JsObjects

Web Application GUI Library

Autor Pablo Santiago Sánchez

Índice

parte I

Introdução

Sobre este documento

Esta é a primeira versão da documentação oficial do JsObjetcs, escrita originalmente em Português pelo autor da biblioteca, Pablo Santiago Sánchez.

A intenção é apresentar a biblioteca aos desenvolvedores e técnicos do mundo da informática que desejarem utilizá-la para escrever seus aplicativos web utilizando-se de uma interface amigável ao usuário de sistemas desktop, sem requerer para tal a instalação de uma série de aplicativos e bibliotecas adicionais, como ocorrem normalmente.

Para tal o documento foi dividido em várias partes, segundo um grau crescente de conhecimento.

Na primeira parte (introdução) fala-se um pouco sobre a documentação e seu autor (totalmente dispensável, hehehe).

Na segunda parte, faz-se uma apresentação um pouco mais detalhada, porém ainda muito superficial, do que é a biblioteca JsObjects, com o objetivo de mostrar ao leitor o que este poderá esperar da biblioteca.

Na terceira parte faz-se uma pequena revisão de Javascript, dado o fato desta ser a linguagem utilizada para a criação da biblioteca.

Na quarta parte falamos sobre o DOM (Document Object Model) conforme definido pela W3C, e graças ao qual a biblioteca existe e é multiplataforma.

Na quinta parte revemos conceitos de Programação Orientada a Objetos e mostramos as limitações inerentes do Javascript neste aspecto.

Na sexta parte analisamos como é feito o desenvolvimento web segundo o modelo tradicional, com páginas HTML servindo como interfaces a banco de dados.

Finalmente na sétima parte, veremos como funciona o desenvolvimento utilizando o JsObjects, com exemplos concretos de uso da biblioteca.

A oitava parte é a referência completa da biblioteca e seus componentes.

Espero que vocês gostem da leitura. Quaisquer críticas, sugestões, dúvidas e, especialmente, elogios ;-) enviem um e-mail para phackwer@gmail.com que eu terei prazer em responder-lhes.

Um abraço,

Pablo Santiago Sánchez
phackwer@gmail.com

Quem deve ler este texto?

Este texto foi escrito tendo em vista pessoas que já tenham conceitos bem formados em lógica de programação, e já tenham alguma experiência com desenvolvimento web. Conhecimento em alguma linguagem de script server-side - como PHP, ASP ou JSP – são bem vindos, pois tais linguagens não serão cobertas por esta documentação, sendo mostradas apenas como exemplos da comunicação do aplicativo cliente (interface JsObjects) com o aplicativo servidor.

Caso você não se encaixe em nenhum dos casos citados acima, meu conselho é:

1. Aprenda HTML
2. Aprenda lógica de programação
3. Aprenda Javascript
4. Aprenda conceitos de orientação a objetos
5. Aprenda DOM
6. Escolha uma linguagem server-side e aprenda-a (preferência por PHP ou JSP)
7. Aprenda Modelagem de Dados
8. Especialize-se em um SGDB (preferência por PostgreSQL ou MySQL)
9. Volte a este documento

Caso você já tenha programado com alguma biblioteca de interface gráfica como Gtk ou QT, não encontrará maiores dificuldades em utilizar esta biblioteca.

Uma boa codificação para vocês e até logo.

Sobre o Autor

Olá! Meu nome é Pablo Santiago Sánchez. Embora não tenha nenhuma formação superior na área de Computação, estou envolvido a bastante tempo com programação. Sempre fui um auto-ditada, e desde muito cedo comecei a "brincar" com computadores. Meu primeiro programa foi feito em Basic, em um Apple II quando eu tinha 8 anos. Nessa época programar era fácil, e todo o PC vinha junto com um livro de como programá-lo e descrevendo cada um dos componentes internos do computador.

O tempo foi passando e informática virou para mim um hobby. Em 1993 tive meu primeiro contato com a Internet pelo provedor Nutecnet (que virou ZAZ que virou Terra), e senti que estava ali uma possibilidade para meu futuro. Eu precisava saber como fazer aquelas páginas! À época, não havia uma documentação tão vasta quanto hoje, mas também os recursos do HTML eram bem mais limitados. Estou falando de páginas com fundo cinza. Nesse mesmo ano, conheci outros sistemas operacionais que não o Windows, mas eram ainda muito crus, e eu não sabia nada de Unix. Estou falando do FreeBSD e do Linux, à época extremamente imaturos.

Quando somos crianças cometemos vários erros, e na época de escolher o curso que faria na faculdade – era 1995 – cometi o erro de cursar Direito. No segundo semestre do curso, uma outra carreira me chamou atenção: Filosofia. Comecei a cursar na UnB, concorrentemente ao Dirento, cursado na AEUDF. Porém, eu também precisava arrumar um emprego, ter meu próprio dinheiro, e tudo que eu sabia fazer era relacionado à informática. Nesta época eu já havia montado 2 redes, uma das quais sou administrador até hoje, e conhecia o suficiente de HTML para conseguir emprego em um pequeno provedor da cidade, chamado BRNet. Infelizmente estudar em uma universidade federal e ter um emprego parecem duas tarefas totalmente incompatíveis. Acabei largando a Filosofia. O ano era 1997.

Como meu emprego era meio período e eu estive trabalhando basicamente em um mini portal (não existia ainda o conceito de portal no Brasil, e o Terra ainda era o ZAZ) despertou em mim um interesse pelo jornalismo. Começo então outra faculdade: Comunicação Social – Jornalismo no UniCEUB. O ano agora é 1998.

Acontece que, graças à minha inconstância em decidir uma carreira, nunca parei para perceber que estive o tempo todo trabalhando com programação, seja codificação ou seja visual: sempre programação.

Saí da BRNet no mesmo ano que entrei no jornalismo, e permaneci 6 meses desempregado fazendo bicos aqui e ali, sendo convidado depois a trabalhar no IPHAN – Instituto do Patrimônio Histórico e Artístico Nacional - como webmaster. Enquanto estive lá, ajudei a colocar no ar a primeira versão do banco de dados de bens roubados. Embora o código já não exista mais, o lay-out criado por mim continua. :-D

Após 6 meses, saí novamente do emprego e fui trabalhar no IBTI – Instituto Brasileiro de Tecnologia da Informação. Lá desenvolvi sistemas web utilizando interface Flash. O ano era 2000. Em 2001 a empresa encerrou suas atividades. Comecei então a trabalhar como webmaster do site do jornalista Cláudio Humberto, onde permaneço até hoje.

Após dois anos trabalhando em casa, fui convidado a trabalhar na Aker Security Solutions, empresa especializada na área de segurança de redes, e única desenvolvedora nacional de firewall. Lá permaneci de 2002 a 2004, quando então fui convidado a trabalhar no Ministério das Cidades como Coordenador de Desenvolvimento de Sistemas e Administração de Dados. O Ministério encontra-se atualmente em fase de implementação de soluções em Software Livre, e me vi forçado a encontrar uma solução para desenvolvimento via web que funcionasse em redes multiplataforma. Minha experiência na Aker me permitiu perceber que seria possível criar uma biblioteca de interface com comportamento similar à aplicações desktop mas que funcionasse em qualquer navegador que aderisse aos padrões da w3c. Como tal browser não existe, limitei-me a criar uma biblioteca que chegasse o mais próximo o possível das especificações definidas por mim. Nasceu então o JsObjects. Atualmente estou trabalhando na ENAP, Escola Nacional de Administração Pública, como Gerente de Desenvolvimento de Sistemas. Academicamente, ainda estou cursando o Jornalismo – extremamente prejudicado por trabalho – e curso também Análise de Sistemas no UniCEUB.

Só para constar: nasci na cidade de Córdoba, província de Córdoba, Argentina, no dia 18 de agosto de 1978, e moro no Brasil desde os 7 meses de idade, sendo naturalizado

brasileiro. Não gosto de futebol, não vejo sentido em 90 minutos de homem suado correndo na tela. Não torço nem pelo Brasil nem pela Argentina: para mim, o que vier é lucro :-D (Tá com inveja? Vira argentino! Hehehe).

Resumindo meu currículo, tenho experiência nas áreas tanto de desenvolvimento (C++, PHP, Java, javascript, SQL, etc) quanto de administração de redes (Firewalls, VPNs, servidores Unix e Windows, servidor de e-mails, de arquivos, etc) , além de bons conhecimentos em aplicativos da indústria gráfica, como o Blender, GIMP, Photoshop, Corel Draw, Flash, etc (sempre gostei de desenhar e adoro computação gráfica). Sempre fui auto-didata, e estou estudando atualmente gerenciamento de projetos, área na qual pretendo conseguir minha certificação PMP em breve.

parte II

O que é JsObjects?

Apresentação
A origem das interfaces gráficas
Porque criar uma interface web com comportamento desktop?
Entendendo os requisitos para trabalhar com JsObjetcs
Requisitos Teóricos do Desenvolvedor
Requisitos de Software
Requisitos de Hardware

Apresentação

O JsObjects é uma biblioteca para a criação de sistemas que utilizem-se de browsers para acesso à sua interface. Esta biblioteca tenta simular o comportamento de componentes encontrados em várias outras bibliotecas para criação de aplicativos, como o QT da TrollTech (de onde vem a KDE), o GTK do Gimp (de onde vem o Gnome), a MFC (Microsoft Foundation Classes), etc.

Funcionalmente, o JsObjects tenta se aproximar mais do QT que qualquer outra das bibliotecas citadas.

O motivo do nascimento desta biblioteca é dar aos programadores de aplicativos web, um conjunto completo de objetos que não requeiram um esforço em repensar formas de aceder a funções de um sistema.

Com as tecnologias atuais de desenvolvimento web (formulários), o maior problema que encontramos é o fato de que o funcionamento de uma determinada tela nem sempre é intuitivo, e seria muito mais fácil para o usuário se este depara-se com comportamentos ja conhecidos de aplicativos, como o caso de menus contextuais, menus com submenus, treeviews, grids, etc.

Outras ferramentas com o mesmo objetivo podem ser encontradas na internet, porém falta a elas unicidade e facilidade de uso. Cada objeto possui métodos próprios e sem similaridade uns com os outros (não existe polimorfismo), forçando o programador a decorar milhões de formas de trabalhar com cada um dos objetos. O JsObjects tenta, no limite do possível, eliminar esse problema. Todos os objetos interativos (combobox, listbox, checkbox, button, lineedit, ,dateedit, etc) possuem praticamente os mesmos métodos para acedê-los, tais como: setEvent, unsetEvent, setValue, getValue, addItem e delItem. Existe ainda um segundo tipo de objeto, criado para a construção visual do aplicativo, chamados de Containers, os quais possuem basicamente os métodos addItem e delItem. Eventualmente, um container é também um objeto interativo, como o caso do JsListView, que contém vários JsListViewItens, ou o JsMenu, que contém JsMenuItens.

Um outro ponto positivo do JsObjects é o fato da mesma ser multiplataforma, sendo executada sem diferenças em ambientes Unix ou Windows, sendo acessível tanto pelo browser Mozilla/Firefox, quanto pelo Internet Explorer. Graças à padronização da W3C para o DOM, a biblioteca provavelmente será compatível com outros browsers no futuro sem problemas.

Embora seja um projeto ambicioso, o JsObjects é um projeto que vem para facilitar sua vida, e permitir que finalmente os aplicativos web possam concorrer com aplicativos desktop ao oferecer os mesmos recursos.

A origem das interfaces gráficas

A idéia da interface gráfica para interação com sistemas computacionais surgiu pela primeira vez em 1945, quando Vannevar Bush publicou um artigo que inspiraria Douglas Engelbart a criar o mouse em 1962. Um ano depois Ivan Sutherland escreveu a tese entitulada “” que definiria vários dos fundamentos para a construção de um interface gráfica, baseado no conceito psicológico de que o cérebro humano entende melhor ícones a texto.

A primeira interface gráfica realmente utilizável surgiu apenas em 1974, nos laboratórios da Xerox em Palo Alto. O projeto porém não foi bem aceito pela diretoria, e anos depois, em 1979, Steve Jobs, super “bambambam” da Apple, em um acordo de cooperação com a Xerox tem acesso à essa tecnologia, e tempos depois lança o primeiro computador pessoal com interface gráfica: o Apple Lisa. Mais tarde a Microsoft utilizaria-se de artimanha similar para criar o Windows.

Este fato é retratado no filme “Piratas do Vale do Silício” (“Pirates of Silicon Valley”, TNT, 1995). Observe que o nome do filme é “Piratas”, e não “Hackers”, fazendo uma clara alusão ao roubo de conceitos como uma forma de pirataria, hoje em dia bastante combatida. Lembram-se que a Microsoft patentou o duplo-clique? Cuidado para não pensar como os outros, ou você pode ser obrigado a pagar por pensar. A polícia do pensamento de George Orwell (“1984”) não está tão longe, chegou apenas com 20 anos de atraso.

Porque criar uma interface web com comportamento desktop?

Certa vez me perguntaram porque criar uma interface para web com comportamento desktop. As vantagens de uma sobre a outra são tantas que fica complicado explicar todas. Mas, vamos tentar.

1. Oferta centralizada do aplicativo – Ao colocar o aplicativo em um servidor web, você está garantindo a todos os funcionários da empresa acesso simplificado e facilitado ao mesmo simultaneamente.
2. Dispensa instalação – A maioria dos sistemas operacionais hoje em dia já vem com algum browser web compatível com o JsObjects
3. Facilita a atualização – Um dos grandes problemas ao desenvolver um aplicativo em constante desenvolvimento ("O mundo não pára... não pára, não, não pára!" - Cazuza) é a questão de manter as versões em desktop atualizadas. Ou se cria um sistema para atualização automática, roubando banda de sua rede no dia da alteração, ou gasta-se muito tempo atualizando cada uma das estações manualmente, o que em uma rede grande (500-n computadores) é um tarefa inviável e desaconselhável, que ainda pode incorrer em erros como estações acidentalmente sem atualização.
4. Mutiplataforma – poucas bibliotecas de interface permitem que você utilize o mesmo código em multiplos sistemas operacionais. Com a Web e JsObjects essa barreira é rompida, permitindo ao desenvolvedor programar apenas uma vez.

É claro que nem tudo são flores. Existem desvantagens em utilizar-se interfaces web, conforme ilustrado agora

1. Interfaces web são mais lentas – é verdade. Elas são mais lentas. Isso ocorre porque não existe uma preocupação por parte dos desenvolvedores de browser em criar uma forma rápida de interpretar e executar scripts, justamente porque as linguagens de scripts surgiram para realizar pequenas tarefas do lado do cliente, tais como a pré-validação dos dados de um formulário. Além disto, existe o fato de que a web é historicamente lenta. Nada ocorre em tempo real. Nunca houve a preocupação de executar um display rápido ou um script rápido, pois a página sempre foi carregada lentamente.
2. Tem certas coisas, que só o desktop faz para você – também é verdade. A interação com o sistema de arquivos do cliente só pode ser feito de duas formas: um campo input do tipo file para realizar upload ou através do download de um arquivo a ser salvo no disco rígido. Os browsers por padrão e segurança não permitem ao script acessar o disco do sistema cliente. O trabalho com imagens e gráficos também não é possível, a menos que se utilizem plugins, ou seja realizado do lado do servidor e retransmitido ao cliente. Enfim, é difícil estar trabalhando com imagens do lado do cliente, bem como manipular arquivos localizados no cliente, através de interfaces web.
3. Relatórios feitos na Internet não são bons para imprimir – mito! Isso depende na verdade da opção do desenvolvedor em utilizar-se de uma biblioteca para gerar os relatórios em formatos de impressão fixa, como o PDF. Eu gosto sempre de falar sobre este ponto poisainda existem pessoas que desconhecem este fator. Como programador PHP, eu uso sempre a biblioteca FPDF (http://www.fpdf.org), uma das melhores atualmente disponíveis e livres.

Como exposto acima, na hora de se decidir por qual tipo de interface utilizar, o desenvolvedor deverá levar em conta não apenas as facilidades de uso do sistema, mas também os requisitos do mesmo. As interfaces web serão sempre limitadas pelos browsers e plugins existentes ou criados.

Entendendo os requisitos para trabalhar com JsObjetcs

Embora seja um biblioteca para a web, o JsObjects é uma ferramenta sofisticada, que requer mais que conhecimento sobre HTML apenas. Na verdade, o conhecimento sobre HTML é totalmente dispensável no uso do JsObjects.

O nome JsObjects vem de Javascript Objects (Objetos Javascript). Ela é portanto, uma coleção de objetos criados utilizando-se do DOM e da linguagem Javascript. Isto requer do desenvolvedor conhecimentos pelo menos sobre os conceitos de Orientação a Objetos, o DOM da W3C, e o uso da linguagem Javascript. Além disto, é requerido que o mesmo conheça alguma linguagem de programação como PHP, JSP ou ASP para a programação do lado do servidor.

Em relação ao software, no tempo desta documentação a biblioteca era compatível apenas com (caso você tenha sucesso com outro, favor avisar):

1. Mozilla, versão 1.6 ou superior;
2. Firefox, versão 0.8 ou superior; e
3. Internet Explorer, versão 6.0 ou superior.

Em relação ao hardware, desconheço os requisitos mínimos. A biblioteca foi desenvolvida utilizando-se um notebook com processador Celeron de 2.0 Ghz e 512MB de RAM, e um desktop com processador pentium 4 de 2.4 Ghz e 640MB. Não houve diferença de desempenho de um para outro durante o desenvolvimento. Tudo que posso lhe dizer é: quanto mais memória e processador você tiver, melhor, pois à medida que a interface vai ficando mais complexa, maior a quantidade de recursos necessários da máquina.

parte III

Programando com Javascript

Histórico
Delimitadores
Comentários
Variáveis
Operadores
Estruturas de Controle
Funções

Histórico

O Javascript surgiu como uma opção para o programador web no Netscape. Com seu surgimento, o programador poderia agora passar a tarefa de processamento de um formulário (validação) para o computador cliente, evitando uma sobrecarga desnecessária no servidor.

Originalmente chamado Hotscript, o Javascript mudou de nome quando a Netscape fez um acordo com a Sun para o desenvolvimento da linguagem, e utilizou o java como base para o mesmo.

Com o sucesso da linguagem, viu-se a necessidade de padronizá-la pois algumas variantes já começavam a surgir e páginas web dinâmicas passaram a mostrar problemas quando visualizadas em diversos browsers. A W3C assumiu a tarefa, renomeando a linguagem para ECMAScript, embora até hoje, exista apenas um suporte parcial nos browsers a estes padrões.

Delimitadores

Quando criando uma página com conteúdo dinâmico do lado do cliente, o código Javascript deve ser delimitado, isto é, seu início e seu fim devem ser demarcados para não misturar-se o HTML. Os delimitadores padrões são:

`<script>` para marcar o início; e
`</script>` para marcar o fim.

Uma prática comum é comentar o código com os delimitadores de comentário HTML, logo após declarar o início do script e antes do fim, conforme abaixo

```
<script>
<!--
-->
</script>
```

Isto é porém uma prática arcaica e caiu em desuso, pois quase todos os browsers hoje em dia dão suporte ao Javascript.

Uma outra prática que pode ser utilizada é declarar a versão da linguagem sendo utilizada, como em:

```
<script language=”javascript 1.5”>
</script>
```

Esta também caiu em desuso, pois o único browser que aceita mais de uma linguagem além do Javascript é o Internet Explorer, que aceita o VBScript, linguagem que eu realmente desaconselho a ser utilizada, não por que seja ruim ou mal feita ou porque eu não gosto da Microsoft, mas porque ela tornará seu site compatível comum número mais reduzido de computadores.

Separador de instruções

O Javascript é composto por um série de instruções. Por padrão, uma linha de instrução só deveria ter fim quando encontrado um ponto-e-vírgula (`;`) no código. Porém, por ser uma linguagem extremamente flexível, convencionou-se também que a quebra de linha (`\n`) indica também o fim da instrução.

Um exemplo de instrução seria o abaixo:

```
alert('asdf');
```

O ponto e vírgula permite que eu coloque mais de uma instrução em uma linha, embora esta prática seja desaconselhável mas válida, como no seguinte exemplo:

```
alert(1);alert(2);alert(3).
```

Como um bom hábito de programação, indique sempre o final de suas linhas com o ponto-e-vírgula, e sempre deixe apenas uma instrução por linha. O código fica mais limpo e organizado.

Comentários

Como toda a linguagem de programação que se preze, o Javascript permite que você comente seu código. Comentários são muito úteis, posto que auxiliam você a entender o código ou que outros entendam seu código, ou ao menos saibam o que está acontecendo naquele ponto.

Os comentários Javascript são como os comentários em C, e podem ser de uma linha (iniciados com `//`) ou de várias linhas (todo o texto localizado entre `/*` e `*/`), como no exemplo abaixo:

```
//comentário de uma linha

/*
comentário de vááááárias linhas
comentário de vááááárias linhas
comentário de vááááárias linhas
comentário de vááááárias linhas
;-)
*/
```

Variáveis

Novamente, como toda a boa linguagem de programação, o javascript permite que você utilize uma variável. Uma variável é um espaço na memória que guardará o valor que lhe foi atribuído sendo este valor posteriormente acessado pelo nome da variável. O valor atribuído a uma variável tem um tipo específico, que delimita o tipo de operações que poderam ser realizadas com ela. A nomenclatura das variáveis deve sempre iniciar com um valor alfabético ou o símbolo _, e após o primeiro caracter, pode-se utilizar quaisquer caracteres alfanuméricos. Ex: var1, var2, _asdf, qwer, mas não pode ser 1var. É desejável também que sempre se utilizem nomes mnemônicos, que lembrem a utilidade ou valor da variável.

Os tipos de uma variável podem ser:

1. Números inteiros
2. Números de ponto flutuantes
3. Strings
4. booleano (verdadeiro ou falso)
5. Arrays de mútiplos tipos
6. Objetos do javascript
7. Objetos customizados

Javascript é, porém, uma linguagem fracamente tipada, aceitando que as variáveis tenham seus tipos modificados on-the-fly através da coersão. Este assunto será visto apenas mais à frente, após termos uma idéia melhor sobre a sintaxe da linguagem.

Em Javascript, é obrigatório declarar as variáveis antes de utilizá-las. São várias as formas válidas de declaração de uma variável, como mostrado abaixo:

```
var nomedavariavel;
nomedavariavel = 1;
outravariavel = 2; /*esta variável é criada automaticamente,
                     on-demand, sem eu tê-la declarado
```

```
                    anteriormente*/
maisumavariavel;    /*esta variável é criada automaticamente,
                    on-demand, sem eu tê-la declarado
                    anteriormente, mas seu valor é nulo (null)*/
```

Inteiros

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

```
var1 = 1234;     # inteiro positivo na base decimal
var2 = -234;     # inteiro negativo na base decimal
var3 = 0234;     # inteiro na base octal-simbolizado pelo 0
                 # equivale a 156 decimal
var4 = 0x34;     # inteiro na base hexadecimal(simbolizado
                 # pelo 0x) - equivale a 52 decimal.
```

A diferença entre inteiros simples e `long` está no número de bytes utilizados para armazenar a variável. Como a escolha é feita pelo interpretador de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

Ponto flutuante

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

```
var1 = 1.234;
var2 = 23e4;      # equivale a 230.000
```

Strings

As Strings são um tipo especial de variável. Na verdade, ela não é um tipo padrão, mas um dos tipos de Objetos Javascript disponíveis para o programador. Por ser um Objeto, ela possue métodos e atributos próprios utilizados para manipulá-la. Veremos mais sobre os métodos da String quando falarmos sobre Orientação a Objetos. Internamente, a String é um objeto que contém como valor um Array de caracteres, como veremos mais à frente.

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (')

b)utilizando aspas duplas (")

Qualquer uma das duas estará correta, conforme mostrado no exemplo abaixo:

```
var1 = 'Minha String';
var2 = “Minha outra String”;
```

Existe um grupo especial de caracteres que podem ser colocados dentro de uma string, para representar caracteres invisíveis, como as quebras de linha. Esse grupo de caracteres é conhecido como caracteres de escape, e são os listados abaixo:

Sintaxe	***Significado***
\n	Quebra de linha
\r	Retorno de carro
\t	Tabulação
\\	A própria barra
\'	Aspas Simples
\”	Aspas Duplas

As aspas simples e aspas duplas são usadas quando os delimitadores da string forem iguais ao que se precisa inserir na variável, como no exemplo abaixo:

```
var1 = “Minha String \”tem aspas duplas\””;
var2 = 'Minha outra String \'tem aspas simples\'';
```

Booleanos

Booleanos são variáveis com valores binários, ou seja, verdadeiro ou falso (true or false, 1 ou 0) e são bastante utilizadas no retorno de funções que executem uma validação, por exemplo. Seu uso é simples:

```
var1 = true;
var2  false;
```

Arrays

O Array é um outro tipo de Objeto do Javascript, e como todos os objetos possue atributos e métodos próprios. Conceitualmente, um Array poderia ser comparado a uma lista indexada de valores identificados por um único nome.

Seu uso é simples e pode ser feito conforme ilustrado abaixo:

```
arr = new Array();
arr[0] = “Valor 0”;
arr[1] = “Valor 1”;
arr[2] = “Valor 2”;
arr[3] = “Valor 3”;
arr[4] = “Valor 4”;

alert(arr.length);      //atributo do array que me indica
                        //quantos elementos existem dentro
                        //dele.
```

Embora este array tenha comprimento de 5 elementos, sua contagem inicia em 0 sempre. Este é um dos pontos mais fáceis de cometemos erros quando programando, pois nós estamos acostumados a contar a partir de 1 e não a partir de 0.

Quando afirmamos que uma String é um tipo derivado do Array, é porque uma String sempre é tratada como um array de caracteres. Ex:

```
str = “Teste”;
alert(str[2]);          //irá abrir um alert escrito “s”
alert(str.length);      //irá abrir um alert escrito “5”
```

Objetos do Javascript

as

Objetos Customizados

as

Conversão de tipos

Os tipos das variáveis (Inteiros, ponto flutuante, e String) podem ser convertidos em tempo de execução. Isto é feito através da utilização de funções do Javascript específicas para isto, conforme mostrado abaixo:

```
str = “123.456 é um número legal”;
var1 = parseInt(str);   //var1 é igual a 123
var2 = parseFloat(str)  //var2 é igual a 123.456
var3 = var1 + “”        //var3 é igual à string 123
```

Operadores

De nada adiantaria termos variáveis e valores, se não pudéssemos manipulá-los. Como calcular o total de uma compra sem multiplicar o valor do produto pela quantidade? Os operadores em linguagem de programação, porém, ultrapassam os matemáticos conhecidos durante o segundo grau. E são os operadores da linguagem que veremos agora. Novamente, seu uso será visto apenas mais à frente.

Aritméticos

Operador	***Significado***
+	Adição
-	Subtração
*	multiplicação
/	divisão
^	potência
%	módulo

Atribuição

Operador	***Significado***
=	atribui
+=	Soma (concatena*) e atribui
-=	Subtrai e atribui
*=	Multiplica e atribui
/=	Divide e atribui
%=	Módulo e Atribui

*depende do tipo das variáveis

Comparação

Operador	***Significado***
==	Igual a
!=	Diferente de
!	FALSO
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Bit a Bit

Operador	*Significado*
&	E lógico
|	OU lógico
^	OU lógico exclusivo
~	Não (inversão)
<<	Desloca um bit à esquerda
>>	Desloca um bit à direita

Lógicos

Operador	*Significado*
&&	E lógico
||	OU lógico
!	Não (inversão)

Expressão Condicional

Operador	*Significado*
(cond) ? (inst1) : (inst2)	SE lógico

Incremento e Decremento

Operador	*Significado*
++	Incremento
--	Decre

Estruturas de Controle

Além de manipularmos as variáveis, é necessário também que nosso programa seja inteligente o suficiente para decidir o que deve ser feito a seguir de acordo com o resultado das operações. Essa inteligência advêm do uso das estruturas de controle.

As estruturas de controle ou estruturas de fluxo, servem para auxiliar o programador na tomada de decisões dentro do código e na execução de operações repetitivas sobre um mesmo grupo de variáveis. Elas, literalmente, controlam o fluxo da execução do programa. Elas podem ser divididas basicamente em estruturas de decisão e estruturas de laço (ou loop).

As estruturas de decisão são o if...else e o switch...case, e as de laço são o for, o while e o do...while. Estas estruturas são ainda auxiliadas pelas palavras-chave continue e break, conforme veremos a seguir.

if...else

O *if* é a estrutura de decisão mais básica de todas, e o que ela faz basicamente é executar um conjunto de operações determinadas, se determinada condição for verdadeira. Ela permite ainda o uso do *else*, para que se execute outro conjunto de operações, no caso da expressão ser falsa. É possível ainda encadear-se uma série de *if*'s com *else*'s validando

várias condições.

A sintaxe básica é:

```
if (condição) {
        conjunto de instruções.
}
```

O uso com else é feito da seguinte forma:

```
if (condição)
{
        conjunto de instruções.
}
else
{
        conjunto de instruções.
}
```

e o encadeamento pode ser feito da seguinte forma:

```
if (condição)
{
        conjunto de instruções.
}
else if (condição)
{
        conjunto de instruções.
}
else if (condição)
{
        conjunto de instruções.
}
.
.
.
```

Como o encadeamento pode ter um tamanho infinito, tornando o código de difícil leitura e causando a repetição desnecessária de código no caso de estarmos apenas comparando o valor de uma variável, criou-se um outro tipo de estrutura de decisão, que é o switch...case. Existe ainda um operador lógico bastante utilizado no caso de decisões simples. O operador só permite porém, a execução de duas instruções, a que atende a condicional e a que não atende.

Seu uso é conforme abaixo:

(condição) ? (instrução se verdade) : (instrução se falso)

switch...case

Se você só precisa testar o valor que está dentro de uma variável, o switch...case é ideal para você. Esta estrutura é capaz de utilizar apenas uma variável e compará-la apenas a um valor. Normalmente ela utiliza-se da instrução especial break devido a uma particularidade do seu uso...

O switch...case, quando encontra um comparação verdadeira, inicía sua execução a partir daquele ponto. Se ela não encontrar um comando break, ela irá executar também o código do próximo case, e assim por diante, até o fim do switch, o que o torna uma estrutura perigosa, pois é fácil distrair-se e ver seu código tendo reações adversas, não esperados para aquele caso, mas apenas para os outros logo abaixo.

Além dos cases definidos, o switch tem também um caso especial, chamado default (padrão), que é executado caso nenhuma das comparações tenha sido verdadeira.

O switch é uma forma elegante de estar organizando seu código, mas requer atenção

redobrada em relação ao uso do if...else encadeado.

A sintaxe do switch case é conforme abaixo:

```
switch (variavel_testada)
{
	case valor_comparado1:
		instruções
		break;
	case valor_comparado2:
		instruções
		break;
	case valor_comparado3:
		instruções
		break;
	default:
		instruções
}
```

for

Das estruturas de laço disponíveis, o for é o mais básico, e é bastante utilizado para fazer a leitura de arrays. Sua sintaxe é simples e permite duas formas para seu uso, uma voltada para um laço de instruções X vezes, e a outra mais simples para a manipulação de arrays.

A sintaxe do for mais clássica vem do C, e é conforme abaixo:

```
for(valor_inicial; condiciona); incremento/decremento do valor_inicial)
{
	instruções
}
```

A sintaxe simplificada para arrays é:

```
for (i in array)
{
	instruções
}
```

while

A estrutura while executa uma série de instruções enquanto, e somente se, a condicional for verdadeira. Sua sintaxe é simples, e segue abaixo:

```
while (codicional)
{
	instruções
}
```

do...while

A estrutura do...while executa uma série de instruções enquanto a condicional for verdadeira. Porém, ao contrário da estrutura while, as instruções são executadas pelo menos uma vez. Sua sintaxe é simples, e segue abaixo:

```
do
{
	instruções
}
while (codicional)
```

continue

A palavra-chave continue, representa exatamente o que ela quer dizer: as instruções da estrutura devem continuar sendo executadas. É raramente utilizada, pois o fluxo esperado de um programa já é que ele continue a executar as instruções esperadas. Enfim, ele apenas confirma e regra.

break

Já a palavra chave break, é muito mais utilizada, por ser a contra-regra. Ela quebra o fluxo do programa naquele ponto, evitando que um loop, por exemplo, continue a ser executado, distorcendo o resultado desejado.

Funções

Embora os loops sejam perfeitos para executar uma repetição de instruções, eles não são muito bons soltos no código, pois se eu preciso executar um mesmo pedaço de código em um ponto diferente, eu teria que copiar o código original para aquele ponto. E ao fazer alguma alteração no código original, seria extremamente trabalhoso atualizar todos os pontos onde ele se repete. Para facilitar a manutenção de códigos repetidos é que surgem as funções.

As funções são blocos de código que podem ser chamados através de uma palavra que defina seu uso, definido pelo programador. Entender como e quando criar funções é o primeiro passo para uma programação bem feita e de fácil manutenção.

Uma função pode retornar um valor ou pode apenas alterar uma variável. Quando uma função termina, é sempre bom que retorne ao menos true ou false, para verificarmos se ela foi executada com sucesso.

Para fazer uma boa função, o ideal é pensar qual o objetivo da mesma, o que dentro dela poderia ser executado por uma outra função, e como eu posso torná-la o mais geral o possível para ser utilizada em diversos pontos.

Uma função que possue diversas linhas de código é, via de regra, uma função mal elaborada. O ideal é que ela seja quebrada em várias outras funções. Algumas correntes da métrica de software defendem que um bom valor para o tamanho de uma função é algo em torno de 20 a 30 linhas.

Um outro erro comum é que a função tenha parâmetros demais. Uma função com muitos parâmetros não é uma função flexível, é uma função mal pensada e de difícil manutenção, provavelmente repleta de ifs e elses mal colocados.

Só para servir de exemplo: o treeview mais simples que eu fiz tinha apenas 90 linhas de código, distribuídas entre 2 objetos (um treeview e um treeviewnode), em PHP. Não existe porém uma fórmula mágica para fazer uma boa função, isso geralmente vem apenas com a prática...

Criando funções

Para criar uma função em javascript, a sintaxe é conforme abaixo:

```
function nomedafuncao (parâmetros)
{
        return (valor retornado);
}
```

Como nas variáveis, a nomenclatura das funções deve sempre iniciar com um valor alfabético ou o símbolo _, e após o primeiro caracter, pode-se utilizar quaisquer caracteres alfanuméricos. É desejável também que sempre se utilizem nomes mnemônicos, que

lembrem a utilidade da função.

A palavra chave return determina o fim da função retornando algum valor determinado. O valor pode ser qualquer um dos tipos existentes no Javascript. Caso não seja utilizada em nenhum ponto, o javascript irá executar a função até a } de encerramento, retornando void (vazio) por padrão.

Passando parâmetros

Uma função não tem como saber o que deve fazer nem com quais valores trabalhar se ela não for informada. Isso é feito através da passagem de parâmetros. Para que uma função receba parâmetros, ela deve ser definida da seguinte forma:

```
function nomedafuncao(par1,par2,par3...)
{
	/*automaticamente, os parametros passados ficam disponíveis
	dentro da função podendo ser chamadas pelo nome definido na
	definição da função, como abaixo*/
	alert(par1);
	alert(par2);
	alert(par3);
}
```

e chamada passando os valores correspondentes

```
nomedafuncao('Valor 1','Valor 2','Valor 3',...);
```

Emulando sobrecarga de funções
Escopo de variáveis e funções
Exemplo: Validação de Formulário

parte IV

Programando Orientado a Objetos com Javascript

O que é Orientação a Objetos?
Suporte Parcial em Javascript
O que falta?
Como contornar as limitações da linguagem?
Objetos Nativos do Javascript
Objetos Customizados no Javascript

parte V

Programando Javascript com DOM

O padrão W3C
Workarounds (gatos) para seu código ser multiplataforma
Workarounds no IE
Workarounds no Mozilla/Netscape
Outros Browsers (Konqueror, Opera, etc...)

parte VI

Desenvolvimento web tradicional

Introdução
Entendendo o que é CGI
Linguagens de Script Server-Side

parte VII

Desenvolvimento web com JsObjects

Introdução
Trabalhando em Camadas
Independência de Linguagem Server-Side

parte VIII

Componentes do JsObjects – Referência Completa

Widgets

Containers
JsDocument
JsWidgetGrid
JsWidgetStack
JsTab
JsDialog

Widgets Comuns
JsLabel
JsLineEdit
JsTextEdit
JsPushButton
JsImageButton
JsCheckBox
JsRadioButton
JsComboBox
JsListBox
JsListBoxItem
JsSpinBox
JsListView
JsListViewItem
JsToolBar
JsToolBarButton
JsMenuBar
JsMenu
JsMenuItem
JsDateEdit
JsCalendar
JsCalendarView
JsTimeEdit
JsClock
JsIpEdit

Miscelânea
JsDataConnector
Funções adicionais complementares