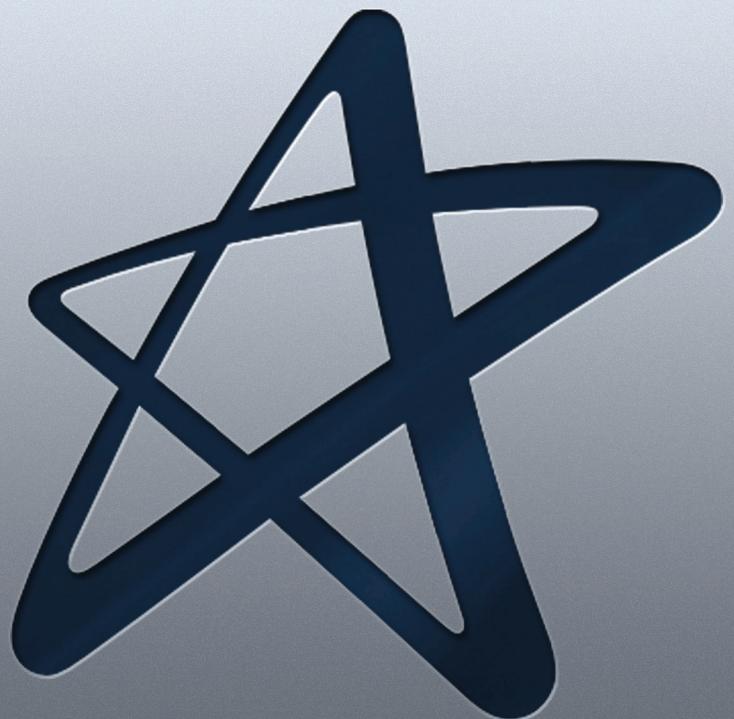


# **Sistema Servidor Cliente**



**Educação a Distância**  
Cruzeiro do Sul Educacional  
**Campus Virtual**



# Material Teórico



Programação de aplicações cliente/servidor na WEB com JSP

**Responsável pelo Conteúdo:**

Prof. Esp. Marcio Funes

**Revisão Técnica:**

Prof. Ms. Luiz Carlos Reis

**Revisão Textual:**

Profa. Ms. Claudio Brites



# UNIDADE

## Programação de aplicações cliente/ servidor na WEB com JSP



- Introdução
- Utilizando Scriptlet em JSP
- Criando Scripts dinâmicos em JSP
- Conclusão



### Objetivo de APRENDIZADO

- Nesta unidade, trabalharemos o seguinte tópico de conteúdo: Conceitos Iniciais: uma visão geral sobre a arquitetura Java sendo aplicada de forma dinâmica na Web. Veremos também o papel de Servelet e Servetlet para utilizarmos JavaServer Pages em busca de soluções Web.

Nesta unidade, veremos alguns assuntos introdutórios da nossa disciplina e aproveitaremos para apresentar a você alguns conceitos que utilizaremos na estrutura de todas as nossas unidades.

Lembramos você da importância de realizar todas as atividades propostas dentro do prazo estabelecido para cada unidade, pois, dessa forma, você evitará que o conteúdo se acumule e que você tenha problemas ao final do semestre.

Uma última recomendação: caso tenha problemas para acessar algum item da disciplina ou dúvidas em relação ao conteúdo, não deixe de entrar em contato com seu professor tutor através do campo de mensagens.

## Contextualização

Seja em qualquer ramo, o mercado sempre está em busca de novidades – novos produtos, novos serviços, novos profissionais e novas tecnologias. A novidade sempre pode trazer uma oportunidade de negócio antes não vista.

Pensando nisso, em meados de 1990, os primeiros passos para a tecnologia Java foram dados – porém, bem longe da Indonésia que, apesar de possuir a “Ilha de Java” em sua geografia e ter forte presença do Budismo (que poderia ajudar os programadores a ficarem mais tranquilos quando aquele código não queria compilar), não foi o berço dessa tecnologia.

Tudo aconteceu na Sun Microsystems, quando uma equipe tinha por intenção desenvolver um aplicativo portátil, que teria a capacidade de rodar em qualquer microchip – inicialmente usando como base a linguagem C/C++. Contudo, os desenvolvedores perceberam que algumas limitações da linguagem não permitiriam que o app se tornasse 100% portátil, como o projeto necessitava.

Figura 1 – Estátua de Buda presente na Ilha de Java na Indonésia.



Thinkstock/Getty Images

Perceba, nesse exemplo, que, quando o mercado deseja criar uma oportunidade de negócio, a limitação de uma tecnologia é solucionada com a inovação daquilo que já existe. Com isso, a Sun removeu algumas características e alterou outras, criando uma nova linguagem de programação, batizada como Oak (carvalho, em português).

Em 1995, a internet já se tornava popular e a busca de inovações nessa área também ocorria, como em muitas outras anteriores. Por conta disso, a linguagem Oak, por ter desde o início de sua concepção a possibilidade de rodar em várias plataformas, era perfeita, já que a internet também possuía essa característica (rodar em várias plataformas).

Com o sucesso iminente, a Sun renomeou a linguagem para Java, alterou algumas características para se adaptar à realidade do mercado e lançou no mesmo ano sua nova solução.

Ainda na mesma ideia de inovação para trazer novidades para o mercado, viu-se a possibilidade de criar uma tecnologia que se relacionava com uma dificuldade na área de web: criar páginas dinâmicas e de fácil implementação sem abrir mão das funções essenciais que o Java podia oferecer a um site. Nesse contexto, surgi a tecnologia JSP (JavaServer Pages), que vem exatamente permitir esse dinamismo, podendo inserir trechos com implementações Java, trazendo assim as funções desejadas dentro de uma linguagem já conhecida e muito utilizada na internet: o HTML.

Podemos, então, ver que a tecnologia JSP tem por foco trazer inovação para mercado e é baseada em Java, que alguns anos antes de sua criação teve o mesmo foco: quebrar barreiras para trazer novas soluções tecnológicas antes não disponíveis.

## Introdução



A evolução está presente na humanidade, temos evidências históricas de sua origem, o próprio conceito de evolução só é possível devido à inteligência adquirida por nós seres humanos até hoje para cunhá-lo. Sabemos que essa evolução nos trouxe a necessidade de criar ferramentas que aperfeiçoassem nosso trabalho – e otimizar os processos humanos é uma das bases da computação.

Ainda dentro da computação, vemos novos conceitos, ferramentas e técnicas que surgem para melhorar a interação que temos com o computador. Um dos conceitos criados para permitir maior dinamismo ao se criar soluções para a web é a tecnologia chamada JSP, abreviação de JavaServer Pages (KOLB 2000).

Como você pode notar, a palavra Java não está por coincidência no nome JavaServer Pages. JSP é baseada na tecnologia Java; mas que “apenas” baseada, o JSP tem por objetivo simplificar o processo de desenvolvimento para web sem deixar o dinamismo muito procurado em sites de internet de fora. As possibilidades que a tecnologia Java pode trazer são incríveis, por esse motivo surge a necessidade de inserir os recursos que o Java pode fornecer na estrutura de linguagens já conhecidas por programadores web (KOLB 2000).

Figura 1 – Comparando linguagens.



Fonte: [vidadeprogramador.com.br](http://vidadeprogramador.com.br)

## Utilizando Scriptlet em JSP



Para entendermos melhor como a tecnologia JSP funciona, precisamos sempre ter em mente que JSP (JavaServer Page) utiliza a arquitetura Java como base e por isso trabalha utilizando seus conceitos. Dentro da arquitetura Java temos os Servlets, entender seu funcionamento nos ajuda a compreender melhor a função do JSP e como ambos trabalham juntos.

### Servlets

Qualquer usuário da internet atual, depois de visitar algumas páginas, já percebe que a maioria delas fornece algum tipo de serviço; diferente do início dos anos 90, onde a maioria das páginas era simplesmente estática e não previa suporte à interação com o usuário. Hoje vemos a internet como fonte de negócio e, portanto, sinônimo de serviços.

Uma das ideias iniciais para transformar páginas estáticas em dinâmicas era prover o dinamismo por meio de um servidor que interpretaria a solicitação do usuário e devolveria a ele uma página resposta. Desse conceito surgiu uma tecnologia chamada CGI, na qual era possível escrever códigos que não rodariam em um navegador como C/C++, eles teriam o servidor como ponte interligando a função que o usuário desejasse com a funcionalidade contida no código que estava no servidor (TODD; SZOLKOWSKI 2003).

Em 1997, na tecnologia Java, esse conceito foi chamado se Servlet, que vem da ideia de um pequeno servidor que tem por objetivo interpretar e compilar as solicitações do usuário através de páginas web, procurar as funções solicitadas em suas classes e códigos Java e gerar uma página dinâmica de resposta ao usuário.

Agora que vimos que podemos usar um Servlet para interpretar solicitações, podemos entender melhor como o JSP se encaixa nessa arquitetura: como possuímos códigos HTML que podem ser interpretados em qualquer navegador e temos os códigos Java que podem ser implementados em um Servlet, podemos conectar ambos criando um código que possua os benefícios de ambos.

### Scriptlet

Para que possamos utilizar o código Java dentro de uma JSP utilizamos essa sintaxe:

<% ... %>

Segundo Todd e Szolkowski, esse tipo de codificação é chamado de Scriptlet. É um nome que vem da palavra Script, que em inglês significa roteiro; adicionando o sufixo “let”, temos, em tradução livre, “scriptzinho”, ou seja, um pequeno script.

Assim, deixamos a convencional estrutura HTML, que utiliza a tag < ... />, e acrescentamos o % para que nosso Servlet consiga diferenciar o que é código Java e o que é HTML.

Vejamos um exemplo.

Vamos utilizar uma expressão Java muito comum para exibir texto, o `out.println`. Porém, vamos colocá-la em uma JSP:

### **Index\_exemplo.jsp**

```
<html>
  <head>
    <title> Pagina JSP</title>
  </head>
  <body>
    <% out.println("Ola Mundo"); %>
  </body>
</html>
```

Perceba que utilizamos o HTML em nossa JSP e embutimos um código exclusivo Java dentro da página usando o Scriptlet `<% ... %>`, assim utilizamos o que cada linguagem tem de melhor a oferecer.

## **Declaração de variáveis**

Uma das mais conhecidas funções de qualquer linguagem compilada é a possibilidade de criar variáveis. Como em HTML não temos essa possibilidade, podemos utilizar JSP e assim teremos a facilidade das tags HTML e as possibilidades da linguagem Java (KURNIAWAN, 2002).

Vejamos um exemplo:

### **Index\_exemplo2.jsp**

```
<html>
  <head>
    <title> Pagina JSP</title>
  </head>
  <body>
    <% String texto = "Ola Mundo"; %>
    <%-- Perceba que na linha acima colocamos "Ola Mundo" dentro da variável "texto" e
       a propósito eu sou um comentário--%>
    <% out.println(variavel); %>
    <%-- Agora podemos exibir o conteúdo da variável texto como codificado na linha
       acima--%>
  </body>
</html>
```

## Estruturas de programação

Assim como no exemplo anterior, podemos utilizar mais conceitos Java e criar estruturas de condição.

Vejamos um exemplo no qual podemos utilizar o IF e o ELSE para exibir o nome do mês atual por extenso:

### **index\_condicao.jsp**

```
<html>
    <head>
        <title> Pagina JSP</title>
    </head>
    <body>
        <%
        int dia = Integer.parseInt((new SimpleDateFormat("dd")).format(new Date()));
        int mes = Integer.parseInt((new SimpleDateFormat("M")).format(new Date()));
        int ano = Integer.parseInt((new SimpleDateFormat("yyyy")).format(new Date()));
        if(mes==1){
            out.println("Hoje é " + dia + " de Janeiro de " + ano);
        }else if(mes==02){
            out.println("Hoje é " + dia + " de Fevereiro de " + ano);
        }else if(mes==3){
            out.println("Hoje é " + dia + " de Março de " + ano);
        }else if(mes==4){
            out.println("Hoje é " + dia + " de Abril de " + ano);
        }else if(mes==5){
            out.println("Hoje é " + dia + " de Maio de " + ano);
        }else if(mes==6){
            out.println("Hoje é " + dia + " de Junho de " + ano);
        }else if(mes==7){
            out.println("Hoje é " + dia + " de Julho de " + ano);
        }else if(mes==8){
            out.println("Hoje é " + dia + " de Agosto de " + ano);
        }else if(mes==9){
            out.println("Hoje é " + dia + " de Setembro de " + ano);
        }else if(mes==10){
```

```
out.println("Hoje é " + dia + " de Outubro de " + ano);
}else if(mes==11){
out.println("Hoje é " + dia + " de Novembro de " + ano);
}else if(mes==12){
out.println("Hoje é " + dia + " de Dezembro de " + ano);
} %>
</body>
</html>
```

Assim como utilizamos estruturas de condição, podemos utilizar qualquer outra estrutura presente em Java, basta apenas conhecer qual as características que ela possui, utilizar Scriptlet para embutir o código Java dentro da JSP e utilizar os códigos HTML para de forma fácil estruturar a sequência do que será exibido ao usuário (KURNIAWAN, 2002).



Para saber mais sobre as características do JSP, acesse:

<http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro2.html>

Esse tutorial criado pela Oracle mostra passo a passo exemplos de utilizações de Servetlet dentro de JSP.

## Criando scripts dinâmicos em JSP



Como a palavra de ordem em JSP é o dinamismo, antes de ver os primeiros exemplos de codificação, devemos entender o funcionamento geral da tecnologia.

Vejamos um exemplo:

Quando queremos criar uma página HTML simples, utilizamos tags para mostrar ao navegador o que queremos exibir ao usuário. No código abaixo, por exemplo:

```
<HTML>
<BODY>
<b>testando tags</b>
</BODY>
</HTML>
```

Apenas mostrando a simples frase “testando tags” em negrito, porém, nosso objetivo é entender como inserir um código Java em meio ao HTML e assim criar JSP. Para isso, vamos inserir uma diretiva JSP.

## Diretivas de JSP

Segundo David (2010), diretivas são tags que fornecem informações ao servidor que irá interpretar o código Java contido no JSP. Elas direcionam o modo que o código será interpretado. Podemos utilizar três tipos de diretivas JSP: include, page e taglib.

Veja sua sintaxe:

```
<%@ nome_da_diretiva atributo(s)_da_diretiva %>
```

Veja que para caracterizar uma tag em JSP utilizamos o “`<%@ ... %>`”; diferente do modo de tag HTML, no qual utilizamos apenas “`< ... />`”.

## Utilizando o *include*

A diretiva include dentro de JSP tem por objetivo inserir o conteúdo de outro arquivo em uma JSP.

Vejamos um exemplo:

```
<%@ include file="endereço_do_arquivo" %> ou  
<jsp:directive.include file="endereço_do_arquivo"/>
```

Vamos pensar em uma aplicação do cotidiano envolvendo a diretiva include: uma empresa deseja dar as boas vindas sempre que alguém faz o acesso a sua página, porém, ela possui várias páginas diferentes e existe a possibilidade de o conteúdo de boas-vindas ser alterado em datas festivas – como natal ou páscoa, por exemplo. Nesse caso uma solução seria criar uma página de boas-vindas separada do código original de cada página. Assim, as páginas que irão exibir as boas-vindas teriam como inclusão o arquivo de boas-vindas que, quando alterado, exibiria a alteração onde foi incluído.

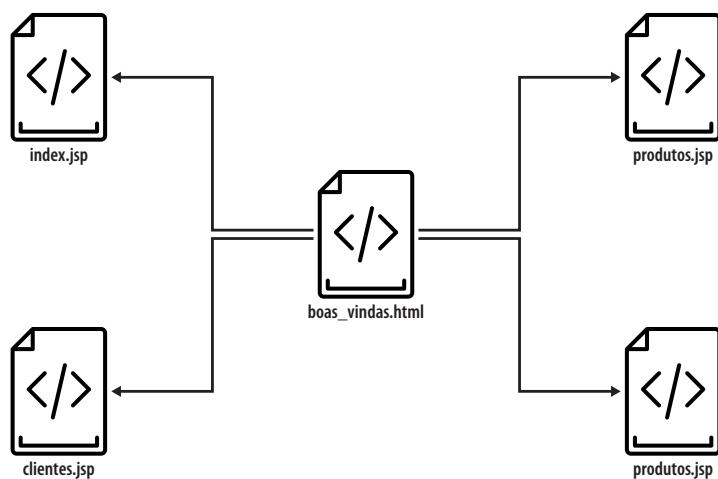


Figura 2 – Estruturas de páginas para include.

Fonte: Adaptado de iStock/Getty Images

Dessa forma, sempre teremos a possibilidade de mudar a página `boas_vindas.html` sempre que necessitamos sem nem precisar abrir o código de cada página, como podemos ver na figura 2. Vejamos agora como seria o código para essa solução:

## **boas\_vindas.html**

```
<HTML>
  <BODY>
    <td align="right" bgcolor="#FFFF99">
      <b>Seja bem vindo a nossa página</b>
    </td>
  </BODY>
</HTML>
```

Agora veja como podemos inserir o conteúdo do arquivo `boas_vindas.html` em nossa página `index.html`, como mostra a figura 2:

## **index.jsp**

```
<HTML>
  <BODY>
    <%@ include file="boas_vindas.html" %>
    Texto da página...
  </BODY>
</HTML>
```

Veja no exemplo que, ao utilizarmos a diretiva `include`, pudemos inserir todo o conteúdo que se encontrava dentro do arquivo `boas_vindas.html` em apenas uma linha de código por meio de uma tag, sendo assim, é como se o código de boas vindas estivesse presente no `index.jsp`. Com essa ação, podemos economizar linhas de código tornando a programação mais enxuta, criando um dinamismo entre as páginas de modo que, ao atualizarmos o arquivo `boas_vindas.html`, todas as páginas automaticamente estarão atualizadas com a alteração.

## **Diretiva Page**

Essa diretiva possui diversos atributos que dão informações variadas sobre a página em que está inserida (DAVID, 2010). Vejamos sua sintaxe:

```
<%@ Page atributo1="valor" atributo2="valor2" atributo3="valor3" ... %>
```

Ou podemos também utilizar essa sintaxe:

```
<jsp:directive.page atributo1="valor1" atributo2="valor2" ... />
```

Vejamos alguns exemplos:

## Atributo **Info**

Por meio dele podemos inserir informações sumarizadas da página:

```
<%@ Page info="Programação cliente/servidor na Web com JSP" %>
```

## Atributo **isErrorPage**

Usado para informar ao servidor que interpretará o código Java qual será a página para onde ele deve encaminhar o usuário – caso a página que esteja acessando possua algum erro –; ou seja, a página que conter essa diretiva se torna a página padrão de erro e sempre será exibida caso aconteça algum erro em outra página – esse atributo é do tipo booleano:

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

## Atributo **Language**

Atributo usado quando queremos especificar qual será a linguagem de criação de script. Em geral, utilizamos esse atributo para definir Java como linguagem geradora de scripts:

```
<%@ Page language="Java" %>
```

## Atributo **isThreadSafe**

Uma página JSP em geral responde a muitas solicitações; porém, em alguns casos, queremos controlar quais páginas deverão responder a essas solicitações, pois foram programadas para isso, e quais não devem. Para isso, usamos esse atributo booleano, com o qual podemos criar essa definição (TODD; SZOLKOWSKI, 2003).

```
<%@ Page isThreadSafe="false" %> ou <%@ Page isThreadSafe="true" %>
```

A diretiva Page possui muitos atributos e permite, por meio das funcionalidades que cada atributo possui, criar páginas de modos diferentes e, mais importante, gerenciar esses atributos moldando e alterando a solução sempre que for necessário de modo fácil e dinâmico (TODD; SZOLKOWSKI, 2003).

## Diretiva **Taglib**

Como podemos perceber através do “lib” de seu nome, a Taglib fornece um library de tags customizadas, que podemos utilizar em nossa JSP, ou seja, podemos criar nossas próprias tags em JSP – desenvolvimento, assim, nossa biblioteca de tags, que poderemos utilizar sempre que desejarmos e quando for necessário (TODD; SZOLKOWSKI, 2003).

Veja sua sintaxe:

```
<%@ taglib uri="endereço_da_bibliotecadetags" prefix="PrefixodaTag" %>
```

ou

```
<jsp:directive.taglib uri=" endereço_da_bibliotecadetags" prefix=" PrefixodaTag" />
```

Vejamos um exemplo da construção de uma solução JSP utilizando a diretiva Taglib para personalizar o formato de exibição da data atual no formato longo, e não abreviado como é de costume em páginas HTML.

## 1º Passo: Criar uma classe Java que faça a exibição de data no formato longo:

```
package minhastaglibs;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

@SuppressWarnings("serial")
public class NovaData extends TagSupport {
    public int doStartTag() throws JspException {
        try {
            String formatoLong = "EEEEEE' dd 'de' MMMM 'de' yyyy";
            SimpleDateFormat formatter = new SimpleDateFormat(formatoLong);
            String dataAtual = formatter.format(Calendar.getInstance().getTime());
            pageContext.getOut().print(dataAtual);
        } catch (IOException e) {
            throw new JspException(e.getMessage());
        }
        return SKIP_BODY;
    }
}
```

## **2º Passo: Criação do arquivo .TLD, que contém as descrição das tags que criamos:**

```
<?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>
    <jspversion>1.1</jspversion>
    <shortname>Minhastaglibs</shortname>
    <info>Minhas Tags</info>
    <tag>
        <name>dataAtual</name>
        <tagclass>minhastaglibs.NovaData</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>Utilizada para informar a data atual em formato longo</info>
    </tag>
</taglib>
```

## **3º Passo: No arquivo web.xml, presente na sua aplicação web, adicione a taglib para sua configuração:**

```
<taglib>
    <taglib-uri>/tags/minhastaglibs</taglib-uri>
    <taglib-location>/WEB-INF/tags/minhastaglibs.tld</taglib-location>
</taglib>
```

## **4º Passo: Criarmos nossa página .jsp para testar nossa Tablib:**

```
<%@ taglib uri="/tags/minhastaglibs" prefix="novoformato"%>
<!-- perceba que na sintaxe acima indicamos a URL onde se encontra nossas Tags: "/tags/minhastaglibs" e indicamos um prefixo que será usado para chamar a Tag--&gt;
<!-- agora iremos criar o código HTML e inserir a data modificada--&gt;
&lt;HTML&gt;
    &lt;HEAD&gt;
        &lt;meta HTTP-equiv="Content-Type" content="text/HTML; charset=ISO-8859-1"&gt;</pre>
```

```
<TITLE>Minha Primeira Página JSP</TITLE>
</HEAD>
<BODY>
    A data atual: <novoformato:dataAtual></novoformato:dataAtual>
    <!-- veja que utilizamos <prefix:nome_da_tag>-->
</BODY>
</HTML>
```

Todo esse trabalho para criar apenas a conversão de tempo para um formato diferente é um dos exemplos que podemos fazer, a vantagem de criar tags é a de que, na próxima vez em que precisar utilizar essa nova formatação de data, você não precisará criar novamente a tag – ela está guardada em sua biblioteca, basta apenas utilizar o código:

```
<%@ taglib uri="/tags/minhastaglibs" prefix="novoformato"%>
```

Para indicar que você deseja utilizar um Taglib e também o caminho no qual se encontra a Tag e um prefix para ser utilizado no código, depois utilize:

```
<prefix:nome_da_tag>
```

Para dizer qual das tags que ali estão em sua biblioteca você deseja utilizar, dessa forma, você terá o trabalho de criá-la apenas uma vez, depois apenas a utilize quando achar necessário.

Perceba que sem essas diretivas, que são a base do JSP, não poderíamos utilizar as funções que o código Java possui – seja usando o include para inserir o conteúdo de outros arquivos em uma JSP, ou utilizando o page para prover informações importantes ao servidor que estará interpretando o código Java contido no JSP ou, ainda, criando nossas próprias funcionalidades por meio da Taglib.

Ainda existe muito a ser visto em JSP, mas com esses três exemplos de possibilidades já podemos observar o potencial para a criação de códigos mais dinâmicos, mesmo em páginas HTML simples.

## Conclusão



Podemos concluir que o uso de JSP vem trazer o melhor de duas linguagens em uma só tecnologia. Se de um lado temos o HTML, que é a linguagem web mais conhecida – por ser uma das primeiras e por ter uma sintaxe simples com possibilidades de rodar em qualquer navegador, embora não tenha funções mais elaboradas –, do outro lado temos o Java, que possui diversas possibilidades e funcionalidades em sua arquitetura, permitindo a criação do dinamismo que as páginas atuais tanto buscam (HANSEN, 2007).

Ao entendermos o funcionamento de JSP, podemos expandir as possibilidades de criar, ao mesmo tempo, projetos simples por meio do HTML e com uma infraestrutura complexa com Servelets e código de Servetlets, embedendo código Java e tornando JavaServer Pages uma solução extremamente atrativa.

## Material Complementar

Como complemento desta unidade, sugerimos a leitura completa do capítulo I do livro:

- TODD, N.; SZOLKOWSKI, M. **Javaserver Pages:** O Guia do Desenvolvedor. Rio de Janeiro: Elsevier, 2003.

Sugerimos, também, a leitura completa do livro:

- KURNIAWAN, B. **Java Para a Web Com Servlets, Jsp e Ejb.** Rio de Janeiro: Ciencia Moderna, 2002.

Boa leitura!

## Referências

TODD, N.; SZOLKOWSKI, M. **Javaserver Pages:** O Guia do Desenvolvedor. Rio de Janeiro: Elsevier, 2003.

KOLB, M A. **Desenvolvendo na Web com Java Server Pages.** São Paulo: Editora Ciência Moderna, 2000.

KURNIAWAN, B. **Java Para a Web Com Servlets, Jsp e Ejb.** Rio de Janeiro: Ciencia Moderna, 2002.

HANSEN, M. D. **Soa Using Java Web Services.** New Jersey: Prentice Hall, 2007.

DAVID M, G. **JavaServer Pages Avançados.** São Paulo: Editora Ciência Moderna, 2010.

# Anotações





**Educação a Distância**  
Cruzeiro do Sul Educacional  
*Campus Virtual*

www.cruzeirodosulvirtual.com.br  
Campus Liberdade  
Rua Galvão Bueno, 868  
CEP 01506-000  
São Paulo SP Brasil  
Tel: (55 11) 3385-3000

