Java Server Pages - JSP

As *JSP* consistem em páginas que integram simultaneamente comandos HTML e scripts Java que são interpretados e executados do lado do servidor. Estes scripts são incorporados no documento entre delimitadores específicos à semelhança do que se passa nas tecnologias similares, *ASP* e *PHP*.

Arquitectura e conceitos básicos

As *Java Server Pages* foram criadas numa lógica de simplicidade de utilização. O conceito que esteve por detrás do seu aparecimento foi o facto dos programadores escolherem sempre as servlets como forma preferencial de construção de páginas Web dinâmicas, enquanto que os Web Designers, habituados à construção de páginas Web estáticas em HTML, estariam muito mais vocacionados para extensões ao tipo de linguagens que habitualmente utilizam.

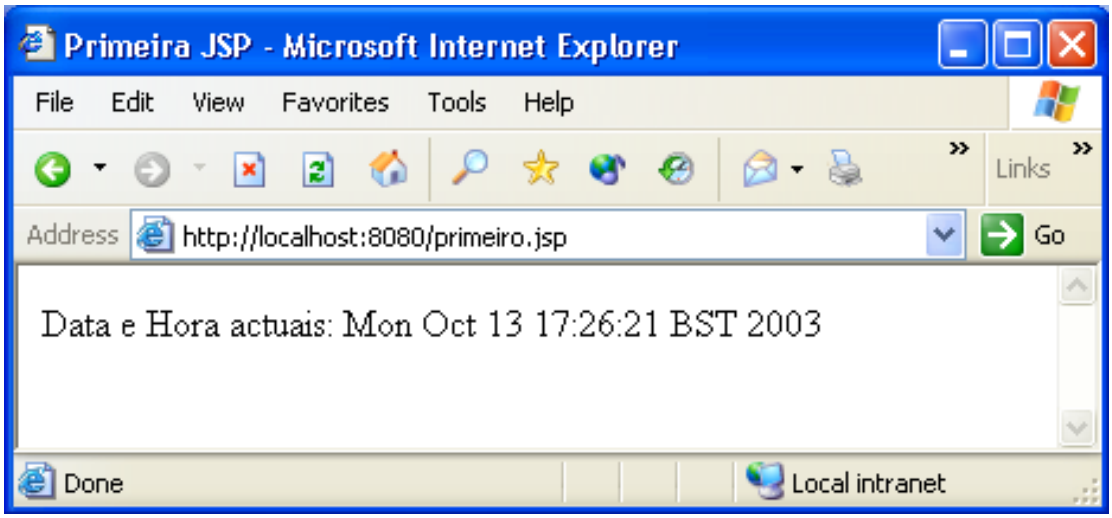
A geração de código HTML é muito mais simples com JSP do que com *servlets*, mas a adequação para a implementação da lógica de negócio também é diferente, com clara vantagem para os servlets.

As JSP são objectos Java com vocação para a geração dinâmica de HTML, pensadas para puderem ser facilmente geridas por ferramentas de edição vocacionadas para HTML e XML.

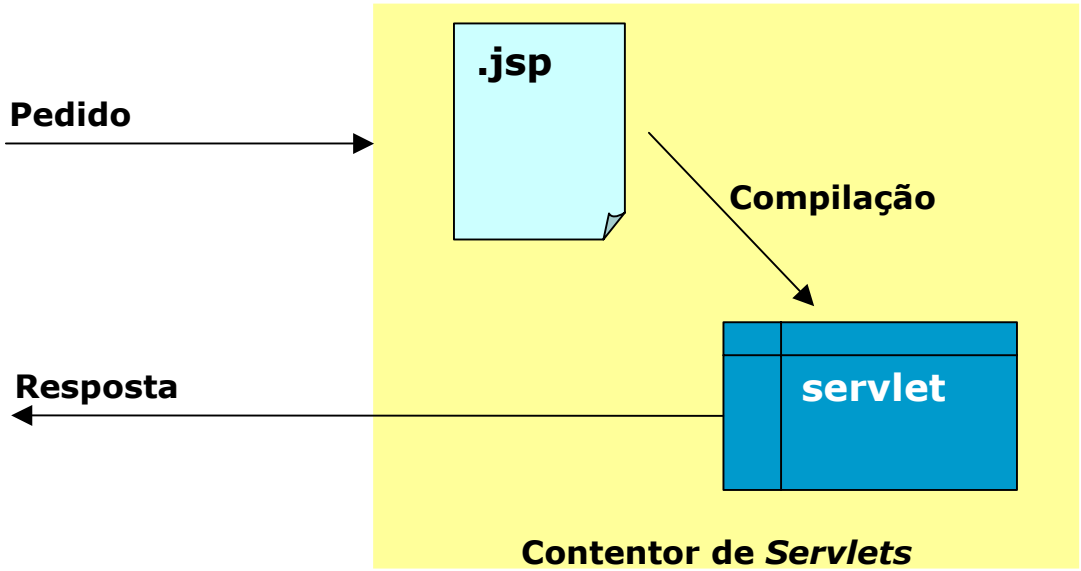
A grande vantagem que as JSP apresentam face a utilização pura e simples de *servlets* é a separação de conteúdo da apresentação. Utilizando servlets para gerar conteúdos, qualquer alteração implica sempre uma compilação.

Exemplo:

```
<%@ Page contentType="text/html"%>
<%@ Page import="java.util.Date" %>
<html>
<head><title>Primeira JSP</title></head>
<body>
        Data e Hora actuais: <%=(new Date()).toString() %>
</body>
</html>
```



A especificação JSP é na verdade uma extensão da API das *servlets* pelo que são tecnologias obviamente muito semelhantes. O ficheiro JSP corre numa entidade chamada ***JSP Engine*** no servidor, que se encarrega de compilar e transformar em *servlets* as JSP. Este processo de conversão é feito unicamente quando a JSP é invocada pela primeira vez.



A implementação de um JSP é feita estendendo a classe ***HttpJspBase*** que por sua vez implementa a interface Servlet. Os métodos essenciais envolvidos na execução de uma JSP são:

- **jspinit()** - executado quando a JSP é inicializada. Análoga ao método *init()* dos *servlets*;
- **jspDestroy()** - executado quando a JSP termina. Comparável ao *destroy()* dos *servlets*.
- **jspservice()** - este método irá conter o código principal da JSP, gerindo os pedidos e as respostas. Este método é análogo ao método *service()* dos *servlets*.

Sintaxe JSP

A sintaxe JSP divide-se nos seguintes tipos de elementos:

- Directivas
- Elementos de Scripting
- Acções

Directivas

As directivas são mensagens enviadas à *JSP engine* de modo a instrui-lo sobre o modo como deverá processar o código JSP. Normalmente não geram resultado visível.

As directivas definem-se com a seguinte sintaxe:

<%@ *... instrução ...* **%>**

e podem ser do tipo *page* e *include*.

A lista de atributos disponível para ser usada com as directivas *page* é a seguinte:

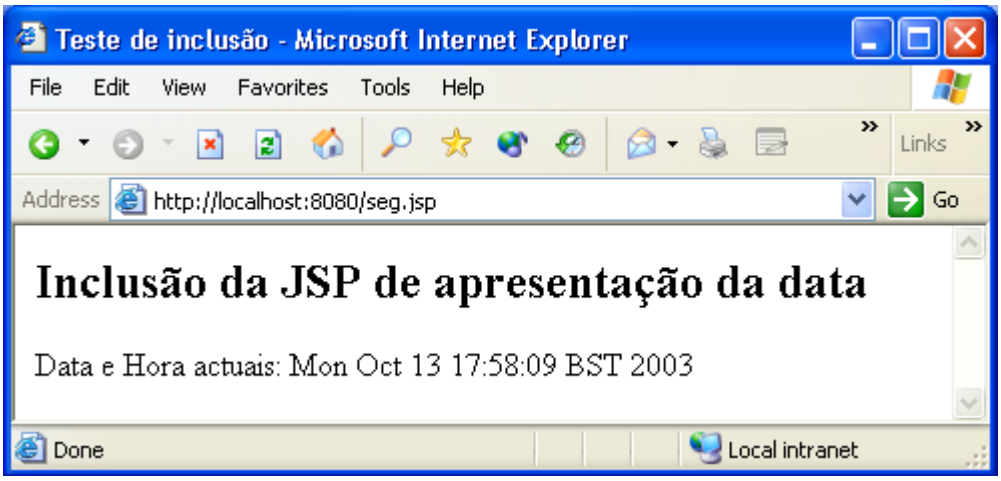
language	Linguagem de script a ser utilizada (por defeito Java)
extends	Classe que a JSP depois de compilada irá extender
import	Lista de *Java packages* a importar (separados por virgula)
session	Define se a página está envolvida numa sessão http (valor booleano)
buffer	Modelo de buffer para a saída para o cliente
autoFlush	Se o valor for true então o buffer automaticamente esvazia quando encher.
isThreadSafe	Define se é permitido a existência de múltiplas ligações de clientes à página.
info	Define informação (string) que pode ser recuperada pelo método *Servlet.getServletInfo()*.
errorPage	URL de outra JSP que será invocada em caso de excepção.
isErrorPage	Indica se a JSP corrente poderá ser definida como erroPage para outra JSP (valor booleano).
contentType	Tipo MIME da resposta

As directivas de inclusão permitem incluir um determinado conteúdo no lugar da directiva. A sintaxe é:

<%@ include file="nome_do_ficheiro" %>

Exemplo:

```
<html>
<head><title>Teste de inclusão</title></head>
<body>
<h2>Inclusão da JSP de apresentação da data</h2>
<%@ include file="primeiro.jsp" %>
</body>
</html>
```



Elementos de scripting

Esta categoria engloba as principais características de sintaxe das JSP. Os principais tipos são:

- **declarações** - permitem a definição de variáveis e métodos que são utilizáveis em toda a página. A sintaxe é:

  <%! *int k=0;* **%>**

  <%! *public void metodo()*
  {
  …
  }
  %>

- **expressões** - consiste num pedaço de código Java que é executado, convertido para *String* e devolvido ao cliente. As expressões colocam-se entre **<%=** e **%>** sem incluir ponto e virgula.

  <%=*k* **%>**

  <%=*metodo("Ambientes de Desenvolvimento")* **%>**

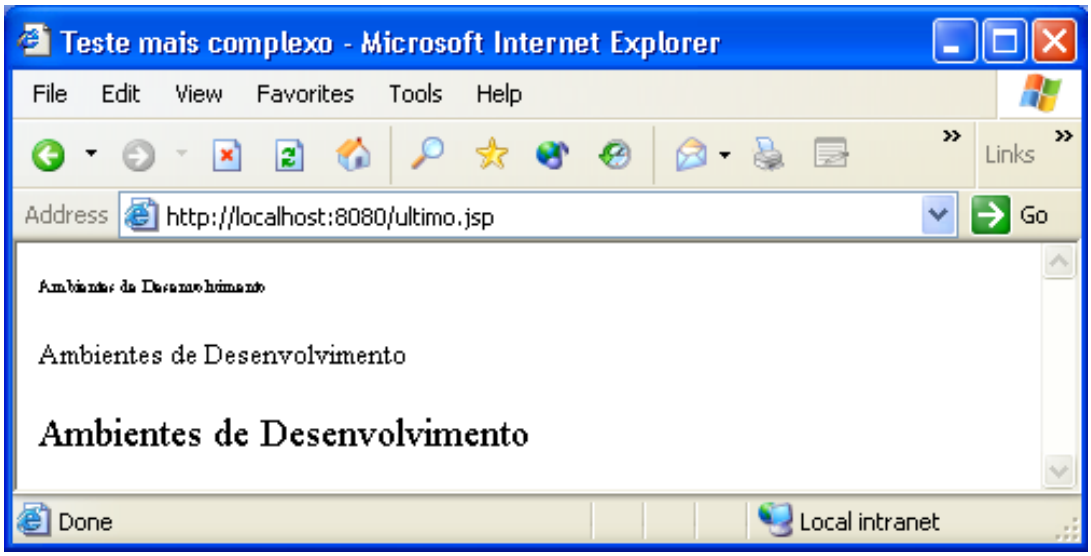
- **scriplets** - consistem em pedaços de código Java. As linhas de código colocam-se entre **<%** e **%>.**

  <% *for (int k=1; k<20; k++)* **%>**

- **comentários** - fazem-se entre **<!-** e **->** e permitem ao programador tecer considerações sobre o código.

Exemplo:

```
<%@ page contentType="text/html" %>
<%! int variavel=3;
      public   String escreve(int tamanho){
        return "<p style=\"font-size:"+tamanho+";\">Ambientes de Desenvolvimento";
      }
%>
<html>
<head><title>Teste mais complexo</title></head>
<body>
<%    for(int t=0; t<variavel; t++){   %>
                <%=escreve(8+5*t) %>
<%    }        %>
</body>
</html>
```



Acções

São comandos que afectam o comportamento do *JSP engine* e como tal influenciam a resposta a dar ao cliente.

Ao encontrar um comando de acção o *JSP engine* gera o código Java correspondente e executa-o.

As acções pré-definidas são:

<jsp:plugin>

Esta acção gera elementos <embed> ou <object> do HTML (dependendo do browser) que irão efectuar o download de plug-ins e a sua posterior execução.

Exemplo:

```
<jsp:plugin type="applet" code="Relogio.class" codebase="applet"
                                    jreversion="1.2" width="160" height="150">

        <jsp:params>
            <jsp:param name="corFundo" value="azul" />
        </jsp:params>

        <jsp:fallback>
            O seu browser não suporta a utilização de Plug-ins.
        </jsp:fallback>

</jsp:plugin>
```

<jsp:fallback>

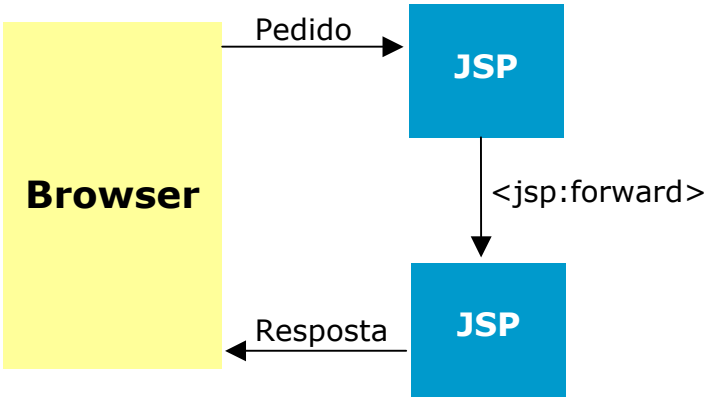
Esta acção só pode ser usada dentro da acção **<jsp:plugin>** e especifica qual a mensagem que será apresentada nos browsers que não suportem os comandos HTML <*embed*> e <*object*>

Exemplo:

```
<jsp:plugin type="applet" code="Relogio.class" codebase="applet"
                                    jreversion="1.2" width="160" height="150">
        <jsp:fallback>
            O seu browser não suporta a utilização de Plug-ins.
        </jsp:fallback>
</jsp:plugin>
```

<jsp:forward>

Permite redirecionar o pedido para outra JSP ou *servlet* dentro da aplicação Web. Quando esta acção for executada o buffer tem de estar vazio de qualquer conteúdo de resposta, caso contrário o redirecionamento falha com um erro de IllegalStateException.



Exemplo:

```
<jsp:forward page="URL_Destino" />
```

<jsp:useBean>

Permite associar uma referência (nome) a um *JavaBean* (componente Java que contem um conjunto privado de variáveis e que disponibiliza métodos públicos para modificar os seus valores) e por seu intermédio permite a sua utilização.

Exemplo:

```
<jsp:useBean id="relogio" class="java.util.Date" />
```

<jsp:setProperty>

Permite alterar o valor de propriedades de um *javaBean*.

Exemplo:

```
<jsp:setProperty name="relogio" property="modDate"
                 value="<%= new java.util.Date() %>" />
```

<jsp:getProperty>

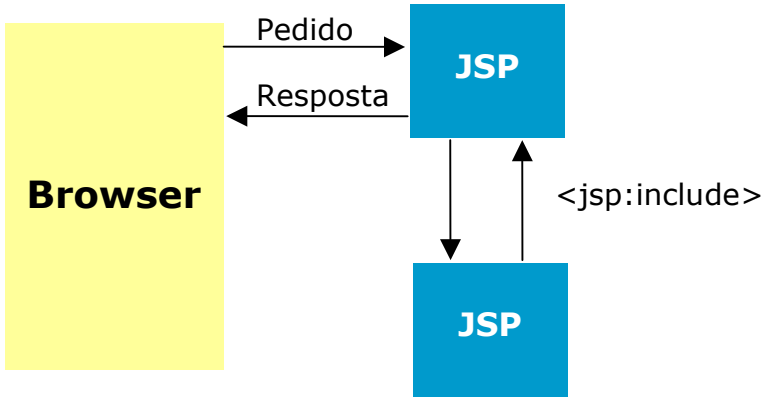
Retorna o valor de uma propriedade de um *javaBean* convertido em *String*.

Exemplo:

```
<jsp:getProperty name="relogio" property="hours" />
```

<jsp:include>

Permite incluir e executar um recurso externo (p.e. página JSP).



Exemplo:

```
<jsp:include page="relogio" property="URL_recurso_externo" flush="true" />
```

<jsp:param>

Pode ser utilizada no corpo de acção de <jsp:forward> ou <jsp:include> e permite parametrizar invocação do recurso acedido, ou pode ser utilizada no corpo de açcão de <jsp:params> para fornecer parâmetros a uma applet.

Exemplo:

```
<jsp:include page="navegacao.jsp" >
        <jsp:param name="corFundo" value="azul" />
</jsp:include>
```

<jsp:params>

Só pode ser utilizada no corpo de acção de <jsp:plugin> para delimitar o conjunto de acções <jsp:param> que especificam os parâmetros da applet.

Exemplo:

```
<jsp:plugin type="applet" code="Relogio.class" codebase="applet"
                                        jreversion="1.2" width="160" height="150">
                <jsp:params>
                        <jsp:param name="corFundo" value="azul" />
                </jsp:params>
</jsp:plugin>
```

Exemplo:

Ficheiro Colorir.html

```
<html>
	<body>
	<form method=”Post” action=http://localhost:8080/Cores.jsp>
		Escolha um cor:<br>
		<input type=”checkbox” name=”cores” value=”verde”>Verde<br>
		<input type=”checkbox” name=”cores” value=”amarelo”>Amarelo<br>
		<input type=”checkbox” name=”cores” value=”vermelho”>Vermelho<br>
		<input type=”submit” value=”Enviar”>
	</form>
	</body>
</html>
```

Ficheiro ObtemCores.java (colocar compilado em em WEB-INF/classes/cores/ObtemCores.class)

```
package cores;
import java.beans.*;

public class ObtemCores{
	String b[]=new String[] {“1”,”2”,”3”,”4”};

	public String[] getCores(){
		return b;
	}

	public void setCores(String [] b){
		this.b=b;
	}
}
```

Ficheiro Cores.jsp

```
<%@page contentType=”text/html”%>
<html>
<body>
<%! String[] strcores; %>
<jsp:useBean id=”xpto” scope=”page” class=”cores.ObtemCores”/>
<jsp:setProperty name=”xpto” property=”cores” param=”cores” />
As cores escolhidas foram:<br>
<%	strcores=xpto.getCores();
	if(!strcores[0].equals(“1”)){
		for(int k=0; k<strcores.length; k++){
%> - <%
			out.println(strcores[k]);
		}
	}
	else
		out.println(“Nenhuma cor foi selecconada”); %>
</body>
</html>
```

Objectos implícitos

O *JSP engine* disponibiliza um conjunto de objectos que podem auxiliar a realização de diversas tarefas. São eles:

Request	Representa *HttpServletRequest*, o pedido http.
Response	Representa *HttpServletResponse*, a resposta http
PageContext	Disponibiliza acesso aos atributos da página
Application	Representa um objecto *ServletContext*
Out	Objecto *JspWriter*, de escrita
Config	*ServletConfig*, parâmetros de configuração.
Page	Simula o "*this*"
Session	Representa *HttpSession*
Exception	Objecto de erro

Objecto *request*

Este objecto permite aceder á definição do pedido enviado pelo cliente.

Método	Descrição
getAttribute(*String****)***	Retorna o valor do parâmetro especificado ou NULL no caso de não existir. Retorna *java.lang.Object*
getAttributeNames()	Retorna uma lista de objectos contendo os nomes de todos os atributos disponíveis na especificação do pedido enviados pelo cliente. Retorna *java.util.Enumeration*
getAuthType()	Retorna o nome do método de autenticação utilizado ou NULL no caso de não haver protecção. Retorna *java.lang.String*
getCharacterEncoding()	Devolve o nome do método de codificação utilizado pelo pedido, ou NULL no caso de este não estar codificado. Retorna *java.lang.String*
getContentLength()	Tamanho da especificação do pedido em bytes. Retorna um *int*.
getContentType()	Obtém o MIME do pedido. Retorna *java.lang.String*
getContextPath()	Devolve a parte da URI do pedido que especifica o contexto do pedido. Retorna *java.lang.String*
getCookies()	Devolve um vector de objectos Cookie enviados pelo cliente. Retorna *Cookie[]*
getDateHeader(*String****)***	Devolve um *long* representando a data inserida no cabeçalho do pedido ou –1 caso este último não inclua cabeçalho
getHeader(*String****)***	Devolve o valor do parâmetro do cabeçalho especificado pelo nome ou NULL caso o pedido não inclua o cabeçalho. Retorna *java.lang.String*
getHeaderNames()	Retorna uma lista de objectos String contendo os nomes de todos os parâmetros de cabeçalho especificados no pedido. Retorna *java.util.Enumeration*
getHeaders(*String****)***	Retorna uma lista de objectos String contendo os valores de todos os parâmetros enviados pelo cliente. Retorna *java.util.Enumeration*
getInputStream()	Acede ao pedido como sendo informação binária por meio de um objecto *ServeltInputStream*.
getIntHeader(*String****)***	Devolve o valor do parâmetro de cabeçalho especificado, como um *int*.
getMethod()	Devolve o nome do método http utilizado no pedido. Retorna *java.lang.String*
getParameter()	Devolve o valor de um dado parâmetro como *String* ou NULL no caso de este não existir.
getParameterNames()	Retorna uma lista de objectos String contendo os nomes de todos os parâmetros enviados pelo cliente. Retorna *java.util.Enumeration*
getParameterValues()	Retorna um array de objectos String contendo os valores de todos os parâmetros enviados pelo cliente. Retorna *java.util.Enumeration*

Método	Descrição
getPathInfo()	Obtém informação de caminho adicional passada na URI, ou NULL no caso desta não existir. Para uma JSP este método devolve sempre NULL. Retorna *java.lang.String*
getPathTranslanted()	Traduz o caminho fornecido pelo método *getPathInfo()* para o caminho real nos discos do servidor Web. Retorna *java.lang.String*
getProtocol()	Obtém o nome e a versão do protocolo utilizado pelo cliente para aceder ao servidor. P.e. HTTP/1.1. Retorna *java.lang.String*
getQueryString()	Obtém toda a *QueryString* que foi passada na URI. Retorna *java.lang.String*
getReader()	Coloca num objecto *BufferedReader* a informação especificada no pedido. Retorna *java.lang.BufferedReader*
geeRemoteAddr()	Devolve o IP do cliente. Retorna *java.lang.String*
geeRemoteHost ()	Devolve o *hostname* do cliente, ou caso não seja possível, devolve o respectivo IP. Retorna *java.lang.String*
geeRemoteUser()	Devolve o *username* do utilizador no cliente se este estiver autenticado ou NULL em caso contrário. Retorna *java.lang.String*
getRequestDispatcher(*String****)***	Devolve um objecto *RequestDispatcher* que actua como um envelope para o recurso localizado no caminho especificado.
getRequestedSessionId()	Devolve o Identificador único da sessão em que o cliente está envolvido. Retorna *java.lang.String*
getRequestURI()	Devolve a parte da URI que se inicia no protocolo e termina no inicio da QueryString. Retorna *java.lang.String*
getScheme()	Devovle o protocolo (schema) utilizado para a conexção ao servidor (p.e :http, https, ftp). Retorna *java.lang.String*
getServerName()	Devolve o *hostname* do servidor web. Retorna *java.lang.String*
getServerPort()	Devolve o numero do porto onde o pedido foi recebido. Retorna *int*
getServletPath()	Devolve a parte da URI que invoca a servlet. No caso de JSP corresponde a todo caminho relativo. Retorna *java.lang.String*
getSession() ***getSession(****Boolean****)***	Devolve o objecto *HttpSession* associado com o pedido.
GetUserPrincipal	Devolve um objecto *Principal* contendo o nome co utilizador autenticado. Retorna *java.securityPrincipal*
isRequestedSessionIdFromCookie()	Verifica se o valor do *sessionID* se encontra nalguma cookie. Método lógico.
isRequestedSessionIdFromURL()	Verifica se o valor do *sessionID* vem na URL. Método lógico.
isRequestedSessionIdValid()	Verifica se a sessão identificado pelo *sessionID* é ainda válida. Método lógico.

Método	Descrição
isSecure()	Devolve *true* quando o pedido tiver sido feito utilzando um cana seguro (p.e. https). Método lógico
isUserInRole(*String****)***	Verifica se o utilizador autenticado se encontra inscrito na *role* especificada. Método lógico.
RemoveAttribute(*String****)***	Remove o atributo especificado do pedido. Método *void*.
setAttribute(*String, Object****)***	Armazena o atributo referenciado (com o respective valor) na especificação do pedido.

Objecto *response*

Este objecto permite enviar informação para o cliente.

Método	Descrição
addCookie(*Cookie****)***	Coloca uma cookie no cliente. Método *void*
addDateHeader(*String, long****)***	Acrescenta um cabeçalho de nome e data especificados à resposta. O valor da data é fornecido em milisegundos referentes ao tempo decorrido desde 1.Jan.1970. Método *void*
addHeader(*String, String****)***	Acrescenta um cabeçalho de nome e valor especificados à resposta. Método *void*
AddIntHeader(*String, int****)***	Acrescenta um cabeçalho de nome e valor inteiro especificados à resposta. Método *void*
containsHeader(*String****)***	Retorna um valor lógico indicando se o cabeçalho especificado foi ou não já enviado para o cliente. Método lógico.
encodeRedirectURL(*String****)***	Codifica a URL especificada, incluindo nela o *sessionID* da sessão, para que possa se utilizada no método *sendRedirect()*. Retorna *java.lang.String*
encodeURL(*String****)***	Codifica a URL especificada, incluindo nela o *sessionID* da sessão, para que possa se utilizada num hyperlink (p.e. <a>). Retorna *java.lang.String*
flushBuffer()	Envia o conteudo do buffer para o cliente. Método *void*.
getBufferSize()	Devolve o tamanho em bytes do buffer. Retorna um *int*
getCaracterEncoding()	Obtém a página de teclado utilizada para a resposta. Retorna *java.lang.String*
getOutputStream()	Devolve um objecto *ServletOutputStream* apropriado para o envio de informação binária ao cliente. Apropriado para Servlets.
getWriter	Devolve um objecto *PrinterWriter* que permite enviar informação de texto para o cliente.
isCommitted()	Devolve um valor lógico indicando se a resposta foi entregue ao cliente.
reset()	Limpa o buffer e os parâmetros de cabeçalho da resposta. Método *void*
sendError(*int****)*** ***sendError(****int, String****)***	Envia um código de erro ao cliente. Método *void*
sendRedirect(*String****)***	Redirecciona para a URL especificada. Método *void*
setBufferSize(*int****)***	Especifica o tamanho do buffer utilizado pela resposta. Método *void*
setContentLength(*int****)***	Define o tamanho em bytes da resposta. Apropriado para Servlets. Método *void*
setContentType(*String****)***	Define o MIME da informação enviada para o cliente. Método *void*

Objecto *application*

Este objecto permite partilhar informação entre diversos utilizadores da aplicação web.

Método	Descrição
getAttribute(*String****)***	Retorna o objecto armazenado na aplicação com o nome especificado ou Null no caso de não existir. Retorna *java.lang.Object*
getAttributeNames()	Retorna uma lista de objectos String contendo os nomes de todos os objectos armazenados na sessão. Retorna *java.util.Enumeration*
getContext(*String URI****)***	Retorna um objecto *ServletContext* correspondente ao URI especificado.
getInitParameter(*String****)***	Devolve o valor do parâmetro de inicialização especificado, ou NULL no caso de este não existir. Retorna *java.lang.String*
getInitParameterNames()	Retorna uma lista de objectos String contendo os nomes de todos os parâmetros de inicialização da aplicação. Retorna *java.util.Enumeration*
getMajorVersion()	Obtém a versão principal da *API Java Servlet* suportada pelo servidor Web. No caso de se tratar da versão 2.3, obter-se-ia 2. Retorna um inteiro.
getMinorVersion()	Obtém a sub-versão da *API Java Servlet* suportada pelo servidor Web. No caso de se tratar da versão 2.3, obter-se-ia 3. Retorna um inteiro.
getMimeType(*String****)***	Devolve o MIME do ficheiro especificado ou NULL caso este não seja conhecido. Retorna *java.lang.String*
getNamedDispatcher()	Devolve um objecto *RequestDispatcher* que actua como um envelope para o nome da aplicação web.
getRequestDispatcher(*String****)***	Devolve um objecto *RequestDispatcher* que actua como um envelope para o recurso especificado pelo relativo fornecido.
getRealPath(*String****)***	Devolve o caminho real do ficheiro especificado por um dado caminho lógico. Retorna *java.lang.String*
getResource(*String****)***	Devolve a URL do recurso especificado por caminho lógico. Retorna *java.net.URL*
getResourceAsStream(*String****)***	Obtém o recurso especificado como um objecto *java.io.InputStream*
getServerInfo()	Devolve o nome e a versão do servidor Web. Retorna *java.lang.String*
log(*String****)***	Escreve a mensagem especificada no ficheiro de *log* do servidor Web. Método *void*.
removeAttribute(*String****)***	Remove da aplicação o objecto com o nome especificado Método *void*.
setAttribute(*String, Object****)***	Armazena um objecto na aplicação com o nome especificado. Método *void*.

Objecto *session*

Este objecto permite armazenar informação que será utilizada pela JSP em futuros pedidos do mesmo cliente.

Método	Descrição
getAttribute(*String****)***	Retorna o objecto armazenado na sessão com o nome especificado ou Null no caso de não existir. Retorna *java.lang.Object*
getAttributeNames()	Retorna uma lista de objectos String contendo os nomes de todos os objectos armazenados na sessão. Retorna *java.util.Enumeration*
getCreationTime()	Retorna a hora de criação da sessão medida em milisegundos desde 1.Jan.1970. Retorna um *long*
getId()	Obtém um identificador único para a sessão. Retorna *java.lang.String*
getLastAccessTime()	Devolve a hora do último acesso á sessão medida em milisegundos desde 1.Jan.1970. Retorna um *long*
getMaxInactiveInterval()	Devolve o intervalo máximo em segundos em que a sessão permanecerá activa sem que haja acessos. Retorna um *int*
getSessionContext()	Método descontinuado desde a versão 2.1 e não substituído.
getValue(*String****)***	Método descontinuado. Na versão 2.2 foi substituído por *getAttribute(String)*
getValueNames()	Método descontinuado. Na versão 2.2 foi substituído por *getAttributeNames()*
invalidate()	Termina a sessão libertando os valores nela contidos. Método *void*.
isNew()	Devolve *true* se o cliente ainda não sabe da existência da sessão ou se optou por não a utilizar. Método lógico
putValue(*String, Object****)***	Método descontinuado. Na versão 2.2 foi substituído por *setAttribute(String,Object)*
removeAttribute(*String****)***	Remove da sessão o objecto com o nome especificado. Método *void*.
removeValue(*String****)***	Método descontinuado. Na versão 2.2 foi substituído por *removeAttribute(String)*
setAttribute(*String,Object****)***	Armazena um objecto na sessão com o nome especificado. Método *void*.
setMaxInactiveInterval(*int****)***	Especifica o intervalo máximo em segundos em que a sessão permanecerá activa sem que haja acessos. Método *void*.

Exemplo:

teste_session1.jsp

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

<!doctype html public "-//w3c//dtd html 4.0 transitional//en"
                        "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>
<body>
<h2>Teste a uma sessão em JSP</h2>
 <%
String tst = "--XX--XX--XX--";
Date d  = new Date();
int num = new Random().nextInt(5) + 1;
%>
<p>
vai guardar numa variavel de sessão a data actual (<em><%= d %></em>),
a string "<em><%= tst %></em>" e o inteiro (<em><%= num %></em>
<%
session.setAttribute("data1", d);
session.setAttribute("stringTeste", tst);
// guardar na sessão o inteiro obriga a transforma-lo num objecto Integer
Integer in = new Integer(num);
session.setAttribute("numInt", in);
%>
<p>para experimentar, siga este <a href="teste_session2.jsp">link</a></p>
</body>
</html>
```

teste_session2.jsp

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

<!doctype html public "-//w3c//dtd html 4.0 transitional//en"
                        "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>
<body>
<h2>Teste a uma sessão em JSP (passo2)</h2>
 <p>vai <strong>obter</strong> duma variável de sessão a data, a string e o inteiro
que foram guardados na sessão
<%
Date d = (Date)session.getAttribute("data1");
String tst = (String)session.getAttribute("stringTeste");
// obter numero inteiro
Integer in = (Integer)session.getAttribute("numInt");
%>

<p>Data: <%= d %><p>String: <%= tst %><p>Inteiro: <%= in %>

</body>
</html>
```

Objecto *out*

Método	Descrição
clear() ***clearBuffer()***	Limpa o buffer. Método *void*.
close()	Fecha o canal estabelecido para comunicação com o cliente. Método *void*.
flush()	Envia imediatamente o conteúdo do buffer para o cliente. Método *void*.
getBufferSize()	Obtém o tamanho do buffer em bytes. Retorna um *int*.
getRemaining()	Devolve o espaço livre no buffer. Retorna um int.
getMaxInactiveInterval()	Devolve o intervalo máximo em segundos em que a sessão permanecerá activa sem que haja acessos. Retorna um *int*
isAutoFlush()	Devolve true se o objecto JspWriter está configurado para efectuar esvaziamentos automáticos do buffer. Método lógico.