ISEL

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia da Electrónica das Telecomunicações e dos Computadores

Infraestruturas Computacionais Distribuídas

WEB - Geração de Páginas dinamicamente - JSP's (JavaServer Pages)

Versão 1.1

Índice

A necessidade de JSP's

- **Com Servlet's é fácil:**
  - Ler os dados de formulários
  - Ler os cabeçalhos dos pedidos HTTP
  - Escrever o estado de retorno e os cabeçalhos da resposta
  - Utilizar Cookies e a API "*Session Tracking*"
  - Partilhar dados entre Servlet's
  - Guardar dados entre pedidos
- **Mas não é produtivo:**
  - Utilizar as instruções de "*println*" para gerar código HTML
  - Efectuar a manutenção desse mesmo HTML

O Framework das JSP's

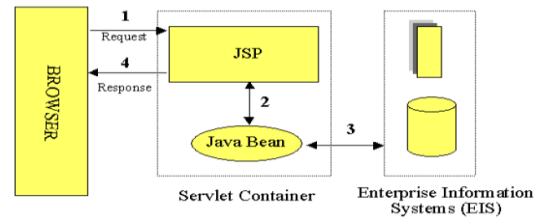
- **JavaServer Page – JSP**
  - É um molde (*template*) para páginas Web que utiliza código Java para gerar conteúdo dinâmico.
  - Tem subjacente a tecnologia das Java Servlet's.
  - São executadas no lado do Servidor (*server-side*).
- **Ideia:**
  - Utilizar o código HTML produzido por um editor de páginas web.
  - Embutir o código de servlet com etiquetas especiais.
  - Toda a JSP é traduzida para uma *servlet* (só na 1ª vez que é chamada) que irá tratar os pedidos relativos a esta página.
- **Exemplo** (Olá Mundo!):
  - JSP:
    - `<html><body> Olá <i><%=request.getParameter("nome")%> </i>!!! </body></html>`
  - URL:
    - `http://meuHost/olaMundo.jsp?nome=Manel`
  - Resultado:
    - Olá *Manel*!!!

Separar a Interface da Lógica

- **Embora as JSP's, tecnicamente, não possam fazer nada que as Servlet's não possam fazer, tornam mais simples:**
  - A escrita e manutenção da páginas HTML.
  - A leitura e manutenção das páginas, separando a lógica de processo do visual.
- **As JSP's tornam possíveis:**
  - Utilização de ferramentas de HTML standard como o FrontPage, Macromedia DreamWeaver, Allaire HomeSite, etc…
  - Ter membros da equipa a fazer a programação em HTML e outros a programação em JAVA.
- **As JSP's encorajam-nos a:**
  - Separar o código (Java), com a lógica da aplicação, do código de apresentação (HTML).

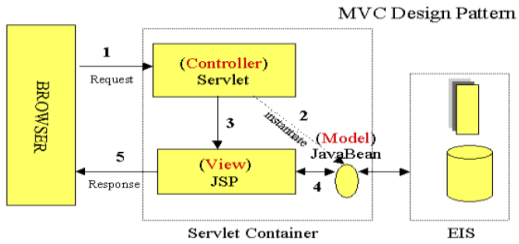
Modelo 3 Níveis

- Segundo esta abordagem a aplicação é dividida em três partes: interface com o utilizador, lógica da aplicação e a persistência dos dados.



Modelo-Vista-Controlador

- Este modelo é, tecnicamente, designado por *Model View Controller* (MVC); propõe uma arquitectura para a estruturação da aplicação web centrada nos três aspectos fundamentais: Modelo ou lógica da aplicação, controlo da execução e interface visual com o utilizador funcionando em ciclo fechado.



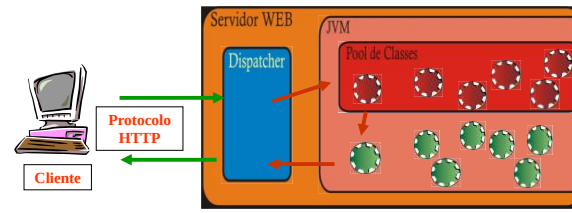
Vantagens das JSP's em relação a tecnologias concorrentes

- **Relativamente a ASP's (Active Server Pages)**
  - Tem uma linguagem melhor para a parte dinâmica
  - Portável para múltiplos servidores e sistemas operativos
- **Relativamente ao PHP**
  - Tem uma linguagem melhor para a parte dinâmica
  - Tem melhores ferramentas de suporte
    - Ambientes com maior capacidade de integração e *debug*
- **Relativamente a servlet's tradicionais**
  - É mais conveniente para criar HTML
  - Podemos utilizar ferramentas tradicionais (ex: FrontPage)
  - Dividir para conquistar
  - Os programadores de JSP's precisam de saber JAVA

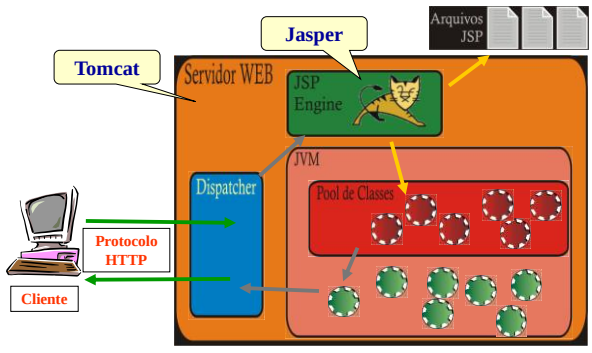
Vantagens das JSP's em relação a tecnologias concorrentes

- **Relativamente a JavaScript do lado do cliente (Browser)**
  - As capacidades normalmente não se sobrepõem, mas:
    - Controlamos o servidor, não o cliente
    - Linguagem mais rica
- **Relativamente a JavaScript do lado do servidor**
  - Linguagem mais rica
- **Relativamente a HTML estático**
  - Geração de conteúdo dinâmico
  - A adição de conteúdo dinâmico já não é uma decisão de "8 ou 80"

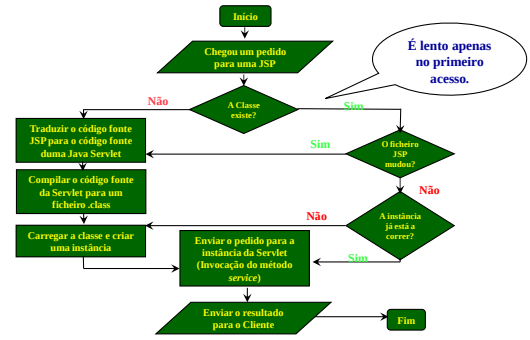
Ambiente executivo das Servlet's



Ambiente executivo das JSP's



Lógica utilizada pelo JSP container para gerir a tradução das JSP's



Ciclo de Vida

- Sendo que as JSP's são páginas que são traduzidas em Servlet's o ciclo de vida é o mesmo.
  - Init(), Service(), Destroy()
- As instancias das servlet's, resultantes da tradução de JSP's, ficam instanciadas até que o servidor seja desactivado ou até que ultrapassem um período grande de inactividade. Altura em que é invocado o método destroy()

Ciclo de Vida

- Como posso acrescentar código ao *init* e ao *destroy* da Servlet gerada?
  - Se redefinirmos, simplesmente, estes métodos temos problemas, pois a *Servlet* construída, automaticamente, já os utiliza.
  - Podemos redefinir os seguintes métodos, que serão activados pela *servlet* gerada:
    - `public void jspInit();`
    - `public void jspDestroy();`

A minha primeira JSP

my1rstJsp.jsp

```
<html>
<head></head>
<body>
<h1>A minha primeira JSP</h1>
<ul>
<li>Data: <% out.println(new java.util.Date()); %></li>
</ul>
</body>
</html>
```

No Tomcat é a Web Application: ROOT

A minha primeira JSP (2)

Ficheiro *my1rstJsp_jsp.java* criado automaticamente pelo *Jasper*.

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class my1rstJsp_jsp extends org.apache.jasper.runtime.HttpJspBase
                implements org.apache.jasper.runtime.JspSourceDependent {

  private static java.util.Vector _jspx_dependants;

  public java.util.List getDependants() {
    return _jspx_dependants;
  }

  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;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
...
```

O método _jspService() é invocado pela implementação do método service() da classe HttpJspBase

A minha primeira JSP (cont.)

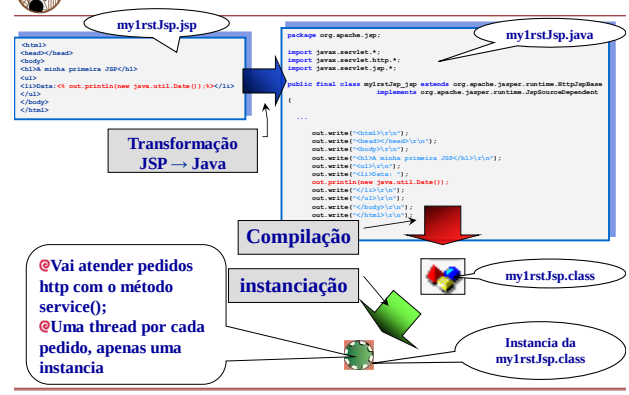
```
try {
  _jspxFactory = JspFactory.getDefaultFactory();
  response.setContentType("text/html");
  pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
  _jspx_page_context = pageContext;
  application = pageContext.getServletContext();
  config = pageContext.getServletConfig();
  session = pageContext.getSession();
  out = pageContext.getOut();
  _jspx_out = out;

  out.write("<html>\r\n");
  out.write("<head></head>\r\n");
  out.write("<body>\r\n");
  out.write("<h1>A minha primeira JSP</h1>\r\n");
  out.write("<ul>\r\n");
  out.write("<li>Data: ");
  out.println(new java.util.Date());
  out.write("</li>\r\n");
  out.write("</ul>\r\n");
  out.write("</body>\r\n");
  out.write("</html>\r\n");
} catch (Throwable t) {
  if (!(t instanceof SkipPageException)){
    out = _jspx_out;
    if (out != null && out.getBufferSize() != 0)
      out.clearBuffer();
    if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
  }
} finally {
  if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
}
}
```

Objectos implícitos sempre disponíveis nas JSP's

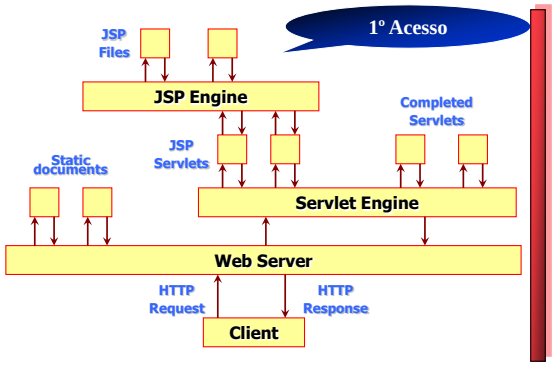
Impressão dos dados para o buffer de output para o cliente

Em caso de erro limpa o buffer para substituir pela informação de debug

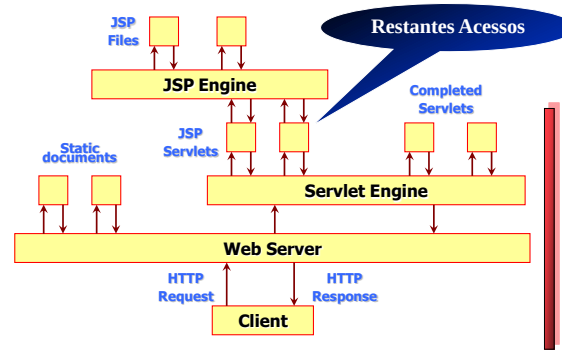
Transformação JSP → Servlet



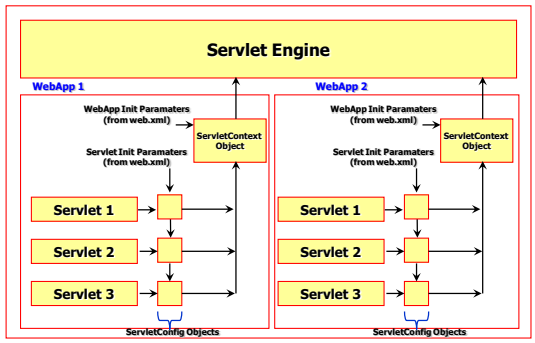
Arquitectura interna de um Servlet Container



Arquitectura interna de um Servlet Container

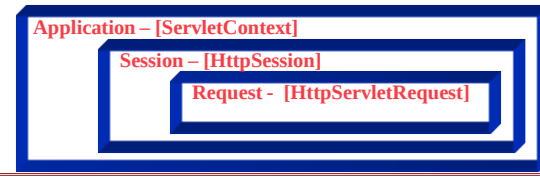


Ambiente de Execução de uma Servlet



Objectos para manutenção de estado

- Request – Durante um pedido
- Session – Guarda informação entre vários pedidos para um mesmo cliente. (informação mantida através de Cookies ou por rescrita do URL)
- Application – Guarda informação ao nível da aplicação, ou seja, a informação está disponível para todas as instâncias da servlet/jsp



Variáveis predefinidas

- **Variáveis implícitas numa página JSP**
  - **request**
    - HttpservletRequest (1º argumento no método *service*)
  - **response**
    - HttpservletResponse (2º argumento no método *service*)
  - **out**
    - O *Writer* utilizado para escrever dados para o cliente (obtido com o response.getWriter())
  - **session**
    - A HttpSession associado com o pedido recebido
  - **application**
    - O ServletContext (para partilha de dados) (obtido via getServletConfig().getContext())
  - **page**
    - O objecto que vai tratar a página (this)

Sintaxe Básica

- **Texto HTML**
  - <h1>bla bla</h1>
  - Passado para o cliente. Na realidade é transformado para código da servlet:

```
out.print("<h1>bla bla</h1>");
```

- **Comentários HTML**
  - <!-- Comentário -->
  - Passado para o cliente como código HTML normal
- **Comentários em JSP**
  - <%-- comentários --%>
  - Não são enviados para o cliente

Tipos de elementos

- Elementos de Scripting
- Elementos de directivas
- Elementos de acções

Elementos de Scripting

- **Expressões:**
  - Formato: `<%= expressão %>`
  - A expressão é avaliada e escrita no output da servlet, ou seja, é inserido na servlet como: `out.println(expressão);`
- **Scriptlets:**
  - Formato: `<% código %>`
  - O código é inserido textualmente no método service();
- **Declarações:**
  - Formato: `<%! código %>`
  - Colocado textualmente no corpo da classe Servlet fora de qualquer método existente

Expressões JSP

- **Sintaxe:**

```
<%= expressão %>
```

- **Semântica:**
  - As expressões são avaliadas, convertidas para Strings e colocadas na página HTML no lugar onde estava a etiqueta no ficheiro JSP
  - Normalmente convertida para: `out.println(expressão);`
- **Exemplos:**
  - Data corrente: `<%= new java.util.Date() %>`
  - O seu HostName `<%= request.getRemoteHost() %>`
- **Sintaxe compatível com XML:**
  - `<jsp:expression>ExpressãoJava</jsp:expression>`
  - Os servidores são obrigados a suportar esta sintaxe a partir versão JSP 1.2
  - Não se pode misturar a sintaxe XML com a tradicional na mesma página

Correspondência JSP/Servlet

- **JSP original:**

```
<h1> O resultado é: </h1>
<%= 5* Math.random()+1024 %>
OU
<jsp:expression>5* Math.random()+1024</jsp:expression>
```

- **Resultado possível em código da Servlet:**

```
public final class my1.1rstJspNotXml_jsp extends org.apache.jasper.runtime.HttpJspBase
                    implements org.apache.jasper.runtime.JspSourceDependent {

  ...
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
      ...
      out.write("<html>\r\n");
      out.write("<head></head>\r\n");
      out.write("<body>\r\n");
      out.write("<h1>O resultado é: </h1>\r\n");
      out.print(5* Math.random()+1024);
      out.write("\r\n");
      out.write("</body>\r\n");
      out.write("</html>");
      ...
  }
}
```


Exemplo

my2ndJsp.jsp

```
<html>
<head></head>
<body>
<h1>A minha segunda JSP</h1>
<ul>
<li>Data: <%= new java.util.Date() %></li>
<li>Esta ligado no hostname: <%= request.getRemoteHost() %></li>
<li>O seu username como um parametro:
<%= request.getParameter("username") %></li>
</ul>
</body>
</html>
```

Scriptlets JSP

- **Sintaxe:**

```
<% Código Java %>
```

- **Semantica:**
  - O código Java é copiado textualmente para o método `service()`
- **Exemplos:**

```
<%
   String queryData = request.getQueryString();
   out.println("Attached GET data: " + queryData);
%>
```

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

- **Sintaxe compatível com XML:**
  - `<jsp:scriptlet>Código Java</jsp:scriptlet>`

Correspondência JSP/Servlet

- **JSP original:**

```
<%= foo() %>
<% bar(); %>
```

Scriptlet

- **Resultado possível em código da Servlet:**

```
public final class myXptordJspNotXml_jsp extends org.apache.jasper.runtime.HttpJspBase
                    implements org.apache.jasper.runtime.JspSourceDependent {
  ...
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
      ...
      out.write(foo());
      bar();
      ...
  }
}
```


Utilização Comum

- Impressão de código HTML condicional:

Exemplo.jsp

```
...
<h1>Impressão de texto condicional</h1>
<% if (xpto>0) {%>
   positivo
<%} else {%>
   negativo
<% } %>
...
```

```
...
out.println("<h1>Impressão de texto condicional</h1>");
if (xpto>0){
 out.println ("positivo");
} else {
 out.println ("negativo");
}
...
```

Exemplo.java

Declarações JSP

- **Sintaxe:**

```
<%! declarações %>
```

- **Semantica:**
  - Colocado textualmente no corpo da classe Servlet fora de qualquer método existente
  - Define métodos ou variáveis da classe gerada
  - Cuidado com a concorrência no acesso a estas variáveis
  - Existe apenas uma instancia da servlet, mas uma Thread por cada pedido
- **Exemplos:**
  - `<%! private int umCampoQq = 5; %>`
  - `<%! private String umMetodoQq(...) {...} %>`
- **Sintaxe compatível com XML:**
  - `<jsp:declaration>Código Java para declarações</jsp:declaration>`

Correspondência JSP/Servlet

- **JSP original:**

```
<%! private int contagem=0;%>
<%! private synchronized int getContagem(){
      return ++contagem;
  }%>
```

- **Resultado possível em código da Servlet:**

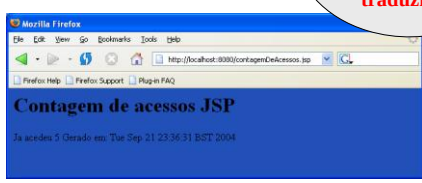
```
public final class myXpto2rdJspNotXml_jsp extends org.apache.jasper.runtime.HttpJspBase
                    implements org.apache.jasper.runtime.JspSourceDependent {
  ...
  private int contagem=0;
  private synchronized int getContagem(){
    return ++contagem;
  }

  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
      ...
  }
}
```


Exemplo de contagem de acessos

contagemDeAcessos.jsp

```
<html>
<head></head>
<body bgcolor="#2250B9">
<h1>Contagem de acessos JSP</h1>
<%! private int contagem=0;%>
<%! private synchronized int getContagem(){
    return ++contagem;
}%>
<p> Ja acedeu <%= getContagem() %>
Gerado em: <% out.println(new java.util.Date()); %>
</p>
</body>
</html>
```

EXERCÍCIO: Como fica esta Servlet depois da página JSP ser traduzida?



Directivas para o RunTime JSP

- Elementos de directivas – são instruções para o RunTime das JSP **(Jasper)**
- Permite especificar por exemplo:
  - Que classes queremos importar
  - Qual o tipo de dados gerado pela JSP
  - Como utilizar o Multi-Threading
  - Se vai ser utilizada a sessão
  - Qual o tamanho do buffer de saída
  - Qual a página de tratamento dos erros
- Sintaxe:

```
<%@ directiva atributo="…" %>
```

Elementos de directivas

- Directivas:
  - **"include"** – Inclusão de um ficheiro estático (ex: html, jsp, etc…)
    - Atributo: file="…"
    - É a única directiva que pode aparecer em qualquer parte do documento jsp, todas as outras têm de vir no início do ficheiro.

    Ex: <%@ include file="headers.html" %>
  - **"taglib"** – Importar uma biblioteca de etiquetas
  - **"page"** – Parâmetros relacionados com a página
    - Atributos:
      - **import**="…" – Classes a importar

        Ex: <%@ page import="java.util.Date" %>
      - **session**="…" – Se é utilizado o estado sessão *(por omissão true)*

        Ex: <%@ page session="FALSE" %>
      - **contentType**="…" – Tipo do documento que a JSP vai retornar

        Ex: <%@ page contentType="text/html" %>
      - **Buffer**="…" – tamanho do buffer a utilizar para o retorno dos dados *(por omissão 8KB, se o valor for "none" não é feita bufferização)*

        Ex: <%@ page buffer="12KB" %>

Continua…

Directivas para o RunTime JSP (cont.)

- "**page**" – Parâmetros relacionados com a página (cont.)
  - Atributos (continuação):
    - **errorPage**="URI" – Handler da página que vai tratar os erros

      Ex: <%@ page errorPage="/pathToPage" %>
    - **isErrorPage**="…" – Indica se a página é para tratamento de erros para ter acesso aos dados sobre as excepções ocorridas. (por omissão FALSE)

      Ex: <%@ page isErrorPage="TRUE" %>
    - **autoFlush="…"** – Indica se deve ser feito o flush do buffer quando este fica cheio

      Ex: <%@ page autoFlush="TRUE" %>
    - **isThreadSafe="…"** – Indica se a servlet é thread safe ou não (por omissão assume TRUE)

      Ex: <%@ page isthreadSafe="FALSE" %>

Exemplos de atributos da directiva "page"

- Exemplo "**import**":

```
<%@page import="java.util.Date,java.util.Vector" %>
<%= new Date() %>
```

- As classes têm de ser alcançáveis através do **CLASSPATH**
  - Em geral, a directoria **WEB-INF/classes** é usada como base de pacotes e/ou classes do utilizador
  - A directoria **WEB-INF/lib** serve para colocar todos os Java Archives (JAR's) que forem necessários.
- Tratamento de erros "**errorPage**" e "**isErrorPage**"

geraErro.jsp

```
<%@page errorPage="trataErro.jsp" %>
<%= 30/0 %>
```

trataErro.jsp

```
<%@page isErrorPage="TRUE" %>
<%= exception.getMessage() %>
```

Apenas neste caso temos acesso à variável "exception"

Tratamento de erros

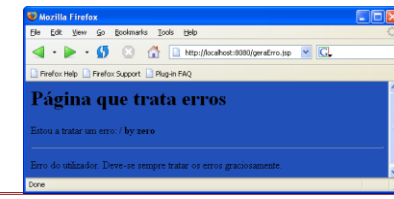
- Tratamento de erros "**errorPage**" e "**isErrorPage**"

geraErro.jsp

```
<%@ page errorPage="trataErro.jsp" %>
<html>
<head></head>
<body bgcolor="#2250B9">
<h1>A minha JSP que gera um erro</h1>
<p>Valor: <%= 30/0 %></p>
</body>
</html>
```

trataErro.jsp

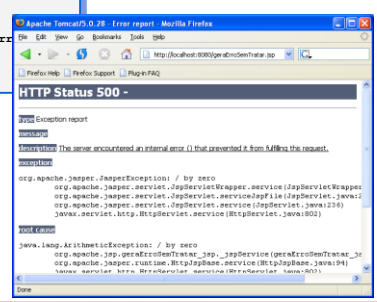
```
<%@ page isErrorPage="TRUE" %>
<html>
<head></head>
<body bgcolor="#2250B9">
<h1>Página que trata erros</h1>
<p>Estou a tratar um erro: <b>
<%= exception.getMessage() %></b></p>
<hr/>
<p>Erro do utilizador. Deve-se sempre
tratar os erros graciosamente.</p>
</body></html>
```



Tratamento de erros (cont.)

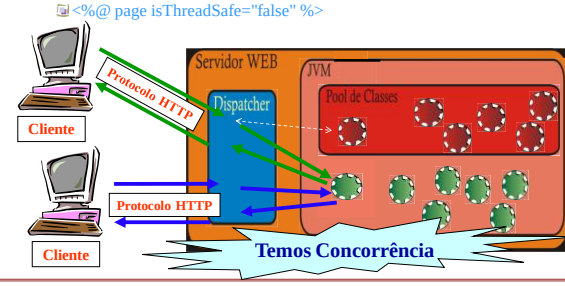
geraErro.jsp

```
<html>
<head></head>
<body bgcolor="#2250B9">
<h1>A minha JSP que gera um err
<p>Valor: <%= 30/0 %></p>
</body>
</html>
```



Exemplos de atributos da directiva "page"

- Exemplo do atributo **isThreadSafe**:
  - Por omisão *isThreadSafe* é **TRUE** indicando que a Servlet deve fazer o seu próprio sincronismo no acesso a recursos partilhados.
  - Caso queiramos forçar o Tomcat a tratar disso com o ***SingleThreadModel***:
    - <%@ page isThreadSafe="false" %>



Elementos de acções

- **Action Elements são como o nome indica, *request-time oriented*.**

- **As acções dividem-se nas seguintes categorias:**
  - **Control Tags**
    - Etiquetas para controlar o fluxo de execução para outras JSP's
  - **Bean Tags**
    - Etiquetas especializadas para utilizar JavaBeans
  - **Custom Tags**
    - Etiquetas que permitem definir e utilizar custom JSP tags

Control Tags

- **Estas etiquetas permitem passar o controlo da execução e parâmetros para outras páginas jsp's.**

- **<jsp:param name="name" value="value" />**
  - Serve para acrescentar parâmetros ao request de modo a serem acessíveis em JSPs que sejam incluídas no mesmo pedido ou em JSPs para onde o fluxo de execução tenha sido reencaminhado.
    - Ex: <jsp:param name="hoje" value="<%= new java.util.Date()%>" />
- **<jsp:include page="URL" />**
  - Permite incluir outros documentos. (ex: html, jsp's, etc…)
  - Inclui os documentos em *request-time*, ao contrário da directiva include que é em translation-time.
- **<jsp:forward page="URL" />**
  - Semelhante ao include, mas a execução é passada para a nova página e nunca retornada para o documento original.
  - Suportado pelo: getServletContext().getRequestDispacher("/…");
- **<jsp:plug-in type="bean|applet" code="codeName" codebase="URL" />**
  - Permite direccionar o plug-in no browser cliente. O plug-in permite receber a informação do código a carregar.

Bean Tags

- **JavaBeans são classes java com propriedades e respectivos *setters* e *getters***
  - Têm de informar o Run-Time que suportam a persistência dos seus dados implementando o interface *Serializable* ou *Externalizable*

- **Existem 3 tags para dar suporte aos JavaBeans:**
  - **<jsp:useBean>**
    - Ex: <jsp:useBean id="today" class="java.util.Date">

    Equivalente a:

    java.util.Date Today = new java.util.Date();
  - **<jsp:setProperty>**
    - Ex: <jsp:setProperty id="transaction" property="transactionType">

    Equivalente a:

    transaction.setTransactionType(request.getParameter("transactionType"));
  - **<jsp:getProperty>**
    - Ex: <jsp:getProperty id="transaction" property="transactionType">

    Equivalente a:

    transaction.getTransactionType();

Exemplo de Utilização - Redirect

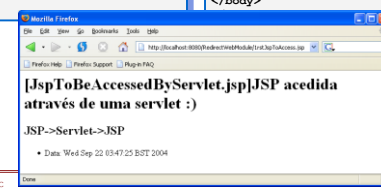
- **Utilização de redireccionamento na execução JSP→Servlet→JSP**

1rstJspToAccess.jsp

```
<html>
<head></head>
<body>
<h1>[1rstJspToAccess.jsp]A minha primeira JSP</h1>
<ul>
<li>Data: <% out.println(new java.util.Date()); %></li>
</ul>
<p>Vou redireccionar para uma servlet</p>
<jsp:forward page="/redirectservlet" />
</body>
</html>
```

JspToBeAccessedByServlet.jsp

```
<html>
<head></head>
<body>
<h1>[JspToBeAccessedByServlet.jsp]JSP acedida através de uma servlet :)</h1>
<h2><b>JSP->Servlet->JSP</b></h2>
<ul>
<li>Data: <% out.println(new java.util.Date()); %></li>
</ul>
</body>
```



Exemplo de Utilização - Redirect

RedirectServlet.java

```
...
public class RedirectServlet extends HttpServlet {

//Process the HTTP Get request
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>RedirectServlet</title></head>");
    out.println("<body bgcolor=\"#ffffff\">");
    out.println("<p>Estou na <b>SERVLET</b></p>");
    out.println("</body></html>");
    getServletContext().getRequestDispatcher("/JspToBeAccessedByServlet.jsp").forward
                                                          (request, response);
  }

  ...
}
```


Tópicos avançados

- **Como construir uma JSP:**
  - Elementos em scriptlets invocando o código da Servlet directamente
  - Elementos em scriptlets invocando o código da Servlet indirectamente (através de classes utilitárias)
  - Utilizando JavaBeans
  - Utilizando Enterprise JavaBeans (J2EE)
  - Etiquetas personalizadas (Custom tags)
  - Combinação de Servlets/JSP (MVC), utilizando Beans e etiquetas personalizadas

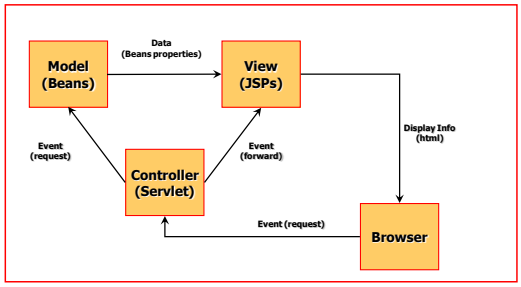
Aplicação Simples

Aplicação Complexa

Assuntos para cadeiras mais avançadas

Padrão Model View Control - MVC

- **Adaptando o modelo às tecnologias referidas temos:**



Conclusão

- **Expressões JSP**
  Formato: <%= expressão %>
  inserida num out.println e adicionada ao service() da Servlet
- **Scriptlets JSP**
  Formato: <% Código Java%>
  Código Java inserido literalmente ao service() da Servlet
- **Declarações JSP**
  Formato: <%! Código Java %>
  Código Java inserido literalmente no corpo da classe da Servlet
- **Variáveis predefinidas**
  request, response, out, session, application, config, pageContext, page

Conclusão

- As **JavaServer Pages** são interpretadas no lado do servidor (server-side)
- As JSP são transformadas para Servlets, depois compiladas e instanciadas pelo Web container para atenderem pedidos HTTP
- Têm todas as vantagens das Servlets como a portabilidade, potencialidade da linguagem Java, etc…
- Têm a vantagem de poder separar a apresentação (**HTML**) da geração dinâmica dos dados (**Java**)
- Lentas apenas no primeiro acesso, após o qual a instancia da Servlet correspondente já está em memória pronta a aceitar novos pedidos

Lista de Exemplos Demonstrativos

- Índice [Exemplos\]
  - *Estrutura básica das JSP's:* [my1rstJsp\]
    - my1rstJsp; [my1rstJsp.jsp]
    - my1rstJsp Notação Xml; [my1rstJspNotXml.jsp]
    - my1.1rstJsp Notação Xml; [my1.1rstJspNotXml.jsp]
    - my2ndJsp; [my2ndJsp.jsp]
    - my3rdJsp; [my3rdJsp.jsp]
  - *Contagem de acessos:* [contagem de acessos\]
    - contagem de acessos; [contagemDeAcessos.jsp]
  - *Tratamento de Erros graciosamente:* [tratamentoDeErros]
    - JSP que gera erro; [geraErro.jsp]
    - JSP que trata erros; [trataErro.jsp]
    - JSP que gera erro, mas não o trata; [geraErroSemTratar.jsp]
  - *Objectos Implicitos nas JSP's:* [ObjImplWebModule\]
    - Lista Objectos; [objectosImplicitos.jsp]

Lista de Exemplos Demonstrativos

- *Redirect*: [redirectJspServletJSp\]
  - JSP→Servlet→JSP; [1rstJspToAccess.jsp] [RedirectServlet.java] [JspToBeAccessedByServlet.jsp]
- *Custom Tags simples:* [SimpleCustomTags\]
  - JSP que usa as tags; [foo.jsp]
  - Tag Library descriptor (tld); [WEB-INF\jsp\example-taglib.tld]
  - Definição das tags em Java; [src*.java]

Glossário de Termos

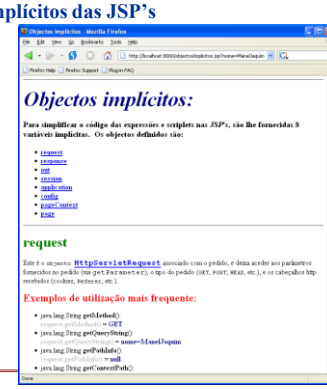
- `JSP` – JavaServer Pages;
- `ASP` – Active Server Pages;
- `PHP` – PHP Hypertext Preprocessor (PHP: Personal Home Page);
- `JVM` – Java Virtual Machine;
- `URL` – Uniform Resource Identifier;
- `API` – Application Programming Interface;

Bibliografia

- Professional JSP 2nd Edition
  - Karl Avedal, Robert Burdick, … , Steve Wilkinson
  - Publisher: Wrox Press Ltd. ISBN: 1861004958
- The Complete Reference JSP;
  - Phil Hanna – MC Graw Hill
- *JSP Tag Libraries;*
  - Gal Shachor, Adam Chace, Magnus Rydin - Manning

Exemplos de Utilização

- **Objectos Implícitos das JSP's**



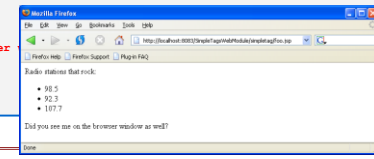
Custom Tags

- **São etiquetas feitas à medida pelo próprio programador ou por um outro fabricante de bibliotecas.**
- **É o meio que temos disponível para separar totalmente o desenvolvimento das aplicações web:**
  - **Programadores** – Têm a tarefa de criar as bibliotecas de etiquetas e todo o código em Java que as implementa;
  - **Web designers** – Têm de saber html e as etiquetas que os programadores fornecem;

Exemplos de Utilização – Custom Tags

foo.jsp

```
<html>
...
<body>
<%@ taglib uri="http://jakarta.apache.org/tomcat/examples-taglib" prefix="eg"%>

Radio stations that rock:
<ul>
<eg:foo att1="98.5" att2="92.3" att3="107.7">
<li><%= member %></li>
</eg:foo>
</ul>

<eg:log>
Did you see me on the stderr window?
</eg:log>

<eg:log toBrowser="true">
Did you see me on the browser
</eg:log>

</body>
</html>
```



Exemplos de Utilização – Custom Tags (cont.)

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web ...
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>SimpleTagsWebModule</display-name>
  <taglib>
    <taglib-uri>http://jakarta.apache.org/tomcat/examples-taglib</taglib-uri>
    <taglib-location>/WEB-INF/jsp/example-taglib.tld</taglib-location>
  </taglib>
</web-app>
```

example-taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
...
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  ...
  <!-- A simple Tag -->
  <!-- foo tag -->
  <tag>
    <name>foo</name>
    <tag-class>examples.FooTag</tag-class>
    <tei-class>examples.FooTagExtraInfo</tei-class>
    <body-content>JSP</body-content>
    <description>
          Perform a server side action; uses 3 mandatory attributes
    </description>

    <attribute>
      <name>att1</name>
      <required>true</required>
    </attribute>
    ...
  </tag>
...
</taglib>
```

Exemplos de Utilização – Custom Tags (cont.)

FooTag.java

```
...
public class FooTag   extends ExampleTagBase
{
    private String atts[] = new String[3];
    int i = 0;

    private final void setAtt(int index, String value) {
        atts[index] = value;
    }
    public void setAtt1(String value) {
        setAtt(0, value);
    }
    public void setAtt2(String value) {
        setAtt(1, value);
    }
    public void setAtt3(String value) {
        setAtt(2, value);
    }

    /**
     * Process start tag
     *
     * @return EVAL_BODY_INCLUDE
     */
    public int doStartTag() throws JspException {
        i = 0;
            return EVAL_BODY_TAG;
    }

    public void doInitBody() throws JspException {
        pageContext.setAttribute("member", atts[i]);
        i++;
    }

                    →→ →→ →→ →→
```

```
→→ →→ →→ →→

public int doAfterBody() throws JspException {
        try {
            if (i == 3) {
                bodyOut.writeOut(bodyOut.getEnclosingWriter());
                return SKIP_BODY;
            } else {
                pageContext.setAttribute("member", atts[i]);
            i++;
            return EVAL_BODY_TAG;
        } catch (IOException ex) {
            throw new JspTagException(ex.toString());
        }
    }
}
```

Exemplos de Utilização

- **Objectos Implícitos das JSP's**