OPEN
Núcleo de Pesquisas e Desenvolvimento de Tecnologia Brasileiro

Projeto:

Tecnologia Empresarial

Material de Estudo:

Java Server Pages (JSP)

Desenvolvido em parceira com:

portaljava

Coordenação: Paulo César Machado Jeveaux
Contato: paulo_jeveaux@open.org.br
jeveaux@portaljava.com

Sumário

1. Introdução

1.1 O que é JSP?

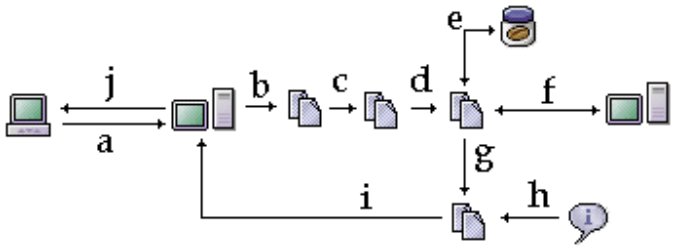
JSP significa "*Java Server Pages*". Esta tecnologia é usada para servir conteúdo dinâmico para o usuário, usando lógica e dados no lado do servidor. JSP faz parte da plataforma J2EE (*Java 2 Enterprise Edition*) e juntamente com os Java Servlets e Java Beans pode ser usada para desenvolver aplicações web eficientes, escaláveis e seguras rapidamente.

A Java Server Pages (JSP) é uma tecnologia para o desenvolvimento de aplicações WEB, criando páginas chamadas dinâmicas semelhante ao Active Server Pages (ASP), porém tem a vantagem da portabilidade de plataforma podendo ser executado em outros Sistemas Operacionais além dos da Microsoft e da compilação das páginas permitindo que elas rodem muito mais velozmente. Permite também que o desenvolvedor de sites produzir aplicações que permitam acesso nativo aos mais diversos bancos de dados graças a tecnologia JDBC, acesso a arquivos texto, a captação de informações a partir de formulários, a captação de informações sobre o visitante e sobre o servidor entre diversas outras coisas.

Para aqueles que conhecem a tecnologia Java Servlets verá que o JSP não oferece nada que não se possa conseguir com as Servlets puras. O JSP, entretanto, oferece a vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção da página dinâmica. Além disso, essa tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, onde programador e o designer podem trabalhar em um mesmo projeto, mas de forma independente. Outra característica da JSP é produzir conteúdos dinâmicos que podem ser reutilizados.

1.2 Como funciona?

Inicialmente o cliente faz uma requisição de uma página JSP através de um Browser, esta página então será processada pelo servidor, se for à primeira vez, esta página JSP é transformada em um programa Java (conhecido como Servlet) este é compilado e gerado um bytecode (conhecido por .class) e a partir deste é gerada uma página HTML que é enviada de volta ao browser do cliente, a partir da segunda vez que esta mesma página for acessada é verificado apenas se ocorreu ou não quaisquer mudanças, se não ocorreram o bytecode é chamado para gerar a página HTML. A figura abaixo ilustra esse funcionamento:



Funcionamento de uma JSP

a. Todo o processo começa quando um cliente faz uma solicitação de uma página JSP, neste momento, é enviado um objeto do tipo request para o WebServer.
b. O WebServer localiza e envia a ação para a página JSP.
c. E verificado mudanças nesta, se for à primeira vez é criada um programa Java Servlet.
d. O programa Java Servlet é compilado e transformado em um ByteCode (.class)
e. Este .class pode ser auxiliados por pacotes .JAR que carregam JavaBeans
f. Durante a montagem da página HTML podem ocorrer também solicitações a Bancos de Dados.
g. A página HTML é gerada
h. Chamadas a TAG´s Personalizadas são transformadas neste momento em TAG´s comuns para a geração final.
i. A página final padrão HTML é enviada de volta ao servidor.
j. Que a devolve para o cliente que fez sua solicitação.

Vale lembrar que existe uma pequena lentidão no primeiro acesso a página JSP por ser a compilação do JSP em Servlets (*bytecodes .class*). Após este processo as páginas JSP rodam mais velozes.

A JSP usa a linguagem **Java** como base para sua linguagem de *Scripts*, utilizando todo o potencial desta e é exatamente por esse motivo que a JSP se apresenta muito mais flexível e muito mais robusta do que outras plataformas.

1.3 Por que usar JSP se já existe ASP, PHP, etc?

Existem várias linguagens usadas para criar aplicações web. Entre elas ASP, PHP, ColdFusion e Perl. Por que usar JSP então?

- JSP usa Java.

Java é uma das linguagens mais populares atualmente e é interpretada, portanto o código escrito em uma arquitetura pode ser portado para qualquer outra.

- JSP é parte do pacote J2EE

J2EE é um dos modelos mais usados para construir aplicações de grande porte, e é suportado por várias gigantes da computação como IBM, Oracle, Sun, etc.

- Programação em rede é inerente a Java

O suporte inerente de Java para a área de redes faz dela uma ótima linguagem para a Internet.

- JSP x ASP

Uma das diferenças que pode ser fundamental para a escolha entre estas duas tecnologias é que ASP é da Microsoft e só roda em ambiente Windows, e também todo software necessário são pagos. JSP, feito pela Sun, roda em qualquer plataforma que tenha

a máquina virtual de Java, e tem vários softwares gratuitos para disponibilizar a aplicação. (Tomcat, por exemplo).

1.4 Executando

Antes de tudo é preciso instalar e configurar o container Tomcat. Consulte pelos materiais do OPEN (em www.open.org.br) pela apostila de **"Instalação e Configuração do Tomcat"**.

2. Java Beans

2.1 O que faz de um Bean um Bean?

Um Bean é simplesmente uma classe de Java que segue um conjunto de convenções simples de design e nomeação delineado pela especificação de JavaBeans. Os Beans não precisam estender uma determinada classe ou implementar uma determinada interface.

2.2 Convenções de um Bean

As convenções de JavaBean são o que nos permitem desenvolver Beans, porque elas permitem que o container Bean analise um arquivo de classe Java e interprete seus métodos como propriedades, designando a classe como um Bean de Java.

- Beans são simplesmente objetos Java. Mas como seguem um padrão fica mais fácil trabalhar com eles.
- Uma boa prática é colocar Bean no nome de uma classe que é um Bean, para que seja mais bem identificado. Assim uma classe que representa uma pessoa ficaria PessoaBean ou BeanPessoa.

O construtor Bean

A primeira regra da criação do Bean JSP é que você tem que implementar um construtor que não tenha argumentos. Este construtor é usado, por exemplo, para instanciar um Bean através da tag <jsp:useBean> visto mais adiante. Se a classe não especificar um construtor sem argumentos, então um construtor sem argumentos e sem código será assumido.

```
public Bean() { }
```

Propriedades de um Bean

Coloque os atributos como privados, e faça métodos get e set para acessa-los, e estes métodos então serão públicos.

```
private String nome;
public String getNome() { return nome;}
public void setNome(String novo) { nome = novo; }
```

Uma boa convenção de nome de propriedades é começar com letra minúscula e colocar em maiúscula a primeira letra de cada palavra subseqüente. Assim como nos métodos de ajuste, a palavra set ou get começa em minúscula e a primeira letra da propriedade será maiúscula.

```
private String corCarro;
public String getCorCarro();
```

Propriedades indexadas

Caso a propriedade seja um conjunto de valores (array), é uma boa prática criar métodos para acessar o conjunto inteiro de valores e para acessar uma posição específica.

```
private String[] telefone;
public String[] getTelefone() { return telefone; }
public String getTelefone(int index){return telefone[index];}
```

Propriedades booleanas

Para propriedades booleanas, você pode substituir a palavra get por is.

```
private boolean enabled;
public boolean isEnabled() { return enabled; }
```

2.3 JavaBeans versus EJB (Enterprise JavaBeans)

Os JavaBeans assim como o EJB são modelos de componente, porém apesar dos dois compartilharem o termo **JavaBeans** os modelos não são relacionados. Uma parte da literatura tem referenciado EJB como uma extensão dos JavaBeans, mas isto é uma interpretação errada. As duas APIs servem para propósitos bem diferentes.

O JavaBean tem a intenção de ser usado em propósitos do mesmo processo (intra-processo) enquanto EJB é projetado para ser usado como componentes inter-processos. Em outras palavras, os JavaBeans não têm a intenção de ser usado como componente distribuído assim como o EJB.

3. JSP Tags

Numa olhada rápida, JSP parece com HTML (ou XML), ambos contém texto encapsulado entre tags, que são definidas entre os símbolos < e >. Mas enquanto as tags HTML são processadas pelo navegador do cliente para mostrar a página, as tags de JSP são usadas pelo servidor web para gerar conteúdo dinâmico.

A seguir estão os tipos de tags válidos em JSP:

3.1 Ações JSP

Executam diversas funções e estendem a capacidade de JSP. Usam sintaxe parecida com XML, e são usadas (entre outras coisas) para manipular Java Beans. Existem seis tipos de ações:

<jsp:forward>

Este elemento transfere o objeto request contendo informação da requisição do cliente de ma página JSP para outro arquivo. Este pode ser um arquivo HTML, outro arquivo JSP ou um servlet, desde que faça parte da mesma aplicação.

Sintaxe:

```
<jsp:forward page="(URL relativa | <%= expressão %>)" />
ou
<jsp:forward page="(URL relativa | <%= expressão %>)" >
<jsp:param name="nome do parâmetro"
value="(valor do parâmetro| <%= expressão %>)" />
</jsp:forward>
```

<jsp:getProperty>

Este elemento captura o valor da propriedade de um bean usando o método get da propriedade e mostra o valor na página JSP. É necessário criar ou localizar o bean com <jsp:useBean> antes de usar <jsp:getProperty>.

Sintaxe:

```
<jsp:getProperty name="nome do objeto (bean)"
property="nome da propriedade" />
```

<jsp:include>

Este elemento permite incluir um arquivo estático ou dinâmico numa página JSP. Os resultados de incluir um ou outro são diferentes. Se o arquivo é estático, seu conteúdo é incluído quando a página é compilada num servlet. Se for dinâmico, funciona como uma requisição para o arquivo e manda o resultado de volta para a página. Quando estiver terminada a ação do include continua-se processando o restante da página.

Sintaxe:

```
<jsp:include page="{URL relativa | <%= expressão %>}"
flush="true" />
```

```
ou

<jsp:include page="{URL relativa | <%= expressão %>}"
flush="true" >
<jsp:param name="nome do parâmetro"
value="{nome do parâmetro | <%= expressão %>}" />
</jsp:include>
```

<jsp:plugin>

Executa ou mostra um objeto (tipicamente um applet ou um bean) no navegador do cliente, usando o plug-in Java que está embutido no navegador ou instalado na máquina. Não será explicado o funcionamento deste elemento nesta apostila.

<jsp:useBean>

Localiza ou instancia um componente. Primeiro tenta localizar uma instância do bean. Se não existe, instancia ele a partir da classe especificada. Para localizar ou instanciar o bean, são seguidos os seguintes passos, nesta ordem:

1. Tenta localizar o bean com o escopo e o nome especificados.
2. Define uma variável de referência ao objeto com o nome especificado.
3. Se o bean for encontrado, armazena uma referência ao objeto na variável. Se foi especificado o tipo, converte o bean para este tipo.
4. Se não encontrar, instancia-o pela classe especificada, armazenando uma referência ao objeto na nova variável.
5. Se o bean tiver sido instanciado (ao invés de localizado), e ele tem tags de corpo (entre <jsp:useBean> e </jsp:useBean>), executa estas tags.

Sintaxe:

```
<jsp:useBean
id="nome da instancia"
scope="page|request|session|application"
{ class="package.class" |
type="package.class" |
class="package.class" type="package.class" |
beanName="{package.class | <%= expressão %>}"
type="package.class"
}
{ /> |
> outros elementos (tags de corpo)
</jsp:useBean>
}
```

<jsp:setProperty>

O elemento <jsp:setProperty> ajusta o valor de uma ou mais propriedades em um bean, usando os métodos de ajuste (set) dele. É necessário declarar o bean com <jsp:useBean> antes de ajustar uma propriedade. Estas duas ações trabalham juntas, portanto o nome de instância usada nas duas deve ser igual.

Sintaxe:

```
<jsp:setProperty
name="nome de instância do bean"
{ property="*" |
property="nome da propriedade" [ param="nome do parâmetro" ] |
property="nome da propriedade" value="{string | <%= expressão %>}"
}
/>
```

3.2 Diretivas

São instruções processadas quando a página JSP é compilada em um servlet. Diretivas são usadas para ajustar instruções no nível da página, inserir dados de arquivos externos, e especificar tag libraries. Diretivas são definidas entre **<%@** e **%>**.

Existem três tipos de diretivas:

include

Inclui um arquivo estático em uma página JSP.

Sintaxe:

```
<%@ include file="relativeURL" %>
```

page

Define atributos que são aplicados a todo o arquivo JSP, e a todos os seus arquivos incluídos estaticamente.

Sintaxe:

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
[ import="{package.class | package.*}, ..." ]
[ session="true|false" ]
```

```
[ buffer="none|8kb|sizekb" ]
[ autoFlush="true|false" ]
[ isThreadSafe="true|false" ]
[ info="text" ]
[errorPage="URL relativa" ]
[ contentType="mimeType [ ;charset=characterSet ]" |
"text/html ; charset=ISO-8859-1" ]
[ isErrorPage="true|false" ]
%>
```

taglib

Define uma tag library e seu prefixo a ser usado na página JSP.

Sintaxe:

```
<%@ taglib uri="localização do descritor da tag"
prefix="prefixo da tag" %>
```

3.3 Declarações

São similares com as declarações de variáveis em Java, e definem variáveis para uso subseqüente em expressões ou scriptlets. São definidas entre **<%!** e **%>**.

Sintaxe:

```
<%! int x = 0; declaração; ... %>
```

3.4 Expressões

Contém um comando válido da linguagem Java que é avaliado, convertido para uma String, e inserido onde a expressão aparece no arquivo JSP. Não é usado ponto e vírgula para terminar a expressão, e só pode haver uma entre **<%=** e **%>**.

Sintaxe:

```
<%= pessoa.getNome() %>
```

3.5 Scriptlets

São blocos de código Java embutidos numa página JSP. O código do scriptlet é inserido literalmente no servlet gerado pela página. É definido entre **<%** e **%>**.

Sintaxe:

```
<% int x = 0;
```

```
x = 4 * 9;
String str = "PET";
...
%>
```

3.6 Comentários

São similares aos comentários HTML, mas são tirados da página quando o arquivo JSP é compilado em servlet. Isto significa que os comentários JSP não aparecem no código fonte da página visualizada pelo navegador do usuário. Comentários em HTML são feitos entre **<!–** e **–>**, enquanto comentários em JSP são entre **<%–** e **–%> ou <%/*** e ***/%>**.

4. Objetos Implícitos

Como uma característica conveniente, o container JSP deixa disponível objetos implícitos que podem ser usados nos scriptlets e expressões, sem que o autor tenha que criá-los. Estes objetos instanciam classes definidas na API (application program interface) de Servlets. São nove objetos:

Objeto	Classe ou Interface	Descrição
page	javax.servlet.jsp.HttpJspPage	Instância de servlet da Página
config	javax.servlet.ServletConfig	Dados de configuração do Servlet
request	javax.servlet.http.HttpServletRequest	Dados de solicitação incluindo parâmetros
response	javax.servlet.http.HttpServletResponse	Dados da resposta
out	javax.servlet.jsp.JspWriter	Fluxo de saída para o conteúdo da página
session	javax.servlet.http.HttpSession	Dados de sessão específicos de usuário
application	javax.servlet.ServletContext	Dados compartilhados por todas as páginas da aplicação
pageContext	javax.servlet.jsp.PageContext	Dados de contexto para execução da página
exception	javax.lang.Throwable	Erros não pegos ou exceção

Objeto page

O objeto page representa a própria página JSP ou, mais especificamente, uma instância da classe de servlet na qual a página foi traduzida.

Objeto config

O objeto config armazena dados de configuração de servlet — na forma de parâmetros de inicialização — para o servlet no qual uma página JSP é compilada. Pelo fato das páginas JSP raramente serem escritas para interagir com parâmetros de inicialização, este objeto implícito raramente é usado na prática.

Objeto request

O objeto request representa a solicitação que acionou o processamento da página atual. Para solicitações de HTTP, este objeto fornece acesso a todas as informações associadas com uma solicitação, incluindo sua fonte, a URL solicitada e quaisquer cabeçalhos, cookies ou parâmetros associados com a solicitação. Dentre os usos mais comuns para o objeto request, encontra-se a procura por valores de parâmetros e cookies.

Objeto response

O objeto response representa a resposta que será enviada de volta para o usuário como resultado do processamento da página JSP.

Objeto out

Este objeto implícito representa o fluxo de saída para a página, cujo conteúdo será enviado para o navegador como o corpo de sua resposta.

Objeto session

Este objeto implícito de JSP representa a sessão atual de um usuário individual. Todas as solicitações feitas por um usuário, que são parte de uma única série de interações com o servidor da web, são consideradas parte de uma sessão. Desde que novas solicitações por aqueles usuários continuem a ser recebidas pelo servidor, a sessão persiste. Se, no entanto, um certo período de tempo passar sem que qualquer nova solicitação do usuário seja recebida, a sessão expira. O objeto session, então armazena informações a respeito da sessão. Os dados específicos de aplicação são tipicamente adicionados à sessão através de atributos, usando os métodos da interface javax.servlet.http.HttpSession. O objeto session não está disponível para todas as páginas JSP, seu uso é restrito às páginas que participam do gerenciamento da sessão. Isto é indicado através do atributo session da diretiva page. O padrão é que todas as páginas participem do gerenciamento de sessão. Se o atributo estiver definido para false, o objeto não estará disponível e seu uso resultará em um erro de compilação quando o recipiente JSP tentar traduzir a página para um servlet.

Objeto application

Este objeto implícito representa a aplicação à qual a página JSP pertence. Ela é uma instância da interface javax.servlet.ServletContext. As páginas JSP estão agrupadas em aplicações de acordo com suas URLs. Este objeto permite acessar informações do container, interagir com o servidor e fornece suporte para logs.

Objeto pageContext

O objeto pageContext é uma instância da classe javax.servlet.jsp.PageContext, e fornece acesso programático a todos os outros objetos implícitos. Para os objetos implícitos que aceitam atributos, o objeto pageContext também fornece métodos para acessar aqueles atributos. Além disso, o objeto pageContext implementa métodos para transferir controle da página atual para uma outra página, temporariamente, para gerar output a ser incluído no output da página atual, ou permanentemente para transferir todo o controle.

Objeto exception

O objeto exception é uma instância da classe java.lang.Throwable. O objeto exception não está automaticamente disponível em todas as páginas JSP. Ao invés disso, este objeto está disponível apenas nas páginas que tenham sido designadas como páginas de erro, usando o atributo isErrorPage da diretiva page.

5. Formulários

Os formulários são ferramentas úteis e muito usadas em diversas aplicações, tais como: cadastro registros em um banco de dados, validação de senhas, envio de e-mail, envio de dados de uma pesquisa, entre outros. Hoje em dia é muito difícil desenvolver uma aplicação para WEB que não exija seu uso. Nesta seção aprenderemos como manipular formulários em aplicações JSP.

5.1 Formulários em HTML

Apresentamos abaixo um código para mostrar o formato de um formulário HTML e de seus objetos.

```
<html>
<body>

<!-- cabeçalho do formulário -->
<form name="nomedoformulario" action="paginajsp.jsp" method="get">

<!-- caixa de texto -->
<input type="text" name="variavel1" size=40 maxlength=40>

<!-- caixa de texto para senha -->
<input type="password" name="variavel2" size=40 maxlength=40>

<!-objeto do tipo radio -->
```

```
<input type="radio" name="variavel2" value="valordavariavel">Texto da Variável 2

<!-objeto do tipo checkbox -->
<input type="checkbox" name="variavel3" value="xxxxx"> Texto da Variável 3

<!-objeto do tipo select -->
<select name="variavel4">
<option value="valor1">Valor 1
<option value="valor2">Valor 2
<option value="valor3">Valor 3
</select>

<!- area de texto -->
<textarea name="variavel5" cols=40 rows=2>
Texto da Variavel 5
</textarea>

<!- objeto hidden, para enviar dados que o usuário não vê no formulário -->
<input type="hidden" name="asd" value="asd">

<!- botão -->
<input type="button" value="textodobotao">

<!- botao de limpar informações do formulário -->
<input type="submit" value="limpar">

<!- botao de enviar formulário -->
<input type="submit" value="ok">

<!- imagem colocada para funcionar com botao de envio de formulário -->
<input type="image" src="pathdaimage/image.gif">

<!- objeto para anexar arquivo -->

<input type="file" name="asdas" accept="asd">

</form>

</body>
</html>
```

É importante fazermos algumas observações sobre o código acima:

- No cabeçalho do formulário, indicamos através de action="pathdoarquivo/paginajsp.jsp" o arquivo JSP que receberá os seus dados.

- Cada objeto do formulário recebe um nome. Deve-se tomar bastante cuidado ao nomear tais objetos, isto porque, como sabemos, as variáveis Java são sensíveis à maiúscula e minúscula. Portanto, os objetos:

```
<input name="variavel1" type="text" value="">
<input name="Variavel1" type="text" value="">
```

São objetos diferentes, pois receberam nomes diferentes (variavel1 e Variavel1).

5.2 Envio de dados

Neste exemplo (bastante simples) veremos como enviar dados a partir de um formulário a uma página JSP.

Arquivo: teste.jsp

```
<html>
<body>

<center><h1> <%= request.getParameter("teste") %> </h1></center>

<form action="teste.jsp" method=get>
<input type="text" name="teste" size=40 maxlength=40><br>
<input type="submit" value="enviar">
</form>

</body>
</html>
```

A página jsp acima, chamada "teste.jsp", contém um formulário que envia para ela mesma. O valor digitado em uma caixa de texto será mostrado como título da página.

Observe como fizemos isso:

- a página para qual nós enviaremos os dados do formulário é designada no cabeçalho do formulário:
  <form **action="teste.jsp"** method=get>
- o nome do objeto caixa de texto caixa de texto ("teste") é usado na expressão request.getParameter("teste"). Note que se usássemos request.getParameter("Teste") (com T maiúsculo), a página não iria retornar o valor digitado na caixa de texto.

6. Cookies

Cookie é um mecanismo padrão fornecido pelo protocolo HTTP e que permite gravarmos pequenas quantidades de dados persistentes no navegador de um usuário. Tais dados podem ser recuperados posteriormente pelo navegador. Esse mecanismo é usado

quando queremos recuperar informações de algum usuário. Com os cookies, pode-se reconhecer quem entra num site, de onde vem, com que periodicidade costuma voltar. Para se ter uma idéia de como eles fazem parte da sua vida, dê uma olhada na sua máquina. Se você usa o Internet Explore 5.1 (ou superior), vá a c:\windows\cookies. No Communicator 4.7 (ou superior), os cookies ficam em c:\arquivos de programas\netscape\users. Os cookies em si não atrapalham ninguém, se propriamente usados.

Como padrão, os cookies expiram tão logo o usuário encerra a navegação naquele site, porém podemos configurá-los para persistir por vários dias. Além dos dados que ele armazena, um cookie recebe um nome; um servidor pode então definir múltiplos cookies e fazer a identificação entre eles através dos seus nomes.

Os cookies são associados ao URL da página que os manipula.

6.1Gerenciando Cookies

Os cookies são definidos por um servidor da web. Quando um usuário solicita um URL cujo servidor e diretório correspondam àqueles de um ou mais de seus cookies armazenados, os cookies correspondentes são enviados de volta para o servidor. As páginas JSP acessam os seus cookies associados através do método **getCookies()** do objeto implícito request. De forma similar, as páginas JSP podem criar ou alterar cookies através do método **addCookie()** do objeto implícito response. Esses métodos são resumidos na tabela abaixo:

Objeto Implícito	Método	Descrição
request	getCookies()	retorna uma matriz de cookies acessíveis da página
response	addCookie()	envia um cookie para o navegador para armazenagem/modificação

6.2 A classe Cookie

Manipulamos um cookie através de instâncias da classe javax.servlet.http.Cookie (calma, você não precisa se preocupar com isso. É que o container JSP insere o comando "import javax.servlet.http.*;" automaticamente no servlet associado que é gerado na compilação da página JSP).

Essa classe fornece apenas um tipo de construtor que recebe duas variáveis do tipo String, que representam o nome e o valor do cookie. O cookie tem a seguinte sintaxe:

```
Cookie cookie = new Cookie("nome da fera" , "valor da fera");
```

Abaixo apresentamos os métodos fornecidos pela classe Cookie:

Método	Descrição
getName()	retorna o nome do cookie
getValue()	retorna o valor armazenado no cookie
getDomain()	retorna o servidor ou domínio do qual o cookie pode ser acessado
getPath()	retorna o caminho da URL do qual o cookie pode ser acessado
getSecure()	indica se o cookie acompanha solicitações HTTP ou HTTPS.
setValue()	atribui um novo valor para o cookie
setDomain()	define o servidor ou domínio do qual o cookie pode ser acessado
setPath(nome do path)	define o caminho de URL do qual o cookie pode ser acessado
setMaxAge(inteiro)	define o tempo restante (em segundos) antes que o cookie expire
setSecure(nome)	retorna o valor de um único cabeçalho de solicitação como um número inteiro

Depois de construir uma nova instância, ou modificar uma instância recuperada através do método getCookies(), é necessário usar o método addCookie( ) do objeto response, com a finalidade salvar no navegador do usuário as alterações feitas no cookie. Para apagar um cookie utilizamos a seguinte técnica: chamamos o método "setMaxAge(0)" com valor zero e depois mandamos gravar chamando o método "addCookie( )". Isso faz com que o cookie seja gravado e imediatamente (após zero segundos) expira.

6.3 Definindo um cookie

O primeiro passo, então, ao usar um cookie dentro de uma página, é defini-lo. Isto é feito criando uma instância da classe Cookie e chamando os métodos "sets" para definir os valores de seus atributos.

Arquivo addcookie.jsp

```
<%
String email = request.getParameter("email");
String cookieName = "cookieJSP";
Cookie cookieJSP = new Cookie(cookieName, email);
cookieJSP.setMaxAge(7 * 24 * 60 * 60); //define o tempo de vida como 7 dias
(604800 segundos)
cookieJSP.setVersion(0); //versão 0 da especificação de cookie
cookieJSP.setSecure(false); //indica que o cookie deve ser transferido pelo
protocolo HTTP padrão
cookieJSP.setComment("Email do visitante"); //insere um comentário para o
cookie
```

```
response.addCookie(cookieJSP); //grava o cookie na máquina do usuário
%>
<html>
<head>
<title>Grava Cookie</title>
</head>
<body>
<h1>Grava Cookie</h1>
Esta página grava um cookie na sua máquina.<br>
<a href='readcookie.jsp'>Lê conteúdo do cookie</a>
</body>
</html>
```

Vamos criar uma página html com um formulário que irá fornecer um email que será gravado pela página "addcookie.jsp":

Arquivo: mailtocookie.html

```
<html>
<body>
<form action="addcookie.jsp">
<input type="text" name="email">
<input type="submit" value="ok">
</body>
</html>
```

No exemplo acima o cookie é identificado pelo nome cookieJSP e recebe o valor passado pelo usuário através de um formulário.

6.4 Recuperando um Cookie

A página jsp vista anteriormente tem a finalidade de receber um valor (email) passado através de um formulário de uma página html. Este valor é armazenado de forma persistente em um cookie, e pode ser acessado pelas outras páginas JSP que compartilham o domínio e o caminho originalmente atribuídos ao cookie. Os cookies são recuperados através do método getCookies( ) do objeto implícito request. A página abaixo mostra um exemplo de recuperação do valor de um cookie.

Arquivo: readcookie.jsp

```
<%
String cookieName = "cookieJSP";
Cookie listaPossiveisCookies[] = request.getCookies();
Cookie cookieJSP = null;
if (listaPossiveisCookies != null) {
//quando não existe cookies associados o método getCookies() retorna um valor
```

```
null
  int numCookies = listaPossiveisCookies.length;
  for (int i = 0 ; i < numCookies ; ++i) {
    if (listaPossiveisCookies[i].getName().equals(cookieName)) { //procura pelo
cookie
      cookieJSP = listaPossiveisCookies[i];
      break;
    }
  }
}
%>
<html>
<body>
<h1>Lê Cookie</h1>
<% if (cookieJSP != null) { %>
A pagina "addcookie" gravou o seguinte email: <%= cookieJSP.getValue() %>
<% }
else { %>
O cookie não gravou ou o prazo do cookie expirou.
<% } %>
</body>
</html>
```

O primeiro scriptlet nesta página recupera os cookies associados aquela página e depois tenta encontrar um cookie identificado pelo nome "cookieJSP".

6.5 Considerações finais sobre cookies

Apesar da praticidade de se utilizar os cookies oferecidos pelo protocolo HTTP, devemos fazer algumas considerações quanto a sua utilização:

- O tamanho dos dados armazenados (nome e valor) não devem ultrapassar 4K.
- O navegador pode armazenar múltiplos cookies, contanto obedece a um certo limite. Um navegador armazena até 20 cookies por configuração de domínio. O navegador armazena também 300 cookies em geral. Se quaisquer uns destes dois limites forem alcançados, espera-se que o navegador exclua cookies menos utilizados.
- O domínio atribuído a um cookie deve ter pelo menos dois pontos. Quando nenhum domínio é especificado na criação de um cookie através do método setDomain(nome do domínio), então ele só poderá ser lido apenas pelo host que originalmente o definiu.

7. Exemplos

Apêndice A

Resumo geral da sintaxe JSP. É um ótimo guia para eventuais dúvidas durante o desenvolvimento.

	Descrição	Exemplo
Declarações	Declara variáveis e métodos válidos no script daquela página.	<%! declaração; [declaração;]+ ... %>
Expressões	Contém uma expressão válida no script daquela página.	<%= expressão %>
Scriptlet	Contém um fragmento de código válido no script daquela página.	<% fragmento de um código com uma ou mais linhas %>
Comentário HTML	Cria um comentário que é enviado para o cliente viabilizar no código da página.	<!-- comentário [<%= expressão %>] -->
Comentário JSP	É visto apenas no fonte do JSP mas não é enviado para o cliente.	<%-- comentário --%> ou <% /* comentário */ %>
Diretiva "Include"	Inclui um arquivo estático, analisando os elementos JSP daquela página.	<%@ include file="URL relativa" %>
Diretiva "Page"	Define atributos que serão aplicados a uma página JSP.	<%@ page [atributo = valor(es)] %> atributos e valores: - language="java" - extends = "package.class" - import = "{package.class" | package.* }, ..."] - session = "true | false" - buffer = "none | 8kb | sizekb" - autoFlush = "true | false" - isThreadSafe = "true | false" - info = "text" - errorPage"relativeURL" - contentType = "{mimeType [; charset = characterSet] text/html; charset = isso-8859-1}" - isErrorPage = "true | false"
Diretiva Taglib	Define uma biblioteca tag e um prefixo para uma tag padrão usada na página JSP.	<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>

<tagPrefix:name>	Acessa um padrão de funcionalidade de uma tag.	<tagPrefix:name attribute="value" + ... /> <tagPrefix:name attribute="value" + ... > other tags and data </tagPrefix:name>
<jsp:forward>	Redireciona uma requisição para um arquivo HTML, JSP ou servlet para processar.	<jsp:forward page="{relativeURL | <%= expressão %>}"

Referências

[1] Star Developer Stardeveloper.com : Java Server Pages and Servlets Articles. Star Developer, 2001. Disponível online em http://stardeveloper.com.

[2] Sun Microsystems JavaServer Pages(TM) Technology. Sun Microsystems, 2001. Disponível online em http://java.sun.com/products/jsp/.

[3] Orion Application Server OrionServer Taglibs Tutorial. IronFlare AB, 2002. Disponível online em http://www.orionserver.com/. Acessado em
setembro de 2002.

Para saber mais sobre o OPEN e seus projetos visite:

http://www.open.org.br