J550

JavaServer Pages

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- *Este módulo apresenta tudo o que é necessário para implementar servlets escrevendo JavaServer Pages*
  - *Sintaxe dos marcadores JSP e objetos*
  - *Funcionamento*
  - *Como implantar e depurar*
- *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

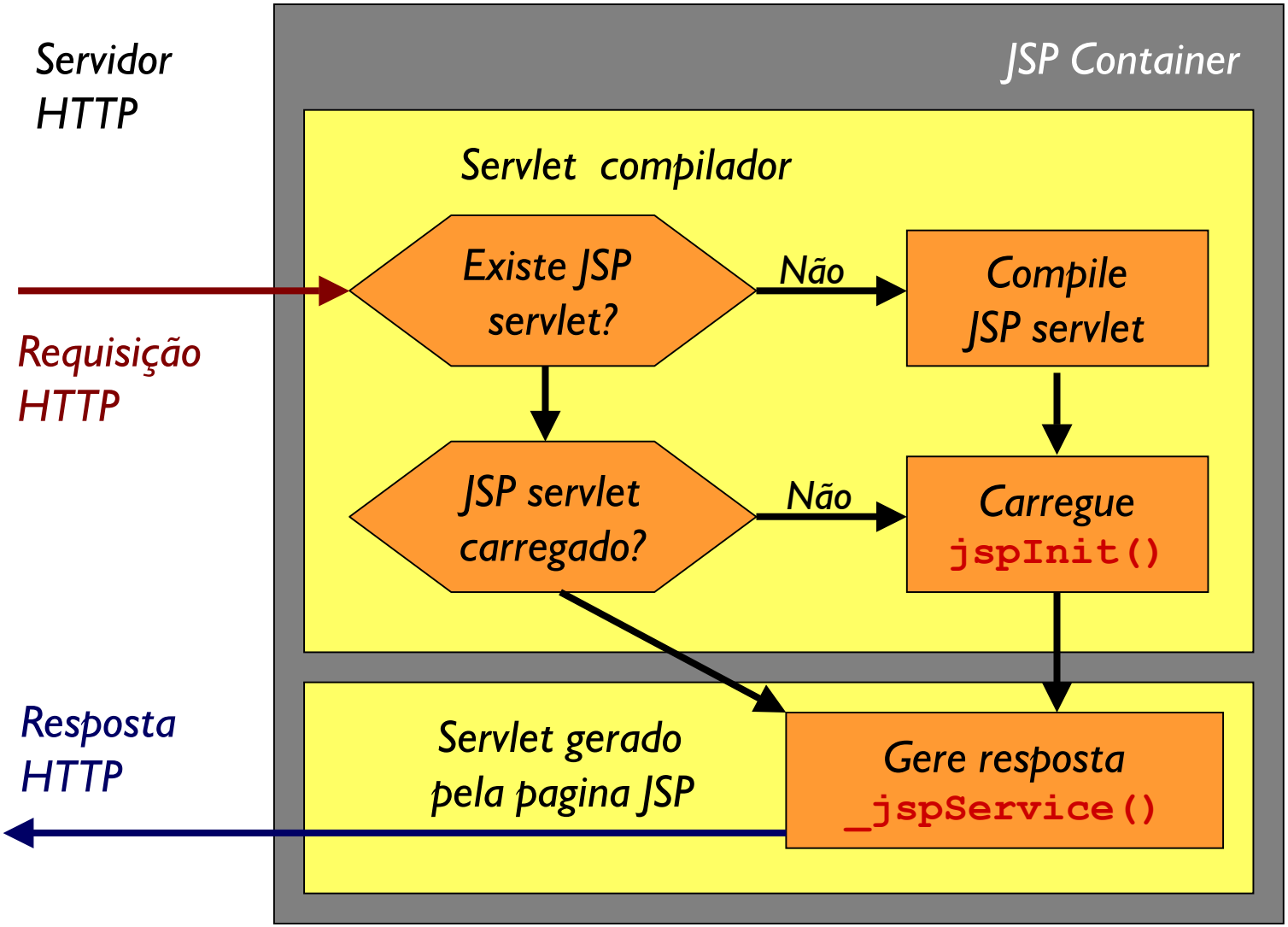
expressão

ação

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. 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"`	*default: nenhum*
`language="java"`	*(default)*
`contentType="text/html; charset=ISO-8859-1"`	*(default)*
`extends="acme.FonteJsp"`	*default: nenhum*
`import="java.io.*, java.net.*"`	*default: java.lang*
`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*
  - *Se true, aplicações JSP podem manter sessões do usuário abertas usando HttpSession*
  - *Se uma página declara session=false, ela não terá acesso a objetos gravados na sessão do usuário (objeto HttpSession)*
- *isThreadSafe*
  - *Se true, só um cliente poderá acessar a página ao mesmo tempo*
- *isErrorPage*
  - *Se true, a página possui um objeto exception (Throwable) e pode extrair seus dados quando alvo de redirecionamento devido a erro. Possui também os dois atributos padrão de páginas de erro.*
- *errorPage*
  - *URL da página para o qual o controle será redirecionado na ocorrência de um erro ou exceção. Deve ser uma página com isErrorPage=true.*

Atributos de @page: 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 que o programador escolha 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.

(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>`

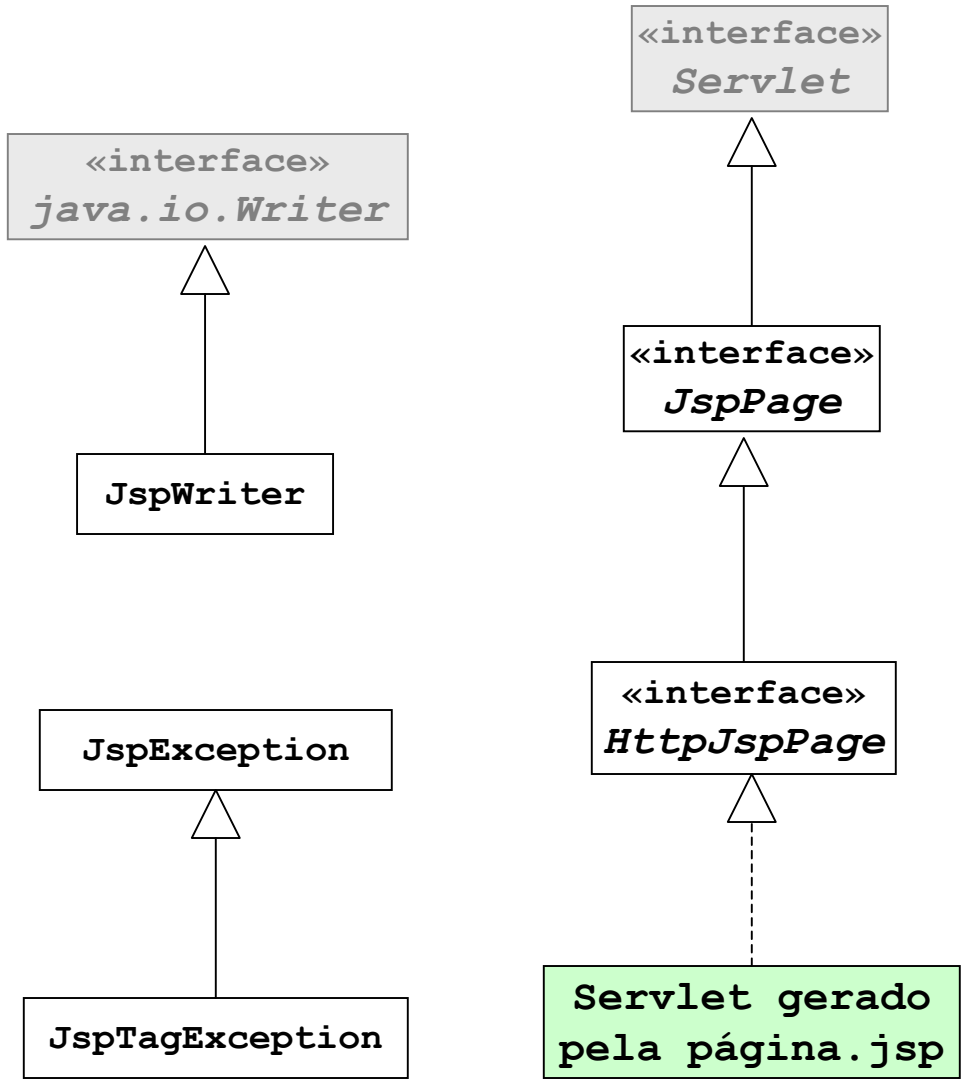
(f) ações (exemplos)

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

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

(a) 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() %>
```

(b) 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 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();
```

(c) 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()
```

(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/thml)
  - Criar cookies

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

  - Definir cabeçalhos de resposta
  - Redirecionar

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

(e) out

- *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 %>
<% } %>
```

(f) 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");
```

(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); %>
```

Depuração de JSP

- Apesar de ser muito mais fácil escrever JSP, a depuração não é simples
  - A página é compilada no servidor, exigindo que se procure por erros de compilação ou de parsing em uma página HTML remota
  - Nem sempre os erros são claros. Erros de parsing (um tag %> faltando, produzem mensagens esdrúxulas)
  - Os números das linhas, nas mensagens do Tomcat, não correspondem ao local do erro no JSP mas ao número da linha do código Java do servlet que foi gerado: você encontra o servlet no diretório work, do Tomcat
  - O servlet gerado pelo Tomcat não é fácil de entender, usa variáveis pouco explicativas e código repetido.

Servlet gerado de hello.jsp (do Cap 1)

- *Procure-o em $TOMCAT_HOME\work\Standalone\localhost_\hello$jsp.java*

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class hello$jsp extends HttpJspBase {
    static {    }
    public hello$jsp( ) {    }
    private static boolean _jspx_inited = false;
    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {
    }
    public void _jspService(HttpServletRequest request, HttpServletResponse  response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String  _value = null;
        try {
            if (_jspx_inited == false) {
                synchronized (this) {
                    if (_jspx_inited == false) {
                        _jspx_init();
                        _jspx_inited = true;
                    }
                }
            }
```

Continuação de hello.jsp

$TOMCAT_HOME\work\Standalone\localhost_\hello$jsp.java

```
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;ISO-8859-1");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                                                      "", true, 8192, true);
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            // HTML // begin [file="/hello.jsp";from=(0,0);to=(8,0)]
                out.write("<HTML><HEAD>\r\n<TITLE>Simple Servlet Output</TITLE>\r\n");
                out.write("</HEAD>\r\n\r\n<BODY>\r\n");

            // end
            // begin [file="/hello.jsp";from=(8,2);to=(12,0)]
                    String user = request.getParameter("usuario");
                    if (user == null)
                        user = "World";
            // end
            // HTML // begin [file="/hello.jsp";from=(12,2);to=(14,10)]
                out.write("\r\n<H1>Simple JSP Output</H1>\r\n<P>Hello, ");
            // end
            // begin [file="/hello.jsp";from=(14,13);to=(14,19)]
                out.print( user );
            // end
            // HTML // begin [file="/hello.jsp";from=(14,21);to=(17,0)]
                out.write("\r\n</BODY></HTML>\r\n\r\n");
            // end
        } catch (Throwable t) {
            if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();
            if (pageContext != null) pageContext.handlePageException(t);
        } finally {
            if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
        }
    }
}
```

Números de linha da página JSP

Código e HTML da página

Não precisa montar projeto. Simplesmente copie os arquivos para o contexto ROOT!

Exercícios

- *1. Escreva um JSP data.jsp que imprima a data de hoje.*
  - *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*
- *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.*

Faça estes arquivos parte de um contexto
(por exemplo: forumjsp)

Exercícios

- *4. JSP simples usando objeto de sessão*
  - *a. Escreva uma página JSP novaMensagem.jsp que mostre um formulário na tela com dois campos: email e mensagem.*
  - *b. Escreva uma outra página gravarMensagem.jsp que receba dois 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 todas as 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.*
  - *Obs: garanta uma gravação thread-safe para os dados.*

helder@acm.org

argonavis.com.br