Desenvolvimento de Aplicações Web

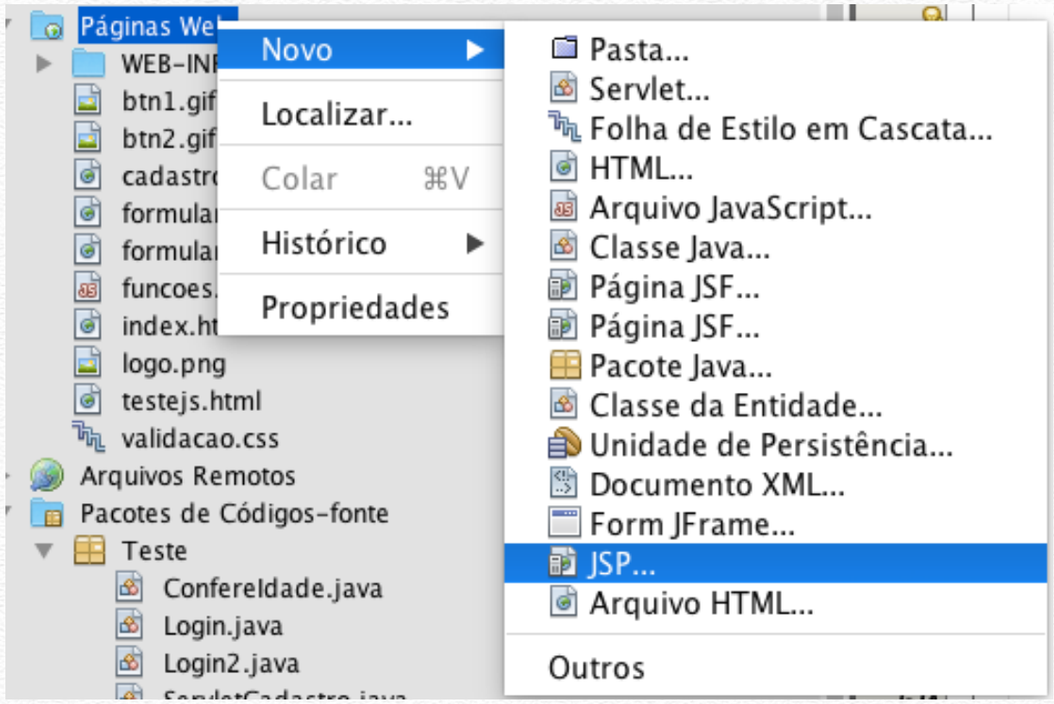
Prof. Pedro Clarindo da Silva Neto

Parte VIII

Introdução ao JSP

A tecnologia JSP permite misturar código HTML estático com conteúdo gerado dinamicamente em Java. Os trechos de código Java são simplesmente inseridos ao longo do HTML dentro de tags especiais, que começam com **<%** e terminam com **%>**. O código a seguir mostra, por exemplo, uma página que utiliza o dado de uma requisição para formar o HTML:

Introdução ao JSP




```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        Olá, <%= request.getParameter("nome") %>. <br/>
        Seja bem-vindo
    </body>
</html>
```

Introdução ao JSP

Podemos pensar em servlets como código Java com HTML por dentro, e em JSP como código HTML com código Java por dentro. Apesar da grande diferença na forma de escrever esses dois tipos de código, serlvets e JSP são, no fundo, a mesma coisa. Os códigos escritos em JSP são traduzidos em servlets. JSP é, portanto, apenas uma forma de escrever servlets na qual o HTML é escrito mais facilmente (afinal, nos servlets devemos imprimir, em Java, cada linha gerada na página retornada ao usuário, o que não é muito produtivo quando temos páginas HTML complexas).

Introdução ao JSP

JSP é mais útil que servlets quando desenvolvemos a parte de apresentação do sistema ao usuário. Serlvets são mais utilizados quando implementamos a lógica de negócios do sistema, que possui mais cálculos e/ou consultas ao banco do que HTML. No exemplo acima, o código HTML será gerado no momento que o usuário fizer uma requisição. O código enviado ao cliente não exibirá o conteúdo entre as tags <% e %>. O que acontecerá é que este conteúdo será substituído pelo valor do campo “nome” enviado na requisição.

Criação de uma padaria virtual

Neste capítulo, criaremos o site de uma padaria virtual chamada "Web Pão". O layout da página inicial do site pode ser visto na Figura:

Criação de uma padaria virtual

A página inicial mostra algumas opções de produtos que podem ser comprados pelo cliente. Os produtos pertencem às categorias “Pães”, “Tortas”, “Bolos”, “Salgados”, “Bebidas” e “Outros Produtos”. A página de cada produto é idêntica à inicial, exceto pelo fato de que só aparecem produtos desta categoria. As páginas do site são criadas dinamicamente e utilizam um banco de dados que contêm:

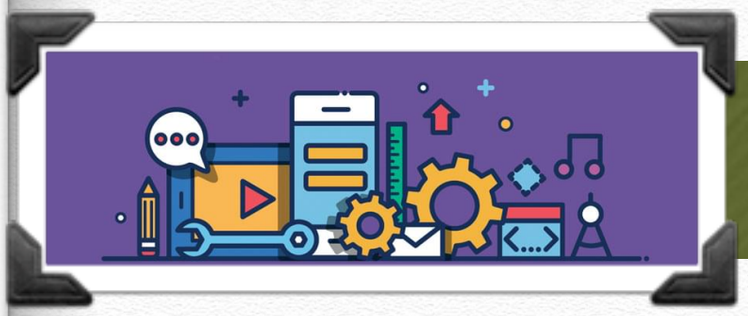
Criação de uma padaria virtual

Tabela contendo os **usuários** cadastrados no sistema. Cada usuário possui um nome, um email, uma senha, um endereço e um telefone.

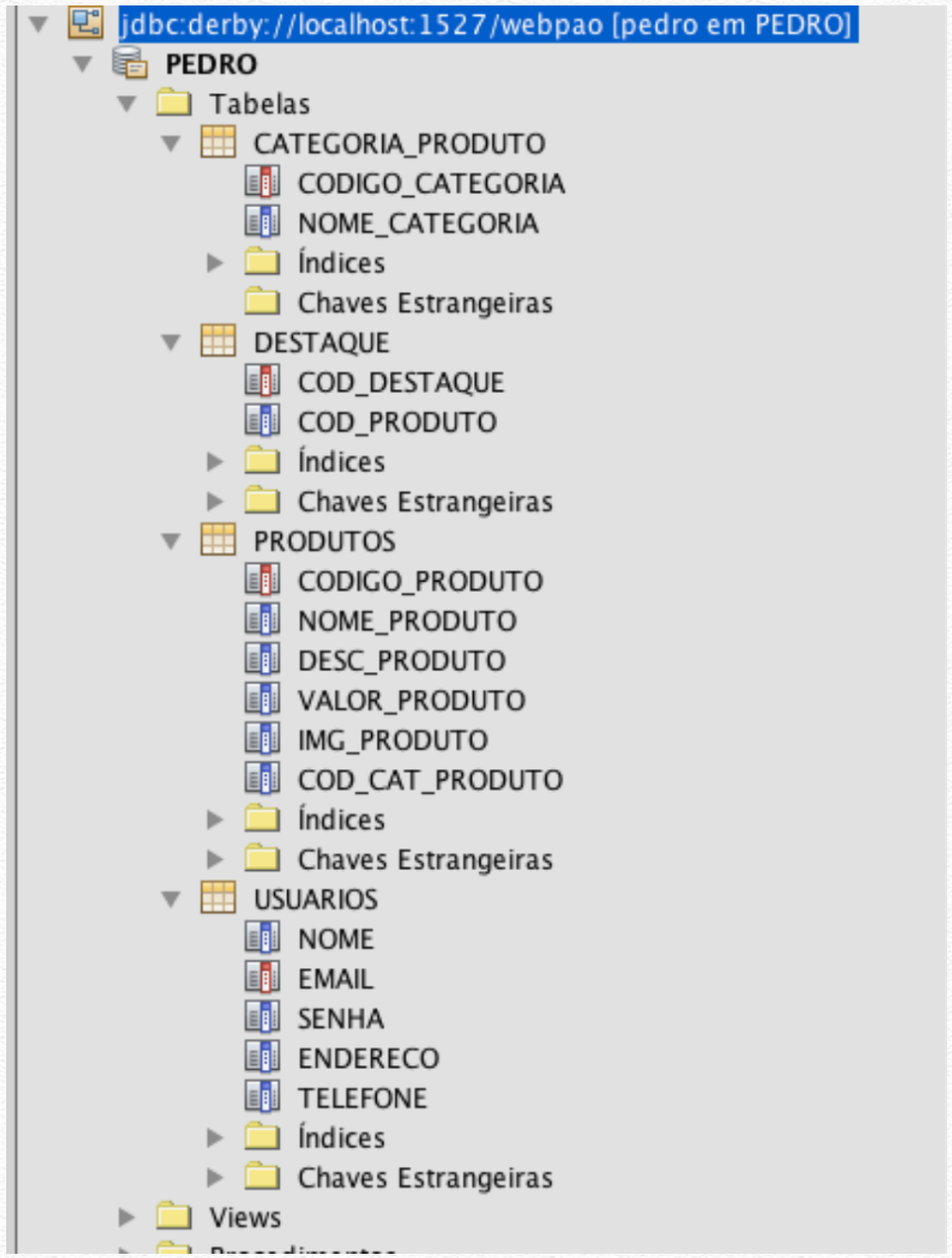
Tabela com as **categorias de produtos**. Cada categoria contém um código (chave primária) e o nome da categoria.

Tabela contendo os **produtos** cadastrados. Cada produto na tabela possui um código (chave primária), um nome, uma descrição, um valor, uma imagem e o código de sua categoria (referente à chave primária da tabela de categorias).

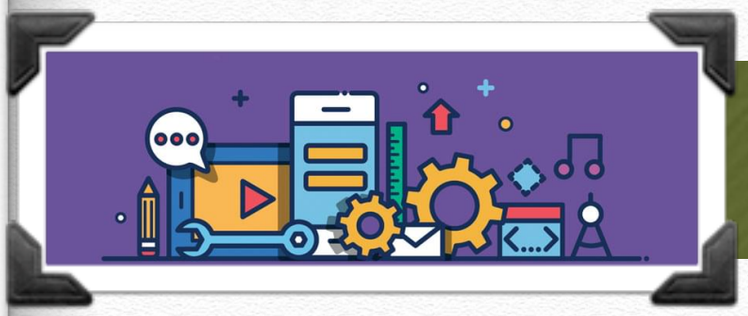
Tabela com **produtos em destaque**, de categorias diversas, que são exibidos na página inicial. Cada destaque possui apenas um código (chave primária) e o código de um produto (referente à chave primária da tabela de produtos).



Criação de uma padaria virtual



Prof. Pedro Clarindo da Silva Neto



Criação de uma padaria virtual

O menu de nosso site depende das categorias cadastradas no banco. Mesmo que categorias sejam incluídas ou removidas, o menu deve continuar funcionando. Para que isso ocorra, ele deve ser dinâmico, baseado na tabela com as categorias existente no banco. O trecho de código abaixo mostra como este menu pode ser feito em JSP:

Criação de uma padaria virtual

```
<a href="index.jsp"> Início</a>

<%
    String driver = "org.apache.derby.jdbc.ClientDriver";
    String url = "jdbc:derby://localhost:1527/webpao";
    String username = "pedro";
    String password = "123456";

    try {
        Class.forName(driver);
        Connection connection = DriverManager.getConnection(url, username, password);
        Statement statement = connection.createStatement();

        String query = "SELECT * FROM categoria_produto";
        ResultSet resultados = statement.executeQuery(query);

        while(resultados.next()){
            out.println("<a href=\"produtos.jsp?cod_cat=");
            out.println(resultados.getString("codigo_categoria")+"\">");
            out.println(resultados.getString("nome_categoria")+"</a>");
        }

        connection.close();

    } catch (ClassNotFoundException cnfe) {

        out.println("Error loading driver" + cnfe);

    } catch (SQLException sqle) {
        out.println("Error with connection" + sqle);

    }
    %>
    <a href="carrinho.jsp"> Meu carrinho</a><br/>
```

Criação de uma padaria virtual

Como podemos ver no código acima, cada link correspondente a uma categoria será da seguinte forma:

```
<a href="produtos.jsp?cod_cat=1">Pães</a>
```

Criação de uma padaria virtual

Neste caso, a página "produtos.jsp" está recebendo como parâmetro a variável "cod_cat" com o valor "1", que corresponde ao código da categoria "Pães" na tabela "categorias" de nosso banco. Note que o código para conexão e consulta ao banco são análogos aos que faríamos em um servlet. Entretanto, para importar a biblioteca "java.sql.*" em JSP precisamos modificar <%@page %> presente no início dos códigos JSP:

```
<%@page contentType="text/html" pageEncoding="UTF-8"
        import="java.sql.*" %>
```

Criação de uma padaria virtual

Normalmente, não inserimos consultas ao banco dentro do JSP. Em geral, os servlets realizam a lógicas mais complexas e o JSP apenas apresenta os resultados.

E quando a conexão com o banco é realmente necessária dentro de um código JSP, costumamos utilizar classes que gerenciam as conexões e consultas.

Reaproveitamento de código

A parte superior da padaria virtual que contém o logotipo, a área de login e o menu aparecem em todas as subpáginas do site. Para não termos que copiar e colar o código em cada página, o JSP permite a criação de um trecho de código JSP que pode ser importado em diversas páginas.

Estas subpáginas em JSP não precisam conter toda a estrutura de um arquivo HTML. Basta que elas contenham o trecho exato de código que deverá ser importado. Para importar um arquivo deste tipo, definimos o atributo "file" da tag **<%@include %>**. O arquivo "cabecalho.jsp", por exemplo, poderia ser importado da seguinte forma:

```
<%@include file="cabecalho.jsp" %>
```

Reaproveitamento de código

Com isso, todo o conteúdo do arquivo será inserido no local onde estiver sendo importado, no momento em que a página for requisitada. Outra vantagem de reaproveitar o código é que, em caso de alteração em seu conteúdo, apenas um arquivo (neste caso, o "cabecalho.jsp") precisa ser modificado.

Catálogo de produtos dinâmicos

Os produtos da padaria virtual são exibidos na página em uma tabela, dois a dois, como podemos ver na Figura. Antes de tornar a página dinâmica, precisamos criar seu layout padrão. Cada item de tabela será semelhante ao trecho de código a seguir:

Catálogo de produtos dinâmicos

```
<td align="left" width="50%">
    <img style="margin-right: 20px;" src="img/pao.png" height="100"
         alt="Pão de Hamburguer com gergelim" align="left"/>
    <b>Pão de Hamburguer</b><br/><br/>

    Pão de Hamburguer com gergelim.<br/>
    Valor: <b>R$ 0.80</b><br/><br/>
    <input type="hidden" name="id" value="4" />

    Quantidade:
    <select name="qtd">
        <option>1</option>
        <option>2</option>
        <option>3</option>
        <option>4</option>
        <option>5</option>
        <option>6</option>
        <option>7</option>
        <option>8</option>
        <option>8</option>
        <option>10</option>
    </select>
    <p align="center"  >
        <input type="submit" value="Adicionar ao carrinho" name="botao"/>
    </p>
</td>
```

Catálogo de produtos dinâmicos

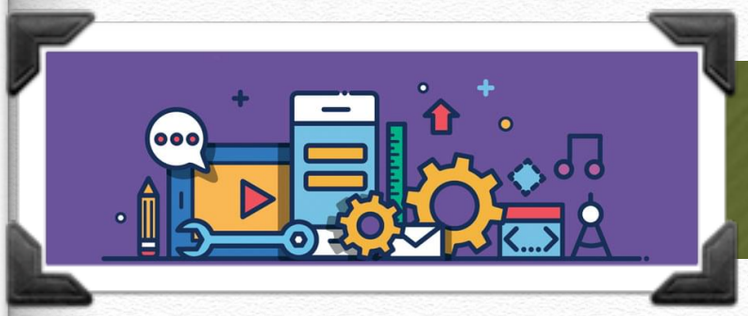
O código gera algo parecido. Lembrando que para a imagem aparecer é preciso colocá-la no servidor. Neste caso em uma pata chamada img.

Pão de Hamburguer

Pão de Hamburguer com gergelim.
Valor: **R$ 0.80**

Quantidade: 1

Adicionar ao carrinho



Catálogo de produtos dinâmicos

O código acima mostra todos os dados de um produto que precisam ser exibidos para o cliente (as informações que podem ser extraídas do banco foram sublinhadas). São elas: (1) localização da imagem do produto; (2) descrição do produto; (3) nome do produto; (4) preço do produto; e (5) código do produto.

Para que o conteúdo seja gerado dinamicamente, é preciso que o trecho de código acima seja gerado para cada produto existente no banco que precise ser exibido: na página de “Pães”, todos os produtos pertencentes à categoria “Pães”, etc.

Desenvolvimento de Aplicações Web

Catálogo de produtos dinâmicos

PLUS

Prof. Pedro Clarindo da Silva Neto

Fábrica de conexões

Em determinado momento de nossa aplicação, gostaríamos de ter o controle sobre a construção dos objetos da nossa classe. Muito pode ser feito através do construtor, como saber quantos objetos foram instanciados ou fazer o log sobre essas instanciações.
Às vezes, também queremos controlar um processo muito repetitivo e trabalhoso, como abrir uma conexão com o banco de dados. Tomemos como exemplo a classe a seguir que seria responsável por abrir uma conexão com o banco:

Fábrica de conexões

```java
public class ConnectionFactory {
    public Connection getConnection() throws ClassNotFoundException {

        String driver = "org.apache.derby.jdbc.ClientDriver";
        System.out.println("Conectando ao banco");
        try {
            System.out.println("Banco conectado");
             Class.forName(driver);
            return DriverManager.getConnection("jdbc:derby://localhost:1527/webpao", "pedro", "123456");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Fábrica de conexões

Agora é só chamar a nossa ConnectionFactory().

```
Connection con = new ConnectionFactory().getConnection();
```

Fábrica de conexões

Antes

```jsp
<a href="index.jsp"> Início</a>

<%
    String driver = "org.apache.derby.jdbc.ClientDriver";
    String url = "jdbc:derby://localhost:1527/webpao";
    String username = "pedro";
    String password = "123456";

    try {
        Class.forName(driver);
        Connection connection = DriverManager.getConnection(url, username, password);
        Statement statement = connection.createStatement();

        String query = "SELECT * FROM categoria_produto";
        ResultSet resultados = statement.executeQuery(query);

        while(resultados.next()){
            out.println("<a href=\"produtos.jsp?cod_cat=");
            out.println(resultados.getString("codigo_categoria")+"\">");
            out.println(resultados.getString("nome_categoria")+"</a>");
        }

        connection.close();

    } catch (ClassNotFoundException cnfe) {

        out.println("Error loading driver" + cnfe);

    } catch (SQLException sqle) {
        out.println("Error with connection" + sqle);

    }
    %>
    <a href="carrinho.jsp"> Meu carrinho</a><br/>
```

Fábrica de conexões

Depois

```
<%@page import="Teste.ConnectionFactory"%>
```

```
<a href="index.jsp"> Início</a>
<%
    try {
        Connection connection = new ConnectionFactory().getConnection();
        Statement statement = connection.createStatement();
        String query = "SELECT * FROM categoria_produto";
        ResultSet resultados = statement.executeQuery(query);

        while(resultados.next()){
            out.println("<a href=\"produtos.jsp?cod_cat=");
            out.println(resultados.getString("codigo_categoria")+"\">");
            out.println(resultados.getString("nome_categoria")+"</a>");
        }

        connection.close();

    } catch (ClassNotFoundException cnfe) {

        out.println("Error loading driver" + cnfe);

    } catch (SQLException sqle) {
        out.println("Error with connection" + sqle);

    }
    %>
    <a href="carrinho.jsp"> Meu carrinho</a><br/>
```

Referências

Programação para internet. / Hilário Seibel Júnior. – Vitória: Ifes, 2010.

CAELUM, Java e Orientação a Objetos, Apostila. [Internet: http://www.caelum.com.br/downloads/apostila/caelum-java-objetos-fj11.pdf]. Acesso em 04/03/2009.

_____. Java Web, FJ-21.

DEITEL, Harvey M.; DEITEL, Paul J., Java: Como Programar, São Paulo: Prentice-Hall, 2005.

GOODMAN, Danny, JavaScript a Bíblia, Ed. Campus, 2001

HALL, Marty; BROWN, Larry, Core Servlets e JavaServer Pages vo1 e vol 2, Ed. Ciência Moderna, 2005

HORSTMANN, Cay; CORNELL, Gary, Core Java 2, Fundamentos, São Paulo: Makron Books, Volume 1, 2000.

KURNIAWAN, Budi, Java para a Web com Servlets, JSP e EJB, Ed. Ciência Moderna, 2002

MUSCIANO, Chuck; KENNEDY, Bill, HTML: The definitive guide, Ed. Orelly, 1997

OLSON, Steven Douglas, Ajax com Java, Ed. Alta Books, 2007