

11 JavaServer Pages

Helder da Rocha
www.argonavis.com.br

- *Este módulo apresenta JavaServer Pages*
 - *Sintaxe dos marcadores JSP e objetos*
 - *Funcionamento*
 - *Como implantar*
- *Tudo o que vale para servlets continua valendo para JavaServer Pages*
 - *Um JSP é um servlet durante a execução*
 - *Escrever código em JSP é como escrever código dentro do doPost() ou doGet() de um servlet com os objetos response, request, out, session e outros já definidos*
 - *Um JSP, depois de carregado, é tão veloz quando um servlet*
 - *É mais fácil escrever e implantar, mas é mais difícil depurar*

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

Usando templates em servlets

- Tendo-se um template, é preciso criar um mecanismo eficiente para processá-los
 - No exemplo mostrado, pode-se ler o arquivo seqüencialmente , jogando tudo na saída até achar a seqüência "**<!--#**"
 - Depois, interpretar o "comando", gera-se o código relacionado com ele e prossegue-se na leitura e impressão do resto do documento
 - Há várias formas de implementar. O código abaixo usa o pacote **javax.util.regex** para localizar os comandos e fazer a substituição

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

HojeServlet.java

- Com o tempo, define-se um **vocabulário** e procura-se fazer o processador de templates cada vez mais **reutilizável**

O que são JavaServer Pages (JSP)

- 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)
- 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 `<%= ... %>` 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>
```

diretivas

bean

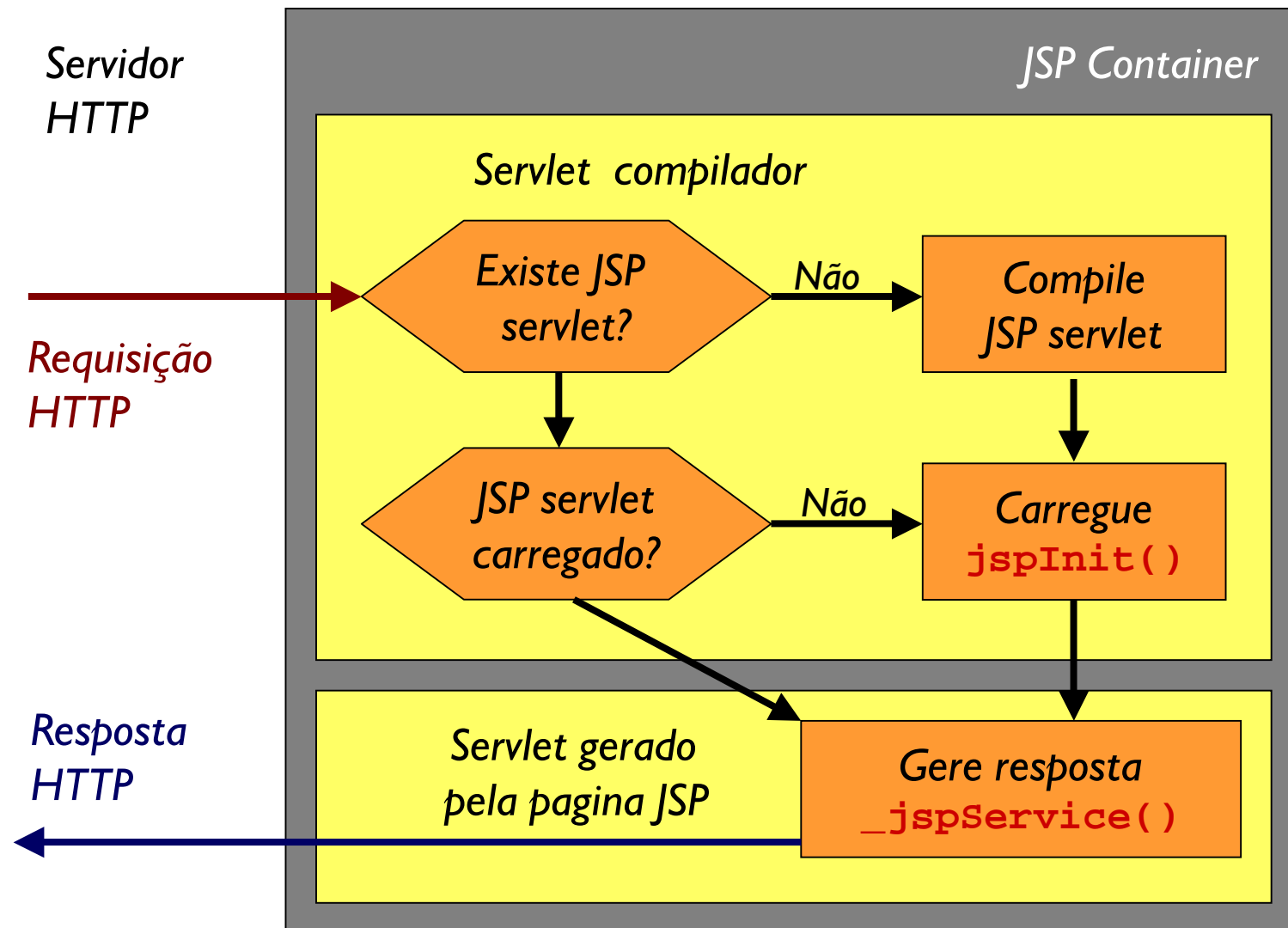
scriptlet

expressão

ação

- 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. Ele 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 ... />`

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

(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() {
    Calendar cal = new GregorianCalendar();
    return meses[cal.get(Calendar.MONTH)];
}
%>
```

(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 do web.xml, 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 %>`
- Equivale a `out.print(expressão)`, portanto, **não pode** terminar em ponto-e-vírgula
 - 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
- Correspondem a inserção de seqüências de instruções no método `_jspService()` do servlet gerado
- Sintaxe:
`<% instruções Java; %>`

(e) comentários

- **Comentários HTML** `<!-- -->` não servem para comentar JSP
`<!--` Texto ignorado pelo browser mas não pelo servidor. Tags são processados `-->`
- **Comentários JSP**: podem ser usados para comentar blocos JSP
`<%--` Texto, código Java, `<HTML>` ou tags `<%JSP%>` ignorados pelo servidor `--%>`
- Pode-se também usar comentários Java quando dentro de *scriptlets*, expressões ou declarações:
`<% código JSP ... /* texto ou comandos Java ignorados pelo servidor */ ... mais código %>`

(f) ações padronizadas

- *Sintaxe:*

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

- *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>
```

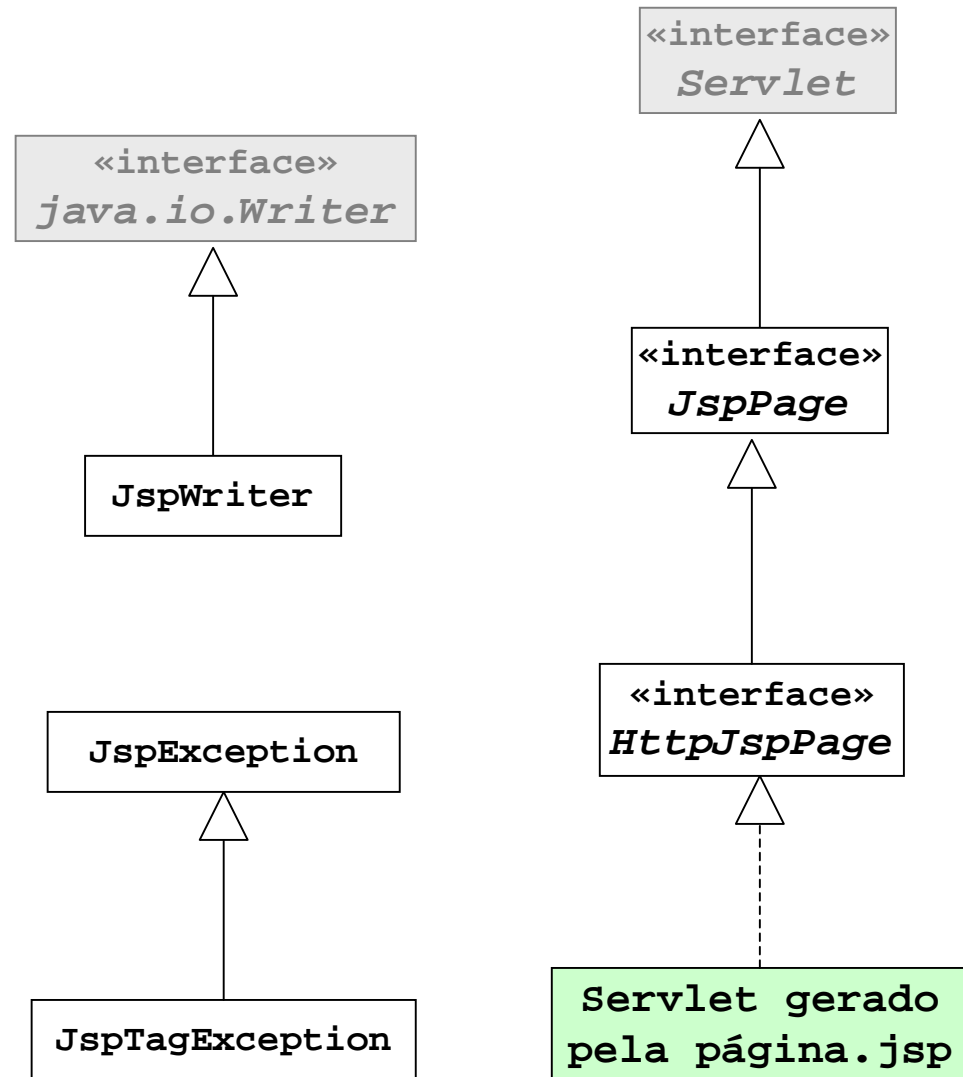
- *Geração de código HTML para Applets*

```
<jsp:plugin>
```

API: Classes de suporte a JSP

Pacote *javax.servlet.jsp*

- Interfaces
 - *JspPage*
 - *HttpJspPage*
- Classes abstratas:
 - *JspEngineInfo*
 - *JspFactory*
 - *JspWriter*
 - *PageContext*
- Classes concretas:
 - *JspException*
 - *JspTagException*



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

- 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() %>
```

- 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 fornecidos na instalação do servlet no servidor, através de `<init-param>` de `<servlet>` em `web.xml`. É preciso declarar a página 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>
```

(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.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)**

```
String method = request.getMethod();
```

- *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()
```

(d) 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/html)*
 - *Criar cookies*
`Cookie c = new Cookie("nome", "valor");`
`response.addCookie(c);`
 - *Definir cabeçalhos de resposta*
 - *Redirecionar*
`response.sendRedirect("pagina2.html");`

- Representa o stream de saída da página (texto que compõe o HTML que chegará ao cliente).

- 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 %>  
<% } %>
```

- 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");
```

(g) 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 (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
 - `pageContext.forward(String)` - mesmo que ação `<jsp:forward>`
 - `pageContext.include(String)` - mesmo que ação `<jsp:include>`

Escopo dos objetos

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis
- 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`
- Métodos de `pageContext` permitem setar ou buscar atributos em qualquer objeto de 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); %>
```

- *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 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 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 aplicação JSP, ele deve estar compilado e localizado dentro do 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">  
</jsp:useBean>
```
- 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

```
<% bean.setValor(12); %>
```

- *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>` simplesmente cria um bean chamando seu construtor. Para inicializá-lo, é preciso chamar seus métodos `setXXX()` ou usar `<jsp:setProperty>` após a definição
- 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 **novο** (se já existir, o código será ignorado)

```
<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>
```

Matando beans

- Beans são sempre gravados em 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);
```

- *Páginas complexas geralmente possuem diversas seções independentes*
 - *Menus*
 - *Corpo da página*
 - *Trechos altamente dinâmicos (notícias, etc.)*
- *Em vez de usar um arquivo único, uma página complexa pode ser formada a partir da composição de textos menores (Composite View pattern)*
- *JSP oferece duas soluções*
 - *Inclusão estática (no momento da compilação do servlet)*
 - *Inclusão dinâmica (no momento da requisição)*

Inclusão estática

- *Mais eficiente: fragmentos são incluídos em único servlet*
- *Indicada quando estrutura não muda com frequência (conteúdo pode mudar)*
 - *Menus, Logotipos e Avisos de copyright*
 - *Telas com miniformulários de busca*
- *Implementada com `<%@ include file="fragmento" %>`*

```
<!-- Menu superior -->
```

```
<table>
```

```
<tr><td><%@ include file="menu.jsp" %></td></tr>
```

```
</table>
```

```
<!-- Fim do menu superior -->
```

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

Fragmento menu.jsp

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"/>`*
- *Pode-se passar parâmetros em tempo de execução usando `<jsp:param>` no seu interior*

```
<!-- 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 -->
```

Tem precedência sobre parâmetros passados via requisição (aparece antes deles mas não sobrepõe)



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.dispatch(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>  
    <% } %>
```

- 1. Escreva um JSP *data.jsp* que imprima a data de hoje na tela do browser.
 - 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 fórmula é $F = 9/5 C + 32$

helder@argonavis.com.br

argonavis.com.br

*J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)*

© 1999-2003, Helder da Rocha