Aprendendo JavaScript

Filipe Del Nero Grillo
Renata Pontin de Mattos Fortes

São Carlos 21 de fevereiro de 2008

Sumário

1 Introdução

– Por que aprender JavaScript?

Se buscarmos na internet por respostas a esta pergunta, encontramos:

- "JavaScript é a ferramenta que dá acesso a um grande número de truques e de funcionalidades avançadas que estão ao alcance de todos".
- "JavaScript é usada em milhões de páginas Web com intuito de melhorar todo projeto"
- "Com JavaScript você pode deixar a sua página muito mais legal!"

De fato, concordamos que essas respostas falam verdade! Mas essa fama de ser uma linguagem 'leve', para fazer truques em páginas, faz com que muitas pessoas pensem que JavaScript não é uma linguagem de programação séria. Muitos lembram dela como se fosse apenas uma extensão do HTML. A sigla DHTML (*Dynamic HTML*) é uma das causadoras dessa impressão, porque sugere que é um novo tipo de HTML e não que é usado JavaScript e HTML juntos.

Na verdade, JavaScript é uma linguagem de programação de propósito geral, dinâmica e possui características do paradigma de orientação a objetos. Ela é capaz de realizar virtualmente qualquer tipo de aplicação, e rodará no *browser* do cliente.

Atualmente, uma empresa que utiliza JavaScript em praticamente todos os seus aplicativos é a Google Inc., como por exemplo: no site de busca Google Search[1], no GMail[2], no Google Maps[3], entre outros. O que torna esses aplicativos tão populares é a forma como são interativos, e isso se deve em maior parte ao JavaScript.

A linguagem JavaScript foi objeto de estudo do projeto de iniciação científica deste autor, e foi utilizada no desenvolvimento de um editor de diagramas online, via Web, utilizando SVG (*Scalable Vector Graphics*) e JavaScript.

Como resultado do estudo realizado, foi elaborado este documento visando servir de base inicial para que outros interessados possam ter um roteiro prático para prosseguir e se aprofundar em seus estudos.

[1] www.google.com
[2] www.gmail.com
[3] www.google.com/maps

2 Histórico de JavaScript

A linguagem JavaScript foi criada pela Netscape Communications Corporation[4] e foi desenvolvida com o nome de *Mocha*, depois passou a se chamar *LiveScript* e foi finalmente lançada como *JavaScript* em 1995 integrando a versão 2.0B3 do navegador Netscape e visava implementar uma tecnologia de processamento modo cliente.

A denominação da linguagem, *JavaScript*, se deve a similaridades com a sintaxe do Java e embora as duas linguagens não tenham nenhuma outra relação além desta, os nomes ainda causam confusão para alguns usuários.

Mais tarde, a linguagem tornou-se um padrão da ECMA (*European Computer Manufacturers Association*) que atualmente é seguido por outros desenvolvedores como os da Adobe com a linguagem *ActionScript*. A implementação da Microsoft do ECMAScript é chamada de JScript ou ActiveScript, popularmente conhecido como ActiveX, usada nos navegadores Internet Explorer.

JavaScript permite criar pequenos programas embutidos no próprio código de uma página HTML e capazes de gerar números, processar alguns dados, verificar formulários, alterar valor de elementos HTML e criar elementos HTML. Tudo isso diretamente no computador cliente, evitando a troca de informações com o servidor e o tempo passa a depender somente do processamento local do cliente, não mais da latência da rede[5].

A versão mais recente da linguagem é a 1.7, até o momento de redação deste documento. Esta versão 1.7 é suportada pela versão 2.0 do Mozilla Firefox, mas versões anteriores continuam funcionando. Além disso, outras versões de testes já estão sendo desenvolvidas, por exemplo, a versão, chamada de JavaScript 2.0, desenvolvida pela Mozilla que pode vir a se tornar padrão.

JavaScript é uma linguagem completa e poderosa que possui muitas das qualidades de diversas outras linguagens, como: listas associativas, tipagem dinâmica[6] e expressões regulares de Perl e a sintaxe similar a C/C++, linguagens de grande reconhecimento tanto no mundo acadêmico quanto comercialmente. Além disso, JavaScript é multiparadigma e entre eles destacam-se a programação estrutural e orientada a objeto; possui funções de ordem superior; entre outros.

Mesmo com o alto potencial de recursos para desenvolvimento de programas oferecido pela linguagem, grande parte das pessoas que usam JavaScript

[4] Netscape foi uma companhia de serviços de computação bastante conhecida em função de seu *web browser*, e atualmente é subsidiária da AOL.

[5] Latência de rede é a diferença de tempo do momento de envio de um pacote pela rede até o momento em que ele é recebido no destino sendo normalmente este tempo o responsável pela demora ao se receber dados durante uma navegação na web por exemplo.

[6] Também conhecida de forma bem humorada como *Duck Typing* em referência ao *Duck test* - "Se se move como um pato e faz barulho de pato, então deve ser um pato!".

não são programadores e isso lhe deu a reputação de ser uma linguagem para amadores, o que não é verdade, como pode ser observado com o surgimento de técnicas Ajax, que será explicado neste documento, e poderosas aplicações web. Além disso, JavaScript se tornou o carro chefe da chamada *Web 2.0* que também será explicada mais adiante.

3 Conceitos

Nesta seção, são descritos alguns conceitos que estão bastante relacionados com a linguagem JavaScript. Esses conceitos possibilitam caracterizar melhor a linguagem e seu potencial de utilização.

Linguagem compilada vs. Linguagem script - de maneira simplificada, podemos explicar esta diferença de linguagens, considerando-se os procedimentos necessários para que os programas codificados nas mesmas possam ser preparados para que se tornem executáveis em máquinas.

Assim, primeiro, observamos que para se executar um programa em uma linguagem **compilada** é necessário escrever o seu código-fonte correspondente e compilá-lo; durante a compilação, o código-fonte é lido pelo compilador que gera então um arquivo de saída com uma tradução daquele código-fonte para linguagem de máquina (o código executável); esse arquivo em linguagem de máquina pode ser então executado no computador e não pode ser facilmente editado, pois não é compreensível por nós seres humanos. Assim, se desejarmos alterar alguma parte desse programa, precisaremos alterar seu código-fonte e compilá-lo novamente para que o executável novo seja gerado.

Diferente disso, para executar um programa em uma linguagem interpretada (**script**) precisamos apenas digitar o código-fonte e o interpretador irá ler esse código e executar as instruções, comando por comando, a partir do próprio texto do código-fonte, cada vez que o script for rodado; assim, não é necessário a criação de um arquivo estático, para alterar o programa basta alterar o código e ele já estará pronto para rodar novamente. Podemos ilustrar isso com um script de cinema, que de forma análoga, diz passo a passo as ações que os atores devem realizar em um filme. Assim, o roteiro descrito no script de cinema possui uma dinâmica que deve se adaptar a cada necessidade das cenas.

A partir dessa noção, uma vantagem óbvia do script é a agilidade para se alterar o programa, eliminando a sequência editar-compilar-linkar-rodar comum em softwares compilados. Como preço dessa flexibilidade, perde-se um pouco em desempenho e será sempre necessário possuir um interpretador no computador onde será rodado o script, enquanto o arquivo binário resultante de compilação não necessita do compilador em seu ambiente de uso.

JavaScript é uma linguagem de script.

Tipagem Dinâmica - também conhecida como *Duck Typing*, indica como os ti-

pos das variáveis são definidos. Nesse modo de tipagem, as variáveis podem assumir qualquer tipo ou objeto definido pela linguagem, por exemplo, se uma variável X receber um valor inteiro ela irá se comportar como uma variável inteira, e se mais tarde X receber uma string passará a se comportar como uma string daquele ponto em diante. Assim, não é preciso definir o tipo da variável no momento da sua declaração. O tipo da variável é definido implicitamente pelo seu valor.

O contrário disso ocorre na tipagem estática, em que o tipo da variável precisa ser definido antes de usá-la e isso é verificado em tempo de compilação. Após declarada com seu tipo, nenhum valor de outro tipo pode ser atribuído a ela, pois geraria um erro de compilação.

JavaScript é uma linguagem de programação de tipagem dinâmica.

Funções de ordem superior - são funções que recebem uma ou mais funções como argumentos ou que têm uma função como saída. Com isso, é possível criar o que são chamadas *function factories* que são funções que a partir de outras funções simples são capazes de realizar ações mais complexas.

JavaScript é uma linguagem de programação que possibilita a definição de funções de ordem superior.

Programação *Client-side* vs. *Server-side* - JavaScript é uma linguagem que nasceu como *Client-side* (que roda no computador cliente) e tem sido muito mais usada essa forma atualmente. Quando o programa é criado com esta característica ele é enviado para o computador cliente ainda na forma de código-fonte, que só então é interpretado e executado, dependendo assim unicamente da capacidade de processamento do cliente.

Já o programa em uma linguagem *Server-side*, é executado no computador Servidor e somente é enviado para o cliente o resultado da execução, sejam dados puros ou uma página HTML.

Neste estudo, tratamos JavaScript apenas como uma linguagem de programação *Client-side*.

Segurança - por ser uma linguagem que é executada no computador do cliente, o JavaScript precisa ter severas restrições para evitar que se façam códigos maliciosos que possam causar danos ao usuário.

As principais limitações do JavaScript para garantia de segurança são a proibição de:

- Abrir e ler arquivos diretamente da máquina do usuario
- Criar arquivos no computador do usuário (exceto *cookies*)

- Ler configurações do sistema do usuario
- Acessar o hardware do cliente
- Iniciar outros programas
- Modificar o valor de um campo de formulário do tipo `<input>`

No entanto, essas limitações interferem muito pouco no desenvolvimento de aplicações web sérias com a linguagem.

4 Núcleo da Linguagem JavaScript

Nesta seção estão descritos os conceitos básicos para se programar em JavaScript. Quem já possui algum conhecimento com outras linguagens de programação sabe que os comandos básicos compõem a "caixa de ferramentas" que deve ser utilizada para criar qualquer aplicação, seja ela muito pequena ou gigantesca.

A aplicação em JavaScript sempre será composta desses comandos e elementos menores, e a lógica com que foram colocados juntos para se relacionarem possibilita a criação de uma solução visando o resultado esperado.

4.1 Tipos de dados

4.1.1 Tipos numéricos

Em JavaScript os números são representados pelo padrão IEEE 754. Todos os valores numéricos são "declarados" pela simples atribuição dos valores a uma variável.
Exemplos:

Inteiros

```
var x = 35; //atribuição na forma comum
var x = 0543; //notação octal que equivale a 357
var x = 0xBF; //notação hexadecimal que equivale a 191
```

Ponto flutuante

```
var x = 12,3; //declarado na forma comum
var x = 4,238e2; //declarado como potência de 10 que equivale a 423,8
```

4.1.2 Booleano

Uma variável do tipo booleano pode assumir apenas dois valores: *true* e *false*. Os valores deste tipo são em geral usados pela linguagem como resultado de comparações e podem ser usados pelo usuário para valores de teste ou para atributos que possuam apenas dois estados. Equivale ao uso de um inteiro com valores 0 ou 1 na linguagem C.

O JavaScript converte automaticamente *true* para 1 e *false* para 0 quando isso for necessário.

Exemplo:

```
var a = 14;
var b = 42;
var tr = (a == 14);
var fl = (a == b);
// Neste caso tr irá conter o valor true e fl o valor false.
var int1 = tr+1;
var int2 = fl+1;
// A variável int1 irá conter o valor 2 (true + 1), pois true é
// automaticamente convertido para 1 e int2 irá conter o valor 1
// (false + 1), pois false é convertido para 0.
```

4.1.3 Indefinido

Uma variável é indefinida quando ela foi declarada de alguma forma mas não possui nenhum valor concreto armazenado. Quando tentamos acessar uma variável que não teve nenhum valor associado a ela teremos como retorno "*undefined*" (indefinido).
Exemplo:

```
var marvin;
window.alert(marvin);
// Quando tentamos imprimir a variável marvin na janela de alerta
// será impresso "undefined" pois não há nenhum valor associado a ela.
var text = "";
// O mesmo não ocorre com o caso acima, pois essa variável contém uma
// sequência de caractéres nula e nada será impresso.
```

4.1.4 null

O *null* é a ausência de valor; quando atribuimos *null* a um objeto ou variável significa que essa variável ou objeto não possui valor válido.

Para efeito de comparação, se usarmos o operador de igualdade "==", JavaScript irá considerar iguais os valores *null* e *undefined*. E isso não afeta o uso da comparação *(var.metodo == null)* quando queremos descobrir se um objeto possui determinado método. No entanto, se for necessário diferenciar os dois valores é recomendável o uso do operador "===" de identicidade. Assim, para efeito de comparação, *undefined* e *null* são iguais, mas não idênticos.
Exemplo:

```
var vazio = null;
var ind;
var res = (vazio == ind);
var res1 = (vazio === ind);
// Quando executado a variável res terá o valor true
// e res1 terá o valor false. E se tentarmos imprimir
// a variável vazio, teremos null impresso.
```

4.1.5 Strings

Strings são sequências de caractéres. Em JavaScript a string pode ser tanto um tipo primitivo de dado como um objeto; no entanto, ao manipulá-la temos a impressão de que sejam objetos pois as strings em JavaScript possuem métodos que podemos invocar para realizar determinadas operações sobre elas. Essa confusão ocorre porque quando criamos uma string primitiva, o JavaScript cria também um objeto string e converte automaticamente entre esses tipos quando necessário.

Este conceito será explicado melhor adiante, quando tratarmos de objetos. Para se declarar uma string, basta colocar uma sequência de caractéres entre aspas simples ou duplas.
Exemplo:

```
var str = "Eu sou uma string!";
var str2 = 'Eu também sou uma string';
// Declaração de strings primitivas
var str3 = new String("Outra string");
// Acima um objeto string declarado de forma explícita
// não há diferença nenhuma entre esses dois tipos no que se refere
// a seu uso.
```

4.1.6 Arrays

Os Arrays são pares do tipo inteiro-valor para se mapear valores a partir de um índice numérico. Em JavaScript os Arrays são objetos com métodos próprios. Um objeto do tipo Array serve para se guardar uma coleção de ítens em uma única variável.

Exemplo:

```
var arr = new Array();
// Por ser um objeto podemos usar o "new" em sua criação
var arr = new Array(elem1,elem2, ... ,elemN);
// Dessa forma criamos um array já iniciado com elementos.
var arr = [1,2,3,4];
// outra forma é iniciar um array com elementos sem usar o "new".
var arr = new Array(4);
// Dessa forma criamos um array vazio de 4 posições.
```

Para acessar as variáveis dentro de um array basta usar o nome do array e o índice da variável que se deseja acessar.
Exemplo:

```
arr[0] = "Até mais e obrigado pelos peixes";
arr[1] = 42;
document.write(arr[1]);
//imprime o conteúdo de arr[1]
```

Do mesmo modo, pode-se fazer atribuições ou simplesmente ler o conteúdo da posição.

Em JavaScript os arrays podem conter valores de tipos diferentes sem nenhum problema; podemos colocar em um mesmo array inteiros, strings, booleanos e qualquer objeto que se desejar.

4.2 Operadores

Nesta seção listaremos, de forma sucinta, os principais operadores que compõem o núcleo da linguagem JavaScript.

As tabelas a seguir mostram todos esses operadores.

4.2.1 Aritméticos

Operador	Operação	Exemplo
+	Adição	x**+**y
-	Subtração	x**-**y
*****	Multiplicação	x***y**
/	Divisão	x**/**y
%	Módulo (resto da divisão inteira)	x**%**y
-	Inversão de sinal	**-**x
++	Incremento	x**++** ou **++**x
--	Decremento	x**--** ou **--**x

4.2.2 Comparação

Operador	Função	Exemplo
==	Igual a	(x **==** y)
!=	Diferente de	(x **!=** y)
===	Idêntico a (igual e do mesmo tipo)	(x **===** y)
!==	Não Idêntico a	(x **!==** y)
>	Maior que	(x **>** y)
>=	Maior ou igual a	(x **>=** y)
<	Menor que	(x **<** y)
<=	Menor ou igual a	(x **<=** y)

4.2.3 Bit a bit

Operador	Operação	Exemplo
&	E (AND)	(x **&** y)
|	OU (OR)	(x **|** y)
^	Ou Exclusivo (XOR)	(x **^** y)
~	Negação (NOT)	**~**x
>>	Deslocamento à direita (com propagação de sinal)	(x **>>** 2)
<<	Deslocamento à esquerda (preenchimento com zero)	(x **<<** 1)
>>>	Deslocamento à direita (preenchimento com zero)	(x **>>>** 3)

4.2.4 Atribuição

Operador	Exemplo	Equivalente
=	x **=** 2	Não possui
+=	x **+=** y	x **=** x **+** y
-=	x **-=** y	x **=** x **-** y
***=**	x ***=** y	x **=** x ***** y
/=	x **/=** y	x **=** x **/** y
%=	x **%=** y	x **=** x **%** y
&=	x **&=** y	x **=** x **&** y
|=	x **|=** y	x **=** x **|** y
^=	x **^=** y	x = x **^** y
>>=	x **>>=** y	x **=** x **>>=** y
<<=	x **<<=** y	x **=** x **<<=** y
>>>=	x **>>>=** y	x **=** x **>>>=** y

4.2.5 Lógicos

Operador	Função	Exemplo
&&	E Lógico	(x **&&** y)
||	OU Lógico	(x **||** y)
!	Negação Lógica	**!**x

4.3 Estruturas de controle

if ... else

A estrutura if é usada quando se deseja verificar se determinada expressão é verdadeira ou não, e executar comandos específicos para cada caso.

Exemplo 1:

```
var a = 12;
var b = 5;
if (a == b) {
    window.alert("12 é igual a 5?!?!");
} else {
    window.alert("a é diferente de b");
}
// No caso acima a frase escrita seria "a é diferente de b"
```

Assim, se a expressão for avaliada como verdadeira, o primeiro bloco de comandos é executado. Caso seja avaliada como falsa, o bloco de comandos que segue o *else* será executado.

Também é possível aglomerar mais testes, utilizando-se o comando *else if*.

Exemplo 2:

```
var a = 10;
if (a < 6) {
    window.alert("a menor que 6");
} else if (a > 6) {
    window.alert("a maior que 6");
} else {
    window.alert("se a não é maior nem menor que 6, a é 6!");
}
```

Outra forma possível de se utilizar o *if* é com sua forma abreviada como na linguagem C, usando o operador ternário *?*. Ele pode criar estruturas de decisão simples em apenas uma linha de comando, porém, muitas vezes isso pode prejudicar a clareza do seu código, tornando-o complicado de entender para alguém que não esteja familiarizado com o uso desde operador condicional.

Exemplo:

```
var a = 8;
(a >= 5 ? window.alert("yes") : window.alert("no"));

// Isso equivale ao código:
var a = 5;
if (a >= 5) {
    window.alert("yes");
} else {
    window.alert("no");
}
```

switch ... case

As estruturas do tipo *switch* são usadas quando queremos selecionar uma opção dentre várias disponíveis.

Exemplo:

```
var marvin = "robot";
switch (marvin) {
    case "human":
        document.write("hello carbon unit!");
        break;
    case "alien":
        document.write("brrr I hate aliens!");
        break;
    case "robot":
        document.write("emergency, to the rescue!");
        break;
    default:
        document.write("what are you?");
        break;
}
```

Ao contrário de outras linguagens, os valores de comparação podem ser *strings* além de valores numéricos.

O comando *break* faz com que o *switch* pare de verificar as outras possibilidades abaixo e pode ser omitido caso se deseje uma estrutura que tornará mais de uma opção como verdadeira. Por fim, default é opcional e corresponde a uma sequência de comandos que será executada quando nenhum dos outros *case* o forem.

while

Os laços do tipo *while* são usados quando se deseja que uma sequência de ações seja executada apenas no caso da expressão de condição ser válida. Assim, primeiro a expressão é testada, para depois o conteúdo do laço ser executado ou não.

Exemplo:

```
var cont = [5,2];
while ((cont[0]+cont[1]) < 15) {
    cont[0]+=1;
    cont[1]+=2;
    document.write('cont0 = '+cont[0]+'cont1 = '+cont[1]);
}
// Com o uso de while, no primeiro teste, cont[0]+cont[1] vale 7;
```

do ... while

Diferentemente do *while*, o *do ... while* primeiro executa o conteúdo do laço uma vez e, depois disso, realiza o teste da expressão pare decidir se continuará executando o laço ou irá seguir o resto do programa.

Exemplo:

```
var cont = [5,2];
do{
    cont[0]+=1;
    cont[1]+=2;
    document.write('cont0 = '+cont[0]+'cont1 = '+cont[1]);
} while ((cont[0]+cont[1]) < 15)
// Com o uso de do...while, no primeiro teste, cont[0]+cont[1]
// já valerá 10, e os contadores já terao sido impressos uma vez
// pois o laço já foi executado a primeira vez antes do teste!
```

for

Na maioria das vezes, quando usamos um laço do tipo *while* também construímos uma estrutura com um contador que é incrementado a cada passo para controle do laço e manipulação interna de objetos, arrays como nos exemplos anteriores. Os laços *for* oferecem a vantagem de já possuírem em sua estrutura essa variável de contador e incrementá-la de maneira implícita.
Exemplo:

```
var cont = [5,2,3];
for(var i=0 ; i < 3 ; i++) {
    cont[i]++;
}
// Ao final do laço cada elemento do vetor cont foi incrementado em 1
```

O significado do comando é for(variável de contador inicializada ; condição de parada ; incremento da variável de contador).

for ... in

Existe uma segunda forma de se utilizar os laços *for* para percorrer propriedades de um objeto.
Exemplo:

```
var doc = document;
for(var prop in doc) {
    document.write(prop+"<br />");
}
// Esse laço automaticamente itera pelas propriedades do objeto,
// No caso ele listara todas as propriedades do objeto Document
// responsavel pelo controle do documento exibido na tela.
// Se olhar com cuidado encontrará nessa lista o proprio método
// Write que usamos para imprimir no documento com document.write.
```

4.4 Funções

Funções possuem um papel muito importante na programação estrutural pelo fato de ajudar muito na modularização no programa, ou seja, viabiliza a divisão do programa em partes menores e logicamente relacionadas. Em JavaScript,

existem diversas maneiras de se declarar uma função; mostraremos todas elas aqui com exemplos.

Um ponto importante é que em JavaScript as funções são consideradas como dados, ou seja, podemos atribuir uma função a uma variável ou propriedade de um objeto e a partir desde momento usar a variável ou a propriedade da mesma forma que se usaria a função. Elas também podem ser passadas como argumentos para outras funções e por isso funções de JavaScript são chamadas funções de alta ordem, elas podem tanto receber funções como argumento quanto retornar uma função.

Expressão *function*

A primeira maneira de se declarar uma função é através do uso da palavra chave *function* de maneira similar a como elas são declaradas na linguagem C, com as diferenças de que em JavaScript não definimos o tipo de retorno e nem mesmo o tipo dos argumentos. Uma função complexa pode ser capaz de tratar argumentos diferentes e retornar argumentos diferentes dependendo das circunstâncias nas quais foi invocada. Deve-se definir seu nome e seus argumentos conforme mostra o exemplo a seguir.

Exemplo:

```
function incArray(array,valor) {
    for(item in array) {
        array[item]+=valor;
    }
    return array;
}
// Para invocar essa função depois basta usar incArray(arg1,arg2)
```

O construtor *Function()*

A segunda forma de se declarar uma função é utilizando o construtor *Function()* e o operador *new*, pois em JavaScript funções e objetos são interligados.

Exemplo:

```
var areaTri = new Function("b","h","return (b*h)/2;");
// a função acima calcula a área de um triângulo dadas sua base
// altura. Para invocá-la basta usar areaTri(arg1,arg2)
```

Esse construtor aceita um número qualquer de strings como argumentos. O ultimo argumento será sempre o corpo da função contendo comandos separados por ponto-e-virgula normalmente e todos os outros argumentos do construtor serão considerados argumentos da função que se está criando. Devido a sua

estrutura, essa forma de se declarar funções costuma ser mais usada quando precisamos declarar um função pequena, ocupando apenas uma linha.

Funções como literais

Uma terceira e última forma de se declarar uma função em JavaScript é através de literais.

Exemplo:

```
var areaTri = function(b,h) { return (b*h)/2; };
// Para invocar a função basta usar areaTri(arg1,arg2) como
// na declaração pelo construtor
```

Essa forma é basicamente a mesma que declarar através do construtor *Function()*. No entanto, ela é melhor porque os comandos podem ser declarados com a sintaxe normal de JavaScript ao invés de ser uma string como é o caso do construtor. Com literais não há necessidade de manter a função em uma linha, dentro das chaves podemos construir a função usando um comando por linha normalmente.

4.5 Objetos

Ao contrário de uma variável, um objeto pode conter diversos valores e de tipos diferentes armazenados nele (atributos) e também possuir funções que operem sobre esses valores (métodos). Tanto os atributos, quanto os métodos, são chamados de **propriedades** do objeto.

Para criar um objeto é muito simples, basta invocar seu construtor através do operador new.

Exemplo:

```
var objeto = new Object();
// Quando usamos o construtor Object() criamos um objeto
// genérico
```

Outra forma de criar um objeto é através de literais.

Exemplo:

```
var nave = {
            nome: "coração de ouro",
            propulsao: "Gerador de improbabilidade infinita",
            dono: "Zaphod Bebblebrox"
           }
           // Dessa forma, o objeto nave foi criado possuindo os atributos
           // nome, propulsão e dono com seus respectivos valores
```

Para acessar uma propriedade de um objeto, basta usar *objeto.propriedade* e no caso de métodos adicionar o operador ().

Podemos definir, como já foi dito, um construtor para um objeto, assim podemos inicializar atributos logo no momento da instanciação do objeto. Para que um construtor inicialize um atributo, ele precisa ser referenciado através da palavra-chave *this*.
Exemplo:

```
function Carro(modelo, marca, ano, motor) {
    this.modelo = modelo;
    this.marca = marca;
    this.ano = ano;
    this.motor = motor;
}
// Depois para instanciar um objeto, basta usar:
var car = new Carro("G800" , "Gurgel" , 1976 , 1.0);

// Agora car já possui todos os atributos com dados:
document.write("Carro: "+car.modelo);
// o comando acima irá imprimir "Carro: G800"
```

Métodos

No paradigma de orientação a objetos, os métodos são simplesmente funções que são invocadas por meio de um objeto! E em JavaScript isso é levado tão a sério que a maneira de se criar métodos para seus objetos leva isso ao pé da letra. Basta criarmos uma função e atribuí-la a uma propriedade do objeto.

Exemplo:

```
// Uma função fictícia para cálculo de um consumo de combustível
function calc_consumo(distancia) {
    return distancia/(15/this.motor);
}

// Agora atribuimos a função, sem os argumentos, para a
// propriedade consumo. Considerando o objeto já instânciado
// do exemplo anterior
car.consumo = calc_consumo;

// Pronto! já podemos invocá-la fazendo:
var gas = car.consumo(200);
// calculando quanto o carro gastaria de
// combustível em 200 kilômetros
```

Também podemos definir os métodos dentro do próprio construtor de uma função, tanto definindo a função fora e atribuindo no construtor, como definindo a própria função dentro do próprio construtor uma vez que JavaScript suporta o aninhamento de funções.

Prototypes

Quando declaramos ou atribuímos um método no construtor de um objeto ele ficará disponível para todas as instâncias criadas a partir desse construtor. No entanto, existe um modo muito mais eficiente de se fazer isso, que é com o uso da propriedade *prototype*. Tudo o que for definido no *prototype* de um objeto poderá ser referenciado por todas as instâncias desse objeto. Mesmo as propriedades do *prototype* que forem definidas ou alteradas depois da instanciação serão acessíveis aos objetos declarados anteriormente. Além disso, é importante ter em mente que os atributos e funções declarados no prototype não são copiados para os objetos, portanto há uma economia significativa de memória quando usamos muitas propriedades compartilhadas e instâncias.
Exemplo:

```
// Vamos elaborar mais o exemplo do carro, mas dessa
// vez usando prototype

function calc_consumo(distancia) {
    return distancia/(15/this.motor);
}

// Classe que representa um carro
function Carro(modelo, marca, ano, motor) {
    this.modelo = modelo;
    this.marca = marca;
    this.ano = ano;
    this.motor = motor;
}

Carro.prototype.rodas = 4;
Carro.prototype.consumo = calc_consumo;
// Agora a classe possui uma constante que informa
// o número de rodas e o método consumo em seu
// prototype

var car1 = new Carro("G800" , "Gurgel" , 1976 , 1.0);
var car2 = new Carro("147" , "Fiat" , 1984 , 2.0);

// Podemos acessar agora tanto a constante rodas
// quanto o método consumo
if(car1.rodas == 4) window.alert("ainda bem!");
var gas = car2.consumo(180);
// e o mais importante é que ambas estão acessando
// apenas uma única constante e um único método
// na memória
```

Arrays Associativos

Para finalizar nossa discussão sobre objetos, vamos mostrar como eles podem ser usados como arrays associativos, ou seja, um array com objetos indexados por valores não numéricos. Isso só pode ser feito porque é possível acessarmos atributos de um objeto usando *MeuObjeto["atributo"]*. Assim podemos simular o comportamento de um array associativo armazenando cada item em um atri-

buto.

Exemplo:

```
var arr = new Object();
arr["nome"] = "Zaphod Beeblebrox";
arr["cargo"] = "Presidente do Universo";

window.alert(arr["nome"]);
// Irá mostrar uma mensagem contendo:
// "Zaphod Beeblebrox"

// Note que não há nenhuma diferença se fizermos:
window.alert(arr.nome)
// Exceto que a string usada como índice no modo []
// pode ser manipulada em tempo de execução
```

A seguir, vamos dedicar atenção aos exemplos de métodos e propriedades de dois importantes objetos nativos do JavaScript: os Arrays e Strings.

4.5.1 Objeto String

Vamos nos ater agora aos métodos das Strings, e embora existam outros, aqui serão relacionados apenas os que fazem parte da ECMA 262-3 que equivale ao JavaScript 1.6, pois estes métodos são comuns a uma grande variedade de *browsers* como FireFox, Netscape e Internet Explorer.

valueOf() - retona o valor primitivo do objeto string. É útil quando desejamos atribuir o valor de um objeto string para uma variável que seja do tipo primitivo string.

Exemplo:

```
var stob = new String("Gerador de campos POP");
var str = stob.valueOf();
// Dessa forma stob será um objeto do tipo Object e
// str será uma variável primitiva do tipo string.
```

charAt(pos) - retorna uma string contendo o caractere de posição *pos* da string. Se não existir nenhum caractere nessa posição, o resultado é uma string vazia.

Exemplo:

```
var str = "Milliways";
var carac = str.charAt(5);
// A variável carac deverá conter o caractére 'w' após a execução.
// pois a contagem começa em zero
```

concat(string1,string2, ... ,stringN) - este método retorma uma string contendo a própria string da qual o método foi chamado e todos os caracteres das strings que lhe foram passadas como argumentos, na ordem em que

foram fornecidos.

Exemplo:

```
var str1 = "Praticamente ",str2 = "Inofensiva";
var result = str1.concat(str2);
// A variável result deverá conter a string
// "Praticamente Inofensiva" apos a execução do código.
```

indexOf(padrao,pos) - procura a ocorrência da string contida em *padrao* a partir da posição *pos* dentro da string sobre a qual se invocou este método e retorna o índice da primeira ocorrência. Caso não encontre o padrão buscado, retorna -1. O atributo *pos* não é obrigatório, se ele estiver indefinido, o valor 0 (zero) será assumido como padrão e a busca nesse caso afetará a string inteira.

Exemplo:

```
var cachalote = "Vou chamar isso de cauda, cauda é um bom nome!";
var ind1 = cachalote.indexOf("cauda");
var ind2 = cachalote.indexOf("cauda",25);
var ind3 = cachalote.indexOf("improbabilidade");
// Após executado a variável ind1 deverá ter o valor 19
// ind2 deverá conter 26, que é o valor da ocorrência de cauda
// após a posição 25, como foi especificado no parâmetro
// e ind3 irá conter −1 pois o padrão não é encontrado.
```

lastIndexOf(padrao,pos) - como o método anterior, no entanto, este retorna o índice da última ocorrência de *padrao* na string sobre a qual se utilizou o método. Do mesmo modo, será retornado -1 se o padrão não for encontrado. No entanto, a posição de *pos* afeta a busca como um limitante superior, ou seja, a busca ocorrerá do índice zero até o índice contido em *pos*. Essa é uma diferença sutil, mas pode causar problemas se não for levada em consideração.

Exemplo:

```
var cachalote = "Vou chamar isso de cauda, cauda é um bom nome!";
var ind1 = cachalote.lastIndexOf("cauda");
var ind2 = cachalote.lastIndexOf("cauda",25);
var ind3 = cachalote.lastIndexOf("improbabilidade");
// Após executado a variável ind1 deverá ter o valor 26
// ind2 deverá conter 19, que é o valor da ocorrência de cauda
// até a posição 25, como foi especificado no parâmetro
// e ind3 irá conter −1 pois o padrão não é encontrado.
```

replace(velho,novo) - busca na string uma substring que seja igual ao conteúdo de *velho* e a substitui pelo conteúdo de *novo* e retona a string resultante da substituição. Além disso, o argumento *novo* pode ser uma função. Dessa maneira, para cada ocorrência da substring a função será chamada com 3 argumentos: a substring encontrada; o deslocamento do início da

string até a ocorrência da substring; e por fim a própria string; finalmente, a substring encontrada será substituída pelo retorno da função chamada.
Exemplo 1:

```
var a = "Não se esqueça de sua toalha!";
var res1 = a.replace("toalha","mochila");
var res2 = a.replace("bicicleta","mochila");
// No primeiro caso a substring toalha foi encontrada e será
// substituída por mochila. No segundo caso, como o padrão não
// é encontrado, não ocorre substituição nenhuma.
```

Exemplo 2:

```
// Uma função que chama o método toUpperCase para transformar
// a substring em caractéres maiúsculos e retorná-la.
function up(sub,pos,str) {
    return sub.toUpperCase();
}
\
var a = "Não se esqueça de sua toalha!";
var res = a.replace("toalha",up);
// Neste caso quando o padrão é encontrado a função up
// é chamada e seu valor de retorno substitui o padrão.
// no caso "toalha" será substitudo por "TOALHA".
```

Note que o exemplo acima ilustra apenas um uso muito simples do que pode ser realmente feito com o uso do método replace chamando uma função. Essa função pode ser tão complexa quanto necessário, desde que receba os mesmos três parâmetros. O método toUpperCase será explicado adiante.

search(padrao) - busca o conteúdo de *padrao* e retorna o índice de início desse padrão ou -1 caso ele não seja encontrado. Também é possível utilizar esse método com uma expressão regular em *padrao*.
Exemplo:

```
var str = "A vida, o universo e tudo mais";
var res1 = str.search("tudo");
var res2 = str.search("cachalote");
// O código acima após executado resultaria no valor 21 na
// variável res1 e -1 na variável res2, porque cachalote
// não foi encontrado na string.
```

slice(inicio,fim) - retona a substring contendo os caractéres da posição *inicio* até a posição *fim*, mas sem a incluir, ou até o final da string se *fim* for indefinido. No caso de inicio e fim serem valores negativos, eles serão tratados como se estivéssemos acessando o array na direção inversa (do fim para o início). Dessa forma, -1 equivale a posição do último caractére, -2 a do penúltimo e assim por diante. No entanto, a posição de fim deve sempre

corresponder a uma posição à direita de início, caso contrário será retornada uma string vazia.
Exemplo:

```
var str = "Até mais e obrigado pelos peixes";
var res1 = str.slice(11); // Armazena "obrigado pelos peixes"
var res2 = str.slice(11,20); // Armazena "obrigado"
var res3 = str.slice(-7,-1); // Armazena "peixe".
```

split(separador,limite) - retorna um objeto Array contendo as substrings que resultaram da separação da string original pelo conteúdo de *separador*, sem incluí-lo. Se separador for indefinido, o Array resultante terá como elemento apenas a string original. O argumento limite define o número máximo de elementos que o array pode ter, excedido esse valor o Array será truncado. Caso o limite seja indefinido, não haverá um limite no número de elementos do array resultante.
Exemplo:

```
var dados = "Terra, Marte, Jupiter";
var arr1 = dados.split(", ");
var arr2 = dados.split(", ",2);
// Em arr1 será armazenado um objeto array contendo na posição 1
// "Terra", na posição 2 "Marte" e na posição 3 "Jupiter".
// Já em arr2 será armazenado um array da mesma forma que descrito
// acima, no entanto apenas com as duas primeiras posições, ou seja,
// sem Jupiter.
```

substring(inicio,fim) - este método funciona da mesma maneira e para os mesmos fins que slice(inicio,fim). Retorna o resultado da conversão do objeto que chamou este método para uma string, desde o caractére da posição *inicio*, até o caractére anterior à posição *fim* ou até o final da string, caso *fim* não seja definido. Se o argumento foi inválido ou negativo, ele será automaticamente substituído por zero e se o comprimento foi maior do que a string, será limitado pelo próprio tamanho da string.

toLowerCase() - os caractéres da string são todos convertidos para letras minúsculas. Se o objeto não for uma string, então ele será convertido automaticamente para string antes da operação ser realizada.
Exemplo:

```
var str = "Pense em um NÚMERO entre um e um zilhão";
var res = str.toLowerCase();
// Depois de executado o código, a string res deverá
// conter a string "pense em um número entre um e um zilhão".
```

toUpperCase() - semelhante ao método toLowerCase(), no entanto, ele converte todas as letras da string para letras maiúsculas.
Exemplo:

```
var str = "Esta frase será convertida";
var res = str.toUpperCase();
// Após executado a variável res irá conter a string
// "ESTA FRASE SERÁ CONVERTIDA".
```

Quanto a propriedades, os objetos do tipo string só possuem uma:

length - valor inteiro que contém o número de caractéres que compõem a string.

4.5.2 Objeto Array

Agora faremos uma breve descrição dos métodos que o objeto Array traz consigo.

concat(item1,item2, ... ,itemN) - agrupa dois ou mais arrays e retorna o resultado
Exemplo:

```
var x = ["a vida","o universo"];
var y = ["e tudo mais"];
var resultado =  x.concat(y);
// Isso geraria o array resultado contendo
// ["a vida","o universo","e tudo mais"]
```

join(separador) - agrupa todos os elementos contidos no array em uma string separados pelo que estiver na variável separador, se um caractere separador não for fornecido a vírgula será usada como padrão. Para tal operação, os elementos do array são convertidos para strings caso já não o sejam.
Exemplo:

```
var x = ["a vida","o universo","e tudo mais"];
var str = x.join("\ ");
// Após a execução do código str terá "a vida o universo
// e tudo mais" pois os valores foram agrupados com
// espaço como separador
```

pop() - o último elemento do array é removido e retornado.
Exemplo:

```
var vetor = [4,8,15,16,23,42];
var resposta = vetor.pop();
// Assim resposta irá conter o valor 42.
```

push(item1,item2, ... ,itemN) - insere os N itens no final do array, na ordem que eles foram passados e retorna o novo tamanho (*length*) do array.
Exemplo:

```
var vetor = ["Arthur","Ford",];
var retorno = vetor.push("Marvin");
// Agora o vetor possui ["Arthur","Ford","Marvin"]
// e o valor de retorno é 3.
```

reverse() - retorna o array com os elementos na ordem inversa.

Exemplo:

```
var vetor = [1,2,3,4];
var inverso = vetor.reverse();
// Agora tanto inverso quanto vetor possuem [4,3,2,1]
```

shift() - o primeiro elemento do array é removido e retornado.

Exemplo:

```
var vetor = [4,8,15,16,23,42];
var elemento = vetor.shift();
// Depois de executado elemento terá o valor 4
// e o vetor será [8,15,16,23,42];
```

slice(inicio,fim) - retorna um array contendo os elementos de inicio até fim, mas sem incluir o elemento da posição fim. Caso fim não seja declarado, retorna um array de inicio até o final do array. Se inicio for negativo, o valor será contado a partir do final do array, por exemplo, -2 para indicar a penúltima posição, o mesmo acontece com fim caso seja negativo. Podemos imaginar o sinal de negativo apenas como um inversor no sentido que os elementos são considerados (inicio-fim para valores positivos e fim-inicio para valores negativos).

Exemplo:

```
var meuarray = [1,2,3,4];
var meio = meuarray.slice(1,3);
var meio = meuarray.slice(-3,-1);
// nas duas atribuições acima a variável meio receberia
// o array [2,3] pois -3 equivale a posição do elementos 2
// e a -1 equivale a posição do elemento 4. é preciso lembrar
// que elemento fim não é incluido no array de retorno.
```

sort(comparador) - método que ordena os elementos objeto do Array por ordem alfabética. Isto significa que ele não funcionará para números; para ele o número 33 virá antes do número 7 pois o primeiro algarismo de 33 é menor. Para comparar números ou outros tipos de elementos você pode fornecer ao método um comparador.

Este deve ser uma função com as seguintes características:

- receber dois argumentos: x e y.

- retornar um valor negativo se x < y.
- retornar zero se x == y.
- retornar um valor positivo se x > y.

Caso contrário, o método terá o comportamento padrão descrito acima.
Exemplo 1:

```
var vetor = ["d","y","a","n","m"];
vetor.sort();
// Após a execução vetor conterá o array
["a","d","m","n","y"].
```

Exemplo 2:

```
// Uma função simples para comparar valores numéricos
// os detalhes sobre a utilização de funções serão abordadas
// mais adiante.
function compara(x,y) {
    if (x < y)
        return -1;
    else if (x == y)
        return 0;
    else
        return 1;
}
var vetor = [5,45,1,4,16];
var errado = vetor.sort();
var certo = vetor.sort(compara);
// O array errado irá conter [1,16,4,45,5]
// e o array certo irá conter [1,4,5,16,45]
```

splice(inicio, quantidade, elem1, ... , elemN) - este método pode ser usado para remover, inserir ou substituir elementos em um Array. Os argumentos "inicio" e "quantidade" são obrigatórios, mas os elementos posteriores são opcionais.

Para apenas remover elementos do array devemos usar o método com: inicio sendo o índice do primeiro elemento a ser removido, quantidade sendo o numero de elementos a remover e nenhum outro argumento.

Para inserir um elemento sem remover nenhum outro, basta utilizar o método com parâmetro inicio sendo o índice da posição onde serão inseridos os elementos, quantidade deve ser 0 pois não desejamos remover nenhum elemento e cada elemento a ser inserido deve ser um novo argumento subsequente.

Para substituir elementos do array, precisamos que inicio contenha o índice a partir do qual os elementos serão substituídos, quantidade deve conter o número de elementos que serão substituídos e mais um argumento para cada elemento quer será inserido substituindo os elementos originais. Note que se houver menos elementos do que o valor em quantidade, o excedente

será apenas removido sem substituição e se houver mais elementos do que posições deletadas o excedente será apenas inserido após as substituições.

Exemplo 1:

```
var array = [1,2,3,4,5,6];
array.splice(2,2);
// Como resultado teremos em array o vetor [1,2,5,6]
// apenas deletamos dois elementos partindo do índice 2.
```

Exemplo 2:

```
var array = [1,2,5,6];
array.splice(2,0,3,4);
// Como resultado teremos em array o vetor [1,2,3,4,5,6]
// Isso significa que a partir do índice 2 deletamos 0 elementos
// e inserimos os valores 3 e 4
```

Exemplo 3:

```
var array = [1,2,3,4,5,6];
array.splice(2,2,7,8);
// Como resultado teremos em array o vetor [1,2,7,8,5,6]
// Ou seja, a partir do índice 2 deletamos 2 elementos e inserimos
// os elementos 7 e 8 no lugar deles
```

Em todos os casos, o método retorna um array contendo os elementos que foram removidos se houve algum.

unshift(item1,item2, ... ,itemN) - adiciona um ou mais elementos ao início do array na ordem que os argumentos foram fornecidos e retorna o novo número de ítens.
Exemplo:

```
var vetor = [4,2,6,1];
var tamanho = vetor.unshift(5,3);
// Após a execução a variável tamanho irá conter o valor 6
// e vetor será [5,3,4,2,6,1];
```

toString() - este método se comporta da mesma maneira que o método join quando chamado sem nenhum parâmetro, ou seja, ele separa os elementos por vírgula.
Exemplo:

```
var vetor = ["Hiro","Peter","Claire"];
var result = vetor.toString();
// result irá conter a string "Hiro,Peter,Claire"
```

A seguir, vamos dar uma olhada nas propriedades do objeto Arrays. As propriedades são acessadas de modo similar aos métodos, basta usar o nome do objeto.propriedade e depois disso teremos um exemplo simples.

constructor - esta propriedade é uma referência à função que criou este objeto array.

length - propriedade que contém o tamanho do array, ou seja, a quantidade de elementos contidos nele.

prototype - permite se adicionar propriedades e métodos a este objeto array. Este é usado na orientação a objetos para que um objeto herde elementos do prototype do outro.
Exemplo:

```
var vetor = ["a","b","c","d"];
var tam = vetor.length;
// A variável tam irá conter o valor 4 porque existem quatro
// elementos no array
```

Encerramos nossa discussão sobre objetos aqui. Embora haja mais detalhes que podem ser apresentados e caracterizem o potencial de uso sobre sobre eles, não é objetivo deste documento nos aprofundarmos muito nos tópicos abordados e sim proporcionar as ferramentas para um bom uso inicial da linguagem. Aqueles que se interessarem pelo assunto poderão encontrar muitas informações nos materiais referenciados na Bibliografia.

4.6 Exceções

Nas versões atuais, comandos para manipulação de exceções foram incluídos em JavaScript, de forma similar aos que a linguagem Java oferece. Temos os comandos *throw, try, catch* e *finally*. Com eles é possível desenvolver uma aplicação em JavaScript capaz de tratar possíveis erros em tempo de execução, aumentando de maneira considerável sua robustez. Vamos mostrar com exemplos o seu funcionamento.
Exemplo 1:

```
var arr = ["oi","ola"];

/* Função para ler uma posição de um array e
 * mostra-la na tela
 */
function lerarr(indice) {
    window.alert(arr[indice]);
}

lerarr();
// Essa chamada resulta em um erro de lógica
```

```
// pois como não passamos o argumento índice
// ele pode conter lixo ou não ser definido.
```

Agora vamos introduzir o comando *Throw* para fazer com que essa função gere uma exceção que possa ser tratada pela própria aplicação.
Exemplo 2:

```
/* Função para ler uma posição de um array e
 * mostra-la na tela
 */
function lerarr(indice) {
    if (!indice) throw Error;
    else window.alert(array[indice]);
}

lerarr();
// Ao executar essa chamada agora temos a mensagem:
// "uncaught exception: function Error() { [native code] }"
```

Agora temos uma função que é capaz de verificar a existência do seu parâmetro e caso ele não tenha sido passado, gera uma exceção.

JavaScript não nos obriga a tratar essas exceções enquanto elas não forem lançadas, por exemplo, no caso acima se tivéssemos passado o argumento corretamente, não receberíamos nenhuma mensagem dizendo que existe uma exceção não tratada. O interpretador irá apenas "reclamar" quando essa exceção for lançada e não estiver sendo tratada, como no caso acima. Agora vejamos como podemos tratá-la.

Exemplo:

```
/* Considerando a função do exemplo anterior já declarada */

try {
    lerarr(); //função que pode lançar exceção
} catch (e) {
    // comandos a serem executados em caso de exceção
    window.alert("Oops! No soup for you!");
} finally {
    document.write("ocorreu um erro");
}
```

No código acima, a chamada da função está protegida pelos comandos *try ... catch*. O interpretador tenta executar os comandos que estão dentro do laço *try*; se houver uma execeção lançada, ele pára a execução imediatamente e pula para o laço *catch* onde ele tentará tratar essa exceção.

O bloco *finally* pode ser utilizado para executar um bloco de comandos, havendo exceção ou não. Pode ser usado, por exemplo, para liberar alguma varíavel ou fechar um arquivo que esteja sendo editado. Em geral, esse bloco de comandos serve para garantir a chamada *graceful degradation* que consiste em uma tolerância a falhas de modo que o usuário possa continuar utilizando o sistema, mesmo que haja uma queda na qualidade em função do erro.

5 Web 2.0

5.1 O que é Ajax?

O nome Ajax foi cunhado por Jesse James Garrett em 2005 para se referir a um conjunto de tecnologias que já existiam, mas que passaram a ser usadas de forma inovadora, enriquecendo as possibilidades de interação na web para torná-las o mais próximo de aplicações *desktop* quanto possível.

As páginas web sempre sofreram com falta de interação devido a própria natureza caótica da internet, de possuir uma latência alta e ser pouco confiável. Assim, é muito comum clicarmos em um link e termos que esperar as vezes até alguns segundos, dependendo da conexão, até que a próxima página seja carregada do servidor para sua máquina. É claro que as conexões têm melhorado dia a dia, mas ainda assim o simples fato de que a cada mudança de página precisamos exibir um documento totalmente novo que contém, além dos dados que requisitamos, todas as informações de *layout* novamente, representa um gasto de banda considerável.

Uma grande diferença do Ajax é que as páginas apresentadas no *browser* passam a ser aplicações, ou seja, a primeira vez que entramos em uma página, a aplicação é carregada para nossa máquina e depois essa aplicação fica responsável por requisitar ou enviar os dados para o servidor de forma assíncrona. Como a aplicação está o tempo todo no *browser* do cliente, este não perde a interação e pode realizar ações mesmo enquanto espera a requisição de algum dado do servidor. Por exemplo, continuar trabalhando com os dados que já foram carregados no passado. A Figura 1 ilustra essa diferença, nela podemos observar que as aplicações Ajax possuem uma camada a mais entre a interface com o usuário e o servidor, essa camada é a aplicação propriamente dita.

5.2 O papel do JavaScript

As quatro principais tecnologias utilizadas para o desenvolvimento de aplicações Ajax são esquematizadas na Figura 2. Elas desempenham as seguintes funções:

JavaScript é responsável por interligar todas as outras tecnologias, é a linguagem de programação e portanto com ela é desenvolvida a aplicação que irá ser executada na máquina do cliente.

CSS (*Cascade Style Sheets*) é um padrão da W3C para estilizar elementos em uma página *web*. Ele é utilizado para dar uma boa aparência às páginas, podendo ser acessado e editado pelo JavaScript.

DOM (*Document Object Model*) permite que uma linguagem como o JavaScript possa manipular e alterar a estrutura de documentos, por exemplo, uma

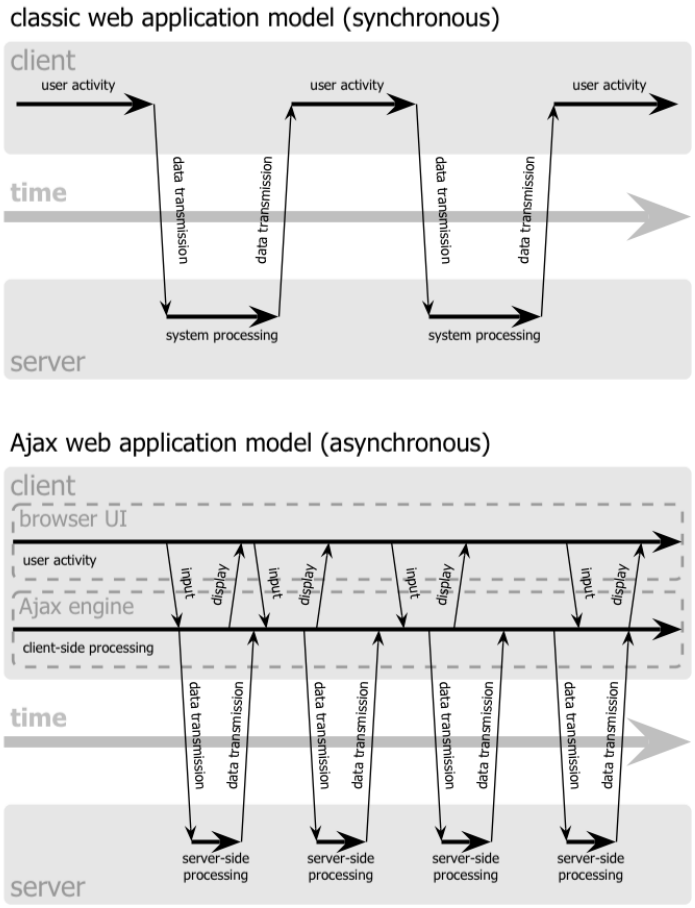

Figura 1: As diferenças entre uma página web comum e uma aplicação Ajax [2]

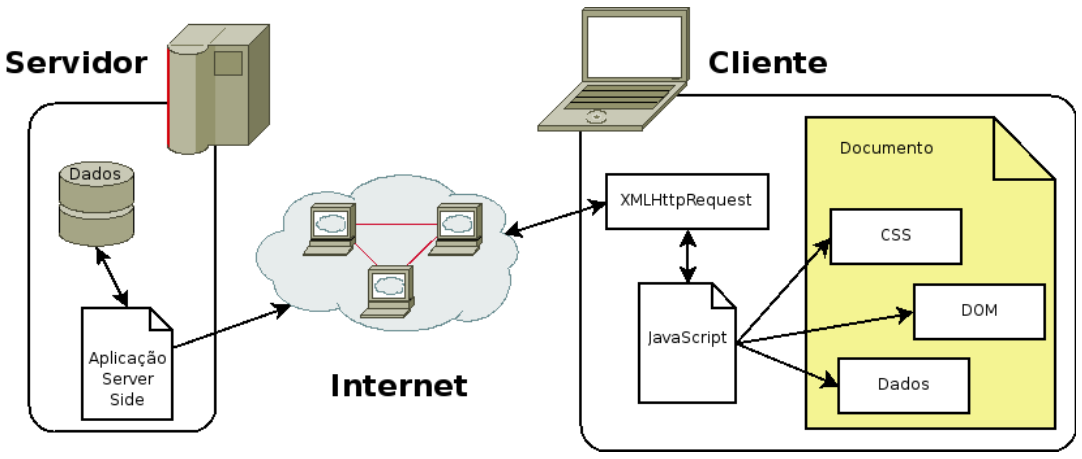

Figura 2: Arquitetura de uma aplicação Ajax. Observe como o código JavaScript é o elemento responsável por organizar todos os outros no cliente e trocar dados com o servidor

página durante seu tempo de vida no *browser* do cliente.

XMLHttpRequest é um objeto existente em JavaScript que permite a troca de dados com o servidor de forma assíncrona.

Aqui daremos atenção especial ao JavaScript e como ele é utilizado para manipular os elementos listados anteriormente, CSS, DOM e XMLHttpRequest. Sugerimos fortemente que procure entender melhor o uso do DOM e CSS pois são ferramentas essenciais a um bom desenvolvedor web.

Com o código JavaScript, podemos acessar todos os elementos da árvore DOM de um documento e podemos alterá-los, removê-los ou mesmo inserir um novo elemento. Isso nos ajuda a exibir os dados novos que foram requisitados do servidor pelo XMLHttpRequest. A seguir, vamos mostrar um exemplo simples de como acessar a árvore DOM com JavaScript.

```
dom.js - Arquivo JavaScript
--------------------------------------------------------------------------------

function divEdit() {

/* busca na árvore DOM o elemento com ID "header" */
var header = document.getElementById("header");

/* guarda o código HTML de dentro do elemento */
var conteudo = header.innerHTML;

/* reescreve o conteúdo adicionando outras tags */
header.innerHTML = "<strong>"+conteudo+"</strong>";

/* cria um novo elemento DOM */
var paragrafo = document.createElement('p');
/* configura a propriedade title do elemento */
paragrafo.setAttribute('title','Novo parágrafo');
/* cria um nó de texto */
var txt = document.createTextNode('Parágrafo adicionado a árvore DOM');

/* insere o texto ao paragráfo */
paragrafo.appendChild(txt);

/* insere o paragráfo na página */
header.appendChild(paragrafo);

}
```

```
index.html - Arquivo HTML
--------------------------------------------------------------------------------
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <script type="text/javascript" src="dom.js"></script>
</head>

<body>
  <div id="header">
    Exemplos de manipulação do DOM por JavaScript.
  </div>
  <hr />
  <br />
  <input type="button" value="Altera árvore DOM" onclick="divEdit()" />
</body>
</html>
```

Quando executar o exemplo acima em seu *browser* a página irá conter apenas uma frase no início e um botão com o nome "Altera árvore DOM". Quando for clicado esse botão, a função divEdit() é chamada e insere um novo parágrafo na página sem necessidade de recarregá-la. Este exemplo simples mostra apenas como usar JavaScript para manipular a árvore DOM, não entramos na área do Ajax ainda, mas essa é a base para o tratamento dos dados obtidos através da técnica.

5.3 Exemplo de aplicação Ajax

Nessa seção mostraremos um exemplo mais completo de uma aplicação Ajax separada em 4 arquivos, um XML com dados, um CSS com o estilo da página, um JS com a aplicação e o HTML que irá conter a aplicação.

A aplicação consiste em um página que fica constantemente requisitando um arquivo de dados com notícias do servidor e atualizando a página caso haja notícias novas nesse arquivo.

Vamos iniciar esse estudo com o arquivo de dados que armazenará as notícias. Ele é um XML muito simples que armazena cada notícia com sua data, título e conteúdo:

```
Arquivo: data.xml
-------------------------------------------------------------------------------

<?xml version="1.0" ?>
<news>
  <new>
    <date>05/06/2007</date>
    <title>
      Produção industrial declina 0,1% em abril ante março,
      informa IBGE
    </title>
    <content>
      RIO - A produção industrial brasileira verificou
      decréscimo de 0,1% em abril no confronto com um mês antes,
      levando em conta ajuste sazonal. Foi a primeira taxa
      negativa em seis meses, mostrou pesquisa do Instituto
      Brasileiro de Geografia e Estatística (IBGE) divulgada há
      pouco. Ante abril do ano passado, houve, no entanto,
      crescimento de 6%, marcando a 10ª elevação na série sem
      ajustes e o melhor resultado desde junho de 2005 (6,4%)...
    </content>
  </new>

  <new>
    <date>05/06/2007</date>
    <title>Bovespa recua 0,13%, aos 53.175 pontos</title>
    <content>
      SÃO PAULO - A Bolsa de Valores de São Paulo (Bovespa)
      começou as operações de hoje com pequena variação, aos
      53.244 pontos. Em mais de 30 minutos de atividades, porém,
      o Ibovespa diminuía 0,13%, aos 53.175 pontos, e volume
      financeiro de R$ 290,025 milhões...
    </content>
  </new>

  <new>
    <date>11/06/2007</date>
    <title>
      Compadre de Lula declara à PF que já sabia de grampo
    </title>
    <content>
      Compadre do presidente Luiz Inácio Lula da Silva, Dario
      Morelli Filho, preso na Operação Xeque-Mate, disse em
      depoimento à Polícia Federal que já sabia da existência de
      grampo em seu telefone, segundo reportagem publicada nesta
      segunda-feira pela Folha...
    </content>
```

```
  </new>
</news>
```

Agora vejamos o arquivo HTML. Devemos notar que o arquivo news.css é importado e o arquivo ajax.js é incluído na página, ambos serão listados a seguir. Além disso, quando a página termina de carregar, a função *timedNews* é chamada do arquivo de JavasScript com o nome do arquivo de dados como argumento. Também temos algumas *divs* que serão utilizadas pela aplicação como pontos de entrada de dados.

```
Arquivo: news.html
-------------------------------------------------------------------------------

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt" >

  <head>
    <title>notícias 2.0</title>
    <style type="text/css" media="all">
      @import "news.css";
    </style>
    <script type="text/javascript" src="ajax.js"></script>
  </head>

  <body onload="timedNews('data.xml');">
    <div id='loadDiv'></div>
    <div id='lastnews'>
      <h2>Últimas notícias</h2>
      <div id='news'></div>
    </div>
  </body>

</html>
```

O arquivo CSS é usado apenas para estilizar o documento. Com CSS é possível tornar vários textos aparentemente sem formatação em uma página muito atraente!

No nosso caso, por se tratar de um página pequena e com o propósito de mostrar apenas o funcionamento do Ajax, não precisamos de uma folha de estilos muito complexa. Assim, os comandos CSS trabalham mais com as margens, cores e posicionamento das *divs* vistas no documento HTML anteriormente.

```
Arquivo: news.css
-------------------------------------------------------------------------------

/* ---------------------- TAGS -------------------*/

body {
  margin: 0;
  text-align: center;
}

h2 {
  font-size: 12pt;
```

```
  font-weight: bold;
}

h3 {
  font-size: 10pt;
  font-weight: bold;
  color: #8B0000;
  padding-top: 10px;
  border-top: 1px dashed #000000;
}


/* ---------------------- IDS -------------------*/

#loadDiv {
  position: relative;
  float: right;
  top: 0;
  background-color: #8B0000;
  padding: 2px 2px 2px 2px;
  font-size: 8pt;
  color: #FFFFFF;
}

#lastnews {
  padding: 4px 4px 4px 4px;
  position: relative;
  width: 192px;
  margin: 0 auto;
  text-align: left;
  background-color: #BEBEBE
}


/* ---------------------- CLASSES -------------------*/

.corpo {
  font-size: 8pt;
  font-family: Verdana,Georgia;
}
```

Agora finalmente o arquivo JavaScript com a aplicação. Seu funcionamento será explicado a seguir.

```
Arquivo: ajax.js
-------------------------------------------------------------------------------

var req=null;

// 'Constante' para teste de quando os dados estão prontos para serem usados
var READY_STATE_COMPLETE = 4;


/**
  * Função que fica verificando as notícias no servidor em períodos
  * de tempo
  * @param data Endereco dos dados no servidor
  */
function timedNews(data)
{
    // Busca os dados no servidor
    sendRequest(data);
```

```
    // Programa para ser executada novamente em 30 segundos
    var t = setTimeout("timedNews('"+data+"')",30000);
}

/**
  * Função para requisitar uma página com dados
  * @param url URL do dado a ser buscado
  * @param params Parâmetos (querystring) para páginas dinâmicas
  * @param HttpMethod Método do protocolo HTTP, se omitido será usado GET
  */
function sendRequest(url,params,HttpMethod)
{
    // Verifica se o metodo foi setado, caso contrário seta para GET
    if (!HttpMethod)
    {
        HttpMethod = "GET";
    }

    // Instância o objeto XMLHttpRequest
    req = initXMLHttpRequest();
    if (req)
    {
        // Seta a função de callback
        req.onreadystatechange = onReadyState;
        req.open(HttpMethod,url,true);
        req.setRequestHeader
            ("Content-Type","text/xml");
        req.send(params);
    }
}

/**
  * Função que inicia o objeto XMLHttpRequest de acordo com o browser cliente.
  * @return objeto do tipo XMLHttpRequest.
  */
function initXMLHttpRequest()
{
    var xRequest = null;
    // código para o Mozilla
    if (window.XMLHttpRequest)
    {
        xRequest = new XMLHttpRequest();
    }
    // código para o IE
    else if (window.ActiveXObject)
    {
        xRequest = new ActiveXObject("Microsoft.XMLHTTP");
    }

    // Retorna o objeto instânciado
    return xRequest;
}

/**
  * Função de callback que verifica se o dado já esta pronto para uso.
  * Caso posistivo, chama um função para trata-lo
  */
function onReadyState()
{
    var ready = req.readyState;
    // Se a requisição estiver completa
    if (ready == READY_STATE_COMPLETE)
```

```
    {
        // Retira mensagem de loading
        loadingMsg(0)
        // Trata os dados recem chegados
        parseNews();
    }
    // Caso contrário coloca a mensagem de loading
    else
    {
        loadingMsg(1);
    }

}



/**
  * Função que dado a data e título de uma notícia verifica se ela já
  * esta sendo exibida na página.
  * @param titulo O título da notícia que se deseja buscar
  * @return 'True' se a notícia já existir e 'False' caso contrário
  */
function newDetect(titulo)
{
    // Recupera a div onde as notícias serão inseridas
    var sitenews = document.getElementById('news');
    // Pega o primeiro filho
    var oldnew = sitenews.firstChild;

    if (oldnew == null)
    {
        // a div de notícias não possui filhos
        // logo não existe nenhuma notícia
        return false;
    }
    else
    {
        // Busca em todas as notícias
        while (oldnew != null)
        {
            // Armazena o título das notícias atual
            var tit = oldnew.getElementsByTagName('h3');
            tit = tit[0].innerHTML;

            // Se for igual ao título que estamos testando
            if (tit == titulo)
            {
                // Retorna que a notícias já existe
                return true;
            }
            oldnew = oldnew.nextSibling;
        }
        // Se nenhuma notícias for igual
        // Retorna que não existe
        return false;
    }
}

/**
  * Função que recebe o XML carregado, separa as notícias e as
  * insere na página caso ela não exista
  */
function parseNews()
```

```
{
    var news = req.responseXML.getElementsByTagName('new');

    for (var i=0 ; i<news.length ; i++)
    {

        var date = getNodeValue(news[i],'date');
        var title = getNodeValue(news[i],'title');
        var titcont = date+" - "+title;

        // Veririca se a notícias já esta na página
        if (newDetect(titcont) == false)
        {
            // Cria a div que ira conter a notícia
            divnew = document.createElement('div');
            divnew.setAttribute('id','new'+i);

            // Cria o título das notícias
            var titulo = document.createElement('h3');
            titulo.appendChild(document.createTextNode(titcont));

            // Cria o corpo das notícia
            var corpo = document.createElement('p');
            corpo.setAttribute('class','corpo');
            // Agrega o conteúdo obtido ao corpo criado anteriormente
            var txt = document.createTextNode(getNodeValue(news[i],'content'));
            corpo.appendChild(txt);

            // Agrega o título e o corpo a div criada no inicio
            divnew.appendChild(titulo);
            divnew.appendChild(corpo);

            // Busca onde a notícias será inserida na página
            var base = document.getElementById('news');
            // Insere a notícia como primeira da página
            base.insertBefore(divnew,base.firstChild);
        }
    }
}

/**
  * Função para pegar o valor de uma propriedade de um objeto
  * @param obj
  * @param tag Nome da propriedade que se deseja saber o valor
  * @return retorna uma string contendo o valor da propriedade
  */
function getNodeValue(obj,tag)
{
    return obj.getElementsByTagName(tag)[0].firstChild.nodeValue;
}


/**
  * Função que indica indica ao usuário quando dados estam sendo carregados
  * @param set Valor 1 para colocar a mensagem, qualquer outro para remover
  */
function loadingMsg(set)
{

    if (set == 1)
    {
        var msg_div = document.getElementById('loadDiv');
```

```
        if (msg_div == null)
        {
            // Cria a div de loading
            msg_div = document.createElement('div');
            msg_div.setAttribute('id','loadDiv');

            // Insere o texto na div criada
            var txt = document.createTextNode('Loading...');
            msg_div.appendChild(txt);

            // Agrega a div no corpo da página
            var corpo = document.getElementsByTagName('body');
            corpo[0].appendChild(msg_div);
        }
        else
        {
            // Altera o conteúdo da div
            msg_div.innerHTML = "Loading...";
        }
    }
    else
    {
        // Busca a div de loading na página
        var msg_div = document.getElementById('loadDiv');

        // Se encontrar remove
        if (msg_div != null)
        {
            var pai = msg_div.parentNode;
            pai.removeChild(msg_div);
        }
    }
}
```

Agora vamos descrever cada uma das funções separadamente, para esclarecer quaisquer dúvidas que tenham restado após a leitura do código todo.

timedNews(data) - esta é a única função que é chamada diretamente pela página HTML da nossa aplicação; ela é a função que chama a sendRequest para requisitar os dados do servidor e em seguida se programa para ser executada novamente a cada 30 segundos, por isso, a cada período de 30 segundos o arquivo de dados é requisitado novamente e as notícias contidas nele são verificadas pela função parseNews.

sendRequest(url,params,HttpMethod) - função responsável por requisitar os dados do servidor de forma assíncrona ao servidor; para isso, ela instancia um objeto XMLHttpRequest utilizando a função initXMLHttpRequest e depois configura a funcão onReadyState como callback da chamada.

initXMLHttpRequest() - esta função existe apenas devido a incompatibilidade entre os browsers. Como nos dois mais usados o objeto XMLHttpRequest possui diferenças na implementação, essa função testa a existência desses objetos e retorna qual foi encontrado para função sendRequest utilizá-lo.

onReadyState() - função configurada como callback na sendRequest. Ela é chamada a cada mudança no estado da requisição e enquanto este estado não for igual a READY_STATE_COMPLETE ele configura uma mensagem escrito "loading..."na página. Quando a requisição estiver completa, a mensagem de loading é retirada e chama-se a função parseNews() para tratar os dados recém chegados.

newDetect(titulo) - dado um título, esta função verifica se já existe alguma notícia com este título inserida na página. Se houver retorna true, caso contrário retornar false.

parseNews() - chamada para tratar os dados quando eles chegam do servidor. Essa função primeiro recupera a data e o título das notícias e os junta formando assim o título que será colocado na página, após isso ela chama a função newDetect para verificar se esse título já se encontra na página, se for retornado false o notícia é inserida na página, caso contrário nada será feito, pois se trata de uma notícia antiga.

getNodeValue(obj,tag) - esta função serve apenas para recuperar o valor de um atributo de um nó DOM, ela foi implementada apenas para facilitar esse processo e evitar linhas de comando muito grandes na função parseNews.

loadingMsg(set) - controla a exibição/remoção da mensagem de loading na página. Se set for 1 a mensagem é colocada, para outro valor a mensagem de loading é removida, se existir.

6 Ferramentas

Inicialmente, quando as tecnologias para *web* surgiram, programar para *web* utilizando JavaScript era extremamente penoso, pois não haviam ferramentas para auxílio de *debugging* e os próprios interpretadores muitas vezes não indicavam qualquer tipo de erro para o usuário. O resultado de um erro era muitas vezes uma tela branca no *browser* e nenhuma dica de onde o erro poderia ter ocorrido!

Com o surgimento das técnicas Ajax e a chamada *Web 2.0*, o uso da linguagem cresceu bastante e vem se tornando cada vez mais popular. Isso trouxe diversos benefícios, pois começaram a surgir consoles melhores, *debuggers*, ferramentas para auxiliar a documentação e até mesmo uma IDE (*Integrated Development Environment*) para desenvolvimento de aplicações JavaScript.

Nas próximas seções, vamos indicar apenas algumas das ferramentas muito utilizadas quando se trabalha com JavaScript.

Firebug

Firebug é uma extensão para o *browser* Firefox que mostra um console muito completo. Ele é capaz de mostrar erros de JavaScript, XML e CSS. Além disso, ele guarda um registro das chamadas ao XMLHttpRequest, o que o torna uma boa ferramenta para *debugging* de aplicações AJAX.

Com Firebug é possível colocar *breakpoints* ao longo do código, observar variáveis específicas, alterar tanto o JavaScript quanto CSS e XML em tempo de execução e ver o resultado naquele momento. Também pode ser usado para mostrar a árvore DOM de uma página.

Na Figura 3 pode-se observar as abas que contêm o Console, o código HTML, CSS, o script em JavaScript, a árvore DOM e por último, a guia Net, que mede quanto tempo foi necessário para carregar cada uma das requisições de dados feitas pela página.

Os aspectos que chamam atenção sobre o aplicativo Firebug é que ele é muito simples de usar, são necessários apenas alguns poucos minutos para se familiarizar com sua interface e com algum tempo de uso, podemos descobrir que ele é uma ferramenta realmente completa e poderosa.

Para usuários de outros *browsers*, o Firebug possui uma versão *lite* que pode ser usada em qualquer *browser*. Essa versão consiste em um arquivo na própria linguagem JavaScript que deve ser incluído na página que se deseja utilizá-lo.

A extensão Firebug se encontra na versão 1.05 e pode ser encontrada em www.getfirebug.com. Recentemente, tivemos a notícia de que a Yahoo! dedicará um de seus engenheiros para auxiliar o autor da ferramenta a desenvolvê-la.

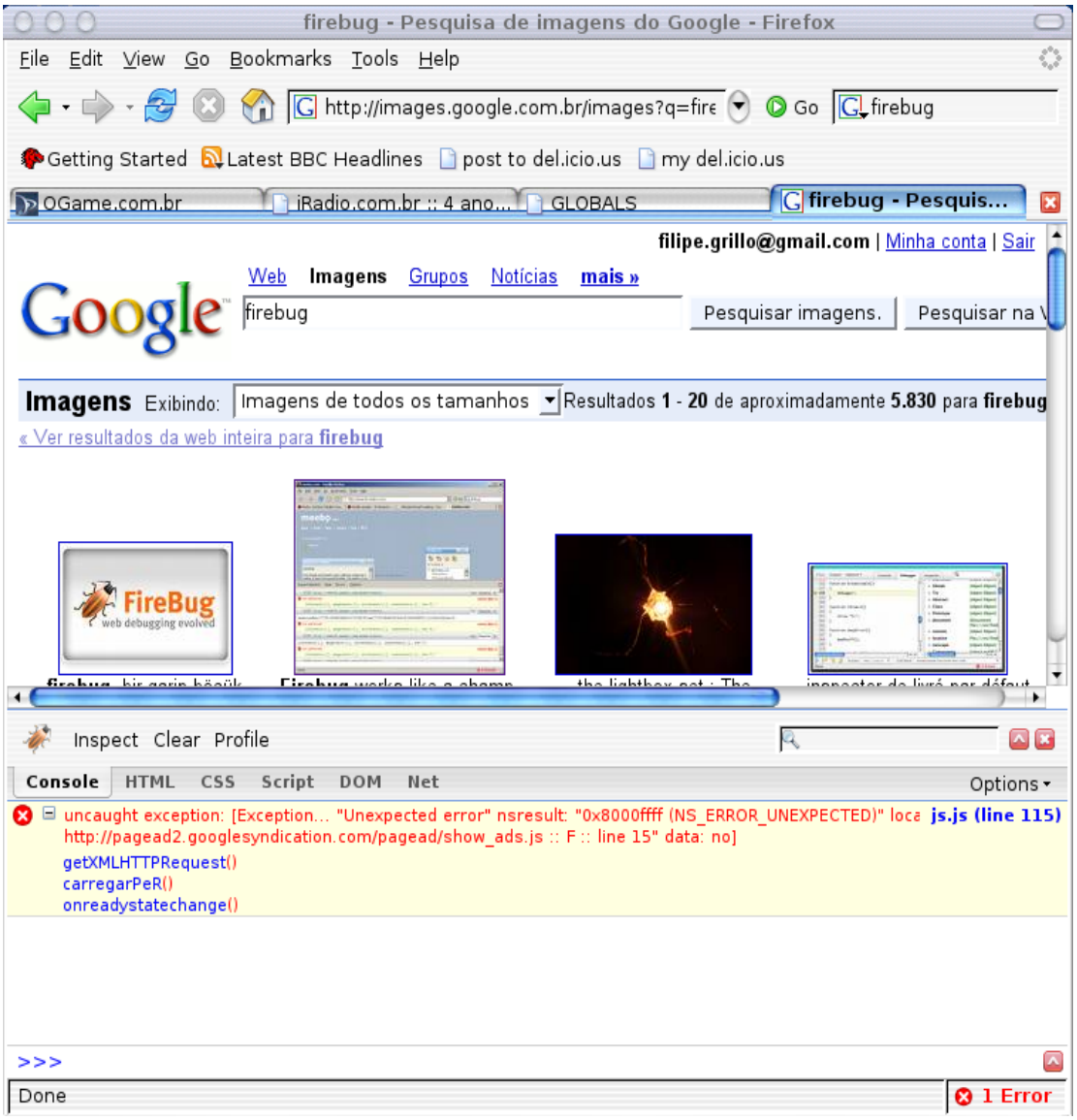

Figura 3: Screenshot do FireBug detectando uma exceção não tratada no Google Images

JSDoc

JSDoc é uma ferramenta implementada em Perl que faz busca por comentários em arquivos JavaScript e gera documentos HTML com toda a documentação. Ela é baseada nos mesmos comandos da ferramenta javadoc, utilizada para gerar documentação para Java.

Usuários que já estão familiarizados com javadoc não terão nenhum problema para usar o JSDoc, pois a maior parte da sintaxe utilizada para os comentários é a mesma e ela serve tanto para documentar arquivos JavaScript estruturais quanto orientados a objeto. Na Figura 4 vemos uma amostra de como são as páginas geradas pelo aplicativo.

A página da ferramenta pode ser encontrada em http://jsdoc.sourceforge.net

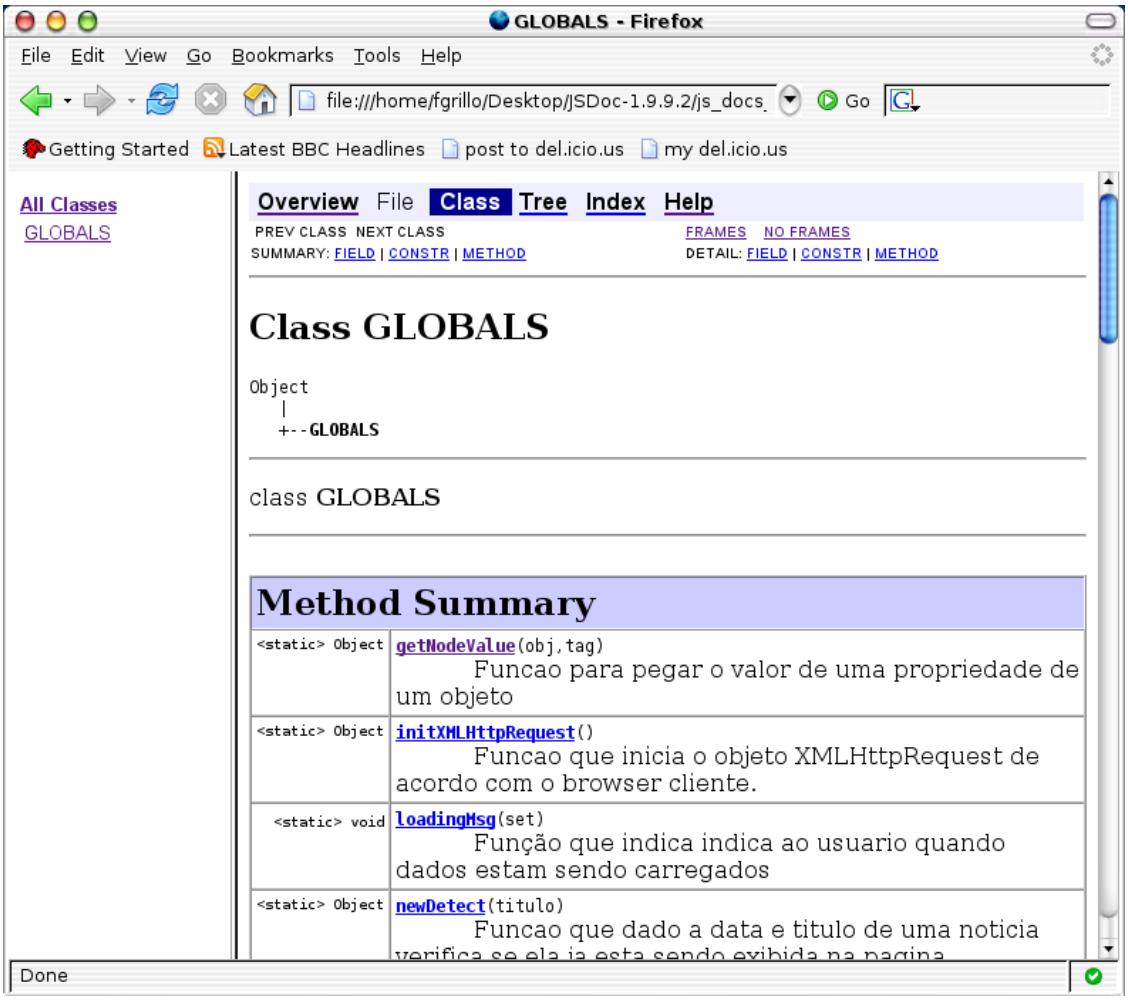

Figura 4: Screenshot do JSDoc mostrando o página de documentação gerada a partir do arquivo ajax.js mostrado no nosso exemplo de aplicação Ajax

Spket IDE

Spket é uma IDE para desenvolvimento de JavaScript, SVG, XUL/XBL e *Yahoo! Widget*. Ele oferece recursos como completar o código enquanto digitamos e destaque visual da sintaxe da linguagem. Podemos encontrá-lo em duas versões, como plugin para o editor Eclipse ou como uma plataforma independente (RCP).

A versão atual é 1.5.9 e pode ser encontrada em http://www.spket.com. Na Figura 5 podemos ver sua tela de edição.

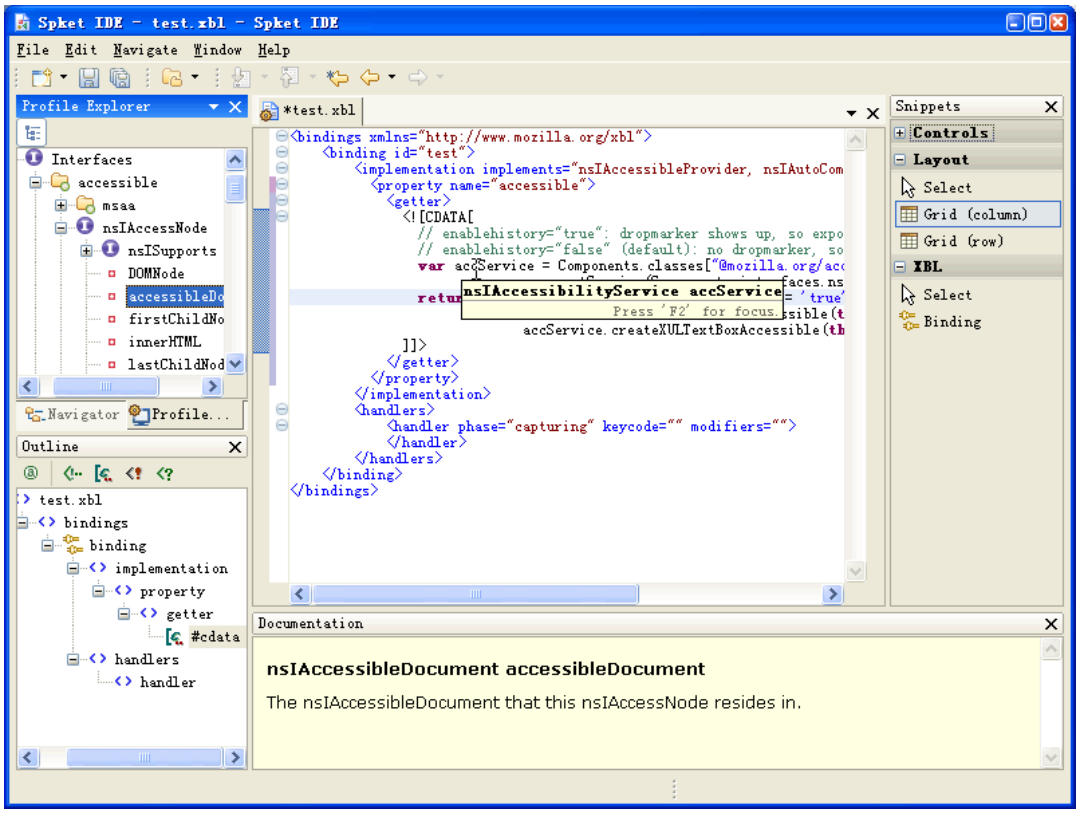

Figura 5: Screenshot da IDE Spket mostrando a edição de um arquivo do tipo xbl

Referências

[1] Douglas Crockford. Javascript: The world's most misunderstood programming language, 2001. Disponível em http://javascript.crockford.com/javascript.html , Último acesso em 10/01/2007.

[2] Jesse J. Garrett. Ajax: A new approach to web applications, 2005. Disponível em http://www.adaptivepath.com/publications/essays/archives/000385.php , Último acesso em 04/06/2007.

[3] Gavin Kistner. Object-oriented programming in javascript, 2003. Disponível em http://phrogz.net/JS/Classes/OOPinJS.html , Último acesso em 10/01/2007.

[4] Mozilla Development Center MDC. Core javascript 1.5 guide. Disponível em http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide , Último acesso em 10/01/2007.

[5] Wikipedia the free encyclopedia. Javascript. Disponível em http://en.wikipedia.org/wiki/Js , Último acesso em 13/01/2007.

[6] w3 schools. Javascript tutorial. Disponível em http://www.w3schools.com/js/default.asp , Último acesso em 16/01/2007.