Modelagem de Hipermídia
Prof. Giangiacomo Ponzo Neto
Aula 6

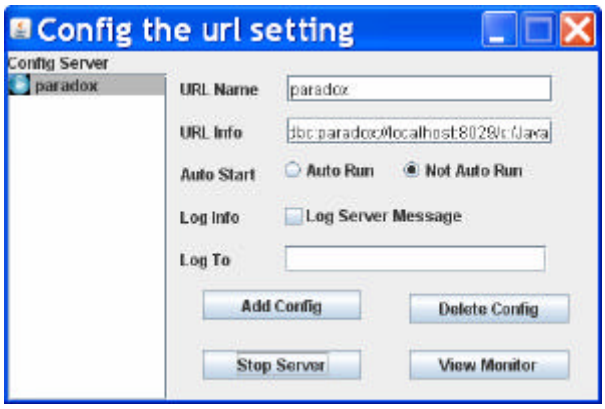
Acesso a Bancos de Dados com JSP

O acesso a um banco de dados pelo Java pode ser sempre feito da mesma forma que vimos no período passado. Ou seja: via JDBC com o mesmo esquema: *import*, declarações, uso da classe do *driver* do banco, abertura da conexão, uso do *statement* passando o SQL, *ResultSet* para ler *selects*, etc. É só colocar isso tudo no "formato" JSP[1]. Lembre-se que devemos ter um gerenciador de banco de dados e um *driver* JDBC que se comunique com esse banco. Pela facilidade de uso, continuarei com o *driver* Paradox (Paradox_JDBC30.jar) que usamos em LP6 e que pode ser baixado da minha página. Coloque-o em c:\Java e descompacte-o.

Abra um novo projeto *web* no NetBeans para fazermos um exemplo. Embora a lógica do programa seja exatamente igual à que vimos em LP6, como estamos em ambiente *web*, algumas coisas precisam ser feitas antes. Para que a comunicação funcione, será necessário ter o *driver* Paradox instalado no local certo do servidor e carregá-lo como biblioteca adicional do projeto no NetBeans, o que faremos agora, da seguinte forma:

- na janela "Projetos", no canto esquerdo superior da tela, selecione o seu novo projeto;
- procure a pasta "Bibliotecas" e clique nela com o botão direito do mouse;
- escolha a opção "Adicionar JAR/Pasta..." ou "Adicionar JAR/Diretório...";
- procure pelo driver JDBC do banco desejado (no nosso caso, o Paradox_JDBC30.jar, que colocamos em c:\Java).

Além disso, a comunicação não é direta como fizemos em *desktop* (pois estamos em ambiente *web* e o banco fica em um endereço da rede). Então devemos abrir uma URL de comunicação com o banco. Faremos isso rodando (duplo clique) no Paradox_JDBC30.jar, que abrirá uma janela de configuração. Em "URL Name", escreva "paradox" (ou qualquer outro nome razoável). Em "URL Info", escreva "jdbc:paradox://localhost:8029/c:/Java" que é a URL que define a porta 8029 para acesso ao banco e localiza onde está o *driver* e as tabelas que formos usar. Clique no botão "Add Config" e depois em "Start Server". Deve ficar parecido com a figura abaixo:



Agora apague o conteúdo do programa JSP que vem no novo projeto e escreva o seguinte:

```
<html>
    <body>
      <%@ page import="java.sql.*" %>
      <h1>Exemplo de JSP com acesso a banco de dados</h1>
      <%
        try {
          Class.forName("com.hxtt.sql.paradox.ParadoxDriver");
```

[1] A esta altura, você já deve ter percebido que a programação para a *web* está sendo bem mais braçal do que usar *jFrames*, por exemplo. Em geral, é assim mesmo, embora existam alguns *frameworks*, como o Visual Web JSF incorporado ao NetBeans a partir da versão 6, ou ferramentas como o Dreamweaver e o Visual Studio que têm a preocupação de integrar facilidades gráficas ao desenvolvimento. Não adianta, no entanto, aprender apenas uma ferramenta específica, pois elas mudam constantemente e os alunos ficariam presos a uma plataforma. Conhecendo a base da programação *web*, usar depois as ferramentas é fácil. O contrário é que seria bem difícil.

```
            Connection
con=DriverManager.getConnection("jdbc:paradox://localhost:8029/c:/Java");
            Statement stmt = con.createStatement();
            stmt.execute("create table Aluno (mat int, nome varchar(40))");
            stmt.execute("insert into Aluno values (1,'JOAO DA SILVA')");
            stmt.execute("insert into Aluno values (2,'MARIA DE OLIVEIRA')");
            stmt.execute("insert into Aluno values (3,'JOSE DAS COUVES')");
            ResultSet rs = stmt.executeQuery("select * from Aluno");
        %>
            <table border=1>
              <tr>
                <th>Matrícula</th>
                <th>Nome</th>
              </tr>
        <%
            while (rs.next()) {
              int mat = rs.getInt(1);
              String nome = rs.getString(2);
        %>
              <tr>
                <td><%= mat %></td>
                <td><%= nome %></td>
              </tr>
        <%
            }
            rs.close();
            stmt.close();
            con.close();
        %>
            </table>
        <%
          }
          catch (Exception e) {
            out.write(e.getMessage()); // é só uma mensagem de erro do Java, então usei o "out"
          }
        %>
      </body>
</html>
```

Rode. Se estiver tudo normal, vai criar uma tabela em c:\Java chamada Aluno.db, inserir 3 registros, selecioná-los e exibi-los na página em formato de tabela HTML. Mais detalhes em sala.

Se quiséssemos usar outro banco em um ambiente separado (não na mesma máquina), bastaria baixar um *driver* JDBC para esse banco, adiciona-lo à biblioteca, alterar a chamada ao *driver* e mudar a String de conexão, de acordo com a documentação do banco, definindo a URL correta (endereço de rede, porta, login, senha, etc.). Por exemplo, se fosse um banco Oracle em um servidor de nome "Ora" na porta 1521, identificador "univ", usuário "adm", senha "adm001", bastaria ter algo como o exemplo abaixo[2]:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@Ora:1521:univ","adm","adm001");
```

Observação: Muitos defendem que não é uma boa prática fazer acesso a bancos de dados diretamente a partir de uma pagina JSP (ou de qualquer linguagem), como visto acima, mas através de alguma camada intermediária de persistência (com algum tipo de mapeamento objeto-relacional, caso usemos bancos relacionais tradicionais). Isto será abordado adiante.

Estudo de Caso

Neste ponto, os alunos já podem criar um pequeno sistema *web*. Nosso "estudo de caso" se dará com um cadastro de livros, que manterá a tabela Livro (código, título, autor). O sistema deve permitir inserir, alterar e excluir registros da tabela e ainda consultar os livros por autor.

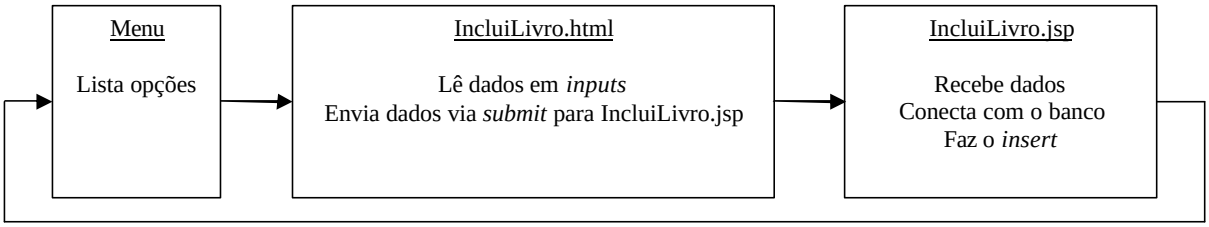
Para tal, devem ser criadas páginas para cada funcionalidade, que serão acessadas por um menu (através de *links* HTML) na página inicial, além de outras páginas JSP e HTML que forem necessárias, como páginas de resposta

[2] Isso é só um exemplo. Dependendo do driver usado, do banco e da configuração da rede, pode ficar um pouco diferente. De qualquer forma, sempre será algo semelhante a isso. Maiores detalhes sobre cada caso específico devem ser obtidos na documentação do banco e/ou do driver JDBC. Um exercício de pesquisa interessante seria passar nosso exemplo para outro banco (o MySQL, por exemplo).

ou gravação de dados no banco. Cada página que se comunique com o banco deverá fazer seu próprio acesso[3]. Não se esqueça de que o HTML é *stateless*, ou seja: não sabe nada sobre o que se passou antes no "sistema" e cada página é, em princípio, um programa isolado carregado integralmente pelo *browser* DEPOIS de ser processado no servidor. Ou seja: não adianta pedir ao usuário para digitar os dados de um livro e em seguida, na mesma página, gravar estes dados, pois a parte de Java não estará mais lá. Para simular esse comportamento "*desktop*", pode ser usada uma técnica – bastante comum na *web* – de colocar a lógica na mesma página e recarregá-la após a entrada de dados. Veja um exemplo de uso dessa técnica na página "Ola.jsp" vista abaixo (que será analisada em detalhes em sala de aula):

```
<html>
  <body>
     <%
        String nome=request.getParameter("txtNome");
        if (nome!=null) {
     %>
          Olá, <%=nome%>
     <%
        }
        else {
     %>
          <form action="Ola.jsp" method="get">
            Digite seu nome<br>
            <input type="text" name="txtNome"><br>
            <input type="submit" value="Entrar">
          </form>
     <%
        }
     %>
  </body>
</html>
```

Você pode usar essa técnica ou colocar as funcionalidades mais isoladas, lendo os dados em HTML puro através de *forms* e trabalhando estes dados em páginas JSP/HTML em separado. Veja o esquema[4] abaixo, que exemplifica a inserção de um registro:



Querendo usar a técnica de uma página só para o pedido e o tratamento dos dados, é só "juntar" a funcionalidade das duas caixas "IncluiLivro" mostradas acima.

Para Saber Mais

Este curso não forma especialistas em desenvolvimento *web*. Ainda assim, vimos bastante coisa. Só que o Java EE é imenso. Para cobrir o tema de forma completa, precisaríamos de muito mais do que meia disciplina. Para comprovar isso, baixe o tutorial em desenvolvimento de aplicações empresariais (*J2EE Tutorial - A beginner's guide to developing enterprise applications*) em http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf. É um guia gratuito, fornecido pela própria Sun, com mais de 1100 páginas. O documento traz várias técnicas e ferramentas, é repleto de conceitos, exemplos e, apesar do seu tamanho, eles o têm apenas como um "*beginner's guide*" (guia para iniciantes). Acho que é suficiente para que tenhamos uma idéia da quantidade de coisas que estão envolvidas no desenvolvimento em Java EE ou qualquer tecnologia com propósitos semelhantes. De qualquer modo, para aqueles que quiserem evoluir no assunto, independente da disciplina ou mesmo do curso como um todo, sugiro o seu estudo (infelizmente não há versão em português, pelo menos até o momento que escrevo). Para quem preferir um livro básico em português, experimente "Dominando NetBeans" de Edson Gonçalves, editora Ciência Moderna, que abrange desenvolvimento *desktop* e *web*, incluindo acesso a bancos de dados, AJAX e uso de *frameworks*. Para algo maior e

[3] Não vamos implementar *pool* de conexões ou coisa semelhante nesta disciplina.

[4] Esse não é um diagrama "oficial", é só um esquema de ligação simples para entender o fluxo.

mais avançado, sugiro, do mesmo autor e editora, “Desenvolvendo Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, EJB 3 Persistence e Ajax”.

Páginas oficiais da Sun sobre os assuntos abordados (em inglês):

- JSP: http://java.sun.com/products/jsp
- Servlets: http://java.sun.com/products/servlets