Sumário

Introdução

A tecnologia JSP é usada para criar aplicativos da Web dinâmicos da mesma forma que a tecnologia Servlet. É outra tecnologia da Web fornecida pela Sun MicroSystem para o desenvolvimento de páginas da Web dinâmicas no navegador do cliente. Ele fornece uma abordagem baseada em tags para desenvolver componentes da Web Java.

JSP tem extensão .jsp , podemos acessar diretamente essas páginas JSP da janela do navegador do sistema cliente. Porque as páginas jsp estão contidas fora da pasta WEB-INF.

Uma página JSP consiste em tags HTML e tags JSP. As páginas jsp são mais fáceis de manter do que servlet. Ele fornece alguns recursos adicionais, como Expression Language, Custom Tag etc.

Um JSP é chamado de página, mas não de programa, porque um JSP contém totalmente tags. Cada JSP é convertida internamente em um Servlet pelo container do servidor.

Por que JSP?

A primeira tecnologia dada para o desenvolvimento de aplicações web é CGI(Common Gateway Interface). Em CGI tem alguma desvantagem, então a Sun MicroSystem desenvolver uma nova tecnologia chamada servlet. Mas trabalhando com Servlet o Sun MicroSystem identifica alguns problemas, a tecnologia Servlet precisa de código java em profundidade e o Servlet não era simples de implementar, logo não era atrativo para programadores. Para superar o problema com a tecnologia Servlet, usamos a tecnologia jsp.

Tag JSP

Uma página JSP contém tags html e tags JSP. As tags HTML produzirão uma saída estática e as tags JSP produzirão uma saída dinâmica. Por causa das tags JSP que chamamos de JSP é uma página da web dinâmica.

Exemplo:

```
<html>
  <head><title>Hello World</title></head>

  <body>
    Hello World!<br/>
    <%
      out.println("Seu ip é: " + request.getRemoteAddr());
    %>
  </body>
</html>
```

API JSP

Uma API é uma coleção de pacotes, classes, interfaces e subpacotes. E o subpacote também é uma coleção de interfaces de classes e sub subpacotes etc.

A API JSP consiste em dois pacotes;

Interfaces:

javax.servlet.jsp

javax.servlet.jsp.tagext

Classes:

JspWriter

JspError

PageContext

JspFactory

JspEngineInfo

JspException

JspPage Interface

A interface JspPage forneceu dois métodos de ciclo de vida.

public void **jspInit()**: Este método é invocado apenas uma vez durante o ciclo de vida da JSP quando a primeira requisição chega à página JSP é solicitada. É usado para executar a inicialização. Este método é igual ao método init() do Servlet

public void **jspDestroy()**: Este método é invocado apenas uma vez durante o ciclo de vida do JSP antes que a página JSP seja destruída.

Interface HttpJspPage

A interface HttpJspPage forneceu um único método de ciclo de vida chamado _jspService().

public void _jspService(): Este método é invocado sempre que a solicitação da página JSP chega ao contêiner. Ele é usado para processar a solicitação. O sublinhado _ significa que você não pode substituir esse método.

Diferença entre JSP e HTML

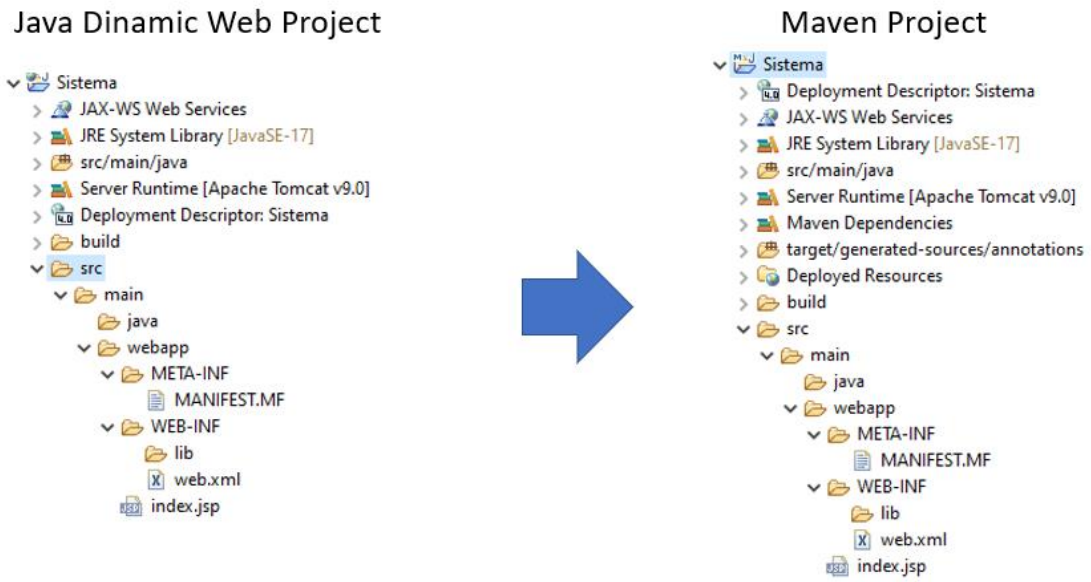
Html é uma tecnologia do lado do cliente e JSP é uma tecnologia do lado do servidor. Algumas outras diferenças são dadas abaixo;

	HTML	JSP
1	Html é dado pelo w3c (World Wide Web Consortium).	JSP é fornecido pelo SunMicro System.
2	Páginas da Web estáticas geradas em HTML.	JSP gerou páginas da web dinâmicas.
3	Não permite colocar código Java dentro de páginas Html.	JSP permite colocar código java dentro de páginas JSP.
4	É a tecnologia do lado do cliente	É uma tecnologia do lado do servidor.
5	Precisa de Html Interpreter para executar esses códigos.	Precisa de contêiner JSP para executar o código jsp.
6	Não permite colocar tag personalizada ou tag de terceiros.	Permite colocar etiqueta personalizada ou etiqueta de terceiros.

Configurando o workspace com Maven

https://maven.apache.org/archetypes/maven-archetype-webapp/

Outra maneira é gerar um Java Dinamic Web Project e converter para Maven.



- lib: onde são adicionadas as bibliotecas, caso não referenciadas no pom.xml.
- webapps: é a pasta onde ficam todas as páginas bibliotecas, imagens, css, js, etc.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<marquee><h1>Agora é a todo vapor!</h1></marquee>
</body>
</html>
```

Classpath

A variável classpath é definida para fornecer o caminho de todas as classes Java que são usadas em nosso aplicativo.

Set Classpath for JSP

Copy jsp-api.jar file location and set the classpath in environment variable.

If your classpath is already set for core java programming, you need to edit classpath variable. for edit classpath just put ; at end of previous variable and paste new copied location (without delete previous classpath cariable).

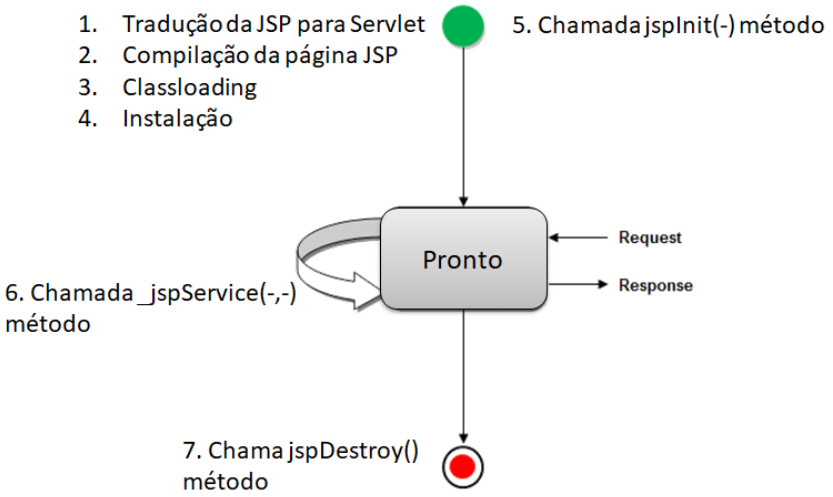
Ciclo de Vida JSP

Um ciclo de vida JSP pode ser definido como todo o processo desde a sua criação até a destruição, que é semelhante ao ciclo de vida de um Servlet com uma etapa adicional que é necessária para converter um JSP em Servlet.

Ciclo de vida de uma página JSP

- Tradução de página JSP para Servlet
- Compilação da página JSP
- Carregamento de classe (o arquivo de classe é carregado pelo carregador de classe)
- Instanciação (Objeto do Servlet Gerado é criado).
- Inicialização (método jspInit() é invocado pelo contêiner).
- Processamento de solicitação (o método _jspService() é invocado pelo contêiner).
- Destroy (método jspDestroy() é invocado pelo contêiner).

Exemplificando graficamente:



Método de ciclo de vida de JSP

jspInit()

_jspService()

jspDestroy()

Podemos substituir jspInti(), jspDestroy(). Mas não podemos substituir o método _jspService().

Para informar ao programador que você não deve substituir o método service(), o nome do método é iniciado com o símbolo '_'.

Tradução JSP

Cada página JSP é traduzida internamente em um Servlet equivalente. Cada tag JSP escrita na página será convertida internamente em um código java equivalente pelos containers, este processo é chamado de tradução .

Existem duas fases que ocorrem no JSP;

Fase de tradução

Fase de processamento da solicitação

Fase de tradução: É o processo de conversão do jsp em um Servlet equivalente e então gerar o arquivo de classe do Servlet.

Fase de processamento da solicitação: É o processo de execução do service() do jsp e geração de dados de resposta no navegador.

Na primeira vez que um pedido para um jsp ocorre, a fase de tradução ocorre primeiro e, em seguida, a fase de serviço será executada. Da próxima requisição para o jsp, somente a fase de processamento do pedido é executada, mas a fase de tradução não é porque o jsp já está traduzido.

Nos dois casos seguintes ocorre apenas a tradução;

Quando o primeiro pedido é dado ao jsp.

Quando um jsp é modificado.

Se um pedido for dado ao jsp depois de ter sido modificado, novamente a fase de tradução é executada primeiro e depois dessa fase de processamento do pedido será executada.

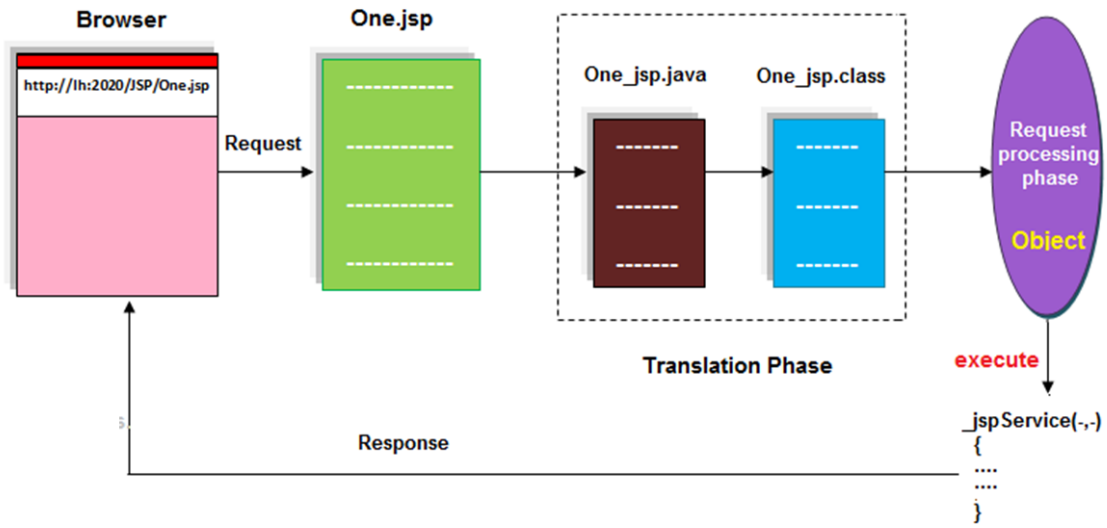
Observação:

Se desligarmos e reiniciarmos o Servlet então somente a fase de processamento de requisições será executada, pois quando desligarmos o servidor java e arquivos de classe não são deletados do servidor.

Se removermos o arquivo de classe do servidor, ocorrerá uma tradução parcial.

Se removermos o java e o arquivo de classe do servidor, ocorrerá novamente a tradução.

Quando um jsp é traduzido em um Servlet equivalente, esse Servlet estende alguma classe base fornecida pelo servidor, mas internamente estende HttpServlet.



Configurando o arquivo JSP

A configuração de um JSP em um arquivo web.xml é opcional porque o JSP é um arquivo público para o aplicativo da web.

Um JSP chamado de arquivo público e Servlet é chamado de arquivo privado do aplicativo da web. Porque os arquivos JSP armazenados no diretório raiz do aplicativo da web e o arquivo de classe Servlet armazenado no subdiretório do aplicativo da web. Como JSP é o arquivo público, podemos enviar uma solicitação diretamente de um navegador usando seu nome de arquivo.

Se quisermos configurar um JSP no arquivo web.xml, os elementos xml são os mesmos da configuração do Servlet. Precisamos substituir o elemento <servlet-class> por <jsp-file>

Sintaxe:

```
<web-app>
<servlet>
<servlet-name>teste</servlet-name>
<jsp-file>/One</jsp-file>
</servlet>
<servlet-mapeamento>
<servlet-name>teste</>
<url-pattern>/srv1</url-pattern>
</servlet-mapeamento>
</web-app>
```

Nota: Se configurarmos um jsp em web.xml, podemos enviar a solicitação para o jsp usando o nome do arquivo jsp ou usando seu padrão de url.

Exemplo:
http://localhost:2014/root/One.jsp
ou
http://localhost:2014/root/srv1

Primeiro programa JSP

O programa JSP deve ser salvo com a extensão .jsp

Exemplo:

```
<html>
<corpo>
<% out.print("Olá palavra"); %>
</body>
</html>
```

Resultado:
Olá palavra

Como executar o programa jsp

Depois de escrever seu código jsp, implante o programa jsp em seu diretório raiz e inicie o servidor. Siga as etapas abaixo para executar o código jsp.

Visite no navegador ou faça o pedido com url como abaixo:

Sintaxe:
http://localhost:8080/**nome_do_projeto**/**nome_arquivo**.jsp

Exemplo:
http://localhost:8080/aula09082022/index.jsp

Script

O elemento JSP Script é escrito dentro <% %> das tags. Esses códigos dentro <% %> das tags são processados pelo mecanismo JSP durante a tradução da página JSP. Qualquer outro texto na página JSP é considerado como código HTML ou texto simples.

Existem cinco tipos diferentes de elementos de script:

Elemento de script	Exemplo
Comente	<%-- comentário --%>
Diretiva	<%@ diretiva %>
Declaração	<%! declarações %>
Roteiro	<% scripts %>
Expressão	<%= expressão %>

Exemplo:

```
<html>
   <head>
      <title>Minha Primeira Página JSP </title>
   </head>
   <%
     int count = 0;
     String turma = “BT002”;
   %>
   <body>
      <%-- Será mostrado exemplo de contagem e String abaixo  --%>
      Page Count is <% out.println(++count); %>
      Boa noite <% out.println(turma); %>
   </body>
</html>
```

Exemplo de Tag Scriptlet JSP

Neste exemplo, criaremos uma página JSP simples que recupera o nome do usuário do parâmetro request. A página index.html obterá o nome de usuário do usuário.

index.html

```
<form method="POST" action=" bemvindo.jsp">
   Name <input type="text" name="usuario" >
   <input type="submit" value="Submit">
</form>
```

No arquivo HTML acima, criamos um formulário, com um campo de texto de entrada para o usuário inserir seu nome e um botão Enviar para enviar o formulário. No envio, uma solicitação HTTP Post ( ) é feita para o arquivo welcome.jsp ( ), com os valores do formulário.method="POST"action="welcome.jsp"

bemvindo.jsp

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Welcome Page</title>
  </head>
  <%
   String user = request.getParameter("usuario");
  %>

  <body>
    Olá, <% out.println(user); %>
  </body>
</html>
```

Como sabemos que um código JSP é traduzido para código Servlet, no qual _jspServiceé executado o método que tem como argumento HttpServletRequest e HttpServletResponse . Assim, no arquivo welcome.jsp , request é o HTTP Request e tem todos os parâmetros enviados do formulário na página index.html, que podemos obter facilmente usando o getParameter()nome do parâmetro como argumento, para obter seu valor.

Misturando Tag de scriptlet e HTML

Vamos ver como podemos utilizar o poder do script JSP com HTML para construir páginas web dinâmicas com a ajuda de alguns exemplos.

Se quisermos criar uma Tabela em HTML com alguns dados dinâmicos, por exemplo lendo dados de alguma tabela ou arquivo MySQL. Como fazer isso? Aqui, descreveremos a técnica criando uma tabela com os números de 1 a n.

```
<table border = 1>
<%
  for ( int i = 0; i < n; i++ ) {
    %>
    <tr>
    <td>Número</td>
    <td><%= i+1 %></td>
    </tr>
    <%
  }
%>
</table>
<table border = 1>
<%
  for ( int i = 0; i < n; i++ ) {
    %>
    <tr>
    <td>Número</td>
    <td><%= i+1 %></td>
    </tr>
    <%
  }
%>
</table>
```

O trecho de código acima ficará dentro da <body>tag do arquivo JSP e funcionará quando você inicializar ncom algum valor.

Além disso, observe atentamente que incluímos apenas o código java dentro da tag scriptlet, e toda a parte HTML está fora dela. Da mesma forma, podemos fazer muitas coisas.

Aqui está mais um exemplo muito simples:

```
<%
   boolean hello = true;
   if ( hello ) {
      %>
      <p>Olá, BT002</p>
      <%
   } else {
      %>
      <p>Até logo, BT002</p>
      <%
   }
%>
```

O código acima está usando a condição if-else para avaliar o que mostrar, com base no valor de uma variável booleana chamada hello.

Etiqueta de declaração JSP

Sabemos que no final uma página JSP é traduzida para a classe Servlet. Portanto, quando declaramos uma variável ou método em JSP dentro da Tag de Declaração , significa que a declaração é feita dentro da classe Servlet, mas fora do método service (ou qualquer outro). Você pode declarar membro estático, variável de instância e métodos dentro da Tag de Declaração . Sintaxe da etiqueta de declaração:

```
<%! declaration %>
```

Exemplo de etiqueta de declaração:

```
<html>
   <head>
      <title>Minha Primeira Página JSP </title>
   </head>
   <%!
      int count = 0;
   %>
   <body>
      Contagem atual é:
      <% out.println(++count); %>
   </body>
</html>
```

Quando usar a tag de declaração e não a tag scriptlet

Se você deseja incluir qualquer método em seu arquivo JSP, deve usar a tag de declaração, pois durante a fase de tradução de JSP, métodos e variáveis dentro da tag de declaração tornam-se métodos de instância e variáveis de instância e também recebem valores padrão.

Por exemplo:

```
<html>
   <head>
      <title>Minha Primeira Página JSP</title>
   </head>
   <%!
      int count = 0;
      int getCount() {
         System.out.println( "In getCount() method" );
         return count;
      }
   %>
   <body>
      Page Count is:
      <% out.println(getCount()); %>
  </body>
</html>
```

O código acima será traduzido para o seguinte servlet:

```
public class hello_jsp extends HttpServlet
{
  int count = 0;
  int getCount() {
     System.out.println( "In getCount() method" );
     return count;
  }
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
                        throws IOException,ServletException
   {
     PrintWriter out = response.getWriter();
     response.setContenType("text/html");
     out.write("<html><body>");

     out.write("Page count is:");
     out.print(getCount());
     out.write("</body></html>");
   }
}
```

Enquanto, qualquer coisa que adicionamos na tag scriptlet, vai dentro do _jspservice()método, portanto não podemos adicionar nenhuma função dentro da tag scriptlet, pois na compilação

ele tentará criar uma função getCount()dentro do método service, e em Java , método dentro de um método não é permitido.

Etiqueta de Diretiva JSP

A Tag Diretiva fornece instruções especiais ao Web Container no momento da tradução da página. As tags diretivas são de três tipos: page , include e taglib .

Diretiva	Descrição
<%@ page ... %>	define propriedades dependentes da página, como idioma, sessão, errorPage etc.
<%@ include ... %>	define o arquivo a ser incluído.
<%@ taglib ... %>	declara a biblioteca de tags usada na página

Você pode colocar a diretiva de página em qualquer lugar no arquivo JSP, mas é uma boa prática torná-la a primeira instrução da página JSP.

A diretiva Page define várias propriedades dependentes da página que se comunicam com o Web Container no momento da tradução. A sintaxe básica do uso da diretiva de página é <%@ page attribute="value" %>onde os atributos podem ser um dos seguintes:

- atributo de importação
- atributo de idioma
- estende atributo
- atributo de sessão
- atributo isThreadSafe
- atributo isErrorPage
- atributo errorPage
- atributo contentType
- atributo autoFlush
- atributo de buffer

O atributo **import** define o conjunto de classes e pacotes que devem ser importados na definição de classe do servlet. Por exemplo:

<%@ page import="java.util.Date" %>

or

<%@ page import="java.util.Date,java.net.*" %>

O atributo **language** define a linguagem de script a ser usada na página.

O atributo **extends** define o nome da classe da superclasse da classe servlet que é gerada a partir da página JSP.

O atributo **session** define se a página JSP está participando de uma sessão HTTP. O valor é verdadeiro ou falso.

O atributo **isThreadSafe** declara se o JSP é thread-safe. O valor é verdadeiro ou falso

O atributo **isErrorPage** declara se a página JSP atual representa a página de erro de outra JSP.

O atributo **errorPage** indica outra página JSP que tratará todas as exceções de tempo de execução lançadas pela página JSP atual. Ele especifica o caminho da URL de outra página para a qual um pedido deve ser despachado para manipular exceções de tempo de execução lançadas pela página JSP atual.

O atributo **contentType** define o tipo MIME para a resposta JSP.

O atributo **autoFlush** define se a saída em buffer é liberada automaticamente. O valor padrão é verdadeiro".

O atributo **buffer** define como o buffer é tratado pelo objeto out implícito.

Tag de expressão JSP

Expression Tag é usado para imprimir a expressão da linguagem Java que é colocada entre as tags. Uma tag de expressão pode conter qualquer expressão da linguagem Java que possa ser usada como argumento para o método out.print() . Sintaxe da Tag de Expressão

<%= Java Expression %>

Exemplo:

<%= (2*5) %>

Ele se transforma nisso:

out.print((2*5));

Nota: Nunca termine uma expressão com **ponto e vírgula** dentro da Expression Tag. Assim:

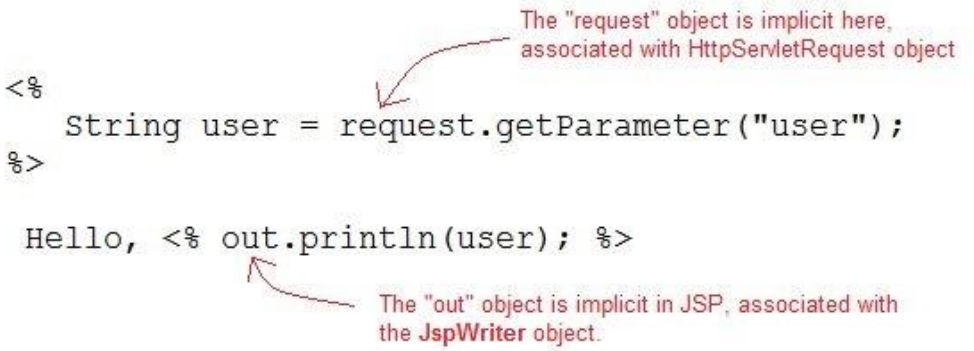
<%= (2*5); %>

Exemplo de Tag de Expressão

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
        <%
    int count = 0;
        %>
        <body>
     Page Count is <%= ++count %>
        </body>
</html>
```

Objetos implícitos em JSP

JSP fornece acesso a algum objeto implícito que representa alguns objetos comumente usados para servlets que os desenvolvedores de páginas JSP podem precisar usar. Por exemplo, você pode recuperar dados de parâmetro de formulário HTML usando a variável de solicitação , que representa o objeto HttpServletRequest .



A seguir está o objeto implícito JSP

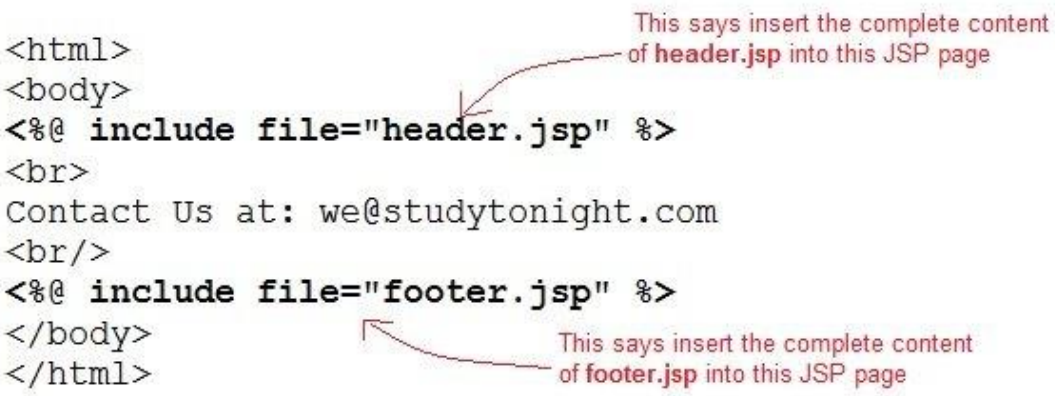
Objeto implícito	Descrição
solicitar	O objeto HttpServletRequest associado à solicitação.
resposta	O objeto HttpServletRequest associado à resposta que é enviada de volta ao navegador.
Fora	O objeto JspWriter associado ao fluxo de saída da resposta.
sessão	O objeto HttpSession associado à sessão para o determinado usuário da solicitação.
inscrição	O objeto ServletContext para o aplicativo da web.
configuração	O objeto ServletConfig associado ao servlet para a página JSP atual.
pageContext	O objeto PageContext que encapsula o ambiente de uma única solicitação para esta página JSP atual
página	A variável de página é equivalente a esta variável da linguagem de programação Java.
exceção	O objeto de exceção representa o objeto Throwable que foi lançado por alguma outra página JSP.

Todos eles são muito úteis e você os conhecerá lentamente à medida que avança em sua carreira trabalhando em projetos ao vivo. Por exemplo: Quando você vai criar um aplicativo em que as sessões do usuário devem ser criadas sessionentrará em cena, requesté usado quando você tem envios de formulários em seu aplicativo etc.

Diretiva de inclusão JSP

A diretiva include diz ao Web Container para copiar tudo no arquivo incluído e colá-lo no arquivo JSP atual. A sintaxe da diretiva include é:

<%@ include file="**nome_do_arquivo**.jsp" %>



Exemplo de diretiva de inclusão

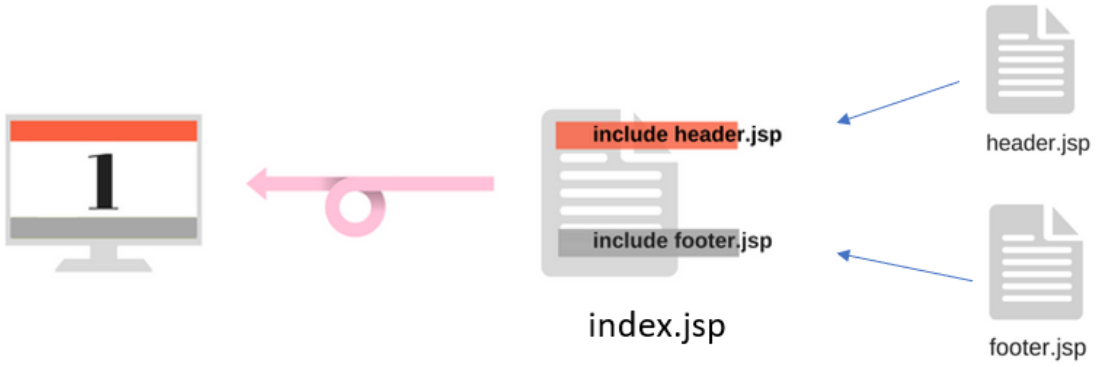
bem-vindo.jsp

```
<html>
  <head>
    <title>Bem-Vindo</title>
  </head>
 <body>
    <%@ include file=" cabeçalho.jsp" %>
    Bem-vindo, Usuário
 </body>
</html>
```

cabeçalho.jsp

```
<html>
 <body>
   <img src="sua_imagem.jpg" alt="Texto da imagem" / >
 </body>
</html>
```

O exemplo acima está mostrando uma prática muito padrão. Sempre que estamos construindo um aplicativo da Web, com páginas da Web, todas com a barra de navegação superior e o rodapé inferior. Nós os criamos como arquivos jsp separados e os incluímos usando a include diretiva em todas as páginas. Portanto, sempre que tivermos que atualizar algo na barra de navegação superior ou no rodapé, basta fazer isso em um só lugar. Útil, não é?



Diretiva JSP Taglib

A diretiva taglib é usada para definir a biblioteca de tags que a página JSP atual usa. Uma página JSP pode incluir várias bibliotecas de tags. JavaServer Pages Standard Tag Library (JSTL), é uma coleção de tags JSP úteis, que fornece as principais funcionalidades comumente usadas. Ele tem suporte para muitas tarefas estruturais gerais, como iteração e condicionais, tags prontas para manipulação de documentos XML, tags de internacionalização e para realizar operações SQL. A sintaxe da diretiva taglib é:

<%@ taglib prefix="prefixOfTag" uri="uriOfTagLibrary" %>

O prefixo é usado para distinguir a tag personalizada de outra tag personalizada da biblioteca. O prefixo é anexado ao nome da tag personalizada. Cada tag personalizada deve ter um prefixo.

O URI é o nome exclusivo da Biblioteca de tags.

Você pode nomear o prefixo qualquer coisa, mas deve ser único.

JSP: usando a diretiva Taglib

Para usar o JSTL em seu aplicativo, você deve ter o jstl.jarno diretório webapps /WEB-INF/lib. Baixe o arquivo jar empágina.

Existem muitas bibliotecas JST prontas disponíveis que você usa para facilitar sua vida. A seguir está uma ampla divisão em diferentes grupos de bibliotecas JST:

Tags principais - URI → http://java.sun.com/jsp/jstl/core

Formatando Tags - URI → http://java.sun.com/jsp/jstl/fmt

Tags SQL - URI → http://java.sun.com/jsp/jstl/sql

Tags XML - URI → http://java.sun.com/jsp/jstl/xml

Funções JSTL - URI → http://java.sun.com/jsp/jstl/functions

Tratamento de Exceções JSP

Manipulação de Exceções é um processo de manipulação de condições excepcionais que podem ocorrer em seu aplicativo. O tratamento de exceção em JSP é muito mais fácil do que o tratamento de exceção da tecnologia Java. Embora a Tecnologia JSP também use os mesmos objetos de classe de exceção.

É bastante óbvio que você não deseja mostrar o rastreamento de pilha de erros para nenhum usuário aleatório navegando em seu site. Você não pode evitar todos os erros em seu aplicativo, mas pode pelo menos fornecer uma página de resposta de erro amigável.

Maneiras de executar o tratamento de exceção em JSP

JSP fornece 3 maneiras diferentes de realizar o tratamento de exceção:

Usando o atributo isErrorPage e errorPage da diretiva de página.

Usando a tag <error-page> no Deployment Descriptor .

Usando try...catchbloco simples.

Exemplo de atributo isErrorPage e errorPage

isErrorPageO atributo na diretiva de página nomeia oficialmente uma página JSP como uma página de erro.

error.jsp

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

<html>
<body>

<strong>You are here because we are not able to
find the page you have asked for.</stong>

<img src="userFriendlyImage.jpg" />

</body>
</html>
```

This attribute officially designate this page as an error page.

errorPage O atributo em uma diretiva de página informa ao Web Container que, se ocorrer uma exceção na página atual, encaminhe a solicitação para a página de erro especificada.

soma.jsp

```
<%@ page errorPage="error.jsp" %>

<html>
<body>

<% int x=10/0; %>

The sum is <%= x %>

</body>
</html>
```

Tells the Web Container that if some exception occurs here, forward the request to **error.jsp**

Sempre que ocorre uma exceção na página sum.jsp o usuário é redirecionado para a página error.jsp , onde você pode exibir uma mensagem legal, ou também pode imprimir o rastreamento de exceção em um arquivo/banco de dados em segundo plano, para verificar posteriormente o que causou o erro.

Declarando a página de erro no Deployment Descriptor

Você também pode declarar páginas de erro no DD para todo o aplicativo da Web. Usando <error-page>tag no Deployment Descriptor . Você pode até configurar diferentes páginas de erro para diferentes tipos de exceção ou tipo de código de erro HTTP (503, 500 etc).

Declarando uma página de erro para todos os tipos de exceção:

```
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.jsp</location>
</error-page>
```

Declarando uma página de erro para uma exceção mais detalhada

```
<error-page>
<exception-type>java.lang.ArithmeticException</exception-type>
<location>/error.jsp</location>
</error-page>
```

Declarando uma página de erro com base no código de status HTTP

```
<error-page>
<error-code>404</error-code>
<location>/error.jsp</location>
</error-page>
```

Usando o bloco try...catch

Usar try...catcho bloco é exatamente como é usado no Core Java .

```
<html>
<head>
  <title>Try...Catch Exemplo</title>
</head>
<body>
 <%
  try{
    int i = 100;
    i = i / 0;
    out.println("A resposta é: " + i);
  }
  catch (Exception e){
    out.println("Uma exceção ocorreu: " + e.getMessage());
  }
 %>
</body>
</html>
```

Tag padrão JSP (elemento de ação)

A especificação JSP fornece tags Padrão (Ação) para uso em suas páginas JSP. Essas tags são usadas para remover ou eliminar o código scriptlet da sua página JSP porque o código scriplet não é tecnicamente recomendado hoje em dia. É considerado uma má prática colocar código Java diretamente dentro de sua página JSP.

As tags padrão começam com o jsp:prefixo. Existem muitas tags de ação padrão JSP que são usadas para executar alguma tarefa específica.

A seguir estão algumas tags de ação padrão JSP disponíveis:

Tag de ação	Descrição
jsp:forward	encaminhar a solicitação para uma nova página Uso:<jsp:forward page="Relative URL" />
jsp:useBean	instancia um JavaBean Uso:<jsp:useBean id="beanId" />
jsp:getProperty	recupera uma propriedade de uma instância JavaBean. Uso: <jsp:useBean id="beanId" ... /> ... <jsp:getProperty name="beanId" property="someProperty" .../> Onde, beanName é o nome do bean pré-definido cuja propriedade queremos acessar.
jsp:setProperty	armazenar dados na propriedade de qualquer instância JavaBeans. Uso: <jsp:useBean id="beanId" ... /> ... <jsp:setProperty name="beanId" property="someProperty" value="some value"/> Onde, beanName é o nome do bean pré-definido cuja propriedade queremos acessar.
jsp:include	inclui a resposta de tempo de execução de uma página JSP na página atual.
jsp:plugin	Gera uma construção específica do navegador do cliente que cria uma tag OBJECT ou EMBED para os Java Applets
jsp:fallback	Fornece texto alternativo se o plug-in java não estiver disponível no cliente. Você pode imprimir uma mensagem usando isso, se o plugin jsp incluído não estiver carregado.
jsp:element	Define elementos XML dinamicamente
jsp:attribute	define o atributo do elemento XML definido dinamicamente
jsp:body	Usado em tags padrão ou personalizadas para fornecer o corpo da tag.
jsp:param	Adiciona parâmetros ao objeto de solicitação.
jsp:text	Usado para escrever texto de modelo em páginas e documentos JSP. Uso:<jsp:text>Template data</jsp:text>

JSP JavaBean

Um componente JavaBeans é uma classe Java com os seguintes recursos:

- Um construtor sem argumentos.
- Propriedades definidas com acessadores e mutadores (método getter e setter).
- A classe não deve definir nenhuma variável de instância pública.
- A classe deve implementar a interface java.io.Serializable.

Exemplo de um JavaBean

Vamos dar um exemplo simples de código Java para entender o que queremos dizer quando dizemos JavaBean:

```
import java.io.Serializable;

public class EstudanteBean implements Serializable
{
 private String nome;
 private int idade;

 // construtor
 public EstudanteBean()
 {
  this.nome = "";
  this.idade = 0;
 }
 // getters and setters
 public void setNome(String name)
 {
  this.nome = nome;
 }
 public String getNome()
 {
  return nome;
 }
 public int getIdade()
 {
  return idade;
 }
 public void setIdade(int idade)
 {
  this.idade = idade;
 }
}
```

Como você pode ver no código acima, um JavaBean nada mais é do que uma classe Java que implementa a interface Serializable.

Usando um JavaBean na página JSP

JavaBeans pode ser usado em qualquer página JSP usando a <jsp:useBean>tag, Por exemplo:

<jsp:useBean id="bean name" scope="fully qualified path of bean" typeSpec/>

Usando qualquer propriedade JavaBean na página JSP
JavaBeans pode ser usado em qualquer página JSP usando a <jsp:useBean>tag,
<jsp:setProperty>tag e <jsp:getProperty>tag , Por exemplo:

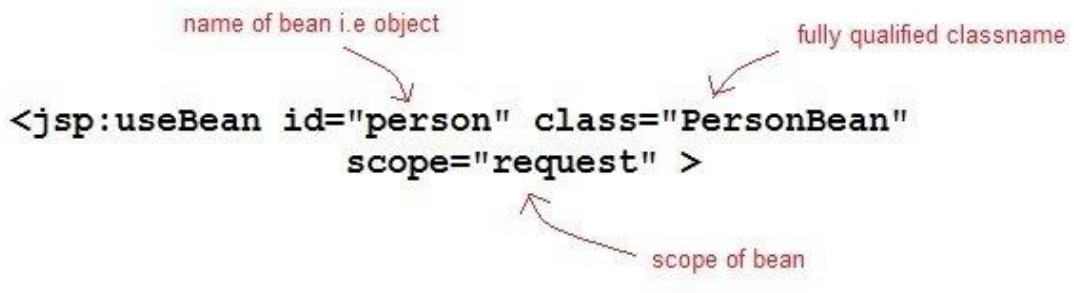
```
<jsp:useBean id="id" class="bean class name" scope="fully qualified path of bean">
  <jsp:setProperty name="beans id" property="property name" value="value"/>
  <jsp:getProperty name="beans id" property="property name"/>
  ..........
</jsp:useBean>
```

jsp:useBeanEtiqueta JSP

Se você deseja interagir com um componente JavaBeans usando a tag Action em uma página JSP, primeiro você deve declarar um bean. O <jsp:useBean>é uma maneira de declarar e inicializar o objeto bean real. Por bean queremos dizer objeto componente JavaBean. Sintaxe da tag <jsp:useBean>

<jsp:useBean id = "beanName" class = "className"
scope = "page | request | session | application">

Aqui o atributo id especifica o nome do bean. O atributo Scope especifica onde o bean está armazenado. O atributo class especifica o nome da classe totalmente qualificado.



Dada uma declaração useBean do seguinte:

<jsp:useBean id="myBean" class="PersonBean" scope="request" />

é equivalente ao seguinte código java:

```
PersonBean myBean = (PersonBean)request.getAttribute("myBean");
if(myBean == null)
{
  myBean = new PersonBean();
  request.setAttribute("myBean", myBean);
}
```

Se a tag jsp:useBean for usada com um corpo, o conteúdo do corpo só será executado se o bean for criado. Se o bean já existir no escopo nomeado, o corpo será ignorado.

Tempo para um exemplo

Neste exemplo veremos como <jsp:useBean>a tag padrão é usada para declarar e inicializar um objeto bean. Usaremos a classe PersonBean como componente JavaBean.

PersonBean.java

```
import java.io.Serializable;

public class PersonBean implements Serializable
{
 private String name;

  public PersonBean()
  {
   this.name="";
  }
  public void setName(String name)
  {
   this.name = name;
  }
  public String getName()
  {
   return name;
  }
}
```

Olá.jsp

```
<html>
   <head>
      <title>Bem-vindo a página</title>
   </head>
   <jsp:useBean id="person" class="PersonBean" scope="request" />
  <body>
      //Use the bean here...
  </body>
</html>
```

Aqui jsp:useBean declara um bean "person" na página jsp que pode ser usado lá. Como usá-lo, modificá-lo, estudaremos nas próximas lições.

JSP jsp:getProperty Tag

A tag getProperty é usada para recuperar uma propriedade de uma instância JavaBeans. A sintaxe da tag getProperty é a seguinte:

<jsp:getProperty name="beanName" property="propertyName" />

O atributo name representa o nome da instância JavaBean. O atributo property representa a propriedade do JavaBean cujo valor queremos obter.

Exemplo de getProperty com Java Bean

Segue nossa classe Java.

PersonBean.java

```
import java.io.Serializable;

public class PersonBean implements Serializable
{
 private String name;

 public PersonBean()
 {
  this.name="";
 }
 public void setName(String name)
 {
  this.name = name;
 }
 public String getName()
 {
  return name;
 }
}
```

hello.jsp

```
<html>
   <head>
      <title>Welcome Page</title>
   </head>
   <jsp:useBean id="person" class="PersonBean" scope="request" />
  <body>
      Name of Person is : <jsp:getProperty name="person" property="name" />
  </body>
</html>
```

Isso imprimirá o valor da propriedade. E se você precisar alterar o valor do imóvel. Vamos aprender como definir o valor da propriedade em nossa próxima lição.

jsp:setPropertyEtiqueta JSP

A setPropertytag é usada para armazenar dados em instâncias JavaBeans. A sintaxe da setPropertytag é:

```
<jsp:setProperty name="beanName" property="*">
<!-- or -->
<jsp:setProperty name="beanName" property="propertyName">
<!-- or -->
<jsp:setProperty name="beanName" property="propertyName" param="parameterName">
<!-- or -->
<jsp:setProperty name="beanName" property="propertyName" value="propertyValue">
```

O atributo name especifica o nome das instâncias javaBean. Isso deve corresponder ao atributo id especificado na jsp:useBeantag. O atributo property especifica qual propriedade do bean acessar.

Exemplo de setProperty com Java Bean
Segue nossa classe Java.

PersonBean.java

```
import java.io.Serializable;

public class PersonBean implements Serializable
{
  private String name;

  public PersonBean()
  {
    this.name="";
  }
  public void setName(String name)
  {
    this.name = name;
  }
  public String getName()
```

```
 {
   return name; }}
olá.jsp

<html>
   <head>
      <title>Página de Bem-vindo</title>
   </head>
   <jsp:useBean id="person" class="PersonBean" scope="request" />
   <jsp:setProperty name="person" property="name" value="Mirandiba" />
   <body>
      Nome da pessoa é : <jsp:getProperty name="person" property="name" />
   </body>
</html>
```

A saída será → O nome da pessoa é: Mirandiba

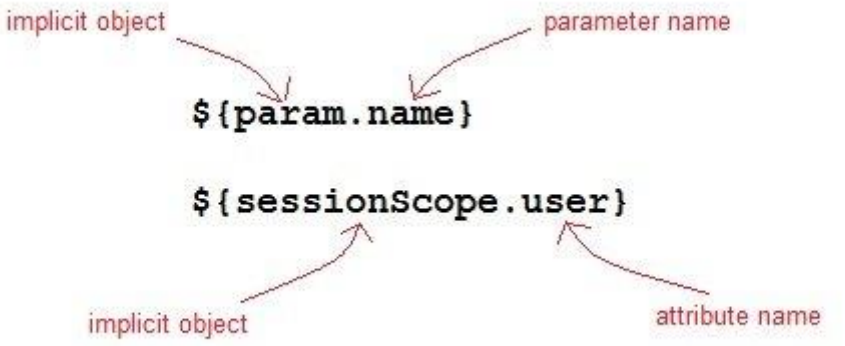
Da mesma forma, também podemos ter um Java Bean muito complexo, com muitas propriedades. Podemos facilmente obter e definir todas as propriedades usando o jsp:useBean, jsp:setProperty, jsp:getProperty.

Linguagem de Expressão JSP

Expression Language(EL) foi adicionado à especificação JSP 2.0. O propósito do EL é produzir páginas JSP sem script. A sintaxe de EL em um JSP é a seguinte:

${expr}

Aqui expr é uma expressão EL válida. Uma expressão pode ser misturada com texto/valores estáticos e também pode ser combinada com outras expressões para formar uma expressão maior.



Como a expressão EL é usada?

A expressão EL pode ser usada de duas maneiras em uma página JSP

1. Como valores de atributo em tags padrão e personalizadas. Exemplo:

<jsp:include page="${location}">

Onde a variável de localização é definida separadamente na página jsp.

Expressões também podem ser usadas jsp:setPropertypara definir um valor de propriedades, usando outras propriedades de bean como : Se tivermos um bean chamado Square com as propriedades comprimento, largura e área.

<jsp:setProperty name="square" property="area" value="${square.length*square.breadth}" />

2. Para saída na tag HTML:

<h1>Welcome ${name}</h1>

Para desativar a avaliação de expressões EL, especificamos o isELIgnoredatributo da diretiva page conforme abaixo:

<%@ page isELIgnored ="true|false" %>

Objetos implícitos JSP EL

A seguir estão os objetos implícitos em EL:

Objeto Implícito	Descrição
pageContext	Ele representa o objeto PageContext
pageScope	É usado para acessar o valor de qualquer variável definida no escopo da página
requestScope	Ele é usado para acessar o valor de qualquer variável definida no escopo da solicitação.
sessionScope	É usado para acessar o valor de qualquer variável definida no escopo da Sessão
applicationScope	É usado para acessar o valor de qualquer variável definida no escopo do aplicativo
param	Mapear um nome de parâmetro de solicitação para um único valor
paramValues	Mapeie um nome de parâmetro de solicitação para um array correspondente de valores de string.
header	Mapa contendo nomes de cabeçalho e valores de string única.
headerValues	Mapa contendo nomes de cabeçalho para um array correspondente de valores de string.
cookie	Mapa contendo dos cookies em uma string única.

Exemplo de JSP EL

Vamos dar um exemplo simples para entender a linguagem de expressão JSP,

index.jsp

```
<form method="POST" action=" bem-vindo.jsp"/>
   Name <input type="text" name="user" />
   <input type="submit" value="Submit"/>
</form>
```

bem-vindo.jsp

```
<html>
   <head>
      <title>Página de boa vindas</title>
   </head>

         <body>
      <h1>Bem-vindo ${param.name}</h1>
         </body>
</html>
```

Operações aritméticas disponíveis em EL

A seguir estão os operadores aritméticos disponíveis em EL:

Operação aritmética	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/ and div
Restante	% and mod

Operadores Lógicos e Relacionais disponíveis em EL

A seguir estão os operadores lógicos e comparadores disponíveis em EL:

Operador lógico e relacional	Operador
É igual a	== and eq
Não é igual	!= and ne
Menor que	< and lt
Maior que	> and gt
Maior ou igual	>= and ge
Menor ou igual	<= and le
e	&& and and
ou	|| and or
não	! and not