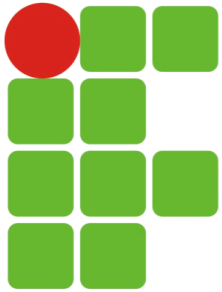


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE
Campus Natal - Central

JavaServer Pages – JSP

Prof. Fellipe Aleixo (*fellipe.aleixo@ifrn.edu.br*)

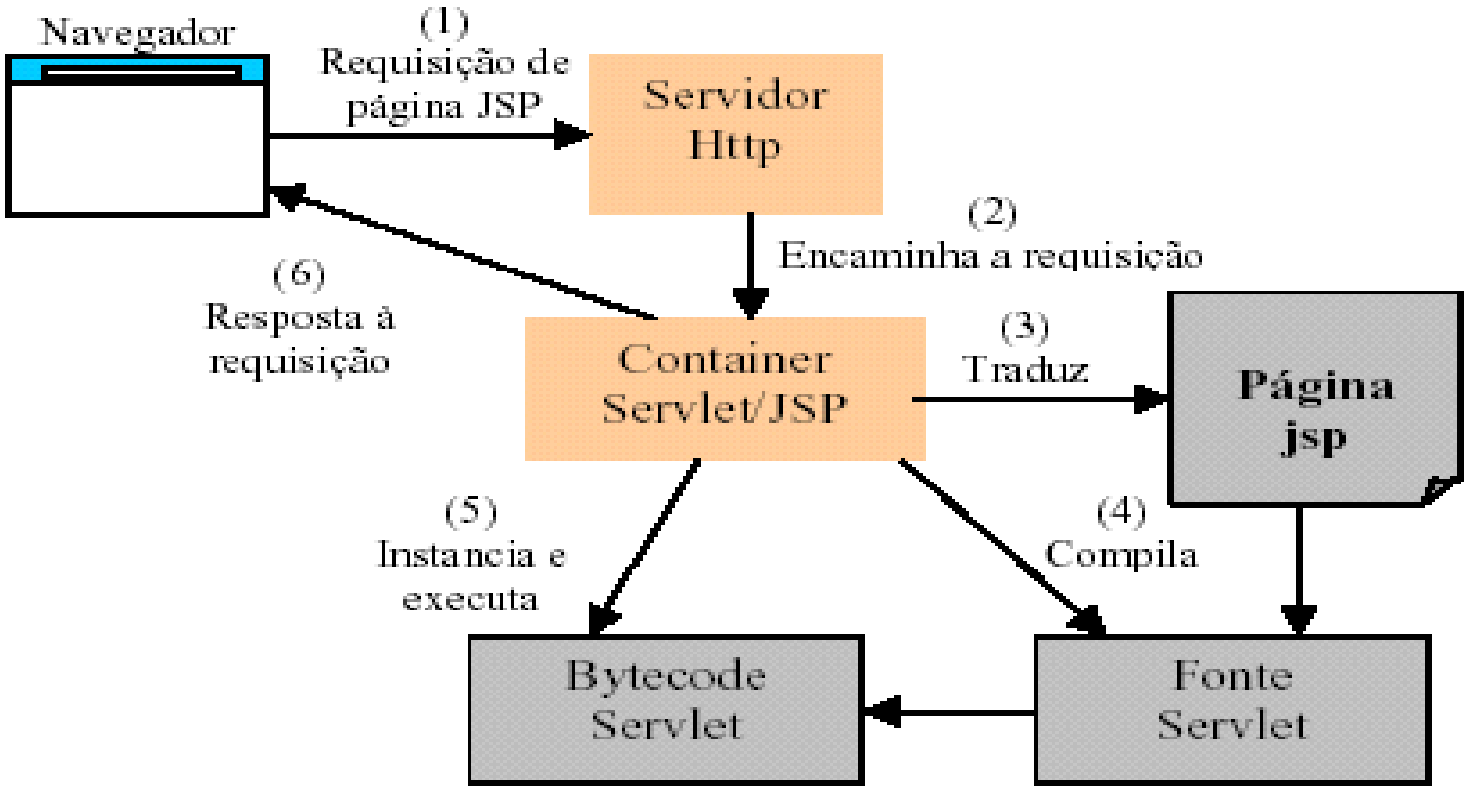
O que é uma Página JSP?

- Tem a forma de uma página HTML com trechos de código Java embutidos e outras *tags* especiais
  - A parte dinâmica da página é gerada pelo código Java
- Simplificam a geração de conteúdo dinâmico para Web Designers
- Uma JSP é automaticamente transformada em servlet
- Dois formatos de JSP: padrão ou formato XML

Servlet x JSP

- Servlets
  - Melhor em termos de Eng. Software
  - Mais flexível
  - Não permite independência entre o designer e o programador
- JSP
  - Mais fácil de aprender
  - Um nível maior de abstração pro *Servlets*
  - O *Web Designer* pode trabalhar independente do *Web Developer* e vice-versa

Introdução


Navegador
(1)
Requisição de
página JSP
Servidor
Http
(2)
Encaminha a requisição
(6)
Resposta à
requisição
Container
Servlet/JSP
(3)
Traduz
Página
jsp
(5)
Instancia e
executa
(4)
Compila
Bytecode
Servlet
Fonte
Servlet

Problemas de Servlets

- Servlets forçam o programador a embutir código HTML dentro de código Java
  - Desvantagem: se a maior parte do que tem que ser gerado é texto ou código HTML estático
  - Mistura as coisas: programador tem que ser bom Web Designer e se virar sem ferramentas de Web Design

```
Date hoje = new Date();
out.println("<body>");
out.println("<p>A data de hoje é "+hoje+".</p>");
out.println("<body>");
```

HojeServlet.java

- Uma solução inteligente é escrever um arquivo de *template*

```
<body>
<p>A data de hoje é <!--#data#-->.</p>
<body>
```

template.html

O que são JavaServer Pages?

- JSP é uma tecnologia padrão, baseada em templates para servlets. O mecanismo que a traduz é embutido no servidor

- Há várias outras alternativas populares
  - Apache Cocoon XSP: baseado em XML (xml.apache.org/cocoon)
  - Jakarta Velocity (jakarta.apache.org/velocity)
  - WebMacro (www.webmacro.org)

O que são JavaServer Pages?

- Solução do problema anterior usando templates JSP

```
<body>
<p>A data de hoje é <%=new Date() %>.</p>
<body>
```

hoje.jsp

- Em um servidor que suporta JSP, processamento de JSP passa por uma camada adicional onde a página é transformada (compilada) em um servlet

- Acesso via URL usa como localizador a própria página

Exemplos de JSP

- A forma mais simples de criar documentos JSP, é
  - 1. Mudar a extensão de um arquivo HTML para .jsp
  - 2. Colocar o documento em um servidor que suporte JSP
- Fazendo isto, a página será transformada em um servlet
  - A compilação é feita no primeiro acesso
  - Nos acessos subseqüentes, a requisição é redirecionada ao servlet que foi gerado a partir da página
- Transformado em um JSP, um arquivo HTML pode conter blocos de código (scriptlets): <% ... %> e expressões <%= ... %>

```
<p>Texto repetido:
<% for (int i = 0; i < 10; i++) { %>
  <p>Esta é a linha <%=i %>
<% }%>
```

Exemplo de JSP

```
<%@ page import="java.util.*" %>
<%@ page import="j2eetut.webhello.MyLocales" %>
<%@ page contentType="text/html; charset=iso-8859-9" %>
<html><head><title>Localized Dates</title></head><body bgcolor="white">
<a href="index.jsp">Home</a>
<h1>Dates</h1>
<jsp:useBean id="locales" scope="application"
                          class="j2eetut.webhello.MyLocales"/>
<form name="locale" action="locale.jsp" method="post">
<b>Locale:</b><select name=locale>
<%
   Iterator i = locales.getLocaleNames().iterator();
   String selectedLocale = request.getParameter("locale");
   while (i.hasNext()) {
      String locale = (String)i.next();
      if (selectedLocale != null && selectedLocale.equals(locale) ) {
         out.print("<option selected>" + locale + "</option>");
      } else { %>
         <option><%=locale %></option>
<%    }
   } %>
</select><input type="submit" name="Submit" value="Get Date">
</form>
<p><jsp:include page="date.jsp" flush="true" />
</body></html>
```

diretivas

bean

scriptlet

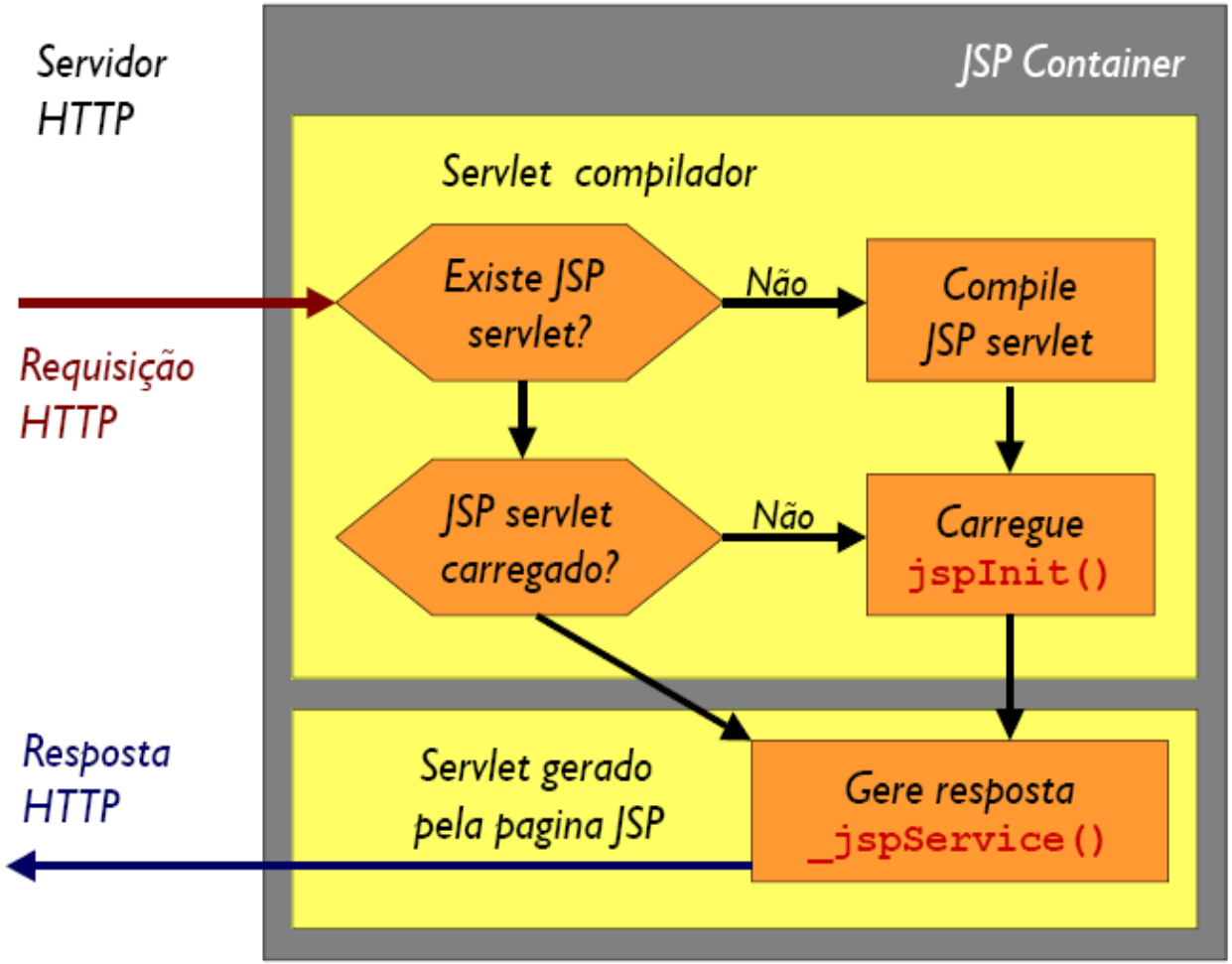
expressão

ação

Ciclo de Vida

1. Quando uma requisição é mapeada em uma JSP, o container
   i. Verifica se o servlet correspondente à página é mais antigo que a página (ou se não existe)
   ii. Se o servlet não existe ou é mais antigo, a página JSP será compilada para gerar novo servlet
   iii. Com o servlet atualizado, a requisição é redirecionada para ele
2. Deste ponto em diante, o comportamento equivale ao ciclo de vida do servlet, mas os métodos são diferentes
   i. Se o servlet ainda não estiver na memória, ele é instanciado, carregado e seu método jspInit() é chamado
   ii. Para cada requisição, seu método _jspService(req, res) é chamado Ele é resultado da compilação do corpo da página JSP
   iii. No fim da vida, o método jspDestroy() é chamado

Funcionamento JSP



Sintaxe dos Elementos JSP

- Podem ser usados em documentos HTML ou XML

- Todos são interpretados no servidor
  - diretivas: <%@ ... %>
  - declarações: <%! ... %>
  - expressões: <%= ... %>
  - *scriptlets*: <% ... %>
  - comentários: <%-- ... --%>
  - ações: <jsp:ação ... />
  - *custom tags*: <prefixo:elemento ... />

Comentários JSP

Comentários	Descrição
<!-- ... -->	Comentário copiado para a saída
<%! // ... %>	Comentários copiados para o código do servlet gerado
<%! /* ... */ %>	
<% // ... %>	
<%-- ... --%>	Comentário JSP. Não é copiado nem para a saída, nem para o servlet

Comentários JSP

- Comentários HTML <!-- --> não servem para JSP

```
<!-- Texto ignorado pelo browser mas não pelo
    servidor. Tags são processados -->
```

- Comentários JSP
  - Podem ser usados para comentar blocos JSP
  - Pode-se também usar comentários Java quando dentro de *scriptlets*, expressões ou declarações

```
<%-- Texto, código Java, <HTML> ou tags
    <%JSP%> ignorados pelo servidor --%>

<% código JSP ... /* texto ou comandos Java
    ignorados pelo servidor */ ... mais código
%>
```

Diretivas

- Contém informações necessárias ao processamento da classe do servlet que gera a página JSP
- Sintaxe :

  <%@ diretiva atrib1 atrib2 ... %>
- Principais diretivas:
  - page: atributos relacionados à página
  - include: inclui outros arquivos na página
  - taglib: declara um biblioteca de tags customizada
- Exemplos:

```
<%@ page import="java.net.*, java.io.*"
        session="false"
        errorPage="/erro.jsp" %>
<%@ include file="navbar.jsp" %>
```

Alguns Atributos da Diretiva page

Atributo	Valor	Descrição	Valor Default
autoFlush	true/false	Define o comportamento quando o buffer está cheio. Esvazia automaticamente ou gera uma exceção quando o buffer estoura	true
buffer	none/n KB	Configura o tamanho do buffer de saída	8 KB
contentType	tipo MIME	Configura o tipo MIME	text/html; charset=ISO-8859-1
errorPage	URL relativo	Configura a JSP de erro	
import	Classes e interfaces a importar	Permite importação de pacotes Java padrão	

Alguns Atributos da Diretiva page

Atributo	Valor	Descrição	Valor Default
isErrorPage	true/false	Indica se a página receberá exceções de outras páginas	false
isThreadSafe	true/false	Determina se o servlet gerado implementa a interface SingleThreadModel	true
session	true/false	Indica se a página atual precisa de informações de sessão	true

Atributos buffer e autoflush

- Pode-se redirecionar, criar um cookie ou modificar o tipo de dados gerado por uma página JSP em qualquer parte dela
  - Essas operações são realizadas pelo browser e devem ser passadas através do cabeçalho de resposta do servidor
  - Lembre-se que o cabeçalho termina ANTES que os dados comecem

- O servidor JSP armazena os dados da resposta do servidor em um buffer (de 8kB, default) antes de enviar
  - Assim é possível montar o cabeçalho corretamente antes dos dados, e permitir a escolha de onde e quando definir informações de cabeçalho
  - O buffer pode ser redefinido por página (diretiva page buffer). Aumente-o se sua página for grande
  - autoFlush determina se dados serão enviados quando buffer encher ou se o programa lançará uma exceção

Declarações

- Dão acesso ao corpo da classe do servlet. Permitem a declaração de variáveis e métodos em uma página

- Úteis para declarar:
  - Variáveis e métodos de instância (pertencentes ao servlet)
  - variáveis e métodos estáticos (pertencentes à classe do servlet)
  - Classes internas (estáticas e de instância), blocos estáticos, etc.

Declarações

- Sintaxe

  <%! declaração %>

- Exemplos:

```
<%! public final static String[] meses =
     {"jan", "fev", "mar", "abr", "mai", "jun"};
%>
<%! public static String getMes() {
       Calendar cal = new GregorianCalendar();
       return meses[cal.get(Calendar.MONTH)];
    }
%>
```

Declarações

- jspInit() e jspDestroy() permitem maior controle sobre o ciclo de vida do servlet
  - Ambos são opcionais
  - Úteis para inicializar conexões, obter recursos via JNDI, ler parâmetros de inicialização do web.xml, etc.
- jspInit() → chamado uma vez, antes da primeira requisição, após instanciar o servlet
- jspDestroy() → quando o servlet deixa a memória

```
<%!
    public void jspInit() { ... }
    public void jspDestroy() { ... }
%>
```

Expressões

- Quando processadas, retornam um valor que é inserido na página no lugar da expressão
- Sintaxe:

```
<%= expressão %>
```

- Equivale a out.print(expressão), portanto, não pode terminar em ponto-e-vírgula
  - Todos os valores resultantes das expressões são convertidos em Strling antes de serem redirecionados à saída padrão

Scriptlets

- Scriptlets: Blocos de código que são executados sempre que uma página JSP é processada
- Correspondem a inserção de sequências de instruções no método _jspService() do *servlet* gerado
- Sintaxe:

```
<% instruções Java; %>
```

Ações Padronizadas

- Sintaxe:

```
<jsp:nome_ação atrib1 atrib2 ... >
    <jsp:param name="xxx" value="yyy"/>
    ...
</jsp:nome_ação>
```

Ações Padronizadas

- Permitem realizar operações (e meta-operações) externas ao *servlet* (tempo de execução)
  - Concatenação de várias páginas em uma única resposta
  - <jsp:forward> e <jsp:include>
  - Inclusão de JavaBeans
  - <jsp:useBean>, <jsp:setProperty> e
  - <jsp:getProperty>
  - Geração de código HTML para Applets
  - <jsp:plugin>

Exemplo de Ações

```
<%
if (Integer.parseInt(totalImg) > 0) {
%>
  <jsp:forward page="selecimg.jsp">
     <jsp:param name="totalImg"
                value="<%= totalImg %>"/>
     <jsp:param name="pagExibir" value="1"/>
  </jsp:forward>
<%
} else {
%>
  <p>Nenhuma imagem foi encontrada.
<%
}
%>
```

Objetos Implícitos JSP

- São variáveis locais previamente inicializadas
  - Disponíveis nos blocos <% ... %> (scriptlets) de qualquer página
  - Exceto session e exception que dependem de @page para serem ativados/desativados

- Objetos do servlet
  - page
  - config

Objetos Implícitos JSP

- Entrada e saída
  - request
  - response
  - out
- Objetos contextuais
  - session
  - application
  - pageContext
- Controle de exceções
  - exception

Objeto page

- Referência para o servlet gerado pela página
  - Equivale a "this" no servlet
- Pode ser usada para chamar qualquer método ou variável do servlet ou superclasses
  - Tem acesso aos métodos da interface javax.servlet.jsp.JspPage (ou HttpJspPage)
  - Pode ter acesso a mais variáveis e métodos se estender alguma classe usando a diretiva @page extends:

```
<%@ page extends="outra.Classe" %>
```

- Exemplo:

```
<% HttpSession sessionCopy = page.getSession(); %>
```

Objeto config

- Referência para os parâmetros de inicialização do servlet (se existirem) através de objeto ServletConfig
- Equivale a page.getServletConfig()
- Exemplo:

```
<%  String user = config.getInitParameter("nome");
    String pass = config.getInitParameter("pass"); %>
```

- Parâmetros de inicialização são definidos no web.xml

```
<servlet>
    <servlet-name>ServletJSP</servlet-name>
    <jsp-page>/pagina.jsp</jsp-page>
    <init-param>
      <param-name>nome</param-name>
      <param-value>guest</param-value>
    </init-param>
</servlet>
```

Objeto request

- Referência para os dados de entrada enviados na requisição do cliente (no GET ou POST, por exemplo)
  - É um objeto do tipo javax.servlet.http.HttpServletRequest
- Usado para
  - Guardar e recuperar atributos que serão usadas enquanto durar a requisição (que pode durar mais de uma página)
  - Recuperar parâmetros passados pelo cliente (dados de um formulário HTML, por exemplo)
  - Recuperar cookies
  - Descobrir o método usado (GET, POST)

Exemplos

- URL no browser:
  - http://servidor/programa.jsp?nome=Fulano&id=5
- Recuperação dos parâmetros no programa JSP:

```
<%
    String nome = request.getParameter("nome");
    String idStr = request.getParameter("id");
    int id = Integer.parseInt(idStr);
%>
<p>Bom dia <%=nome %>! (cod: <%=id %>
```

- Cookies

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

Objeto response

- Referência aos dados de saída enviados na resposta do servidor enviada ao cliente
  - É um objeto do tipo

    javax.servlet.http.HttpServletResponse
- Usado para
  - Definir o tipo dos dados retornados (default: text/thml)
  - Criar cookies

```
Cookie c = new Cookie("nome", "valor");
response.addCookie(c);
```

  - Definir cabeçalhos de resposta
  - Redirecionar

```
response.sendRedirect("pagina2.html");
```

Objeto out

- Representa o stream de saída da página
- É instância da classe javax.servlet.jsp.JspWriter (implementação de java.io.Writer)
- Equivalente a response.getWriter();
- Principais métodos

  print() e println() - imprimem Unicode
- Os trechos de código abaixo são equivalentes

```
<% for (int i = 0; i < 10; i++) {
out.print("<p> Linha " + i);
} %>

<% for (int i = 0; i < 10; i++) { %>
<p> Linha <%= i %>
<% } %>
```

Objeto session

- Representa a sessão do usuário
  - O objeto é uma instância da classe javax.servlet.http.HttpSession
- Útil para armazenar valores que deverão permanecer durante a sessão (set/getAttribute())

```
Date d = new Date();
session.setAttribute("hoje", d);
  ...
Date d = (Date)
          session.getAttribute("hoje");
```

Objeto application

- Representa o contexto ao qual a página pertence
  - Instância de javax.servlet.ServletContext
- Útil para guardar valores que devem persistir pelo tempo que durar a aplicação
- Exemplo:

```
Date d = new Date();
application.setAttribute("hoje", d);
  ...
Date d = (Date)
  application.getAttribute("hoje");
```

Objeto pageContext

- Instância de javax.servlet.jsp.PageContext
- Oferece acesso a todos os outros objetos implícitos
- Constrói a página (mesma resposta) com informações localizadas em outras URLs
  - pageContext.forward(String) - mesmo que ação <jsp:forward>
  - pageContext.include(String) - mesmo que ação <jsp:include>

Objeto pageContext

- Métodos:
  - getPage() - retorna page
  - getRequest() - retorna request
  - getResponse() - retorna response
  - getOut() - retorna out
  - getSession() - retorna session
  - getServletConfig() - retorna config
  - getServletContext() - retorna application
  - getException() - retorna exception

Escopo dos Objetos

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis

- Métodos de pageContext permitem setar ou buscar atributos em qualquer objeto de escopo:
  - setAttribute(nome, valor, escopo)
  - getAttribute(nome, escopo)

Escopo dos Objetos

- Constantes da classe javax.servlet.jsp.PageContext identificam escopo de objetos
  - pageContext PageContext.PAGE_SCOPE
  - request PageContext.REQUEST_SCOPE
  - session PageContext.SESSION_SCOPE
  - application PageContext.APPLICATION_SCOPE

Objeto exception

- Não existe em todas as páginas - apenas em páginas designadas como páginas de erro

  <%@ page isErrorPage="true" %>
- Instância de java.lang.Throwable
- Exemplo:

```
<h1>Ocoreu um erro!</h1>
<p>A exceção é
<%= exception %>
Detalhes: <hr>
<% exception.printStackTrace(out); %>
```

Sessões

- Representa a sessão atual de um usuário individual
- Em geral, expira após 30 min
  - pode ser configurado através de setMaxInactiveInterval();
- Definindo atributo em uma sessão:
  - session.setAttribute(“nomeDaSessao”,”valor”);
- Recuperando atributo de uma sessão:
  - String temp = (String) session.getAttribute(“sessao”);

Autenticando um Usuário

```
<html>
<head></head>
<body>
    <form action="autenticar_usuario2.jsp" method="post">
     Login: <br>
    <input name="login" type="text" size="12" maxlength="15" >
    <br><br>
    Senha<br>
    <input name="senha" type="password" size="12" maxlength="15">
    <br><br>
    <input type="submit" value="Entrar">
    </form>
</body>
</html>
```

Autenticando um Usuário

```
<%@ page import="pacotes necessários" %>
<%
String login = request.getParameter("login");
String senha = request.getParameter("senha");

Fachada fachada = new Fachada();
Usuario usuario = fachada.autenticaUsuario(login, senha);

if (usuario != null) { /* Criar session do usuario */
        session.setAttribute("codFunc",usuario.getCodFunc());
     out.println("Acesso Permitido!");
} else {
    out.println("Acesso Negado!");
}
%>
```

Listando e Atualizando Objetos

```
<%@ page import="métodos necessários" %>
<%
Fachada fachada = new Fachada();

Vector carros = fachada.getCarros();
Enumeration e = carros.elements();
while (e.hasMoreElements()){
    Carro car = (Carro) e.nextElement(); %>
    <tr>
       <td>Carro:</td><td><%= car.getNome(); %></td>
    </tr>
<%
}
%>
```

Listando e Atualizando Objetos

```
<%@ page import="métodos necessários" %>
<%
Fachada fachada = new Fachada();
String nome = "Uno Mille";
String marca = "Fiat";

Carro car = new Carro(nome, marca);
try {
    fachada.inserirCarro(car);
} catch (Exception e) {
    /* Trato a exceção */
}
%>
```

Imports

- Funcionam como os imports de java
- Exemplo:

```
<%@page import = ”fachada.Fachada” %>
<%@page import = ”usuario.Usuario” %>
<%@page import = ”java.util.Vector” %>
<%@page import = ”java.util.Enumeration” %>
```

Include

- Serve para dar modularidade ao sistema
- Exemplo:
  - header.inc ou header.jsp

```
<html>
    <head><title>Meu site</title>
    </head>
    <body>
    <img src="meu_logotipo.jpg">
```

  - footer.inc ou footer.jsp

```
    </body>
</html>
```

Include

- form_logar.jsp

```
<form action="autenticar_usuario2.jsp" method="post">
    Login: <br>
    <input name="login" type="text" size="12" maxlength="15" >
    <br><br>
    Senha<br>
    <input name="senha" type="password" size="12" maxlength="15">
    <br><br>
    <input type="submit" value="Entrar">
</form>
```

Include

– Juntando tudo num único arquivo... Logar.jsp

```
<%@ include file="header.jsp" %>
<%@ include file="form_logar.jsp" %>
<%@ include file="footer.jsp" %>
```

Exercícios

1. Escreva um JSP data.jsp que imprima a data de hoje.
   a) Use Calendar e GregorianCalendar
2. Escreva um JSP temperatura.jsp que imprima uma tabela HTML de conversão Celsius-Fahrenheit entre -40 e 100 graus Celsius com incrementos de 10 em 10.
   a) A fórmula é F = 9/5 C + 32
3. Altere o exercício anterior para que a página também apresente um campo de textos para entrada de temperatura em um formulário que envie os dados com POST. Faça com que a própria página JSP receba a mensagem.
   a) Identifique, no início, o método com request.getMethod() (retorna POST ou GET, em maiúsculas).
   b) Se o método for POST, mostre, em vermelho, antes da exibição do formulário, o texto: "x graus F = y graus C" onde x é o valor digitado pelo usuário e y é a resposta.

Exercícios

4. Escreva uma JSP simples usando objeto de sessão
   a) Escreva uma página JSP novaMensagem.jsp que mostre formulário na tela com dois campos: email e mensagem.
   b) Escreva uma outra página gravarMensagem.jsp que receba parâmetros: email e mensagem e grave esses dois parâmetros na sessão do usuário.
   c) Faça com que a primeira página aponte para a segunda.
   d) Crie uma terceira página listarMensagens.jsp que mostre mensagens criadas até o momento.
5. Altere o exercício anterior fazendo com que
   a) A página gravarMensagem.jsp mostre todas as mensagens da sessão como resposta, mas grave a mensagem em disco usando parâmetro de inicialização do web.xml.
   b) A página listarMensagens.jsp liste todas as mensagens em disco.
   c) Obs: garanta uma gravação thread-safe para os dados.

Usando beans

- JavaBeans são objetos escritos de acordo com um determinado padrão que permite tratá-los como componentes de um framework
  - Ótimos para separar os detalhes de implementação de uma aplicação de seus “serviços”
  - Permitem encapsular dados da aplicação e torná-los disponíveis para alteração e leitura através de uma interface uniforme
- Podem ser usados com JSP para remover grande parte do código Java de uma página JSP
  - Maior facilidade de manutenção e depuração
  - Separação de responsabilidade e reuso de componentes

Como Incluir um bean

- Para que um bean possa ser usado por uma JSP, ele deve estar compilado e no CLASSPATH reconhecido pelo servidor
  - No subdiretório WEB-INF/classes do seu contexto
- Para incluir:

```
<jsp:useBean id="nome_da_referência"
             class="pacote.NomeDaClasse"
             scope="page|session|request|application" />
```

- O atributo de escopo é opcional e indica o tempo de vida do Java Bean. Se omitido, será page, que o limita à página
  - Com escopo de request, o bean pode ser recuperado com outra instrução <jsp:useBean> que esteja em outra página que receber a mesma requisição (via dispatcher)
  - Com escopo de session, o bean é recuperável em páginas usadas pelo mesmo cliente, desde que <%@page> não tenha session=false

Como Incluir um bean

- O nome do bean (atributo id) comporta-se como uma referência a um objeto Java
- Incluir o tag

```
<jsp:useBean id="bean" class="bean.HelloBean" scope="request" />
```

é o mesmo que incluir na página

```
<% Object obj = request.getAttribute("bean");
   bean.HelloBean bean = null;
   if (obj == null) {
       bean = new bean.HelloBean();
       request.setAttribute("bean", bean);
   } else {
       bean = (bean.HelloBean) obj;
   } %>
```

- O id pode ser usado em *scriptlets* para usar membros do bean

Propriedades

- JavaBeans possuem propriedades que podem ser somente leitura ou leitura-alteração.
- O nome da propriedade é sempre derivada do nome do método getXXX():

```
public class Bean {
    private String mensagem;
    public void setTexto(String x) {
        mensagem = x;
    }
    public String getTexto() {
        return mensagem;
    }
}
```

- O bean acima tem uma propriedade (RW) chamada texto

Propriedades

- Páginas JSP podem ler ou alterar propriedades de um bean usando os tags

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

- que equivale a `<% bean.setPropriedade(valor); %>` e

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

- que equivale a `<%=bean.getPropriedade() %>`
- Observe que o nome do bean é passado através do atributo name, que corresponde ao atributo id em <jsp:useBean>
- Valores são convertidos de e para String automaticamente
- Parâmetros HTTP com mesmo nome que as propriedades têm valores passados automaticamente com <jsp:setProperty>
  - Se não tiverem, pode-se usar atributo param de <jsp:setProperty>
  - <jsp:setProperty ... property="*"/> lê todos os parâmetros

Inicialização de beans

- A tag <jsp:useBean> cria um bean chamando seu construtor. Para inicializá-lo, é preciso chamar seus métodos setXXX() ou <jsp:setProperty>

```
<jsp:useBean id="bean" class="bean.HelloBean" />
<jsp:setProperty name="bean" property="prop" value="valor"/>
```

- Se um bean já existe, porém, geralmente não se deseja inicializá-lo.
- Neste caso, a inicialização pode ser feita dentro do marcador <jsp:useBean> e o sistema só a executará se o bean for novo

```
<jsp:useBean id="bean" class="bean.HelloBean" />
<jsp:setProperty name="bean" property="prop" value="valor"/>
</jsp:useBean>
```

- ou

```
<jsp:useBean id="bean" class="bean.HelloBean" />
<% bean.setProp(valor); %>
</jsp:useBean>
```

Condicionais e Iterações

- Não é possível usar beans para remover de páginas Web o código Java de expressões condicionais e iterações
  - Para isto, não há tags padrão. É preciso usar Taglibs (JSTL)
- Beans podem ser usados dentro de iterações e condicionais, e ter seus valores alterados a cada repetição ou condição

```
<jsp:useBean id="mBean" class="MessageBean" scope="session" />
<% MessageBean[] messages = MessagesCollection.getAll();
   for (int i = messages.length -1; i >= 0; i--) {
   mBean = messages[i];
%>
   <tr><td><jsp:getProperty name="mBean" property="time"/></td>
   <td><%=mBean.getHost()%></td>
   <td><%=mBean.getMessage()%></td></tr>
<% } %>
```

Matando beans

- Beans são associados a algum objeto de escopo: page, request, session ou application
  - Persistem até que o escopo termine ou expirem devido a um timeout (no caso de sessões)
- Para se livrar de beans persistentes, use os métodos removeAttribute(), disponíveis para cada objeto de escopo:

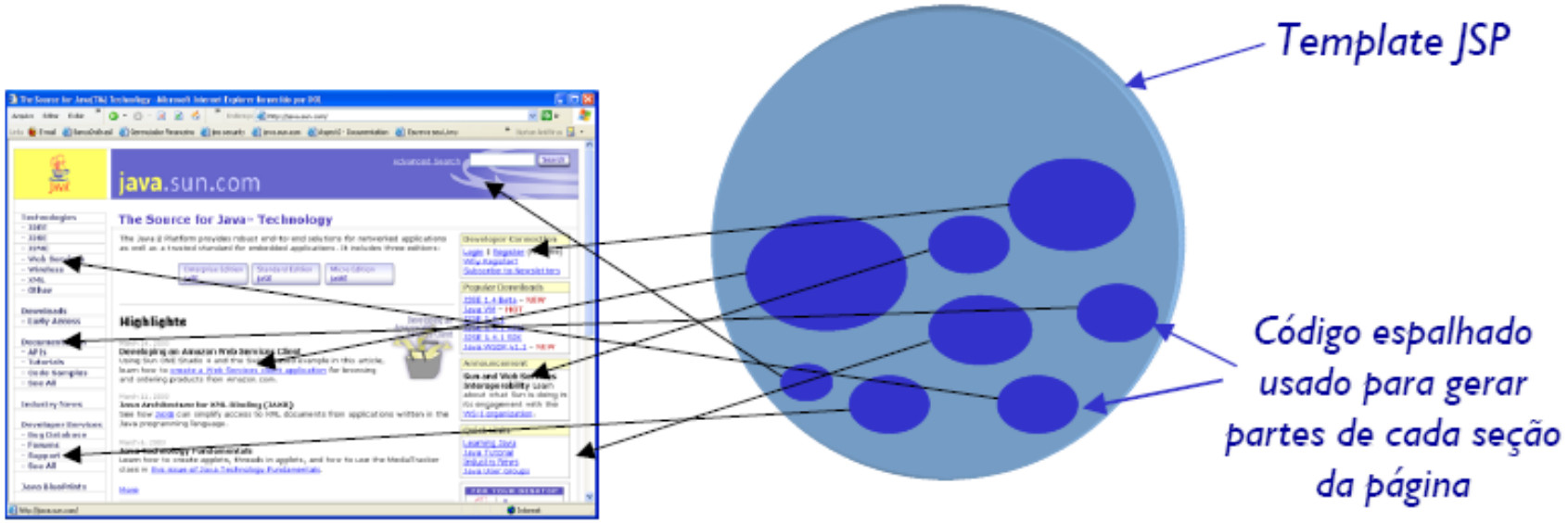
```
session.removeAttribute(bean);
application.removeAttribute(bean);
request.removeAttribute(bean);
```

Composite View

- Páginas Web complexas frequentemente são divididas em partes independentes
  - Algumas partes são altamente dinâmicas, mudando frequentemente até na estrutura interna
  - Outras partes mudam apenas o conteúdo
  - Outras partes sequer mudam o conteúdo

Composite View

- Gerar uma página dessas usando apenas um template é indesejável

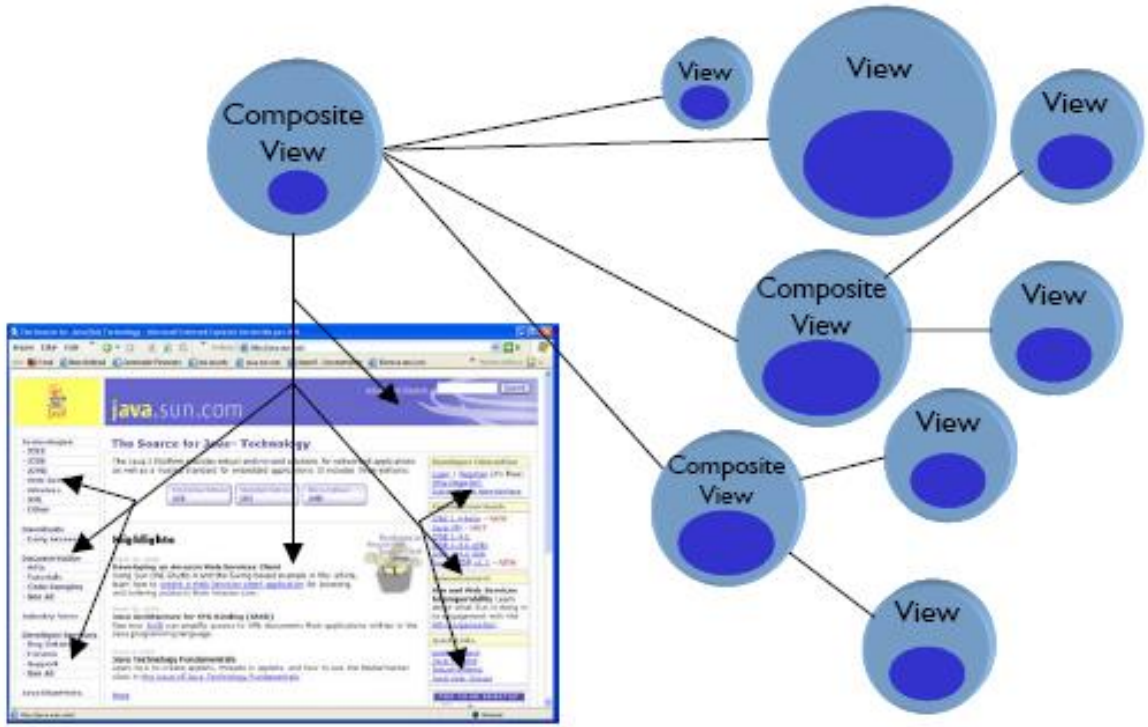


Composite View

- O padrão de projeto Composite View sugere que tais páginas sejam separadas em blocos menores
  - que possam ser alterados individualmente e
  - compostos durante a publicação (deployment) ou exibição

Composite View

- JSP oferece duas soluções para obter esse efeito:
  - Usando inclusão estática (no momento da compilação do servlet)
  - Usando inclusão dinâmica (no momento da requisição)



Inclusão Estática

- Mais eficiente: fragmentos são incluídos em único servlet
- Indicada quando estrutura não muda com frequência
  - Menus, Logotipos e Avisos de copyright
  - Telas com mini-formulários de busca
- Implementada com <%@ include file="fragmento" %>

```
<!-- Menu superior -->
<table>
<tr><td><%@ include file="menu.jsp" %></td></tr>
</table>
<!-- Fim do menu superior -->
```

Fragmento menu.jsp

```
<a href="link1">Item 1</a></td>
<td><a href="link2">Item 2</a></td>
<a href="link3">Item 3</a>
```

- Se tela incluída contiver novos fragmentos, eles serão processados recursivamente

Inclusão Dinâmica

- Mais lento: fragmentos não são incluídos no servlet mas carregados no momento da requisição
- Indicada para blocos cuja estrutura muda com frequência
  - Bloco central ou notícias de um portal
- Implementada com <jsp:include page="fragmento"/>
- Podem ser passados parâmetros usando <jsp:param>

```
<!-- Texto principal -->
<table>
<tr><td>
<jsp:include page="texto.jsp">
    <jsp:param name="data" value="<%=new Date() %>">
</jsp:include>
</td></tr> </table>
<!-- Fim do texto principal -->
```

Repasse de Requisições

- Uma requisição pode ser repassada de uma página JSP para outra página ou servlet usando RequestDispatcher

```
<% RequestDispatcher rd =
   request.getRequestDispatcher("url");
   rd.forward(request, response); %>
```

- O mesmo efeito é possível sem usar scriptlets com a ação padrão <jsp:forward>
- Assim como <jsp:include>, pode incluir parâmetros recuperáveis na página que receber a requisição usando request.getParameter() ou <jsp:getProperty> se houver bean

```
<% if (nome != null) { %>
<jsp:forward page="segunda.jsp">
    <jsp:param name="nome" value="<%=nome %>">
</jsp:forward>
<% } %>
```

JSP Standard Tag Library (JSTL)

- Esforço de padronização do JCP: JSR-152
  - Baseado no Jakarta Taglibs (porém bem menor)
- Oferece dois recursos
  - Conjunto padrão de tags básicos (Core, XML, banco de dados e internacionalização)
  - Linguagem de expressões do JSP 1.3
- Oferece mais controle ao autor de páginas sem necessariamente aumentar a complexidade
  - Controle sobre dados sem precisar escrever scripts
  - Estimula a separação da apresentação e lógica
  - Estimula o investimento em soluções MVC

Como usar JSTL

- 1. Fazer o download da última versão
- 2. Copiar os JARs das bibliotecas desejadas para o diretório WEB-INF/lib/ da sua aplicação Web

```
<%@ taglib uri="uri_da_taglib"
prefix="prefixo" %>
```

- 3. Incluir em cada página que usa os tags
- 4. Usar os tags da biblioteca com o prefixo definido no passo anterior

```
<prefixo:nomeTag atributo="..."> ...
</prefixo:nomeTag>
```

Cinco Bibliotecas de tags

- Core library: tags para condicionais, iterações, urls, ...
  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
  - Exemplo: <c:if test="..." ... >...</c:if>
- XML library: tags para processamento XML
  <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
  - Exemplo: <x:parse>...</x:parse>
- Internationalization library
  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
  - Exemplo: <fmt:message key="..." />
- SQL library
  <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
  - Exemplo: <sql:update>...</sql:update>
- Function library
  <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
  - Exemplo: ${fn:length(...)}

Linguagem de Expressões

- Permite embutir em atributos expressões dentro de delimitadores ${...}
  - Em vez de request.getAttribute("nome")

    ${nome}
  - Em vez de bean.getPessoa().getNome()

    ${bean.pessoa.nome}
- Suporta operadores aritméticos, relacionais e binários
- Converte tipos automaticamente

  <tag item="${request.valorNumerico}" />
- Valores default

  <tag value="${abc.def}" default="todos" />

Exemplo de JSTL

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
   <head>Exemplo de Uso de tags JSP</head>
   <body>
      <h1>Lista de Codinomes</h1>
      <table border="1">
         <c:set var="nomes"
                value="Bazílio, Instrutor, Carlos, Fulano" scope="page"/>
         <tr><th>Codinome</th><th>Preferência</th></tr>
         <c:forEach var="codinome" items="${nomes}”varStatus="status">
            <tr><td><c:out value="${codinome}"/></td>
             <td><c:out value="${status.count}"/></td></tr>
         </c:forEach>
      </table>
   </body>
</html>
```

JSTL – Core

- Ações Básicas Gerais
  - <c:out value="expressão" />
    - Semelhante a tag de expressão JSP
    - O conteúdo pode ser estático ou dinâmico (composto de EL)
  - <c:set var="nome_da_variável" value="expressão" [scope="page | request | session | application"] />
    - Define uma variável com o valor da expressão fornecida em algum escopo
    - O escopo default é o page
  - <c:catch [var="nome_da_exceção"] />
    - Utilizada para capturar exceções geradas pelo código JSP englobado
    - Estas exceções não são tratadas pelo mecanismo de página de erros de JSP
    - var armazenará o conteúdo da exceção gerada

JSTL – Core

- Ações Básicas Condicionais
  - <c:if test="condição_de_teste" [var="nome_da_variável"] [scope="page | request | session | application"] />
    - Corresponde ao if das linguagens de programação
    - Esta ação pode ter o valor do seu teste armazenado na variável var, do escopo fornecido
  - <c:choose />
    - Corresponde ao switch, para agrupamento de sentenças case;
  - <c:when test="condição_de_teste" />
    - Corresponde à uma sentença case
  - <c:otherwise />
    - Ação padrão a ser executada quando nenhuma ação when for tomada

JSTL – Core

- Ações Básicas de Iteração
  - <c:forEach items="collection" [var="nome_da_variável"] [varStatus="nome_da_variável_status"] [begin="início"] [end="fim"] [step="passo"] />
    - Corresponde ao for das linguagens de programação
    - Itera sobre coleções em geral
    - varStatus contém informações sobre o índice numa iteração
  - <c:forTokens items="string_tokens" delims="delimitadores" [var="nome_da_variável"] [varStatus="nome_da_variável_status"] [begin="início"] [end="fim"] [step="passo"] />
    - Análoga a classe StringTokenizer
    - A string fornecida é subdividida em tokens de acordo com os delimitadores fornecidos

JSTL – Core

- Ações Básicas de Iteração
  - <c:import url="recurso" [context="contexto"] [var="variável"] [scope="escopo"] [charEncoding="codificação"] />
    - Análogo à ação padrão <jsp:include />
    - Entretanto, permite a utilização de urls absolutas; ou seja, conteúdo fora da própria aplicação
  - <c:param name="nome_parâmetro" value="valor_parâmetro" />
    - Utilizado para criação de parâmetros de solicitação para URLs pelas ações c:import, c:url e c:redirect;
  - <c:redirect url="recurso" [context="contexto"]/>
    - Necessário quando se deseja abortar o processamento da JSP corrente; Frequentemente utilizado em situações de erro
  - <c:url />
    - Útil na regravação de URL com ID de sessão do cliente
    - Necessário quando cookies estão desabilitados

JSTL – XML

- Ações Básicas
  - Expressões de manipulação que utilizam a linguagem XPath
    - Exemplos:
      - $variable : procura por um atributo chamado variable no contexto de página. Equivale a pageContext.findAttribute(“variable”)
      - $applicationScope$variable : procura por um atributo chamado variable no contexto de aplicação (idem para sessionScope, pageScope, requestScope e param)
  - <x:parse xml=“documento_XML” var=“nome_variável” [scope=“escopo”] />
    - Percorre um documento XML
    - Não faz validação frente a esquemas ou DTDs
    - O atributo xml deverá conter uma variável, cujo conteúdo foi importado através de uma url de um xml (<c:import/>)

JSTL – XML

- Ações Básicas
  - <x:out select="expressão_xpath" [escapeXML="{true | false}"] />
    - Após a chamada da ação parse, está ação permite a extração de partes do documento XML à partir de expressões XPath
  - <x:set var="nome_variável" select="expressão_xpath" [scope="escopo"] />
    - Permite armazenar o resultado de uma consulta XPath para futura utilização
    - A variável utilizada pode ser definida em qualquer dos escopos vistos anteriormente
- Ações de Fluxo de Controle
  - <x:if />, <x:choose /> <x:when /> <x:otherwise /> <x:forEach />
  - Similares às tags da biblioteca básica
  - Atributo chamado select é utilizado sempre que referenciamos o documento XML

JSTL

- Internacionalização (I18N)
  - Oferece funções de internacionalização (localidade) e formatação de datas e números;
- Banco de Dados
  - Tags para consultas a banco de dados;
  - Naturalmente, só devem ser utilizadas em protótipos ou aplicações simples;
  - Para aplicações corporativas, esses dados devem estar encapsulados em componentes JavaBeans;
- Funções
  - Basicamente oferece funções para a manipulação de strings;

Exercícios

1. Use um JavaBean Mensagem, com propriedades email e mensagem para implementar a aplicação de Mensagen da aula anterior
   a) Substitua todas as chamadas de new Mensagem() por <jsp:useBean> no escopo da sessão
   b) Use Expressões para exibir os dados
2. Altere gravarMensagens para que use <jsp:forward> para despachar a requisição para uma página erro.jsp, caso o usuário deixe os campos do formulário em branco, e para listarMensagens.jsp se tudo funcionar corretamente
3. Instalar tags do JSTL
   a) Veja a documentação e os tags disponíveis
4. Use os tags de lógica <c:if> e <c:forEach> para remover as expressões condicionais e iterações das páginas da aplicação de mensagens

Segurança

- Não depende só da aplicação, envolve:
  - Servidor Web
  - Sistema Operacional
  - Rede
  - Pessoal ligado à aplicação direta ou indiretamente
  - Orçamento
  - Necessidade

Segurança

- Na prática são usados quatro métodos:
  1. Nas páginas que só devam ser acessadas após o login do usuário, informação na sessão
  2. Verificar de onde o usuário veio e só permitir o acesso a partir de elementos do próprio site
  3. Verificar código malicioso nos campos onde o usuário insere informações
  4. Verificar variáveis que compõe instrução SQL antes de enviá-la ao SGBD

Segurança

- Implementando o 1º método (seguranca.inc):

```
<%
if(session.getAttribute("codFunc") == null) {
    response.sendRedirect("index.html");
} else if(session.getAttribute("codFunc").equals("")) {
    response.sendRedirect("index.html");
}
%>
```

Segurança

- Implementando o 2º método:
  - Geralmente usa-se uma variável de servidor chamada HTTP_REFER
  - Compara-se o HTTP_REFER com a URL do seu site
  - Se a parte padrão bater, então o acesso é válido
  - Caso contrário, redirecione o usuário para a página inicial

Segurança

- Implementando o 3º método:
  - Nos campos recebidos com informações do usuário, substitua:
    - “<Script” por “<!--”;
    - “</Script” por “-->”;

Segurança

- Implementando o 4º método:
  - Verifique se variáveis string possuem aspa simples e substitua por duas aspas simples ou pelo correspondente ASCII
  - Verifique se variáveis string possuem sinais interpretados pelo SGBD como comentário
  - Verifique se o tipo das variáveis a serem inseridas no SGBD correspondem aos tipos declarados na tabela
  - Verifique se o tamanho das variáveis correspondem aos tamanhos declarados na criação das tabelas

Segurança

- Outras opções:
  - Criptografia;
  - Buffer Overflow;
  - Acesso de escrita e de leitura aos diretórios que compõem a página;

  ... entre outros!