Java Server Pages e Java Beans

João Correia Lopes

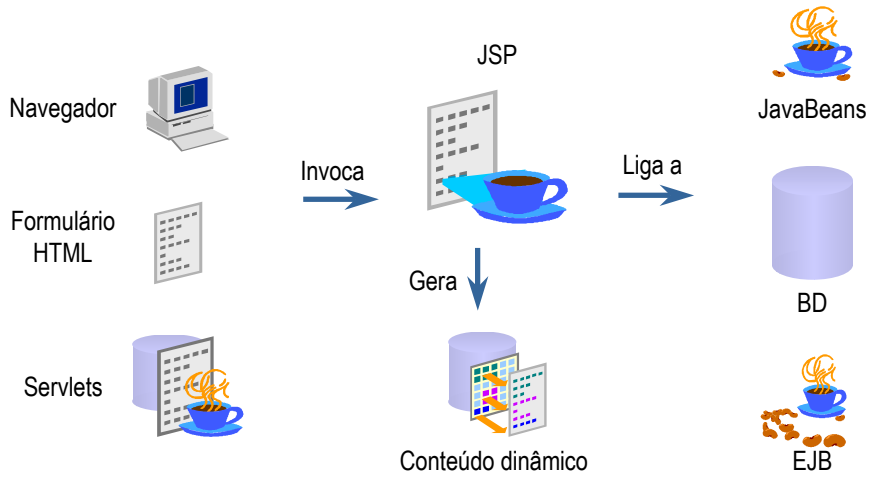
Faculdade de Engenharia Universidade do Porto

http://www.fe.up.pt/~jlopes/
jlopes@fe.up.pt

Conteúdo

- Vantagens da tecnologia JSP
- Arquitectura JSP
- Ciclo de vida de uma página JSP
- Sintaxe e semântica de JSP
- Papel de componentes *JavaBeans* dentro de páginas JSP
- Exemplos
- JSP *Custom Tags*

Generalidades



O que está errado com os servlets?

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class MyDearServlet extends HttpServlet {
  //Process the HTTP GET request
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
    doPost(request, response);
  }
  //Process the HTTP POST request
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    StringBuffer output= new StringBuffer(2048);
    output.append("<HTML>");
    output.append("<HEAD><TITLE>Using Servlets</TITLE></HEAD>");
    output.append("<BODY BGCOLOR=#123123>");
```

O que está errado com os servlets? (2)

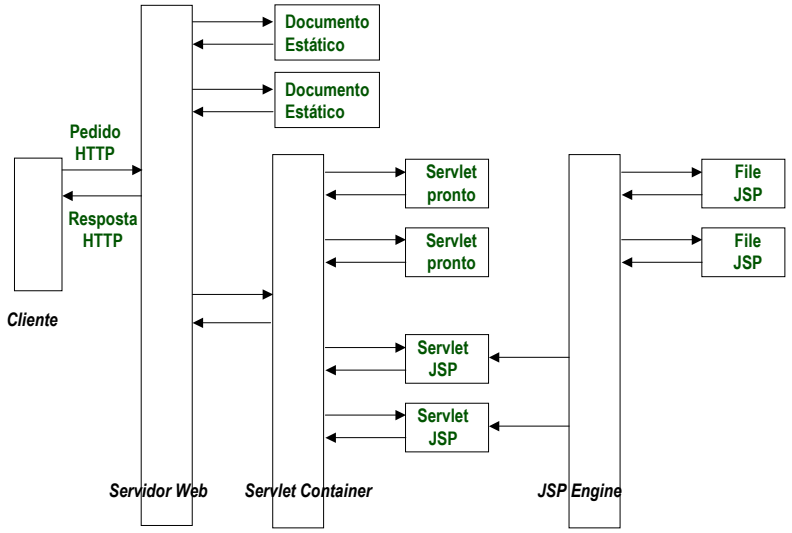
```
    //Get parameter names
    Enumeration parameters = request.getParameterNames();
    String param = null;
    while (parameters.hasMoreElements()) {
      param = (String) parameters.nextElement();
      output.append(param + ":" + request.getParameter(param) + "<BR>");
    }
    output.append("</BODY>");
    output.append("</HTML>");
    PrintWriter out = response.getWriter();
    out.println(output);
    out.close();

  } //End of doPost method
  /* other parts of the class goes here
     ...
  */
} //End of class
```


O mesmo com JSP!

```
<%@ page import="java.util.Enumeration" %>
<HTML>
<HEAD><TITLE>Using JSP</TITLE></HEAD>
<BODY BGCOLOR=#DADADA>
<%
  //Get parameter names
  Enumeration parameters = request.getParameterNames();
  String param = null;
  while (parameters.hasMoreElements()) {
    param = (String) parameters.nextElement();
    out.println(param + ":" + request.getParameter(param) + "<BR>");
  }
  out.close();
%>
</BODY>
</HTML>
```

Arquitectura de Aplicações na Web



Java Server Pages (JSP)

- Suportada por variados servidores em variadas plataformas
- Maneira eficiente de colocar uma aplicação na Web.
- Extensão definida em cima da API de servlets.
- Maneira de escrever aplicações para a Web usando guiões (*scripting*).
  - Opera no modo pedido-resposta.
  - Gera conteúdo dinâmico com pouco ou nenhum código (bom para os não-programadores).
  - Contém texto HTML livremente misturado com código Java (para os programadores).
- Páginas JSP podem ser construídas por ferramentas convencionais de HTML/XML.

Exemplo

```
<jsp:useBean id="myBean" class="package2.DateCounterBean" />
<HTML>
<BODY>

A new hit at

<jsp:getProperty  name="myBean"  property="date" />
<br>

Number of hits to this page is now
<jsp:getProperty name="myBean" property="counter"/>

</BODY>
</HTML>
```


Benefícios de JSP

- Separa conteúdo de interface com o utilizador de conteúdo dinâmico dirigido pela lógica de negócio
  - Pode ser usado por programadores
  - e por desenhadores de HTML
- Baseado em servlets Java:
  - Eficiente
  - Robusto
  - Independente da plataforma
  - Os servlets devem ser usados estritamente para extensões aos servidores (controladores especializados para autenticação, acessos a bases de dados, etc.) e para comunicar com Applets e aplicações.
- Uso eficiente de tecnologia de componentes *JavaBean*
- Porta de entrada para *Enterprise Java Beans*

Invocar Páginas JSP

- Pode invocar-se uma página JSP directamente:

  ```
  http://host/main.jsp
  ```

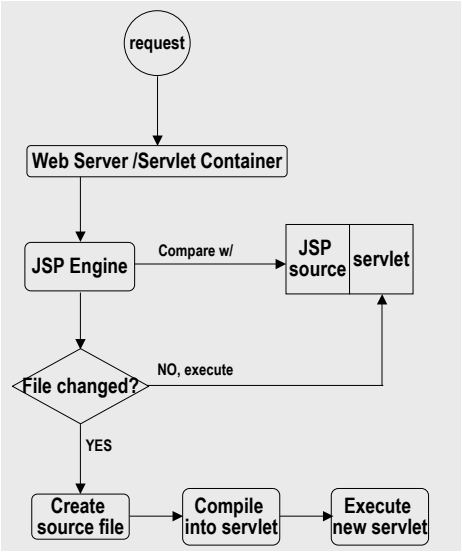
- Pode invocar-se uma página JSP indirectamente:
  - A partir de outra página JSP
  - A partir de um servlet
  - A partir de um formulário HTML

Dentro de JSP

- Uma página JSP é convertida automaticamente num servlet a primeira vez que é invocada.
  - São gerados ficheiros de código fonte Java
  - São gerados ficheiros .class
  - Pode ser usado o compilador JIT (*just-in-time*)
- Uma página JSP pode conter JavaBeans
  - Propriedades dos Bean são automaticamente inspeccionadas
  - Beans são reutilizados

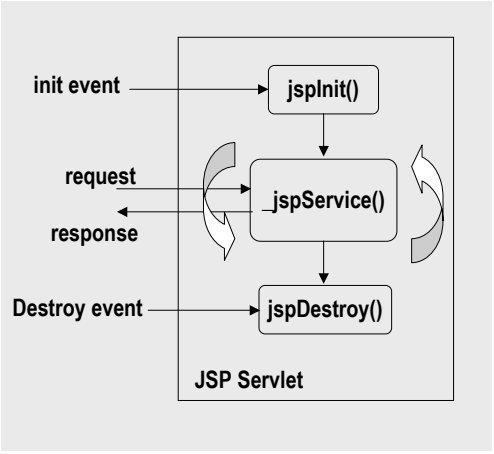
Arquitectura JSP

- Quando um cliente pede uma página JSP a um servidor Web, se esta página nunca correu, ou foi alterada, é primeiramente passada a *JSP Engine* que a compila para servlet.
- O servlet corre e o seu resultado é devolvido para o cliente (navegador)
- *JSP Engine* é também um servlet



Ciclo de Vida de uma Página JSP

- Depois da classe ser carregada no "Servlet Container", `jspService()` é responsável por responder ao pedido do cliente.
- Por omissão, este método é despachado num *thread* separado para responder a pedidos concorrentes.



Exemplo

```
<!-- set global info for the page -->
<%@ page language="java" %>
<HTML><HEAD>
<TITLE>Demo of a JSP page</TITLE>
</HEAD>
<BODY>
<!-- declares -->
<%! char c = 0; %>
<!- Scriplet - Java code -->
<%
   for (int i = 0; i < 26; i++) {
      for (int j = 0; j < 26; j++) {
         c= (char) (0x41 + (26 - i + j) % 26);
%>
<!- output the value of c.toString() to the HTML page -->
<%= c %>
<%
      }
%>
<br>
<%
   }
%>
</BODY></HTML>
```


Objectos Implícitos

- Os *scripts* JSP podem usar os seguintes objectos:
  - `request`
  - `response`
  - `out`
  - `session`
  - `application`
  - `config`
  - `pageContext`
  - `page`
  - `exception`

Elementos JSP Básicos

- `useBean` permite invocar JavaBeans.
- Directivas permitem a declaração de variáveis globais.
- Expressões permitem obter o valor de variáveis Java e expressões em cadeias de caracteres.
- *Scriptlets* contêm código Java reduzido.
- Declarações permitem a escrita de métodos Java completos.

Directivas

- São mensagens para a "*JSP Engine*"
- Permitem declarar valores globais.
- Começam com a sequência `<%@`
- Acabam com a sequência `%>`
- Tipos de directivas:
  - Directivas de página (definem atributos que se aplicam a toda uma página JSP
  - Directivas de inclusão (inclui um ficheiro de texto ou código)
  - Directivas de bibliotecas de etiquetas (*tag libraries*)

```
<%@ page language="java" %>
<%@ include file="signature.html" %>
```

Expressões

- Começam com a sequência <%=
- Contêm expressões curtas de código Java que são enviadas para a saída como cadeias.
- Acabam com a sequência %>
- Não acabam com ponto e vírgula
- Aumentam muito o poder de páginas JSP

```
<%@ page info="This line was created for …" %>
<%= this.getServletInfo() %>
<%= fooBean.getName() %>
<%= "a" + "b" %>
```

```
Hora: <%= java.util.Calendar.getInstance().getTime() %>
```


Scriptlets

- Começam com a sequência <%
- Contêm código Java que é interpretado de cada vez que um pedido é feito.
- Acabam com a sequência %>

```
Examples
<% if(request.getParameter("firstName")==null){ %>
Hello World
<% } else { %>
Hello,
<%= request.getParameter("firstName") %>
<% } %>
```

Exemplo: acesso a BD

```
<%@ page session="false" %>
<%@ page import="java.sql.*" %>
<%
  try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    System.out.println("JDBC driver loaded");
  }
  catch (ClassNotFoundException e) {
    System.out.println(e.toString());
  }
%>
<HTML><HEAD>
<TITLE>Display All Users</TITLE>
</HEAD>
<BODY>
<CENTER>
<BR><H2>Displaying All Users</H2>
<BR><BR>
<TABLE>
<TR><TH>First Name</TH>
<TH>Last Name</TH>
<TH>User Name</TH>
<TH>Password</TH>
</TR>
```


Exemplo: acesso a BD (2)

```
<%
 String sql = "SELECT FirstName, LastName, UserName, Password FROM Users";
 try {
  Connection con = DriverManager.getConnection("jdbc:odbc:JavaWe
  Statement s = con.createStatement();
  ResultSet rs = s.executeQuery(sql);
  while (rs.next()) {
    out.println("<TR>");
    out.println("<TD>" + rs.getString(1) + "</TD>");
    out.println("<TD>" + rs.getString(2) + "</TD>");
    out.println("<TD>" + rs.getString(3) + "</TD>");
    out.println("<TD>" + rs.getString(4) + "</TD>");
    out.println("</TR>");
  }
  rs.close();
  s.close();
  con.close();
 }
 catch (SQLException e) {
 }
 catch (Exception e) {
 }
%>
```

Declarações

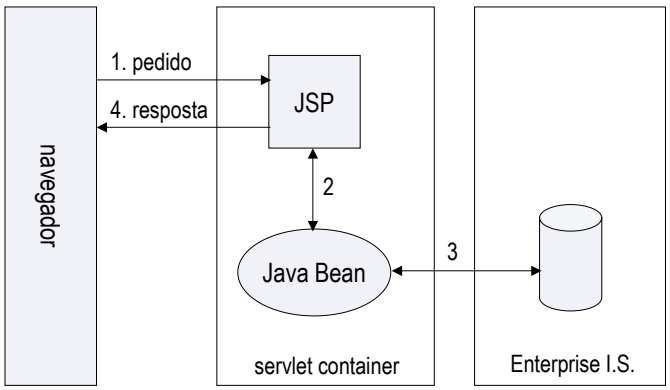
- Começam com a sequência `<%!`
- Contêm declarações Java
- Acabam com ponto e vírgula
- Acabam com a sequência `%>`
- Trabalham bem com expressões e *scriptlets*

```
<%! int n=10, sum=0; %>
<%  if (request.getParameter("num") != null)
      n= Integer.parseInt(request.getParameter("num"));
    for (int i=0; i <= n; i++ ) sum = sum + i;
%>
The sum of the first <%= n %> numbers is <%= sum %>
and n(n+1)/2 works out to <%= n*(n+1)/2 %>
```


Acções

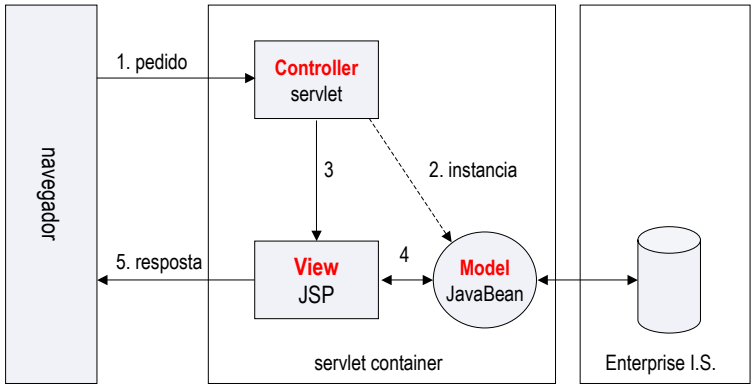
- *Control tags*
  - `<jsp:include page="URL" flush="true" />`
    - ao contrário da directiva de inclusão, a página é incluída aquando do pedido em vez de ser na altura da tradução
  - `<jsp:forward page="URL" />`
    - Neste caso o controlo é passado para a nova página
  - `<jsp:param name="name" value="value" />`
    - Para passar parâmetros nos dois casos anteriores
- *Bean tags*
  - `<jsp:useBean ... />`
- *Custom tags*
  - Bibliotecas de etiquetas

Arquitectura: Modelo 1



João Correia Lopes
Faculdade de Engenharia da Universidade do Porto

Arquitectura: Modelo 2 (MVC)



João Correia Lopes
Faculdade de Engenharia da Universidade do Porto

Tratamento de Excepções

- JSP fornece um mecanismo elegante para tratar excepções em tempo de execução.
- Podem colocar-se rotinas de tratamento de erros dentro das páginas JSP
- É possível fazer seguir para uma rotina de tratamento de erros uma excepção para um erro

```
<%@ page isErrorPage="false" errorPage="errorHandler.jsp" %>
```

- Essa página declara que é uma página de tratamento de erros.

```
<%@ page isErrorPage="true" %>
```


Gestão de Sessões

- Por omissão todas as páginas JSP participam numa sessão

```
<%@ page session="false" %>
```

- O objecto `HttpSession` pode ser acedido por *scriptlets* através do objecto `Session`
- Sessões servem para guardar objectos e *Beans* que podem ser partilhados por outras páginas JSP e *servlets*
- Objectos da sessão são identificados por um ID de sessão e guardados como *cookies* no navegador.

Gestão de Sessões...

- Tornar *foo* disponível durante a sessão

```
<%
Foo foo = new Foo();
session.putValue("foo",foo);
%>
```

- Retirar *foo* noutra página

```
<%
Foo myFoo = (Foo) session.getValue("foo");
%>
```


Componentes JavaBeans

- O modelo de componentes da tecnologia JSP é baseado na arquitectura de componentes *JavaBeans*.
- Componentes *JavaBeans* são objectos Java que mantêm dados (propriedades) e seguem um padrão bem definido de projecto/nomeação
  - a classe deve ter um construtor sem argumentos (ou não ter construtor que será criado pelo compilador)
  - implementa a interface `java.io.Serializable` ou `java.io.Externalizable`
  - O *Bean* encapsula as suas propriedades declarando-as `private` e fornece métodos (getter/setter) para ler e modificar os seus valores.

JavaBeans

- JavaBeans parte da capacidade do Java "Write Once, Run Anywhere" e estende-a para incluir "reuse everywhere".
- JavaBeans é um modelo de componentes independente da plataforma, portável, escrito em Java.
- Permite a criação de componentes de software pequenos, reutilizáveis
- Um programa de construção/integração visual pode combinar componentes de fontes diversas para criar aplicações, fácil e rapidamente.
- Um Bean é um componente JavaBean.
- Beans podem ser objectos visíveis (e.g. componentes AWT) ou invisíveis, tais como filas ou stacks

Arquitectura de JavaBeans

- Eventos
  - servem para notificar outros de uma dada ocorrência
  - `EventObjects`, `EventListeners`, fontes de eventos
- Propriedades
  - definem as características de um Bean

```
public void setPropertyName(PropertyType value);
public PropertyType getPropertyName();
```

- Métodos
  - métodos de Beans estão disponíveis para serem chamados desde que sejam declarados públicos

Componentes JavaBeans em JSP

- Antes de usar um Bean é necessário obter uma referência com useBean

```
<jsp:useBean id="user" class="com.jguru.Person" scope="session" />
```

  - A instância Person é criada só uma vez e colocada na sessão.
- As propriedades de um Bean podem ser alteradas com

```
<jsp:setProperty id="user" property="name" value="jGuru" />
```

- E retiradas com

```
<jsp:getProperty id="user" property="name" />
```


scope

- page
  - visível apenas no primeiro servlet ou JSP onde o pedido foi mapeado.
- request
  - visível no primeiro servlet ou JSP onde o pedido foi mapeado e em outros servlets ou páginas de seguida (*forward*) ou incluídas (*includes*).
- session (por omissão)
  - visível em todos os servlets ou JSP pedidos da mesma sessão do cliente.
- application
  - visível para todos os servlets na mesma aplicação.

JSP e JavaBeans

```
<jsp:useBean id="myBean" class="package2.DateCounterBean" />
<HTML>
<BODY>

A new hit at

<jsp:getProperty  name="myBean"  property="date" />
<br>

Number of hits to this page is now
<jsp:getProperty name="myBean" property="counter"/>

</BODY>
</HTML>
```


JSP e JavaBeans (2)

```
package package2;
import java.lang.*;
import java.util.*;
public class DateCounterBean {
  private int counter;
  private Date d;
  public DateCounterBean() {
    counter = (int) (1000 * Math.random());
    d = new Date();
  }
  public String getDate() {
    return (d = new Date()).toString();
  }
  public int getCounter() {
    return ++counter;
  }
}
```

Valores de propriedade

```
<HTML>
<HEAD>
<TITLE>Passing a value</TITLE>
</HEAD>
<BODY>
<CENTER>
Please type in a number in the box
<BR>
<FORM METHOD=POST ACTION=SimplePage.jsp>
<INPUT TYPE=TEXT NAME=memory>
<INPUT TYPE=SUBMIT>
</FORM>
</CENTER>
</BODY>
</HTML>
```

para mais do que uma, property="*"

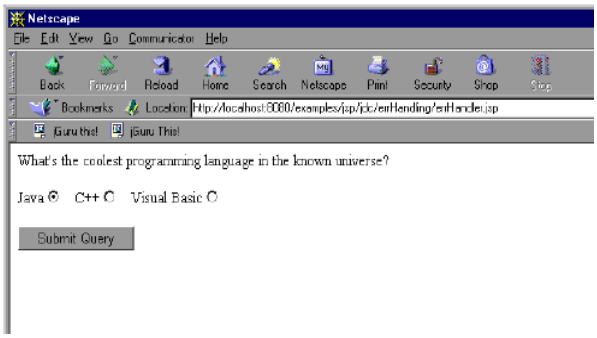
o mesmo nome, logo poderia ser omitido

```
<jsp:useBean id="theBean" class="com.brainysoftware.CalculatorBean"/>
<jsp:setProperty name="theBean" property="memory" param="memory"/>
The value of memory is
<jsp:getProperty name="theBean" property="memory"/>
```


Exemplo 1: errHandler.jsp

```
<%@ page errorPage="errorpage.jsp" %>
<html>
<body>
<form method=post action="errhandler.jsp">
What's the coolest programming language in the known universe?<p>
Java<input type=radio name=language value="JAVA" checked>
C++<input type=radio name=language value="CPP">
Visual Basic<input type=radio name=language value="VB">
<p>
<input type=submit>
</form>
<%
if (request.getMethod().equals("POST")) {
   if (request.getParameter("language").equals("JAVA"))
      out.println("<hr><font color=red>You got that right!</font>");
   } else {
      throw new Exception("You chose the wrong language!");
   }
}
%>
</body>
</html>
```

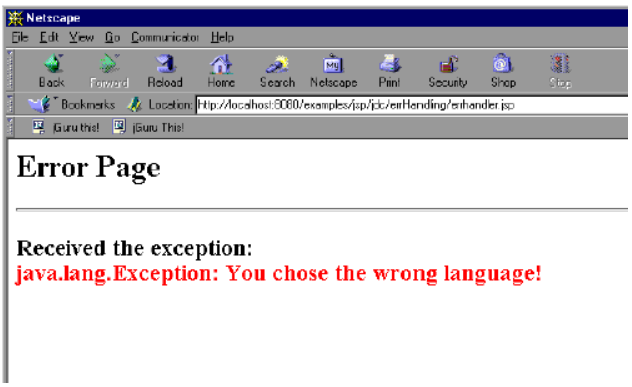
Exemplo 1: Visualizar HTML



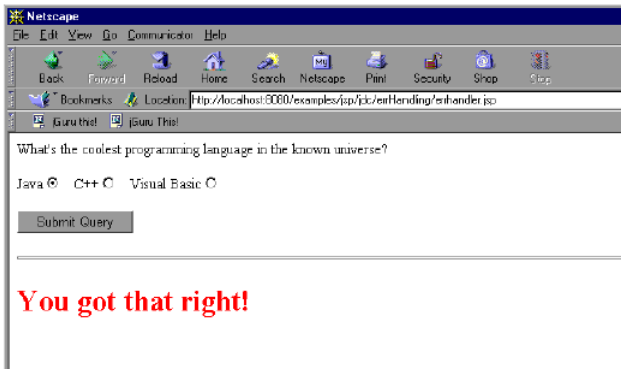
Exemplo 1: errorpage.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>
<h1>
Error Page
</h1>
<hr>
<h2>
Received the exception:<br>
<font color=red>
<%= exception.toString() %>
</font>
</h2>
</body>
</html>
```

Exemplo 1: Visualizar errorpage.jsp



Exemplo 1: Visualização s/ erro



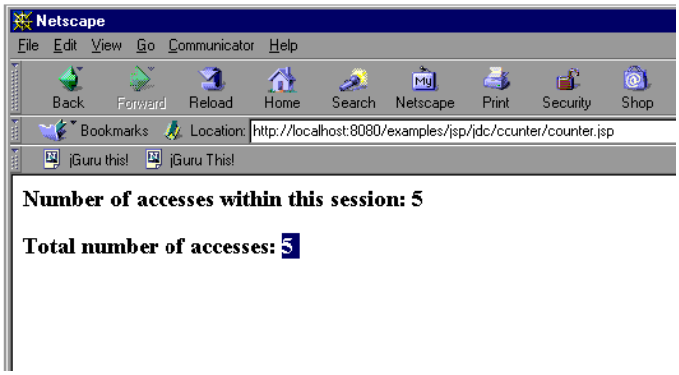
Exemplo 2: counter.jsp

```
<%@ page import="com.jguru.CounterBean" %>
<jsp:useBean id="session_counter" class="CounterBean" scope="session" />
<jsp:useBean id="app_counter" class="CounterBean" scope="application" />
<%
session_counter.increaseCount();
synchronized(page) {
  app_counter.increaseCount();
}
%>
<h3>
Number of accesses within this session:
<jsp:getProperty name="session_counter" property="count" />
</h3>
<p>
<h3>
Total number of accesses:
<% synchronized(page) { %>
<jsp:getProperty name="app_counter" property="count" />
<% } %>
</h3>
```


Exemplo 2: counterBean.java

```
package com.jguru;
public class CounterBean {
  int count;
  public int getCount() {
    return count;
  }
  public void increaseCount() {
    count++;
  }
}
```

Exemplo 2: Visualização



JSP Custom Tags

- Com Beans pode separar-se a parte de apresentação da página JSP da implementação das regras do negócio (código Java)
- Só há 3 acções disponíveis (jsp:usebean, jsp:getProperty e jsp:setProperty) e por isso é necessário usar código nas páginas JSP (scriptlets) impedindo a separação completa
- Para além disso os Beans são pensados para reutilização pelo que escreverem HTML directamente não é boa ideia
- A solução é utilizar "*custom Tags for custom actions*"
  - Têm acesso a todos os elementos da página JSP
  - Podem aceitar atributos que alteram o seu funcionamento

First Custom Tag

- WEB-INF/taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
 <tlibversion>1.0</tlibversion>
 <shortname></shortname>
 <tag>
  <name>myTag</name>
  <tagclass>pacote.MyCustomTag</tagclass>
 </tag>
</taglib>
```


First Custom Tag (2)

- WEB-INF/classes/pacote/MyCustomTag.Java

```
package pacote;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class MyCustomTag extends TagSupport {
  public int doEndTag() throws JspException {
    JspWriter out = pageContext.getOut();
    try {
      out.println("Hello from the custom tag.");
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return super.doEndTag();
  }
}
```

First Custom Tag (3)

- SimplePage.jsp

```
<%@ taglib uri="/myTLD" prefix="easy"%>
<easy:myTag/>
```

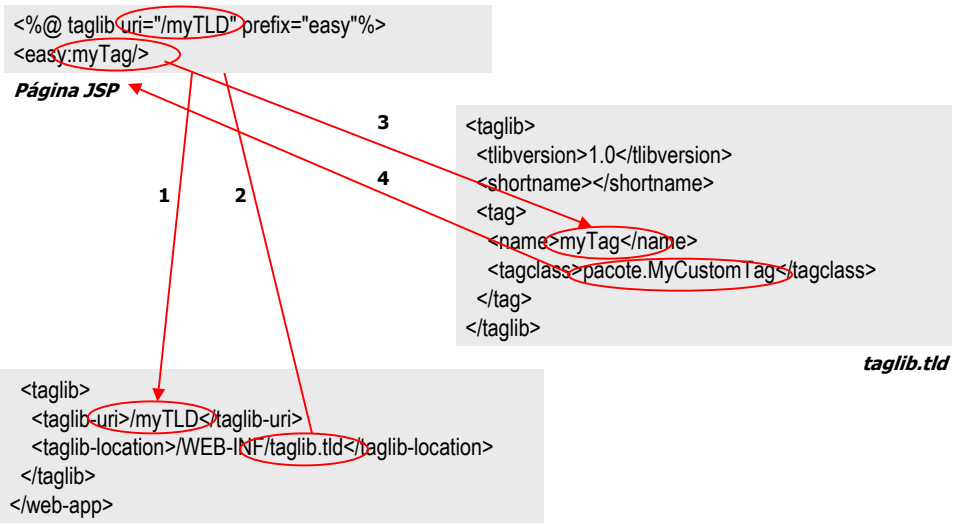
```
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="easy"%>
<easy:myTag/>
```

- WEB-INF/web.xml

```
<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
 <display-name>template</display-name>
 <taglib>
  <taglib-uri>/myTLD</taglib-uri>
  <taglib-location>/WEB-INF/taglib.tld</taglib-location>
 </taglib>
</web-app>
```


First Custom Tag (4)



Custom Tag, sintaxe

- Directiva *taglib*

  <@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
- *custom tag* com corpo

  <prefix:tagName>body</prefix:tagName>
- Uso de atributos

  <m:myTag number="2" power="8"/>

JSP Custom Tag API

- A classe Java ligada à "*custom tag*" e que será invocada de cada vez que o "*JSP container*" encontra a tag é chamada ***tag handler***
- Para ter alguma funcionalidade a *tag handler* deve implementar uma interface da pacakge `javax.servlet.jsp.tagext` ou estende uma classe desta package
- As interfaces mais importantes são `Tag` e `BodyTag`
  - uma ou outra devem ser implementadas directa ou indirectamente
- Tag
  - `doStartTag`, `doEndTag`, `getParent`
  - `setParent`, `setPageContext`, `release`

Ciclo de vida de um *Tag Handler*

- O *JSP container* obtém uma instância da classe ou cria uma nova instância e chama `setPageContext()` passando-lhe o objecto que representa a página JSP onde foi encontrada a tag

  public void setPageContext(PageContext pageContext)
- chama o método `setParent()` passando-lhe um objecto que representa a tag que envolve o *tag handler* corrente ou null

  public void setParent(Tag parent)
- coloca os atributos na tag, se existirem chama todos os métodos *setter* (como nos Beans)
- chama o método `doStartTag()`, que pode devolver `SKIP_BODY` (não processa o corpo da tag) ou `EVAL_BODY_INCLUDE`

  public int doStartTag() throws javax.servlet.jsp.JspException
- chama o método `doEndTag()`, que pode devolver `SKIP_PAGE` (não processa o resto da página) ou `EVAL_PAGE`

  public int doEndTag() throws javax.servlet.jsp.JspException
- chama o método `release()` que pode devolver recursos

Exemplo 3: DoublerTag

```
package pacote;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class DoublerTag implements Tag {
  private int number;
  public void setNumber(int number) {
    this.number = number;
  }
  PageContext pageContext;

  public void setParent(Tag t) {
  }

  public void setPageContext(PageContext p) {
    pageContext = p;
  }
```

DoublerTag (2)

```
  public void release() {
  }

  public Tag getParent() {
    return null;
  }
  public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.println("Double of " + number + " is " + (2 * number));
    }
    catch(Exception e) {
    }
    return EVAL_BODY_INCLUDE;
  }

  public int doEndTag() throws JspException {
    return EVAL_PAGE;
  }
}
```


DoublerTag (3)

- doubler.jsp

```
<%@ taglib uri="/myTLD" prefix="easy"%>
<easy:myTag number="12" />
```

- WEB-INF/taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
 <tlibversion>1.0</tlibversion>
 <shortname></shortname>
 <tag>
   <name>myTag</name>
   <tagclass>pacote.DoublerTag</tagclass>
   <attribute>
     <name>number</name>
     <required>true</required>
   </attribute>
 </tag>
</taglib>
```

Exemplo 4: PowerTag

```
package pacote;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class PowerTag implements IterationTag {
  PageContext pageContext;

  private int number;
  private int power;
  private int counter;
  private int result = 1;

  // the setter for number
  public void setNumber(int number) {
    this.number = number;
  }
  // the setter for power
  public void setPower(int power) {
    this.power = power;
  }
```


PowerTag (2)

```
public void setParent(Tag t) {
}
public void setPageContext(PageContext p) {
  pageContext = p;
}
public Tag getParent() {
  return null;
}
public int doStartTag() {
  return EVAL_BODY_INCLUDE;
}
public int doAfterBody() {
  counter++;
  result *= number;
  if (counter==power)
    return SKIP_BODY;
  else
    return EVAL_BODY_AGAIN;
}
```

PowerTag (3)

```
  public int doEndTag() throws JspException {
    System.out.println("doEndTag");
    try {
      JspWriter out = pageContext.getOut();
      out.println(number + "^" + power + "=" + result);
    }
    catch(Exception e) {
    }
    return EVAL_PAGE;
  }
}
```


PowerTag (4)

- power.jsp

```
<%@ taglib uri="/myTLD" prefix="easy"%>
<easy:myTag number="2" power="3" />
```

- WEB-INF/taglib.tld

```
...
<taglib>
 <tlibversion>1.0</tlibversion>
 <shortname></shortname>
 <tag>
  <name>myTag</name>
  <tagclass>pacote.PowerTag</tagclass>
  <attribute>
   <name>number</name>
   <required>true</required>
  </attribute>
  <attribute>
   <name>power</name>
   <required>true</required>
  </attribute>
 </tag>
</taglib>
```

Manipular o corpo

- Uma *custom tag* pode ter corpo

```
<%@ taglib uri="/myTLD" prefix="x"%>
<x:myTag>This is the body content</x:myTag>
```

- Com as interfaces `Tag` ou `IterationTag` não é possível manipular o corpo; deve ser usada a interface `BodyTag` e a classe `BodyContent`
- A interface `BodyTag` junta os métodos `doInitBody()` e `setBodyContent()` e dois inteiros `EVAL_BODY_BUFFERED` e `EVAL_BODY_TAG` (*deprecated* em JSP 1.2)

Exemplo 5: EncoderTag

- encoder.jsp

```
<%@ taglib uri="/myTLD" prefix="easy"%>
<easy:myTag><BR> means change line.</easy:myTag>
```

- WEB-INF/taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
 <tlibversion>1.0</tlibversion>
 <shortname></shortname>
 <tag>
  <name>myTag</name>
  <tagclass>pacote.EncoderTag</tagclass>
  <bodycontent>tagdependent</bodycontent>
 </tag>
</taglib>
```

EncoderTag (2)

```
package pacote;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class EncoderTag implements BodyTag {

  PageContext pageContext;
  BodyContent bodyContent;

  /**
   * Encode an HTML tag so it will be displayed as it is on
   * the browser. Particularly, this method searches the
   * passed in String and replace every occurrence
   * of the following character:
   * '<' with "&lt;"
   * '>' with "&gt;"
   * '&' with "&amp;"
   * //'"' with "&quot;"
   * ' ' with " "
   */
```


EncoderTag (3)

```
  private String encodeHtmlTag(String tag) {
    if (tag==null)
      return null;
    int length = tag.length();
    StringBuffer encodedTag = new StringBuffer(2 * length);
    for (int i=0; i<length; i++) {
      char c = tag.charAt(i);
      if (c=='<')
        encodedTag.append("&lt;");
      else if (c=='>')
        encodedTag.append("&gt;");
      else if (c=='&')
        encodedTag.append("&amp;");
      else if (c=='"')
        encodedTag.append("&quot;");
        //when trying to output text as tag's value as in
        // values="???".
      else if (c==' ')
        encodedTag.append(" ");
      else
        encodedTag.append(c);
    }
    return encodedTag.toString();
  }
```

EncoderTag (4)

```
  public void setParent(Tag t) {
  }

  public void setPageContext(PageContext p) {
    pageContext = p;
  }

  public void release() {
  }

  public Tag getParent() {
    return null;
  }

  public int doStartTag() {
    return EVAL_BODY_BUFFERED;
  }
  // cannot manipulate body!
  public void setBodyContent(BodyContent bodyContent) {
    this.bodyContent = bodyContent;
  }
```


EncoderTag (5)

```
  public void doInitBody() {
  }

  public int doAfterBody() {
    String content = bodyContent.getString();
    try{
      JspWriter out = bodyContent.getEnclosingWriter();
      out.print(encodeHtmlTag(content));
    }
    catch(Exception e) {}

    return SKIP_BODY;
  }

  public int doEndTag() throws JspException {
    return EVAL_PAGE;
  }
}
```

Classes de suporte

- Para não ter de fornecer código para todos os métodos das interfaces, existem as classes
  - `TagSupport` para ser usada como classe base de *tag handlers*

    public class TagSupport implements IterationTag, java.io.serializable
  - `BodyTagSupport` para ser usada como super-classe pelos *tag handlers* que precisem de implementar a interface `BodyTag`

    public class BodyTagSupport extends TagSupport implements BodyTag

Exemplo 6: CapitalizerTag

```
package pacote;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class CapitalizerTag extends BodyTagSupport {

  public int doAfterBody() {
    String content = bodyContent.getString();
    try{
      JspWriter out = bodyContent.getEnclosingWriter();
      out.print(content.toUpperCase());
    }
    catch(Exception e) {}
    return SKIP_BODY;
  }

}
```