# Aula 16: Os Objetos `document` e `navigator`

Nesta aula continuaremos a tratar dos objetos em JavaScript. Veremos agora detalhes sobre os objetos `document` e `navigator`.

**Objetivos:**
Aprender:
- as características e métodos do objeto `document`
- como gerar páginas dinamicamente
- as características e métodos do objeto `navigator`
- como fazer páginas com código dependente do navegador

**Pré-requisitos:**
As aulas 13, 14 e 15 são importantes para esta aula.

## 1. Objeto `Document`

O objeto `Document` serve para definir, consultar ou modificar características do documento atual mostrado pelo navegador. Freqüentemente, este é um dos objetos mais usados na forma *client side* de JavaScript. Como esta forma de JavaScript existe para transformar os documentos HTML estáticos em programas interativos, é através do objeto `Document` que é possível acessar e modificar o conteúdo dos documentos.

Cada objeto `window` tem uma propriedade `document`. Esta propriedade se refere ao objeto `Document` que representa o documento (tipicamente um documento HTML ou XML) mostrado na janela. Este objeto tem propriedades que permitem conhecer informações sobre o documento mostrado no browser: sua última modificação, as cores nas quais ele será mostrado etc. Este objeto também tem métodos que permitem aos programas JavaScript mostrar texto e criar outros novos documentos. Algumas propriedades deste objeto são *arrays* que representam formulários, imagens, *links*, âncoras e *applets* do documento.

XML é a sigla correspondente a Extensible Markup Language.

O *array* `images[]` contém os objetos que representam as imagens do documento. Esses objetos permitem aos programadores da linguagem trocar imagens do documento criando diversos efeitos interessantes. O *array* `forms[]` contém objetos do tipo `Form`, que representam todos os formulários HTML do documento. Cada elemento `<form>` em um documento HTML cria um elemento numerado no *array* `forms[]`. Idem para cada elemento `<applet>` que cria um elemento no *array* `applets[].` O mesmo se aplica às *tags* `<a>`, `<embed>` e `<img>.`

Além da identificação pelo seu número como elemento no *array*, se alguma *tag* tiver a propriedade `NAME`, o valor deste atributo pode ser usado como um nome de uma propriedade do objeto `Document`. Por exemplo, suponha que uma página tenha um formulário com o nome `f1`, definido pelas linhas:

```
<form name="f1">
```

```
<input type="button"value="pressione">
</form>
```

Seu código JavaScript pode se referir ao objeto `form` usando ambas as expressões que seguem:

```
//referencia pela ordem no documento:
document.forms[0]
//referencia pelo nome:
document.f1
```

Como você já deve ter concluído, é bastante útil dar nomes adequados para facilitar a interpretação e a referência a alguns elementos nos códigos JavaScript.

Este objeto tem as propriedades mostradas na tabela 16.1.

**16.1 - Propriedades do objeto `Document`**

| Propriedade | Significado |
|---|---|
| `alinkColor` | Cor do *link* enquanto ativo. Atributo `ALINK` de `<BODY>.` |
| `anchors[]` | *Array* com as âncoras. (`<a name=...>`) . |
| `applets[]` | *Array* de objetos que representam os *applets* Java, um para cada `<applet>` que aparecer no documento |
| `bgColor` | Cor de fundo (definida em `<BODY>).` |
| `cookie` | Permite ler e escrever *cookies* HTTP. Contém uma *string* que é o valor de um *cookie* associado com este documento. |
| `domain` | Permite aos servidores do mesmo domínio relaxar algumas restrições de segurança. Uma *string* especificando o domínio. |
| `embeds[]` | *Array* de objetos embutidos (*plugins* e controles ActiveX) do documento. |
| `fgColor` | Cor do texto *default* (definida no atributo `TEXT` em `<BODY>).` |
| `forms[]` | *Array* com os formulários (*tags* `<form>` ... `</form>` do documento). |
| `images[]` | *Array* com as imagens (*tags* `<img src=...>` do documento). |
| `lastModified` | *String* com a data da última modificação do documento. |
| `links[]` | *Array* com os *links* (`<a href=...>`) do documento. |
| `linkColor` | Cor dos *links* não visitados. Corresponde ao atributo `LINK` de `<BODY>.` |
| `location` | Sinônimo da URL, existe apenas por compatibilidade com a versão 1.0 . |
| `referrer` | *String* de leitura com a URL do documento que chamou a página atual, se houver. |
| `title` | Uma *string* só de leitura com o texto no interior das *tags* `<TITLE>...<TITLE>.` |
| `URL` | Uma *string* só para leitura com a URL da qual o documento foi carregado. |
| `vlinkColor` | Cor dos *links* já visitados. Atributo `VLINK` de `<BODY>`. |

O conjunto de nomes de cores pré-definidos pode ser encontrado em: http://developer.netscape.com/docs/manuals/communicator/jsguide4/colors.htm .

As propriedades de cores do documento devem ser definidas antes da tag `<body>` ser executada. Elas não podem ser definidas em qualquer lugar, mas

podem ser definidas dinamicamente na seção `<head>` ou estaticamente como atributos da tag `<body>` do documento. A única exceção a esta regra é a propriedade `bgColor`, que pode ser definida a qualquer hora. Todas estas propriedades de cores têm como parâmetro uma das formas possíveis de predefinir cores já comentadas.

As linhas de código abaixo, por exemplo, usam esta propriedade para alterar a cor de fundo:

```
<html>
<head>
<title>Exemplo Propriedades de Document
</title>
<script language=javascript>
function dataModif()
{
    document.write("<h3>ultima ",
            "atualizacao nesta pagina",
            document.Modified, "</h3>");
}
function mudaCor(cor)
{
    document.bgColor = cor;
}
</script>
</head>
<body bgcolor=lightpink>
<h1 align=center>Utilizando o Objeto
Document</h1>
<script language =javascript>
dataModif();
</script>
<h2>Mudar a Cor do Fundo</h2>
<a href="javascript:mudaCor('white')">
Branco</a>
<br>
<a href="javascript:mudaCor('green')">
Verde</a>
<br>
<a href="javascript:mudaCor('blue')">
Azul</a>
<br>
<a href="javascript:mudaCor('yellow')">
Amarelo</a>
<br>
<script language=javascript>
if (document.referrer)
  document.write("<h3>Referenciada por:",
                 document.referrer,
                 "</h3>");
</script>
</body>
</html>
```

Diversas outras propriedades que aparecem na tabela 16.1 são mais interessantes que as de cores. Por exemplo, no código anterior nós também usamos `document.referrer` e `document.lastModified`.

Um dos usos possíveis de `referrer` é armazenar os valores do site que referiu a sua página para fins estatísticos. Também poderia ser útil para analisar links não-autorizados que se referem a sua página, ou fazer com que os links que venham de páginas que não tenham passado antes pela sua página principal a visitem primeiro.

Usar `document.lastModified` pode ser muito útil para automaticamente mudar a data de última alteração em documento, o que é um dado bastante útil em uma página. Como a forma do texto que aparece pela impressão direta do valor de retorno desta função inclui `hora:minutos:segundos` (separados por `":"` como mostrado) e pode aparecer como `mês/dia/ano` (isto é a posição usual em nossa língua com o dia e o mês invertida e separados por `"/"` como mostrado), às vezes, pode ser muito mais útil associá-la a um objeto `Date`, para ter as opções dos métodos de manipulação de data deste a nosso dispor. Assim, as linhas de código para uma nova função `dataModif()` abaixo atribuem o retorno de `document.lastModified` a um objeto `Date` (que estudamos na Aula 14) e usam os métodos `getDate()`, `getMoth()` e `getUTCFullYear`(), para imprimir o dia, mês e ano.

Como uma última observação repare que ao mês foi adicionado `1`, pois os valores de retorno de `getMoth()` são entre `0` e `11`:

```
//outra versão da função anterior
function dataModif1()
{ //cria objeto Date
  modif=new Date(document.lastModified);
  //usa alguns metodos de Date
  document.write("<h3>Atualizado em:",
         modif.getDate(), "/",
         (modif.getMonth()+1),"/",
         modif.getUTCFullYear(),"</h3>");
}
```

### 1.1. Documentos Gerados Dinamicamente

Os métodos `clear, write e writeln` do objeto `Document` permitem alterar dinamicamente textos HTML nos documentos que estão sendo interpretados. Os métodos `close` e `open` permitem criar documentos inteiramente novos. A tabela 16.2 mostra os métodos deste objeto.

**16.2. Métodos do objeto do browser `Document`**

| Método | Significado |
|---|---|
| `clear()` | Apaga o conteúdo todo de um documento. |
| `close()` | Fecha o documento aberto por `open()`. Termina de incluir coisas num documento. |
| `open()` | Cria um novo documento. Abre um canal para escrever conteúdos em um documento. |
| `write (v1, … vn)` | Insere *string* ou *strings* `(v1... vn)` no documento cujo canal foi aberto por `open()`. |
| `writeln (v1, … vn)` | Imprime *strings* `(v1... vn)` em uma linha do documento. Idêntico ao anterior, mas adicionando a mudança de linha na saída. |

Já estamos usando as funções de escrita no mesmo documento que está sendo interpretado deste a primeira aula deste módulo. Para criar um novo documento é necessário primeiro usar o método `open()` do objeto `Document`, e então chamar as função de escrita tantas vezes quantas necessárias para gerar o conteúdo do novo documento e mostrá-lo. Finalmente, deve-se usar o método `close()` para indicar que o documento está terminado. Este último passo é importante porque o browser pode armazenar temporariamente o HTML que você está escrevendo e só mostrar esta tela de saída quando o método `close()` for chamado.

De maneira oposta ao fechamento, abertura pelo método `open()` é opcional. Se você usar `write()` em um documento que já foi fechado será implicitamente aberto um novo documento HTML, como se você tivesse usado o método de abertura. Neste processo, no entanto, o documento corrente e seu conteúdo será descartado.

Quando `open()` é usado sem argumentos é aberta uma nova página HTML. Mas os browsers podem mostrar outros tipos de formato de dados além de HTML. O formato mais comum de dados são textos simples (ou do tipo `plain text`). Se você quiser usar `write()` para saída de um texto deve usar a string `"text/plain"` como argumento:

```
document.open("text/plain");
```

Com estes métodos, você gera documentos dinamicamente, mas para que um documento HTML seja realmente dinâmico deve ser interativo, o documento e os seus elementos devem responder a atitudes (eventos) do usuário. Nós discutiremos eventos na próxima aula.

## 2. Objeto `navigador`

As propriedades deste objeto contêm informações apenas para leitura, isto é, não modificáveis, sobre o browser ou navegador em uso. Estas propriedades podem ser usadas para configurar a página para uma plataforma específica. Este nome é obviamente devido ao browser Navigator, embora seja portável e esteja presente nos outros browsers que interpretam JavaScript. O Internet Explorer usa o nome `clientInformation` para se referir ao objeto

`navigator`. Mas este nome embora mais descritivo e menos relacionado a um produto não é portável e nem suportado pelo Netscape Navigator.

A forma como este objeto está implementado difere do Explorer para o Netscape, mas pelo menos as propriedades mostradas na tabela 16.3 são comuns e permitem a criação de documentos adaptáveis a cada tipo de navegador. Como há uma única instância deste objeto não é necessário se referir a ele como `window.navigator`.

**16.3 - Propriedades do objeto** `Document`

| Propriedade | Significado |
|---|---|
| `appName` | O nome do *browser*. |
| `appVersion` | Versão do *browser*, seu número e outras informações. |
| `userAgent` | Todas as informações de `appName` e `appVersion`. |
| `appCodeName` | Codinome do navegador. |
| `platform` | Tipo de máquina onde está sendo executado. |
| `language` | Código em 2 letras especificando a língua que a versão em uso suporta. Inglês =`"en"`, francês =`"fr"` etc. |

As linhas de código que seguem mostram cada uma destas propriedades do objeto Navigator. Mas como você poderá ver se executá-las, algumas delas têm muito mais valores que o realmente necessário.

```
<html>
<head>
<title>O objeto navigator</title>
</head>
<body bgcolor=lightcyan>
<h2>O objeto navigator</h2>
<hr>
<script language="JavaScript">
<!--
if (navigator.javaEnabled())
{
   document.write("browser admite Java);
}
var browser="<br>Todas as propriedades" +
            " do browser:<br>";

for(var propriedades in navigator)
{
  browser += propriedades + ": " +
             navigator[propriedades] +
             "<br>";
}
document.write(browser);
//-->
</script>
<hr>
</body>
</html>
```

Além das propriedades acima, há outras definidas apenas no Netscape Navigator ou no Internet Explorer. A maioria destas propriedades

incompatíveis são principalmente usadas em DHTML. Para usar estas propriedades é melhor você antes verificar o tipo de navegador usado porque vai visualizar a página. A forma de fazer isso é usar as técnicas que discutiremos na próxima seção.

O objeto Navigator também tem métodos usáveis apenas para o Netscape Navigator. A única função que faz parte da linguagem desde sua versão 1.1 é `navigator.javaEnabled()`. No exemplo acima nós a usamos também. A função `javaEnabled()` testa se o *browser* corrente permite o uso de Java. Em caso positivo, o valor de retorno será `true`, e `false` em caso contrário.

### 2.1. Páginas Dependentes do Navegador

Devido às diferenças de implementação entre os principais tipos de navegadores, em muitas situações é necessário distinguir qual o navegador para saber que código utilizar. Uma forma de fazer isso é através do objeto `navigator`. Este objeto, como o próprio nome diz, apresenta uma série de informações acerca do browser que está sendo utilizado.

A técnica de criar código adaptável ao tipo do browser consiste em colocar o código dependente do navegador abaixo de um **desvio condicional,** onde se testa o valor da propriedade `appName` . Como JavaScript é uma linguagem interpretada, o navegador só acusa o erro de não reconhecer um código quando o executa e o desvio condicional evita que isso ocorra. O exemplo abaixo mostra como incluir código dependente do navegador:

```
ns=(navigator.appName.indexOf("Netscape") != -1);
ie=(navigator.appName.indexOf("Explorer") != -1);
document.write("Seu navegador é: ");
if (ns)
  document.write("Netscape");
else if (ie)
  document.write("Explorer");
else
  document.write("desconhecido");
```

### Exercícios:

**1.** Digite o código que está sendo discutido no `Exemplo Propriedades de Document`, na seção 1 desta aula.

Agora faça a página ser carregada na forma original da função `DataModif()` e depois usando a sugestão de modificação apresentada na seção 1: `DataModif1().`

Como você poderia fazer para ao invés de apresentar os meses com números mostrá-los pelos nomes? Faça isso agora, criando uma nova versão deste mesma função: `DataModif2()` (Dica: crie um `Array` com os nomes dos meses do ano e use o valor de retorno de `getMonth` para acessar os elementos deste *array*)

Tente usar agora neste código (como fizemos nos objetos estudados nas aulas 13 e14) todas as demais propriedades do objeto `document` apresentadas na tabela 16.1. Verifique o que ocorre com cada uma delas no documento mostrado pelo seu navegador.

**2.** Digite o código que foi apresentado na seção 2 e veja o resultado em uma página Web. Depois substitua o código abaixo pelo correspondente mais completo do exemplo apresentado na seção sobre o objeto `navigator`. Verifique o resultado visualizando novamente a página.

```
document.write("<hr><br>Agora as mais " +
               "importantes:<UL>");
document.write("<LI><B>Code Name:</B> ",
               navigator.appCodeName);
document.write("<LI><B>App Name:</B> ",
               navigator.appName);
document.write("<LI><b>App Version:</b>",
               navigator.appVersion);
document.write("<LI><B>User Agent:</B> ",
               navigator.userAgent);
document.write("<LI><B>Platform:</B>",
               navigator.platform);
nome = navigator.appName;
if (nome.indexOf("Netscape") != -1)
   document.write("<LI><B>Language:</B>",
                  navigator.language);
else
   document.write("<LI><B>Language:</B>",
                 navigator.userLanguage);
```

**3.** Utilize o exemplo dependente do navegador na seção 2.1 no exercício anterior. Visualize o resultado que terá abrindo uma página com o código em um navegador.

## Resumo:

Nesta aula você conheceu em detalhes as características de dois objetos muito importantes: `Document` e `Navigator`.
O primeiro deles é especialmente útil para criar páginas com maior dinamismo e que podem ser geradas automaticamente Na próxima aula continuaremos neste assunto de geração de páginas não estáticas, aprendendo como fazê-las interagir com o usuário, o que permitirá a criação de páginas muito versáteis e úteis.

## Auto-avaliação:

Quantas propriedades e métodos novos! Se algum ponto não ficou muito claro, releia-o antes da próxima aula, na qual você continuará a crescer na direção de ser um criador de páginas WEBs poderosas e repletas de objetivos! Nela você aprenderá as coisas mais importantes para uma programação dinâmica: como

reagir às atitudes de quem usa a página , ou em outras palavras como usar os eventos.