VOLTAR

O uso de JavaBeans e JSP com banco de dados

Capacitar o aluno a integrar JSP com banco de dados para dar maior flexibilidade ao desenvolvimento de aplicações dinâmicas na web.

NESTE TÓPICO

Introdução

Para melhorar ainda mais este dinamismo, vamos acrescentar a utilização de JavaBens nas aplicações web.

Classe DAO

No projeto anterior, foi criada uma classe DAO (Data Access Object) no próprio JSP. Neste projeto, vamos pegar essa classe e transferi-la para um pacote denominado **br.com.meuSite.BD.** Basta copiar todo conteúdo da classe desenvolvida na aula anterior para o novo arquivo DAO.java.

Às vezes ficamos imaginando por que tantas tecnologias diferentes se em apenas uma delas podemos resolver. Por exemplo, o mural já foi desenvolvido em JSP. Por que então agregar outras tecnologias?

A ideia é desenvolver uma aplicação bem organizada, de modo a ter uma performance aceitável para o usuário e um código fonte bem escrito, bem organizado, para facilitar a manutenção ou novas implementações quando necessário.

Além de já separarmos a classe que estava no JSP em uma classe Java, vamos também acrescentar o JavaBeans em nossa aplicação, a fim de que ela comece a ser fragmentada de forma a ter cada funcionalidade separada para facilitar a manutenção e, ao mesmo tempo, não alterar a dinâmica da execução.

JavaBeans

Em caso de dúvida em relação ao que é um JavaBeans e o seu funcionamento estude novamente a aula 10.

Primeiro vamos construir o nosso, que nada mais é do que uma classe Java, à qual daremos o nome de Mural. Vejamos o código:

```
1.  package br.com.meuSite.JB;
2.  public class Mural {
3.      private int codigo;
4.      private String nome;
5.      private String texto;
6.      private String email;
7.      private String data;
8.
9.      public int getCodigo() {
10.         return codigo;
11.     }
12.     public void setCodigo(int codigo) {
13.         this.codigo = codigo;
14.     }
15.     public String getNome() {
16.         return nome;
17.     }
18.     public void setNome(String nome) {
19.         this.nome = nome;
20.     }
21.     public String getTexto() {
22.         return texto;
23.     }
24.     public void setTexto(String texto) {
25.         this.texto = texto;
26.     }
27.     public String getEmail() {
28.         return email;
29.     }
30.     public void setEmail(String email) {
31.         this.email = email;
32.     }
33.     public String getData() {
34.         return data;
35.     }
36.     public void setData(String data) {
37.         this.data = data;
38.     }
39. }
```

Novo método no JavaBeans

Vamos acrescentar um método que retorne todos os itens de uma tabela do banco de dados em uma coleção de objetos (em caso de dúvida, reveja a aula 11). Vejamos o código acrescentado na classe DAO():

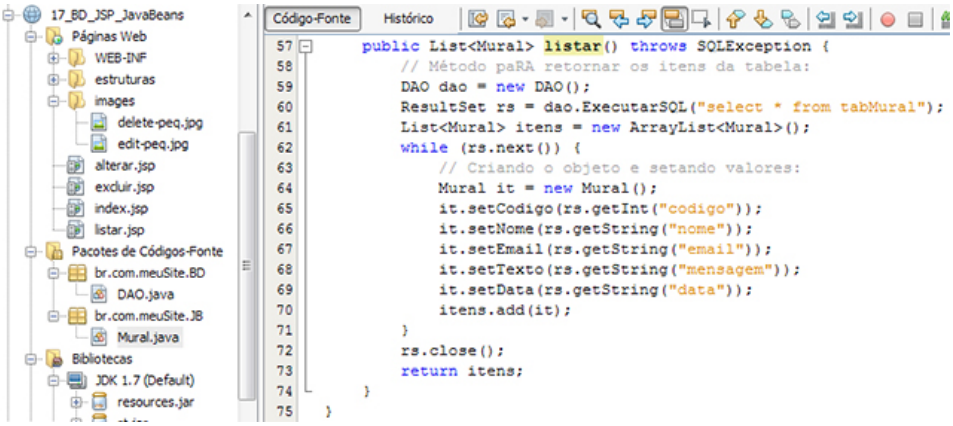
```
    // Este método, instancia o JavaBeans para auxiliar a montar a lista:
    public List<mural> listar() throws SQLException {
        // Utilizando o método da própria classe para retornar os itens da tabela:
        ResultSet rs = ExecutarSQL("select * from Mural");
        List<mural> itens = new ArrayList<mural>();
        while (rs.next()) {
            // Criando o objeto e setando valores:
            Mural it = new Mural();
            it.setCodigo(rs.getInt("codigo"));
            it.setNome(rs.getString("nome"));
            it.setEmail(rs.getString("email"));
            it.setTexto(rs.getString("mensagem"));
            it.setData(rs.getString("data"));
            itens.add(it);
        }
        rs.close();
        return itens;
    }


</mural></mural></mural>
```

Com a inclusão desse método, será necessário também a inclusão de import para a classe Mural, para ArrayList e List:

```
import br.com.meuSite.JB.Mural;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
```

Objetivando melhorar ainda mais nossa aplicação, vamos criar um método no JavaBean para retornar uma coleção de objetos, com todos os registros da tabela tabMural do banco de dados Mural. Essa coleção de objetos será lida no JSP e mesclada com HTML, CSS e JavaScript para apresentar um resultado satisfatório ao usuário.

Vamos observar o código fonte do método implementado no JavaBean:



Vejamos os passos de execução do novo método:

- Observe que o método é do tipo List com base na classe Mural.
- Abre-se conexão com o banco de dados.
- Faz-se um select na tabela, criando um ResultSet (**rs**) na memória com o retorno dos registros da tabela.
- Cria-se um objeto de coleção List para armazenar os objetos desejados.
- Entra-se em um laço que se executa enquanto **rs.next** retornar true.
- Criando objeto **it** da classe Mural e atribuindo cada campo da tabela a ele.
- Adiciona-se o objeto **it** na coleção **itens**.
- Fecha-se o laço, a tabela e retorna-se à coleção de **itens** a ser trabalhada no JSP.

Os códigos JSP

Agora, vejamos o código fonte de cada JSP para entendermos todo o mecanismo. Primeiro a página **index.jsp**, que apenas define um CSS interno e faz um include do listar.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mural - Com JavaBeans</title>
        <style type="text/css">
            body { font-family:Verdana;  }
            table {background-color:#CFF5FE;
                   color:Blue;border-collapse: collapse;border-width: 0; }
            td {border-style: none; border-width: medium;}
            #cor0 {background-color:#CFF5aa; }
            #cor1 {background-color:#CFF5dd; }
            span {font-size: 10px; color:brown}
        </style>
    </head>
    <body>
        <%@include file="listar.jsp"%>
    </body>
</html>
```

O código do **listar.jsp** é o mais complexo e merece toda atenção. Com o conhecimento adquirido até o momento, nada deve parecer coisa do outro mundo.

Vejamos os passos de execução:

```
<%@page import="java.util.List"%>
<%@page import="br.com.meuSite.BD.DAO, br.com.meuSite.JB.Mural" %>
<table width="80%" align="center">
   <tr><th colspan='4'> M U R A L <hr></th></tr>
    <jsp:useBean id="jb" class="br.com.meuSite.JB.Mural">
    <%
    String sDestaque = "onMouseOver=\"this.style.backgroundColor='#ECECFF';
                        this.style.cursor='hand';\"";
    sDestaque += "onMouseOut=\"this.style.backgroundColor='';\"";

    List<Mural> lista = jb.listar();

    int cor = 0;
    for (Mural obj : lista) {
        String sCor = "cor" + (cor % 2);
        cor++;

        out.print("<tr id='" + sCor + "' " + sDestaque + ">");
        if (cor == 1) {
            out.print("<td colspan='2'>Texto de: " + obj.getNome() + "<br>"
                    + "" + obj.getTexto() + "<br>"
                    + "<span>Contato: " + obj.getEmail() + "<br>"
                    + "Postado em: " + obj.getData() + "</span><hr></td>");
        } else {

            out.print("<td>" + obj.getNome() + "</td>");
            out.print("<td>" + obj.getTexto() + "<br>"
                    + "<span>Contato: " + obj.getEmail() + "<br>"
                    + "Postado em: " + obj.getData() + "</span><hr>v/td>");
        }
        // EXCLUSÃO:
        out.print("<td id='cmd'><a href='excluir.jsp?id="
                + obj.getCodigo()
                + "'><img src='images/delete-peq.jpg' /></a></td>");
        // ALTERAÇÃO:
        out.print("<td id='cmd'><a href='alterar.jsp?id="
                + obj.getCodigo()
                + "'><img src='images/edit-peq.jpg' /></a></td>");
        out.print("v/tr>");

      if (cor == 1) {
            out.println(
                 "<tr><th colspan='2'>COMENTÁRIOS<hr></th><th colspan='2'></tr>");
            out.println("<tr style='text-decoration: underline'>");
            out.println("   <td>Usuário</td><td>Texto</td><td colspan='2'></td>");
            out.println("</tr>");
        }
    }
%>
</table>
```

- No topo, foi utilizada a diretiva @page para especificar as classes List, DAO e Mural que serão utilizadas no JSP.
- Inicia-se uma tabela (<TABLE>), uma primeira linha com o título MURAL.
- A tag <jsp:useBean> é utilizada para instanciar o objetojb, que poderia ser feito dentro do scriptlet da seguinte forma: Mural jb = new Mural().
- A variável sDestaque será utilizada para ajudar a criar o HTML a ser exibido no lado cliente.
- O objeto lista irá receber a coleção de objetos vindas do método listar().
- A variável cor controlará a alternância das cores entre os registros exibidos.

- O laço **for()** será realizado para a exibição de todos os itens da coleção.
- É realizado um **if()** com a variável **cor** para saber se é o primeiro registro a ser exibido, o que significa que é o texto principal do mural e deve ter um destaque diferenciado.
- São colocadas imagens para servir de link para exclusão e alteração.
- Nova condição com a variável **cor**, porque após a exibição do texto principal (primeiro registro), será exibido um cabeçalho para os comentários.

O conselho é que se execute esse código, inclusive no modo debug, para entender o passo a passo da execução e que depois também seja analisado o HTML, código fonte exibido pelo navegador.

O debug

Para a execução com o debug no NetBeans, basta colocar o cursor na linha em que deseja uma interrupção na execução e pressionar (Ctrl+F8). A linha ficará marcada e o programa permitirá a execução passo a passo a partir desse ponto.

Para executar no modo debug, vá ao menu debug → Executar Projeto ou pressione (Ctrl+F5). Quando o programa chegar à linha marcada e for interrompido, continue a execução com F7, F8, F5, ou, ainda, estude a opção que melhor lhe atenda no menu debug do NetBeans.

O resultado final

O resultado final de nossa aplicação deve apresentar algo mais ou menos assim :

Referências

DERBY, Manual de Referência do Derby. *Version 10. Derby Document build:* November 14, 2012, 7:45:27 PM (UTC). Disponível em: <http://db.apache.org> (http://db.apache.org/derby/docs/10.2/pt_BR/ref/refderby.pdf). Acesso em: 17 nov. 2012.

ORACLE, *Class DriverManager.* Disponível em:ww.oracle.com (http://www.oracle.com). Acesso em: 17 nov. 2012.

_____. *Class SQLException.* Disponível em: <http://docs.oracle.com> (http://docs.oracle.com/javase/6/docs/api/java/sql/SQLException.html). Acesso em: 17 nov. 2012.

_____. *Interface Connection.* Disponível em: <http://docs.oracle.com> (http://docs.oracle.com/javase/6/docs/api/java/sql/Connection.html). Acesso em: 17 nov. 2012.

_____. *Interface ResultSet:* Disponível em: www.oracle.com (http://www.oracle.com). Acesso em: 17 nov. 2012.

_____. *Interface Statement:* Disponível em: www.oracle.com (http://www.oracle.com). Acesso em: 17 nov. 2012.

_____. *Package java.sql.* Disponível em: www.oracle.com (http://www.oracle.com). Acesso em: 17 nov. 2012.

Avalie este tópico

ANTERIOR
Implementação de métodos com declarações em JSP

Índice

PRÓXIMO
O uso de sessão

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site