Desenvolvimento Web I

Aula 10 - JavaScript: Acessando Objetos - Parte 1

Apresentação

Na aula passada, você começou a aprender uma maneira de tornar suas páginas Web mais interativas e, consequentemente, mais interessantes para os usuários finais que a acessarem. Isso se dá por meio da programação em JavaScript. Nesta aula, você vai aprender aspectos mais avançados do uso da linguagem JavaScript em páginas Web, em particular como trabalhar com objetos e como manipular objetos que representam a página Web e seus elementos (tags), o navegador, as janelas, etc. Bons estudos!

Vídeo 01 - Apresentação

Objetivos

- Manipular objetos da biblioteca de JavaScript.
- Definir e usar seus próprios objetos.

Uso de Objetos em JavaScript

JavaScript não é uma linguagem completamente orientada a objetos, mas ainda assim possui objetos. Apesar disso, ela não implementa alguns conceitos típicos de linguagens dessa natureza, tais como herança, interfaces, visibilidade etc. Por esse motivo, podemos dizer que ela é baseada em objetos e não orientada a objetos. Além disso, iremos perceber mais adiante que a implementação do conceito de objetos em JavaScript é algo extremamente simplificado, bem diferente de Java.

Além da possibilidade de se criar novos tipos de objetos e manipulá-los através da programação, também é possível acessar todos os elementos HTML presentes em uma página Web, através de JavaScript. Em outras palavras, quando o browser carrega uma página Web, ele cria uma série de objetos representando os diversos elementos da página (imagens, formulários, botões etc.). Esses objetos, por sua vez, podem ser acessados e modificados via JavaScript, mesmo após a página ter sido carregada completamente.

Vídeo 02 - JavaScript: Acessando Objetos (Parte 1)

Definindo e manipulando objetos

Assim como em Java, um objeto é criado no JavaScript a partir da definição de uma estrutura. Essa estrutura em Java é chamada de classe, mas, em JavaScript, ela se resume a um construtor. Um construtor nada mais é que uma função. O exemplo abaixo demonstra a definição de um construtor e, logo em seguida, como uma instância de um objeto desse tipo é criada através da palavra-chave **new**:

```
1 function Livro() {
2    // Pode definir ações que devem ser executadas ao se instanciar um objeto do tipo Livro
3 }
4 var meuLivro = new Livro();
```

Um objeto pode possuir atributos e métodos, como veremos a seguir.

Atributos

Ao se definirem objetos, é natural pensar em como definir os seus atributos. Em JavaScript, esses atributos são definidos dentro do próprio construtor do objeto, já que não existem classes. O trecho de código abaixo demonstra a definição de dois atributos para objetos do tipo Livro.

```
function Livro() {
  this.titulo = "Harry Potter";
  this.autor = "J.K. Rowling";
}
var meuLivro = new Livro();
alert(meuLivro.titulo);
```

O exemplo acima é bem limitado, visto que ele só irá permitir a criação de objetos sempre com o mesmo título e autor. Na prática, seria interessante que o título e autor pudessem ser passados como parâmetros. O trecho de código abaixo demonstra essa possibilidade, em um exemplo bem mais real, em que o construtor é parametrizado e, dessa forma, diversos objetos podem ser criados com títulos e autores diferentes.

```
function Livro(tituloPar, autorPar) {
  this.titulo = tituloPar;
  this.autor = autorPar;
}
var meuLivro = new Livro("Harry Potter","J.K. Rowling");
var meuLivro2 = new Livro("As Crônicas de Nárnia","C.S. Lewis");
alert(meuLivro.titulo);
alert(meuLivro2.titulo);
```

Atividade 01

1. Defina classes para representar os conceitos existentes em uma Livraria, em particular. Crie classes para cada um desses conceitos: Livro, Autor, Gênero. Pense nos atributos de cada um e na relação entre eles.

2. Desenvolva um script que, a partir da função prompt, cria um livro (com todas as informações relacionadas) e depois exibe as informações do livro (dados digitados pelo usuários), usando a função alert.

Métodos

Assim como atributos, objetos também podem possuir métodos que representam as ações as quais podem ser executadas sob o objeto. Para criarmos métodos, devemos fazer o mesmo que foi feito para se definir atributos. O trecho de código a seguir demonstra a definição de um novo método chamado getDescricao(), que, ao ser chamado, retorna um string, resultado da concatenação do título e autor do livro.

```
function Livro(tituloPar,autorPar) {
  this.titulo = tituloPar;
  this.autor = autorPar;
  this.getDescricao = function(){ return this.titulo +" - "+this.autor}
}

var meuLivro = new Livro("Harry Potter","J.K. Rowling");
var meuLivro2 = new Livro("As Crônicas de Nárnia","C.S. Lewis");
alert(meuLivro.getDescricao());
alert(meuLivro2.getDescricao());
```

Apesar do exemplo não ter demonstrado isso, os métodos podem receber parâmetros e também podem alterar os atributos do objeto no qual ele foi chamado. O trecho de código que segue demonstra um novo método chamado adicionarAutor() que, além de receber um parâmetro, altera o valor do atributo autor.

```
function Livro(tituloPar,autorPar) {
  this.titulo = tituloPar;
  this.autor = autorPar;
  this.getDescricao = function(){ return this.titulo +" - "+this.autor}
  this.adicionarAutor = function(novoAutor){ this.autor = this.autor + ","+novoAutor}
}

var meuLivro = new Livro("Harry Potter","J.K. Rowling");
meuLivro.adicionarAutor("XXXX");
alert(meuLivro.getDescricao());
```

Atividade 02

1. Crie o método de validar, que é responsável por verificar se todos os atributos foram informados (se são diferentes de null).
2. Crie o método temMesmoAutor() que recebe um livro como parâmetro e verifica se ele tem a mesma autoria, retornando true caso positivo e false caso contrário.
3. Crie o atributo ano e o método mesmoAnoPublicacao(), que recebe um livro como parâmetro e que verifica se o livro foi publicado no mesmo ano. Retorne true, em caso positivo, e false, caso contrário.

Trabalhando com vetores

Assim como Java e boa parte das linguagens de programação, JavaScript permite a manipulação de variáveis como vetores ou matrizes. Um vetor é usado em situações quando precisamos que uma única variável possa guardar vários valores diferentes. Além disso, vetores e matrizes permitem que a manipulação desse conjunto de valores possa ser feita de maneira simples.

Em JavaScript, um vetor é criado de forma similar a um objeto, usando a estrutura Array, conforme pode ser visto no exemplo a seguir. Nesse exemplo, três vetores são criados, ilustrando as três formas possíveis de se criar vetores. Perceba que, assim como qualquer variável, o tipo de dados que será armazenado no vetor não é definido e não é necessário que os elementos de um vetor sejam do mesmo tipo.

Vídeo 03 - Vetores

```
//declaração regular de um vetor
var titulos = new Array();

//declaração condensada do vetor
var codigos = new Array("65454", "12312");

// declaração literal
var codigos2 = ["um", "dois", "três"];
```

O trecho em seguida demonstra como é feita a manipulação dos valores do vetor. Inicialmente, são atribuídos valores a cada uma das posições do vetor e depois essas informações são acessadas e impressas na página. Por fim, é impressa a informação da quantidade de títulos através do acesso à propriedade length, que informa o tamanho do vetor.

```
titulos[0] = "As crônicas de Nárnia";
titulos[1] = "Contato";
for(i=0;i<2;i++){
  document.writeln(codigos[i] + " --- " + titulos[i]);
}
document.write("Total de Títulos: "+titulos.length)
```

Biblioteca de objetos de JavaScript

Com JavaScript, além de definir e criar novos objetos, como mostrado na Seção 2, é possível acessar e manipular outros objetos que estão acessíveis durante a execução do JavaScript, contido em uma página Web. São diversos objetos disponíveis, desde funções básicas, como manipulação de string ou funções matemáticas, até objetos que representam os elementos (tags) contidos em uma página Web. De um modo geral, esses objetos podem ser divididos em três grupos, são eles:

- **Objetos DOM** — objetos que representam os elementos HTML contidos na página Web;
- **Objetos JavaScript** — objetos que possuem funções de caráter utilitário, tais como funções matemáticas (seno, cosseno, máximo, mínimo, potencial etc.), manipulação de texto (substrings, busca em texto etc.), entre outras;

- **Objetos do Browser** — acesso a objetos que representam as ferramentas do ambiente no qual a página Web está executando, em particular o navegador. Com esses objetos, é possível abrir novas janelas, acessar histórico de navegação, acessar informações do navegador ou do próprio servidor.

Cada um desses grupos e os objetos pertencentes aos mesmos serão apresentados em mais detalhes nas seções seguintes.

Objetos DOM

Nesse contexto, é possível manipular os objetos que representam a página Web e que foram criados pelo navegador no momento em que a página foi carregada, além de ser possível acessar os elementos da página, como imagens e inputs, e alterar propriedades dos mesmos após a ela ter sido carregada. Isso é muito útil na criação de página mais dinâmicas onde mesmo após terem seu conteúdo criado no servidor esse pode ser alterado no navegador depois de terem sido exibidos.

Para cada elemento (tag) presente numa página Web, um objeto correspondente é criado pelo navegador de forma a poder ser acessado via JavaScript. Esses objetos possuem propriedades e métodos que podem ser acessados e executados, respectivamente, através de JavaScript. Esse modelo de objetos (tipos de objetos, atributos, métodos e relacionamento entre eles) é padronizado pelo W3C, sendo conhecido como HTML DOM (Document Object Model). A especificação completa desse modelo de objetos pode ser vista em <http://www.w3schools.com/jsref/default.asp>. Acesso em: 19 jan. 2015.

Objeto Document

Para cada página HTML que é carregada no navegador, é criado um objeto document. Através desse objeto, é possível acessar todos os elementos HTML (tags) definidos na página com o uso de JavaScript. O quadro a seguir apresenta os principais atributos que podem ser acessados a partir desse objeto.

Atributo	Descrição
anchors[]	Vetor contendo todas as âncoras HTML (tag <a name=”...” />)
forms[]	Vetor contendo todos os formulários HTML definidos na página (tags <form>)
images[]	Vetor contendo todas as imagens (tags <img>)
links[]	Vetor contendo todos os links definidos na página (tag <a href=”...” />)
domain	Domínio do servidor da página carregada
title	Título da página (tag <title>)
URL	URL completa do document

Quadro 1 - Atributos do objeto document

No exemplo a seguir, são definidas duas imagens na página HTML e, no código JavaScript, é escrito na página a quantidade de imagens existentes. Essa contagem das imagens é feita através do acesso ao vetor de imagens (**document.images**).

Vídeo 04 - Objeto Document

```
<html>
   <body>
      <img border="0" src="image1.jpg" width="150" height="113" />
      <img border="0" src="image2.jpg" width="152" height="128" />

      <p>Número de imagens:
        <script type="text/javascript">
          var numeroImagens = document.images.length;
          document.write(numeroImagens);
        </script>
      </p>
   </body>
</html>
```

Além disso, o próximo código HTML que você verá apresenta um exemplo bem interessante, que aumenta o tamanho de uma figura sempre que o usuário passar o mouse por cima dela e retorna ela ao tamanho original ao mover o mouse para fora. Nesse exemplo, iremos combinar tratamento de eventos (visto na primeira aula sobre JavaScript) com a biblioteca DOM:

```
<html>
   <head>
   <script type="text/javascript">
      function aumentar(posicao){
         document.images[posicao].width=300;
         document.images[posicao].height=300;
      }
      function diminuir(posicao){
         document.images[posicao].width=150;
         document.images[posicao].height=150;
      }
   </script>
   </head>
   <body>
      <img id="imagem1" border="0" src="img1.png" width="150" height="150" onmouseover="aum
      <img id="imagem2" border="0" src="img2.png" width="150" height="150" onmouseover="aum
   </body>
</html>
```

Como pode ser visto, duas funções são definidas (aumentar e diminuir), as quais recebem como parâmetro uma variável que representa a posição da imagem no vetor de imagens do documento. Lembre-se de que as imagens são armazenadas no vetor de imagens de acordo com a ordem como elas foram definidas na página HTML. Essas funções aumentam a largura e altura (propriedades **width** e **height**, respectivamente) da imagem de índice passado como parâmetro.

Por fim, pode-se verificar que, na declaração das imagens, é definido o tratamento para os eventos **MouseOver** e **MouseOut**, que representam os tratamentos do evento de pôr e de tirar o mouse de cima da figura, respectivamente. Perceba que a imagem com **id imagem1** é referenciada através do parâmetro 0 (representa a posição 0 do vetor) e a **imagem2** é referenciada com o parâmetro 1 (representa a posição 1 do vetor).

Assim como as imagens da página podem ser acessadas e suas propriedades alteradas, é possível fazer o mesmo para os links, âncoras e formulários, somente mudando o atributo do objeto document a ser modificado. Além dos atributos, o objeto document possui alguns métodos, como mostrado na Quadro 2.

Métodos	Descrição
getElementById()	Retorna um elemento a partir de seu ID.
getElementsByName()	Retorna todos os elementos com determinado nome.
getElementsByTagName()	Retorna todos os elementos de determinada tag.
write()	Escreve texto HTML no ponto onde o comando está inserido.
writeln()	Escreve texto HTML no ponto onde o comando está inserido, adicionando-se uma quebra de linha no final.

Quadro 2 - Outros métodos do objeto document

Os três primeiros métodos são métodos que permitem acessar qualquer elemento da página HTML, diferentemente dos atributos, que só permitem o acesso a elementos do tipo imagem, link, âncora ou formulário. Com esses métodos, é possível ter acesso a qualquer elemento da página, desde elementos de título (**<h1>, <h2>, <h3>**, etc.) até elementos de formulários (**<select>, <input>, <textarea>,** etc.). O trecho de código abaixo exemplifica o uso do método **getElementById()**.

```
<html>
  <head>
    <script type="text/javascript">
      var contador = 0;
      function getValue() {
        var cabecalho1 = document.getElementById("header1");
        var cabecalho2 = document.getElementById("header2");
        alert(cabecalho1.innerHTML);
        contador++;
        cabecalho2.innerHTML= "Cliques: " + contador;
      }
    </script>
  </head>
  <body>
    <h1 id="header1" onclick="getValue()">Clique Aqui!</h1>
    <h2 id="header2"></h2>
  </body>
</html>
```

Como pode ser visto, o método **getElementById()** recebe como parâmetro um id e retorna o elemento HTML com esse id. No exemplo mostrado, ele é usado para acessar dois elementos HTML (**<h1>,** com id **header1** e **<h2>**, com id **header2**). Esses dois elementos são guardados nas variáveis **cabecalho1** e **cabecalho2**, respectivamente. A partir daí, é possível acessar e modificar qualquer atributo desses elementos, inclusive o conteúdo que está contido no mesmo. Nesse exemplo, usamos o atributo innerHTML, que guarda o conteúdo definido dentro da tag HTML. Por exemplo, no caso de <h1>Teste, o atributo innerHTML vai ter valor igual a “Teste”.

Os métodos **getElementByName()** e **getElementByTagName()** são similares ao **getElementById()**, a diferença é que eles podem retornar vários elementos, já que podem ter vários elementos do mesmo tipo de tag (, etc.) ou até com o mesmo nome (atributo **name** das tags HTML). Apesar de também ser possível (ainda que não recomendado) ter diferentes elementos com o mesmo id em uma página HTML. No caso do método getElementById, ele sempre retorna somente um elemento, o primeiro que ele encontrar.

Por fim, os métodos **write()** e **writeln()**, que já foram usados em exemplos anteriores, servem para escrever algum conteúdo no ponto da página onde ele é executado. A diferença entre eles é basicamente que o **writeln()** adiciona uma quebra de linha ao final do texto escrito.

Objetos HTML

Existem diferentes formas de acessar os objetos referentes às tags HTML definidas na página. Isso pode ser feito através do uso dos vetores presentes como atributos do objeto **document** ou através dos métodos do objeto **document**. Independente da forma como o objeto é acessado, após recuperá-lo, é possível acessar seus atributos e invocar seus métodos. Existem diversos atributos e métodos em qualquer objeto HTML, a lista completa pode ser vista em <http://www.w3schools.com/jsref/dom_obj_all.asp>. Acesso em: 19 jan. 2015.

Além disso, também existem métodos e atributos que são específicos de determinados tipos de elementos. De um modo geral, qualquer atributo que for possível ser definido na tag HTML pode ser acessado e alterado através de JavaScript. Nesta aula, iremos mostrar somente alguns exemplos de manipulação desses tipos de objetos. Para saber os atributos e métodos presentes em todos os objetos, consulte <http://www.w3schools.com/jsref/>. Acesso em: 19 jan. 2015.

Link

O exemplo a seguir demonstra como é possível alterar o endereço pra onde o link está apontando:

```
<html>
   <head>
      <script type="text/javascript">
         function mudarendereco(){
           var link = document.getElementById("x");
           link.href="http://www.google.com";
         }
      </script>
   </head>
   <body>
      <a id="x" href="http://www.hotmail.com" onclick="mudarendereco()">
         Mudar endereço
      </a>
   </body>
</html>
```

Neste exemplo, ao invés do usuário ser redirecionado para o Hotmail, como está definido no atributo **href** do link, ele será enviado para o google. Isso acontece porque o tratamento do evento **onclick** é executado antes que o navegador redirecione o usuário; mas, como a função que trata o evento **onclick** altera o endereço, então, o navegador vai redirecionar o usuário para o novo valor do atributo **href**, ou seja, para o site do google.

Input Text

O exemplo abaixo mostra um campo de texto que aumenta de tamanho à medida que o usuário vai digitando informação nele.

```
<html>
  <head>
    <script type="text/javascript">
      function aumentar()  {
        var input = document.getElementById("textoCrescente");

        //tamanho do campo
        var tamanhoCampo = input.size;

        var texto = input.value;

        //tamanho de um string
        var tamanhoTexto = texto.length;


        //Para ficar sempre com um espaço a mais
        if(tamanhoTexto - 1 >= tamanhoCampo){
          input.size = tamanhoCampo + 1;
        }
      }
    </script>
  </head>
  <body>
    <h2>Digite algo para aumentar:</h2>
    <br/>
    <form>
    <input id="textoCrescente" type="text"  onkeypress="aumentar()" size=3="" />
    </form>
  </body>
</html>
```

Form

O objeto **Form** representa um formulário em HTML, mais especificamente um elemento HTML definido pela tag **<form>**. Os formulários são usados para coletar dados digitados pelo usuário e enviá-los para o servidor. Através de JavaScript, além de ser possível manipular as informações digitadas dentro do formulário, é possível controlar as ações executadas pelo formulário. Por exemplo, é possível determinar o momento quando os dados do formulário serão enviados ao servidor ou mesmo apagar todos os dados dos campos contidos no formulário.

O exemplo a seguir representa uma situação muito comum em sistemas Web. Nesse exemplo, é definido um formulário contendo dois campos de entrada de texto, um para digitação do login e outro para digitação da senha. Normalmente, para submeter os dados dos campos contidos em um formulário para o servidor Web é usado um elemento chamado "**Submit**", que nada mais é do que um tipo de input HTML (**<input type="submit" value="Enviar dados"/>**).

Esse elemento, ao ser clicado, tem como comportamento padrão submeter o formulário ao servidor (consequentemente os seus dados). Mas, como pode ser visto no trecho de código abaixo, ao invés de usarmos um "Submit", usamos uma imagem.

Apesar de parecer estranho, esse tipo de substituição é muito comum, visto que o componente "**Submit**" nada mais é do que um botão padrão e não possui uma estética de visual muito refinada. Sites modernos normalmente usam figuras feitas sob medida para representar os seus botões nos estados de clicado e não clicado.

O problema dessa abordagem é que uma imagem não possui a mesma funcionalidade de um botão "**Submit**", que é a de enviar os dados do formulário ao servidor. Então, como acrescentar essa funcionalidade a uma imagem? A resposta você já deve saber: através de JavaScript.

A ideia básica dessa solução é: atribui-se uma ação (função JavaScript) ao evento onclick na imagem e essa função, entre outras coisas, faz a submissão do formulário ao servidor. Veja o código do exemplo para entender essa solução.

```
<html>
<head>
   <script type="text/javascript">
      function logar(){
         var loginInput = document.getElementById("login");
         var senhaInput = document.getElementById("senha");
         var formulario = document.getElementById("formulario")

         if(loginInput.value == ""){
            alert("Digite o login!");
         }
         else if(senhaInput.value == ""){
            alert("Digite a senha!");
         }
         else{
            formulario.submit();
         }
      }
   </script>
</head>
   <body>
      <form id="formulario" action="servlet/LogarServlet">
        Login: <input type="text" id="login" name="login" /><br />
        Senha: <input type="password" id="senha" name="senha" /><br />
        <img src="http://via.placeholder.com/100x40/6666ff/000000/&text=Entrar" onclick="logar()"/>
      </form>
   </body>
</html>
```

Como pode ser visto, a função logar(), que é chamada ao se clicar na imagem, primeiramente valida se o usuário digitou o login e a senha (avaliando se o valor contido nos campos do formulário é vazio) e em caso positivo submete o formulário (chamada ao método submit() no objeto formulário). Repare que a imagem que utilizamos tem o src=”http://via.placeholder.com/100x40/6666ff/000000/&text=Entrar”. Nessa imagem estamos utilizando um serviço bastante interessante do site placeholder.com, que permite que você crie e utilize imagens de testes dinamicamente. Nesse caso estamos solicitando uma imagem com o tamanho 100x40 pixels, com a cor de fundo #6666ff (que é um tom de azul) com a cor do texto #000000 (cor preta) e com o texto “Entrar” escrito no meio da imagem. Colocando essa URL no src da tag IMG faz com que o site placeholder.com gere essa imagem para nós e a retorne para a nossa tag. É bem útil para criar imagens temporárias de teste sem precisar utilizar nenhum programa de edição gráfica nem salvar as imagens em disco, entretanto é recomendado que você crie suas próprias imagens do seu site ou sistema para o coloca em produção. Legal né?

Chegamos ao fim da nossa segunda aula sobre JavaScript. Na próxima aula, você estudará os avanços recentes na programação do lado do cliente. Esses avanços tiveram como objetivo melhorar a interatividade de páginas Web. Atualmente, os bons sites não abrem mão desse novo modelo de programação.

Na próxima aula, iremos continuar vendo como manipular os objetos criados pelo browser, com exemplos mais completos e que combinam muitos conhecimentos já adquiridos até aqui.

Resumo

Nesta aula, você aprendeu como definir e criar objetos que você deseja em JavaScript, permitindo assim uma aproximação da programação orientada a objetos. Você aprendeu também que, além dos objetos criados por você, é possível manipular (através de JavaScript) vários outros objetos que são criados pelo navegador ao carregar sua página Web. Ainda nesta aula, você viu como manipular alguns dos objetos do HTML.

Autoavaliação

1. Explique como podemos definir novos objetos, com suas propriedades e métodos em JavaScript.
2. Cite objetos existentes da biblioteca de JavaScript e como podemos acessá-los, como podemos acessar seus atributos, métodos, etc.

Referências

JAVASCRIPT and HTML DOM reference. Disponível em: <http://www.w3schools.com/jsref/>. Acesso em: 23 ago. 2012.

W3C. Disponível em: <http://www.w3.org/>. Acesso em: 23 ago. 2012.

W3SCHOOL. Disponível: <http://www.w3schools.com/>. Acesso em: 23 ago. 2012.