

A Plataforma Java

Programação Web com Java



Desenvolvimento de Aplicações WEB na plataforma J2EE

Prof. Giuliano Prado de Moraes Giglio, M.Sc.

giucontato@gmail.com

<http://www.professorgiuliano.vai.la>

A Plataforma Java

Programação Web com Java



Instalação e preparação do ambiente



Prof. Giuliano Prado de Moraes Giglio, M.Sc.



Instalação e configuração do ambiente

- JDK 1.8.x ou superior instalado
- NetBeans 8.0 ou superior
- Servidor de Aplicação TOMCAT 7.0

- Faça o download do site <http://jakarta.apache.org>
- Descompacte a cópia para um diretório que será o diretório padrão do Tomcat e referido como <%catalina_home%>.
- Inclua a seguinte variável de ambiente:

CATALINA_HOME= C:\Arquivos de programas\Apache Software Foundation\Apache Tomcat 6.0.16

- Inclua a seguinte biblioteca no classpath

CLASSPATH=%CLASSPATH%;c:\.....\Tomcat-6.0.16\common\lib\servlet.jar

Instalação e configuração do Tomcat

■ Executando o Tomcat

- Para iniciar o servidor basta executar o arquivo

C:\Arquivos de programas\...\Apache Tomcat 6.0.16\bin\startup.bat

Para interromper a execução servidor basta executar o arquivo

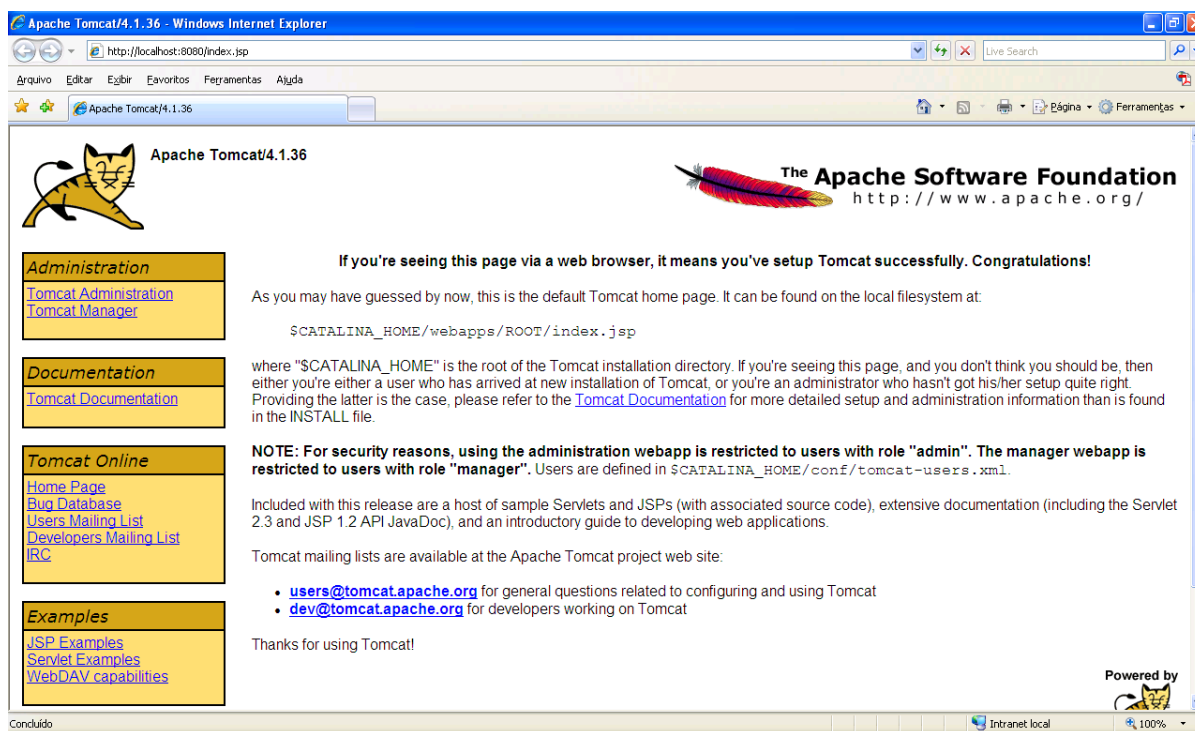
C:\Arquivos de programas\...\Apache Tomcat 6.0.16\bin\shutdown.bat

■ OBS:

- Caso ao iniciar o servidor apareça a mensagem de falta de espaço de memória, clique com o botão direito do mouse no arquivo .bat e edite as propriedades definindo o ambiente inicial com 4096.
- Ao entrar em execução o servidor lê as configurações do constantes no arquivo server.xml e por default se anexa à porta 8080 (podemos mudar a porta se quisermos)

Instalação e configuração do Tomcat

- Chamando através do browser
- Para acessarmos o servidor Web, bastar abrirmos o browser utilizado (IE no nosso caso) e digitarmos o seguinte endereço:
- **http://localhost:8080**

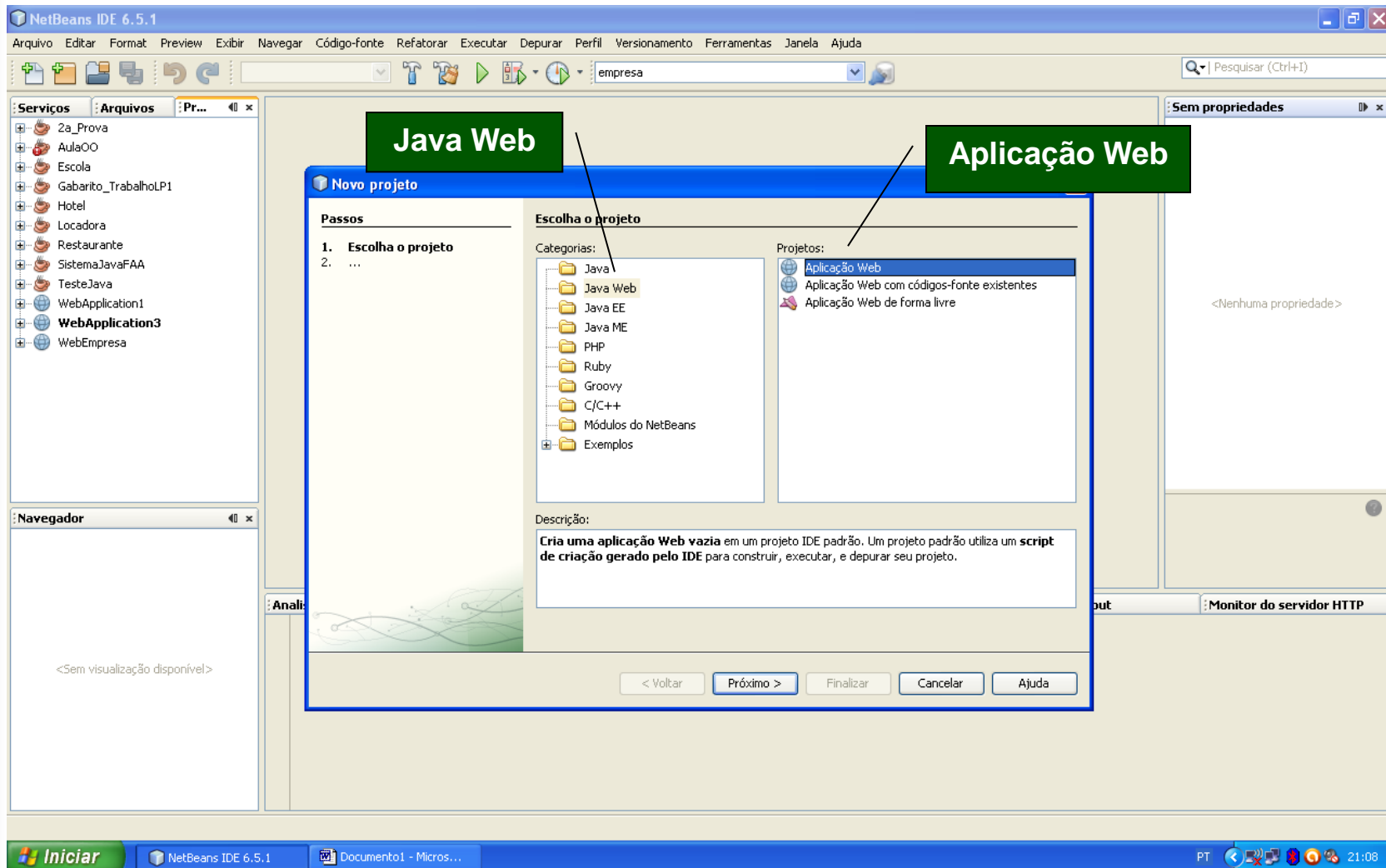


Instalação e configuração do Tomcat

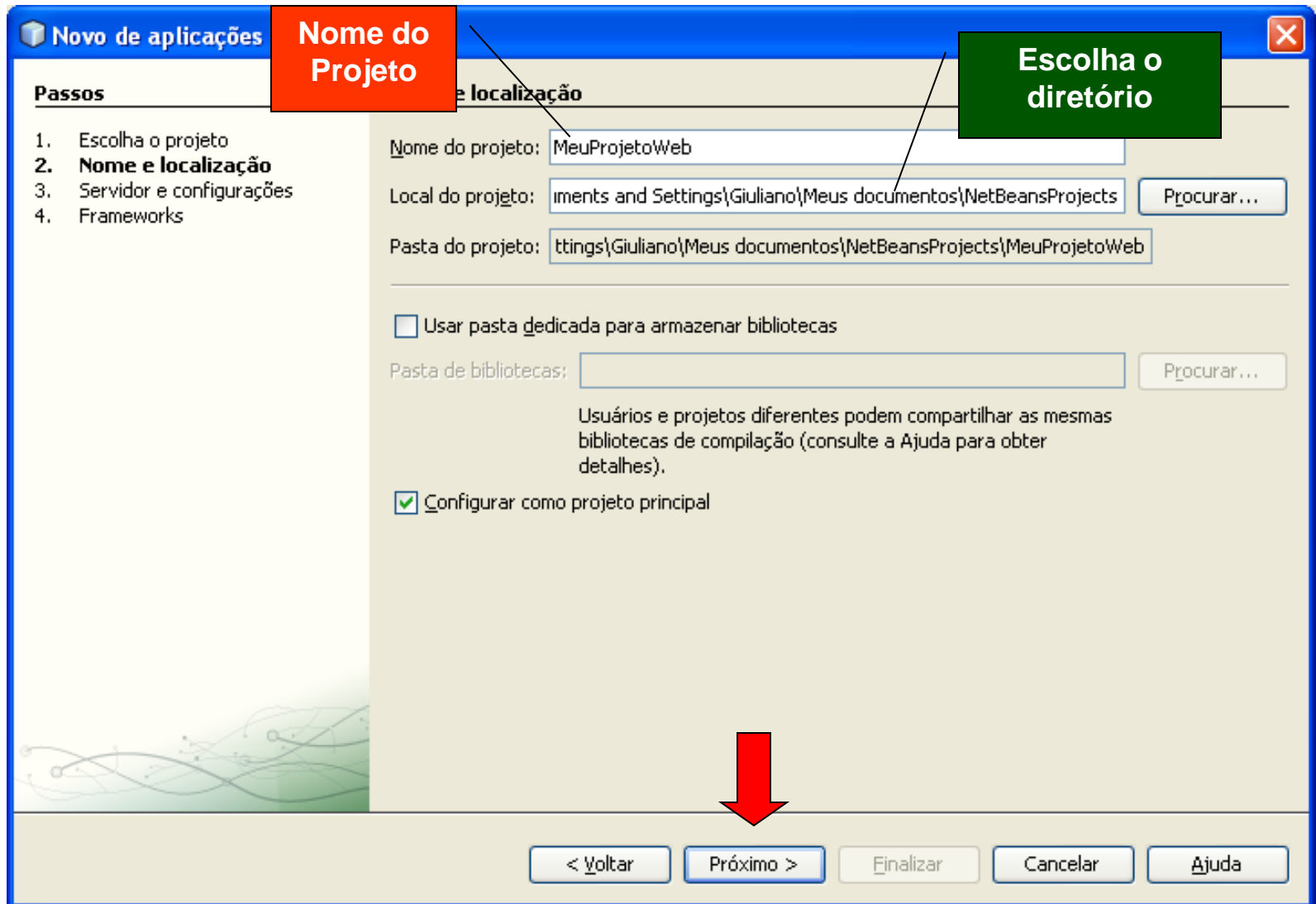
■ Diretórios do Tomcat

Diretório	Descrição
bin	Scripts de inicialização e encerramento e outros arquivos
classes	Classes não empacotadas globais e aplicativos web
conf	Arquivos de configuração, incluindo server.xml e web.xml
server	Arquivos de arquivamento do Tomcat
lib	Classes comuns em arquivos .jar
logs	Arquivos de registro do tomcat
common	Classes comuns para aplicativos web e catalina
webapps	Aplicativos servlets e jsp
work	Servlets resultantes de páginas jsp

Criando um projeto Web pelo NetBeans



Criando um projeto Web pelo NetBeans



The screenshot shows the 'Novo de aplicações' (New Application) dialog box in NetBeans. The 'Passos' (Steps) list on the left indicates the current step is '2. Nome e localização' (Name and location). The dialog is divided into two main sections: 'Nome do Projeto' (Project Name) and 'Escolha o diretório' (Choose the directory). The 'Nome do projeto' field contains 'MeuProjetoWeb'. The 'Local do projeto' field contains 'Documents and Settings\Giuliano\Meus documentos\NetBeansProjects', with a 'Procurar...' (Browse...) button next to it. The 'Pasta do projeto' field contains 'Documents and Settings\Giuliano\Meus documentos\NetBeansProjects\MeuProjetoWeb'. Below these fields, there is a checkbox for 'Usar pasta dedicada para armazenar bibliotecas' (Use dedicated folder for storing libraries), which is currently unchecked. A text box for 'Pasta de bibliotecas' (Library folder) is also present, with a 'Procurar...' button. A note below this text box states: 'Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes)'. At the bottom, there is a checkbox for 'Configurar como projeto principal' (Configure as main project), which is checked. A large red arrow points down to the 'Próximo >' (Next) button at the bottom of the dialog. Other buttons at the bottom include '< Voltar' (Back), 'Finalizar' (Finish), 'Cancelar' (Cancel), and 'Ajuda' (Help).

Novo de aplicações

Nome do Projeto

Escolha o diretório

Passos

1. Escolha o projeto
2. **Nome e localização**
3. Servidor e configurações
4. Frameworks

Nome do projeto: MeuProjetoWeb

Local do projeto: Documents and Settings\Giuliano\Meus documentos\NetBeansProjects **Procurar...**

Pasta do projeto: Documents and Settings\Giuliano\Meus documentos\NetBeansProjects\MeuProjetoWeb

☐ Usar pasta dedicada para armazenar bibliotecas

Pasta de bibliotecas: **Procurar...**

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Configurar como projeto principal

< Voltar **Próximo >** **Finalizar** **Cancelar** **Ajuda**

Criando um projeto Web pelo NetBeans

Novo de aplicações Web

Passos

1. Escolha o projeto
2. Nome e localização
- 3. Servidor e configurações**
4. Frameworks

Servidor e configurações

Adicionar ao aplicativo corporativo: <Nenhuma>

Servidor: Apache Tomcat 6.0.16
Apache Tomcat 6.0.16
GlassFish V3

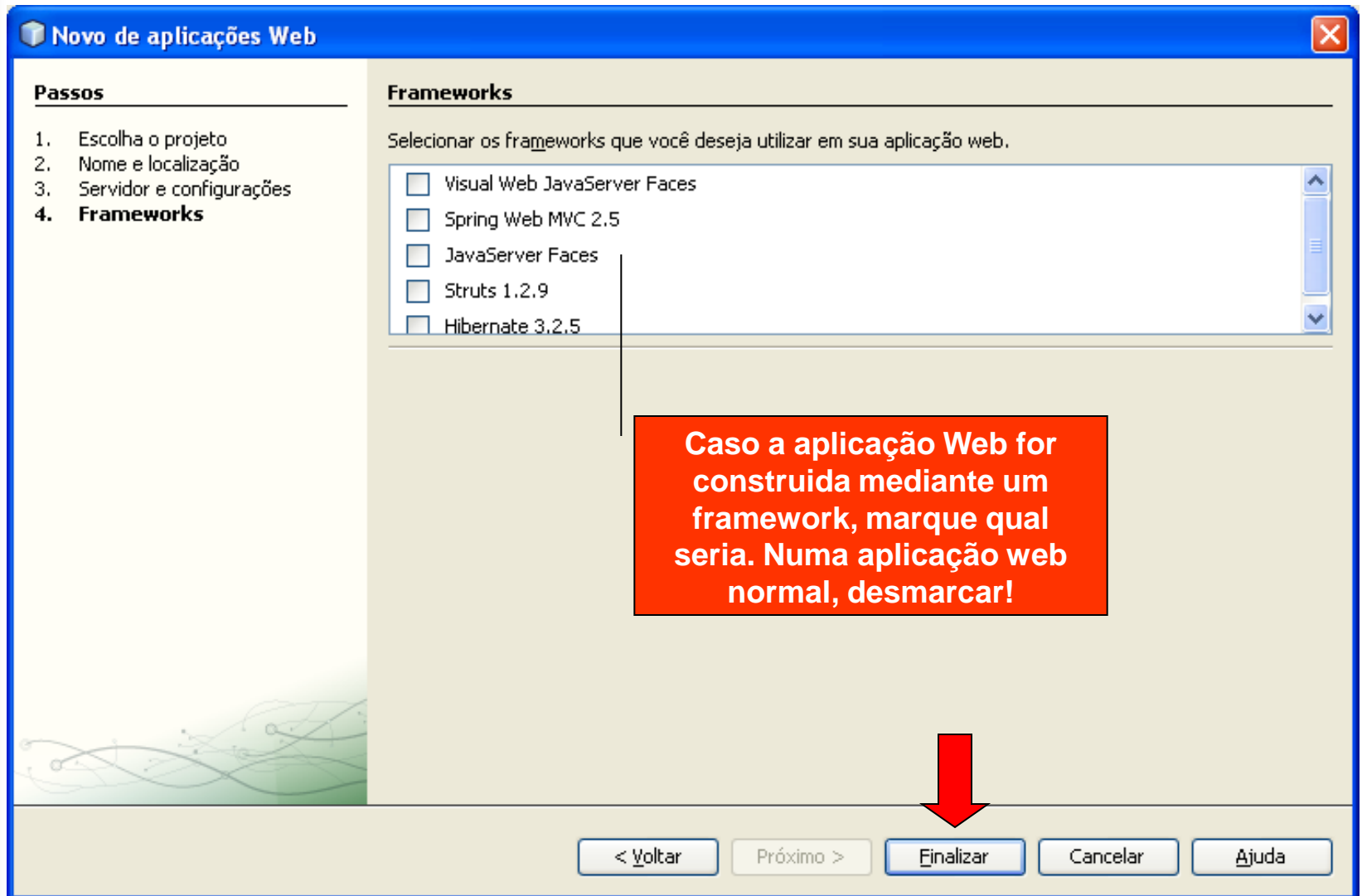
Versão do Java EE: Java EE 5

Caminho do contexto: /MeuProjetoWeb

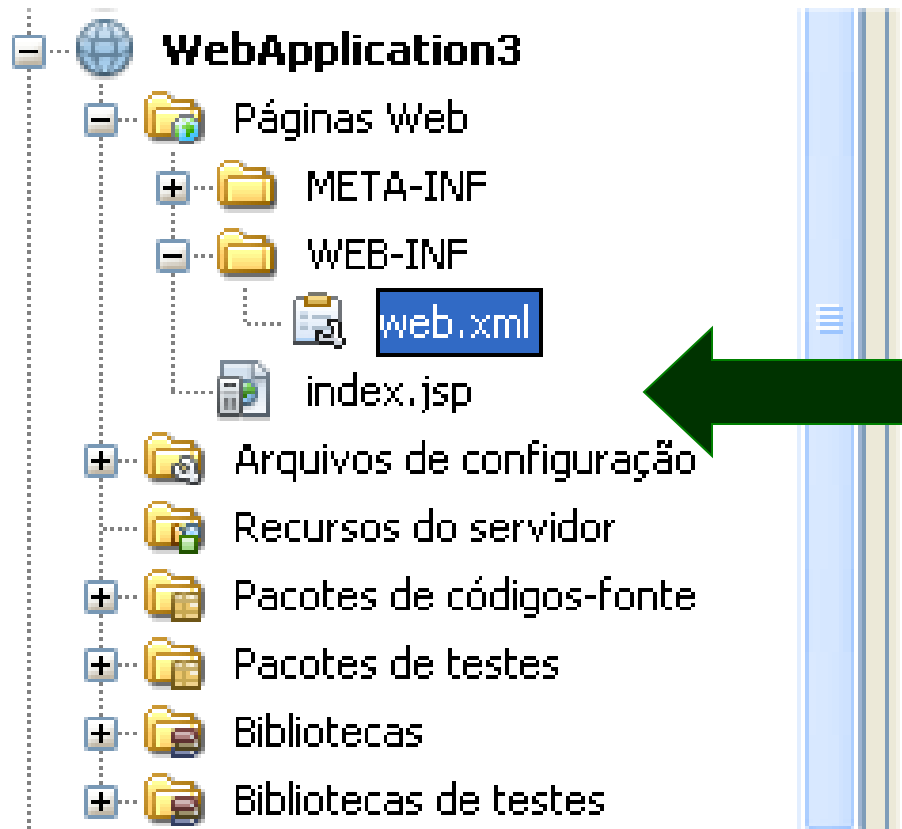
Escolha o servidor de Aplicações

< Voltar Próximo > Finalizar Cancelar Ajuda

Criando um projeto Web pelo NetBeans



Estrutura de um projeto Web pelo NetBeans



A Plataforma Java

Programação Web com Java



JSP

Java Server Page



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





Tecnologias Java para WEB

■ Servlets

- **Programas** escritos em JAVA que são executados no **lado servidor**.

■ Java Server Page (JSP)

- **Tecnologia** que permite a mistura permite misturar **código HTML estático** com **conteúdo dinâmico** gerado pelos **Servlets**.

■ Applets

- **Programas** escritos em JAVA que são executados no **lado cliente**.



JSP e Servlets

- Rodam no servidor com o objetivo de monitorar requisições
- Web Servers para Servlets/JSP
 - Apache Tomcat
 - IBM Websphere
 - JBoss
 - GlassFish
 - Sun Application Server
 - Mais Servidores:
 - ✓ <http://java.sun.com/products/servlet/industry.html>

JSP

- O texto HTML é escrito junto com as tags JSP e código Java
- Não é uma idéia nova sendo usado em tecnologias concorrentes: ASP, PHP.

<HTML>

<HEAD>

</HEAD>

<BODY>

<% out.println (“Jsp é muito fácil”); %>

</BODY>

</HTML>



Uso de JSP

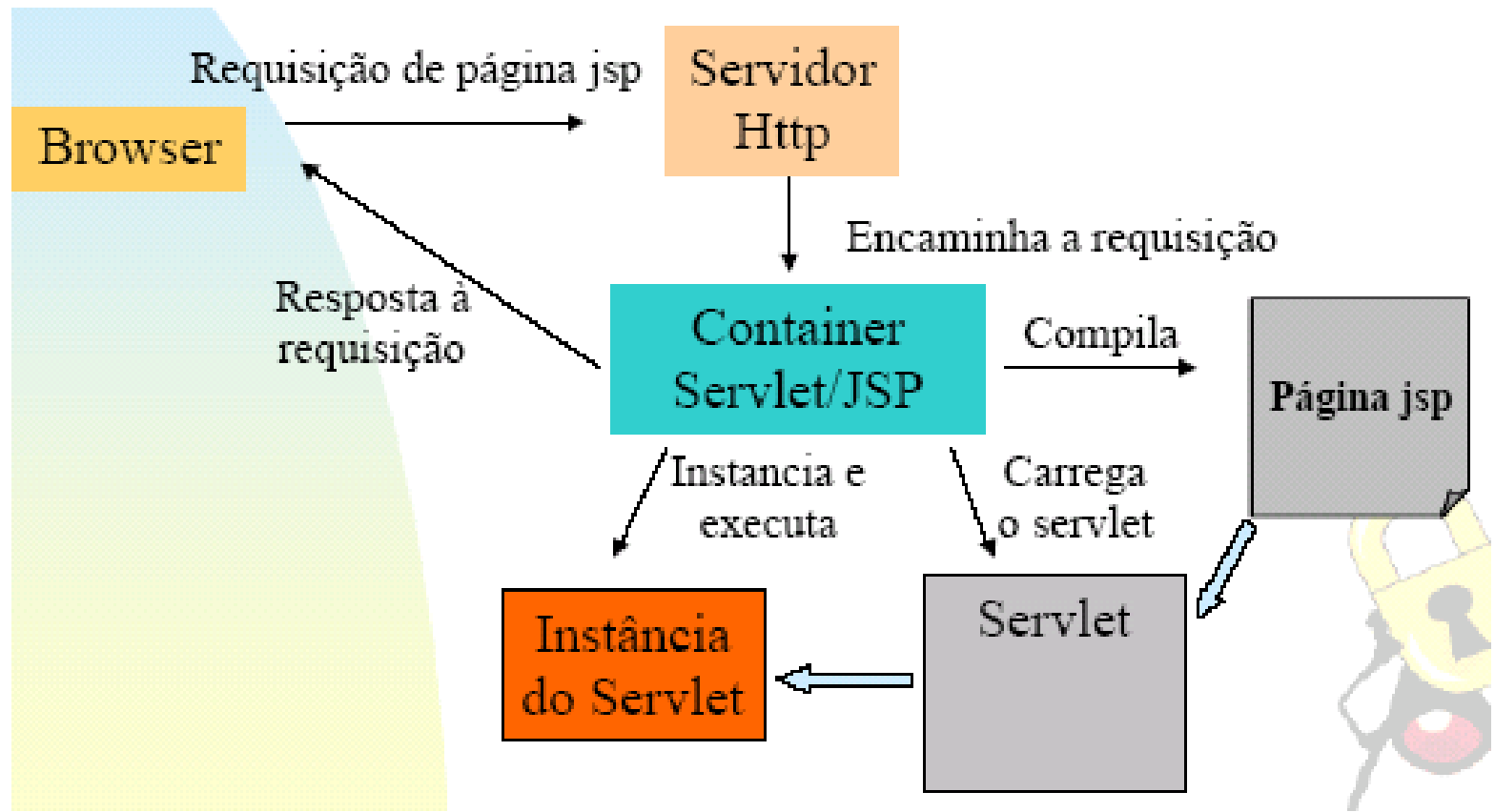
- Páginas JSP também necessitam de Web Servers específicos.
- Em geral, um servidor de servlets também é um servidor de JSP (no nosso caso, o Tomcat)
- Para disponibilizar um arquivo JSP basta gravá-lo em qualquer pasta visível do servidor com extensão jsp



Execução de JSP

- Quando um usuário faz uma requisição a uma página JSP:
 - Se for a primeira, a página jsp é convertida em um servlet e compilada
 - Este servlet é invocado e retorna como resposta uma string HTML
 - Esta string é retornada para o cliente
 - O servlet gerado é colocado no diretório works do Tomcat.
- Todo o processamento ocorre no servidor, apenas o HTML é retornado

Como os dados são passados do cliente para o servidor





Elementos JSP

- Uma página JSP pode ter três tipos de elementos:
 - Elementos de Script que são inseridos diretamente no servlet
 - Diretivas que permitem a manipulação do código gerado como servlet
 - Ações que possibilitam o uso de outros componentes, como Java Beans



JSP

- A maior parte de um arquivo jsp consiste em tags HTML
- São passadas como saída do servlet
- Páginas jsp são criadas com ferramentas HTML
- Comentários JSP
- `<%-- -->` não são passados para o cliente
- **Comentários HTML**
- `<!-- comentário -->` **Passados para o cliente**

JSP – Sintaxe Básica

- Os tags JSP possuem a seguinte forma geral:

<% Código JSP %>

- Exemplo

<%

int val = Math.random() * 100;

out.println("Valor sorteado = " + val);

%>

- O primeiro caractere % pode ser seguido de outros caracteres que determinam o significado preciso do código dentro do tag.

JSP – Sintaxe Básica

- **Scriptlets**

<% código java %>

- Para tarefas mais complicada existem os Scriptlets permitem inserir trechos de código em Java.

- **Exemplo:**

<%

String queryData = request.getQueryString();

out.println("Dados GET associado: " + queryData);

%>

JSP – Sintaxe Básica

- Expressões

<%= expressões %>

- Expressões são avaliadas, convertidas para String e colocadas na página enviada. A avaliação é realizada em tempo de execução, quando a página é requisitada.

- Exemplos:

<%= new java.util.Date() %>

<%= variavel %>

<%= calcularDobro(valor) %>

JSP – Sintaxe Básica

■ Exemplo: Previsão do Tempo

```
<%  
    if (Math.random() < 0.5)      ⇒ trecho Java  
    {  
%>  
    Hoje vai <B>fazer sol</B>!    ⇒ trecho HTML  
  
<%  
    }  
    else                          ⇒ trecho Java  
    {  
%>  
    Hoje vai <B>chover</B>!      ⇒ trecho HTML  
<% } %>
```


JSP – Sintaxe Básica

- Declarações

<%! Código Java %>

- Exemplo

```
<%! private int numAcesso = 0; %>
```

**Acessos desde até o momento desde o início da
sessão:**

```
<%= ++ numAcesso %>.
```

- Uma declaração JSP permite definir variáveis e métodos que são inseridos no corpo do servlet.
- Como as declarações não geram saída, elas são normalmente usadas em combinação com expressões e scriptlets.
- As variáveis declaradas desta forma são variáveis de instância. As variáveis declaradas dentro de scriptlets são variáveis locais e não acessíveis os métodos declarados.

JSP - Considerações

- Os objetos implícitos não são acessíveis para métodos declarados no contexto de uma página JSP.
- Se precisarmos acessar algum objeto implícito de dentro de um método jsp devemos passá-los como parâmetro para o método.

<%!

```
private void metodoXXX(HttpSession s) {  
    processaS(s);  
}
```

%>

```
<% metodoXXX(session) ; %>
```

Objetos que podem ser usados em páginas Jsp diretamente

- Para simplificar o código JSP existem um conjunto de objetos implícitos que podem ser usados pelo programador. Os mais importantes são:

• HttpServletRequest	request
• HttpServletResponse	response
• HttpSession	session
• PrintWriter	out
• ServletContext	application
• ServletConfig	config
• PageContext	pageContext
• HttpJspPage	page
• Throwable	exception

- **Exemplo**

Host: <%request.getServerName()%>



Objetos Implícitos

■ request - classe `HttpServletRequest`

- Encapsula os dados do cliente: Contém os dados enviados pelo cliente para serem processados no servidor
- Formulários, cookies, parâmetros

■ response - classe `HttpServletResponse`

- Encapsula os dados para o cliente: contém os dados que devem ser enviados como resposta ao cliente
- Header, cookies, conteúdo

Objetos Implícitos

■ request e response

- Exemplos:

```
<% String nome = request.getParameter("nome");  
    response.sendRedirect(http://www.sun.com); %>
```

■ out

- Este objeto funciona para escrita de conteúdo HTML na página onde está aplicado, a qual será enviada ao usuário-cliente.
- Exemplos:

```
<% out.println(produto.getNome()); %>  
<% out.println("<b>Cadastro feito!</b>"); %>
```



Request – HttpServletRequest

- Objeto que recebe o resultado do envio de dados de um form html.
- Recuperação de parâmetros (a partir do objeto **request**):
 - **getParameter**: para pegar um parâmetro específico
 - **getParameterNames**: para recuperar todos os parâmetros passados
 - **getParameterNames**: retorna uma enumeração (classe Enumeration) contendo o nome dos parâmetros.
 - **getQueryString**: retorna a string de consulta enviada para o Servlet.
 - **getParameterValues**: retorna mais de um valor selecionado em parâmetros de formulários com retorno de múltiplos valores. Ex: ComboBoxes, CheckBoxes, etc.



Response – HttpServletResponse

- `setContentType`: método para setar o tipo de dados a ser enviado como resposta;
- `getWriter`: pegar o contexto para escrita de dados para o objeto response;
- `setHeader`: permite que se acrescente um campo nome/valor ao cabeçalho de resposta.
- `sendRedirect`: permite redirecionar o usuário para uma outra página
- `sendError`: enviar mensagem de erro de acordo com o erro capturado.



Primeiro exemplo de JSP

- Crie um arquivo, ***agradecimento.jsp*** com o seguinte conteúdo

```
<HTML>
```

```
<HEAD>
```

```
  <TITLE> Usando JSP </TITLE>
```

```
</HEAD>
```

```
  <BODY BGCOLOR=#DADADA>
```

```
    Obrigada por acessar a página do
```

```
      <I><%= request.getParameter("titulo") %></I>
```

```
  </BODY>
```

```
</HTML>
```

- Execute esta página, passando o parâmetro:

<http://localhost.../agradecimento.jsp?titulo=Curso JavaWeb>

Objetos Implícitos

■ session

- Objetos para controle de sessão.
- Exemplo:

```
<% String contador = (String) session.getAttribute("contador"); %>
```

■ page

- Possui os métodos `jspInit()` e `jspDestroy()`, que podem ser chamados no corpo da página jsp para configurar algum recurso na inicialização ou destruição da página JSP. É só implementar estes métodos no corpo da página
- Exemplo:

```
<% public void jspInit(){  
    System.out.println("Iniciando");  
}  
%>
```

- **pageContext, page, application e exception são utilizados da mesma forma que em um Servlet.**

JSP - Exercícios

**Implementar os exercícios
01 e 06 da lista**





Diretivas JSP

- Afetam a estrutura geral do servlet gerado da página JSP
- Possuem o seguinte formato:

`<%@diretiva atributo="valor" %>`



Tipos de Diretivas

- Existem três tipos de diretivas JSP:
- **page**, para importação de classes, alteração do tipo do conteúdo, etc.
- **include**, para inclusão de arquivos durante a execução do JSP
- **taglib**, para definição de tags próprias usadas em bibliotecas de tags



Propósito da diretiva page

■ Permite o controle

- Quais classes serão importadas
- Quais classes o servlet resultante herda
- Como o multiprocessamento é tratado
- Se o servlet resultante participará de sessões
- Como tratar erros resultantes

Diretiva page Import

■ Atributo import

- Usado para importar classes para o servlet gerado pelo JSP
- Pode aparecer várias vezes no JSP
- Exemplo:

```
<%@page import = "java.util.*" %>
```

- Obs: os pacotes a serem importados pela diretiva import, caso não sejam os pacotes básicos java ou tomcat, devem estar no diretório web-inf/classes de sua aplicação, seguindo a estrutura de diretório do pacote.



Diretiva page contentType

- **Atributo contentType**
- Usado para alterar o formato MIME do texto de saída
- Exemplo:

```
<%@page contentType="text/html"%>
```

Diretiva page contentType

- Exemplo:

```
<%@ page contentType="text/plain" %>
```

- possui o mesmo efeito do scriptlet

```
<% response.setContentType("text/plain"); %>
```

- Exemplo (geração de saída planilha excel):

```
<%@ page contentType="application/vnd.ms-excel" %>
```

```
<%-- Entre as colunas temos tabulação e não espaços,--%>
```

```
1997 1998 1999 2000 2001
```

```
12.3 13.4 14.5 15.6 16.7
```




Diretiva page Session

- **Atributo session**
- Indica se a página em questão faz parte de uma sessão sendo que o seu valor padrão é true
- Ex:

```
<%@page session="false" %>
```
- Neste caso a variável pré-definida session não pode ser acessada

Diretiva page extends

- **Atributo extends**
- Altera a superclasse do servlet gerado
- Exemplo:

```
<%@page extends="MeuServlet.class"%>
```

- Deve ser usado com extrema cautela e só em casos de extrema necessidade.



Diretiva page errorPage

- **Atributo errorPage**

- Indica o nome da página que deve ser mostrada em caso de erro

- Exemplo:

<%@page errorPage="URL relativa"%>



Diretiva page isErrorPage

- **Atributo isErrorPage**
- Indica se a página atual pode ser usada como página de erro
- O valor padrão deste atributo é false
- Exemplo:
`<%@page isErrorPage="true"%>`

Exemplo errorPage

(página Converte.jsp)

...

<BODY>

<%@page errorPage="PaginaErros.jsp" %>

<TABLE BORDER=5 ALIGN="CENTER">

 <TR><TH>Conversão de Numeros</TH></TR>

</TABLE>

<% !

private double toDouble(String value)

{

 return (Double.valueOf(value).doubleValue());

}

%>

Exemplo isErrorPage

(Página PaginaErros.jsp) - simples

...

```
<BODY>
```

```
<%@page isErrorPage="true" %>
```

```
<TABLE BORDER=5 ALIGN="CENTER">
```

```
  <TR><TH>Erros Encontrados</TH></TR>
```

```
</TABLE>
```

```
<P>
```

```
  <I>Ocorreu um Erro de Conversão</I>
```

```
</BODY>
```

...

Exemplo isErrorPage

(Página PaginaErros.jsp) – exibindo exceções

...

<BODY>

<%@page isErrorPage="true" %>

<TABLE BORDER=5 ALIGN="CENTER">

<TR><TH>Erros Encontrados</TH></TR>

</TABLE>

<P>

PaginaErros.jsp executou o seguinte erro:

<I><%= exception %></I>.

Este problema ocorreu na seguinte parte:

<PRE>

<% exception.printStackTrace(
new java.io.PrintWriter(out)); %>

</PRE>

...



Diretiva include

- Usada para incluir outros arquivos em páginas JSP
- Possui dois formatos, o primeiro inclui arquivos em tempo de compilação e o segundo em tempo de requisição
- A diretiva deve aparecer no ponto em que o arquivo será incluído



Incluindo arquivos (compilação)

- Sintaxe:
<%@include file="URL relativa"%>
- Arquivos incluídos podem ser arquivos JSP
- Se o arquivo incluído mudar, todos os JSP que o utilizam devem ser recompilados



Incluindo arquivos (requisição)

- Sintaxe:

<jsp:include page="URL relativa" flush="true"%>

- Os arquivos incluídos não podem conter comandos JSP
- Se o arquivo mudar, ele será lido novamente.

Exemplo de JSP com Includes

- Faça um arquivo chamado **cabecalho.jsp**
 - Coloque o cabeçalho padrão de jsp
 - Coloque nele o nome “Concessionária Carro Travado” (centralizado);
 - Um endereço fictício
 - Três links para as paginas:
 - ✓ ***Loja.jsp, ShowRoom.jsp, Contato.jsp***
- Faça um arquivo chamado **rodape.jsp**
 - Coloque nele o email de contato, seu nome completo, e um link para a Honda (www.honda.com.br)
- Crie a página **Loja.jsp** e coloque um texto explicativo sobre a concessionária (pode pegar na internet);
 - Faça os includes devidos.

A Plataforma Java

Programação Web com Java



Gerenciamento de Sessão



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





Técnicas para gerenciamento de sessão

- Cookies
- Objetos de sessão
- Outras adaptadas (envio de campos hidden, envio de identificadores junto com a chamada dos servlets).



Uso de Cookies

- Um cookie é uma informação que é passada para a frente e para trás na solicitação http e resposta.
- Cookies não são usados apenas para gerenciamento de sessão. São também usados para lembrar qualquer informação que desejarmos.
- Uso indiscriminado traz problemas.....
- Como funciona
 - O JSP envia um nome e um valor para o cliente.
 - O cliente retorna o mesmo nome e mesmo valor quando ele se conectar novamente com aquele site ou domínio
- Usos de cookies
 - Identificar um usuário em uma sessão de comércio eletrônico
 - Passar login e senha para o usuário.
 - Customizar um site de acordo com as preferências do usuário
 - Propaganda focada



Problemas no uso de cookies

■ Privacidade

- Servidores podem lembrar suas ações anteriores
- Se você disponibiliza informações pessoais, os servidores podem ligar estas informações com suas ações anteriores.
- Servidores podem compartilhar seus cookies com outros....
- Sites mal projetados armazenam informação confidencial (numero de cartão de crédito diretamente em cookies....)

■ O que devemos fazer?

- Se o uso de cookies não for tão crítico para a funcionalidade de seu site, evite servlets que falham quando cookies estão desabilitados nos browsers.
- NÃO COLOQUE INFORMAÇÃO CONFIDENCIAL EM COOKIES.

Criação e recuperação de um cookie

- Criar um objeto da classe Cookie, passando como parâmetros para o método construtor o nome e o valor do cookie (ambos string).

```
Cookie c1 = new Cookie("nome", "giuliano");
```

- Depois, você pode acrescentar o cookie a saída http.

```
response.addCookie(c1);
```

- Para tornarmos um cookie persistente, usamos o método `setMaxAge(milisegundos)`.

```
Cookie c = new Cookie("nome", "valor") ;  
c.setMaxAge (60*60*24*30) ; //persiste por 30 dias....  
response.addCookie (c) ;
```

- OBS: como os cookies são escritos nos cabeçalhos de solicitação e resposta, não é possível acrescentar um cookie depois de uma saída ter sido escrita ao objeto `HttpServletResponse`



Cookies persistentes

- Para recuperar cookies, usamos o método `getCookies`, da interface `HttpServletRequest`

```
Cookie[ ] cookies = request.getCookies();
```

- **Exemplo**

```
Cookie[ ] cookies = request.getCookies();
```



Exemplo do uso de Cookies

- Faça uma página ***formLogin.jsp***
 - Construa um formulário de Login, com usuário e senha
 - Defina o Action do formulário para a página ***processaLogin.jsp***, onde os dados serão enviados.
- Faça a página ***processaLogin.jsp***
 - Retire todo o conteúdo desta página;
 - Codifique a autenticação de login, segundo o código abaixo:

Armazenando em Cookies

<%

```
String u = request.getParameter("pUser");  
String s = request.getParameter("pSenha");
```

```
if ((u.equals("giuliano")) &&  
(s.equals("123456")))  
{
```

```
    Cookie c1 = new Cookie("usuario",u);  
    c1.setMaxAge(60*60*24*30);  
    response.addCookie(c1);
```

%>



Exemplo do uso de Cookies

- Faça uma página ***BemVindo.jsp***
 - Esta página irá recuperar o usuário armazenado em Cookies, através do login feito, mostra-o nesta página junto com uma mensagem de bem-vindo!
 - O código abaixo representa a recuperação de informações em cookies:

Recuperando Cookies

<%

```
Cookie cookies [ ] = request.getCookies();
if (cookies!= null)
{
    int tamanho = cookies.length;
    for (int i=0; i<tamanho; i++)
    {
        Cookie cookie = cookies[i];
        if (cookie.getName().equals("usuario"))
            out.println(cookie.getValue());
    }
}
```

%>



Uso de Session

- Java possui uma classe HttpSession criada especificamente para o gerenciamento de sessão
- Para cada usuário, o JSP possui um objeto da classe HttpSession implícito, que é associado ao usuário e só pode ser acessado por aquele usuário em particular.
- Age como uma hashtable, onde você pode armazenar qualquer quantidade **de pares chave/objeto**.
 - Este objeto HttpSession fica acessível para outros arquivos Jsp (ou Servlets) no mesmo aplicativo.
 - Para recuperar um objeto previamente armazenado, você só precisa **passar a chave**.



Vantagens

- Geralmente o gerenciamento de sessão com o HttpSession usa “por baixo dos panos” cookies.
- No entanto, caso cookies não esteja habilitado no browser, ele passa automaticamente para reescrita de url, sem intervenção do desenvolvedor.
- Pode armazenar objetos descendentes da classe Object
- Fácil recuperação da sessão do usuário atual

HttpSession

- Contém métodos para:
 - Recuperar/retirar/armazenar dados na sessão:
getAttribute(nome)
removeAttribute(nome)
setAttribute("nome", valor)
 - Descartar sessões:
setMaxInactiveInterval(segundos)
invalidate()
 - Recuperar/criar sessões (no caso de Servlets):
getSession()
getSession(true)



HttpSession

- Recuperando Sessões

session =request.getSession(true);

- O valor true indica que uma sessão deve ser criada se não existir
- Para verificar se uma sessão é nova, utiliza-se o método isNew()

```
if ( session.isNew() )
```

```
{.....}
```



HttpSession

- Associando Valores: podem ser associados objetos de qualquer tipo
- Métodos
 - `getAttribute`
 - `setAttribute`
 - `removeAttribute`
 - `getAttributeNames`



Exemplo de Uso de Sessão

- Refaça o exercício de Login anterior, agora colocando o usuário na sessão e recuperando da mesma na página BemVindo.jsp

JSP - Exercícios

**Implementar o exercício
08 da lista**



A Plataforma Java

Programação Web com Java



Servlets



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





Servlets

- Definição: Código Java, carregado em um Servidor Web, que é usado para tratar requisições de clientes, como CGI.
- Características:
 - Persistente;
 - Seguro;
 - Independente de plataforma;
 - Acessa Banco de Dados; e
 - Pode ser integrado com Applets, no cliente, iniciando uma conversação

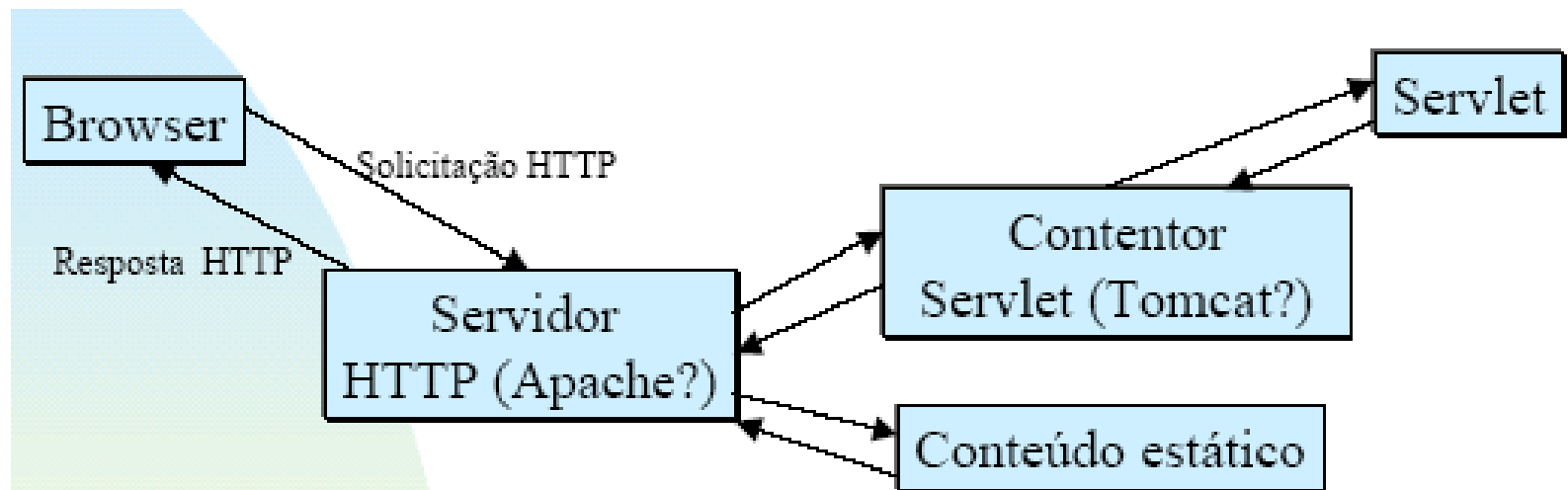


Servlets

■ Uso de Servlets

- Geração de Conteúdo HTML Dinâmico
- Sincronização de requisições
- Redirecionamento de requisições
- **Lê dados enviados explicitamente pelo cliente (formulários HTML oapplets)**
- **Acessa banco de dados/aplicações legadas/ regras de negócio e retorna para o cliente (em HTML)**

Como executar um servlet



Servlets

- **A API Servlet**
- A API Servlet é composta por um conjunto de interfaces e Classes. O componente mais básico da API é interface Servlet. Ela define o comportamento básico de um Servlet.

```
public interface Servlet {  
    public void init(ServletConfig config) throws  
        ServletException;  
    public ServletConfig getServletConfig();  
    public void service(ServletRequest req, ServletResponse  
        res) throws ServletException, IOException;  
    public String getServletInfo();  
    public void destroy();  
}
```

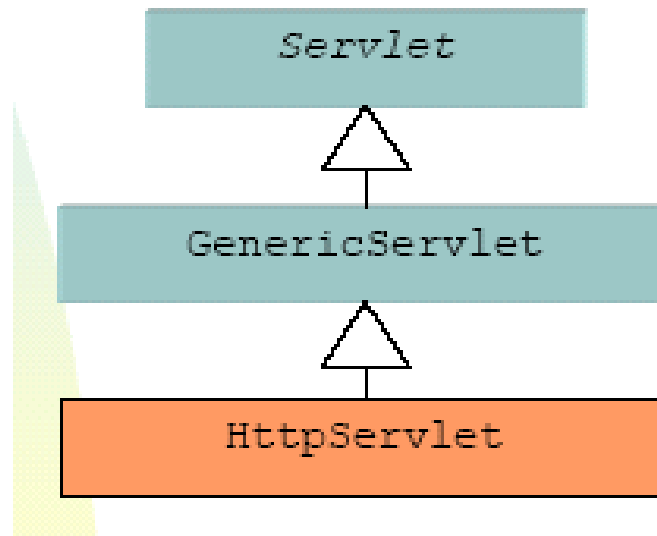


Servlets

- **A API Servlet**
- O método **service()** é responsável pelo tratamento de todas as requisições dos clientes.
- Os métodos **init()** e **destroy()** são chamados quando o Servlet é carregado e descarregado, respectivamente.
- O método **getServletConfig()** retorna um objeto ServletConfig que contém os parâmetros de inicialização do Servlet.
- O método **getServletInfo()** retorna um String contendo informações sobre o Servlet, como versão e autor.

Servlets

- A API Servlet
- Tendo como base a interface Servlet o restante da API Servlet se organiza hierarquicamente como mostra a figura abaixo.





Servlets

- A classe `GenericServlet` implementa um servidor genérico e geralmente não é usada.
- A classe **`HttpServlet`** é mais utilizada e foi especialmente projetada para lidar com o protocolo HTTP.

```
public abstract class HttpServlet extends GenericServlet  
    implements java.io.Serializable
```



Servlets

- A classe derivada da `HttpServlet` deve sobrescrever pelo menos um dos métodos abaixo:
 - **`doGet`** para tratar requisições HTTP GET.
 - **`doPost`** para tratar requisições HTTP POST.
 - **`doPut`** para tratar requisições HTTP PUT.
 - **`doDelete`** para tratar requisições HTTP DELETE.
- O método `service()`, que recebe todas as requisições, em geral não é sobrescrito, sendo sua tarefa direcionar a requisição para o método adequado.

Servlets

- Uma aplicação ou simplesmente um Servlet herda da classe HttpServlet.
- Os Servlets são mapeados no arquivo web.xml de sua aplicação, segundo exemplo abaixo:

```
?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>Teste</servlet-name>
        <servlet-class>TestaServlet</servlet-class>
    </servlet>
</web-app>
```

Primeiro Exemplo de Servlets

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class TestaServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE> Teste de Servlet</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<B>Primeiro Teste de Servlets de Desenvolvimento WEB");
        out.println("</BODY>");
        out.println("</HTML>");
    } }
}
```



Executando o Servlet

- Chame seu servlet a partir de um browser web:

<http://localhost:808/.../TestaServlet> ou

<http://localhost:8080/.../Teste>

Servlets

- Quando implementamos um servlet, geralmente temos que importar dois pacotes básicos:
 - javax.servlet: contém classes básicas para a programação servlet
 - javax.servlet.http: contém classes e interfaces mais avançadas para programação.
- O método doGet() recebe dois objetos: um da classe **HttpServletRequest** e outro da classe **HttpServletResponse**.
- O **HttpServletRequest** é responsável pela comunicação do cliente para o servidor e o **HttpServletResponse** é responsável pela comunicação do servidor para o cliente.
- Primeiramente é usado o método setContentType() para definir o tipo do conteúdo a ser enviado ao cliente. Esse método deve ser usado apenas uma vez e antes de se obter um objeto do tipo PrintWriter ou ServletOutputStream para a resposta.
- Se o programador desejar enviar a resposta em bytes deve usar o método getOutputStream() para obter um objeto OutputStream.
- Uma vez carregado o Servlet não é mais descarregado, a não ser que o servidor Web tenha sua execução interrompida.



Servlets – doGet e doPost

- Existem duas maneiras de se enviar dados pelo browser: Get e Post
- Como o tratamento é o mesmo chama-se um método a partir do outro
 - Get:: dados vem junto com a URL
 - Post:: dados vem separados da URL



Servlets – doGet e doPost

```
public void doGet (HttpServletRequest request,  
                  HttpServletResponse response)  
{  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
  
    String nome = request.getParameter("nome");  
    out.println("Texto recebido: " + nome);  
    out.close();  
}
```



Servlets – Service

- A cada nova requisição o servidor cria um novo thread e chama *service*
- O método *service* verifica a requisição e chama o método apropriado
- Se for sobreposto deverá fazer todo o tratamento de verificação e chamada dos métodos



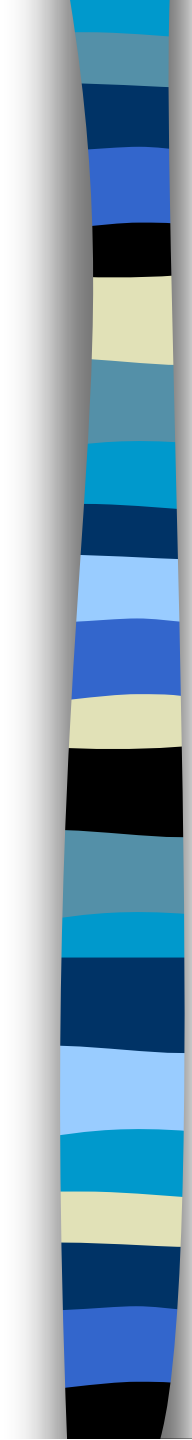
Outros métodos interessantes

- Existem outros métodos que permitem que se passe o processamento de uma solicitação.
 - Forward: é usado para encaminhar uma solicitação de um servlet para o outro.
 - ✓ Diferenças entre forward e sendRedirect:
 - O sendRedirect faz uma volta completa, passando pelo browser para chamar o outro servlet. Além disso, o objeto HttpServletRequest é perdido.
 - O forward redireciona para o outro servlet sem a ajuda do browser e os objetos HttpServletRequest e HttpServletResponse vão junto.
 - Para trabalharmos com este métodos temos que ter um objeto **RequestDispatcher**



Comunicação entre Servlets e JSP

- Para ilustrarmos a comunicação entre Servlet e JSP (ou mesmo entre Servlets), modificaremos a aplicação de Login feita totalmente por JSP:
 - Crie um servlet chamado *ServletLogin.java* no pacote Controladores
 - Coloque no *processRequest()* o código da página *processaLogin.jsp*
 - Modifique o Action do formulário da página *formLogin.jsp* para o servlet
 - Modifique o *sendRedirect* para o *RequestDispatcher*, da seguinte forma:



```
String u = request.getParameter("pUser");
String s = request.getParameter("pSenha");
if (u != null && s != null && u.equals("giuliano") &&
    s.equals("123456"))
{
    RequestDispatcher rd =
request.getRequestDispatcher("BemVindo.jsp");

rd.forward(request, response);
}
else
{
    out.println("<b>Login Incorreto!</b>")
}
```



Recuperando a informação

- Os dados são passados (“carregados”) pelo objeto request.
- Logo, pode-se recupera-los normalmente:

```
request.getParameter(“pUser”);
```

Bemvindo.jsp

A Plataforma Java

Desenvolvimento de Aplicações com Java



Java e Conexão com Banco de Dados





Conexão com Banco de Dados

- JDBC: Java Database Connectivity
- Acesso a Banco de Dados SQL
 - Independente da Plataforma e do Banco de Dados;
 - Disponível no pacote J2SDK
- Vários Drivers disponíveis
 - 100% Puro Java
 - Diversos drivers disponíveis

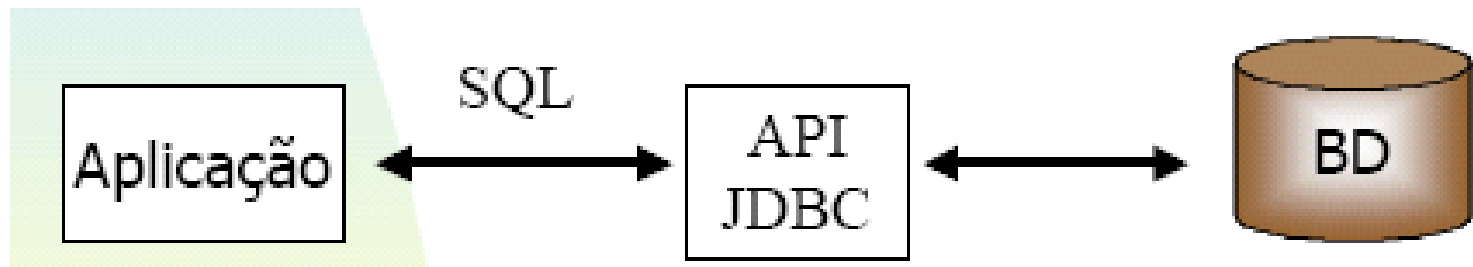


JDBC

- Bancos de dados em Java – JDBC
- Java DataBase Connectivity é a API que nos permite acessar bancos de dados através de nossa querida e amada linguagem Java.
- A idéia é simples, existem diferentes tipos de drivers para bancos de dados, mas para os programadores as chamadas são uniformes.

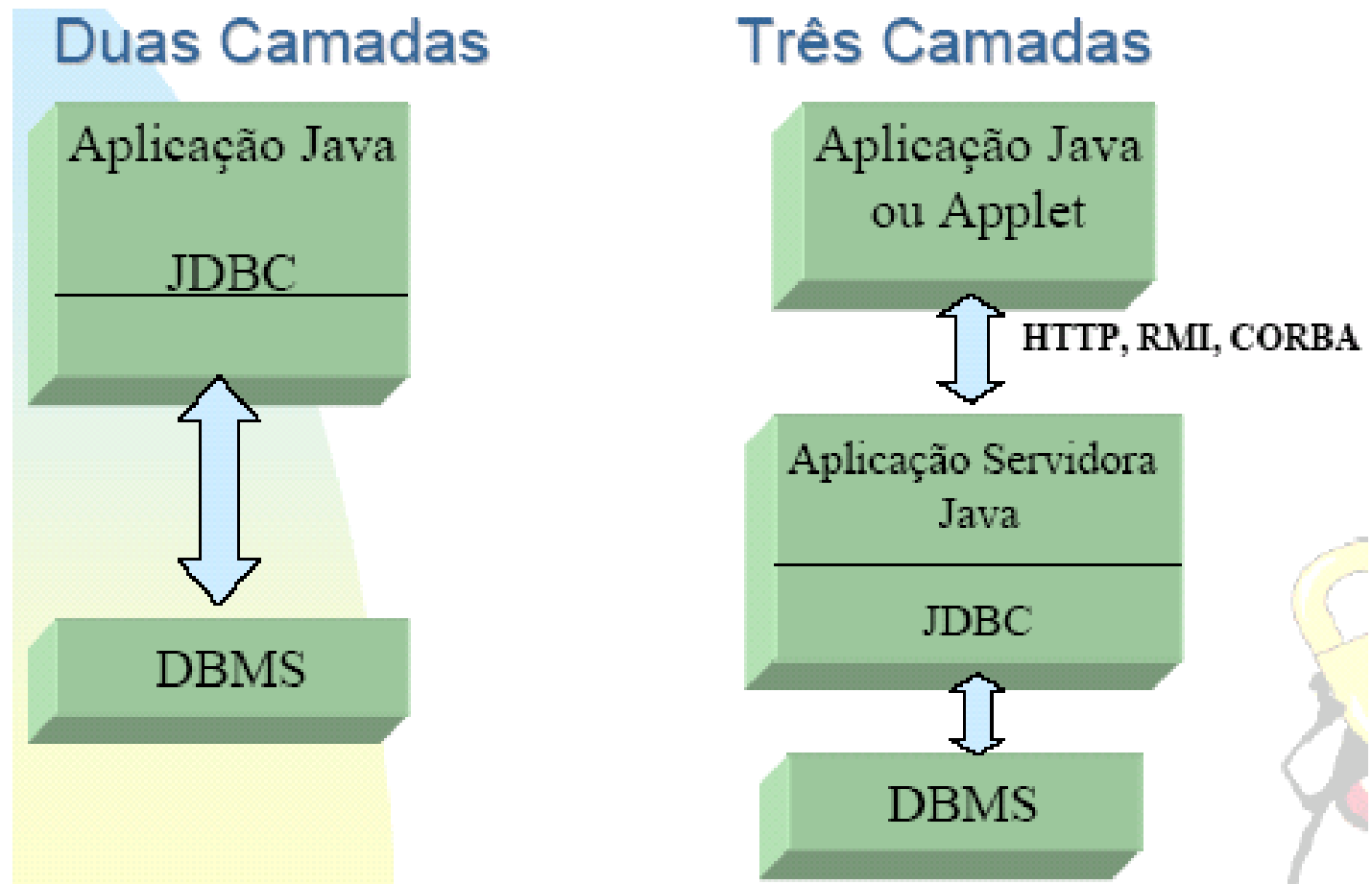
JDBC

- Estabelece conexão com o Banco de Dados
- Envia requisições SQL
- Processa os resultados



JDBC

- Modelo de desenvolvimento de aplicações com BD

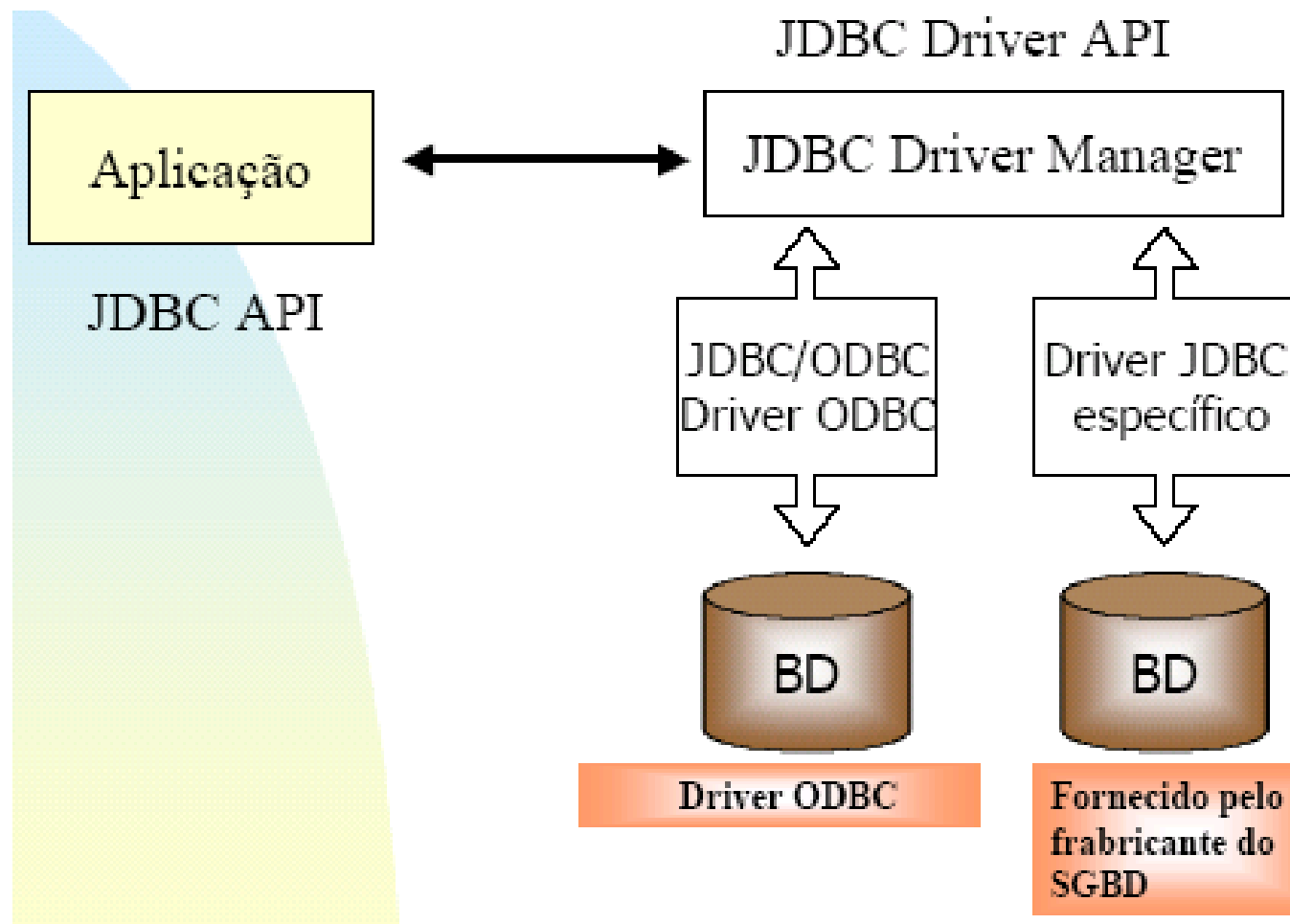




Recursos on-line sobre JDBC

- **Site da sun sobre JDBC**
- <http://java.sun.com/products/jdbc/>
- **Tutorial sobre JDBC**
- <http://java.sun.com/docs/books/tutorial/jdbc/>
- **JDBC Drivers disponíveis**
- <http://industry.java.sun.com/products/jdbc/drivers>

JDBC – Detalhes de Conexão



Tipos de Datos Suportados

JDBC Type	Java Type	JDBC Type	Java Type
BIT	boolean	NUMERIC	BigDecimal
TINYINT	byte	DECIMAL	
SMALLINT	short	DATE	java.sql.Date
INTEGER	int	TIME	java.sql.Timestamp
BIGINT	long	TIMESTAMP	
REAL	float	CLOB	Clob*
FLOAT	double	BLOB	Blob*
DOUBLE		ARRAY	Array*
BINARY	byte[]	DISTINCT	
VARBINARY		STRUCT	Struct*
LONGVARBINARY		REF	Ref*
CHAR	String	JAVA_OBJECT	
VARCHAR			
LONGVARCHAR			





Uso de um Banco de Dados

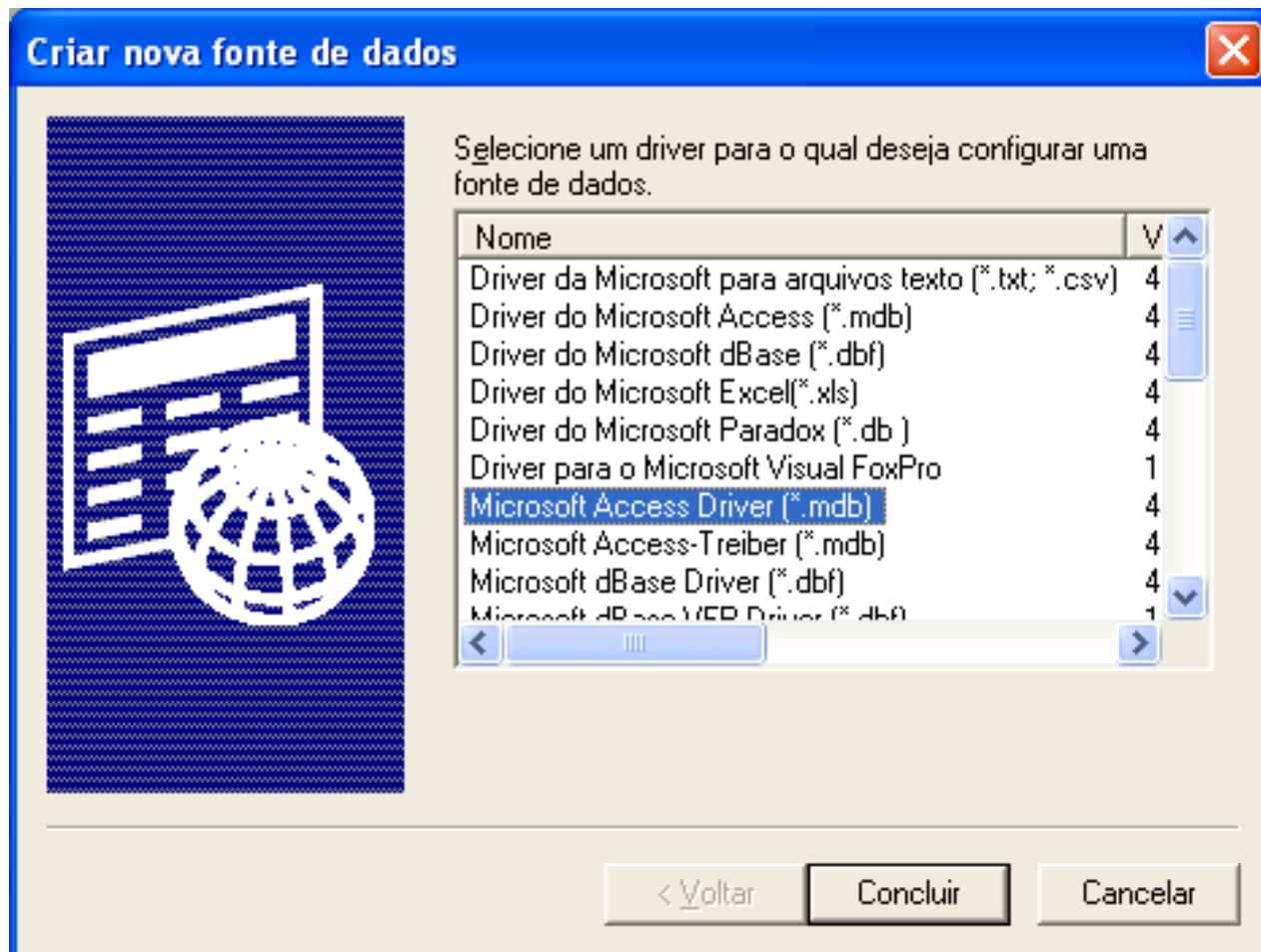
■ Duas opções

- Acessar o Banco via ponte JDBC-ODBC
 - ✓ Necessidade de configurar um DSN do usuário no ODBC.

■ Via driver JDBC direto para o Banco

- Necessidade de se obter o driver adequado.

Configuração ODBC



Configuração ODBC

Configurar ODBC para Microsoft Access

Nome da fonte de dados: LojaABG

Descrição:

Banco de dados

Banco de dados:

Selecionar... Criar... Reparar... Compactar...

Banco de dados do sistema

☒ Nenhum

☐ Banco de dados:

Banco de dados do sistema...

OK

Cancelar

Ajuda

Avançado...

Opções>>



Passos básicos para Conexão JDBC

- Importar `java.sql.*`
- Carregar driver JDBC
- Especificar BD
- Abrir conexão com BD
- Criar um objeto comando (statement)
- Submeter o comando SQL
- Receber o resultado
- Utilizar os resultados no programa

Passos básicos para Conexão JDBC

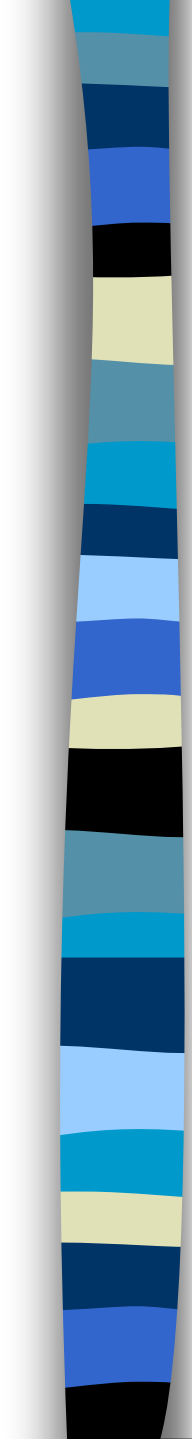
- 1. Importar java.sql.*
`import java.sql.*;`

- 2. Carregar driver JDBC

```
try {  
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
}  
catch { ClassNotFoundException cnfe) {  
    System.out.println("Erro no carregamento do driver: "+ cnfe);  
}
```

Obs: toda tentativa de conexão com driver de BD tem que tratar possíveis exceções.

- a JVM deve poder encontrar estas classes: para aplicações, a variável de ambiente CLASSPATH deve incluir os drivers

- 
- A abertura da conexão é feita pelo método `getConnection`, que recebe uma URL (Universal Resource Location) como argumento
 - JDBC URLs são formadas por: um protocolo (normalmente `jdbc`), um subprotocolo (normalmente é o tipo de driver) e informações para o SGBD.

```
Connection conn = DriverManager.getConnection  
("jdbc:odbc:Empresa", "login", "senha");
```

- 3. Especificar BD

```
String host = "jdbc:odbc:Empresa";
```

- 4. Abrir conexão com BD

```
String nomeUsuario = "giuliano";
```

```
String senha = "secreta";
```

```
Connection connection =
```

```
DriverManager.getConnection(host, nomeUsuario, senha);
```

Passos básicos para Conexão JDBC

- Opcionalmente, podemos obter algumas informações sobre o Banco.

```
DatabaseMetaData dbMetaData = connection.getMetaData();
```

```
String nomeBanco = dbMetaData.getDatabaseProductName();
```

```
System.out.println("Database: " + nomeBanco);
```

```
String versaoBanco = dbMetaData.getDatabaseProductVersion();
```

```
System.out.println("Version: " + versaoBanco);
```

Conectando via ODBC...

```
import java.sql.*;
class BDConexao {
    public static void main (String args[]) {
        try {

            String url = "jdbc:odbc:Empresa";
            String usuario = "";
            String senha = "";

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            Connection con;
            con = DriverManager.getConnection(url,usuario,senha);

            System.out.println("Conexão realizada com sucesso.");

            con.close();
        }

        catch(Exception e)
        {
            System.out.println("Problemas na conexão.");
        }
    }
}
```


Conectando via Driver Específico...

```
import java.sql.*;

class BDConexao {

    public static void main (String args[]) {

        try {

            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost/livraria", "root", "admin");

        }
        catch(Exception e)
        {
            System.out.println("Problemas na conexão.");
        }
    }
}
```



Statements

- Uma vez conectado queremos executar statements
- Assim podemos utilizar
 - **executeUpdate(*String sql*)** – para executar create table, insert, update, delete.
 - **executeQuery(*String sql*)** – para executar consultas (retorna ResultSet).

- 
- 5. Criar um objeto comando (statement)
Statement **st** = **connection.createStatement();**

- 6. Submeter o comando SQL

```
String atual = "UPDATE algumaTabela SET col = valor";  
st.executeUpdate(atual);
```

- **executeUpdate** recebe por parâmetro um string contendo comandos UPDATE, INSERT, ou DELETE

```
String consulta = "SELECT col1, col2, col3 FROM  
algumaTabela";
```

```
ResultSet resultado = st.executeQuery(consulta);
```

- Usar **setQueryTimeout** para especificar o tempo máximo de espera pelo resultado da consulta.

Criando uma tabela

```
stmt = cnt.createStatement();  
try {  
    stmt.executeUpdate("CREATE TABLE LIVRO " +  
        "(TITULO VARCHAR(50), ISBN VARCHAR(30), "+  
        "AUTOR VARCHAR(50), PAGINAS INTEGER)");  
}catch (SQLException e) {  
    System.out.println(e);  
}
```

Inserindo Dados

```
import java.sql.*;
class BDConsulta{
    public static void main (String args[]) {
        try {
            String url = "jdbc:odbc:Empresa";

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(url, " ", " ");

            Statement st = con.createStatement();

            st.executeUpdate("INSERT INTO Pedidos (CodPedido, CodCli,
            Descricao, Valor, DataPedido) VALUES (1001,1230,'Dicionário
            Aurélio Século XXI.',36.50,#16-03-2002#)");

            System.out.println("Operação realizada com sucesso.");

            st.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Problemas na conexão.");
        }
    }
}
```

Recebendo dados via Query SQL

- A classe **ResultSet** oferece à aplicação a tabela resultante de um Select e:
 - mantém um cursor posicionado em uma linha da tabela. Inicialmente este cursor está antes da primeira linha e a mensagem `next()` movimenta o cursos para frente.
 - Permite à aplicação pegar os dados das colunas da linha corrente através dos métodos `getXXX(<coluna>)`.
 - ✓ XXX é o tipo da coluna
 - ✓ <coluna> é o nome da coluna ou sua posição (a partir de 1)
 - ✓ A primeira coluna tem índice 1, não 0

Usando ResultSet

```
Statement st = con.createStatement();
```

```
ResultSet rs = st.executeQuery("select a, b, c  
    from table1");
```

Varrendo o conjunto de dados do ResultSet

```
while ( rs.next() )  
{  
    int x = rs.getInt("a");  
    String s = rs.getString(2);  
    float f = rs.getFloat("c");  
}
```

Consultando dados

```
// conexão estabelecida

Statement st = con.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM Pedidos");

while (rs.next()){
    System.out.print(rs.getString("CodPedido") + " ");
    System.out.print(rs.getString("CodCli") + " ");
    System.out.print(rs.getString("Descricao") + " ");
    System.out.print(rs.getString("Valor") + " ");
    System.out.println(rs.getString("DataPedido"));
}

System.out.println("Operação realizada com sucesso.");

st.close();
con.close();
}

catch(Exception e)
{ System.out.println("Problemas na conexão. "); }

}
```


- Permite obter informações sobre o tipo de tabela que resultou o select.
 - Quais os nomes e os tipos das colunas
 - Se a coluna do tipo é numérico com sinal

```
ResultSet result = stmt.executeQuery(  
"SELECT * FROM TabEx ORDER BY id DESC");
```

```
ResultSetMetaData meta = result.getMetaData();  
int columns = meta.getColumnCount();  
for (int i=1;i<=columns;i++) {  
System.out.println (meta.getColumnLabel(i) + "\t"  
+ meta.getColumnTypeName(i)); }
```

Outros métodos da interface ResultSet

isFirst(): retorna verdadeiro se o curso está posicionado sobre o primeiro registro da tabela resultante

isLast(): retorna verdadeiro se o curso está posicionado sobre o ultimo registro da tabela resultante

next(): posiciona o cursor da tabela sobre o próximo registro.



Uso de driver JDBC direto

- Colocar o arquivo com os drivers do Banco de Dados no diretório de Bibliotecas do NetBeans, ou incorporá-lo no arquivo .jar de sua aplicação.
 - ✓ Geralmente é um arquivo .jar.



PreparedStatement

- Também podemos usar PreparedStatement para ter um resultado mais rápido...



Prepared Statements

- Cada vez que se executa um comando SQL passado por meio de uma String, este String deve ser analisado pelo processador de SQL do SGBD que irá, no caso da String estar sintaticamente correta, gerar um código binário que será executado para atender à solicitação.
- Todo esse processo é caro e sua execução repetidas vezes terá um impacto significativo sobre o desempenho da aplicação e do SGBD como um todo. *Prepared Statement* é indicado nos casos onde um comando será executado várias vezes em uma aplicação.
- Neste caso é melhor compilar o comando uma única vez e toda vez que for necessário executá-lo basta enviar o comando compilado.
- Além disso, o comando pré-compilado pode ser parametrizado, tornando-o mais genérico e, portanto, apto a expressar um maior número de consultas.

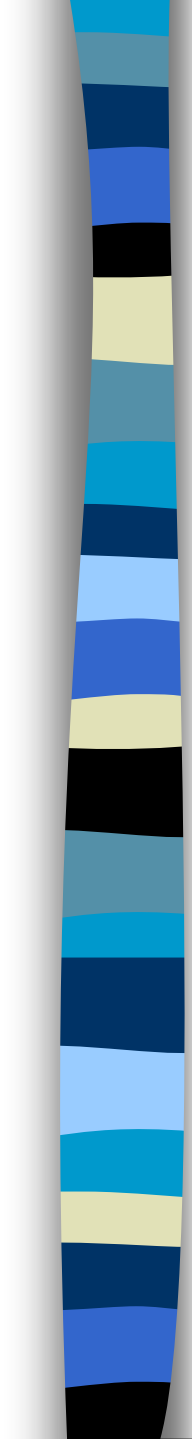


Prepared Statements

- Para criar um Prepared Statement é necessário:

PreparedStatement pst =

```
con.prepareStatement (“INSERT  
INTO usuarios(login,senha)  
VALUES(?, ? )”);
```

- 
- O comando anterior insere uma nova linha na tabela **usuarios** com os valores das colunas **login** e **senha** passados por parâmetro. O caractere '?' representa o parâmetro. Este tipo de comando só possui valor tendo parâmetros, caso contrário teria pouca chance de ser reutilizado.
 - Para executar o comando devemos especificar o valor dos parâmetros e executar o comando, como mostrado no exemplo abaixo:

```
pst.clearParameters();  
pst.setInt(1,8);  
pst.setString(2,"Clara Maria");  
pst.executeUpdate();
```



**Setando os
Parâmetros**

Exemplo

PreparedStatement

```
Connection con = DriverManager (...)  
....  
PreparedStatement prep = con.prepareStatement(  
    "SELECT * FROM LIVRO WHERE ISBN = ?");  
  
prep.setString(1,"123456789");  
  
ResultSet rs = prep.executeQuery();  
while (rs.next()) {  
    String s = rs.getString("TITULO");  
    String i = rs.getString("ISBN");  
    String a = rs.getString("AUTOR");  
    int pa = rs.getInt("PAGINAS");  
    System.out.println(s + " " + i + " " + a + " " + pa);  
}
```



Informações adicionais sobre os dados retornados

```
DatabaseMetaData dbMetaData =  
    connection.getMetaData();
```

```
String nomeBanco =  
    dbMetaData.getDatabaseProductName();
```

```
String versaoBanco  
    =dbMetaData.getDatabaseProductVersion();
```

```
Statement statement = connection.createStatement();
```

```
String query ="SELECT * FROM " + nomeTabela;
```

```
ResultSet resultSet = statement.executeQuery(query);
```


Informações adicionais sobre os dados retornados

```
ResultSetMetaData resultsMetaData = resultSet.getMetaData();
```

```
int qtdeColunas = resultsMetaData.getColumnCount();
```

```
for(int i=1; i<qtdeColunas+1; i++) {  
    out.println(resultsMetaData.getColumnName(i));  
}
```

```
while(resultSet.next()) {  
    for(int i=1; i<qtdeColunas+1; i++) {  
        out.println(resultSet.getString(i));  
    }  
}
```

A Plataforma Java

Programação Web com Java



Uso de *JavaBeans*



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





Uso de JavaBeans

- A medida que o código Java dentro do HTML torna-se cada vez mais complexo o desenvolvedor deve-se sempre preocupar-se em **não** misturar “conteúdo” com “forma”.
- **Para solucionar esse problema pode-se usar Javabeans para manipular a parte dinâmica em Java.**



Uso de JavaBeans

- Um Javabeen nada mais é que uma classe Java que obedece a uma certa padronização de chamada.
- Os atributos de um bean são acessados por meio de métodos que obedecem a convenção getXxxx e setXxxx. Exemplo: getItem()
- Os atributos booleanos seguem a seguinte convenção: isXxx() para acessar seu valor e getXxx() para setar seu valor.
- Todos os atributos de um Bean devem ser privados.
- Se houver construtor, pelo menos um deles deve vir sem argumento.

Uso de get e set

- Assim, você deve modificar qualquer atributo público de uma classe Java.
- **Exemplo:**

public double velocidade;

Devemos substituir por:

private double velocidade;

```
public double getVelocidade() {  
    return(velocidade);  
}
```

```
public void setVelocidade(double novaVelocidade) {  
    velocidade = novaVelocidade;  
}
```

Uso de get e set

- **Porque???**
- **Você colocar algumas restrições em seus valores**

```
public void setVelocidade(double novaVelocidade) {  
    if (novaVelocidade < 0) {  
        enviaMsgErro(...);  
        novaVelocidade = Math.abs(novaVelocidade);  
    }  
    velocidade = novaVelocidade;  
}
```
- Se os usuários acessarem diretamente os atributos, eles serão os responsáveis por verificar estas restrições.
- **Você pode mudar sua representação interna sem mudar seus valores.**
- **Você pode realizar outras operações (atualização de valores relacionados, por exemplo.**

Uso de JavaBeans em páginas JSP

- A sintaxe para o uso de um bean é:

`<jsp:useBean id="nome" class="package.class" />`

- Exemplo de uso:

`<jsp:useBean id= "livro" class= "desweb.Livro" />`

Que seria equivalente ao Scriptlet

`<% desweb.Livro livro = new desweb.Livro(); %>`

- Obs: É por isso que precisamos de um construtor sem parâmetro....



Uso de JavaBeans em páginas JSP

- Você também pode modificar o atributo **scope** para estabelecer o escopo do bean além da página corrente:

```
<jsp:useBean id="name" scope="session"  
class="package.class" />
```

- Vantagens adicionais no uso de Beans:
 - Facilita o compartilhamento de objetos entre páginas e servlets.



Exemplo do uso de JavaBean

CalculatorBean.java

```
package beans;
```

```
public class CalculatorBean  
{  
    public int dobra(int numero)  
    {  
        return 2* numero;  
    }  
}
```



Instalação do Bean

- Instalado no diretório
Pacotes Código Fonte do seu projeto NetBeans
- Observe que criamos diretórios para refletir o pacote onde está o Bean
 - Caso o coloque em um package (o que devo obrigatoriamente fazer) esta estrutura de diretório do package deve estar abaixo do classes.



Exemplo de uso de um JavaBean com JSP

Página *Dobra.jsp*

```
<jsp:useBean id="CalcBean" class="beans.CalculatorBean"/>
<HTML>
<HEAD>
</HEAD>
<BODY>
<%
    int i = 4;
    int j = CalcBean.dobra(i);
    out.print(" 2 x 4 = " + j);
%>
</BODY>
</HTML>
```



Acessando as propriedades de um Bean

■ Formato

```
<jsp:getProperty name="nome" property="propriedade" />
```

■ Objetivo

- Permitir o acesso as propriedades de um Bean (i.e., chamadas aos métodos *getXxx*) sem programação java explícita

■ Exemplo:

```
<jsp:getProperty name="livro" property="titulo" />
```

- Seria o equivalente a seguinte expressão jsp:

```
<%= livro.getTitulo() %>
```

Modificando as propriedades de um Bean

■ Formato

```
<jsp:setProperty name="nome" property="propriedade"  
  value="valor" />
```

■ Objetivo

- Permitir a modificação das propriedades de um Bean (i.e., chamando os método *setXxx*) sem programação Java explícita.

■ Exemplo

```
<jsp:setProperty name="livro" property="titulo" value="Core  
  Servlets and JavaServer Pages" />
```

- É o equivalente ao seguinte scriptlet

```
<% livro.setTitulo("Core Servlets and JavaServer Pages"); %>
```

Exemplo: página JSP

```
<HTML> <HEAD>
```

```
<TITLE>Uso de beans</TITLE>
```

```
</HEAD> <BODY> <CENTER>
```

```
<P>
```

```
<jsp:useBean id="teste" class= " beans.BeanSimples" />
```

```
<jsp:setProperty name="teste" property="mensagem" value= " Ola  
    mundo!" />
```

```
<H1>Mensagem: <I>
```

```
<jsp:getProperty name="test" property= "mensagem" /> </I></H1>
```

```
</BODY> </HTML>
```



Exemplo: **JavaBean** beans/BeanSimples.java

```
package beans;
```

```
public class BeanSimples{  
    private String mensagem = "Nenhuma  
    mensagem especificada";  
  
    public String getMensagem() {  
        return(mensagem);  
    }  
    public void setMensagem(String mensagem) {  
        this.mensagem = mensagem;  
    }  
}
```



Modificando as propriedades de um Bean

- Alternativamente, podemos modificar alguma propriedade de um Bean utilizando diretamente algum parâmetro passado pelo cliente

- Exemplo

```
<jsp:setProperty name="livro" property="titulo"  
  value'<%= request.getParameter("tituloBusca") %>' />
```




Modificando as propriedades de um Bean

- Se o nome do parâmetro for o mesmo da propriedade, podemos usar (já se faz a conversão de tipos automaticamente):

```
<jsp:setProperty name="livro" property="titulo"param="titulo"/>
```

- Se todas as propriedades forem setadas através dos parâmetros vindos em um request, podemos simplificar da seguinte forma:

```
<jsp:setProperty name="entry" property="*" />
```



Compartilhamento de Beans

- Podemos compartilhar Beans através do atributo **scope** da tag `jspBean`.
- Existem quatro valores possíveis: `page`, `request`, `session` e `application`.
 - O default é `page`.



Compartilhamento de Beans

■ scope page

- Objetos declarados com nesse escopo são válidos até a resposta ser enviada ou a requisição ser encaminhada para programa no mesmo ambiente, ou seja, só podem ser referenciados nas páginas onde forem declarados. Objetos declarados com escopo page são armazenados no objeto pagecontext

■ O Bean é instanciado como uma variável local da página.

- Podemos acessá-lo chamando getAttribute da variável pageContext.

■ Geralmente acessados usando:

- jsp:getProperty,
- jsp:setProperty,
- scriptlets, ou expressions na mesma página.



Compartilhamento de Beans

■ scope request

- Objetos declarados com nesse escopo são válidos durante a requisição e são acessíveis mesmo quando a requisição é encaminhada para programa no mesmo ambiente. Objetos declarados com escopo request são armazenados no objeto request.
- Com isso a acessibilidade ao Bean é incluída a páginas referenciadas por `jsp:forward` e `jsp:include`. A página encaminhada não precisa ter um elemento de ação `jsp:useBean`. Podemos usar `getProperty` e `setProperty` na página encaminhada sem problemas.



Compartilhamento de Beans

■ scope session

- Objetos declarados com nesse escopo são válidos durante a sessão desde que a página seja definida para funcionar em uma sessão. Objetos declarados com escopo session são armazenados no objeto session (objeto implícito da página jsp).
- Assim, podemos recuperar este Bean chamando o método `getAttribute` do objeto session.
- Se a página jsp estiver definida para não usar sessão, uma exceção ocorrerá.



Compartilhamento de Beans

■ scope application

- Objetos declarados com nesse escopo são acessíveis por páginas no mesmo servidor de aplicação. Objetos declarados com escopo application são armazenados no objeto application.
- Assim, o Bean será armazenado em um objeto ServletContext disponível através do objeto implícito **application** ou chamando através do método **getServletContext**.
- Relembrando, um ServletContext é compartilhado por todos os servlets em uma mesma aplicação WEB.

■ Os valores em um ServletContext podem ser acessados através do método getAttribute.

- Isso traz uma série de vantagens:
 - ✓ Mecanismo que permite que vários servlets e páginas JSP acessem o mesmo objeto.
 - ✓ Permite que um servlet crie um Bean que será usado em um JSP, permitindo o uso mais adequado do modelo MVC.

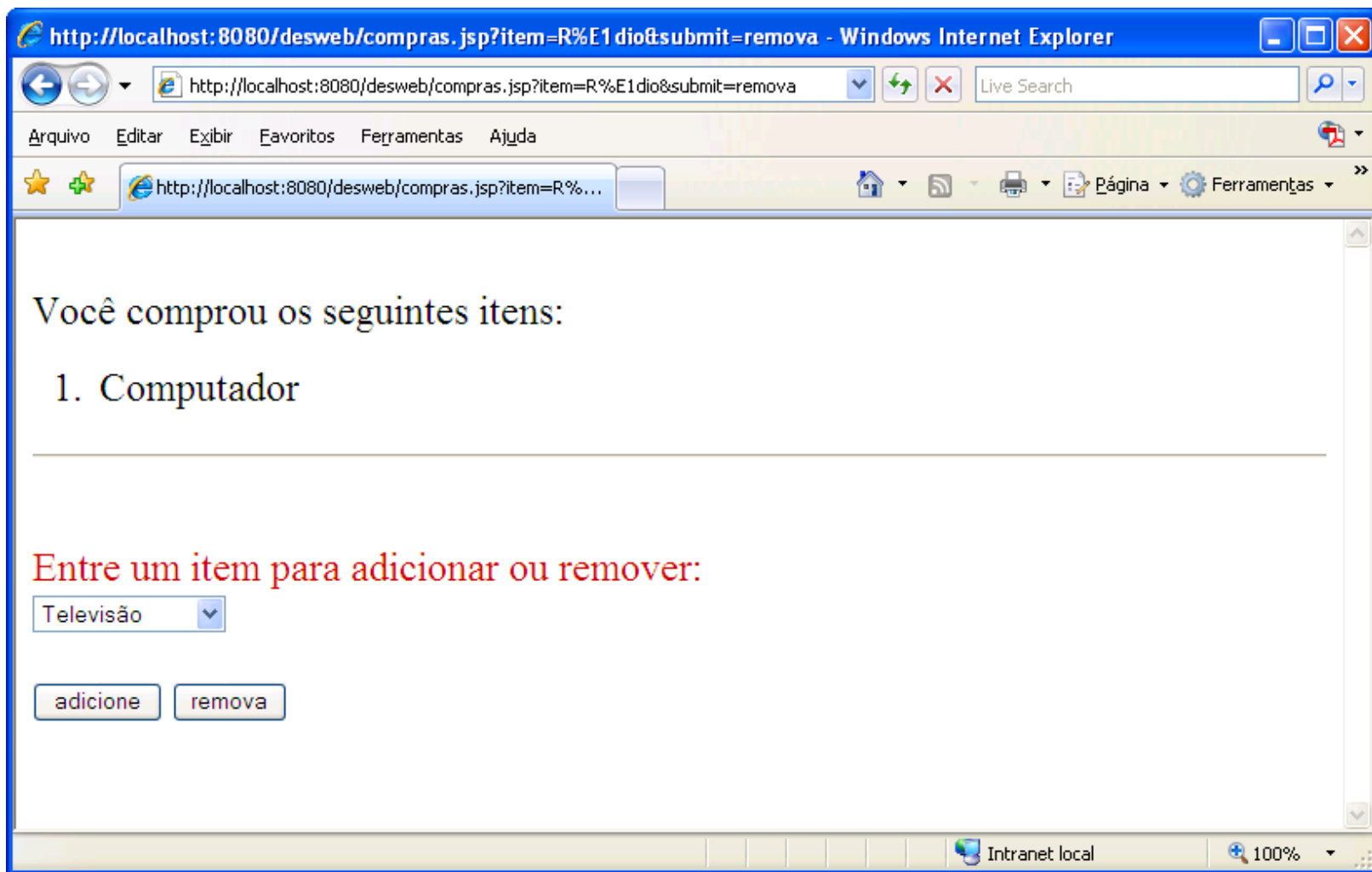


Compartilhamento de Beans

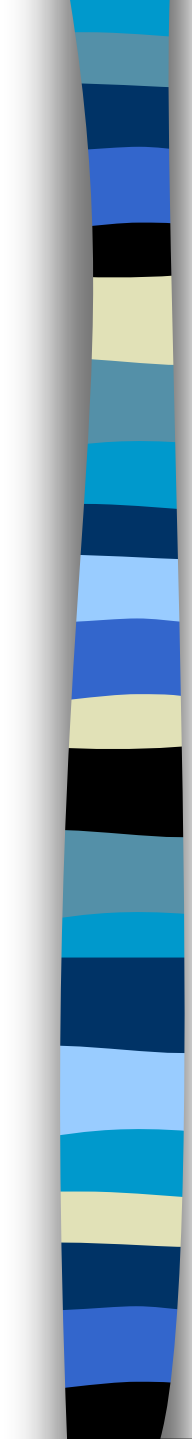
■ Importante

- Se no `setProperty` usarmos o valor “*” significa que toda modificação em elementos do formulário será automaticamente passada para a propriedade do bean de mesmo nome.
- Os valores são automaticamente convertidos para o tipo correto no bean.

Uso de Beans: carrinho de compras



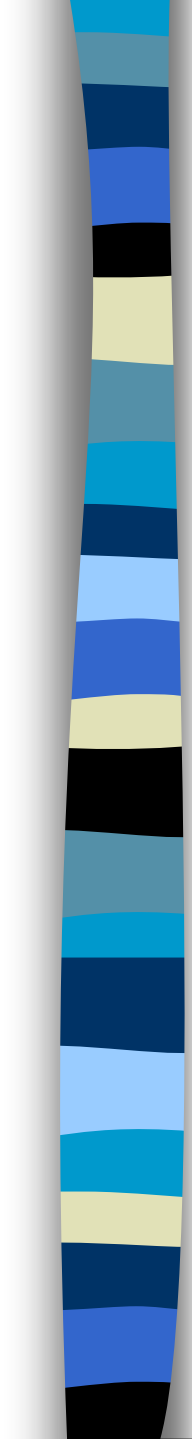

```
<html>
<jsp:useBean id= "compra" scope="session"
class= "Compras" />
<jsp:setProperty name= "compra" property="*" />
<% compra.ProcessaRequisicao(request);%>
<FONT size = 5>
<br> Voc&ecirc; comprou os seguintes itens:
<ol>
<%
    String[] items = compra.getItems();
    for (int i=0; i<items.length; i++) {
        %>
        <li> <%= items[i] %>
        <%
            }
        %>
    }
</ol>
</FONT>
<hr>
```



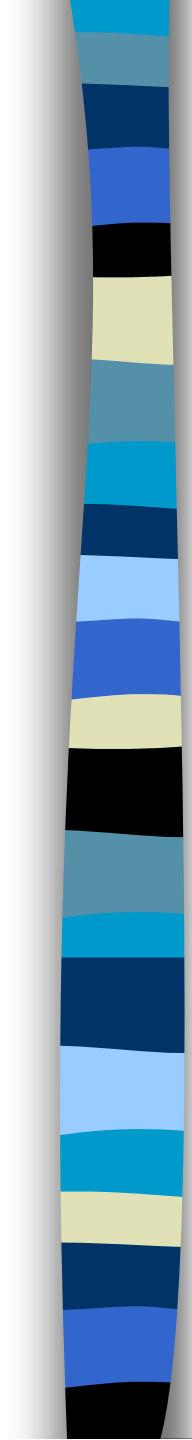
```
<body bgcolor="white">
<font size = 5 color="#CC0000">
<form type=POST action=compras.jsp>
<BR>Entre um item para adicionar ou
  remover:<br>
<SELECT NAME="item">
<OPTION>Televis&atilde;o
<OPTION>R&aacute;dio
<OPTION>Computador
<OPTION>V&iacute;deo Cassete
</SELECT>
<br> <br>
<INPUT TYPE=submit name="submit"
  value="adicione">
<INPUT TYPE=submit name="submit"
  value="remova">
</form></FONT></body></html>
```

O javabean Compras

```
import javax.servlet.http.*;
import java.util.Vector;
import java.util.Enumeration;
public class Compras {
    Vector v = new Vector();
    String submit = null;
    String item = null;
    private void addItem(String name) {
        v.addElement(name);
    }
    private void removeItem(String name) {
        v.removeElement(name);
    }
}
```



```
public void setItem(String name) {
    item = name;
}
public void setSubmit(String s) {
    submit = s;
}
public String[] getItems() {
    String[] s = new String[v.size()];
    v.copyInto(s);
    return s;
}
private void reset() {
    submit = null;
    item = null;
}
```



```
public void
processaRequisicao(HttpServletRequest
request) {
    if (submit == null)
        return;
    if (submit.equals("adicione"))
        addItem(item);
    else if (submit.equals("remova"))
        removeItem(item);
    reset();
}
}
```

A Plataforma Java

Programação Web com Java



Padrões de Projeto e Persistência com DAO



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





Introdução a Padrões de Projeto

- Padrões de Projeto foram inicialmente percebidos na área de construção quando foram catalogados permitindo agilizar o processo de construção de prédios e casas.
- Posteriormente padrões de desenvolvimento de software começaram a ser identificados e catalogados. Hoje contamos com diversos livros para várias situações.



Importância de Padrões

- A importância desse tipo de técnica é o ganho que temos em desempenho no desenvolvimento de sistemas.
- Soluções que já foram testadas e comprovadas, isso aumenta a qualidade do software final e permite que analistas troquem informações com um vocabulário similar.
- Quando alguém diz que usou um certo padrão isso já transmite um conjunto de informações.



Principais Padrões

- Contamos com diversos padrões:
 - DAO – Data Access Object
 - VO – Value Object é um objeto do domínio da aplicação
 - MVC – Model View Controller – permite construir a aplicação de forma modular sem amarrar a camada de apresentação com a parte de negócio



Padrão de Projeto – DAO

- DAO – Data Access Object é uma classe especial que contém todas as informações necessárias para acessar uma ou mais tabelas e criar os objetos relacionados às linhas das tabelas.

Exemplo - DAO

■ Pacote "modelo"

- Criação da classe Cliente.java
 - ✓ Controla a criação e controle de objetos Cliente.

■ Pacote "persistencia"

- Criação da classe Conexao.java
 - ✓ Controla a abertura e fechamento da conexão com o banco de dados;
- Criação da classe ClienteDAO.java
 - ✓ Controla a manipulação dos dados de clientes no banco
 - ✓ Vários métodos para:
 - Consultar todos clientes;
 - Consultar um cliente específico;
 - Inserir, excluir e atualizar um cliente específico

Criando classe de Conexão

```
public class Conexao {  
    public static Connection getConexao( ){  
        Connection con = null;  
        try{  
            Class.forName("com.mysql.jdbc.Driver");  
            con = DriverManager.getConnection("jdbc:mysql://localhost/livraria",  
"root", "");  
        }  
        catch (Exception e)  
        {  
            System.out.println("falha na conexão");  
        }  
        return con;  
    }  
  
    public static void fecharConexao(Connection conn, Statement stmt,  
ResultSet rs){  
        try {  
            if (rs != null) rs.close( );  
            if (stmt != null) stmt.close( );  
            if (conn != null) conn.close( );  
        } catch (Exception e) { }  
    }  
}
```

Criando classe de Conexão

```
public static void fecharConexao(Connection conn) {  
    try {  
        if (conn != null) conn.close( );  
    } catch (Exception e) { }  
}  
  
public static void fecharConexao(Connection conn,  
PreparedStatement ps)  
{  
    try {  
        if (ps != null) ps.close( );  
        if (conn != null) conn.close( );  
    } catch (Exception e) { }  
}  
}
```

Criando classe de Persistência

```
public class ClienteDAO {  
  
    private Connection conn;  
  
    public ClienteDAO( ) throws Exception{  
        try {  
  
            this.conn = Conexao.getConexao( );  
  
        } catch( Exception e ) {  
            throw new Exception( "Erro: " +  
                ":\n" + e.getMessage( ) );  
        }  
    }  
}
```

Criando classe de Persistência

- Método Salvar() -

```
public void salvar(Cliente cliente) throws Exception{
    PreparedStatement ps = null;
    Connection conn = null;
    if (cliente == null)
        throw new Exception("O valor passado não pode ser nulo");
    try {
        String SQL = "INSERT INTO clientes (nome, endereco, telefone,
estado) values (?, ?, ?, ?)";
        conn = Conexao.getConexao();
        ps = conn.prepareStatement(SQL);
        ps.setString(1, cliente.getNome());
        ps.setString(2, cliente.getEndereco());
        ps.setString(3, cliente.getTelefone());
        ps.setInt(4, cliente.getEstado());

        ps.executeUpdate();
        JOptionPane.showMessageDialog(null,"Cliente
cadastrado!", "Concluído", JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException sqle) {
        throw new Exception("Erro ao inserir dados "+ sqle);
    } finally {
        Conexao.fecharConexao(conn,ps);
    }
}
```


Criando classe de Persistência

- Método atualizar() -

```
public void atualizar(Cliente cliente) throws Exception {
    PreparedStatement ps = null;
    Connection conn = null;

    if (cliente == null)
        throw new Exception("O valor passado não pode ser nulo");
    try {
        String SQL = "UPDATE autores SET nome=?, " +
            " endereco=?, telefone=? estado=? " +
            "where autor_id=?";
        conn = this.conn;
        ps = conn.prepareStatement(SQL);
        ps.setString(1, cliente.getNome());
        ps.setString(2, cliente.getEndereco());
        ps.setString(3, cliente.getTelefone());
        ps.setInt(4, cliente.getEstado());

        ps.executeUpdate();
    } catch (SQLException sqle) {
        throw new Exception("Erro ao atualizar dados: "+ sqle);
    } finally {
        Conexao.fecharConexao(conn, ps);
    }
}
```


Criando classe de Persistência

- Método todosAutores() -

```
public List todosClientes( ) throws Exception{
    PreparedStatement ps = null;
    Connection conn = null;
    ResultSet rs = null;
    try {
        conn = this.conn;
        ps = conn.prepareStatement("select * from clientes");
        rs = ps.executeQuery( );
        List<Cliente> list = new ArrayList<Cliente>( );
        while( rs.next( ) ) {
            String nome = rs.getString( 2 );
            String endereco = rs.getString( 3 );
            String telefone = rs.getString( 4 );
            Integer estado = rs.getInt( 5 );

            list.add( new Cliente(nome,endereco,telefone,estado) );
        }
        return list;
    } catch (SQLException sqle) {
        throw new Exception(sqle);
    } finally {
        Conexao.fechar(conn, ps, rs);} }
```

Criando classe de Persistência

- Método procurarCliente() -

```
public Cliente procurarCliente(Integer id) throws Exception {
    PreparedStatement ps = null;
    Connection conn = null;
    ResultSet rs = null;
    try {
        conn = this.conn;
        ps = conn.prepareStatement("select * from clientes where
cliente_id=?");
        ps.setInt(1, id);
        rs = ps.executeQuery( );
        if( !rs.next( ) ) {
            throw new Exception( "Não foi encontrado nenhum registro
com o ID: " + id );
        }
        String nome = rs.getString( 2 );
        String endereco = rs.getString( 3 );
        String telefone = rs.getString( 4 );
        Integer estado = rs.getInt( 5 );

        return new Cliente(nome, endereco, telefone, estado) ;
    } catch (SQLException sqle) {
        throw new Exception(sqle);
    } finally { Conexao.fechar(conn, ps, rs);}
```

Criando classe de Persistência

- Método excluir() -

```
public void excluir(Cliente cliente) throws Exception {
    PreparedStatement ps = null;
    Connection conn = null;

    if (cliente == null)
        throw new Exception("O valor passado não pode ser nulo");

    try {
        conn = this.conn;
        ps = conn.prepareStatement("delete from clientes where
cliente_id=?");
        ps.setInt(1, cliente.getIdCliente( ));
        ps.executeUpdate( );

    } catch (SQLException sqle) {
        throw new
            Exception("Erro ao excluir dados:" + sqle);
    } finally {
        Conexao.fecharConexao(conn, ps);
    }
}
```

Evento de Cadastro

.....

```
String n = request.getParameter("nome")
```

```
String e = request.getParameter("endereco");
```

```
String t = request.getParameter("telefone");
```

```
int es = Integer.parseInt(request.getParameter("estado"));
```

```
ClienteDAO cli = new ClienteDAO();
```

```
cli.salvar(new Cliente(n,e,t,es));
```

.....

Evento de Consulta

```
.....  
int id = request.getParameter("identificacao");  
  
    ClienteDAO cd = new ClienteDAO();  
    Cliente cli = cd.procurarCliente(id);  
  
    out.println(cli.getNome());  
    out.println(cli.getEndereco());  
    out.println(cli.getTelefone());  
    out.println(cli.getEstado());  
    .....
```

A Plataforma Java

Programação Web com Java



Uso de JSTL

- Taglibs -



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





O JSP sem padrão

- É muito comum!
- Scriptlets espalhados em várias páginas
- Código confuso = manutenção +trabalhosa
- Regras de negócio não deveriam ser tratadas dentro do JSP (foge ao padrão MVC!)
- JSP deve ser usado somente para a camada de visualização (viewer), somente exibindo o conteúdo dos objetos e nada de scriptlets



JSTL: o JSP com padrão

■ HISTÓRICO

- **Julho/2001** - Criado o projeto (JSR-052)
- **Junho/2002** – lançado o JSTL 1.0 baseado no JSP 1.2 (Tomcat4 e maioria dos servidores corporativos)
- **Janeiro/2004** - lançado o JSTL 1.1 baseado no JSP 2.0 (Tomcat 5)
- **Julho/2004** - lançado o JSTL 1.1.1



JSTL: o JSP com padrão

- Padronizar as aplicações JSP !
- Dar soluções fáceis de usar para tarefas mais comuns

JSTL: o JSP com padrão

Tipos de soluções JSTL 1.0

Tipo	URI	Prefixo	Exemplo
Core	http://java.sun.com/jstl/core	c	<c:nomeDaTag...>
XML	http://java.sun.com/jstl/xml	x	<x:nomeDaTag...>
Internationalization	http://java.sun.com/jstl/fmt	fmt	<fmt:nomeDaTag...>
Database	http://java.sun.com/jstl/sql	sql	<sql:nomeDaTag...>

JSTL: o JSP com padrão

Tipos de soluções JSTL 1.1

Tipo	URI	Prefixo	Exemplo
Core	http://java.sun.com/jsp/jstl/core	c	<c:nomeDaTag...>
XML	http://java.sun.com/jsp/jstl/xml	x	<x:nomeDaTag...>
l18n	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:nomeDaTag...>
Database	http://java.sun.com/jsp/jstl/sql	sql	<sql:nomeDaTag...>
Funções	http://java.sun.com/jsp/jstl/functions	fn	<fn:nomeDaTag...>



Como instalar o JSTL?

■ 1. Faça o download:

<http://www.apache.org/dist/jakarta/taglibs/standard/>

■ 2. Descompacte o arquivo e copie:

a) `/jakarta-taglibs-standard-1.*/tld/*` para **WEB-INF**

b) `/jakarta-taglibs-standard-1.*/lib/*` para **WEB-INF/lib**

Como instalar o JSTL?

3. Adicione essas informações no **web.xml**:

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
  <taglib-location>/WEB-INF/fmt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt-rt</taglib-uri>
  <taglib-location>/WEB-INF/fmt-rt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
  <taglib-location>/WEB-INF/c-rt.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
  <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql-rt</taglib-uri>
  <taglib-location>/WEB-INF/sql-rt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
  <taglib-location>/WEB-INF/x.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/x-rt</taglib-uri>
  <taglib-location>/WEB-INF/x-rt.tld</taglib-location>
</taglib>
```

Como instalar o JSTL?

- 4. Na sua página JSP declare os tipos que for utilizar:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
```

- 5. Depois é só sair usando!

```
<H1><c:out value="Conexão Java 2004" /></H1>
```

Sintaxe

- Não importa onde a Expressão Lógica é usada, ela sempre é invocada de uma maneira consistente:

- `${exp}` ou `#{exp}`

Obs.: Na plataforma J2EE não podem ser usadas em conjunto.

- Exemplos:

- `${true}`
- `${ "aqui jaz um texto" }`
- `${5*4} <!-- resulta no numero 20--%>`



Entendendo o JSTL

- O JSTL é uma coleção de quatro bibliotecas tags.
- Cada biblioteca de tags fornece ações úteis (ou tags) baseados nas seguintes áreas funcionais:
 - Core
 - Internacionalização (l18n) e formatação
 - Acesso a banco de dados relacional (tags SQL)
 - Processamento de XML (tags XML)



A Core Tag Library

- Contém um centro de ações de propósito geral, que fornecem soluções simples, mas efetivas, a problemas comuns que os desenvolvedores experimentam em quase toda aplicação JSP.
- Grupo de tags mais usadas freqüentemente:
 - **<c:if />** para condições
 - **<c:forEach />** para interação
 - **<c:choose /> ... <c:when />... <c:otherwise />** para um fluxo seletivo
 - **<c:set />** e **<c:remove />** para trabalhar com escopo de variáveis
 - **<c:out />** para fazer a saída de valores de variáveis e expressões
 - **<c:catch />** para trabalhar com exceções Java
 - **<c:url />** para criar e trabalhar com URL's



Trabalhando com JavaBeans

■ Exemplo:

```
<jsp:useBean id="jbean" class="meusBeans.BeanSimples" />
```

```
${jbean.mensagem}
```

Exibindo Objetos

- O formato padrão é:
`<c:out value="{objeto.atributo}" />`
- Imagine um objeto Pessoa, com os atributos de nome e idade:

Aluno : `<c:out value="{Pessoa.nome}" />`
com `<c:out value="{Pessoa.idade}" />` anos.

Exibindo Objetos

- Para setar valores de variáveis:

```
<c:set var="nomeDaVariavel" value="valor"/>
```

ou

```
<c:set var="nomeDaVariavel">valor</c:set>
```

- Para exibir o valor da variável acima:

```
<c:out value="${nomeDaVariavel}" />
```

Exibindo Objetos

- Para exibir o valor de parâmetros passados para um formulário :

```
<c:out value="${param.nomeDoParametro}" />
```

- Exemplo:

Bem-vindo <c:out value="\${param.usuario}" />!

Operadores Condicionais

- O operador condicional para apenas uma opção:

```
<c:if test="${expressão}">
```

...

```
</c:if>
```

- Exemplo:

```
<c:if test="${param.nome == 'nobody'}">
```

Acesso Negado

```
</c:if>
```

Operadores Condicionais

O operador condicional para duas ou mais opções:

```
<c:choose>
```

```
<c:when test="{expressão}">
```

```
...
```

```
</c:when>
```

```
<c:when test="{expressão}">
```

```
...
```

```
</c:when>
```

```
<c:otherwise>
```

```
...
```

```
</c:otherwise>
```

```
</c:choose>
```

Operadores Condicionais

- Exemplo:

```
<c:choose>
```

```
<c:when test="${hora<12}">
```

Bom dia!

```
</c:when>
```

```
<c:when test="${hora<18}">
```

Boa tarde!

```
</c:when>
```

```
<c:otherwise>
```

Boa noite!

```
</c:otherwise>
```

```
</c:choose>
```


Loops

- O loop **forEach** é usado para percorrer dados de uma Collection:

```
<c:forEach var="nome" items="${nomeDaCollection}">  
  <c:out value="${nome}"/>  
</c:forEach>
```

- Exemplo:

```
<c:forEach var="aluno" items="${listaAlunos}">  
  <c:out value="${aluno.nome}"/>  
</c:forEach>
```

Melhorando a Aplicação MVC

- Criar a página JSP **mostrarClientes.jsp**
- Criar nesta página uma tabela para mostrar os dados de **Clientes**.
 - Incluir as bibliotecas JSTL no projeto
 - Incluir as URI na página

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
```

- Utilizar um Loop na lista de clientes

```
<c:forEach var="lista" items="${ requestScope.clientesList }">
```

Melhorando a Aplicação MVC

- Definir os campos para exibir os dados:

```
<tr>
    <td>${lista.idCliente}</td>
    <td>${lista.nome}</td>
    <td>${lista.endereco}</td>
    <td>${lista.telefone}</td>
    <td>${lista.estado}</td>
    <td>
        <a
href="ServletCliente?cmd=excluir&id=${lista.idCliente}">
            Excluir
        </a>
    </td>
</tr>
</c:forEach>
```

Melhorando a Aplicação MVC

■ Modificar no ServletCliente:

```
if cmd.equals("listar")
{
    List clientesList = null;

    cli = new ClienteDAO();
    clientesList = cli.todosClientes();

    request.setAttribute("clientesList", clientesList);
    rd = request.getRequestDispatcher( "/mostrarClientes.jsp" );
}
```

Melhorando a Aplicação MVC

■ Modificar no ServletCliente:

```
If cmd.equals("adicionar")
{
    ....
    rd = request.getRequestDispatcher(
        "ServletAutores?cmd=listar" );
}
```



Funções e SQL

- Diversas funções de manipulação de Strings, entre elas: `length` , `toUpperCase`, `toLowerCase`, `substring`, `substringAfter`, `substringBefore`, etc.

```
<c:out value="${fn:trim(stringComEspacos)}"/>
```

Funções e SQL

- SQL direto do JSTL com opcional uso de Data Source

```
<sql:setDataSource  
var="exemplo"  
driver="${sessionScope.driverDoBanco}"  
url="${sessionScope.urlDoBanco}"  
user="${sessionScope.usuarioDoBanco}"  
password="${sessionScope.senhaDoBanco}"  
>  
<sql:query var="buscaTodosClientes">  
    SELECT * FROM CLIENTES  
</sql:query>
```



Funções e SQL

Da mesma forma, temos:

```
<sql:update var="atualizaTodosClientes">
```

```
    UPDATE CLIENTES SET (....)
```

```
</sql:update>
```

```
<sql:update var="insereCliente">
```

```
    INSERT INTO CLIENTES (....) VALUES (....)
```

```
</sql:update>
```

```
<sql:update var="excluiClientes">
```

```
    DELETE FROM CLIENTES WHERE (....)
```

```
</sql:update>
```




Formatação e i18n

- Dinamicamente podemos exibir textos de outros idiomas:

```
<fmt:setLocale value="it_IT"/>
```

```
<fmt:setBundle
```

```
  basename="org.apache.taglibs.standard.examples.i  
  18n.Resources" var="itBundle" scope="page"/>
```

```
<fmt:message key="greetingMorning"  
bundle="${itBundle}"/>
```

Resultado: Buon giorno!

Formatação e i18n

- Dinamicamente exibimos facilmente valores numéricos e datas:

```
<fmt:formatNumber value="${carro.ipva}"  
type="currency"/>
```

```
<fmt:formatDate value="${compraDoProduto}"  
type="date" dateStyle="full"/>.
```



Internacionalizando

■ Passos:

- Criar dois arquivos com a extensão **.properties**
>> pacote **internacional**
- Digitar neste arquivos as chaves e os textos:

Sample ResourceBundle properties file

titulo=Internacionalização

ingles=Inglês

portugues=Português

nome=Nome

email=E-mail

enviar=Enviar

Internacionalizando

■ Passos:

- Alternativamente, pode-se adicionar outras localidades >> opções do arquivo de propriedades

Chave	idioma padrão	en_US - English (United States)
titulo	Internacionalização	Internationalization
ingles	Inglês	English
portugues	Português	Portuguese
nome	Nome	Name
email	E-mail	E-mail
enviar	Enviar	Send

- Altere os textos e suas chaves.

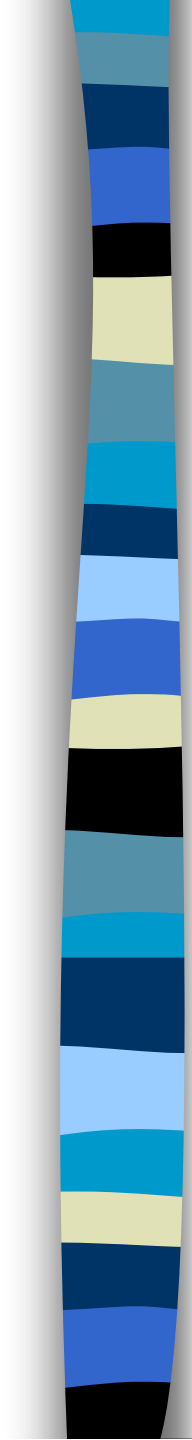
Internacionalizando

- Criar uma página *TesteInternacional.jsp* para teste:

.....

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<c:choose>
  <c:when test="${param.locale eq 'en_US'}">
    <fmt:setLocale value="en_US" />
  </c:when>
  <c:otherwise>
    <fmt:setLocale value="pt_BR" />
  </c:otherwise>
</c:choose>

<fmt:setBundle basename="br.com.integrator.rotulos"/>
```



```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=ISO-8859-1">
    <title><fmt:message key="titulo" /></title>
  </head>
  <body>

    <a href="?locale=en_US"><fmt:message key="ingles" /></a>
    <a href="?locale=pt_BR"><fmt:message key="portugues" /></a>
    <br />
    <form action="">
      <fmt:message key="nome" />: <input type="text"
      name="nome" />
      <br />
      <fmt:message key="email" />: <input type="text"
      name="email" /><br />
      <input type="submit" value="<fmt:message key="enviar" />" />
    </form>
  </body>
</html>
```



Action `<fmt:setLocale>`

- Este action pode ser usada para alterar o local do cliente especificado no processamento de uma página JSP.
- Exemplo:

```
<fmt:setLocale value="en_US" scope="session"/>
```

- **Value:** especifica um código de duas partes que representa o código de idioma ISSO-639 e o código do país ISSO-3166



Exibindo os textos do idioma definido.

- Com local definido, ou pela configuração do browser do cliente ou através de uso da action `<fmt:setLocale />`, o JSTL precisa usar os textos pré-definidos no idioma escolhido para exibir o conteúdo no browser com o idioma especificado por seu local.
- Utiliza-se uma coleção de recursos que é conhecida como ***resource bundle***
 - Utilizada pelo desenvolvedor para cada local que se pretende aderir.
 - É implementada por padrão de uma **chave = valor** em um arquivo de propriedades (extensão .properties).



Actions **<fmt:Bundle />** **<fmt:setBundle />**

- Para habilitar o uso de textos no idioma definido, especificamos o pacote de recursos exigido que forneçam as mensagens localizadas.
- Utiliza-se as tags **<fmt:Bundle />** ou **<fmt:setBundle />** para especificarmos um recurso >> uma vez declarado, pode ser utilizado para fornecer os textos no idioma definido.
- As tags **<fmt:Bundle />** ou **<fmt:setBundle />** embora semelhantes, podem ser usadas de diferentes modos:
- A action **<fmt:Bundle />** é usada para declarar uma localização de contexto l18n para usar por tags dentro de seu corpo:



<fmt:Bundle />

```
<fmt:bundle basename="Rotulos"/>  
    <fmt:message key="rotulos.nome"/>  
    <fmt:message key="rotulos.email"/>  
</fmt:bundle>
```

- O resource bundle com o nome rótulo é declarado para fornecer recursos localizados para as actions <fmt:message />



<fmt:Bundle />

- Como a action **<fmt:bundle />** é projetada para trabalhar com aninhamento da action **<fmt:message />**, um atributo opcional também pode ser usado:

...

<fmt:bundle basename="Rotulos" prefix="rotulos" />

...

- O atributo opcional prefix habilita a colocação de um prefixo pré-definido que é fundamental para qualquer action **<fmt:message />** aninhado tornando seu uso mais simplificado.

<fmt:setBundle />

- Também fornece funcionalidade semelhante a `<fmt:bundle />`, com uma diferença sutil:
 - Em vez de ter que aninhar qualquer action `<fmt:message />` como conteúdo de corpo, a action `<fmt:setBundle />` habilita um pacote de recursos a serem armazenados na variável de contexto da aplicação.
 - Assim, qualquer action `<fmt:message />` que aparecer, em qualquer parte da página JSP, pode acessar o pacote sem ter que ser declarada continuamente:

```
<fmt:setBundle basename="Rotulos" />
```

```
<fmt:message prefix="rotulos.nome" />
```



Action `<fmt:Message />`

- Usa um parâmetro fundamental, **key**, para extrair a mensagem do pacote de recursos e imprimir com JspWriter.
- Outro parâmetro opcional, **var**, habilita a mensagem localizada a ser armazenada em um parâmetro em vez de ser impressa pelo JspWriter
 - A extensão desta variável pode ser fixada usando o atributo **scope**.

A Plataforma Java

Programação Web com Java



Uso de JSTL

- Taglibs Simples -



Prof. Giuliano Prado de Moraes Giglio, M.Sc.





O JSP sem padrão

- É muito comum!
- Scriptlets espalhados em várias páginas
- Código confuso = manutenção +trabalhosa
- Regras de negócio não deveriam ser tratadas dentro do JSP (foge ao padrão MVC!)
- JSP deve ser usado somente para a camada de visualização (viewer), somente exibindo o conteúdo dos objetos e nada de scriptlets



JSTL: o JSP com padrão

■ HISTÓRICO

- **Julho/2001** - Criado o projeto (JSR-052)
- **Junho/2002** – lançado o JSTL 1.0 baseado no JSP 1.2 (Tomcat4 e maioria dos servidores corporativos)
- **Janeiro/2004** - lançado o JSTL 1.1 baseado no JSP 2.0 (Tomcat 5)
- **Julho/2004** - lançado o JSTL 1.1.1



JSTL: o JSP com padrão

- Padronizar as aplicações JSP !
- Dar soluções fáceis de usar para tarefas mais comuns

JSTL: o JSP com padrão

Tipos de soluções JSTL 1.0

Tipo	URI	Prefixo	Exemplo
Core	http://java.sun.com/jstl/core	c	<c:nomeDaTag...>
XML	http://java.sun.com/jstl/xml	x	<x:nomeDaTag...>
Internationalization	http://java.sun.com/jstl/fmt	fmt	<fmt:nomeDaTag...>
Database	http://java.sun.com/jstl/sql	sql	<sql:nomeDaTag...>

JSTL: o JSP com padrão

Tipos de soluções JSTL 1.1

Tipo	URI	Prefixo	Exemplo
Core	http://java.sun.com/jsp/jstl/core	c	<c:nomeDaTag...>
XML	http://java.sun.com/jsp/jstl/xml	x	<x:nomeDaTag...>
l18n	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:nomeDaTag...>
Database	http://java.sun.com/jsp/jstl/sql	sql	<sql:nomeDaTag...>
Funções	http://java.sun.com/jsp/jstl/functions	fn	<fn:nomeDaTag...>



Como instalar o JSTL?

■ 1. Faça o download:

<http://www.apache.org/dist/jakarta/taglibs/standard/>

■ 2. Descompacte o arquivo e copie:

a) `/jakarta-taglibs-standard-1.*/tld/*` para **WEB-INF**

b) `/jakarta-taglibs-standard-1.*/lib/*` para **WEB-INF/lib**

Como instalar o JSTL?

3. Adicione essas informações no **web.xml**:

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
  <taglib-location>/WEB-INF/fmt.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt-rt</taglib-uri>
  <taglib-location>/WEB-INF/fmt-rt.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
  <taglib-location>/WEB-INF/c-rt.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
  <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql-rt</taglib-uri>
  <taglib-location>/WEB-INF/sql-rt.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
  <taglib-location>/WEB-INF/x.tld</taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/x-rt</taglib-uri>
  <taglib-location>/WEB-INF/x-rt.tld</taglib-location>
</taglib>
```

Como instalar o JSTL?

- 4. Na sua página JSP declare os tipos que for utilizar:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

- 5. Adicione a biblioteca JSTL em seu projeto

- 6. Depois é só sair usando!

```
<H1><c:out value="Conexão Java 2004" /></H1>
```

Sintaxe

- Não importa onde a Expressão Lógica é usada, ela sempre é invocada de uma maneira consistente:

- `${exp}` ou `#{exp}`

Obs.: Na plataforma J2EE não podem ser usadas em conjunto.

- Exemplos:

- `${true}`
- `${ "aqui jaz um texto" }`
- `${5*4} <!-- resulta no numero 20--%>`



Entendendo o JSTL

- O JSTL é uma coleção de quatro bibliotecas tags.
- Cada biblioteca de tags fornece ações úteis (ou tags) baseados nas seguintes áreas funcionais:
 - Core
 - Internacionalização (l18n) e formatação
 - Acesso a banco de dados relacional (tags SQL)
 - Processamento de XML (tags XML)

A Core Tag Library

- Contém um centro de ações de propósito geral, que fornecem soluções simples, mas efetivas, a problemas comuns que os desenvolvedores experimentam em quase toda aplicação JSP.
- Grupo de tags mais usadas freqüentemente:
 - **<c:if />** para condições
 - **<c:forEach />** para interação
 - **<c:choose /> ... <c:when />... <c:otherwise />** para um fluxo seletivo
 - **<c:set />** e **<c:remove />** para trabalhar com escopo de variáveis
 - **<c:out />** para fazer a saída de valores de variáveis e expressões
 - **<c:catch />** para trabalhar com exceções Java
 - **<c:url />** para criar e trabalhar com URL's

Exibindo Objetos

- O formato padrão é:
`<c:out value="{objeto.atributo}" />`
- Imagine um objeto Pessoa, com os atributos de nome e idade:

Aluno : `<c:out value="{Pessoa.nome}" />`
com `<c:out value="{Pessoa.idade}" />` anos.

Usando variáveis

- Para setar valores de variáveis:

```
<c:set var="nomeDaVariavel" value="valor"/>
```

ou

```
<c:set var="nomeDaVariavel">valor</c:set>
```

- Para exibir o valor da variável acima:

```
<c:out value="${nomeDaVariavel}" />
```

Recuperando valores

- Para exibir o valor de parâmetros passados para um formulário :

```
<c:out value="${param.nomeDoParametro}" />
```

- Exemplo:

Bem-vindo <c:out value="\${param.usuario}" />!

Recuperando valores

- Para exibir o valor de parâmetros passados para um formulário via **session**:

```
<c:out value="${sessionScope.nomeDoParametro}" />
```

- Exemplo:

Bem-vindo

```
<c:out value="${sessionScope.usuario}" />!
```

Recuperando valores

- Para exibir o valor de parâmetros passados para um formulário via ***request***:

```
<c:out value="${requestScope.nomeDoParametro}" />
```

- Exemplo:

Bem-vindo

```
<c:out value="${requestScope.usuario}" />!
```



Recuperando valores

Redefina a aplicação sobre Login
para que os dados sejam
recuperados e utilizados na
página **bemVindo.jsp** via
TagLibs

Operadores Condicionais

- O operador condicional para apenas uma opção:

```
<c:if test="${expressão}">
```

...

```
</c:if>
```

- Exemplo:

```
<c:if test="${param.nome == 'nobody'}">
```

Acesso Negado

```
</c:if>
```


Operadores Condicionais

O operador condicional para duas ou mais opções:

```
<c:choose>
```

```
<c:when test="{expressão}">
```

```
...
```

```
</c:when>
```

```
<c:when test="{expressão}">
```

```
...
```

```
</c:when>
```

```
<c:otherwise>
```

```
...
```

```
</c:otherwise>
```

```
</c:choose>
```

Operadores Condicionais

- Exemplo:

```
<c:choose>
```

```
<c:when test="${hora<12}">
```

Bom dia!

```
</c:when>
```

```
<c:when test="${hora<18}">
```

Boa tarde!

```
</c:when>
```

```
<c:otherwise>
```

Boa noite!

```
</c:otherwise>
```

```
</c:choose>
```



Loops

- O loop **forEach** é usado para percorrer dados de uma coleção ou para iterações simples:

```
<c:forEach var="nome" items="${nomeDaCollection}">  
  <c:out value="${nome}"/>  
</c:forEach>
```

- Ou seja, ele percorrerá a coleção identificada em items, sendo a variável definida nome contendo cada item da coleção, a cada loop realizado.

Loops

- Exemplo:

```
<c:forEach var="aluno" items="${listaAlunos}">  
  <c:out value="${aluno.nome}"/>  
</c:forEach>
```

Loops

- Podemos utiliza-lo também como um FOR simples de controle de iterações:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title><c:forEach> Tag Example</title>
    </head>
    <body>
        <c:forEach var="i" begin="1" end="5" step="1">
            Item <c:out value="${i}"/><p>
        </c:forEach>
    </body>
</html>
```



Aplicando *forEach*

Modifique o exercício 2 de fixação de Servlets, utilizando *forEach* para exibição da tabela de parcelamento da página **simulacaoFinaciamento.jsp**, além do uso de tags condicionais para a verificação das linhas pares e ímpares

Formatação e i18n

- Dinamicamente podemos exibir textos de outros idiomas:

```
<fmt:setLocale value="it_IT"/>
```

```
<fmt:setBundle
```

```
  basename="org.apache.taglibs.standard.examples.i  
  18n.Resources" var="itBundle" scope="page"/>
```

```
<fmt:message key="greetingMorning"
```

```
bundle="${itBundle}"/>
```

Resultado: Buon giorno!

Formatação e i18n

- Dinamicamente exibimos facilmente valores numéricos e datas:

```
<fmt:formatNumber value="${carro.ipva}"  
type="currency"/>
```

```
<fmt:formatDate value="${compraDoProduto}"  
type="date" dateStyle="full"/>.
```




Internacionalizando

■ Passos:

- Criar dois arquivos com a extensão **.properties**
>> pacote **internacional**
- Digitar neste arquivos as chaves e os textos:

Sample ResourceBundle properties file

titulo=Internacionalização

ingles=Inglês

portugues=Português

nome=Nome

email=E-mail

enviar=Enviar

Internacionalizando

■ Passos:

- Alternativamente, pode-se adicionar outras localidades >> opções do arquivo de propriedades

Chave	idioma padrão	en_US - English (United States)
titulo	Internacionalização	Internationalization
ingles	Inglês	English
portugues	Português	Portuguese
nome	Nome	Name
email	E-mail	E-mail
enviar	Enviar	Send

- Altere os textos e suas chaves.

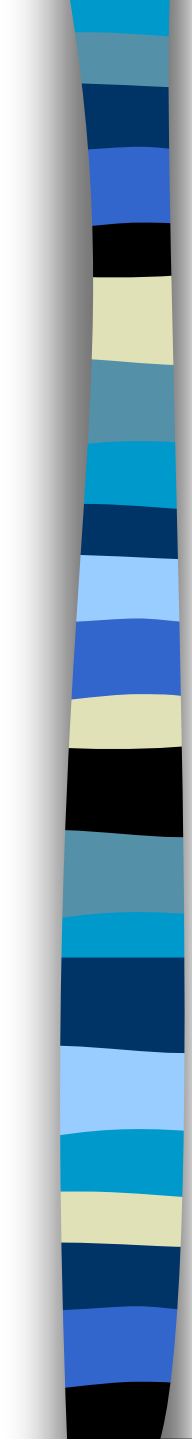
Internacionalizando

- Criar uma página *TesteInternacional.jsp* para teste:

.....

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<c:choose>
  <c:when test="${param.locale eq 'en_US'}">
    <fmt:setLocale value="en_US" />
  </c:when>
  <c:otherwise>
    <fmt:setLocale value="pt_BR" />
  </c:otherwise>
</c:choose>

<fmt:setBundle basename="br.com.integrator.rotulos"/>
```



```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=ISO-8859-1">
    <title><fmt:message key="titulo" /></title>
  </head>
  <body>

    <a href="?locale=en_US"><fmt:message key="ingles" /></a>
    <a href="?locale=pt_BR"><fmt:message key="portugues" /></a>
    <br />
    <form action="">
      <fmt:message key="nome" />: <input type="text"
name="nome" />
      <br />
      <fmt:message key="email" />: <input type="text"
name="email" /><br />
      <input type="submit" value="<fmt:message key="enviar" />" />
    </form>
  </body>
</html>
```



Action `<fmt:setLocale>`

- Este action pode ser usada para alterar o local do cliente especificado no processamento de uma página JSP.
- Exemplo:

```
<fmt:setLocale value="en_US" scope="session"/>
```

- **Value:** especifica um código de duas partes que representa o código de idioma ISSO-639 e o código do país ISSO-3166



Exibindo os textos do idioma definido.

- Com local definido, ou pela configuração do browser do cliente ou através de uso da action `<fmt:setLocale />`, o JSTL precisa usar os textos pré-definidos no idioma escolhido para exibir o conteúdo no browser com o idioma especificado por seu local.
- Utiliza-se uma coleção de recursos que é conhecida como ***resource bundle***
 - Utilizada pelo desenvolvedor para cada local que se pretende aderir.
 - É implementada por padrão de uma **chave = valor** em um arquivo de propriedades (extensão .properties).



Actions **<fmt:Bundle />** **<fmt:setBundle />**

- Para habilitar o uso de textos no idioma definido, especificamos o pacote de recursos exigido que forneçam as mensagens localizadas.
- Utiliza-se as tags **<fmt:Bundle />** ou **<fmt:setBundle />** para especificarmos um recurso >> uma vez declarado, pode ser utilizado para fornecer os textos no idioma definido.
- As tags **<fmt:Bundle />** ou **<fmt:setBundle />** embora semelhantes, podem ser usadas de diferentes modos:
- A action **<fmt:Bundle />** é usada para declarar uma localização de contexto l18n para usar por tags dentro de seu corpo:



<fmt:Bundle />

```
<fmt:bundle basename="Rotulos"/>  
    <fmt:message key="rotulos.nome"/>  
    <fmt:message key="rotulos.email"/>  
</fmt:bundle>
```

- O resource bundle com o nome rótulo é declarado para fornecer recursos localizados para as actions <fmt:message />



<fmt:Bundle />

- Como a action **<fmt:bundle />** é projetada para trabalhar com aninhamento da action **<fmt:message />**, um atributo opcional também pode ser usado:

...

<fmt:bundle basename="Rotulos" prefix="rotulos" />

...

- O atributo opcional prefix habilita a colocação de um prefixo pré-definido que é fundamental para qualquer action **<fmt:message />** aninhado tornando seu uso mais simplificado.

<fmt:setBundle />

- Também fornece funcionalidade semelhante a `<fmt:bundle />`, com uma diferença sutil:
 - Em vez de ter que aninhar qualquer action `<fmt:message />` como conteúdo de corpo, a action `<fmt:setBundle />` habilita um pacote de recursos a serem armazenados na variável de contexto da aplicação.
 - Assim, qualquer action `<fmt:message />` que aparecer, em qualquer parte da página JSP, pode acessar o pacote sem ter que ser declarada continuamente:

```
<fmt:setBundle basename="Rotulos" />
```

```
<fmt:message prefix="rotulos.nome" />
```



Action `<fmt:Message />`

- Usa um parâmetro fundamental, **key**, para extrair a mensagem do pacote de recursos e imprimir com JspWriter.
- Outro parâmetro opcional, **var**, habilita a mensagem localizada a ser armazenada em um parâmetro em vez de ser impressa pelo JspWriter
 - A extensão desta variável pode ser fixada usando o atributo **scope**.