8

A página HTML

AS PROPRIEDADES DE UMA PÁGINA HTML incluem seus elementos, representados pelos descritores HTML, atributos como cor de fundo ou cor dos links, e informações enviadas pelo servidor como cookies, URL, referenciador e data da última modificação. Todas essas propriedades são acessíveis através de JavaScript, e várias podem ser alteradas.

Além de permitir o acesso às propriedades, JavaScript também define vários métodos para gerar HTML e criar páginas novas, em tempo de exibição.

A única forma de ter acesso a uma página é através da propriedade `document`, que qualquer objeto do tipo *Window* possui. A página da janela onde roda o script pode ser acessada diretamente pela propriedade `window.document`, ou simplesmente `document`. Esta propriedade possui métodos e propriedades definidos pelo tipo *Document*, apresentado neste capítulo.

Objeto Document

O objeto `document` representa o documento HTML atual. `document` é uma propriedade de `window` e, portanto, pode ser usado sem fazer referência a `window`:

```
window.document     // ou simplesmente document
```

Para ter acesso a páginas de outras janelas, é preciso citar a referência *Window* que possui a propriedade `document`:

```
janela = open("nova.html");
janela.document.bgColor = "green";
parent.fr1.fr1_2.document.forms[0].b1.click();
```

No restante deste capítulo, usaremos `document` (e não `window.document`), como fizemos com as propriedades de `window` nos capítulos anteriores, para se referir à página da janela atual.

As propriedades do tipo *Document* estão listadas na tabela abaixo, indicando quais as propriedades que podem ser alteradas, e o tipo de dados que contém.

Propriedade	**Acesso**	**Função**
`bgColor`	read / write	Contém *String*. Define ou recupera cor de fundo da página. Pode um string contendo código hexadecimal do tipo `#rrggbb` ou nome da cor (red, blue, navy, etc.)
`fgColor`	r / w	Contém *String*. Define ou recupera cor do texto na página.
`linkColor`	r / w	Contém *String*. Define ou recupera cor de links na página.
`alinkColor`	r / w	Contém *String*. Define ou recupera cor de links ativos.
`vlinkColor`	r / w	Contém *String*. Define ou recupera cor de links visitados.
`title`	r	Contém *String*. Recupera o título (`<TITLE>`) do documento.
`links`	r	Contém *Array* de objetos *Link*. Para obter a quantidade de links `<A HREF>` no documento: `document.links.length`
`applets`	r	Contém *Array* de objetos *Applet*. Para obter a quantidade de applets `<APPLET>` no documento: `document.applets.length`
`anchors`	r	Contém *Array* de objetos *Anchor*. Para obter a quantidade de âncoras `<A NAME>` no documento: `document.anchors.length`
`embeds`	r	Contém *Array* de objetos *PlugIn*. Para obter a quantidade de plugins `<EMBED>` no documento: `document.plugins.length`
`plugins`	r	Contém *Array* de objetos *PlugIn*. Mesma coisa que `embeds`
`images`	r	Contém *Array* de objetos *Image*. Para obter a quantidade de imagens `<IMG>` no documento: `document.images.length`
`location`	r	Contém *String* com URL do documento.
`URL`	r	Mesma coisa que `location`.
`referrer`	r	Contém *String* com URL do documento que contém um link para o documento atual.
`lastModified`	r	Contém *String*. A string recebida informa a data da última modificação do arquivo. Está no formato de data do sistema. Pode ser convertida usando `Date.parse()` e transformada em objeto ou automaticamente em String.
`domain`	r / w	Contém *String* com o domínio dos arquivos referenciados.
`cookie`	r / w	Contém *String*. Usado para ler e armazenar preferencias do usuário no computador do cliente.

As propriedades `bgColor`, `fgColor`, `linkColor`, `vlinkColor` e `alinkColor` alteram a aparência da página. Correspondem aos atributos `BGCOLOR`, `TEXT`, `LINK`, `VLINK` e `ALINK` do descritor HTML `<BODY>`, respectivamente. Existem desde as primeiras versões do JavaScript, mas só podiam ser alteradas antes que a página fosse montada. Nos browsers modernos, é possível mudar essas propriedades em tempo de exibição. Pode-se ter, por exemplo, um link que 'apaga a luz' quando o mouse passa sobre ele:

```
<a href="..."
   onmouseover="document.bgColor='black'"
   onmouseout=""document.bgColor='white'"> Não se aproxime! </a>
```

As seis últimas propriedades da lista acima possibilitam a obtenção de informações do arquivo HTML e do domínio onde reside. Uma das propriedades mais úteis de `document` é `lastModified`, que retorna a data de última modificação do arquivo. Para imprimi-la na página:

```
document.write("<p>Última modificação: " + document.lastModified);
```

As propriedades `location`, `domain` e `referrer` contém informações sobre a localização da página. A propriedade `document.referrer` contém a URL da página que contém um link para o documento atual. `document.referrer` pode apresentar o valor `null` se a carga da página não foi resultante da seleção de um link:

```
<p>Você acaba de vir de
<script language=JavaScript>
   document.write(document.referrer);
</script>. Seja bem vindo!
```

A propriedade `document.domain` representa o domínio da página atual. É a mesma informação que existe em um `<BASE HREF="`*`url`*`">`. `document.location` representa o endereço da página atual. É útil quando se precisa gerar uma página nova com um link para a página atual (que a criou).

Métodos

Os métodos de *Document* são usados principalmente para gerar HTML e criar novas páginas em tempo de exibição e de carga. Os métodos tanto podem ser aplicados na janela atual ou em outras janelas de forma a gerar documentos novos.

Método	**Ação**

`write("string")` ou `writeln()`
`write("arg1","arg2",...`
`,"argn")`
	Recebe e concatena zero ou mais argumentos separados por vírgulas e os escreve na página atual.

`open()` ou `open("tipo/subtipo")`	Abre um canal de fluxo de dados para `document` de forma que as chamadas `document.write()` subseqüentes possam acrescentar dados ao documento. o `tipo/subtipo` é por *default* `text/html`.
`close()`	Imprime na página qualquer texto não impresso enviado à `document` e fecha o fluxo de saída de dados.
`clear()`	limpa a janela ou frame que contém `document`.

O método `write()` é provavelmente o método mais usado em JavaScript. Sua principal aplicação é a geração de HTML durante a carga da página. Uma chamada ao

método `write()`, depois que a carga e exibição da página foi concluída, não funciona, pois o canal de gravação do arquivo já foi fechado. Mas o canal pode ser reaberto com uma chamada à `document.open()`:

```
document.open();
document.write("Esta é a última linha do arquivo");
document.close();
```

A linha `document.close()` é essencial para que o texto seja exibido. Uma chamada ao método `document.clear()`, depois de um `document.open()`, limpa a tela, permitindo que o texto apareça no início da tela.

Com exceção de `write()`, porém, há poucas aplicações para os outros métodos no documento atual. Imprimir no final do arquivo não é a melhor forma de criar páginas dinâmicas. As aplicações mais interessantes são a geração dinâmica de páginas novas. Mostraremos, na seção a seguir, um exemplo de geração de páginas *on-the-fly*.

Geração de páginas on-the-fly

A melhor forma de gerar uma nova página é começar com uma nova janela. Através da referência da nova janela, pode-se abrir então um fluxo de dados para escrever nela.

O primeiro passo, portanto, é criar a nova janela:

```
w1 = open("");     // abre janela vazia, sem arquivo
```

Em seguida, abrir o documento para que possa receber dados enviados por instruções `write()`, posteriores:

```
w1.document.open();// abre documento para gravação
w1.document.write("<html><head>\n<title>Nova Página</title>\n");
w1.document.write("</head>\n<body bgcolor=white>\n");
w1.document.write("<h1 align=center>Nova Página</h1>");
```

Para que o documento completo seja exibido, é preciso que o fluxo de dados seja fechado, depois que todos as instruções `write()` tenham sido enviadas. Quando isto ocorre, quaisquer linhas que estejam na fila são impressas, e o documento é fechado.

```
w1.document.close();     // imprime documento na nova janela
```

Ao se visualizar o código-fonte do documento gerado, encontra-se:

```
<html><head>
<title>Nova Página</title>
</head>
<body bgcolor=white>
<h1 align=center>Nova Página</h1>
(...)
```

O exercício resolvido a seguir apresenta um exemplo completo. Veja mais exemplos no diretório `cap8/`.

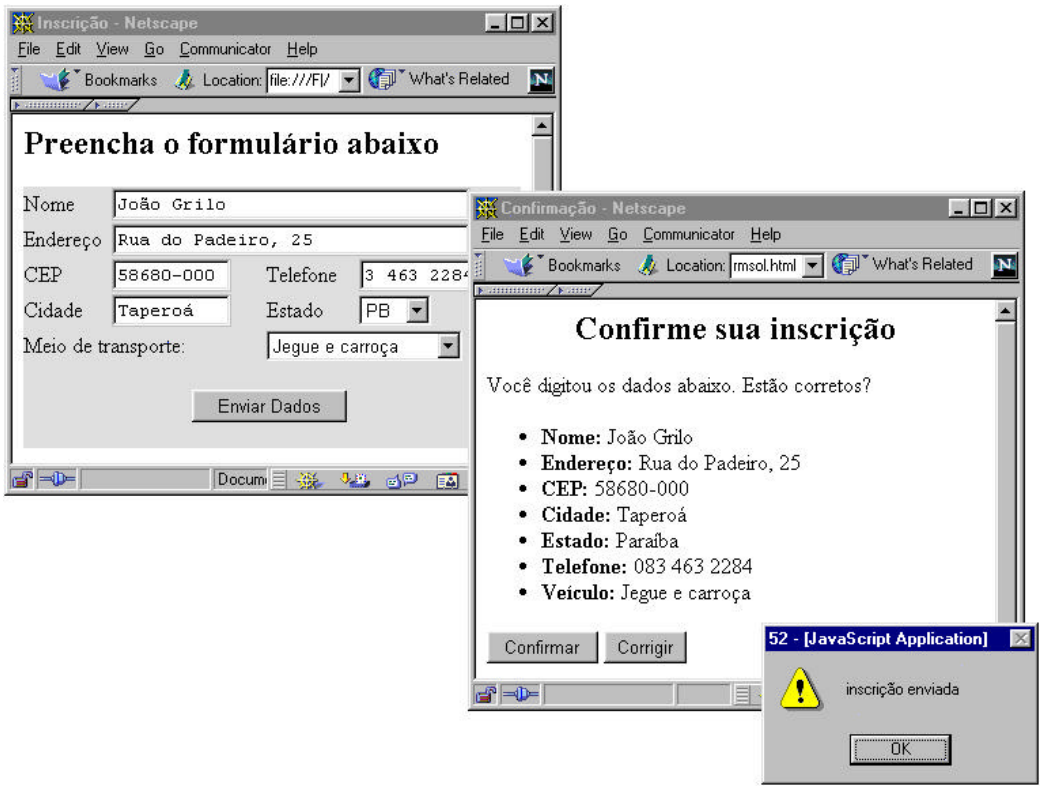
Exercício Resolvido

Este exercício consiste na construção de uma janela de confirmação para dados digitados em um formulário, antes do envio ao servidor, permitindo que o usuário verifique se os dados que selecionou estão corretos antes de enviá-los.

Crie um formulário como o mostrado na figura (use o esqueleto `form.html` disponível). Quando o usuário apertar o botão "Enviar Dados", uma nova janela deverá ser criada contendo a lista das informações que ele selecionou e dois botões "Corrigir" e "Confirmar".

Caso o usuário decida fazer uma correção, deve apertar o botão "Corrigir" que tornará ativa (`focus()`) a janela do formulário. O usuário pode então alterar quaisquer informações e enviar novamente. Após novo envio, a janela de confirmação deverá ser re-ativada ou criada caso tenha sido fechada (verifique se é preciso criar a janela ou não).

Caso o usuário confirme, os dados devem ser enviados para o servidor. Os dados digitados na primeira janela, portanto, devem estar presentes também na segunda janela em campos de formulário (`<hidden>`) para que o programa CGI no servidor possa recuperá-los. Como este exemplo é apenas uma simulação, o envio pode ser sinalizado através de uma janela de alerta.



A solução está na página seguinte.

Solução

O primeiro passo é a criação do formulário. Nós já fornecemos um esqueleto com um formulário (`cap8/form.html`) pronto. Precisaremos utilizar os nomes dos campos (atributo `NAME`) e saber o tipo de dispositivo de entrada usado. A listagem abaixo destaca os nomes dos campos e seus tipos em negrito:

```
<form>
(...)
<td>Nome</td>     <td colspan=3><input type=text name=Nome></td><tr>
<td>Endereço</td> <td colspan=3><input type=text name=Endereco></td></tr>
<tr><td>CEP</td>  <td><input type=text name=CEP size=10></td>
<td>Telefone</td> <td><input type=text name=Telefone></td></tr><tr>
<td>Cidade</td>   <td><input type=text name=Cidade size=10></td>
<td>Estado</td>   <td><select name=UF size=1 >
                      <option value="Acre">           AC </option>
                      <option value="Alagoas">        AL </option>
(...)
                      <option value="Santa Catarina"> SC </option>
                      <option value="Tocantins">      TO </option>
                    </select></td></tr>
<tr><td colspan=2>Meio de transporte:</td>
                    <td colspan=2><select name=Veiculo>
                      <option selected>Escolha um veículo</option>
                      <option> Porsche 911 </option>
(...)
                      <option> Jegue e carroça </option>
                    </select></td></tr>
<tr><td align=center colspan=4><br>
<input type=button value="Enviar Dados" onclick="enviar(this.form)">
<br> </td></tr></table>
</form>
```

No nosso formulário só há dois tipos de dispositivos de entrada: caixas de texto (`<input type=text>`) e caixas de seleção (`<select>`). No evento `onclick` do botão, um método `enviar()` é chamado, que passa o próprio formulário como argumento (para que possamos, dentro da função, referenciar o formulário através de uma variável local).

Para ler um valor da caixa de texto, utilizamos a propriedade `value`, do objeto *Text*. A leitura dos valores de objetos *Select* é mais complicada, pois as informações são armazenadas em objetos *Option*, de duas formas diferentes. O tipo *Select* têm uma propriedade `options`, que retorna um *Array* de objetos *Option*. Outra propriedade, `selectedIndex`, retorna o índice atualmente selecionado (veja mais sobre *Select* e *Options* no capítulo 10). Combinando os dois, obtemos o item selecionado. Veja como obter o ítem selecionado pelo usuário na caixa de seleção `UF`:

```
indice = document.forms[0].UF.selectedIndex; // obtem índice
item = document.UF.options[indice];
```

As informações do item selecionado podem estar em dois lugares:

- entre `<option>` e `</option>` (siglas dos estados) e
- no atributo `VALUE` de cada `<option>` (nomes dos estados por extenso)

O primeiro valor é obtido através da propriedade `text`, o segundo, através da propriedade `value`, por exemplo:

```
sigla = document.UF.options[indice].text;   // PR, BA, AC
estado = document.UF.options[indice].value; // Paraná, Bahia, Acre
```

Para evitar escrever sempre este longo *string* todas as vezes que uma variável do formulário for usada, criamos novas variáveis dentro da função `enviar()`:

```
function enviar(dados) {       // dados = document.forms[0]
      nome     = dados.Nome.value;
      endereco = dados.Endereco.value;
      cep      = dados.CEP.value;
      cidade   = dados.Cidade.value;
      uf       = dados.UF.options[dados.UF.selectedIndex].value;
      tel      = dados.Telefone.value;
      carro    = dados.Veiculo.options[dados.Opcao_1.selectedIndex].text;
  (...)
}
```

Antes de gerar uma nova página, o ideal é criá-la da forma convencional, diretamente em HTML como uma página estática. Desta forma, é fácil verificar sua aparência e até acrescentar mais recursos visuais. Nesta versão temporária, preenchemos os espaços onde estarão os dados a serem gerados dinamicamente com os nomes das variáveis (em negrito). Observe que eles também são enviados em campos `<hidden>`, para que possam ser repassados ao servidor (programa CGI). Esta é a página que queremos gerar:

```
<HTML>
<HEAD>
<TITLE>Confirmação</TITLE>
</HEAD>
<BODY bgcolor=white>
<H2 align=center>Confirme sua inscrição</H2>
<FORM action="/cgi-local/inscricao.pl"> <!-- programa CGI -->
<P>Você digitou os dados abaixo. Estão corretos?
<UL>
<LI><B>Nome:</B> nome </LI>
<LI><B>Endereço:</B> endereco </LI>
<LI><B>CEP:</B> cep </LI>
<LI><B>Cidade:</B> cidade </LI>
<LI><B>Estado:</B> estado </LI>
```

```
<LI><B>Telefone:</B> telefone </LI>
<LI><B>Veículo:</B> carro </LI>
</UL>
<!-- Campos necessarios para enviar os dados ao servidor -->
<INPUT type=hidden name=Nome value=" nome ">
<INPUT type=hidden name=Endereco value=" endereco ">
<INPUT type=hidden name=CEP value=" cep ">
<INPUT type=hidden name=Cidade value=" cidade">
<INPUT type=hidden name=Estado value=" estado">
<INPUT type=hidden name=Telefone value=" telefone ">
<INPUT type=hidden name=Veiculo value=" carro ">
<P>
<INPUT type=button onclick="alert('inscrição enviada')"
                   value="Confirmar">
<INPUT type=button value="Corrigir"
                   onclick="opener.focus()">
</FORM>
</BODY></HTML>
```

Agora é só colocar todo o código acima dentro de instruções `document.write()`, isolando as variáveis. As aspas precisam ser precedidas de um escape (`\"`) para que possam ser impressas. Primeiro precisamos abrir uma nova janela:

```
if (w1 != null) {  // verifica se a janela já está aberta!
   w1.focus();     // ... se estiver... simplesmente ative-a
} else w1 = open("", "janconf"); // ... se não, abra uma nova!
```

Depois abrimos um canal para escrever no documento da janela (`w1.document`), através do método `open()` e enviamos uma seqüência de instruções `write()`. Usamos as variáveis definidas antes para passar os dados à nova página:

```
w1.document.open();
w1.document.write("<html><head>\n<title>Confirmação</title>\n");
w1.document.write("</head>\n<body bgcolor=white>\n");
w1.document.write("<h2 align=center>Confirme sua inscrição</h2>\n");
w1.document.write("<form action=\"/cgi-local/inscricao.pl\">\n");
w1.document.write("<p>Você digitou os dados abaixo. Estão corretos?");
w1.document.write("<ul><li><b>Nome:</b> " + nome + "</li>\n");
w1.document.write("<li><b>Endereço:</b> " + endereco + "</li>\n");
w1.document.write("<li><b>CEP:</b> " + cep + "</li>\n");
w1.document.write("<li><b>Cidade:</b> " + cidade + "</li>\n");
w1.document.write("<li><b>Estado:</b> " + uf + "</li>\n");
w1.document.write("<li><b>Telefone:</b> " + tel + "</li>\n");
w1.document.write("<li><b>Veículo:</b> " + carro + "</li></ul>\n");

w1.document.write("<input type=hidden name=Nome value=\"" + nome + "\">\n");
w1.document.write("<input type=hidden name=Endereco value=\"" + endereco + "\">\n");
w1.document.write("<input type=hidden name=CEP value=\"" + cep + "\">\n");
```

```
w1.document.write("<input type=hidden name=Cidade value=\"" + cidade + "\">\n");
w1.document.write("<input type=hidden name=Estado value=\"" + uf + "\">\n");
w1.document.write("<input type=hidden name=Telefone value=\"" + tel + "\">\n");
w1.document.write("<input type=hidden name=Veiculo value=\"" + carro + "\">\n");

w1.document.write("<p><input type=button onclick=\"alert('inscrição enviada')\"
                                          value=\"Confirmar\">\n");
w1.document.write("<input type=button value=\"Corrigir\"
                                        onclick=\"opener.focus();\">\n");
w1.document.write("</form></body></html>\n");

w1.document.close();
w1.focus();

}
```

A solução com o código completo da página acima está em `cap8/formsol.html`. Veja também outros exemplos, como `job_form1.html`, no mesmo diretório.

Eventos

Os eventos do JavaScript que afetam a página também afetam as janelas. Estes eventos são chamados a partir dos atributos HTML listados abaixo, e são aplicáveis aos descritores `<BODY>` e `<FRAME>`. Já os vimos no capítulo anterior:

- `ONLOAD` – depois que a página é totalmente carregada
- `ONUNLOAD` – antes que a página seja substituída por outra

Exercícios

8.1 Escreva uma aplicação onde o usuário possa escolher três cores de uma lista (use um `<select>` ou `<input type=radio>`): uma cor de fundo, uma cor do texto e uma cor do link. Após os seletores, o documento deve conter um título, alguns parágrafos e links. Depois que o usuário clicar um botão “Visualizar”, a mesma página deve ser recarregada mostrando a combinação que ele escolheu.

8.2 Escreva uma aplicação que, através de uma interface de formulários, permita que o usuário monte uma página HTML sem precisar saber HTML. O usuário deverá poder escolher o título, dois subtítulos e os textos de duas seções. Também deve poder escolher cores (veja exercício anterior). A página deve oferecer duas opções:

- Um botão de “preview”, que irá abrir uma nova janela com o resultado esperado.

- Um botão “gerar página”, que irá colocar o código HTML correspondente à escolha do usuário em um `<textarea>`, para que possa copiar e colar em um arquivo de texto.

8.3 Escreva uma aplicação de ‘bate-papo narcisista’, utilizando duas janelas de browser com frames. A parte superior de cada janela é um documento em branco (use `blank.html`, no diretório `cap8/`) que irá crescer a medida em que linhas de texto são enviadas a partir dos frames inferiores de cada janela. Cada vez que uma mensagem for digitada no campo de texto do frame inferior e o botão “Enviar” apertado, ela deve aparecer na parte superior de ambas as janelas. A primeira janela deverá ter um botão “Arranjar Companhia” para que outra janela seja criada, tornando a conversa possível. Veja as figuras abaixo. A solução está no diretório `cap8/` (veja no arquivo `papoframe1.html`)