Java 2 Enterprise Edition

JavaServer Pages

Helder da Rocha
www.argonavis.com.br

Sumário

- Introdução
  - Ciclo de vida de páginas JSP
- Geração de conteúdo
  - Geração de conteúdo estático
  - Diretivas e elementos para scripting
  - Objetos implícitos
  - Inicialização e finalização
- Comunicação com outros componentes
  - Transferência de controle
  - Inclusão de applets
- Componentes JavaBeans em páginas JSP
  - Convenções de design para componentes JavaBeans
  - Usando JavaBeans em JSP

Servlet design pattern: templates

- *Para imprimir HTML a partir de um servlet, pode-se*
  - *Imprimir o HTML através de várias instruções println()*

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

HojeServlet.java

  - *ou, criar páginas HTML contendo trechos para substituição (templates) preenchidos durante a leitura*

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

template.html

```
Date hoje = new Date();
String pagina = abreHTML("template.html");
Pattern p = Pattern.compile("<!--#data#-->");
Matcher m = p.matcher(pagina);
m.replaceAll(hoje);
```

HojeServlet.java

JavaServer Pages

- JSP é uma tecnologia baseada em templates para servlets padrão da Sun
- Há outras alternativas populares
  - Cocoon XSP: baseado em XML (xml.apache.org/cocoon)
  - Jakarta Velocity (jakarta.apache.org/velocity)
  - WebMacro (www.webmacro.org)
- Solução do problema anterior usando 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 localiza a própria página (default)

Exemplos de JSP

- *A forma mais simples de criar documentos JSP, é*
  - *mudar a extensão do arquivo HTML para .jsp*
  - *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*
- *Tendo sido transformado em um JSP, o arquivo HTML pode conter blocos de código (scriptlets):* `<% ... %>` *e expressões* `<%= ... %>` *que são os elementos mais frequentemente usados*

```
<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="localeForm" 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>
```

diretiva

bean

scriptlet

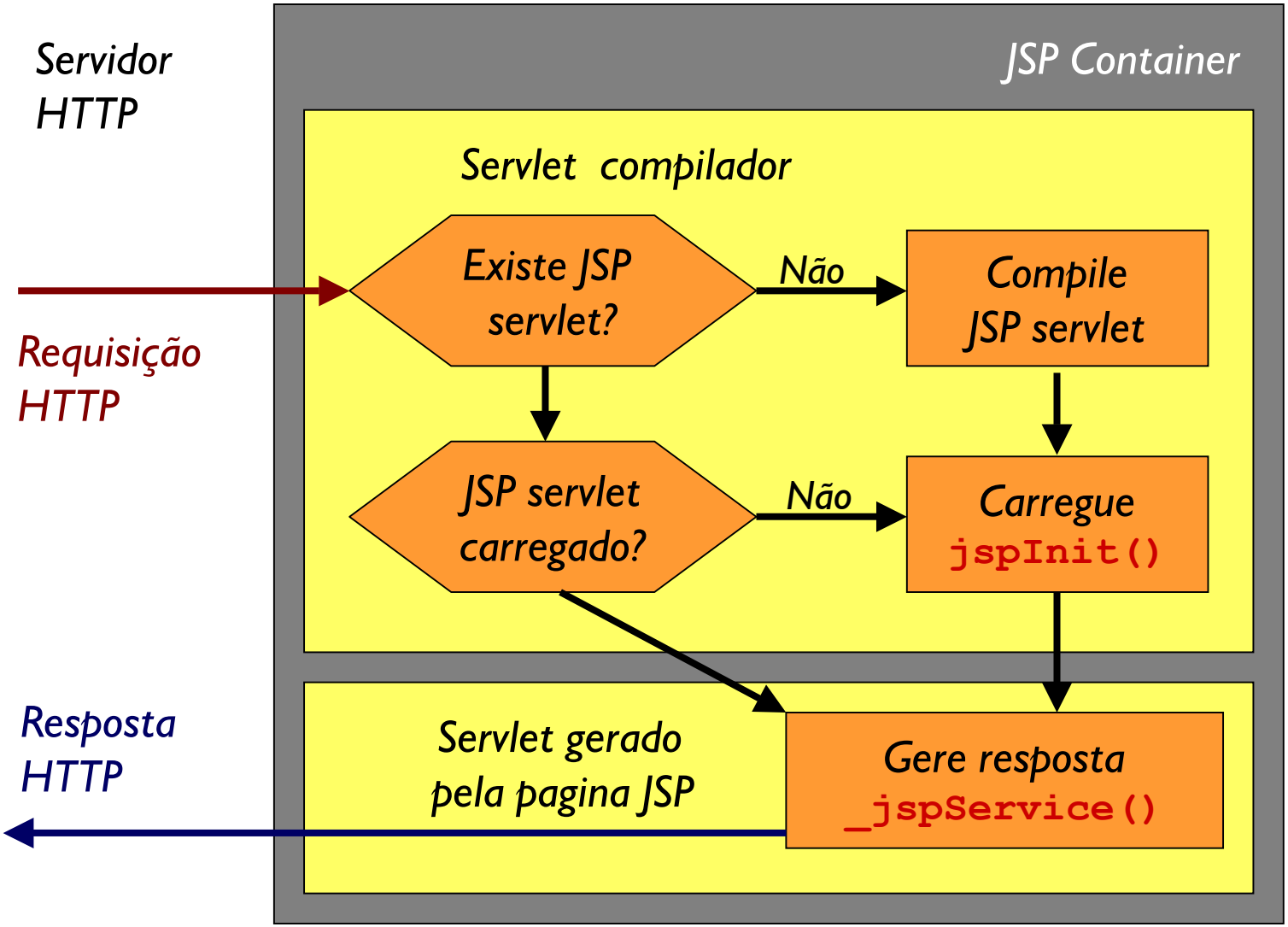
expressão

ação

Ciclo de vida

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

Como funciona JSP



Sintaxe dos elementos JSP

- *Podem ser usados em documentos de texto (geralmente HTML ou XML)*
- *Todos são interpretados no servidor (jamais chegam ao browser)*
  - *diretivas:* `<%@ ... %>`
  - *declarações:* `<%! ... %>`
  - *expressões:* `<%= ... %>`
  - *scriptlets:* `<% ... %>`
  - *comentários:* `<%-- ... --%>`
  - *ações:* `<jsp:ação ... />`
  - *custom tags:* `<prefixo:elemento ... />`
- *Há sintaxe alternativa XML para todos os elementos*

(a) diretivas

- *Contém informações necessárias ao processamento da página JSP (geralmente usadas no início do documento)*
- *Sintaxe :*

```
<%@ diretiva atrib1 atrib2 ... %>
```

- *Principais diretivas:*
  - **page**: *atributos relacionados à página*
  - **include**: *inclui outros arquivos na página*
  - **taglib**: *declara biblioteca de custom tags usada no documento*
- *Exemplos*

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

(a) diretiva page

- *Atributos de* `<%@page ... %>`

`info="Texto informativo"`	*default: nenhum*
`language="java"`	*(default)*
`contentType="text/html; charset=ISO-8859-1"`	*(default)*
`extends="acme.FonteJsp"`	*default: nenhum*
`import="java.io.*, java.net.*"`	*default: nenhum*
`session="true"`	*(default)*
`buffer="8kb"`	*(default)*
`autoFlush="true"`	*(default)*
`isThreadSafe="true"`	*(default)*
`errorPage="/erros/404.jsp"`	*default: nenhum*
`isErrorPage= "false"`	*(default)*

Alguns atributos de @page

- *session*
  - *Aplicações JSP podem manter sessões do usuário abertas usando HttpSession*
  - *Se uma página declara session=false, não tem acesso a objetos gravados na sessão do usuário*
- *isThreadSafe*
  - *Se true, só um cliente poderá acessar a página ao mesmo tempo*
- *isErrorPage*
  - *Se true, a página possui um objeto exception e pode extrair seus dados quando alvo de redirecionamento devido a erro*
- *errorPage*
  - *URL da página para o qual o controle será redirecionado na ocorrência de um erro ou exceção*

Atributos de @page: buffer e autoflush

- Você pode redirecionar, criar um cookie ou modificar o tipo de dados gerado por um programa JSP em qualquer parte da página.
  - 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 que o programador escolha onde e quando definir informações de cabeçalho
  - O buffer pode ser redefinido por página (diretiva page buffer)
  - autoFlush determina se dados serão enviados quando buffer encher ou se o programa lançará uma exceção.

(b) 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 static, etc.*
- *Sintaxe*

```
<%! declaração %>
```

- *Exemplos*

```
<%! public final static String[] meses =
      {"jan", "fev", "mar", "abr", "mai", "jun"};

    public static String getMes() {
        return meses[(new Date()).getMonth()];
    }
%>
```

(b) declarações (métodos especiais)

- *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, etc.*
- *Inicialização da página (chamado uma vez, antes da primeira requisição, após o instanciamento do servlet)*

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

- *Destruição da página (ocorre quando o servlet deixa a memória)*

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

(c) expressões e (d) scriptlets

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

```
<%= expressão %>
```

- *Todos os valores resultantes das expressões são convertidos em String antes de serem redirecionados à saída padrão*
- ***Scriptlets***: *Blocos de código que são executados sempre que uma página JSP é processada*
- *Permite inserir seqüências de instruções na página (dentro do método _jspService() do servlet correspondente)*
- *Sintaxe:*

```
<% instruções Java %>
```

(e) comentários

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

```
<!-- texto ignorado pelo browser mas não pelo
  servidor -->
```

- *Comentários JSP: podem ser usados para comentar blocos JSP - Sintaxe:*

```
<%-- texto, código Java, HTML ou marcadores
JSP ignorados pelo servidor --%>
```

- *Pode-se também usar comentários Java quando dentro de scriptlets, expressões ou declarações:*

```
<%  código JSP ... /* texto ignorado pelo
servidor */ ... mais código %>
```

(f) ações

- *Sintaxe:*

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

- *Permitem realizar operações (e meta-operações) externas ao servlet*
  - *concatenação de várias páginas em uma única resposta*
    <jsp:forward> e **<jsp:include>**
  - *inclusão de JavaBeans*
    <jsp:useBean>
    <jsp:setProperty>, **<jsp:getProperty>**
  - *operações definidas pelo programador (diretiva taglib)*

(f) ações (exemplos)

```
<%
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.
<% } %>
```

javax.servlet.jsp

- *Interfaces*
  - *JspPage - estende javax.servlet.Servlet*
  - *HttpJspPage - estende JspPage - base para páginas JSP HTTP*
- *Classes abstratas:*
  - *JspEngineInfo*
  - *JspFactory*
  - *JspWriter (estende java.io.Writer)*
  - *PageContext*
- *Classes concretas:*
  - *JspException*
  - *JspTagException (estende JspException)*

Objetos implícitos JSP

- Disponíveis em blocos <% . . . %> (scriptlets) de qualquer página (exceto session e exception que dependem de outros parâmetros da página)
- Objetos do servlet
  - page
  - config
- Entrada e saída
  - request
  - response
  - out
- Objetos contextuais
  - session
  - application
  - pageContext
- Controle de exceções
  - exception

(a) page

- *Referência para o servlet gerado pela página*
- *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:*

```
<% String param =
            page.getInitParameter() %>
```

(b) config

- *Referência para os parâmetros de inicialização do servlet (se existirem)*
- *Equivale a page.getServletConfig()*
- *Exemplo:*
  - ```
    <%! public void jspInit()  {
    String user =
            config.getInitParameter("nome");
    String pass =
            config.getInitParameter("pass");
    } %>
    ```
  - *parâmetros de inicialização são fornecidos na instalação do servlet no servidor, através do elemento <init-param> de <servlet>*
    ```

(c) request

- Referência para os dados de entrada enviados na requisição do cliente (no GET ou POST, por exemplo, em HTTP)
  - É um objeto do tipo javax.servlet.ServletRequest (ou javax.servlet.http.HttpServletRequest)
- Usado para
  - guardar variáveis que serão usadas enquanto durar a requisição (que pode consistir de mais de uma página)
  - recuperar parâmetros passados pelo cliente (dados de um formulário HTML, por exemplo)
  - recuperar cookies
- Principais métodos
  - getHeader(String) - recupera conteúdo de cabeçalho
  - getCookies() - recupera cookies da requisição (se houver)
  - getParameter(String) - recupera parâmetro de requisição

(c) exemplos

- *URL no browser:*

```
http://servidor/programa.jsp?nome=Fulano&id=5
```

- *Recuperação dos parâmetros no programa JSP:*

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

- *Ou para um bean com propriedades de mesmo nome (neste caso, não usamos o objeto request)*

```
<jsp:setProperty name="nome_bean"
                 property="nome"/>
<jsp:setProperty name="nome_bean"
                 property="id"/>
```

- *Cookies*

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

(d) response

- Referência aos dados de saída enviados na resposta do servidor enviada ao cliente
  - É um objeto do tipo javax.servlet.ServletResponse (ou javax.servlet.http.HttpServletResponse)
- Usado para
  - definir o tipo dos dados retornados, criar cookies, definir cabeçalhos de resposta, redirecionar, controlar cache
- Principais métodos
  - setHeader(String, String) - define novo cabeçalho
  - addCookie(Cookie) - cria um cookie
  - setContentType(String) - muda tipo de dados enviados
- Para criar cookies:
  - `Cookie c = new Cookie("nome", "valor");`
    `response.addCookie(c);`
- Gera cabeçalho:
  - `Set-Cookie: nome=valor`

(e) out

- *Representa o OutputStream da página (texto que compõe o HTML que chegará ao cliente).*
  - *É instância da classe javax.servlet.jsp.JspWriter (extensão 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 %>
<% } %>
```

(f) session

- *Representa a sessão do usuário (representada por uma coleção de requisições do mesmo endereço IP, durante um determinado tempo.)*
  - *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");
```

- *Principais métodos*
  - *getId() - retorna o código da sessão*
  - *get/setMaxInactiveInterval() - recupera/define o tempo entre requisições da mesma origem que caracteriza uma sessão.*

(g) application

- Representa a aplicação à qual a página pertence
  - a identificação de uma aplicação depende de configuração - (geralmente consiste das páginas abaixo de um determinado diretório)
  - Instância de javax.servlet.ServletContext
- Útil para guardar valores que devem persistir pelo tempo que durar a aplicação (até que o servlet seja descarregado do servidor)
- Exemplo

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

(h) pageContext

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

Escopo dos objetos

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis
- Constantes identificam escopo de objetos (classe javax.servlet.jsp.PageContext)
  - pageContext PageContext.PAGE_SCOPE
  - request PageContext.REQUEST_SCOPE
  - session PageContext.SESSION_SCOPE
  - application PageContext.APPLICATION_SCOPE
- Métodos de pageContext que lidam com escopo:
  - `setAttribute(nome, valor, escopo)`
  - `getAttribute(nome, escopo)`

(i) 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); %>
```

Usando beans

- JavaBeans são componentes reutilizáveis escritos em Java
  - Ótimos para separar os detalhes de implementação de uma aplicação de seus "serviços"
  - Permitem encapsular dados recebidos de outras partes da aplicação e torná-los disponíveis para alteração e leitura através de uma interface uniforme.
- Podem ser usados com JSP para remover praticamente todo o código Java de uma página JSP, deixando-o isolado em classes Java
  - Maior facilidade de manutenção
  - Maior reuso de componentes

Como incluir um bean

- *Para que um bean possa ser usado por uma aplicação JSP, ele deve estar compilado e localizado dentro do CLASSPATH reconhecido pelo servidor*
  - *No Jakarta-Tomcat, o subdiretório WEB-INF/classes da aplicação*
- *Para incluir:*

```
<jsp:useBean id="nome_da_referência"
             class="pacote.NomeDaClasse"
             scope="objeto_de_escopo">
</jsp:useBean>
```

  - *objeto_de_escopo pode ser request, session ou application*

Como incluir um bean

- O nome do bean (atributo id) comporta-se como uma referência a um objeto Java
  - Pode ser usado para chamar métodos do bean ou acessar suas variáveis públicas.

```
<%
    nome_bean.metodo(12, 34);
    int z = nome_bean.stat;
%>
```

- Se definido com escopo de sessão, só será instanciado uma vez
  - Será reutilizado (preservando o seu status) em cada página que contiver o tag <jsp:useBean> com bean de mesmo nome

Propriedades

- *Beans possuem propriedades que podem ser somente-leitura ou leitura-alteração.*
- *O nome da propriedade é 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 (read-write) chamada* **texto**

Propriedades

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

```
<jsp:setProperty name="nome_do_bean"
     property="nome_da_propriedade"
     value="valor_da_propriedade"/>
<jsp:getProperty name="nome_do_bean"
     property="nome_da_propriedade"/>
```

- *Observe que o nome do bean é passado através do atributo name, que corresponde ao atributo id no tag* `<jsp:useBean ...>`
- *Parâmetros HTTP que tenham o mesmo nome que as propriedades têm seus valores passados automaticamente para o bean.*

Matando beans

- *Para se livrar de beans persistentes, use os métodos disponíveis em cada objeto de escopo:*
  - `session.removeAttribute(nome_do_bean);`
  - `application.removeAttribute(nome_do_bean);`
  - `request.removeAttribute(nome_do_bean);`

helder@ibpinet.net

www.argonavis.com.br