



**INSTITUTO  
FEDERAL**

Paraíba

Campus  
Cajazeiras

# PROGRAMAÇÃO P/ WEB 1

## 7. JSP

(BASEADO NO MATERIAL DO PROF. FABIO GOMES)

**PROF. DIEGO PESSOA**

✉ [DIEGO.PESSOA@IFPB.EDU.BR](mailto:DIEGO.PESSOA@IFPB.EDU.BR)

🐙 @DIEGOEP



**CST em Análise e  
Desenvolvimento  
de Sistemas**

# Introdução

- Servlets possuem algumas limitações importantes;
  - A geração do código HTML da página é feita dentro de classes java;
  - Muitas vezes, o desenvolvedor precisa exercer as funções de programador e web designer;
  - Páginas com conteúdo mais sofisticado são difíceis de serem criadas e mantidas;

# Introdução

- A tecnologia Java Server Pages (JSP) ajuda a resolver estas limitações;
  - Embora os servlets não tenham sido abandonados;
- Com ela, nós podemos embutir código java dentro de páginas HTML;

# Introdução

- Assim, um web designer pode projetar todo o conteúdo estático da página;
  - Resolvendo todas as questões referentes ao layout da mesma, como cores, tipos de fonte, estilo, etc;
  - O programador apenas escreve o código java para os trechos que necessitam de conteúdo dinâmico;

# Introdução

- Ferramentas CASE podem ser usadas para facilitar o desenvolvimento das páginas;
- O desenvolvimento da aplicação se torna bem mais rápido;
  - E bem mais fácil;

## O Ciclo de Vida de uma Página JSP

- Quando a página JSP é solicitada pela primeira vez no servidor:
  - A página é convertida em um servlet;
  - Uma thread é alocada para tratar a solicitação;
  - A solicitação é processada;
  - O resultado gerado pelo servlet é impresso na página JSP;

# O Ciclo de Vida de uma Página JSP

- A página é convertida para servlet:
  - Neste momento, o código JSP embutido na página que está sendo acessada é traduzido para um servlet pelo container;
  - É nessa hora que os erros de compilação são detectados;
  - Caso a compilação seja feita com sucesso, o servlet é instanciado e carregado na memória;



## O Ciclo de Vida de uma Página JSP

- A página é convertida para servlet:
  - A compilação da página é realizada apenas uma vez;
    - ✓ Futuras requisições usam o servlet gerado na primeira requisição;
  - Alguns containers permitem a pré-compilação das páginas;
    - ✓ Visando acelerar o tempo de resposta;



# O Ciclo de Vida de uma Página JSP

- Alocação e execução da thread:
  - Depois que o servlet é instanciado, o container aloca uma thread para tratar a solicitação do cliente;
  - Feito isso, o container chama o método **service** do servlet para executá-lo;

# O Ciclo de Vida de uma Página JSP

- Geração do resultado:
  - Terminada a execução da thread, a resposta gerada pelo servlet é impressa na página JSP;
  - Depois disso, a thread alocada para tratar a solicitação do usuário é destruída;
    - ✓ Mas o servlet correspondente à página permanece instanciado para tratar as demais solicitações;

# Escopos

- JSP usa diversos escopos para a utilização de alguns componentes;
  - `application;`
  - `request;`
  - `session;`
  - `page;`

# Escopos

- A classe `PageContext`:
  - Usada para implementar o escopo page;
  - Ela encapsula as várias informações referentes à uma página JSP;
    - ✓ Atributos, requisição, contexto da aplicação, configuração do servlet, JSP Writer, etc;
  - Toda página JSP tem uma referência para um objeto deste tipo;

# Escopos

- A classe `PageContext`:
  - Os principais métodos desta classe são:
    - ✓ `getOut()`;
      - Recupera uma referência ao `JSPWriter`, que é o objeto usado para a geração da página de resposta;
    - Funciona de forma similar ao método `getWriter` usado em um servlet;

# Escopos

- A classe PageContext:
  - Os principais métodos desta classe são:
    - ✓ `getRequest();`
      - ▶ – Recupera a referência para a requisição do usuário;
    - ✓ `getResponse();`
      - ▶ – Recupera a referência para a resposta que será retornada ao usuário

# Escopos

- A classe `PageContext`:
  - Os principais métodos desta classe são:
    - ✓ `getSession();`
      - Obtém a referência para o objeto que representa a sessão do usuário;
    - ✓ `getServletContext ();`
      - Obtém a referência para o objeto que encapsula informações sobre a aplicação na qual a página está sendo executada;



# Escopos

- A classe PageContext:
  - Os principais métodos desta classe são:
    - ✓ `getAttribute(String attributeName):`
      - Recupera o valor de um atributo do escopo da página onde o método está sendo chamado;
      - Atributos são sempre representados como `Object`;
      - O valor `null` é retornado caso o nome seja inválido;

# Escopos

- A classe `PageContext`:
  - Os principais métodos desta classe são:
    - ✓ `getAttribute(String attributeName, int scope)`:
      - Recupera o valor de um atributo presente em um determinado escopo;
    - A classe `PageContext` define constantes que podem ser usadas como o valor do escopo;
      - » `APPLICATION_SCOPE`, `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE`;

# Escopos

- A classe `PageContext`:
  - Obtendo um atributo a partir do contexto da aplicação;

```
Object userValue = pageContext.getAttribute("user", PageContext.APPLICATION_SCOPE);  
String userValueStr = userValue.toString();
```

# Escopos

- A classe `PageContext`:
  - Os principais métodos desta classe são:
    - ✓ `setAttribute(String name, Object value)`:
      - Define/altera o valor de um atributo definido no escopo da página;
    - ✓ `setAttribute(String name, Object value, int scope)`:
      - Define/altera o valor de um atributo em um determinado escopo;

# Escopos

- A classe `PageContext`:
  - Os principais métodos desta classe são:
    - ✓ `findAttribute(String attributeName)`:
      - Recupera o valor de um determinado atributo, procurando em todos os escopos possíveis;
  - A ordem dos escopos usados na consulta: página, requisição, sessão e aplicação

# Java Server Pages

- Páginas JSP tradicionalmente podem conter:
  - Template data;
  - Diretivas;
  - Elementos de scripting:
    - ✓ Scriptlets, expressões e declarações;
  - Ações;
  - Tags customizadas;

# Template Data

- Representa qualquer tipo de informação que não é conhecida pelo *container* JSP;
- Exemplos de dados template incluem HTML, XHTML, XML, SVG, etc;



# Diretivas

- Uma diretiva é um recurso usado para dar instruções especiais ao container no momento da criação da página;
- JSP define três tipos de diretivas:
  - `page`, `include` e `taglib`;
- Diretivas são declaradas entre os símbolos `<%@ %>`

# A Diretiva Page

- Nesta diretiva, definimos as informações necessárias para a geração da página JSP, como:
  - Os pacotes que devem ser importados;
  - O tipo de codificação de caracteres que deve ser utilizado;
  - O tipo de conteúdo da resposta;
  - A referência para a página de erro;

# A Diretiva Page

- Os atributos mais importantes da diretiva page são:
  - `import`:
    - ✓ Especifica os pacotes que devem ser importados para a execução dos comandos existentes na página;
    - ✓ Os nomes dos pacotes devem ser separados por vírgulas;

# A Diretiva Page

- Os atributos mais importantes da diretiva page são:
  - `contentType`:
    - ✓ Especifica o tipo do conteúdo que vai ser gerado;
    - ✓ O valor default deste atributo é `text/html`;

# A Diretiva Page

- Os atributos mais importantes da diretiva page são:
  - **isErrorPage:**
    - ✓ Especifica se a página atual é uma página de erro;
    - ✓ Caso afirmativo, a página tem acesso ao objeto que referencia a exceção capturada pela mesma;
    - ✓ O valor default deste atributo é false;

# A Diretiva Page

- Os atributos mais importantes da diretiva page são:
  - **errorPage:**
    - ✓ Especifica a página de erro que deve tratar alguma exceção ocorrida na página;
    - ✓ Se ocorrer uma exceção durante a geração da página, a sessão do cliente é redirecionada para a página definida neste atributo;

# A Diretiva Page

- Os atributos mais importantes da diretiva page são:
  - **session:**
    - ✓ Atributo booleano que indica se o controle de sessões deve ser habilitado;
    - ✓ O seu valor default é true;



# A Diretiva Page

- **Exemplo 1:**

- Especifica que a interface `Iterator` será usada na página;

```
<%@ page import="java.util.Iterator"%>
```

- **Exemplo 2:**

- Especifica que os pacotes `java.util` e `myapplication.source` devem ser importados;

```
<%@ page import="java.util.*,myapplication.source.*"%>
```

# A Diretiva Page

- Exemplo 3:

```
<%@ page contentType="text/xml" errorPage="erro.htm"%>
```

- Especifica que a página vai retornar um documento XML
- E que, caso ocorra uma exceção durante a sua geração, a sessão deve ser redirecionada para a página erro.htm;

# A Diretiva Page

- Exemplo 4:

```
<%@ page contentType="text/html" isErrorPage="true"%>
```

- Especifica que a página que está sendo descrita é uma página de erro;

# A Diretiva Include

- Define textos e códigos que devem ser incorporados à página;
- O container insere o conteúdo do arquivo no local no qual se encontra a diretiva;
  - Usada para construir trechos de página reutilizáveis, como menus, cabeçalhos, etc;

# A Diretiva Include

- É composta por um atributo chamado **file**, no qual definimos o arquivo que deve ser incluído na página;
- A inclusão é feita em tempo de compilação;
- Exemplo: `<%@ include file="header.htm"%>`

# A Diretiva Taglib

- Define as bibliotecas de tags usadas para a geração da página;
- Este recurso é usado quando trabalhamos com a tecnologia Java Server Pages Standard Tag Library;
- Falaremos sobre este tipo de diretiva nos próximos capítulos do curso;

# Scriptlets

- Scriptlets são trechos de código java embutidos em uma página JSP;
- Uma página JSP pode ter várias seções de scriptlets;
- Dentro de um scriptlet podemos especificar vários comandos java;



# Scriptlets

- São definidos entre as tags `<% %>`
- Importante: Os pacotes que contém as classes usadas nestes trechos de código devem ser declarados no atributo `import` da diretiva `page`;
- Falaremos novamente sobre scriptlets mais adiante;

# Scriptlets

- Exemplo de um trecho de scriptlet:

```
<%  
    int soma = 0;  
    Iterator it = numbers.iterator();  
    while(it.hasNext())  
        soma += Integer.parseInt(it.next().toString());  
%>
```

# Expressões

- Usadas para imprimir o resultado de alguma expressão java na página JSP;
  - O valor de uma variável, o resultado de um método, etc;
- A expressão é usada como se fosse o parâmetro de entrada para o método println do fluxo de saída do servlet;

# Expressões

- O container imprime o resultado da expressão na página no local onde a tag é encontrada;
- Expressões são definidas entre as tags  
`<%= %>;`

# Expressões

- Exemplos de expressões:

```
<%= soma%>
```

```
<%= session.getId()%>
```

```
<%= soma + 15%>
```

```
<%= new java.util.Date() %>
```

# Expressões

- Importante:
  - Se a expressão for o resultado de um método, ele não pode ter o tipo de retorno void;
  - Não se usa ponto-e-vírgula ao fim da expressão;

# Declarações

- Usadas para declarar variáveis e métodos usados ao longo da página;
- As informações declaradas nessas seções podem ser usadas pelos demais elementos de scripting da página;

# Declarações

- Declarações não geram qualquer saída para impressão no fluxo de saída da página;
- As declarações são especificadas através da construção `<%! %>`



# Declarações

- Exemplos de declarações:

```
<%! int i; %>
```

```
<%! int i = 0; %>
```

```
<%! public String f(int i) {if (i<3) return("..."); ... } %>
```

# Ações

- Realizam uma determinada tarefa dentro da página;
- São classificadas em dois tipos:
  - Padrão;
  - Não padrão;

# Ações

- Ações padrão:
  - São ações pré-definidas pela especificação JSP;
  - Podem ser usadas diretamente nas páginas JSP, sem a necessidade de declarar ou usar qualquer biblioteca específica;

# Ações

- Ações não padrão:
  - São definidas através de tags customizadas;
  - Estas tags são criadas pelo desenvolvedor da aplicação;
  - Falaremos sobre estes tipos de ação nos próximos capítulos;

# A Ação Include

- Usada para incluir recursos estáticos ou dinâmicos no conteúdo da página;
  - Semelhante ao método include de um despachante usado com servlets;
  - O conteúdo do recurso especificado é incorporado ao fluxo de saída de uma página

# A Ação Include

- O atributo page é usado para identificar o recurso que deve ser adicionado à página;
- O caminho do recurso deve ser informado como uma URL;
  - Que pode mapear para uma página HTML, JSP, um servlet, etc;

# A Ação Include

- URLs relativas também podem ser utilizadas;
  - Neste caso, elas são avaliadas com relação ao diretório no qual a página corrente está localizada;

# A ação include

```
<body>
  <p align="center">
    <font color="#0000FF" size="2" face="Verdana, Arial, Helvetica, sans-serif">
      <strong>Oi, seja bem vindo à minha home page!</strong></font>
    </p>

    <p></p>

    <p align="left"> <jsp:include page="pessoal.htm" /></p>

    <p></p>

    <p align="left"> <jsp:include page="passatempo.htm" /></p>

    <p></p>

    <p align="left"> <jsp:include page="esportes.htm" /></p>
</body>
```



# A Ação Forward

- Usada para despachar a requisição do cliente para outro recurso;
  - Semelhante ao método forward de um despachante usado com servlets;
- O recurso para o qual a requisição é despachada obtém controle total sobre a mesma;

# A Ação Forward

- O recurso para o qual a requisição deve ser despachada é definido através do atributo **page**;
  - Cujo valor é processado de forma semelhante ao atributo **page** da ação **include**;
- Quando a ação **forward** é executada, a geração da página é automaticamente interrompida;

# A Ação Forward

- Exemplos de aplicação da ação forward:

```
<jsp:forward page="pessoal.htm"/>  
<jsp:forward page="processaRequisicao"/>
```

# A Ação useBean

- Ação que nos permite trabalhar com objetos java dentro da página JSP;
- Os objetos usados com esta ação precisam ser oferecidos na forma de um JavaBean;
- Estas ações são normalmente usadas junto com uma ação `getProperty` ou `setProperty`;

# A Ação useBean

- O que são JavaBeans?
  - São componentes que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento;
  - Na prática, são classes java que possuem algumas restrições;

# A Ação useBean

- Restrições de classes que representam JavaBeans:
  - Devem ser serializáveis;
  - Devem ter um construtor default;
  - Devem oferecer métodos get e set para todos os seus atributos;

# A Ação useBean

- Atributos da ação useBean:
  - id:
    - ✓ Representa a identificação do objeto na aplicação;
    - ✓ A combinação id/escopo deve ser única na aplicação;
    - ✓ Este nome é usado para referenciar este objeto ao longo da página;
      - ▶ – Por exemplo, em expressões e scriptlets;

## Exemplo de um JavaBean

```
public class User {  
    private String name;  
    private int age;  
    private String course;  
    private String city;  
    private String state;  
  
    public String getName() { ...3 linhas }  
  
    public void setName(String name) { ...3 linhas }  
  
    public int getAge() { ...3 linhas }  
  
    public void setAge(int age) { ...3 linhas }  
  
    public String getCourse() { ...3 linhas }  
  
    public void setCourse(String course) { ...3 linhas }  
  
    public String getCity() { ...3 linhas }  
  
    public void setCity(String city) { ...3 linhas }  
  
    public String getState() { ...3 linhas }  
  
    public void setState(String state) { ...3 linhas }  
}
```



# A Ação useBean

- Atributos da ação useBean:
  - scope:
    - ✓ Define o escopo relacionado ao objeto;
    - ✓ Esta informação define a visibilidade e o ciclo de vida do objeto;
    - ✓ O valor deste atributo deve ser um dos escopos usados em páginas JSP;

# A Ação useBean

- Atributos da ação useBean:
  - class:
    - ✓ Representa a classe a qual o bean está relacionado;
    - ✓ O valor deste atributo deve ser o nome da classe totalmente qualificado;

# A Ação useBean

- Ao ver uma ação useBean, o container verifica se já existe um objeto com o id e o escopo especificado na mesma;
  - Em caso afirmativo, ele obtém esta referência, fazendo o casting para o tipo especificado;
  - Em caso negativo, ele cria uma nova instância para o objeto;

# A Ação useBean

- Exemplo:

```
<jsp:useBean id="myUser" class="mypackage.User" scope="session" />
```

- O exemplo indica que:
  - ✓ O container dever recuperar (ou criar) um objeto chamado myUser;
  - ✓ O objeto é uma instância da classe User do pacote mypackage;
  - ✓ O objeto tem escopo de sessão;

## A Ação `setProperty`

- Ação utilizada para alterar o estado de um bean da aplicação;
- Ao encontrar uma ação deste tipo, o container chama o método `set` que altera o valor do atributo desejado;
- Esta ação é sempre usada em conjunto com a ação `useBean`;

# A Ação setProperty

- O valor de uma propriedade pode ser alterado de três formas:
  - A partir de um parâmetro da requisição, um string literal ou uma expressão avaliada em tempo de execução;
- O valor desejado é sempre passado como um string;
  - O container se encarrega de converter o valor da propriedade para o tipo apropriado;

# A Ação setProperty

- Atributos da ação setProperty:
  - name:
    - ✓ Corresponde ao nome do bean que terá uma ou mais propriedades alteradas;
    - ✓ Deve ter o nome de algum bean declarado por uma ação useBean da página;

# A Ação SetProperty

- Atributos da ação SetProperty:
  - value:
    - ✓ Atributo opcional no qual podemos especificar diretamente o novo valor que deve ser atribuído ao atributo;
    - ✓ Usamos este atributo quando o valor da propriedade vai ser definido através de um string ou de uma expressão;



# A Ação setProperty

- Atributos da ação setProperty:
  - **property:**
    - ✓ Corresponde ao nome da propriedade cujo valor será alterado;
  - ✓ Podemos atribuir o valor \* para indicar que todos os atributos devem ser alterados;
    - Neste caso, o container casa os nomes dos parâmetros de requisição com os nomes das propriedades do bean;

# A Ação setProperty

- Atributos da ação setProperty:
  - param:
    - ✓ Corresponde ao nome do parâmetro da requisição cujo valor deve ser associado à propriedade;
    - ✓ Se o seu valor for omitido, o container considera que o nome do parâmetro é igual ao da propriedade;

# A Ação setProperty

- Exemplo 1:

```
<jsp:setProperty name="user" property="name" param="username" />
```

- Significa que a propriedade name do objeto user deve ser alterada. O valor deve ser obtido através do parâmetro username da requisição;

# A Ação setProperty

- Exemplo 2:

```
<jsp:setProperty name="user" property="*" />
```

- Significa que todas as propriedades do objeto user devem ser alteradas, e que seus respectivos valores devem ser obtidos através dos parâmetros enviados pela requisição;

# A Ação setProperty

- Exemplo 3:

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

- Significa que a propriedade “name” do objeto user deve ser alterada. O seu valor será determinado através da variável “name”, e será conhecido em tempo de execução;

# A Ação setProperty

- Exemplo 4:

```
<jsp:useBean id="user" class="jsp.User">  
  <jsp:setProperty name="user" property="name" value="fabio"/>  
  <jsp:setProperty name="user" property="age" value="25"/>  
  <jsp:setProperty name="user" property="cpf" value="9999999"/>  
</jsp:useBean>
```

- Significa que as propriedades “name”, “age” e “cpf” do objeto user devem ser alteradas. Os seus respectivos valores devem ser “fabio”, “25” e “9999999”;

## A Ação getProperty

- Usada para recuperar o valor de um atributo de um bean;
- Usamos esta ação para imprimir o valor deste atributo na página JSP;
- O valor do atributo é sempre convertido para String antes de ser utilizado;

## A Ação getProperty

- Assim como a ação setProperty, esta ação é normalmente usada em conjunto com uma ação useBean;
- Caso o objeto indicado não seja encontrado, uma exceção é lançada;



# A Ação getProperty

- Atributos da ação getProperty:
  - name:
    - ✓ Indica o nome do objeto que será utilizado;
    - ✓ O nome deve ter o valor de um java bean que foi declarado em uma ação useBean;
  - property:
    - ✓ Indica o nome do atributo que deve ser recuperado;

# A Ação getProperty

- Exemplo 1:

```
<jsp:getProperty name="client" property="name"/>
```

- Significa que neste local da página deve ser impresso o valor do atributo “name” do objeto client;

# Exemplo

- Vamos ver um exemplo de uso das ações `useBean`, `setProperty` e `getProperty`;
- Neste exemplo, o usuário preenche um formulário cujos campos correspondem aos atributos de um java bean;

## Exemplo

### Exemplo

- Os dados são enviados para uma página JSP de confirmação dos dados;
- A página instancia o bean, altera as suas propriedades e depois as exibe para o usuário

## Visão do formulário

Usuário

+

—

□

×

←

→

i

127.0.0.1:8080/psd-jsp/formulario.html

↻

🔍 Pesquisar

🔧

☆

📅

📧

↓

🏠

💬

🌐

☰

*Entre com os seus dados:*

Nome:

Idade:

Curso:

Cidade:

Estado:

PB

▼

Enviar

## Página JSP que processa a requisição

```
<jsp:useBean id="newUser" class="jsp.User" scope="session" />
<jsp:setProperty name="newUser" property="*" />

<html>
<head>
<title>Confirmação das informações</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<p>Confirmando os seus dados: </p>
<p><b>Nome:</b> <jsp:getProperty name="newUser" property="name" /></p>
<p><b>Idade:</b> <jsp:getProperty name="newUser" property="age" /></p>
<p><b>Curso:</b> <jsp:getProperty name="newUser" property="course" /></p>
<p><b>Cidade:</b> <jsp:getProperty name="newUser" property="city" /></p>
<p><b>Estado:</b> <jsp:getProperty name="newUser" property="state" /></p>
</body>
</html>
```



```
<jsp:useBean id="client" class="jsp.Client" scope="session" />
```

```
<html><head><title>Mostra Cliente</title></head>
```

```
<body>
```

```
<p>
```

```
  <font color="#0000FF" size="2" face="Verdana, Arial, Helvetica, sans-serif">
```

```
    <em>
```

```
    <strong>Dados informados:</strong>
```

```
    </em>
```

```
  </font></p>
```

```
<p>
```

```
  <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
```

```
    <strong>Nome:</strong> <jsp:getProperty name="client" property="name"/> </font>
```

```
</p>
```

```
<p>
```

```
  <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
```

```
    <strong>Idade:</strong> <jsp:getProperty name="client" property="age"/> </font>
```

```
</p>
```

```
<p>
```

```
  <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
```

```
    <strong>CPF:</strong> <jsp:getProperty name="client" property="cpf"/></font>
```

```
</p>
```

```
</body></html>
```

# Trabalhando com Scriptlets

- Um scriptlet é um trecho de código java definido em uma página JSP;
  - São definidos entre as tags `<%` e `%>`;
- Podem ser usados de duas formas:
  - Apenas um trecho de código java;
  - Aplicação de um código java a um código HTML;



# Trabalhando com Scriptlets

- Na primeira forma de aplicação, o scriptlet é usado apenas para execução de um trecho de código Java.
- Neste caso, o trecho é contido dentro de um único bloco `<% %>;`

# Trabalhando com Scriptlets

- Exemplo:

```
<%  
    int number = Integer.parseInt(request.getParameter("number"));  
    int quantity = Integer.parseInt(request.getParameter("quantity"));  
%>
```

# Trabalhando com Scriptlets

- Na segunda forma, o scriptlet é usado para gerar parte do código do documento que será enviado na resposta;
- Nestes casos, o trecho de scriplet é dividido em dois ou mais scriptlets;

# Trabalhando com Scriptlets

- Exemplo:

```
<%  
    String number = request.getParameter("number");  
    int num = Integer.parseInt(number);  
    String quantity = request.getParameter("quantity");  
    int quant = Integer.parseInt(quantity);  
%>  
<html><head><title>Exemplo da Tabuada</title></head>  
  
<body>  
    <p align="center">A tabuada do número <%=num%> </p>  
    <%  
        for(int k=1;k<=quant;k++){  
    %>  
    <p><%=num%> * <%=k%> = <%=num * k%></p>  
    <% } %>  
</body>  
</html>
```

# Trabalhando com Scriptlets

- Note que neste caso, o comando for é aplicado a uma série de comandos HTML;
  - Sempre que uma iteração for executada, o bloco HTML será impresso na página;
- Veja também que apenas o trecho java fica dentro do scriptlet;
  - Quando vamos aplicar a um comando HTML, fechamos o trecho do scriptlet;

# Trabalhando com Scriptlets

- Exemplo:

```
<%  
    if(number % 2 ==0 ) {  
%>  
    <p> <h1>O número eh par </h1> </p>  
  
<%  
    } else {  
%>  
    <p> <h1>O número eh par </h1> </p>  
  
<% } %>
```

## Trabalhando com Objetos Implícitos

- Quando uma página JSP é criada, ela possui alguns objetos que podem ser diretamente manipulados pela mesma;
- Este objetos são criados pelo container no momento em que a página é gerada;
- Estes objetos são conhecidos como objetos implícitos;

## Trabalhando com Objetos Implícitos

- Objetos implícitos representam uma forma simples de se acessar os objetos relacionados a um PageContext;
- Objetos implícitos podem ser usados dentro de scriptlets ou de expressões;



## Trabalhando com Objetos Implícitos

- Os objetos implícitos que o container cria para toda página JSP são:
  - `out (JSPWriter);`
  - `request (HttpServletRequest);`
  - `response (HttpServletResponse);`
  - `session (HttpSession);`
  - `application (ServletContext);`

## Trabalhando com Objetos Implícitos

- Os objetos implícitos que o container cria para toda página JSP são:
  - `config (ServletConfig);`
  - `exception (JSPException);`
  - `pageContext (PageContext);`
  - `page (Object);`

# Trabalhando com Objetos Implícitos

- Exemplo:

```
<%  
    String name = request.getParameter("name");  
    String id = request.getParameter("cpf");  
    User newUser = new User (name, id);  
    session.setAttribute("user", newUser);  
%>
```

## Trabalhando com Objetos Implícitos

- No exemplo anterior:
  - Note que os objetos `request` e `session` são usados diretamente:
    - ✓ Os objetos não são declarados dentro do scriptlet;
    - ✓ Não existe nenhum código inicializando estes objetos;
    - ✓ Os objetos não são declarados em nenhum import;

# Trabalhando com Objetos Implícitos

- Exemplo:

```
<%  
    String option = request.getParameter("option");  
    if(option.equals("esportes"))  
        response.sendRedirect("http://globoesporte.com");  
    else if(option.equals("noticias"))  
        response.sendRedirect("http://www.uol.com.br");  
    else response.sendRedirect("http://www.uol.com.br");  
%>
```

# Configuração de Páginas JSP

- Podemos definir algumas características das páginas JSP no arquivo web.xml;
- As informações de configuração podem ser aplicadas a uma página específica ou a um grupo de páginas;
- Esta configuração deve ser feita através do elemento `jsp-property-group`;

# Configuração de Páginas JSP

- O elemento url-pattern:
  - Usamos este elemento para definir a página (ou o grupo de páginas) a qual a configuração deve ser aplicada;
  - O padrão é definido através da extensão do arquivo;

# Configuração de Páginas JSP

- O elemento url-pattern:
  - Exemplos:

```
<jsp-property-group>  
  <url-pattern>mypage.jsp</url-pattern>  
</jsp-property-group>  
  
<jsp-property-group>  
  <url-pattern>*.jsp</url-pattern>  
</jsp-property-group>
```



# Configuração de Páginas JSP

- O elemento url-pattern:
  - Exemplos:

```
<jsp-property-group>  
  <url-pattern>/estoque/*.jsp</url-pattern>  
</jsp-property-group>
```

# Configuração de Páginas JSP

- O elemento `el-ignored`:
  - Usamos para definir se as expressões da EL devem ser ignoradas;
  - Se o seu valor for `true`, as expressões da EL serão ignoradas;
  - O seu valor default é `false` a partir da versão 2.4 do `web.xml`;
    - ✓ E `true` nas versões anteriores;

# Configuração de Páginas JSP

- O elemento `el-ignored`:
  - Esta informação também pode ser configurada diretamente na página JSP, através do atributo `isELIgnored` da diretiva `page`;

```
<jsp-property-group>  
  <url-pattern>*.jsp</url-pattern>  
  <el-ignored>true</el-ignored>  
</jsp-property-group>
```

# Configuração de Páginas JSP

- O elemento `scripting-invalid`:
  - Usamos este elemento para controlar a validade de elementos de scripting descritos na página;
  - Caso o seu valor seja definido como `true`, trechos de scriptlets são considerados inválidos;
    - ✓ Um erro é lançado na hora da tradução da página;

# Configuração de Páginas JSP

- O elemento `scripting-invalid`:
  - Exemplo:

```
<jsp-property-group>  
  <url-pattern>*.jsp</url-pattern>  
  <scripting-invalid>true</scripting-invalid>  
</jsp-property-group>
```

# Configuração de Páginas JSP

- O elemento `page-encoding`:
  - Usamos este elemento para definir o tipo de codificação que deve ser usada para a geração da página;
  - Este valor também pode ser codificado através do atributo `pageEncoding` da diretiva `page`;

# Configuração de Páginas JSP

- O elemento `include-prelude`:
  - Usamos este elemento para definir conteúdos que devem ser incluídos na página JSP;
  - O seu valor deve indicar o caminho de um componente presente na aplicação;
  - O conteúdo é incluído no início da página JSP;

# Configuração de Páginas JSP

- O elemento `include-coda`:
  - Usamos este elemento para definir conteúdos que devem ser incluídos na página JSP;
  - Entretanto, neste elemento, o conteúdo é adicionado no fim da página JSP;



# Configuração de Páginas JSP

- Exemplo:

```
<jsp-property-group>  
  <url-pattern>*.jsp</url-pattern>  
  <include-prelude>/prelude1.html</include-prelude>  
  <include-coda>/coda1.html</include-coda>  
</jsp-property-group>
```