Aula 15: Os objetos do navegador

Nesta aula continuaremos a tratar dos objetos em JavaScript. Veremos agora detalhes sobre a hierarquia dos objetos de *window*, que funciona como objeto global. Depois você aprenderá dois comandos novos e detalhes dos objetos `location` e `history`.

Objetivos:
Aprender:
- a hierarquia dos objetos
- os comandos `with` e `for...in`
- as características e métodos do objeto `location`
- as características e métodos do objeto `history`

Pré-requisitos:
As aulas 1 e 2 são importantes para a seção que estuda o objeto `location`. A aula 13, na seção que introduziu os conceitos de objeto. A aula 12, onde são explicados os conceitos de variáveis globais.

1. Objetos do *browser*

Os objetos do *browser* são criados pelo próprio *browser* de forma a refletir características do navegador e da página em exibição. Há dois objetos básicos: `navigator` e `window`.

O objeto `navigator` representa o próprio navegador e através dele é possível controlar o *browser* e obter informações sobre suas características. O objeto *window* representa uma janela do *browser* e contém nele todos os elementos da janela.

Já falamos que quando o interpretador da linguagem inicia, uma das primeiras coisas que faz, antes de executar qualquer código de JavaScript, é criar um **objeto global.** O objeto global está no topo da cadeia de escopo. As propriedades deste objeto global são as variáveis globais dos programas JavaScript. Quando você define uma variável global, o que está realmente fazendo é definir uma propriedade do objeto global. Além disso, todas as funções predefinidas e propriedades do ambiente JavaScript também são propriedades do objeto global.

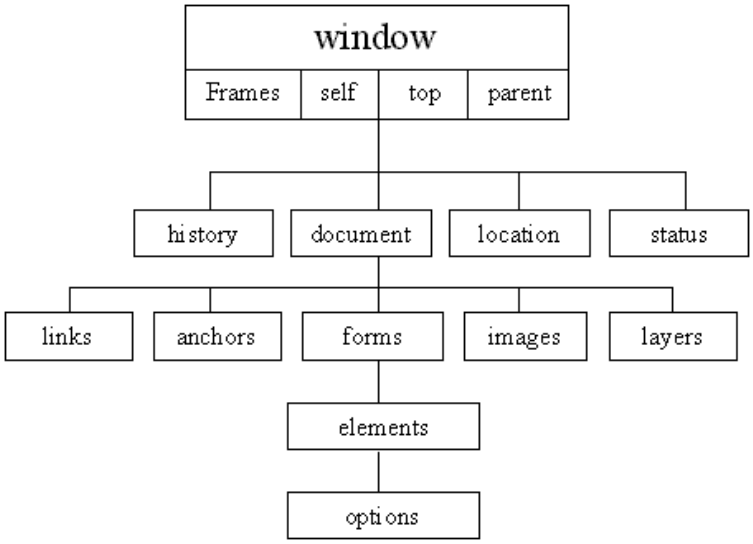
Na forma *client-side* da linguagem, que é o objetivo desse nosso curso, o objeto *window* serve como objeto global para todo o código contido na janela do navegador que ele representa. O objeto *window* define propriedades e métodos que permitem manipular a janela do *browser*. Ele também define propriedades que referem a outros objetos, como as propriedades `document` (do objeto `Document`). Há ainda duas propriedades que referenciam a si mesmas (auto-referenciais): `window` e `self`. Todas as variáveis globais são definidas como propriedades do `window`.

2. Hierarquia dos Objetos

Acabamos de ver que *window* é o objeto principal na forma *client-side* de JavaScript e quase todos os outros objetos do *browser* podem ser acessados através dele seguindo uma cadeia de propriedades. Esta cadeia é formada por objetos contidos no objeto *window*, que por sua vez contém outros objetos numa relação hierárquica. Neste sentido, pode-se dizer que o objeto *window* está no topo da hierarquia.

A figura abaixo mostra a hierarquia de objetos dentro de *window*, onde cada objeto ligado a um objeto acima representa uma relação de "contido em".

Figura 15.1 - Hierarquia dos objetos de `window`



O esquema acima mostra que dentro do objeto *window*, na forma de propriedades, encontramos os objetos `history, location, document` e `status`. Para se ter acesso a qualquer um deles é necessário fazer uso do operador ponto ("."). Veja a sintaxe abaixo:

```
window.history.back();
window.location = "http://www.cederj.gov.br";
window.document.write("<H1>Seu " +
                      "Site<H1>");
window.document.forms[0].text =
          "as vezes as coisas podem " + "ficar bem
```

```
          complexas";
```

Como a cadeia de objetos conectados pode ficar bem longa, por uma questão de simplificação, JavaScript permite referenciar os objetos internos a *window* sem citar a própria *window*. Assim, é possível reescrever as linhas anteriores como:

```
history.back();
location = "http://www.cederj.gov.br";
document.write("<H1>Seu Site</H1>");
document.forms[0].text =
          "as vezes as coisas podem " +
          "ficar bem complexas";
```

Já deve ter dado para notar que entender a hierarquia e que objetos estão contidos em outros é crucial para uma programação bem-sucedida. Na figura anterior mostrou-se só algumas propriedades de alguns objetos. A maioria dos objetos tem mais propriedades que as mostradas ali.

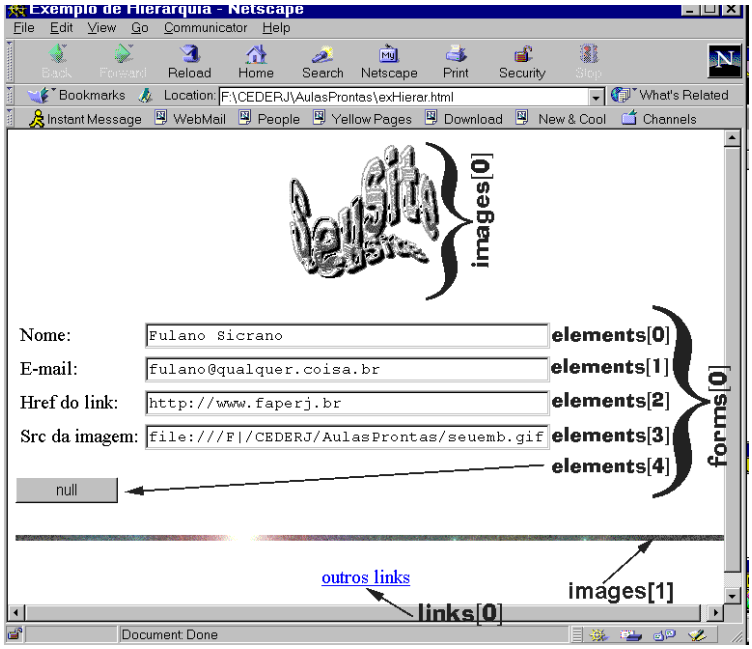
Para melhor compreender a hierarquia de objetos dentro do *window* é melhor observar um exemplo real. Observe as seguintes linhas de código:

```
<HTML>
<HEAD>
<TITLE>Exemplo de Hierarquia</TITLE>
</HEAD>
<BODY bgcolor=white>
<CENTER><IMG SRC="seuemb.gif"></CENTER>
<FORM>
<TABLE>
<TR>
<TD>Nome:</TD>
<TD><INPUT TYPE=TEXT NAME="nome" SIZE=40
VALUE="Fulano Sicrano"></TD></TR>
<TR>
<TD>E-mail:</TD>
<TD><INPUT TYPE=TEXT SIZE=40 VALUE=
"fulano@qualquer.coisa.br"></TD></TR>
<TR>
<TD>Href do link:</TD>
<TD><INPUT TYPE=TEXT SIZE=40></TD></TR>
<TR>
<TD>Src da imagem:</TD>
<TD><INPUT TYPE=TEXT SIZE=40></TD></TR>
</TABLE>
<BR><INPUT TYPE=BUTTON Name=botao Value="Pressione">
</FORM>
<CENTER>
<IMG SRC="0.gif" height=6>
<P><A HREF="http://www.faperj.br">outros
links</A></P>
</CENTER>
<SCRIPT LANGUAGE="javascript">
document.forms[0].elements[2].value =
document.links[0].href;
```

```
document.forms[0].elements[3].value =
document.images[0].src;
document.write("O nome que aparece " +
    "no campo nome do formulário é " +
    document.forms[0].elements[1].value);
texto = prompt("Qual texto deve " +
                "aparecer no botão ? ");
document.forms[0].elements[4].value = texto;
</SCRIPT>
</BODY>
</HTML>
```

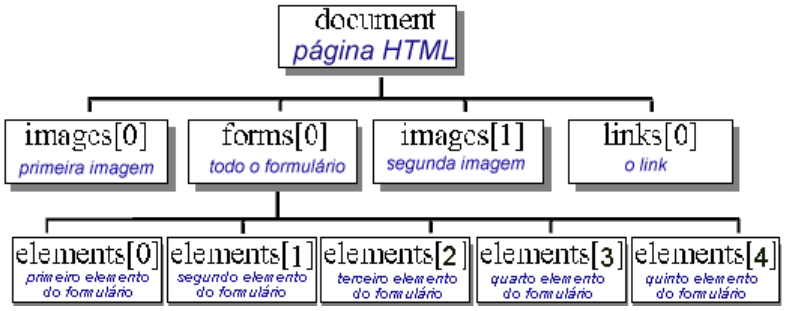
Estas linhas produzem uma página como a mostrada na figura 15.2.

Figura 15.2 - Página gerada e os diversos níveis dos seus componentes



Esta página tem um objeto *window* com os elementos mostrados na figura 15.3.

Figura 15.3 - Elementos da página exemplo



O objeto `document`, como pode ser observado, tem um *array* de imagens, um de formulários e um de *links*. O número de elementos de cada um destes tipos contidos na janela indica o número de elementos de cada *array*. Assim, a página do exemplo contém duas imagens e o *array* imagens contém dois objetos, um para cada imagem numerada de acordo com a ordem que aparece

na página. O formulário por sua vez também pode ter diversos elementos, onde cada elemento pode ser um dos elementos de interação possíveis de pertencer a um formulário. No formulário do exemplo das figuras 15.2 e 15.3 tem-se um *array* de 5 elementos, todos `<input>` , 4 do tipo `text` e o último do tipo `button`.

3. Comando `with`

Neste momento torna-se necessário introduzir um novo elemento que facilita a manipulação dos objetos, permitindo uma economia de digitação quando se deseja referenciar várias propriedades e métodos de um mesmo objeto. O comando `with` é usado para temporariamente modificar a cadeia de escopo.

Ele tem a seguinte sintaxe:
`with (` `objeto` **`)`** `comandos`

A sua função é adicionar `objeto` para a frente da cadeia de escopo, executar os `comandos` e então re-estabelecer o estado original da cadeia de escopo.

Ao indicar um objeto entre os parênteses, os métodos e propriedades dentro das chaves vão ser considerados como pertencentes ao objeto indicado (se for possível). Assim, no exemplo anterior, no lugar do texto que está escrito entre as tags <SCRIPT....>...</SCRIPT>, é possível escrever:

```
with(document){
  forms[0].elements[2].value=links[0].href;
  forms[0].elements[3].value=images[0].src;
  write("O nome que aparece no campo " +
      "nome do formulário é " +
        forms[0].elements[0].value +
        "cujo e-mail é:" +
      forms[0].elements[1].value);
  texto = prompt("Qual texto deve " +
                  "aparecer no botão ? ");
  forms[0].elements[4].value = texto;
}
```

Este comando é especialmente útil no caso de encadeamentos maiores de objetos, como:

```
with(document.forms[0].elements[1]){
        value = "Fulano da Tal";
        name  = "Nome";}
```

Mas é importante dizer que o código usando `with` roda mais devagar que um equivalente sem `with`, além de poder em alguns casos ser de difícil entendimento e ocasionar má interpretação.

Observe que uma outra maneira possível de economizar digitalização seria criar variáveis, no caso acima esta forma legítima de economizar código poderia ser reescrever as linhas anteriores como:

```
var elemento=document.forms[0].elements[1];
elemento.value="Fulano de Tal";
```

```
elemento.name="Nome";
```

4. Comando `for…in`

O comando `for` tem uma outra sintaxe diferente daquela que vimos em um capítulo anterior: `for(...;...;...).` Esta outra sintaxe simplifica a manipulação e permite percorrer todas as propriedades de um objeto. Esta sintaxe é:

```
for ( variável in objeto )
     comandos
```

Nesta sintaxe `variável` pode ser o nome de uma variável, um elemento de um *array* ou uma propriedade de um objeto (ou seja, alguma coisa possível de estar no lado esquerdo de uma expressão de atribuição). `objeto` é o nome de um objeto ou uma expressão que pode ser equivalente a um objeto, e como sempre `comandos` representa um comando único ou um grupo de comandos em bloco.

O comando `for...in` proporciona um meio de percorrer em *loop* as propriedades de um objeto. O corpo do comando é executado uma vez para cada uma das propriedades do objeto. Antes do corpo do *loop* ser executado, o nome de uma propriedade do objeto é atribuída à `variável` como uma *string*. Dentro do corpo do *loop*, você pode usar a variável para observar o valor da propriedade do objeto usando o operador `[ ].`

É possível através do `for` recuperar, um a um, os nomes das propriedades e, posteriormente, acessá-las utilizando o objeto como um *array*. O laço a seguir imprime na página cada uma das propriedades do objeto `navigator`:

```
for (prop in navigator)
   document.write(prop + " = " +
                  navigator[prop] );
```

Enquanto que as linhas:

```
primElem=document.forms[0].elements[0];
for (i in primElem)
   document.write("<br>" + i + "= " +
                  primElem[i]);
```

imprimem todas as propriedades do primeiro elemento de `document.forms[0].`

O *loop* `for/in` não especifica a ordem com que as propriedades do `objeto` serão atribuídas à `variável.` As propriedades dos objetos definidas como `não enumeradas` (como muitas propriedades embutidas e todos os métodos embutidos) não podem ser examinadas pelo `for/in.`

5. Objeto `location`

O objeto `location` contém informações relacionadas a URL corrente. Ele representa e controla a localização do navegador, sendo armazenado na propriedade `location` do objeto `window`.

As propriedades de `location` são todas do tipo *read/write* e *strings*. Referem-se à *string* da URL e suas várias partes, que foram vistas nas aulas 1 e 2 e têm o seguinte formato geral:

```
protocol://hostname:port/pathname?search#hash
```

A propriedade mais importante deste objeto é a *string* que contém o endereço completo da URL corrente: propriedade `href`. Cada pedaço da URL também pode ser referenciada individualmente através das propriedades: `protocol`, `host`, `hostname`, `port`, `pathname`, `hash` e `search` (dados de um formulários enviados pelo método `get`).

Ao invés de utilizar `location` ou `location.href` para substituir a URL corrente por uma completamente nova, você pode modificar só uma porção da URL corrente atribuindo *strings* para a propriedade do objeto `location` que você quer modificar. Por exemplo, se você quiser mudar para uma certa localização dentro da mesma página pode usar só a propriedade `hash` com o nome da *âncora* do ponto desejado.

As propriedades e os métodos do objeto `location` encontram-se nas tabelas 15.1 e 15.2 que seguem. Para entender melhor estas propriedades, considere a seguinte URL fictícia:

```
"http://www.ic.uff.br:1234/~aconci/CV.html?query=JavaScript&matches=66#results"
```

15.1 - Propriedades do objeto do browser `location`

Propriedade	Significado
`hash`	Especifica a âncora na *string* da URL corrente, deve incluir o símbolo # inicial, na URL exemplo seria `#results` .
`host`	Combina as partes da URL correspondentes ao *hostname* e a port. Na URL exemplo seria `www.ic.uff.br:1234` .
`hostname`	Parte da URL correspondentes ao *hostname*, no exemplo: `www.ic.uff.br`
`href`	Especifica o texto completo da URL: `http://www.ic.uff.br:1234/~aconci/CV.html?JavaScript#results`
`pathname`	Parte da URl, correspondente ao `path`: `/~aconci/CV.html`
`port`	Parte da URL correspondente à `port`. No exemplo é: `1234`
`protocol`	Especifica a parte da URL correspondente ao protocolo, inclui `":"`. No exemplo é `http:`
`search`	Parte da URL correspondente à consulta (query), inclui "?". No exemplo seria: `?query=JavaScript&matches=66`

Este objeto contém dois métodos. O método `reload()` permite recarregar a página corrente a partir do servidor Web. O método `replace()` carrega e mostra uma URL especificada sem incluir um novo dado na lista de páginas visitadas, ou seja, substituindo a página antiga pela nova na lista `history` do *browser*. (Esta lista ficará mais clara depois que estudarmos o objeto `History`, na seção 6 desta aula). Como você deve ter observado, este método não tem o mesmo efeito de apenas definir a propriedade `location` ou `location.href` com um novo endereço de site. Depois de usar `replace()` não será possível retornar à página anterior usando o botão **`Back`**

do seu *browser*.

15.2 - Métodos do objeto do browser `location`

Método	Significado
`reload()` `reload(booleano)`	Recarrega o documento da cache ou do servidor.
`replace(url)`	Substitui o documento corrente pelo especificado no parâmetro `url` tomando a sua posição na seção `history` do *browser*.

O método `reload` pode ser usado com ou sem parâmetros. Se o argumento for omitido ou for o booleano `false`, o método só realmente recarrega uma nova página do servidor se ela tiver sido modificada desde a última vez que foi visitada, caso contrário carrega a cópia do `cache`. Se o argumento for `true` ela sempre será recarregada do servidor, não sendo consultado o cache do usuário.

6. Objeto `history`

A propriedade `history` (histórico) do objeto *window* se refere ao objeto `history` da janela ou frame: `window.history` ou `frame.history`. O objeto `history` dá acesso (apenas para leitura) à lista de URLs navegadas por uma janela ou *frame*. Este objeto contém as 4 propriedades mostradas na tabela 15.3 e suporta os 3 métodos da tabela 15.4.

15.3 - Propriedades do objeto do *browser* `history`

Propriedade	Significado
`length`	Número de páginas armazenadas na lista do objeto `history`.
`current`	URL da página atual.
`next`	URL da página posterior a atual na lista.
`previous`	URL da página anterior a atual na lista.

Como o histórico das páginas visitadas é uma informação privada, por questões de segurança, alguns *browsers*, que não suportam ***signed scripts* ,** restringem as formas como as propriedades deste objeto podem ser usadas.

15.4 - Métodos do objeto do browser `History`

Método	Significado
`go(n)`	Permite ir para a *n-ésima* página. Valores positivos movem para as próximas URL e negativos para as anteriores.
`back()`	Volta para a página anterior.
`forward()`	Vai para a página seguinte.

O exemplo a seguir ilustra a utilização do objeto `history`:

```
<form>
<input type=button value="<-- 2"
onclick="history.go(-2)">
<input type=button value="Prev."
```

```
onclick="history.back()">
<input type=button value="Prox."
onclick="history.forward()">
<input type=button value="2 -->"
onclick="history.go(2)">
</form>
<script language=JavaScript>
document.write("<br> há " +
               history.length +
               " páginas visitadas" );
</script>
```

Observe que os métodos `back()` e `forward()` têm o mesmo efeito que usar os botões `Back` e `Forward` do *browser*. Também são equivalentes a `history.go(-1) e history.go(1).` Eles não terão efeito se já tiver atingido o final da lista.

Exercícios:

1. Digite o código que gerou a página mostrada na figura 15.2 (ele é fornecido antes da figura). Desative neste código a parte em que é pedido para por um novo texto no botão, isto é, *delete* ou transforme em comentário o trecho:

```
texto=prompt("Qual texto deve aparecer"+
" no botão? ");
```

Depois crie uma *âncora* na parte da página onde está o `link`, com o texto "outros links". Isto é, inclua o trecho:

```
<a name="#link">
```

depois da linha:

```
<IMG SRC="0.gif" height=6>
```

Agora faça a página, ao ser carregada, ir imediatamente para esta seção incluindo as linhas:

```
<SCRIPT LANGUAGE="javascript">
window.location.hash="link";</script>
```

na seção `<Head>...</Head>` .

Mostre nesta seção todas as demais propriedades do objeto `location.` Finalmente inclua mais um botão e faça ser carregada uma nova página ao ser pressionado qualquer botão. Em um dos botões faça a ação ocorrer, usando a propriedade `location.href.` No outro botão use o método `replace(url)`. Verifique se o que ocorre quando você pressiona `"back"` do seu navegador ao usar o método `replace()` é idêntico ao que ocorre quando você usa `location.href="url".`

2. Inclua o código do exemplo `history` na página anterior. Verifique o resultado visualizando a página. Veja se seu navegador possibilita que a página tenha acesso às propriedades do objeto `history` do usuário, tentando incluir as linhas:

```
document.write("<br>A página anterior é "
        + toString(history.previous));
```

```
document.write("<br>A próxima é: "
        + toString(history.next));
```

dentro das tags `<script...></script>`

Resumo:

Nesta aula você foi apresentado ao conceito de hierarquia dos objetos da linguagem JavaScript. Conheceu em detalhes as características de duas classes de objetos do navegador: `location` e `history`. Na próxima aula continuaremos neste assunto estudando outra classe de objetos do navegador.

Auto-avaliação:

Você achou complexo este conceito de hierarquia? Então releia-o antes da próxima aula para usar bem os objetos. E, quanto aos exercícios, usou com facilidade os diversos comandos e métodos novos? Nesta etapa do nosso curso, você já deve saber o que fazer, não é mesmo?