Desenvolvimento Web I

Aula 05 - Introdução e Recursos JavaServer Pages

Apresentação

Olá, seja bem-vindo(a) a nossa quinta aula da disciplina de Desenvolvimento Web utilizando a linguagem Java. Até agora, você aprendeu a fazer uso dos Servlets para acessar os parâmetros das requisições web, processá-los e montar as páginas de resposta. Pois bem, os Servlets realmente são a base da programação web em Java, porém, seu uso direto nos leva a alguns problemas.

Um dos principais que podemos perceber com o uso de Servlets puro, ou seja, quando não usamos nenhuma outra tecnologia além dos Servlets, é que o código HTML das páginas de resposta fica embutido dentro do código Java. Essa mistura nos traz diversas dificuldades, como dificuldade de entendimento e manutenção do código e necessidade de reiniciar o servidor web para que alterações de layout do HTML sejam carregadas - caso não esteja desenvolvendo o código em modo de depuração (debug). Nesta aula, você aprenderá a usar uma tecnologia chamada de JavaServer Pages (JSP), visando remover a edição do código HTML de dentro das classes Java.

Imagine que você possua uma empresa desenvolvedora de aplicações para Internet e que exista um projetista web (web designer), ou seja, uma pessoa responsável por fazer a edição das páginas web. Normalmente, essa pessoa tem muito conhecimento de HTML, CSS e de outras tecnologias que tornam as páginas Web bonitas e fáceis de serem utilizadas. Entretanto, os projetistas web geralmente não possuem conhecimento de programação em linguagens de uso geral, como Java, C, .Net etc. Essas pessoas normalmente não têm condições de fazer alterações no código Java da aplicação, mesmo que a alteração seja relativa ao código HTML que está embutido no código do Servlet Java. Dessa forma, qualquer alteração no código HTML, mesmo que simples (ex: mudar um texto em negrito para itálico), requer do programador Java a alteração da classe do Servlet baseado na alteração sugerida pelo projetista web.

Além disso, essa alteração no Servlet requer usualmente que o servidor seja reiniciado, já que uma nova implementação do Servlet precisa ser carregada pelo servidor. Em ambientes de desenvolvimento (usados pelos desenvolvedores), essa

carga é feita geralmente de forma automática pelo servidor, mas em ambientes de produção (usados pelos clientes) isso não acontece por questões de desempenho e pelos seguintes inconvenientes causados pelo reinício do servidor:

- O sistema ficará indisponível para os usuários durante o tempo de reinicialização;
- As sessões dos usuários serão perdidas, ou seja, os objetos sessão que guardam os estados dos clientes no servidor serão descartados durante a reinicialização do servidor.
    - Dessa forma, os clientes já autenticados no sistema terão que se autenticar novamente.
    - Clientes podem vir a perder dados já digitados. Imagine que um cliente, usando o sistema de cadastro desenvolvido nas aulas anteriores, tenha preenchido a primeira tela de cadastro (dados pessoais) e estivesse preenchendo a segunda tela. Toda essa informação seria perdida durante a reinicialização do serv

Além de todos esses problemas, a mistura do código HTML com o código Java atrapalha também a vida do programador. Imagine como o código Java dos Servlets ficaria menor sem todos aqueles comandos write() que escrevem código HTML.

A tecnlogia JSP é completamente funcional na plataforma Java EE mais recente, entretanto a existe também outra tecnologia chamada JSF (Java Server Faces) como ferramenta para construção de interfaces Web com Java. A base de sistemas utilizando JSP no mundo é muito grande e ainda é uma ótima tecnologia para que seus conhecimentos tenham valor no mercado de trabalho.

Estaremos utilizando JSP em Desenvolvimento Web I também iremos aprender JSF em Desenvolvimento Web II e você aprenderá no final a utilizar ambas as tecnologias.

Vídeo 01 - Apresentação

Objetivos

- Descrever o funcionamento da solução JSP.
- Entender a sintaxe utilizada na criação de páginas JSP.

JavaServer Pages

JavaServer Pages (JSP) é uma tecnologia utilizada para se criar conteúdo web tanto estático como dinâmico. Por conteúdo estático, queremos dizer toda aquela parte do código HTML das páginas de resposta que não mudam de acordo com os parâmetros da requisição ou com o estado do cliente (sessão). São exemplos dessa parte: as imagens, textos e outros conteúdos que são sempre mostrados pelo sistema.

Já o que chamamos de conteúdo dinâmico é a parte das páginas de resposta cujo valor depende dos parâmetros da requisição ou do estado do cliente (sessão). Por exemplo, em um sistema de compras on-line, o conteúdo do carrinho de compras depende da sessão do cliente. Isso porque o que será mostrado é a lista de produtos selecionados pelo usuário, a qual possivelmente foi sendo montada ao longo de várias telas do sistema, correto?

Muito bem! Vamos, então, ao que interessa. Assim como no caso das páginas HTML, o JSP é um arquivo texto que, quando acessado, resultará em uma página web. Os arquivos JSP tem extensão .jsp e podem conter conteúdo estático, geralmente definido por marcadores HTML, e conteúdo a ser gerado dinamicamente, definido por marcadores específicos do JSP.

Veja o exemplo simples de página JSP, de nome aleatorio.jsp, mostrado na Listagem 1. Nesse exemplo, um número aleatório é gerado toda vez que a página é acessada. Note que a maior parte do conteúdo do arquivo é de código HTML. Vamos ver então o que muda. Primeiro, você pode observar um novo tipo de instrução nas linhas 1 e 2. Essa instrução <%@ page %> é chamada de diretiva JSP e seu uso é opcional. Essa diretiva basicamente indica informações sobre o conteúdo do JSP: a linguagem de programação utilizada no JSP é Java, o conteúdo da página a ser gerada é texto HTML (text/html) e a codificação de caractere utilizada é a ISO-8859-1. Falaremos mais sobre diretivas JSP depois.

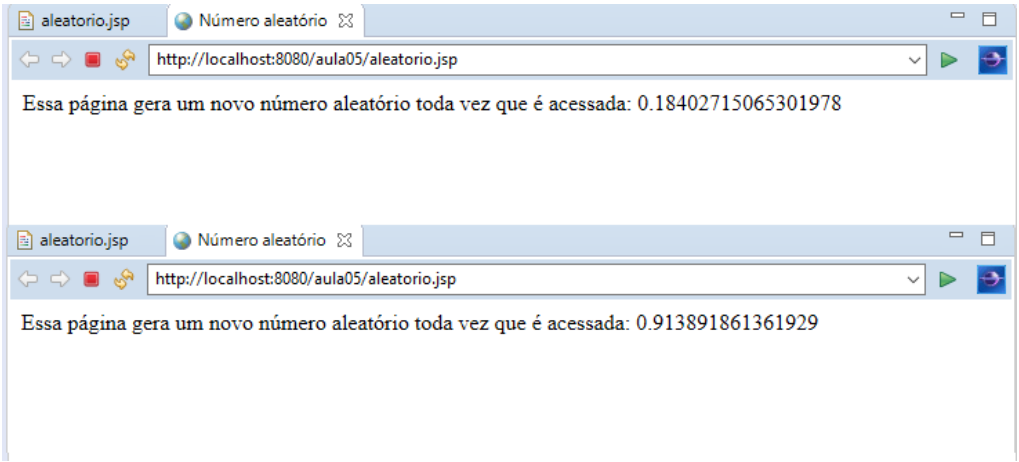
Vídeo 02 - Introdução ao JSP

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859
2 <html>
3 <head>
4 <title>Número aleatório</title>
5 </head>
6    <body>
7       Essa página gera um novo número aleatório toda vez que é acessada: <%= Math.random() %>
8    </body>
9 </html>
```

Listagem 1 - Página JSP (aleatorio.jsp) para geração de números aleatórios.

A outra novidade encontra-se na linha 9. Note o uso do marcador <%= %>. Esse marcador JSP delimita um espaço que deve ser preenchido com uma expressão Java, ou seja, com uma instrução que retorne um valor de qualquer tipo, podendo ser uma String, um número, um booleano, etc. No caso do nosso exemplo, a expressão utilizada é Math.random(), uma instrução Java utilizada para retornar um número aleatório diferente toda vez que ela é executada. Dessa forma, toda vez que a página JSP é acessada pelo usuário, um novo número aleatório é gerado. A Figura 1 mostra diferentes acessos à página JSP, ilustrando o fato de que um número diferente é apresentado toda vez que acessamos novamente a página.

Figura 01 - Página JSP de geração de números aleatórios



Nesse momento, provavelmente ainda não está claro para você quem executa a instrução Math.random(), se é o navegador ou o servidor web. Vamos responder essa pergunta a seguir, mas já podemos adiantar que todo o processamento é feito no lado do servidor!

Atividade 01

1. Implemente o arquivo aleatorio.jsp conforme mostrado na Listagem 1. Salve o arquivo na pasta do projeto em que você usualmente coloca seus arquivos com extensão .html (exemplo: dentro da pasta WebContents). Acesse a URL relativa à página JSP, como, por exemplo, <http://localhost:8080/ProgramacaoWeb/aleatorio.jsp>. Verifique se o conteúdo da página está sendo mostrado corretamente.

2. Faça novos acessos à página aleatório.jsp e confirme que o número gerado é aleatório e diferente dos anteriormente mostrados.

3. Realize uma alteração no conteúdo estático do arquivo aleatório.jsp, mudando, por exemplo, o texto que diz que um número aleatório será gerado a cada novo acesso. Salve o arquivo e acesse através de sua URL, sem realizar, porém, nenhuma reinicialização. Confirme que a nova página de resposta será apresentada.

Ciclo de vida do JSP

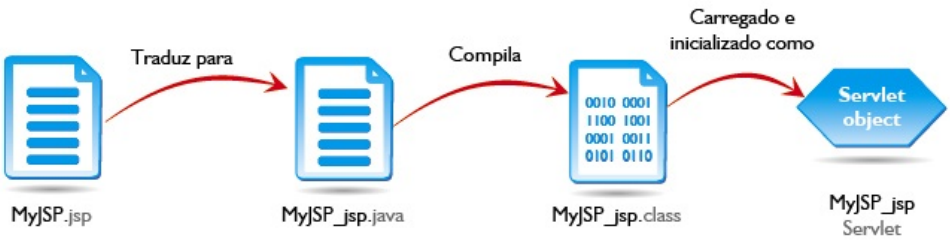
Você já viu que páginas HTML possuem conteúdo estático e que Servlets podem ser responsáveis por gerar tanto conteúdo estático como dinâmico (gerar resposta de acordo com parâmetros da requisição). Uma página JSP, assim como os Servlets, pode conter tanto conteúdo estático quanto dinâmico. Apesar disso, seu código parece mais com páginas HTML puras do que com o código dos Servlets.

Para entender o funcionamento geral do JSP, é importante saber que essa tecnologia foi desenvolvida em cima da tecnologia dos Servlets. Você vai ver que, em Java, é muito comum que as novas tecnologias sejam desenvolvidas baseadas em tecnologias anteriores, já maduras e estabelecidas. No caso, para entender a relação entre Servlets e JSP, vamos estudar o chamado ciclo de vida das páginas JSP.

Uma página JSP é normalmente escrita pelo programador. Ela pode também ser editada diretamente pelo projetista web, já que seu conteúdo é muito parecido com o de páginas HTML (veja novamente a Listagem 1). Uma vez criada a página, ela deve ser disponibilizada em um servidor web, da mesma forma que ocorre com as páginas HTML. A página aleatorio.jsp, por exemplo, foi colocada na pasta WebContent do projeto de nome ProgramacaoWeb, garantindo assim seu acesso através do servidor Web. Dessa forma, seu conteúdo poderá ser acessado pela URL <http://localhost:8080/aula05/aleatorio.jsp>, considerando que o servidor está rodando localmente e na porta 8080.

Ao ser acessado pela primeira vez, o arquivo JSP será traduzido para um Servlet. É isso mesmo, o JSP será transformado pelo servidor web em um Servlet. Isso é feito de forma automática e transparente para o usuário e para o programador. Veja esse processo ilustrado na Figura 2.

Figura 02 - Processo de transformação transparente do JSP em Servlet



Fonte: Adaptada de Basham, Sierra e Bates (2008).

Note na figura que o arquivo MyJSP.jsp gera um Servlet de nome MyJSP_jsp.java. Na verdade, o nome do Servlet gerado não importa para nós, já que esse processo é feito de forma automática e transparente. Por fim, o código gerado para esse Servlet é então compilado e carregado no servidor web.

Para podermos entender melhor, vamos considerar que o arquivo aleatorio.jsp seja traduzido para o Servlet mostrado na Listagem 2. Note que esse código, apesar de seu tamanho, está simplificado, sendo ainda maior na realidade.

```
package aula05;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
// Vários outros imports foram omitidos aqui

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

 // Vários atributos foram omitidos aqui

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

 // Várias declarações de variáveis foram omitidos aqui
     HttpSession session = null;
     ServletContext application = null;
     ServletConfig config = null;
     JspWriter out = null;
     Object page = this;
     JspWriter _jspx_out = null;
     PageContext _jspx_page_context = null;

     try {
            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;

            write("<html>\r\n");
            out.write("<head>\r\n");
            out.write("<title>Número aleatório</title>\r\n");
            out.write("</head>\r\n");
            out.write("<body>\r\n");
            out.write("Esta página gera um novo número aleatório\r\n");
            out.write("toda vez que é acessada: ");
            out.print(Math.random());
            out.write("\r\n");
            out.write("</body>\r\n");
            out.write("</html>");
       } catch (Throwable t) {
```

```
52                 // Código de tratamento de exceção foi omitido aqui
53             }
54         }
55     }
```

Listagem 2 - Simplificação do código-fonte gerado para o Servlet que representa o arquivo aleatorio.jsp

Ciclo de vida do JSP II

O primeiro ponto que devemos notar é a forma de tradução do JSP para o Servlet. Essa tradução, representada principalmente pelo bloco de linhas de código 39 a 49 da Listagem 2, basicamente segue o raciocínio demonstrado abaixo.

- Todo o conteúdo da página de resposta será escrito através de uma variável de nome out. Apesar de seu tipo ser JspWriter, ela se comporta de forma semelhante à classe PrintWriter que estávamos acostumados a usar ao escrevermos Servlets.

- Conteúdos estáticos do arquivo aleatorio.jsp, como a linha que contém o marcador , serão traduzidos pelo comando out.write(), passando o conteúdo estático como parâmetro. Esse exemplo é visto na linha 39, cujo código gerado é o out.write("\r\n");.

- Conteúdos dinâmicos, representados por marcadores e diretivas JSP, são traduzidos de forma a gerar o código Servlet equivalente, ou seja, do código que iríamos escrever se precisássemos escrever esse Servlet manualmente. Veja alguns exemplos:

    - A diretiva <%@ page ... %> diz que o conteúdo da página de resposta é texto HTML, através do trecho (contentType="text/html; charset=ISO-8859-1"). É muito importante você especificar o charset como está vendo no exemplo (nesse caso o ISO-8859-1) pois isso garante que o cliente (Navegador geralmente) vai exibir corretamente os textos do seu site com os acentos e caracteres especiais. Esse conteúdo poderia também ser de outro tipo, como texto XML. No caso do HTML, essa instrução é traduzida para o comando da linha 29 (response.setContentType("text/html");).

- A linha 9 da Listagem 1 (toda vez que é acessada: <%= Math.random() %>) contém tanto código estático (texto) como conteúdo dinâmico (expressão JSP que resultará em um número aleatório).
    - Nesse caso, a primeira parte, que é estática, será traduzida usando o comando out.write("toda vez que é acessada: "); (linha 45). Basicamente o texto é transformado em uma String (texto entre aspas duplas) e escrito na saída pelo comando write.
    - Já a parte do conteúdo dinâmico é traduzida colocando-se o resultado da expressão Java como parâmetro do comando write. No caso, a expressão <%= Math.random() %> é traduzida por comando out.write(Math.random()); (linha 46).

Com isso, vimos que é possível traduzir as instruções contidas no arquivo JSP em um Servlet. Isso é feito de forma automática e de forma transparente, então você não precisa se preocupar em entender exatamente como essa tradução é feita. O importante é você saber que ela existe e ter uma noção de como ela funciona!

Dito isso, você pode se perguntar: esse processo de tradução não pode impactar o desempenho do sistema? E a resposta é sim! O primeiro acesso a um arquivo JSP é sempre mais lento, pois exige a tradução do arquivo em um Servlet e a compilação do código Java gerado. Entretanto, a partir do segundo acesso, não há impacto no desempenho quando comparado à implementação através de Servlets puros.

Vídeo 03 - Estrutura do JSP

JSP e variáveis declaradas

Nós já vimos que arquivos JSP são traduzidos para Servlets antes de serem executados. Esse processo de tradução já foi inclusive ilustrado. Pois bem, agora surge o questionamento do porquê de se criar um JSP e não um Servlet, já que no final o JSP vai acabar virando um Servlet mesmo. A resposta para essa pergunta será dada a seguir.

Existe um consenso de que é muito mais fácil e legível trabalhar com o arquivo JSP do que com o Servlet gerado. Você concorda? É claro que o código gerado não está muito fácil de ser lido porque ele foi gerado automaticamente. Um ser humano teria condições de gerar um código muito mais legível, mas, mesmo assim, ainda seria mais fácil trabalhar direto com o arquivo JSP. Para confirmar isso, é só olharmos o código do arquivo JSP e o trecho de código das linhas 39 a 49 da Listagem 2. O que é mais legível, o código JSP ou esse trecho de código no Servlet? Espero que possamos concordar que o arquivo JSP é mais fácil de ser trabalhado pelo programador e por um possível projetista web!

Sobre a tradução dos JSPs nos Servlets, é importante você notar a existência do método _jspService() (linha 15) no Servlet gerado. Esse método é responsável pela geração da página de resposta. Note que o Servlet gerado não implementa os métodos doGet() e doPost() e herda de HttpJspBase (linha 10) ao invés de herdar da classe HttpServlet. Entretanto, o método _jspService() possui os parâmetros request e response, da mesma forma que ocorre com os métodos doGet() e doPost(). Além disso, observe também o trecho de código das linhas 20 a 26 da Listagem 2. Esse bloco de código inclui a declaração de diversas variáveis locais ao método _jspService(), as quais são inicializadas mais adiante no código (linhas 32 a 37):

- *session*
    - Referência ao objeto sessão do usuário, se a mesma tiver sido criada.

- *application*
    - Referência ao objeto do tipo ServletContext, o qual dá acesso ao contexto do Servlet.
- *config*
    - Referência ao objeto do tipo ServletConfig, o qual dá acesso a informações de configuração do Servlet.
- *out*
    - Referência ao objeto de escrita na página de resposta a ser gerada

Ok, mas por que saber disso tudo? Simples, tanto os parâmetros do método como essas variáveis locais podem ser acessadas pelo código JSP. Por exemplo, podemos criar uma página JSP para indicar se o usuário possui uma sessão aberta. Para isso, criaremos o arquivo de nome info-sessao.jsp mostrado na Listagem 3. Esse arquivo, como mostrado na linha 8, usa o valor da variável session para acessar a sessão aberta para o usuário e nos mostrar o identificador único gerado para representar a sessão do usuário.

Mas cuidado ao usar arquivos JSP, pois os objetos sessão são sempre criados se não existirem. O comportamento é semelhante ao método getSession() de HttpServletRequest que não possui parâmetros.

Importante!

Ao usar arquivos JSP, os objetos sessão são criados se não existirem.

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859
2 <html>
3    <head>
4       <title>Informações sobre sessão</title>
5    </head>
6    <body>
7       Id da sessão do usuário: <%= session.getId() %> <BR>
8    </body>
9 </html>
```

Listagem 3 - Arquivos JSP com informação sobre o ID da sessão

Atividade 02

1. Implemente o exemplo mostrado para imprimir o ID da sessão do usuário. Execute e observe que, ao recarregar a página, o mesmo ID é retornado. Observe também que, ao reiniciar o servidor, um novo ID será retornado, já que os dados da sessão antiga serão perdidos e uma nova será criada

2. Se seu computador possuir mais de um navegador web (Internet Explorer, Mozilla, Chrome etc.), acesse ao mesmo tempo a página info-sessao.jsp usando os vários navegadores instalados. Note que o ID da sessão é diferente para cada navegador. Isso acontece porque o ID é armazenado no navegador, então, cada um deles receberá um ID diferente. Dessa forma, cada navegador será interpretado como um usuário diferente, apesar de ser a mesma pessoa e o mesmo computador acessando o servidor.

3. Escreva uma página JSP novaMensagem.jsp que mostre um formulário na tela com dois campos: email e mensagem.

4. Escreva outra página gravarMensagem.jsp que receba os parâmetros email e mensagem e grave esses dois parâmetros na sessão do usuário. Faça com que o formulário da primeira página novaMensagem.jsp submeta os dados para a página gravarMensagem.jsp. Obs.: você deve manter todo o histórico de mensagens na sessão. Dica: ao chegar uma nova mensagem, salve na sessão o histórico de mensagens previamente salvo concatenado com a nova mensagem

5. Crie uma terceira página listarMensagens.jsp que mostre mensagens criadas até o momento. Lembre-se de executar as páginas para verificar seu correto funcionamento.

Scriptlets

Vamos agora a outro exemplo de página JSP. Vamos implementar uma página que mostra quantas vezes ela já foi acessada. Para isso, vamos fazer uso da classe Contador, como mostrado na Listagem 4. A classe Contador possui um método estático chamado novoAcesso(), o qual incrementa a contagem (atributo contagem)

de acessos realizados àquele método. Já o método getQuantidadeAcessos(), retorna a quantidade de acessos realizados até o momento. O uso da palavra-chave static faz com que o atributo e os métodos da classe Contador possam ser utilizados sem serem criados objetos, você está lembrado? Qualquer dúvida, consulte novamente o material da disciplina de Programação Orientada a Objetos.

```
1  package aula05;
2
3  public class Contador {
4    private static int contagem;
5
6    public static void novoAcesso() {
7      contagem = contagem + 1;
8    }
9    public static int getQuantidadeAcessos() {
10     return contagem;
11   }
12 }
```

Listagem 4 - Classe Contador usada para armazenar a quantidade de acessos.

Para usar essa classe em um arquivo JSP, precisamos fazer com que ele execute o método novoAcesso() e, em seguida, execute e apresente o conteúdo retornado pelo método getQuantidadeAcessos(). Isso é feito pelo arquivo acessos.jsp, mostrado na Listagem 5.

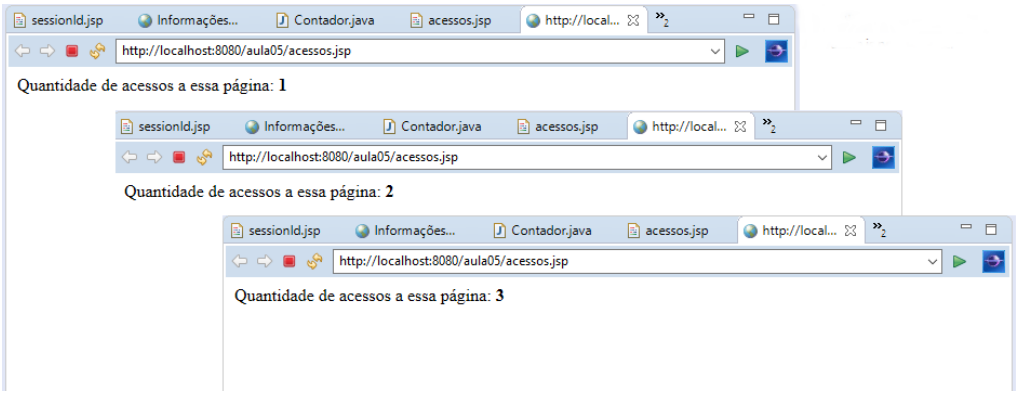
```
1  <html>
2    <body>
3    <% aula05.Contador.novoAcesso(); %> Quantidade de acessos a essa página:
4    <b>
5       <%= aula05.Contador.getQuantidadeAcessos() %>
6    </b>
7    </body>
8  </html>
```

Listagem 5 - Código-fonte do arquivo acessos.jsp

Para executar o método novoAcesso(), fazemos uso de scriptlets. Os scriptlets são blocos de código Java delimitados pelos marcadores <% e %>. No exemplo da Listagem 5, o scriptlet se refere às linhas 3 a 5. Dentro desse bloco de código, pode existir qualquer comando Java válido. Isso porque, como você se lembra, o arquivo JSP será traduzido para um Servlet. O que estiver entre o <% e %> será simplesmente copiado para o método do Servlet responsável pela criação da página de resposta (jsp_service()).

Voltando para o exemplo, quando o JSP for executado (na verdade, seu Servlet gerado), o método novoAcesso() também será executado, incrementando em uma unidade a quantidade de acessos armazenada pela classe Contador (atributo contagem). Por fim, o valor dessa contagem é apresentado na página de resposta através da linha 8. O resultado de vários acessos a esse arquivo JSP é mostrado na Figura 3.

Figura 03 - Acessos sequenciais ao arquivo



Declaração de variáveis

Nas páginas anteriores, você viu a implementação de uma página JSP (acessos.jsp) para contagem da quantidade de acessos realizados. Para implementar essa página, foi utilizada a classe Contador, bem como seus métodos estáticos novoAcesso() e getQuantidadeAcessos(). Vamos tentar agora implementar essa mesma funcionalidade, sem fazer uso da classe auxiliar Contador.

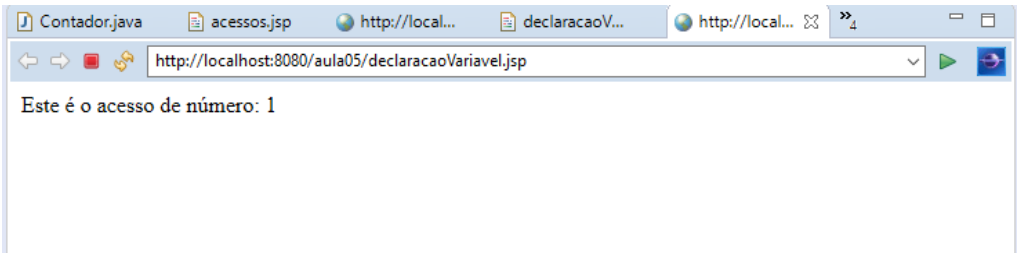
Você já aprendeu que os marcadores <% e %> são utilizados para delimitar scriptlets, ou seja, código Java embutido nos arquivos JSP. Sendo assim, será que podemos utilizar scriptlets para declarar uma variável no próprio arquivo JSP e utilizá-la como contador (antigo código da classe Contador)? Veja o código JSP a seguir.

```
1 <html>
2    <body>
3    <% int cont = 1; %>
4    Este é o acesso de número: <%= cont %>
5    <% cont = cont + 1; %>
6    </body>
7 </html>
```

Listagem 6 - Código JSP declarando e usando variáveis

No código mostrado na Listagem 6, temos a declaração de uma variável de nome cont e sua inicialização com o valor 1. Seu valor é impresso indicando a quantidade de acessos já realizada (inicialmente igual a 1) e depois o seu valor é incrementado em uma unidade, indicando que o próximo valor a ser impresso deve ser 2 (segundo acesso). Ao rodar esse código, a tela da Figura 1 é apresentada, indicando que o código está correto. Entretanto, os acessos posteriores também apresentarão o número 1, como se fosse o primeiro acesso. Você consegue perceber qual o problema do código mostrado?

Figura 04 - Página JSP mostrando primeiro acesso.



Vamos observar um possível código de Servlet gerado para esse JSP:

```
1  public class contadorAcessos_jsp extends HttpJspBase {
2    public void _jspService(HttpServletRequest request,
3      HttpServletResponse response) throws java.io.IOException,
4      ServletException {
5     PrintWriter out = response.getWriter();
6     response.setContentType("text/html");
7     out.write("<html><body>");
8     int cont = 1;
9     out.write("Este é o acesso de número: ");
10    out.print(cont);
11    cont = cont + 1;
12    out.write("</body></html>");
13   }
14 }
```

Listagem 7 - Código (simplificado) do Servlet correspondente ao JSP da Listagem 1.

Quando o arquivo JSP foi traduzido para um Servlet, a variável cont virou uma variável local ao método _jspService(). Sendo uma variável local, seu valor final é perdido no final da execução do método.

Então, o que nós precisamos para resolver nosso problema é fazer com que essa variável cont seja uma variável de instância (atributo do Servlet) e não simplesmente uma variável local. Isso pode ser feito através de um elemento JSP chamado de declaração. Uma declaração é delimitada pelos marcadores <%! e %> e contém a declaração de uma variável. Na tradução do JSP para o Servlet, essa variável será traduzida para um atributo da classe gerada. Vejamos agora como deve ficar a declaração da variável cont:

```
1 <%! int cont = 1; %>
```

E vejamos também como fica o Servlet gerado para o JSP com a nova forma de declaração de variável:

```
1  public class contadorAcessos_jsp extends HttpJspBase {
2
3    int cont = 1;
4    public class _jspService(HttpServletRequest request, HttpServletResponse response) throws java.
5       PrintWriter out = response.getWriter();
6       response.setContentType("text/html");
7       out.write("<html><body>");
8       out.write("Este é o acesso de número: ");
9       out.print(cont);
10      cont = cont + 1;
11      out.write("</body></html>");
12   }
13 }
```

Listagem 8 - Código (simplificado) do Servlet correspondente ao JSP usando <%! %>

Atividade 03

---

1. Implemente e execute o arquivo acessos.jsp e a classe Contador. Observe se sua implementação tem o comportamento mostrado na Figura 3.
2. Altere o exemplo mostrado para apresentar não só a quantidade de acessos, mas a data e a hora do primeiro e do último acesso.
3. Implemente a página JSP de contagem de acesso das duas formas mostradas na aula: declarando uma variável em um scriptlet e declarando a variável contadora com a instrução <%! %>. Relate se você tiver alguma dificuldade.

4. Desenvolva duas novas páginas: a primeira deve apresentar um formulário para digitar o valor recebido por um estagiário e o percentual de aumento que ele deve ter; a segunda página deve receber os parâmetros enviados pelo formulário da primeira página e calcular o novo valor a ser pago para o estagiário. Além de mostrar esse novo valor, a resposta deve imprimir uma mensagem de acordo com as regras abaixo:

    a. Menor que 200: Mude de estágio!

    b. De 200 a 500: Negocie que esse valor pode melhorar!

    c. Acima de 500: Agora está ficando bom!

Declaração de variáveis II

Vamos agora pensar em uma contagem que leva em consideração a hora na qual o usuário acessou o sistema. Por exemplo, casos de acessos em horários tarde da noite ou de madrugada serão contabilizados como 2 acessos. Os acessos realizados durante o restante do horário são considerados como apenas 1 acesso. Você imagina como poderíamos implementar essa contagem “ponderada”?

Bom, além de declarar variáveis, você pode também declarar operações (métodos) em um arquivo JSP. Veja na Listagem 9 um arquivo JSP com a declaração de um método calculaContagem(), responsável por calcular a contagem de acesso baseada na hora atual.

```
<%@ page import="java.util.Calendar" %>
<html>
  <body>
  <%!
  int cont = 0;
  int calculaContagem() {
    int valor = 1;
    int hora = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
    if (hora >= 22 || hora <= 6) {
      valor = 2;
    }
    return valor;
  }
  %>
  <%
  cont = cont + calculaContagem();
  %>
  Este é o acesso de número: <%= cont %>*
  <br>
  * Considerando que acessos noturnos valem por 2.
  </body>
</html>
```

Listagem 9 - JSP com contagem dependente da hora de acesso do usuário.

Observe que, além da declaração da variável cont, temos também o método calculaContagem(), que retorna 2 apenas quando o acesso é feito entre às 22h e às 6h da manhã. O código JSP começa pela declaração de import da classe java.util.Calendar, utilizada para retornar a hora atual (Calendar.getInstance().get(Calendar.HOUR_OF_DAY)). O método getInstance() retorna uma referência para o objeto Calendar que representa a data/hora atual. Já o método get() retorna uma propriedade da data/hora, como a hora do dia (hora atual, entre 0h e 23h59).

Comentários em arquivos JSP

Em HTML, você sabe que comentários podem ser escritos, desde que delimitados entre <!-- e -->. Em JSP, comentários podem ser feitos tanto usando os marcadores HTML de comentários como também usando marcadores específicos: <%-- e --%>. Em ambos os casos, o código escrito não irá afetar a tela de resposta do arquivo JSP, mas existe uma diferença sutil. Comentários usando os marcadores JSP não são colocados na página HTML de resposta.

Para entender isso, veja o seguinte exemplo:

```
1 <html>
2   <body>
3     Olá!
4     <!-- Esse comentário vai para o navegador Web, -->
5     <%-- já esse comentário não vai --%>
6   </body>
7 </html>
```

Listagem 10 - Código do JSP com comentários

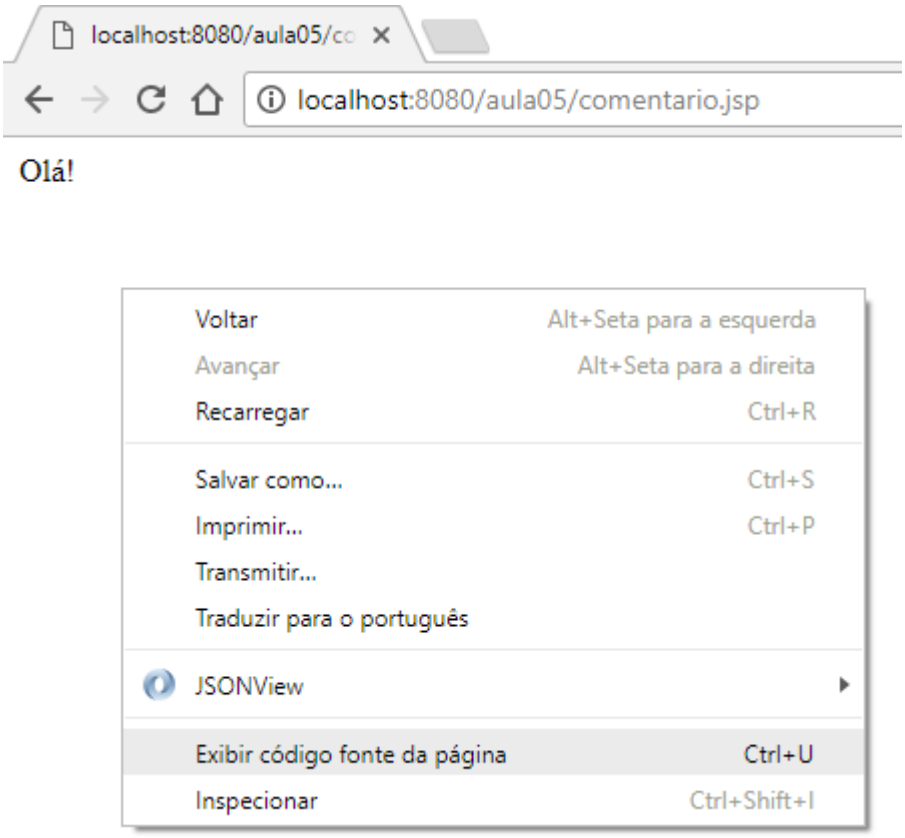
Esse código JSP gera a página mostrada na Figura 6. Se você observar o código-fonte da página recebida pelo navegador web (no navegador web, clique no botão direito para aparecer essa opção), vai visualizar apenas o seguinte:

```
1 <html>
2   <body>
3     Olá!
4     <!-- Esse comentário vai para o navegador Web, -->
5   </body>
6 </html>
```

Listagem 11 - Código HTML da página de resposta do arquivo JSP com comentários

Note que no lugar do comentário JSP (<%-- -->) é mostrada apenas uma linha em branco. O comentário foi suprimido. Esse comportamento é interessante, caso queiramos colocar comentários no código do JSP como, por exemplo, “esta funcionalidade não está robusta, ocorre um erro se informarmos determinado valor como parâmetro” etc. Imagine se um cliente curioso acessar a página, ver seu código-fonte e observar esse tipo de mensagem! Ou então um hacker, que pode explorar esse tipo de informação para derrubar seu sistema!

Figura 06 - Página de Olá com comentários HTML e JSP



Atividade 04

1. Altere a página JSP de contagem de acesso, adicionando comentários em HTML e em JSP. Observe quais comentários aparecem no código-fonte da página HTML enviada para o navegador web.

2. Escreva um arquivo JSP de nome temperatura.jsp que imprima uma tabela HTML de conversão Celsius (C) para Fahrenheit (F) entre -40 e 100 graus Celsius, com incrementos de 10 em 10 (A fórmula é F = 9/5 C + 32). Escreva no arquivo comentários HTML e JSP.

Inclusão de arquivos

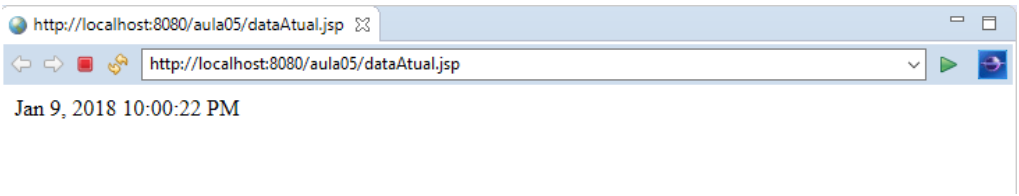
Outra característica interessante de JSP é a capacidade de modularizar seu código, ou seja, de dividir o conteúdo de seus arquivos em partes menores que depois serão compostas para resultar nas páginas do sistema. Vejamos o seguinte

exemplo de arquivo JSP:

```
1  <%@ page import="java.util.*" %>
2    <%= (new java.util.Date()).toLocaleString() %>
```

Esse código é responsável por criar um objeto Date que representa a data/hora atual e por transformar esse objeto em uma String com a data no formato local, apresentando-a no HTML gerado. A página resultante da execução desse código é mostrada na Figura 7.

Figura 07 - Arquivo dataAtual.jsp que mostra a data e hora atual



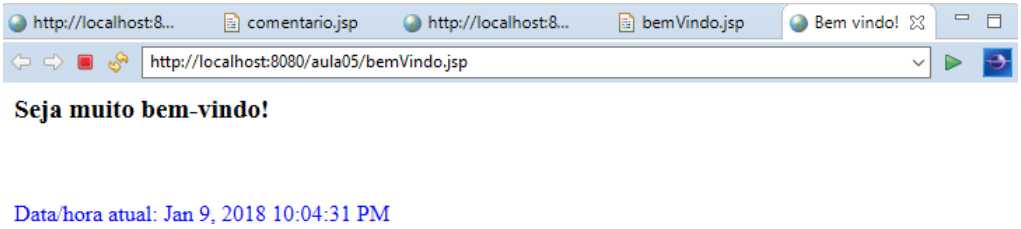
Essa informação pode ser útil em várias páginas do sistema. Dessa forma, esse código passa a ser repetido em vários arquivos JSP. Entretanto, para você não ter que duplicar esse código em todos os outros arquivos, você pode importar o conteúdo de dataAtual.jsp para esses outros arquivos.

Vamos imaginar um arquivo de boas-vindas, no qual é mostrada uma mensagem de boas-vindas e a data e hora atual. O código desse arquivo JSP é mostrado a seguir, e sua tela resultante é mostrada na Figura 8.

```
1  <html>
2    <head>
3      <title>Bem vindo!</title>
4    </head>
5    <body bgcolor="white">
6      <h3>Seja muito bem-vindo!</h3>
7      <br>
8      <br>
9      <font color="blue">
10     Data/hora atual:
11     <%@ include file="dataAtual.jsp"%>
12     </font>
13   </body>
14 </html>
```

Listagem 12 - Código JSP com <%@include %>

Figura 08 - Tela de boas-vindas, a qual importa o conteúdo do arquivo dataAtual.jsp



Veja que o responsável por mostrar a data e a hora do sistema é o código escrito no arquivo dataAtual.jsp. Esse link é feito pela instrução <%@ include file="dataAtual.jsp"%> (ver linha 11 da Listagem 7). Basicamente, essa instrução diz que o conteúdo do arquivo dataAtual.jsp deve ser copiado para o Servlet gerado a partir do arquivo bemVindo.jsp. Note que é como se o compilador editasse o arquivo do Servlet gerado para incluir as instruções contidas no arquivo dataAtual.jsp.

Atividade 05

1. Implementar e executar os arquivos bemVindo.jsp e dataAtual.jsp, conforme apresentado. Relatar se houver alguma dificuldade.

2. Criar um arquivo endereço.jsp contendo um texto que mostra o endereço de uma empresa. Alterar o arquivo bemVindo.jsp para mostrar também o conteúdo do arquivo endereço.jsp, o qual também poderia ser incluído em diversos outros arquivos JSP do sistema.

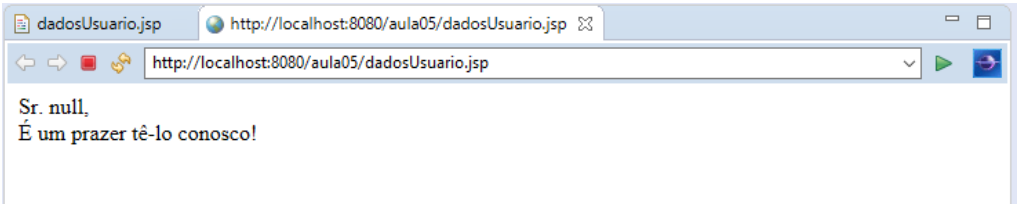
Inclusão de arquivos II

Outra forma de montar o conteúdo de uma tela baseado no conteúdo de outras é através da diretiva <jsp:include >. Essa diretiva faz a inclusão tanto estática como dinâmica de outros arquivos. Inclusão estática é aquela feita pela instrução <%@ include >, ou seja, o código do arquivo referenciado é colocado no Servlet gerado para o arquivo JSP que está querendo executar. Já a inclusão dinâmica é diferente. Na hora em que o arquivo que faz a inclusão é executado, ele executa o arquivo incluído e seu resultado (código da página de resposta) é colocado dentro da resposta final a ser montada.

Imagine o código mostrado a seguir para um arquivo JSP chamado de dadosUsuario.jsp. Ele apresenta uma mensagem de identificação e saudação baseada no parâmetro “usuario”. Se executarmos esse arquivo, teremos o resultado mostrado na Figura 6. Como nenhum valor é passado para o parâmetro “usuario”, o valor null aparece na tela de resposta:

```
1 Sr. <%= request.getParameter("usuario") %>,<br>
2 É um prazer tê-lo conosco!
```

Figura 09 - Acesso ao arquivo dadosUsuario.jsp sem passagem de parâmetros

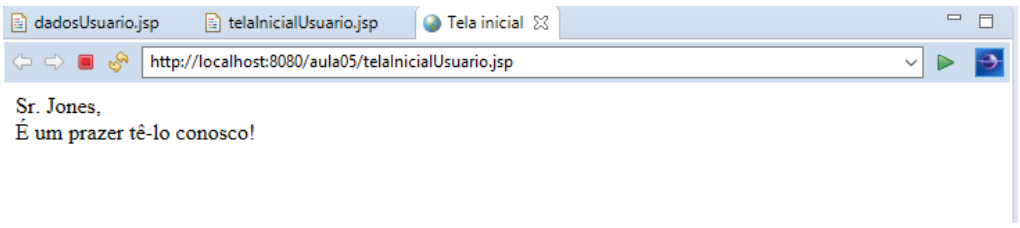


Você pode, no entanto, criar outros arquivos JSP que queiram mostrar esse conteúdo de identificação e boas-vindas como, por exemplo, no código da Listagem 13 (telaInicialUsuario.jsp). O resultado da execução desse arquivo é mostrado na Figura 10. A página de resposta gerada inclui o conteúdo gerado pelo arquivo dadosUsuario.jsp quando o parâmetro “usuario” é passado com valor igual a Jones. Para tal, utiliza-se a diretiva <jsp:param>, indicando o parâmetro a ser passado e seu valor. O valor do exemplo é passado de maneira estática, mas, em um programa real, poderia ter sido recuperado de uma base de dados, por exemplo.

```
1  <html>
2    <head>
3      <title>Tela inicial</title>
4    </head>
5    <body>
6      <jsp:include page="dadosUsuario.jsp">
7        <jsp:param name="usuario" value="Jones" />
8      </jsp:include>
9    </body>
10 </html>
```

Listagem 13 - Inclusão do arquivo dadosUsuario.jsp com passagem de parâmetro

Figura 10 - Inclusão do arquivo dadosUsuario.jsp com passagem de parâmetro



Atividade 06

1. Implemente o arquivo telaInicialUsuario.jsp, conforme mostrado na aula. Execute-o e veja o seu comportamento.
2. Altere o arquivo dadosUsuario.jsp para receber e imprimir na tela um parâmetro relativo ao sobrenome do usuário. Altere o arquivo telaInicialUsuario.jsp para passar o sobrenome do usuário. Execute o código e observe o novo comportamento.

Conclusão

Chegamos ao fim de nossa aula! Daremos continuidade ao assunto na próxima aula, momento no qual você aprenderá a utilizar Servlets em conjunto com os arquivos JSP. Até lá!

Leitura Complementar

Para complementar a fixação do conhecimento, faça uma breve leitura das aulas anteriores sobre JSP.

Veja também artigos na internet sobre esse assunto, como por exemplo:

- <https://docs.oracle.com/javaee/5/tutorial/doc/bnakc.html>. Acesso em: 09 janeiro 2018.

Estude também a tecnologia Java Server Faces para complementar seus conhecimentos:

- <https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm>. Acesso em: 09 janeiro 2018.

Resumo

Nesta aula, você aprendeu o que são arquivos JSP e como funciona seu ciclo de vida. Você viu que essa tecnologia é baseada nos Servlets e que arquivos JSP, ao serem executados, são traduzidos para Servlets e depois compilados e executados. Para a escrita dos arquivos JSP, utilizamos a sintaxe da linguagem HTML (já conhecida por você), além de marcadores especiais traduzidos pelo JSP. Vimos também alguns exemplos de marcadores especiais: a diretiva <%@page >, usada para dar informações sobre a página JSP, os scriptlets (<% e %>) e as expressões (<%= e %>), usados para incorporar instruções Java no código JSP.

Além disso, você aprendeu que variáveis podem ser declaradas dentro de scriptlets (tornando-se variáveis locais) ou dentro de declarações entre <%! e %> (tornando-se variáveis de instância, ou seja, atributos). Essa segunda forma de declaração pode ser utilizada também para declarar métodos, os quais farão parte do Servlet gerado pela compilação do arquivo JSP. Além disso, você aprendeu a utilizar comentários dentro de arquivos JSP, os quais podem ser iguais aos comentários HTML ou podem utilizar marcadores específicos, evitando que os

comentários sejam enviados para o navegador web. Por fim, você viu o uso do <%@ include > e do <jsp:include >, marcadores especiais utilizados para incluir em um arquivo JSP o conteúdo de outros arquivos, promovendo, assim, o reuso de código entre esses arquivos.

Autoavaliação

1. Descreva o funcionamento da solução JSP, em especial, o ciclo de vida dos arquivos JSP.
2. Descreva os recursos que você aprendeu para a criação dos arquivos JSP, como o uso de diretivas, scriptlets e variáveis implícitas (out, session etc).
3. Descreva como declarar variáveis e métodos em um arquivo JSP.
4. Descreva como você pode comentar código em arquivos JSP.
5. Descreva como você pode reusar código de outros arquivos JSP.

Referências

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com J2EE J2EE e UML.** Rio de Janeiro: Ciência Moderna, 2003. 324 p.

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Head First Servlets and JSP**: passing the Sun Certified Web Component Developer Exam (SCWCD). 2nd ed. O'Reilly Media, 2008.

CATTELL, Rick; INSCORE, Jim. **J2EE:** criando aplicações comerciais. Rio de Janeiro: Editora Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), 2001. 752 p. Disponível em: <http://pdf.moreservlets.com/>. Acesso em: 11 maio 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP).** 2nd ed. New Jersey: Prentice Hall PTR (InformIT), 2003. 736p. (Core Technologies, 1). Disponível em: <http://pdf.coreservlets.com/>. Acesso em: 11 maio 2012.

HYPERTEXT Transfer Protocol: HTTP/1.1 – Methods Definition. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Acesso em: 11 maio 2012.

J2EE Tutorial. Disponível em: <http://java.sun.com/javaee/reference/tutorials/>. Acesso em: 11 maio 2012.