência para o request dispatcher
            RequestDispatcher rd;

            // Verifica logon...
            boolean logon = false;
            if (nome.compareTo("admin")==0 && pass.compareTo("12345")==0) {
                logon = true;
            }
            // Se logon passou...
            if (logon == true) {
                // Coloca nome do usuário na sessão, com etiqueta
                // Igual a "user"
                sessao.setAttribute("user",nome);
                // Redireciona para destino em caso de sucesso.
                rd = request.getRequestDispatcher(sucesso);
                rd.forward(request,response);
                return;
            }
            // SE logon NÃO passou...
            else {
                // Remove usuário da sessão
                sessao.removeAttribute("user");
                 // Adiciona código de erro na requisição
                 request.setAttribute("erro","Usuário ou senha incorretos!");
                // Redireciona para destino em caso de fracasso
                rd = request.getRequestDispatcher(fracasso);
                rd.forward(request,response);
                return;
            }
        }
    }
```

4.2. Criando o Formul]ario d Logon na Aplicação Web

PASSO 5: Vamos, agora, criar um formulário de logon. Esse formulário será criado no index.jsp, como a tela de entrada do sistema. Modifique o HTML do **index.jsp** assim:

index.jsp (apenas html)

```
<%    String erro = (String)request.getAttribute("erro");
      if (erro == null) erro = "";
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acesso ao Sistema</title>
    </head>
    <body>
        <h1>Logon no Sistema</h1>
        <p><%= erro %>
        <form action="WLogon" method="post">
            <p>Usuário: <input type="text" name="user"></p>
            <p>Password: <input type="password" name="pass"></p>
            <input type="hidden" name="sucesso" value="/Teste">
            <input type="hidden" name="fracasso" value="/index.jsp">
            <input type="submit" value="Entrar!">
        </form>
    </body>
</html>
```

Este formulário enviará os dados para o servlet WProcessaLogon, que precisaremos criar. **Observe os campos "input hidden".** Estes campos servem para indicar os **servlets destino**, caso o logon ocorra com sucesso ou com fracasso.

Teste! O logon já deve funcionar... mas o acesso direto ainda é permitido. Veja:

http://localhost:8080/WProjeto8/Teste

4.3. Verificando o Logon

PASSO 6: Iremos agora modificar os Servlets/JSPs para que verifiquem se o logon está realizado, através da ***bandeira*** "user" (flag). Por exemplo, vamos começar a modificação pelo servlet Teste.

Teste.java (método processRequest)

```
protected void processRequest(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

    RequestDispatcher rd;

    // Verifica Logon
    HttpSession s = request.getSession();
    String user = (String)s.getAttribute("user");
    if (user == null) {
        rd = request.getRequestDispatcher("/index.jsp");
        rd.forward(request,response);
        return;
    }

    try {

        request.setAttribute("msg","Olá, você obteve acesso!");
        rd = request.getRequestDispatcher("/Teste.jsp");
        rd.forward(request, resposnse);
        return;
    } finally {
    }
}
```

PASSO 7: Experimente! Observe que isso funciona, mas é um tanto tosco. Será que teremos que adicionar esse código a todas as páginas? Muito código para pouco serviço! Para dar um jeito nisso, vamos transferir essa função, muito comum, para o servlet WLogon.

A idéia é criar criar um método estático chamado **verifica** no Servlet **WLogon**. E podemos fazer isso? Wlogon não é um Servlet?

Claro que podemos. Servlet ou não, Wlogon continua sendo uma **classe** Java e, sendo assim, podemos fazer com ela o que bem entendermos. Observe o código que deve ser acrescentado na classe WLogon:

WLogon.java (método verifica)

```
    /**
     * Verifica se logon já foi realizado.
     * Verifica se existe marca de logon na sessão. Se houver,
     * retorna true. Se não houver, encaminha execução para
     * servlet de logon e retorna falso.
     * @param request Objeto de requisição.
     * @param response Objeto de resposta.
     * @param logUrl URL, iniciada por /, indicando o endereço de logon
     * @return true caso o estado seja de logon ativo
     * @throws ServletException
     * @throws IOException
     */
    public static boolean verifica( HttpServletRequest request,
            HttpServletResponse response, String logUrl)
            throws ServletException, IOException {
        // Recupera "user" da sessão
        HttpSession s = request.getSession();
        String user = (String)s.getAttribute("user");
        // Se não existir, vai pra tela de logon
        if (user == null) {
            RequestDispatcher rd;
            rd = request.getRequestDispatcher(logUrl);
            rd.forward(request,response);
            return false;
        }
        return true;
    }
```

Observe a necssidade de mandar um monte de informações para que esse método possa fazer seu serviço: o objeto de requisição, o objeto de resposta e a URL de logon. Infelizmente, tudo isso é necessário.

PASSO 8: O método **verifica** que acabamos de criar fará o "serviço sujo"; ele responde verdadeiro caso usuário esteja logado e, se não, responde falso e encaminha a execução para a tela de login fornecida. Mas para que ele faça seu serviço, precisamo usá-lo! O jeito de usá-lo é esse:

WLogon.verifica(request, response, "/index.jsp");

Todos os parâmetros, como já ressaltado, são necessários para o check realizar o processamento dele; o último é o único que eventualmente mudaremos, pois ele indica o endereço da tela de logon.

PASSO 9: Vamos agora voltar ao código do WProjeto8 e vamos substituir aquele monte de código por esta nova linha de código, lembrando de corrigir as importações... e que o texto "cortado" abaixo deve ser **apagado**!

Teste.java (método processRequest)

```
protected void processRequest(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

    RequestDispatcher rd;

    // Verifica Logon
     if (WLogon.verifica(request,response,"/index.jsp") == false) return;

    // Verifica Logon
    HttpSession s = request.getSession();
    String user = (String)s.getAttribute("user");
    if (user == null) {
        rd = request.getRequestDispatcher("/index.jsp");
        rd.forward(request,response);
        return;
    }

    try {

        request.setAttribute("msg","Olá, você obteve acesso!");
        rd = request.getRequestDispatcher("/Teste.jsp");
        rd.forward(request, resposnse);
        return;
    } finally {
    }
}
```

PASSO 10: Experimente! Sem realizar o Logon, tente entrar direto no servlet WSisCli, usando: http://localhost:8080/WProjeto8/Teste

Agora, faça o logon e veja como funciona!

Sempre que quisermos usar o sistema de Logon, temos que acrescentar essa linha no topo da página:

if (WLogon.verifica(request, response, "/index.jsp") == false) return;

Incluindo o JSP... obviamente dentro de uma seção **scriplet**:

<% if (WLogon.verifica(request, response, "/index.jsp") == false) return; %>

Essas verificações devem ser **sempre** as primeiras coisas em cada arquivo.

4.4. Saindo do Sistema

PASSO 11: Um último toque, relevante, é acrescentar a opção para fazer o logout... ou seja, precisamos de um jeito de realizar o logout de maneira simples. Na pasta Logon, crie um servlet chamado "Logoff", com o seguinte código:

WLogoff.java (método processRequest)

```
protected void processRequest(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

    String destino = (String)request.getAttribute("logoff");

    HttpSession session = request.getSession();
    session.invalidate();

    RequestDispatcher rd = request.getRequestDispatcher(destino);
    rd.forward(request, response);
    return;

    } finally {
    }
}
```

Para fazer o logoff, basta, no HTML, usar um link:

Sair

Experimente acrescentar isso ao **TesteView.jsp**!

Com isso o nosso sistema de logon básico está implementado.

5. INCREMENTANDO O LOGON DO SISTEMA (OPCIONAL)

Como poderíamos modificar o componente Logon de maneira a verificar, em um banco de dados, se o usuário existe? Bem, basicamente precisaríamos criar uma tabela no banco de dados, como indicado a seguir, usando o administrador de banco de dados do NetBeans, como visto em aulas anteriores.

PASSO 1: Crie um bando de dados chamado **logon**, usuário **logon** e senha **logon**.

PASSO 2: Crie uma tabela chamada **usuario**, que irá armazenar os dados de nossos usuários. Como os usuários possuem apenas **nome** e **pass** teremos duas colunas nesta tabela, conforme indicado a seguir.

nome : PK, VARCHAR(20)	password : VARCHAR(50)
...	...

PASSO 3: Vamos, agora, insira manualmente os dados nessa tabela, por exemplo:

nome: **admin**
password: **12345**.

```
INSERT INTO usuarios VALUES ('admin', '12345');
```

PASSO 4: Execute o select a seguir e observe que o password foi gravado de forma codificada!

```
SELECT * FROM usuarios;
```

PASSO 5: Agora, ao código. Crie, dentro do pacote Logon, uma classe chamada **LogonDAO** como indicada a seguir:

LogonDAO.java

```
package Logon;
import java.sql.*;
/**
 * Responsável por Verificar Logon
 * Essa classe torna o uso do BD "transparente" para o logon.
 * @author djcaetano
 */
public class LogonDAO {
    /**
     * Verifica Logon.
     * @param nome Nome do usuário.
     * @param pass Password do usuário.
     * @return True se usuário existe com esse password.
     */
    public static boolean verifica(String nome, String password) {
        try {
            // *** CONECTA AO BANCO
            // Seleciona driver
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            // Indica endereco do banco
            Connection con = DriverManager.getConnection(
                "jdbc:derby://localhost:1527/logon",
                "logon", "logon");
            if (con == null) throw new SQLException();
            // Cria a transação
            Statement transacao = con.createStatement();

            //*** Executa Busca (Query)!
            // Cria a query de busca...
            String query  = "SELECT nome FROM usuarios WHERE ";
            query += "nome LIKE '" + nome + "'";
            query += " && ";
            query += "password LIKE '" + password + "'";
            ResultSet res = transacao.executeQuery(query);

            if (res.next()) {
                transacao.close();
                con.close();
                return true;
            }
```

```
            //*** Finaliza transação e conexão
            transacao.close();
            con.close();
            }

        // se houve algum erro nas transações de SQL...
        catch (SQLException ex) {
            System.err.println(ex); // Código apenas para debug
            }
        // Se não encontrou o driver
        catch (ClassNotFoundException ex) {
            System.err.println(ex); // Código apenas para debug
            }

        // Por padrão, retorna não encontrado.
        return false;
    }
```

PASSO 6: Agora modifique a classe **WLogon**, deste mesmo pacote **Logon**, como indicado a seguir, lembrando que as linhas riscadas devem ser removidas:

WLogon (método processRequest)

```
    protected void processRequest(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
        try {
            // Recupera parâmetros
            String nome = request.getParameter("user");
            String pass = request.getParameter("pass");
            String sucesso = request.getParameter("sucesso");
            String fracasso = request.getParameter("fracasso");
            // Pega referência para a Sessão
            HttpSession sessao = request.getSession();
             // Define referência para o request dispatcher
            RequestDispatcher rd;

            // Verifica logon...
            boolean logon = LogonDAO.verifica(nome,pass);
             ~~boolean logon = false;~~
            ~~if (nome.compareTo("admin")==0 && pass.compareTo("12345")==0) {~~
             ~~logon = true;~~
             ~~}~~

            // Se logon passou...
            if (logon == true) {
                // Coloca nome do usuário na sessão, com etiqueta
                // Igual a "user"
                sessao.setAttribute("user",nome);
                // Redireciona para destino em caso de sucesso.
                rd = request.getRequestDispatcher(sucesso);
                rd.forward(request,response);
                 return;
            }
            // SE logon NÃO passou...
            else {
                // Remove usuário da sessão
                sessao.removeAttribute("user");
                 // Adiciona código de erro na requisição
                 request.setAttribute("erro","Usuário ou senha incorretos!");
                // Redireciona para destino em caso de fracasso
                rd = request.getRequestDispatcher(fracasso);
                rd.forward(request,response);
                 return;
            }
        }
    }
```

Pronto! Seu sistema já é capaz de verificar logon a partir de informações no banco de dados!

6. COMENTÁRIOS SOBRE BANCO DE DADOS DE LOGON

Você deve ter observado que é muito simples a criação de um banco de dados de logon. Entretanto, alguns cuidados são essenciais:

a) **Evite colocar o password do usuário em um objeto**. Dependendo da forma com que o objeto for usado, poderá significar que o seu password viajará pela rede sem nenhuma codificação, criando uma falha de segurança.

b) **NUNCA armazene os passowrds sem codificação**. Fizemos uma implementação simplificada que armazena o password "em aberto". Não faça isso. Use comandos de codificação da linguagem Java para codificar o password **antes** de armazená-lo e posteriormente codifique para verificar também. Se você armazenar os dados de password de forma não codificada e algum hacker roubar seu banco de dados, automaticamente ele saberá os passwords de todos os usuários!

O item (b) pode ser observado para qualquer informação "sensível". Algumas informações devemos evitar armazenar (como, por exemplo, números de cartão de crédito). Mas, caso os guardemos, devemos sempre guardá-los de maneira **codificada**.

Uma informação importante: **não deve ser possível** recuperar o valor original da informação de password armazenada. Sugere-se usar algo como SHA-1 ou algo do gênero, e uma vez codificada, a informação não poderá mais ser decodificada. Por isso que, na verificação, ao invés de decodificar o password do banco, nós devemos codificar o password fornecido pelo usuário.

7. BIBLIOGRAFIA

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

QIAN, K; ALLEN, R; GAN, M; BROWN, R. **Desenvolvimento Web Java.** Rio de Janeiro: LTC, 2010.