

INTEGRATION

JSP, A INTERFACE DO USUÁRIO EM JAVA

ALEXANDRE JESUS



03

LISTA DE FIGURAS

Figura 1 – Resultado da execução da página JSP.....	6
Figura 2 – Processamento da página JSP	8
Figura 3 – Resultado do processamento da página olaMundo.jsp.....	10
Figura 4 – Código HTML gerado após processamento da página AloMundo.jsp	11
Figura 5 – Resultado de uma <i>Expression</i> exibindo uma data	13
Figura 6 – Resultado da página inicio.jsp com <i>include</i> do menu.jsp no browser.....	16
Figura 7 – Código-fonte do resultado da página inicio.jsp com <i>include</i> do menu.jsp	16
Figura 8 – Resultado da página com comentários	18
Figura 9 – Resultado da página com comentários – Código HTML/FONTE	18
Figura 10 – Página origem.jsp.....	21
Figura 11 – Página destino.jsp.....	22
Figura 12 – Página login.jsp.....	25
Figura 13 – Página servicos.jsp apresentando o valor do atributo da sessão.....	26
Figura 14 – Página index.jsp apresentando o valor do atributo da sessão	26
Figura 15 – Página index.jsp apresentando o resultado do código contador de visitas	28
Figura 16 – Resultado de não utilizar as boas práticas.....	30
Figura 17 – Sucesso por usar boas práticas	30
Figura 18 – Resultado da página inicio.jsp com <i>include</i> do menu.jsp	32
Figura 19 – Resultado da página inicio.jsp com a <i>Action forward</i> para a página index.jsp	34
Figura 20 – Criando um projeto Java Web – Parte 1	35
Figura 21 – Criando um novo projeto Java Web – Parte 2.....	36
Figura 22 – Criando um novo projeto Java Web – Parte 3.....	37
Figura 23 – Site do <i>bootstrap</i>	38
Figura 24 – Download do <i>bootstrap</i>	38
Figura 25 – Criando uma nova pasta no projeto	39
Figura 26 – Estrutura final dos arquivos do <i>bootstrap</i> e JQuery	40
Figura 27 – Download do JQuery.....	41
Figura 28 – Criando uma nova página JSP – Parte 1	42
Figura 29 – Criando um novo projeto Java Web – Parte 2.....	43
Figura 30 – Resultado da execução da página inicial	46

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de uma página JSP	6
Código-fonte 2 – Exemplo de <i>Scriptlets</i> para introdução de código Java no HTML..	7
Código-fonte 3 – Exemplo de <i>Scriptlets</i>	9
Código-fonte 4 – Exemplo de <i>Scriptlet</i> no formato <i>tag</i> XML.....	9
Código-fonte 5 – Página olaMundo.jsp	10
Código-fonte 6 – Exemplo de <i>Declaration</i>	11
Código-fonte 7 – Exemplo de <i>Expression</i>	12
Código-fonte 8 – Sintaxe geral de uma <i>Directiva</i>	13
Código-fonte 9 – Exemplo de <i>Directiva page</i>	13
Código-fonte 10 – Exemplo de <i>Directiva page</i> com import	14
Código-fonte 11 – Exemplo de utilização de <i>Directiva include</i>	15
Código-fonte 12 – <i>SNIPPET</i> menu.jsp	15
Código-fonte 13 – <i>Directiva taglib</i>	17
Código-fonte 14 – Exemplo de comentário em páginas JSP	17
Código-fonte 15 – Exemplo de comentários JSP e HTML	18
Código-fonte 16 – Comentários JAVA dentro dos <i>Scriptlets</i>	19
Código-fonte 17 – Exemplo de objeto implícito <i>out</i>	20
Código-fonte 18 – Página origem.jsp	21
Código-fonte 19 – Página destino.jsp com o objeto <i>request</i>	22
Código-fonte 20 – Exemplo de objeto implícito <i>response</i>	23
Código-fonte 21 – Página login.jsp com o objeto <i>session</i> sendo criado.....	24
Código-fonte 22 – Página index.jsp recebendo o atributo que foi criado na sessão .	24
Código-fonte 23 – Página serviços.jsp recebendo o atributo que foi criado na sessão	25
Código-fonte 24 – Exemplo de código de contador de visitas.....	27
Código-fonte 25 – Exemplo de Action JSP	29
Código-fonte 26 – Exemplo de <i>Action include</i>	29
Código-fonte 27 – Página inicio.jsp	31
Código-fonte 28 – <i>SNIPPET</i> menu.jsp	31
Código-fonte 29 – Sintaxe <i>Action Forward</i> sem parâmetros com URL relativa.....	33
Código-fonte 30 – Sintaxe <i>Action forward</i> com parâmetros com URL relativa.....	33
Código-fonte 31 – Página index.jsp recebendo os parâmetros da <i>Action forward</i>	33
Código-fonte 32 – Página header.jsp com link para o css do <i>bootstrap</i>	43
Código-fonte 33 – Página footer.jsp com os <i>javascripts</i>	44
Código-fonte 34 – Página inicial da aplicação.....	44
Código-fonte 35 – Menu para a aplicação.....	45
Código-fonte 36 – Página inicial com o menu	46

SUMÁRIO

1 JSP, A INTERFACE DO USUÁRIO EM JAVA	5
1.1 JSP	5
1.2 Vantagens de utilizar JSP	7
1.3 Processamento do JSP	8
1.4 Elementos do JSP	9
1.5 Scriptlets.....	9
1.5.1 Declarations	11
1.5.2 Expressions.....	12
1.5.3 Directivas.....	13
1.5.3.1 Directiva Page	14
1.5.3.2 Directiva include	14
1.5.3.3 Directiva taglib.....	17
1.5.4 Comentários	17
2 OBJETOS IMPLÍCITOS	20
2.1 Objeto out.....	20
2.1.1 Objeto request.....	20
2.1.2 Objeto response	22
2.1.3 Objeto session.....	23
2.2 Objeto application.....	26
2.3 Objeto pageContext	28
2.4 JSP actions	29
2.4.1 Action <jsp:include>	29
2.4.2 Action <jsp:forward>.....	32
2.5 Prática!	34
REFERÊNCIAS.....	48

1 JSP, A INTERFACE DO USUÁRIO EM JAVA

1.1 JSP

Anteriormente, vimos como integrar as páginas HTML com a plataforma Java por meio das *Servlets*. Nós recuperamos as informações das páginas HTML na *Servlet* e depois construímos uma página de resposta para o usuário, dentro da própria classe. Isso funciona! Porém, é muito trabalhoso, difícil de desenvolver e dar manutenção. É por isso que existem as páginas JSP, que facilitam muito a nossa vida de desenvolvedor.

Esse é o começo da separação em camadas da aplicação, a famosa arquitetura MVC, que vamos utilizar para implementar o sistema Fintech. A arquitetura MVC, basicamente, separa a aplicação em três principais camadas, cada uma com sua responsabilidade. Uma das camadas é a *View* (MVC), que é responsável por fazer a interface da aplicação com o usuário, nesse caso, as telas web. É nessa camada que a tecnologia JSP estará. Não se preocupe com essa arquitetura, que abordaremos com mais profundidade nesta fase.

Java Server Pages (JSP) é uma tecnologia que auxilia os desenvolvedores de software a criarem páginas web geradas dinamicamente e baseadas em HTML, XML ou outros tipos de documentos. Lançada em 1999 pela Sun Microsystems, JSP é similar ao PHP, porém utiliza a linguagem de programação Java. Para implantar e executar *Java Server Pages* é requerido um servidor web compatível com um container *servlet*, como Apache Tomcat, Jetty ou Glassfish.

Essa tecnologia permite ao desenvolvedor web produzir aplicações que acessem o banco de dados, manipulem arquivos, capturem informações a partir de formulários e captem informações sobre o visitante e o servidor. Uma página criada com a tecnologia JSP, após ser instalada em um servidor de aplicação compatível com a tecnologia Java EE, é transformada em uma *Servlet*.

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Exemplo JSP</title>
</head>
<body>

    <ul>

    <%for(int x = 0; x < 10 ; x++){%>

        <li><%=x %></li>

    <%}%>

    </ul>

</body>
</html>
```

Código-fonte 1 – Exemplo de uma página JSP
Fonte: Elaborado pelo autor (2016)

A página JSP apresentada no código-fonte acima possui a extensão **.jsp**, porém possui características de um arquivo HTML comum, com o diferencial de permitir o uso de código Java embutido, que ao ser processado por um *Web Container* gera o resultado exibido na Figura Resultado da execução da página JSP.

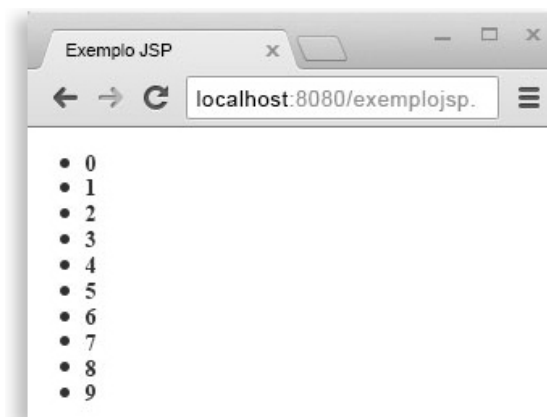


Figura 1 – Resultado da execução da página JSP
Fonte: Elaborado pelo autor (2016)

Na Figura Resultado da execução da página JSP, podemos ver que foi gerado o resultado esperado, a iteração do loop for dentro dessa *tag* especial, que é apresentada no Código-fonte “Exemplo de *Scriptlets* para introdução de código Java no HTML”. Neste momento, não se preocupe com essas *tags*, veremos com mais detalhes no decorrer do capítulo. Agora é importante entender que estamos utilizando código Java junto ao código HTML.

```
<% . . . . %>
```

Código-fonte 2 – Exemplo de *Scriptlets* para introdução de código Java no HTML
Fonte: Elaborado pelo autor (2016)

1.2 Vantagens de utilizar JSP

Como vimos, podemos utilizar as *Servlets* para criar e enviar uma página HTML para o usuário. Porém, utilizar as instruções **println** para criar o conteúdo de uma página inteira é trabalhoso. Primeiro porque o método **println** recebe uma *string*, assim podemos inserir qualquer texto junto às *tags*, que não são reconhecidas. Segundo, para dar manutenção ou construir páginas complexas com CSS e Javascript é praticamente impossível. Imagine o tamanho da *servlet* que precisaríamos criar para gerar uma página que tenha um menu, um campo de busca e uma tabela, por exemplo. É mais amigável desenvolver uma página web em um editor que entenda as *tags* e tenha suporte a essa tecnologia. Outra vantagem é que estamos separando as responsabilidades em camadas, o que deixa nosso sistema com uma arquitetura mais adequada.

Por que não utilizar somente código HTML? Infelizmente o código HTML é estático, ou seja, não podemos construir uma página dinamicamente. Por exemplo, no sistema Fintech, podemos desenvolver uma funcionalidade que permite ao usuário verificar os últimos dados dos sensores que monitoram o seu sono. Para isso, precisamos de uma página web que exiba os valores de acordo com os dados que estão armazenados no banco de dados. Dessa forma, a página precisa ser “criada” dinamicamente para apresentar esses dados.

1.3 Processamento do JSP

Como funciona o processamento de um JSP? A Figura “Processamento da página JSP” apresenta a sequência de processos para que um JSP possa ser processado a fim de se tornar a página HTML final.

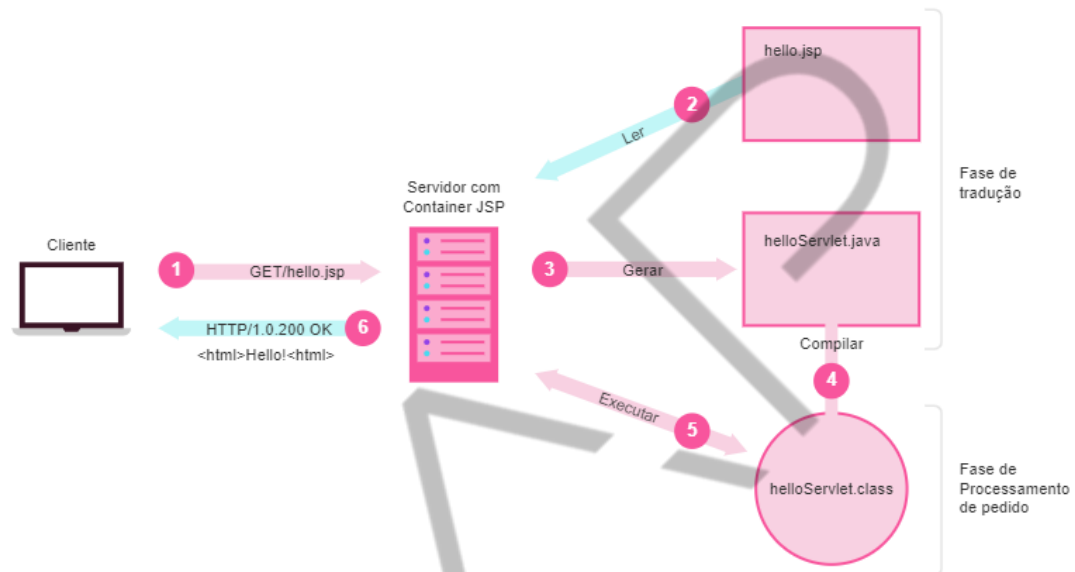


Figura 2 – Processamento da página JSP
Fonte: Tutorial Spoint (2011)

- Ocorre uma requisição HTTP do browser para servidor onde está a aplicação web.
- 2 e 3. Os componentes JSP presentes no servidor de aplicação realizam a conversão do texto do JSP para o código Java, transformando o em uma *Servlet*.
- 4. Esses mesmos componentes JSP realizam a compilação do arquivo transformado.
- 5. O servidor de aplicação executa a *Servlet* que foi gerada a partir do JSP e gera uma resposta HTML para o cliente.
- 6. O servidor de aplicação encaminha a resposta HTML para o cliente.

1.4 Elementos do JSP

Neste momento abordaremos os principais elementos de uma página JSP. Cada elemento é responsável por alguma função na página, por exemplo, para utilizar uma biblioteca de *tags*, precisamos utilizar o elemento “Diretiva”, a fim de escrever código Java, utilizamos os *Scriptlets*, e assim por diante.

1.5 Scriptlets

Os *Scriptlets* são blocos de código Java inseridos dentro das páginas JSP.

Os *Scriptlets* devem iniciar com `<%` e terminar com `%>`:

```
<%  
    String str = "JSP";  
%>
```

Código-fonte 3 – Exemplo de *Scriptlets*
Fonte: Elaborado pelo autor (2016)

Mas também podem ser escritos no formato de *tag* XML:

```
<jsp:scriptlet>  
    String str = "JSP";  
</jsp:scriptlet>
```

Código-fonte 4 – Exemplo de *Scriptlet* no formato *tag* XML
Fonte: Elaborado pelo autor (2016)

Entre esses sinais ou *tags* podemos inserir código Java e, assim que o JSP for processado, será interpretado e executado. Os elementos HTML devem estar fora dos *Scriptlets*, somente código Java pode ser inserido nesse bloco. Segue um exemplo de página JSP.

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>  
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=ISO-8859-1">  
<title>Alô Mundo!</title>  
</head>
```

```
<body>

    <p>Alô Mundo!</p>

    <p>
        <%
            out.println("Seu IP é : " +
request.getRemoteAddr());
        %>
    </p>

</body>
</html>
```

Código-fonte 5 – Página olaMundo.jsp
Fonte: Elaborado pelo autor (2016)

Podemos perceber que no Código-fonte “Página olaMundo.jsp” existe a implementação dos *Scriptlets*, utilizando um método do objeto *request* para recuperar o IP do usuário. Na Figura “Resultado do processamento da página olaMundo.jsp”, é apresentado o resultado da execução da página JSP no browser do usuário.

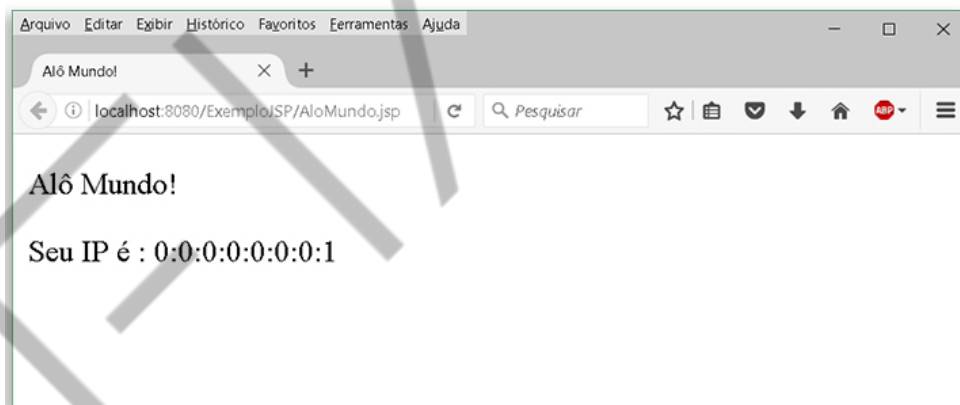
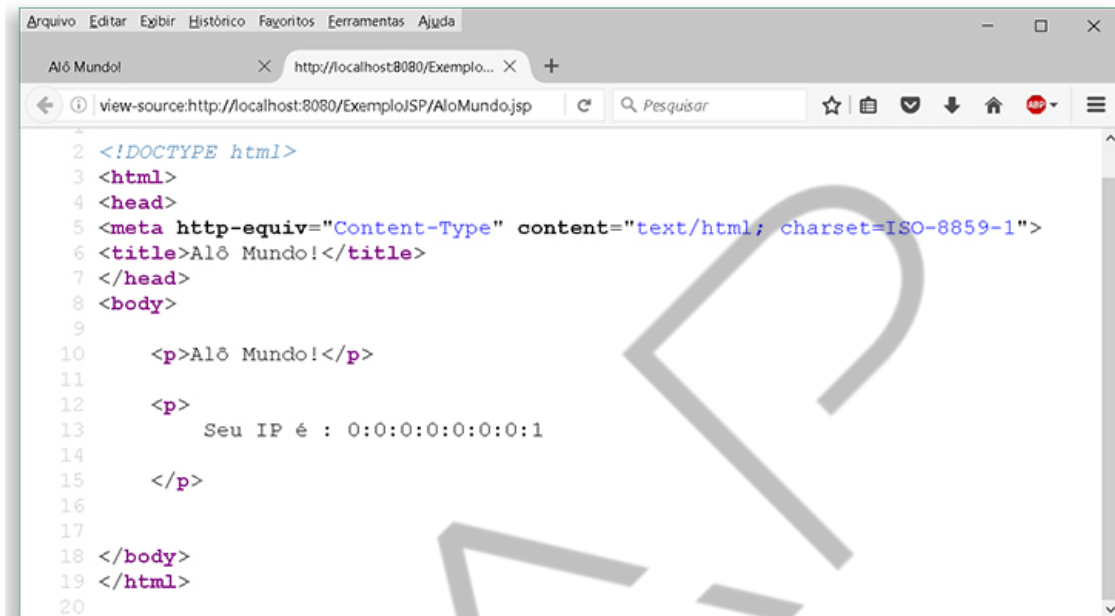


Figura 3 – Resultado do processamento da página olaMundo.jsp
Fonte: Elaborado pelo autor (2016)

Já na Figura a seguir exibimos o código-fonte da página HTML. Para ver o código-fonte, você pode clicar com o botão direito do mouse e escolher a opção “Código-Fonte” ou algo similar, dependendo do navegador que estiver utilizando.

Podemos observar o código HTML final e que os *Scriptlets* sumiram! O que aconteceu, foram abduzidos? Não, podemos ver apenas o resultado do processamento da página JSP, ou seja, HTML regular puro, gerado pelo servidor de

aplicação. Lembre-se de que o browser consegue interpretar o código HTML, mas ele não consegue compreender código Java.



```
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
6 <title>Alô Mundo!</title>
7 </head>
8 <body>
9
10 <p>Alô Mundo!</p>
11
12 <p>
13     Seu IP é : 0:0:0:0:0:0:1
14 </p>
15
16
17
18 </body>
19 </html>
20
```

Figura 4 – Código HTML gerado após processamento da página AloMundo.jsp
Fonte: Elaborado pelo autor (2016)

1.5.1 Declarations

Declarations é a forma de declarar variáveis ou métodos que podem ser utilizados dentro da página JSP. Segue exemplos:

```
<%! String nome = "JSP"; %>
<%! int nr = 0; %>
<%! Object obj = new Object(); %>
```

Código-fonte 6 – Exemplo de *Declaration*
Fonte: Elaborado pelo autor (2016)

Uma *Declaration* começa com o sinal de <%! e termina com %>. Note que é muito parecido com *Scriptlets*, porém o início da declaração possui o caractere “!” (exclamação).

1.5.2 Expressions

O elemento *Expression* é parecido com os *Scriptlets* e *Declarations*, porém a principal diferença é que o utilizamos para dar saída em informações que serão apresentadas nas páginas JSP.

O detalhe está na sintaxe da *Expression*, que, diferentemente dos outros dois elementos, não se utiliza de ponto e vírgula para encerrar a linha. Para apresentar a informação, utiliza-se operador de atribuição (sinal de igual). Vamos ver um exemplo de *Expression*:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Expression</title>
</head>
<body>

A data de hoje é : <%= (new
java.util.Date()).toLocaleString() %>

</body>
</html>
```

Código-fonte 7 – Exemplo de *Expression*
Fonte: Elaborado pelo autor (2016)

O resultado da execução da página JSP pode ser visto na Figura “Resultado de uma *Expression*” exibindo uma data.

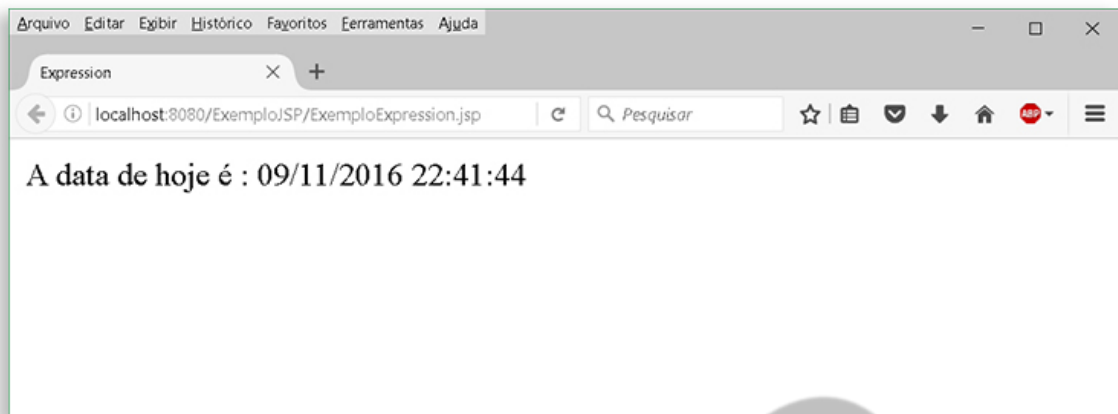


Figura 5 – Resultado de uma *Expression* exibindo uma data
Fonte: Elaborado pelo autor (2016)

1.5.3 Directivas

As *Directivas* são utilizadas para alterar informações em um contexto geral das páginas JSP. A *Directiva* é formada pela mesma sintaxe dos *Scriptlets*, porém no sinal que abre o elemento deve ser adicionado o caractere @ (arroba). Sintaxe:

```
<%@ nome_directiva atributos %>
```

Código-fonte 8 – Sintaxe geral de uma *Directiva*
Fonte: Elaborado pelo autor (2016)

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
```

Código-fonte 9 – Exemplo de *Directiva page*
Fonte: Elaborado pelo autor (2016)

Podemos definir juntamente com as *Directivas*:

- Propriedades na página.
- Incluir código de uma fonte externa na página.
- Definir TagLibs.

TagLibs – São um conjunto de *tags* HTML customizadas para utilização específica em páginas JSP. Seu formato é igual ao de uma *tag* HTML, porém no momento do processamento são interpretadas de maneira diferente.

1.5.3.1 Directiva Page

A *Directiva page* é responsável por definir propriedades da página, tais como a linguagem que será utilizada, o tipo de texto que será formatado, o *charset* para a codificação da página.

```
<%@ page import="java.util.*" language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
```

Código-fonte 10 – Exemplo de *Directiva page* com import
Fonte: Elaborado pelo autor (2016)

Entre as propriedades dessa *Directiva*, podemos destacar a **import**, que nos permite utilizar toda a API Java na página JSP.

No exemplo do Código-fonte “Exemplo de *Directiva page* com import”, podemos ver que a *Directiva page* utiliza o **import** do pacote **java.util**. Com isso, podemos utilizar as classes e interfaces desse pacote, como o **List** e o **ArrayList**, lembrando que, para utilizar o código Java na página JSP, precisamos dos *Scriptlets*.

1.5.3.2 Directiva include

A *Directiva include* é responsável por realizar a inclusão de um arquivo na página atual. O *include* pode ser utilizado para adicionar arquivos a fim de compor a página atual, hoje o desenvolvimento é modularizado e com *include* você pode criar partes do seu site com essa *directiva*, como se fosse um lego. O interessante aqui é que, quando o arquivo é incluído na página atual, ele se torna parte dela, então o ideal é você criar “*SNIPPETS*”.

SNIPPETS – Trechos de códigos independentes do contexto em que está inserido.

Segue abaixo um exemplo de página JSP que inclui um menu. A página inicial.jsp e a página *SNIPPET* menu.jsp são apresentadas nos códigos-fontes “Exemplo de utilização de *Directiva include*” e 3.12, respectivamente.

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
```

```
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta    http-equiv="Content-Type"    content="text/html;
charset=ISO-8859-1">
<link    rel="stylesheet"            type="text/css"
href="css/estilo.css">
<title>Página Inicial</title>
</head>
<body>

        <!-- Aqui está inserido o menu. -->
        <nav> <%@ include file="menu.jsp" %> </nav>

</body>
</html>
```

Código-fonte 11 – Exemplo de utilização de *Directiva include*
Fonte: Elaborado pelo autor (2016)

```
<p>SCRIPTS WEB</p>
<ul>
    <li>JSP</li>
    <li>ASP</li>
    <li>PHP</li>
    <li>CFM</li>
    <li>CGI</li>
</ul>
```

Código-fonte 12 – *SNIPPET* menu.jsp
Fonte: Elaborado pelo autor (2016)

No exemplo, observamos que temos duas páginas, uma com a estrutura HTML completa e a outra, que é o *SNIPPET*, com apenas o código que é necessário para utilização na inclusão.

O resultado é apresentado na Figura “Resultado da página inicio.jsp com *include* do menu.jsp no browser”, em que podemos ver que a página inicio.jsp foi exibida no browser junto ao código do menu.jsp.



Figura 6 – Resultado da página inicio.jsp com *include* do menu.jsp no browser
Fonte: Elaborado pelo autor (2016)

Podemos ainda analisar o código HTML final, que o browser está exibindo. É possível ver que no final é uma página só, com a união das duas páginas.

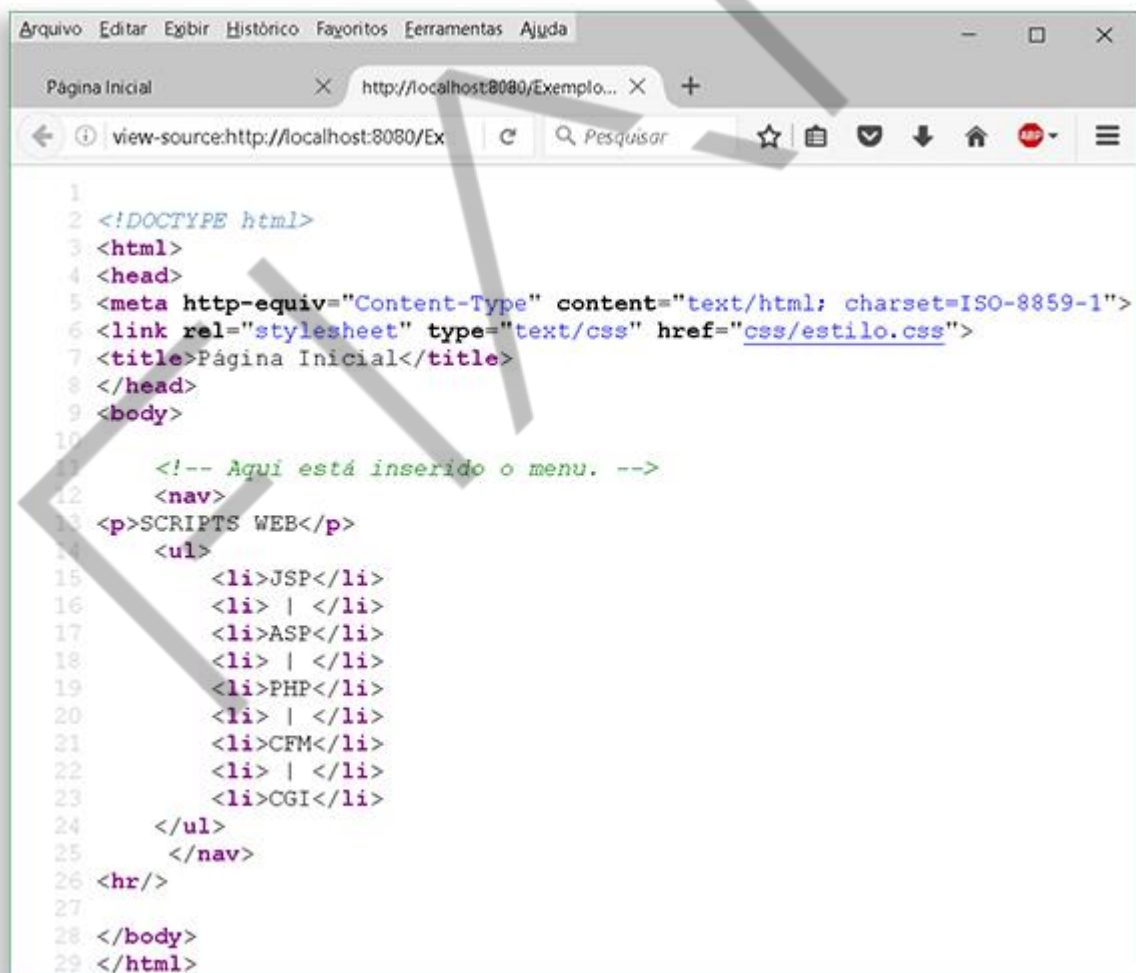


Figura 7 – Código-fonte do resultado da página inicio.jsp com *include* do menu.jsp
Fonte: Elaborado pelo autor (2016)

1.5.3.3 Directiva taglib

A Directiva **taglib** é responsável por referenciar **TagLibraries** na página JSP. Com essa Directiva é possível utilizar uma infinidade de *tags* customizadas para a construção das páginas JSP. Utilizaremos essa Directiva no próximo capítulo.

```
<%@           taglib           prefix="c"
uri="http://java.sun.com/jsp/jstl/core"%>
```

Código-fonte 13 – Directiva taglib
Fonte: Elaborado pelo autor (2016)

1.5.4 Comentários

Os comentários em JSP são parecidos com comentários em HTML, porém não aparecem no resultado do processamento da página.

Comentários de páginas JSP são iguais à notação dos *Scriptlets*, contudo em seu interior eles possuem dois hífen.

```
<!-- Comentário em páginas JSP -->
```

Código-fonte 14 – Exemplo de comentário em páginas JSP
Fonte: Elaborado pelo autor (2016)

Para ilustrar a diferença entre comentários JSP e HTML, veja o Código-fonte “Exemplo de comentários JSP e HTML”.

```
<%@   page   language="java"   contentType="text/html;
charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta   http-equiv="Content-Type"   content="text/html;
charset=ISO-8859-1">
<title>Comentários</title>
</head>
<body>

      <p>Aqui foi feito um comentário JSP</p>
      <!-- JAVA SERVER PAGES -->

      <p>Aqui foi feito um comentário HTML</p>
      <!-- HYPER TEXT MARKUP LANGUAGE -->
```

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

Código-fonte 15 – Exemplo de comentários JSP e HTML
Fonte: Elaborado pelo autor (2016)

Como esperado, o código comentado não aparece no browser do usuário, Figura “Resultado da página com comentários”.



Figura 8 – Resultado da página com comentários
Fonte: Elaborado pelo autor (2016)

Entretanto, no código HTML final, podemos ver que somente o comentário HTML permanece, pois como já foi dito, os elementos JSP são processados no servidor e o comentário com JSP é retirado do HTML final.

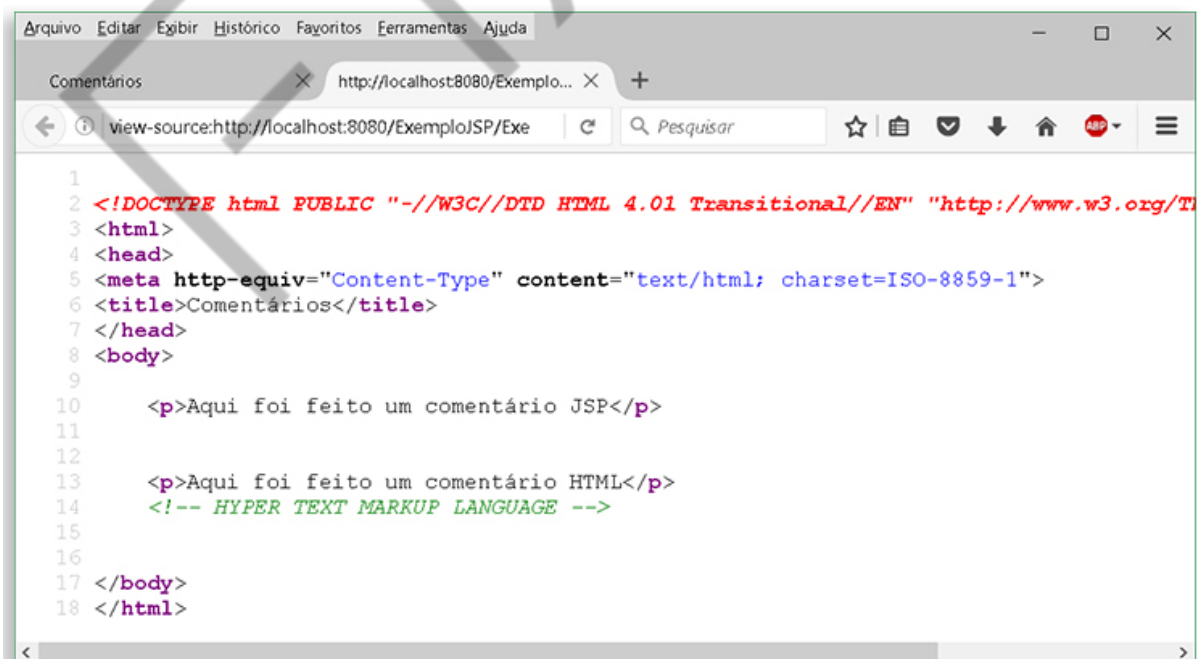


Figura 9 – Resultado da página com comentários – Código HTML/FONTE
Fonte: Elaborado pelo autor (2016)

Além dos comentários que podemos lançar diretamente nas páginas, podemos nos utilizar dos comentários da linguagem Java dentro dos *Scriptlets* normalmente, como se estivéssemos em uma classe.

```
<%  
    //Comentário de Linha  
    /*  
    Comentário de Bloco  
    */  
%>
```

Código-fonte 16 – Comentários JAVA dentro dos *Scriptlets*
Fonte: Elaborado pelo autor (2016)

2 OBJETOS IMPLÍCITOS

Objetos implícitos são componentes que o servidor disponibiliza em cada página sem a necessidade de declaração. “É SÓ CHAMAR QUE EU VOU, É SÓ PEDIR QUE EU DOU LÁLÁLÁ”, parece música da Placa Luminosa, mas não é não. Esses objetos já estão prontos para utilização, estão disponíveis e podem ser acessados diretamente dentro dos *Scriptlets*. Acho que mais fácil que isso não dá!

2.1 Objeto out

O objeto **out** é derivado da classe **JspWriter**, que permite dar saída aos dados. Ele utiliza um método chamado **print()** ou **println()** para imprimir as informações na tela.

print() – Imprime as informações sem a quebra de linha.

println() – Imprime as informações com uma quebra de linha.

```
<%  
    //Objeto out  
    out.println("Objetos Implícitos JSP!");  
%>
```

Código-fonte 17 – Exemplo de objeto implícito *out*.
Fonte: Elaborado pelo autor (2016)

2.1.1 Objeto request

O objeto **request** é um dos objetos mais importantes a se comentar, ele é responsável pela transmissão de parâmetros, atributos e redirecionamentos. É impossível imaginar um cenário web sem esse objeto. Mas vamos parar de babar nele e continuar. Derivado da classe **HttpServletRequest**, é responsável por realizar requisições internas e externas.

Para podermos exemplificar corretamente o objeto **request**, vamos criar duas páginas JSP, uma chamada **origem.jsp** (Código-fonte “Página origem.jsp”) e outra **destino.jsp** (Código-fonte “Página destino.jsp com o objeto *request*”) para passar um parâmetro por meio do **request**.

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>ORIGEM DOS DADOS</title>
</head>
<body>

<!-- Link para o envio de parâmetro para a página
origem.jsp -->
<p><a
href="destino.jsp?parametro1=JavaServer_Pages">ENVIO      DOS
DADOS</a></p>

</body>
</html>
```

Código-fonte 18 – Página origem.jsp
Fonte: Elaborado pelo autor (2016)

A Figura “Página origem.jsp” apresenta o resultado da execução da página origem.jsp.

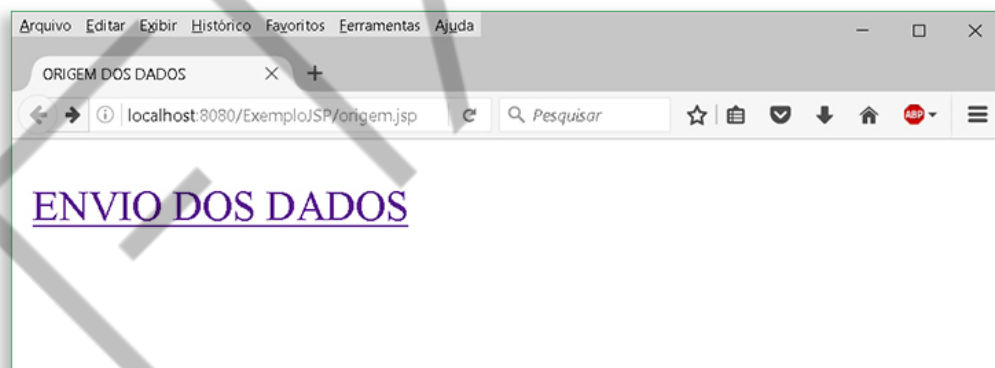


Figura 10 – Página origem.jsp
Fonte: Elaborado pelo autor (2016)

Na página destino.jsp, utilizamos o objeto *request* para recuperar o parâmetro enviado pela página origem. Depois utilizamos uma *expression* a fim de exibir esse valor na tela.

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>DESTINO DOS DADOS</title>
</head>
<body>

    <!-- Bloco de Scriptlets para receber o parâmetro da
    página origem.jsp utilizando o objeto implícito request -->
    <%

        String dados =
request.getParameter("parametro1");

    %>

    <!-- Bloco de Expression para apresentar o
    parâmetro recebido -->
    <p>Parâmetro que foi enviado : <%=dados%></p>

</body>
</html>
```

Código-fonte 19 – Página destino.jsp com o objeto *request*
Fonte: Elaborado pelo autor (2016)

O resultado pode ser visto na Figura “Página destino.jsp”. Lembre-se de que para chegar nessa página foi preciso utilizar o link da página **origem.jsp**.



Figura 11 – Página destino.jsp.
Fonte: Elaborado pelo autor (2016)

2.1.2 Objeto response

O objeto **response** trabalha em conjunto com o **request**. Sempre que for realizada uma requisição, deverá ser gerada uma resposta ao cliente. Ou seja, o **request** é o *input* (entrada) e o **response** é o *output* (saída).

O objeto **response** deriva da classe **HttpServletResponse** e, assim como os demais objetos, pode ser utilizado implicitamente dentro dos blocos *Scriptlets*. Com esse objeto, podemos controlar o *refresh* (atualização) da página, conforme o exemplo (Código-fonte “Exemplo de objeto implícito *response*”).

```
<%  
    response.setIntHeader("Refresh", 1);  
%>
```

Código-fonte 20 – Exemplo de objeto implícito *response*
Fonte: Elaborado pelo autor (2016)

2.1.3 Objeto session

O objeto **session** deriva da classe **HttpSession** e é criado a partir do primeiro **request** do cliente para o servidor. Esse objeto possui um ciclo de vida maior do que o *request*, assim, podemos adicionar atributos que permanecerão enquanto o cliente estiver conectado ao sistema. As duas formas de terminar uma sessão do usuário são por *timeout*, ou seja, por um tempo de inatividade do usuário no sistema, e por um método do próprio objeto de sessão, que pode ser utilizado na funcionalidade de *logout*, por exemplo.

O objeto *session* é muito utilizado nos sistemas para identificar o usuário na aplicação. Vamos criar um exemplo com a utilização do objeto de sessão, para isso vamos desenvolver a página login.jsp e outras duas páginas: index.jsp e servicos.jsp.

No login, vamos colocar o nome do nosso usuário na sessão e replicar esse atributo pelas páginas, capturando-o em cada uma delas, conforme o Código-fonte “Página login.jsp com o objeto *session* sendo criado”.

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=ISO-8859-1">  
<title>Exemplo Sessão</title>  
</head>  
<body>  
<%  
    if(request.getParameter("nomeUsuario") != null)
```

```

        session.setAttribute("attrUsuario",
request.getParameter("nomeUsuario"));
    %>
    <form method="post" action="login.jsp">
        <fieldset>
            <legend>Login</legend>
            <div>
                <label>Nome de Usuário</label><br/>
                <input
name="nomeUsuario">
            </div>
            <div>
                <label>Senha</label><br/>
                <input type="password" name="senha">
            </div>
            <div>
                <input type="submit" value="LOGIN">
            </div>
        </fieldset>
    </form>

    <p><a href="servicos.jsp">SERVIÇOS</a></p>
    <p><a href="index.jsp">INÍCIO</a></p>

</body>
</html>

```

Código-fonte 21 – Página login.jsp com o objeto *session* sendo criado
 Fonte: Elaborado pelo autor (2016)

No Código-fonte “Página login.jsp com o objeto *session* sendo criado”, podemos observar que o formulário envia as informações para o próprio JSP (login.jsp). Ele valida se existe o parâmetro “nomeUsuario” no *request*, caso exista, adiciona o valor enviado na sessão, com a chave “attrUsuario”.

Os códigos-fontes “Página index.jsp recebendo o atributo que foi criado na sessão” e “Página servicos.jsp recebendo o atributo que foi criado na sessão” recuperam o valor armazenado na sessão e exibem na página.

```

    <h2>Bem                vindo                usuário
[<%=session.getAttribute("attrUsuario") %>]
    a página Inicial do Sistema!</h2>
    <p><a href="servicos.jsp">SERVIÇOS</a></p>
    <p><a href="login.jsp">LOGIN</a></p>

```

Código-fonte 22 – Página index.jsp recebendo o atributo que foi criado na sessão
 Fonte: Elaborado pelo autor (2016)


```
<h2>Bem vindo usuário  
[<%=session.getAttribute("attrUsuario") %>]  
a página de Serviços do Sistema!</h2>  
  
<p><a href="index.jsp">INÍCIO</a></p>  
<p><a href="login.jsp">LOGIN</a></p>
```

Código-fonte 23 – Página serviços.jsp recebendo o atributo que foi criado na sessão
Fonte: Elaborado pelo autor (2016)

As figuras “Página login.jsp”, “Página servicios.jsp apresentando o valor do atributo da sessão” e “Página index.jsp apresentando o valor do atributo da sessão” apresentam o resultado da execução das páginas. Observe que o valor do parâmetro enviado foi “fiap”, que pode ser visto nas outras páginas, pois esse valor foi armazenado na sessão do usuário. Para o teste ficar completo, utilize outro navegador para abrir uma nova sessão. Os valores podem ser diferentes para esses navegadores, já que cada navegador terá uma sessão do usuário.

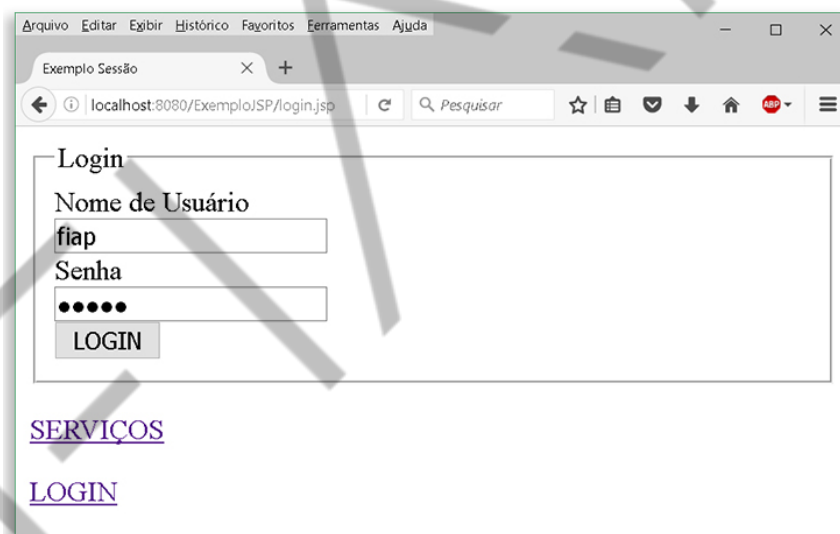


Figura 12 – Página login.jsp
Fonte: Elaborado pelo autor (2016)



Figura 13 – Página servicos.jsp apresentando o valor do atributo da sessão
Fonte: Elaborado pelo autor (2016)



Figura 14 – Página index.jsp apresentando o valor do atributo da sessão
Fonte: Elaborado pelo autor (2016)

2.2 Objeto application

O objeto **application** é criado assim que o servidor é inicializado e removido logo que o servidor é finalizado. Um atributo no objeto **application** fica disponível para todas as páginas e usuários da aplicação web. Assim, podemos criar um exemplo de um contador de visitas.

Um contador de visitas deve contar a quantidade de usuários que visitam a página. Vamos utilizar o método **application.setAttribute(String Key, Object Value);** para definir um atributo no objeto **application**, e o método **application.getAttribute(String Key);** para recuperar esse valor que foi armazenado.

O Código-fonte “Exemplo de código de contador de visitas” apresenta a implementação do contador, primeiro recupera o valor do atributo armazenado no objeto **application** e depois valida se está vazio ou com o valor 0. Caso esteja com vazio ou zero, o contador é inicializado com o valor 1, caso contrário, o contador é incrementado em 1. Depois o valor é adicionado novamente na **application** e, finalmente, é apresentado na página por meio da **expression**.

```
<%
    Integer                hitsCount                =
    (Integer)application.getAttribute("hitCounter");

    if( hitsCount ==null || hitsCount == 0 ){
        /* First visit */
        out.println("Welcome to my website!");
        hitsCount = 1;
    }else{
        /* return visit */
        out.println("Welcome to my website!");
        hitsCount += 1;
    }
    application.setAttribute("hitCounter", hitsCount);
%>

<p>Total number of visits: <%= hitsCount%></p>
```

Código-fonte 24 – Exemplo de código de contador de visitas
Fonte: Elaborado pelo autor (2016)

O resultado é apresentado na Figura “Página index.jsp apresentando o resultado do código contador de visitas”. A cada atualização da página, o valor do contador é incrementado. Caso outro navegador seja aberto, o valor continuará igual para os dois navegadores, ou seja, o valor é único para toda a aplicação.



Figura 15 – Página index.jsp apresentando o resultado do código contador de visitas
Fonte: Elaborado pelo autor (2016)

2.3 Objeto `pageContext`

O objeto **`pageContext`** é utilizado como um meio de acesso às informações da página. Está em contato direto com a parte da memória que lida com os erros gerados por meio da URL, bem como o **`errorPageURL`**.

Por meio desse objeto, podemos ter acesso a todos os outros objetos já citados anteriormente e diversos métodos, contudo um se destaca: o método **`removeAttribute`** que aceita um ou dois argumentos.

Por exemplo, **`pageContext.removeAttribute("attrName");`** remove o atributo de todos os escopos, bem como `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE` e `APPLICATION_SCOPE`.

Enquanto o código a seguir remove apenas do escopo de página: **`pageContext.removeAttribute("attrName", PAGE_SCOPE);`** no segundo argumento, podemos determinar o escopo alvo de onde queremos remover o atributo. Lembrando que esses métodos devem ser chamados dentro de *Scriptlets*.

`errorPageURL`: são aqueles códigos de erros gerados por meio das páginas. Por exemplo: Erro - 404.

2.4 JSP actions

As *actions* JSP usam *tags* na sintaxe XML para controlar o comportamento do mecanismo da *Servlet*. Você pode inserir dinamicamente um arquivo, reutilizar componentes *JavaBeans*, encaminhar o usuário para outra página ou gerar HTML dinamicamente, (fazer café, ops!, café não, né?), mas ele faz todas essas outras coisas. Existe apenas uma sintaxe para o elemento *Action*, uma vez que está em conformidade com o padrão XML. Sintaxe:

```
<jsp:action _name attribute="value" />
```

Código-fonte 25 – Exemplo de Action JSP
Fonte: Elaborado pelo autor (2016)

Os elementos das *Actions* são basicamente funções predefinidas. Abaixo veremos algumas das mais importantes, para exemplificar sua utilização.

2.4.1 Action <jsp:include>

Por meio dessa *Action*, podemos inserir outros arquivos na página que está sendo gerada. Sintaxe:

```
<jsp:include page="menu.jsp" />
```

Código-fonte 26 – Exemplo de Action include
Fonte: Elaborado pelo autor (2016)

Esta *Action* é muito útil quando se trata de modularizar a aplicação, ou seja, fazer de sua página um verdadeiro lego, isso mesmo, um lego no qual você encaixa cada peça onde quiser. O grande barato é o ganho que ele dá ao nosso projeto, imagine a cena: você, 1.000 páginas e um cliente impaciente (coisa que não existe), que chega e lhe diz, “Amigo, eu quero mudar a cor, posição, tamanho e tipo do menu principal”! Até aí tudo bem, porque estamos aqui para atender o cliente, não é? Mas você não utilizou boas práticas e não utilizou a *Action include*! Que dó!! Que dó!! Que dó!!



Figura 16 – Resultado de não utilizar as boas práticas
Fonte: O minuto do saber (2011)

Você, 1.000 páginas e o cliente impaciente querendo para ontem o menu alterado, ainda bem que você utilizou as boas práticas e os recursos disponíveis, você utilizou *include*!



Figura 17 – Sucesso por usar boas práticas
Fonte: Shutterstock (2011)

Sim, pequeno gafanhoto, sucesso! Não tem como ser diferente. Vamos às explicações. No case do Unicórnio, *ops!*, quer dizer, do cliente impaciente acima, você utilizou uma *Action* JSP que simplesmente replicou o mesmo arquivo nas 1.000 páginas do projeto fazendo com que, ao invés de modificar as 1.000 páginas, você apenas tenha que realizar a manutenção em uma única página, naquela que foi incluída como menu. Muito bom!

Vamos à prática! Os códigos-fontes “Página inicio.jsp” e “*SNIPPET* menu.jsp” apresentam a página que utiliza a *Action include* e a página que implementa o menu, respectivamente.

```
<body>
  <!-- Aqui está inserido o menu. -->
  <nav> <jsp:include page="menu.jsp"></jsp:include> </nav>
  <hr/>

  <p>Conteúdo da página</p>

</body>
```

Código-fonte 27 – Página inicio.jsp
Fonte: Elaborado pelo autor (2016)

```
<p>SCRIPTS WEB</p>
<ul>
  <li>JSP</li>
  <li>ASP</li>
  <li>PHP</li>
  <li>CFM</li>
  <li>CGI</li>
</ul>
```

Código-fonte 28 – *SNIPPET* menu.jsp
Fonte: Elaborado pelo autor (2016)

A Figura “Resultado da página inicio.jsp com *include* do menu.jsp” exibe o resultado da execução da página.

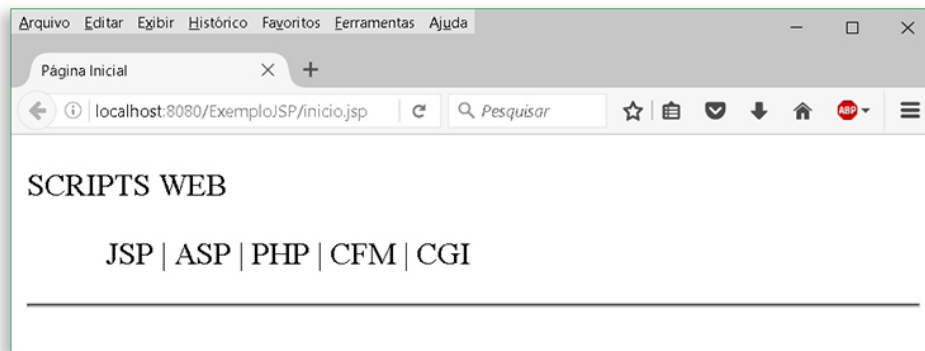


Figura 18 – Resultado da página inicio.jsp com *include* do menu.jsp
Fonte: Elaborado pelo autor (2016)

Você deve estar lembrado de outra forma de incluir uma página JSP em outra: a *Directiva include*. A *Action* e a *Directiva include* produzem o mesmo resultado, entretanto possuem algumas pequenas diferenças. A *Directiva include* insere o JSP no tempo de compilação, assim gera somente uma *Servlet*. Já a *Action include* insere o JSP em tempo de execução, dessa forma gera uma *Servlet* para cada JSP. Ela também pode inserir além de JSP, *Servlets* e HTML. No final, podemos utilizar qualquer um dos métodos para adicionar o menu na nossa aplicação. O importante é não duplicar os códigos!

2.4.2 Action <jsp:forward>

Esta *Action* é utilizada para transferir permanentemente o controle para outro local, onde a propriedade *page* indicar. Esse local sempre é indicado por uma URL relativa.

Trocando em miúdos, podemos utilizar essa *Action* para redirecionar o cliente para qualquer lugar da aplicação. Imagine, por exemplo, um sistema de controle de acesso em que a validação é realizada mediante o nome de usuário e senha. Caso a senha ou o nome de usuário sejam digitados incorretamente, o controle é direcionado para uma página de alerta. O Código-fonte “*Sintaxe Action Forward* sem parâmetros com URL relativa” apresenta um exemplo de redirecionamento:


```
<body>

    <jsp:forward page="sucesso.jsp"/>

</body>
```

Código-fonte 29 – Sintaxe *Action Forward* sem parâmetros com URL relativa
Fonte: Elaborado pelo autor (2016)

Existe uma forma de passar parâmetros no *forward*, facilitando, assim, a vida do desenvolvedor.

Vamos desenvolver um exemplo, implementando uma página que envia e encaminha dois parâmetros para outra página (Código-fonte “Sintaxe *Action forward* com parâmetros com URL relativa”).

```
<jsp:forward page="index.jsp">

    <jsp:param value="Jorginho" name="nome"/>
    <jsp:param value="25" name="idade"/>

</jsp:forward>
```

Código-fonte 30 – Sintaxe *Action forward* com parâmetros com URL relativa
Fonte: Elaborado pelo autor (2016)

Agora vamos criar o código que vai receber os parâmetros no redirecionamento (Código-fonte “Página *index.jsp* recebendo os parâmetros da *Action forward*”).

```
<%--Recuperando os parâmetros passados no forward--%>

<p>Nome :<%=request.getParameter("nome") %></p>
<p>Idade :<%=request.getParameter("idade") %></p>
```

Código-fonte 31 – Página *index.jsp* recebendo os parâmetros da *Action forward*
Fonte: Elaborado pelo autor (2016)

Agora o resultado é apresentado na Figura “Resultado da página *inicio.jsp* com a *Action forward* para a página *index.jsp*”.

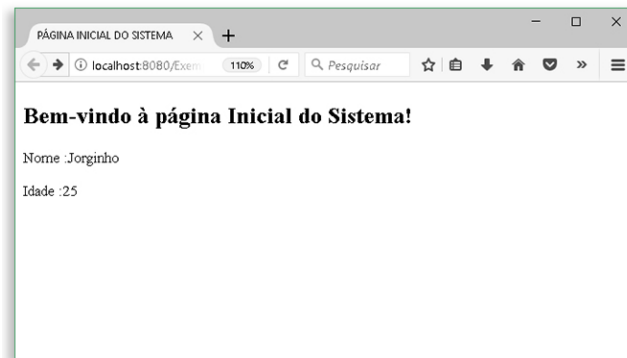


Figura 19 – Resultado da página inicio.jsp com a *Action forward* para a página index.jsp
Fonte: Elaborado pelo autor (2016)

Notem que o endereço na barra do navegador ainda é o da página inicial e não o da página que está na *Action forward*.

Isso acontece pelo simples fato de a *Action forward* encaminhar juntamente para o valor do seu atributo “page” os objetos request/response, e se você tiver colocado parâmetros ali, eles também serão encaminhados.

Então, resumindo, a URL não muda pelo simples fato de o contexto da página atual ser encaminhado para o alvo, dando oportunidade assim para podermos recuperar as informações que foram enviadas.

2.5 Prática!

Pensando no futuro, vamos desenvolver algumas páginas JSP para criar um menu e separar as partes comuns a todas as páginas. Com isso, ganhamos na produtividade e na manutenção de nossas páginas. Primeiramente, vamos criar um projeto. Clique na opção “File” > “New” > “Dynamic Web Project”, conforme a Figura “Criando um projeto Java Web – Parte 1”.

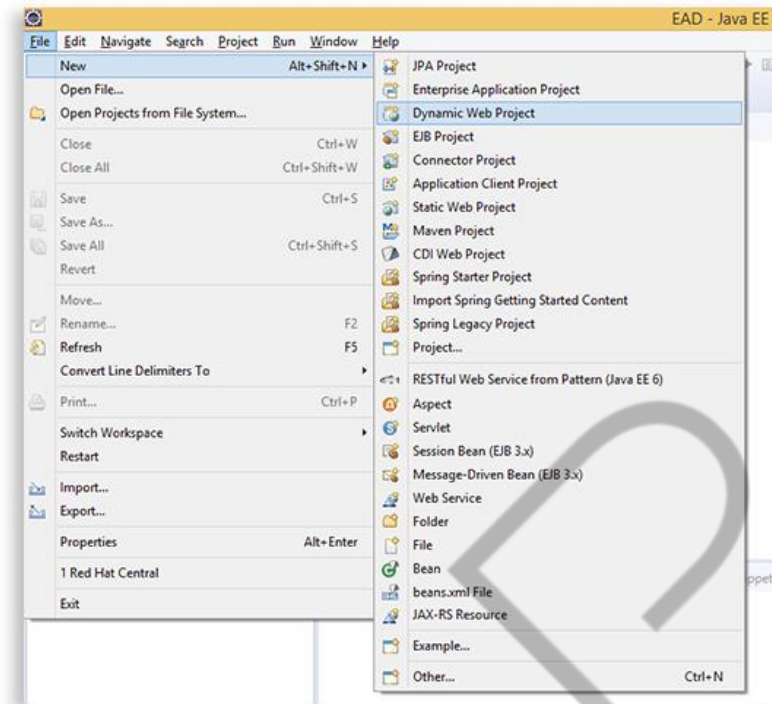


Figura 20 – Criando um projeto Java Web – Parte 1
Fonte: Elaborado pelo autor (2017)

O próximo passo é dar um nome ao projeto, como sugestão, utilizamos o nome “JSPs” (Figura “Criando um projeto Java Web – Parte 2”). Não podemos nos esquecer de configurar o servidor, faremos da mesma forma que fizemos no Capítulo 5. Clique em “New Runtime...”, escolha a opção “Apache Tomcat v9.0”, depois configure o caminho onde o servidor está em sua máquina, por meio do botão “browser” e finalize o processo.

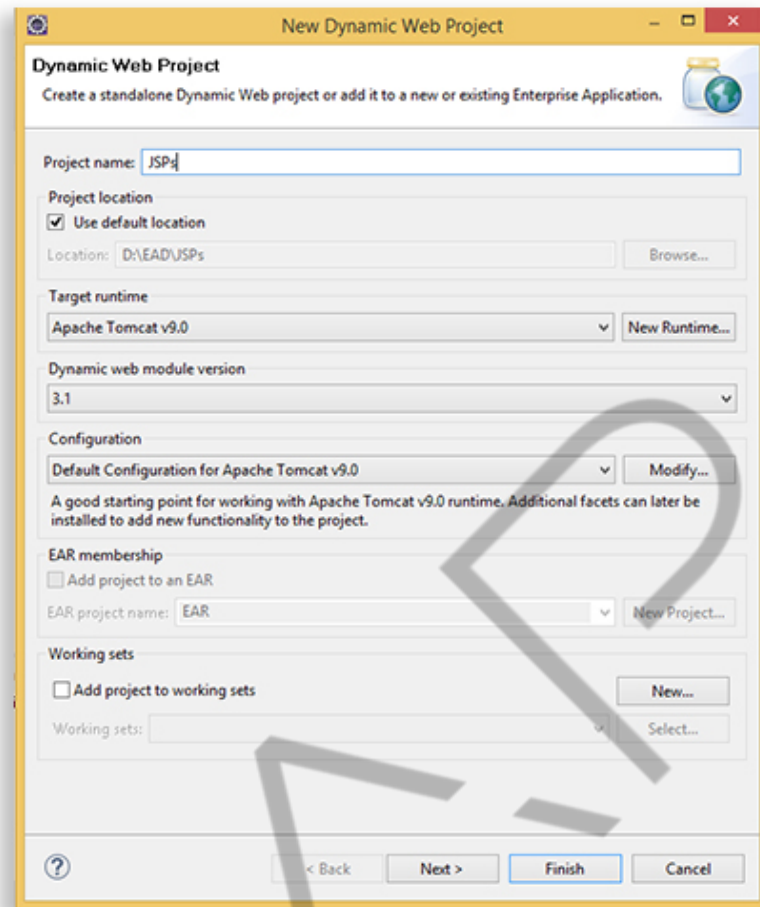


Figura 21 – Criando um novo projeto Java Web – Parte 2
Fonte: Elaborado pelo autor (2017)

Depois de configurar o nome do projeto e o servidor, vamos para os próximos passos, clicando no botão “Next”. Na última tela, marque a opção de criar o arquivo “web.xml”, conforme a Figura “Criando um projeto Java Web – Parte 3”.

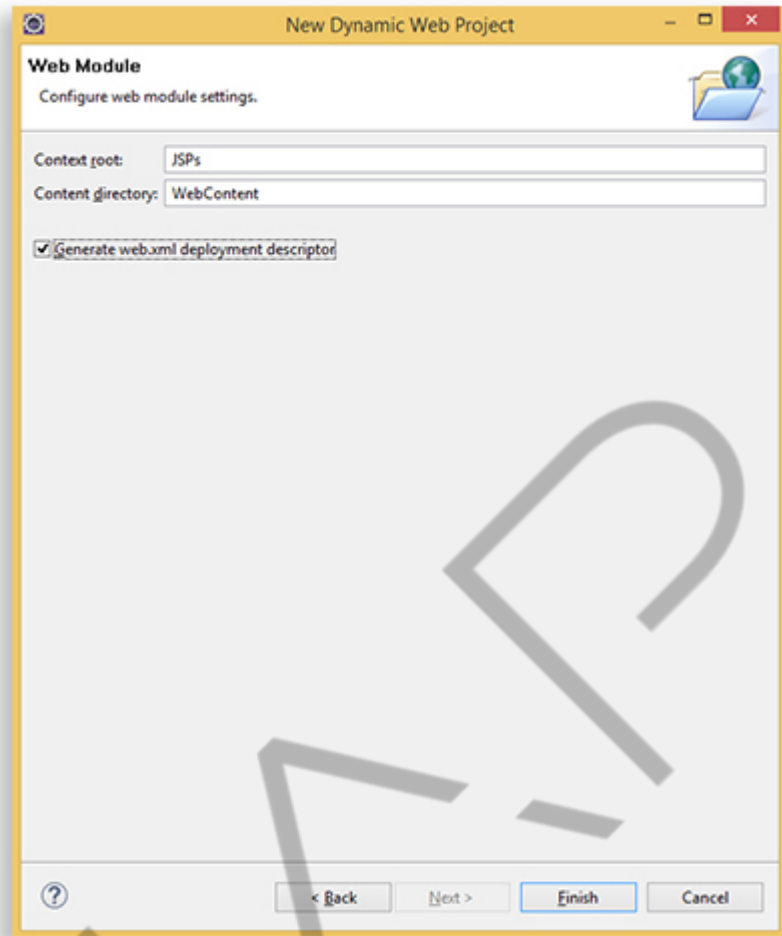


Figura 22 – Criando um novo projeto Java Web – Parte 3
Fonte: Elaborado pelo autor (2017)

Para construir a interface do sistema Fintech, iremos utilizar um *framework* muito utilizado no mercado e que vimos em algumas fases anteriores, o *bootstrap*. Para utilizar o *bootstrap*, vamos baixar os arquivos por meio do endereço: <http://getbootstrap.com/> (Figura “Site do *bootstrap*”).

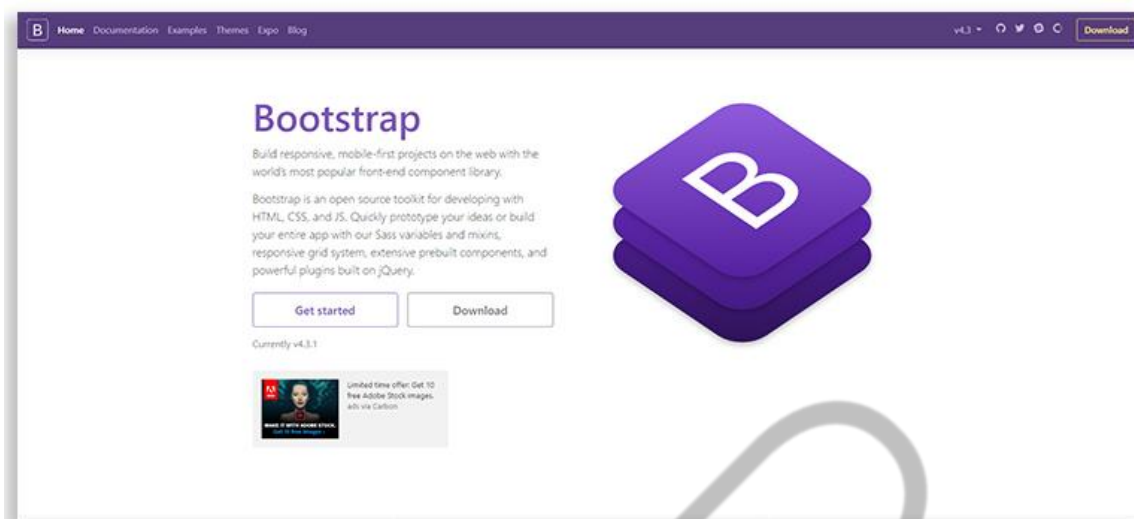


Figura 23 – Site do *bootstrap*
Fonte: Elaborado pelo autor (2017)

Clique no botão “Download” para ir à página de download. Depois, aproveite e dê uma olhada na documentação do *bootstrap* para ver as novidades. A documentação é muito rica e fácil de entender, você vai adorar! Na página de download, utilize o link de download para obter o CSS e JS compilados (Figura “Download do *bootstrap*”).

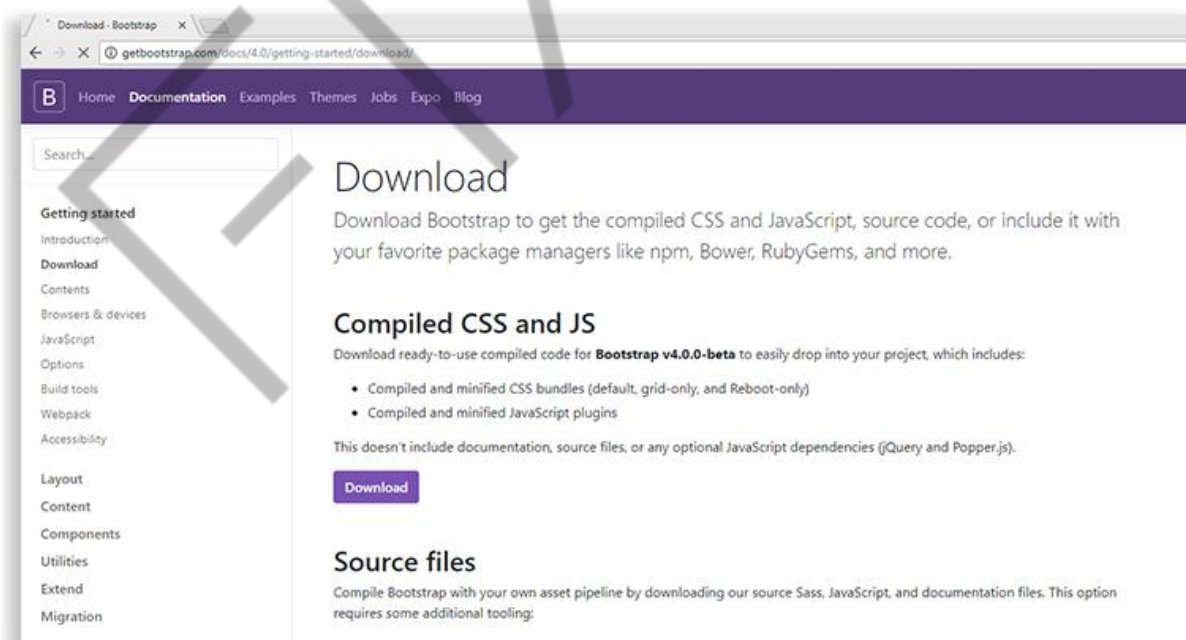


Figura 24 – Download do *bootstrap*
Fonte: Elaborado pelo autor (2017)

Agora que obtivemos os arquivos do *bootstrap*, precisamos copiá-los para dentro do projeto. Porém, para deixar a nossa aplicação bem organizada, vamos criar uma pasta para agrupar os arquivos de css, javascript, imagens, ícones etc.

Para criar essa pasta, clique com o botão direito do mouse na pasta “Web Content” e escolha a opção “New” > “Folder” (Figura “Criando uma nova pasta no projeto”).

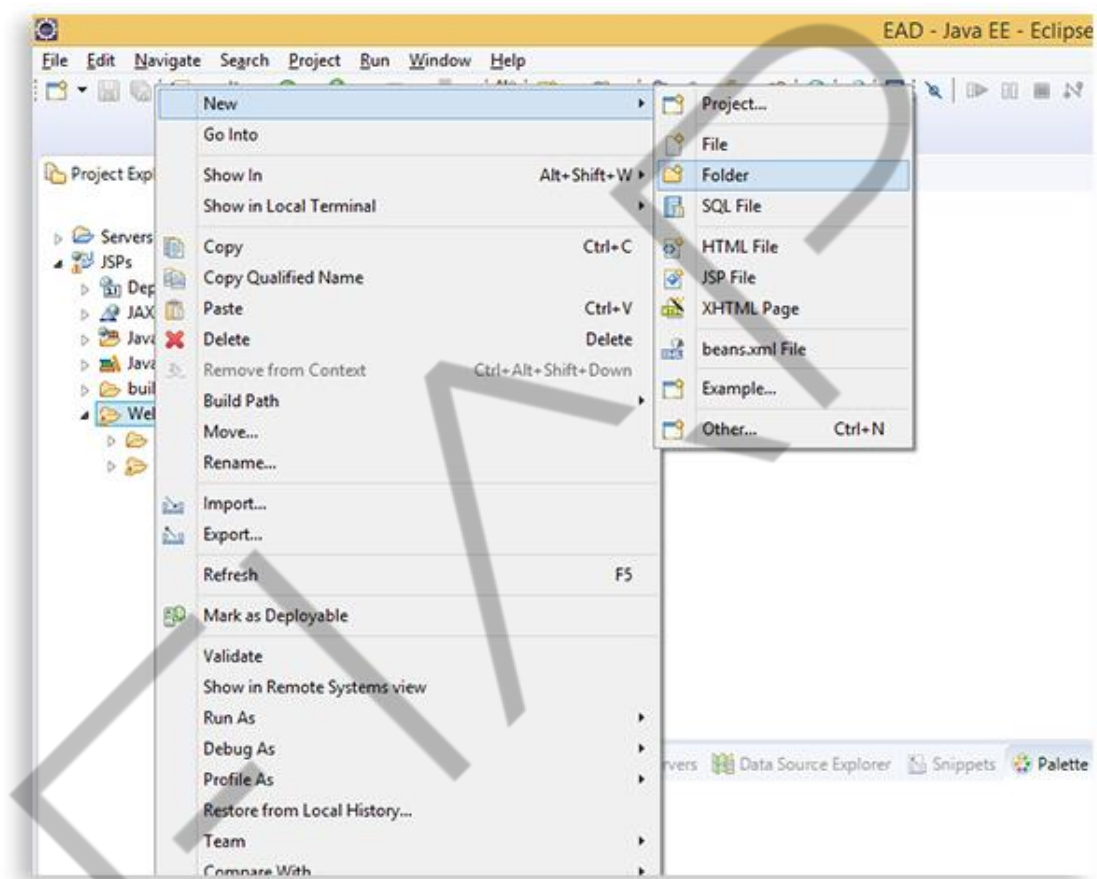


Figura 25 – Criando uma nova pasta no projeto
Fonte: Elaborado pelo autor (2017)

A pasta pode possuir qualquer nome, porém os arquivos css e js geralmente ficam em uma pasta chamada “resources”, ou recursos. Esse é o nome que iremos utilizar em nosso projeto.

Agora podemos copiar os arquivos para dentro dessa pasta. Note que, dentro da pasta *resources*, deixamos a separação dos arquivos javascript e css, cada um em sua respectiva pasta (Figura “Estrutura final dos arquivos do *bootstrap* e JQuery”).

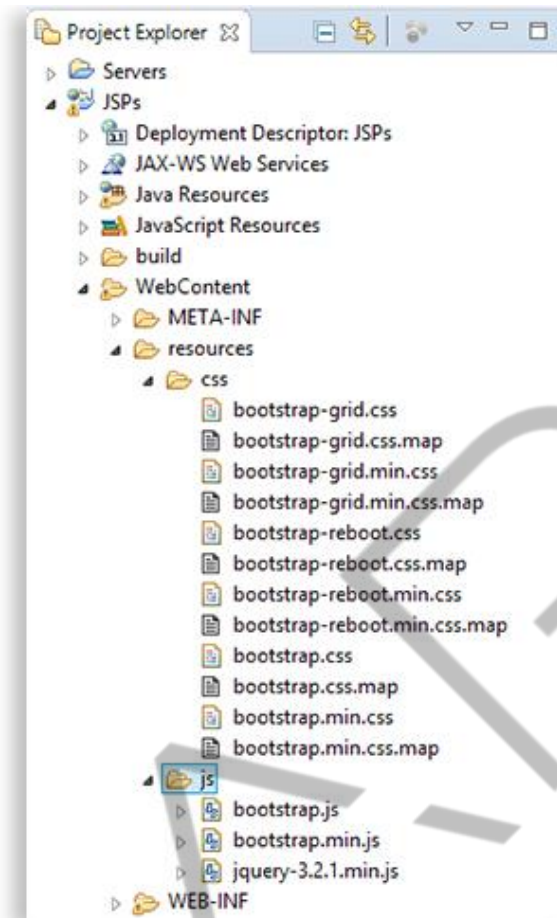


Figura 26 – Estrutura final dos arquivos do *bootstrap* e JQuery
Fonte: Elaborado pelo autor (2017)

Não podemos esquecer que o *bootstrap* depende do JQuery, por isso precisamos copiar o arquivo do JQuery para dentro da pasta resources/js. Para isso, vá até o endereço do JQuery e faça o download (Figura Download do JQuery).

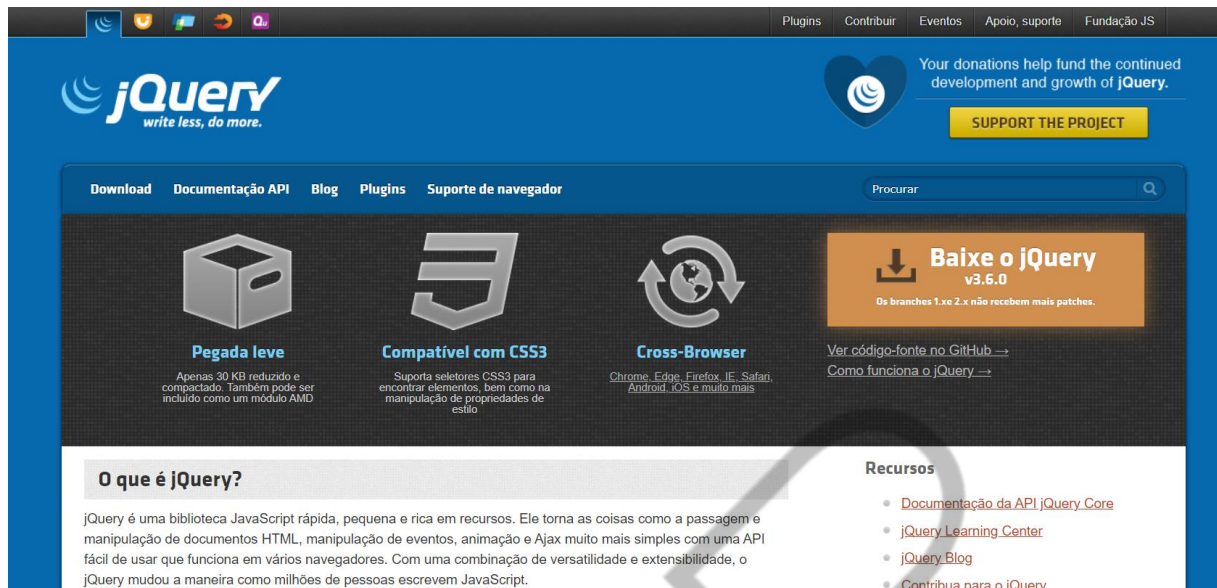


Figura 27 – Download do JQuery
Fonte: Elaborado pelo autor (2021)

Pronto! Agora podemos utilizar o *bootstrap* para construir as páginas da nossa aplicação. Lembra-se de como utilizamos o *bootstrap*? Precisamos adicionar o css e js em todas as páginas, ou seja, precisamos replicar os códigos <link> e <script> em todas as páginas que utilizam o *bootstrap*!

Já vimos a dificuldade e os problemas que isso pode causar. Por isso, vamos criar uma página JSP para adicionar esse código que se repete e fazer com que as outras páginas JSP a utilizem. Assim, caso seja necessária alguma alteração, basta modificar essa página.

Vamos criar o JSP para ter o link do css. Para isso, clique com o botão direito do mouse na pasta “Web Content” e escolha a opção “New” > “JSP File” (Figura “Criando uma nova página JSP – Parte 1”).

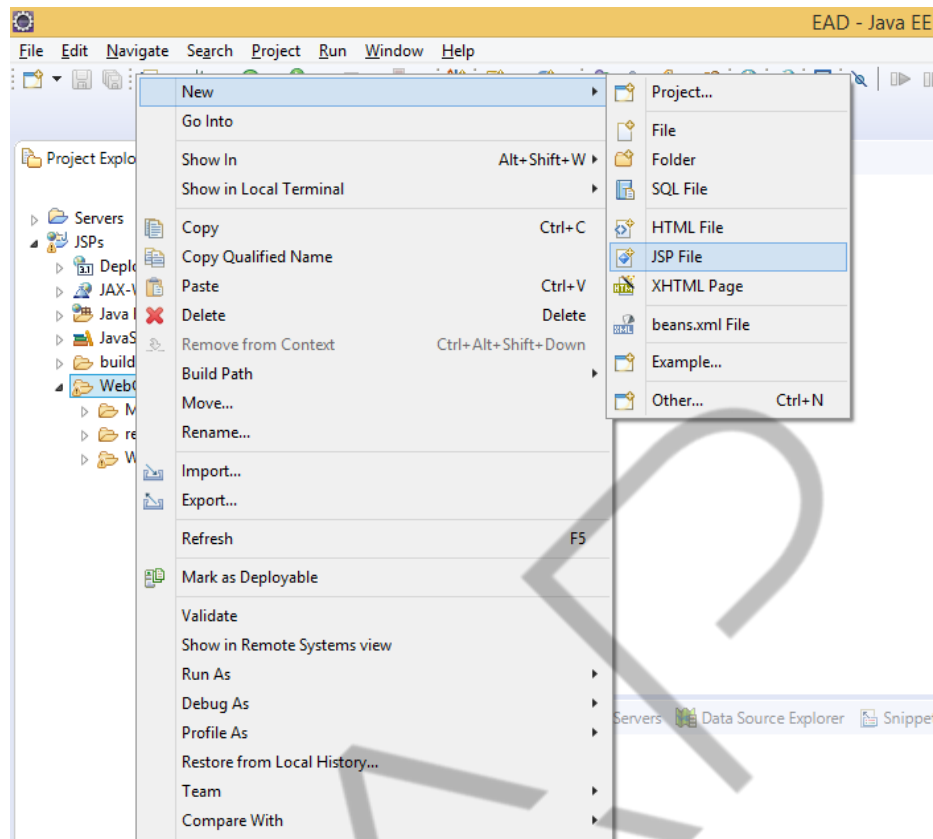


Figura 28 – Criando uma nova página JSP – Parte 1
Fonte: Elaborado pelo autor (2017)

O próximo passo é dar nome ao JSP. Vamos colocar o nome de “*header*” e finalizar o processo (Figura “Criando um novo projeto Java Web – Parte 2”).

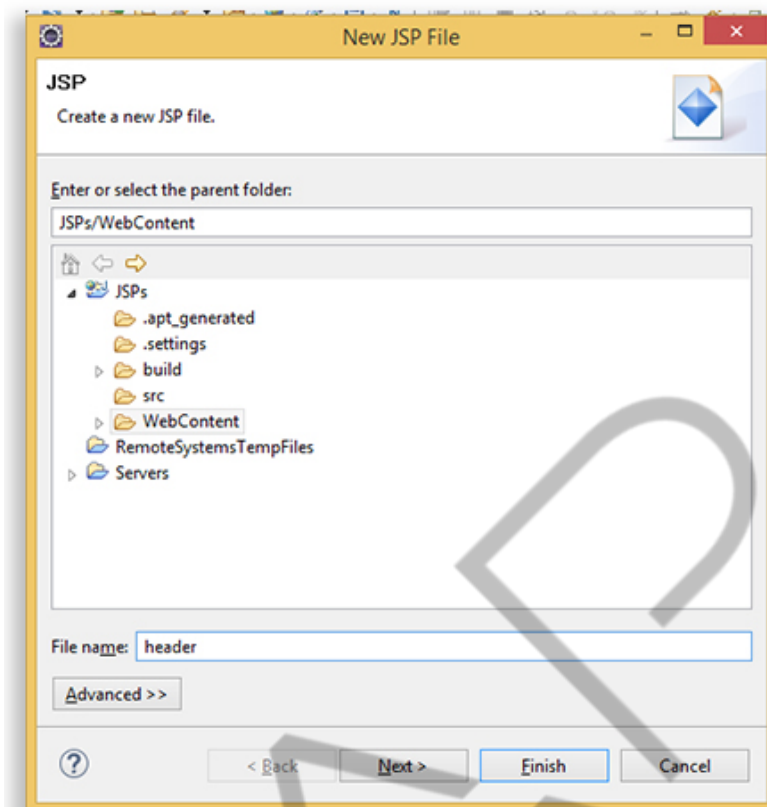


Figura 29 – Criando um novo projeto Java Web – Parte 2
Fonte: Elaborado pelo autor (2017)

Como as outras páginas vão “incluir” o código dessa página que acabamos de criar, não é necessário que tenha a estrutura completa de uma página HTML, mas sim a parte que queremos adicionar nas outras páginas. Assim, como vamos utilizar somente o css do *bootstrap*, adicionaremos o link para esse css (Código-fonte “Página header.jsp com link para o css do *bootstrap*”). É aqui que você também pode adicionar o link para o seu arquivo de css para customização.

```
<link rel="stylesheet" type="text/css" href="resources/css/bootstrap.min.css">
```

Código-fonte 32 – Página header.jsp com link para o css do *bootstrap*
Fonte: Elaborado pelo autor (2016)

Por que esse código não tem os *javascripts*? Claro que poderíamos adicionar os javascript do JQuery e Bootstrap nesse arquivo também, porém esse JSP deve ser “incluído” no cabeçalho (*head*) das páginas e sabemos que é recomendado adicionar os *javascripts* no final da página, já que uma página é carregada de cima para baixo e é melhor carregar primeiramente as informações visuais, para dar a sensação de que a página é carregada mais rápido.

Assim, vamos criar outra página JSP (footer.jsp) para adicionar os *javascripts* e que possa ser incluída no final das páginas (Código-fonte “Página footer.jsp com os javascripts”).

```
<script type="text/javascript" src="resources/js/jquery-3.2.1.min.js"></script>
<script type="text/javascript" src="resources/js/bootstrap.min.js"></script>
```

Código-fonte 33 – Página footer.jsp com os *javascripts*
Fonte: Elaborado pelo autor (2016)

Agora podemos utilizar essas duas páginas JSPs. Para isso, vamos criar uma página de exemplo, que será a página inicial da aplicação. Diferentemente das outras duas páginas, essa terá a estrutura completa de uma página (Código-fonte “Página inicial da aplicação”). No cabeçalho, vamos incluir a página header.jsp, e no final, vamos incluir o código da página footer.jsp.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Home - JSPs</title>
<%@ include file="header.jsp" %>
</head>
<body>
<div class="container">
<h1>JSP - Java ServerPages</h1>
</div>
<%@ include file="footer.jsp" %>
</body>
</html>
```

Código-fonte 34 – Página inicial da aplicação
Fonte: Elaborado pelo autor (2016)

Note que adicionamos também uma `<div>` com a class “*container*” para “zerar” algumas propriedades e dar uma margem à esquerda e à direita. Definimos também um título para a página, para o exemplo.

Para finalizar, vamos criar um menu para a aplicação. Se olharmos a documentação do *bootstrap*, na área de componentes, existem vários exemplos de navbar. Vamos utilizar o primeiro exemplo para adicionar o menu em nosso sistema.

Crie um novo jsp, vamos chamar de “menu.jsp”. Nesse menu, adicione o código da navbar do *bootstrap* (Código-fonte “Menu para a aplicação”).

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse"
id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span
class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled"
href="#">Disabled</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="text"
placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-success my-2 my-sm-
0" type="submit">Search</button>
    </form>
  </div>
</nav>
```

Código-fonte 35 – Menu para a aplicação
Fonte: Elaborado pelo autor (2016)

Observe que fizemos um pequeno ajuste para deixar a navbar de outra cor. Substituímos a classe “navbar-light bg-ligth” por “navbar-dark bg-dark”, para alterar a cor de fundo da barra de navegação.

Agora, basta adicionar o *include* na página inicial (Código-fonte “Página inicial com o menu”).

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Home - JSPs</title>
<%@ include file="header.jsp" %>
</head>
<body>
<%@ include file="menu.jsp" %>
    <div class="container">
        <h1>JSP - Java ServerPages</h1>
    </div>
<%@ include file="footer.jsp" %>
</body>
</html>
```

Código-fonte 36 – Página inicial com o menu
Fonte: Elaborado pelo autor (2016)

O resultado pode ser observado na Figura “Resultado da execução da página inicial”.

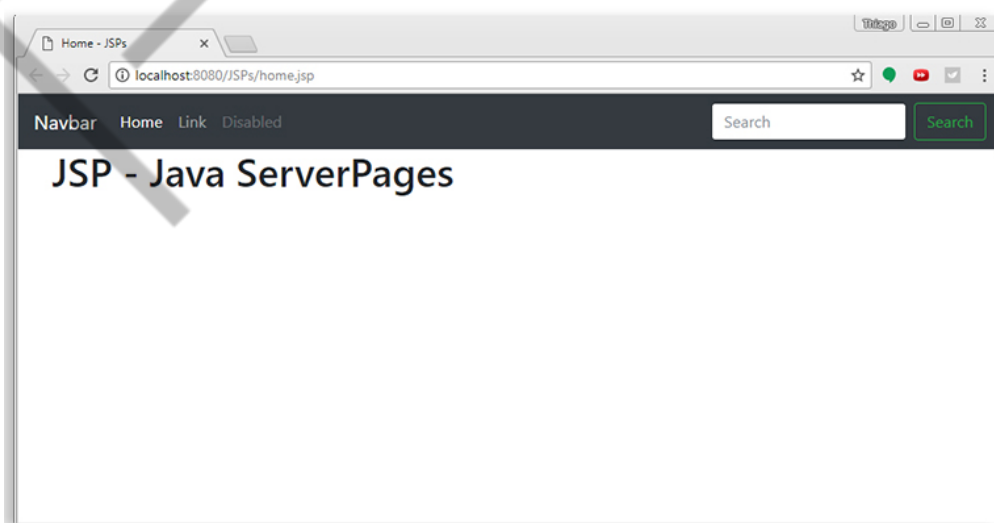


Figura 30 – Resultado da execução da página inicial
Fonte: Elaborado pelo autor (2017)

Não se preocupe em modificar a barra de navegação neste momento. Vamos fazer isso logo mais, quando desenvolvermos a nossa aplicação completa! Nesse exemplo, focamos na implementação de um layout que podemos reaproveitar em todas as páginas e que será muito útil no desenvolvimento do Fintech.



REFERÊNCIAS

SINGH, C. **JSP Interview Questions and Answers**. Disponível em: <<http://beginnersbook.com/category/jsp-tutorial/>>. Acesso em: 13 set. 2021.

TUTORIALSPPOINT. **JSP** – Overview. Disponível em: <https://www.tutorialspoint.com/jsp/jsp_overview.htm>. Acesso em: 13 set. 2021.

EXEMPLO