Web Sites com Ajax

Muito se tem falado sobre Ajax nos últimos meses e, de fato, não é por menos. Com ele temos outros níveis de interfaces e interações com o usuário.

Com essa metodologia, pode-se levar à web possibilidades que antes eram possíveis somente no âmbito desktop. Muita coisa boa ainda está por vir e, inicialmente, a melhor delas é que você será capaz de trabalhar com os recursos do Ajax em seus aplicativos.

O curso abordará a metodologia Ajax (*sim, metodologia, pois Ajax não é uma tecnologia. Acalme-se, você entenderá)* e as linguagens utilizadas por tal recurso: JavaScript, DOM, DHTML entre outros assuntos.

Ao término dos estudos realizaremos um projeto final com o intuito de praticarmos o que aprendemos no decorrer do curso. Essa será a hora de colocarmos tudo em prática e conhecermos as inúmeras possibilidades que este recurso irá nos proporcionar.

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

O entendimento de todos os passos será primordial para uma boa compreensão e aproveitamento de todo o processo.

Prefácio

Neste primeiro capítulo abordaremos uma linguagem fantástica, capaz de tornar o seu site mais dinâmico e interativo. O melhor de tudo é que aprendê-la é **super fácil** e divertido. Abordaremos **o JavaScript**.

Para início de conversa, o Javascript é uma linguagem client-side, ou seja, ela trabalha (*roda, funciona*) do lado do cliente, o que significa dizer, que ela não é executada no servidor e sim no browser, ao contrário de algumas linguagens como o PHP, ASP, JSP, .NET, dentre outras.

O cliente tem total controle sobre o suporte a essa linguagem no browser utilizado, então lhe afirmo que ele pode desabilitar o funcionamento desse recurso. Assim, o que desenvolvemos fica obsoleto e inútil. Por isso trabalharemos com o **JavaScript não-obstrutivo**, ou seja, se o cliente desabilitar o funcionamento dessa linguagem ou se ele não tiver suporte a ela, ele utilizará nosso sistema normalmente, mas sem os recursos desenvolvidos com a referida linguagem.

Introdução ao JavaScript

Os principais browsers do mercado, Firefox, Mozilla, Opera, Netscape, Safari, Internet Explorer e outros mais, suportam a linguagem JavaScript. Ela é utilizada para a validação de formulários, ler e inserir conteúdo em uma página, criar cookies, menus dinâmicos e diversos outros recursos.

Inserindo o JavaScript em uma página HTML

Para inserirmos o JavaScript em uma página HTML, fazemos uso da tag **<script>** e do atributo type com o valor: **text/javascript**. O que resultaria no seguinte:

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
<script type="text/javascript">
//códigos JavaScript
</script>
```

Ao iniciarmos a tag **<script>**, dizemos o tipo de script utilizado **type="text/javascript"** e assim o browser entenderá que no conteúdo dessa tag há comandos JavaScript para serem executados e os reconhecerão.

Para finalizar a execução dos códigos usa-se o final da tag como **</script>**.

Onde inserimos o JavaScript em uma página HTML

Há três formas de inserirmos o JavaScript em uma página HTML, são elas: um arquivo externo **-.js**, no head, no body ou utilizando os três de uma só vez.

Vamos conhecer a peculiaridade de cada uma delas.

Arquivo (script) externo

O arquivo externo é chamado desta forma:

```
...
<head>
<script type="text/javascript" src="externo.js"></script>
</head>
...
```

Ou seja, através do atributo **src** da tag **<script>**, indicamos a localização do arquivo externo, para que assim o browser possa localizá-lo e executá-lo.

Em arquivos externos não utilizamos a tag **<script>**, somente os códigos.

Vamos conhecer algumas das vantagens de se trabalhar dessa forma:

Facilidade na manutenção

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Uma vez que o script está localizado em apenas um arquivo, facilita a edição ou correção dos códigos.

Carregamento mais rápido da página

O arquivo externo é armazenado no cache do navegador. Assim, evita-se carregá-lo toda vez que a página for chamada.

Semântico

O arquivo externo separa a camada de comportamento (JavaScript) da camada de conteúdo (HTML).

Script no head da página

O script é inserido desta forma:

```
...
<head>
<script type="text/javascript">
//códigos JavaScript
</script>
</head>
...
```

Os códigos JavaScript localizado no head da página são executados ao serem chamados, ou seja, quando algum **evento** for provocado, como o evento onclick, onmouseover, onload, dentre outros.

Nessa forma o script é carregando antes de alguém utilizá-lo, ou seja, antes do carregamento do conteúdo do body.

Script no body da página

O script é inserido desta forma:

```
...
<body>
<script type="text/javascript">
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
//códigos JavaScript
</script>
</body>
...
```

Os códigos inseridos no body da página são inicialmente executados enquanto a página é carregada mas também podem ser chamados quando algum evento for provocado.

A tag <noscript>

A tag **<noscript>** nos permite disponibilizar um conteúdo alternativo ao disposto via script. Isso significa dizer que, se o usuário não tiver suporte ao JavaScript em seu browser, ele verá o conteúdo inserido dentro dela e, caso tenha suporte, o seu conteúdo é omitido.

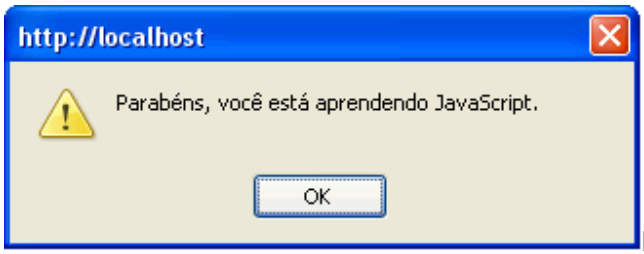
Veja um exemplo:

```
<script type="text/javascript">
alert("Parabéns, você está aprendendo JavaScript.");
</script>

<noscript>
<p>Parabéns, você está aprendendo JavaScript.</p>
</noscript>
```

No exemplo acima será exibida a mensagem, "Você está aprendendo JavaScript", através do **alert()** - *uma função do JavaScript que exibe uma mensagem através de um popup box* - para quem tem suporte ao JavaScript. Caso contrário, mesma mensagem aparecerá para quem não tem suporte ao JavaScript, porém através da tag **<noscript>** na forma de texto.

Veja as imagens:



Exemplo de uma mensagem via alert().

Parabéns, você está aprendendo JavaScript.

Exemplo de uma mensagem via <noscript>

A tag **<noscript>** é muito importante quando se trata de acessibilidade, pois oferece um conteúdo alternativo para os usuários sem suporte à linguagem em questão.

Comentários em JavaScript

Os comentários na linguagem JavaScript tem função similar às demais linguagens, ou seja, de explicar o que determinado comando faz, nota de algum autor, lembretes, dentre outras.

Tudo que estiver englobado pelos comentários é ignorado pelo interpretador do JavaScript.

Há dois tipos de comentários, são eles: ***//*** e ***/* */***. O primeiro é o comentário de uma linha somente, enquanto o segundo suporta várias.

Observe os exemplos:

```
<script type="text/javascript">
// Isto é um comentário de uma linha.

/* Isto é um comentário de várias linhas, e o
interpretador ignora todo esse conteúdo. O que nos
permite a criação de notas e lembretes em nossos
códigos. */
</script>
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Um detalhe importante que não podemos esquecer, é a utilização do comentário em HTML ao utilizar códigos JavaScript in-line (*disposto no head ou body*). Antes de lhe explicar a sua função, veja o exemplo:

```
...
<head>
<script type="text/javascript">
<!--
//códigos JavaScript
//-->
</script>
</head>
...
```

Agora vamos compreender isso melhor. Os browsers que não suportam o JavaScript irão exibi-lo como se fosse parte do conteúdo da página. Para prevenir essa prática, utilizamos o comentário em HTML.

Porém, você pode se perguntar: Para que serve as duas barras (//) antes do fechamento do comentário? Bom, elas previnem que o interpretador do JavaScript as interpretem.

Alguns detalhes para finalizar

"Case sensitive"

JavaScript é "case sensitive", o que significa dizer, que o interpretador diferencia minúsculas de maiúsculas. Veja um exemplo:

```
<script type="text/javascript">
<!--
var txtMsg = "Você foi introduzido ao JavaScript.";
alert(txtmsg); // Isto irá causar um erro.
```

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

No exemplo acima, gostaríamos de exibir através do alert() a mensagem armazenada na variável **txtMsg** (*não se preocupe, você aprenderá sobre variáveis ainda*), mas como chamamos a variável por **txtmsg,** o JavaScript não a encontrou no código, causando um erro - *txtmsg is not defined*.

Moral da história, uma simples letra maiúscula ou minúscula causa toda a diferença.

Ponto e vírgula (;) no final das declarações

Em JavaScript a utilização do **;** (*ponto e vírgula*) é quase que opcional. Digo quase, porque se você dispor de duas declarações de código na mesma linha, precisará do **;** (*ponto e vírgula*).

Vejamos o exemplo abaixo:

```
alert("Estes detalhes..."); alert("... são
importantes");
alert("Notou...")
alert("A diferença?")
```

Eu sempre utilizo o **;** (*ponto e vírgula*) no final de minhas declarações.

Outro detalhe: se você utilizar algum compressor de JavaScript - *para reduzir o tamanho do arquivo* - e os seus códigos não tiverem o **;** (*ponto e virgula*), aparecerá erros, pois os compressores distribuem os códigos em uma única linha.

Variáveis

São de grande utilidade e empregadas ao extremo. Utilizarei uma metáfora para você entendê-las melhor.

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Imagine você em um supermercado com uma sacola para colocar as suas compras - o que você terá dentro dela é bastante variável, uma vez que você pode pegar um produto na prateleira e depois desistir de levá-lo, trocar por outro produto ou adicionar algum outro.

Qual a moral da história? O conteúdo da sua sacola de compras é variável, ou seja, não sabemos ao certo o que teremos dentro dela. O que definirá isto será o tempo em que estivermos realizando a compra e as nossas necessidades.

Com as variáveis é do mesmo jeito, você tem o seu nome definido (sacola), mas o conteúdo (produto) pode iniciar-se com X, depois alterar-se para Y ou até mesmo ficar nulo. O que definirá o valor da variável em si será a execução do script por completo.

Resumindo: variáveis são como um depósito que você utiliza para guardar informações e, o seu valor (conteúdo), pode alterar durante a execução do script.

Criando variáveis

Vejamos um exemplo de como declarar uma variável:

```
var nomeVariavel = "valorVariavel";
```

ou

```
nomeVariavel = "valorVariavel";
```

Você pode criar uma variável com ou sem a declaração **var**, mas é aconselhável utilizá-la.

Agora, vamos conhecer algumas regrinhas necessárias ao criar as variáveis:

- Como dito no Capítulo 1, JavaScript é "case-sensitive", o que significa dizer que uma variável com o nome **ILUMINATI** é diferente de **iluminati**.
- Deve-se iniciar o nome de uma variável com uma letra ou com um sublinhado (_).

Atribuindo valor a uma variável

Vamos atribuir o valor **JavaScript** à variável assunto. Observe:

```
var assunto = "JavaScript";
```

Variáveis globais e locais

Os nomes **global** e **local** são bem explicativos. Vamos compreendê-los bem, pois são de grande importância.

Quando criamos uma variável dentro de uma função, a tornamos **local**, ou seja, acessível somente dentro da função em que foi criada. Podemos até criar outras variáveis com o mesmo nome e com valores diferentes, porém elas terão validade somente dentro da função em que foi declarada.

Já as variáveis globais, são declaradas fora das funções e ficam acessíveis a toda e qualquer função na página em que a variável esteja.

Operadores

Operadores são símbolos utilizados para atuar sobre valores. Conheceremos os diferentes operadores utilizados no JavaScript.

Operadores aritméticos

Operador	Descrição	Exemplo	Resultado
+	Adição	x=2 y=8 x+y	10
-	Subtração	x=10 y=3 x-y	7
*	Multiplicação	x=4 x*5	20
/	Divisão	15/5	3
%	Módulo (restante da divisão)	5%2 10%8	1 2

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Operador	Descrição	Exemplo	Resultado
		10%2	0
++	Incrementos	x=5 x++	x=6
--	Decrementos	x=5 x--	x=4

Operadores de comparação

Operador	Descrição	Exemplo	Resultado
==	é igual a	5==8	falso
===	é igual a (compara o valor e o tipo)	x=5 e y="5" x==5 x===5	verdadeiro falso
!=	não é igual	5!=8	verdadeiro
>	é maior que	5>8	falso
<	é menor que	5<8	verdadeiro
>=	é maior que ou igual a	5>=8	falso
<=	é menor que ou igual a	5<=8	verdadeiro

Operadores lógicos

Operador	Descrição	Exemplo	Resultado
&&	e	x=6 y=3 (x < 10 && y > 1)	verdadeiro
||	ou	x=6 y=3	falso

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Operador	Descrição	Exemplo	Resultado
		(x==5 || y==5)	
!	não	x=6 y=3 !(x==y)	verdadeiro

Operadores de atribuição

Operador	Exemplo	É o mesmo que
=	x=y	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

Operador condicional

Operador condicional é aquele que atribui um valor a uma determinada variável com base em uma condição, ou seja, se a condição for verdadeira, teremos um valor x, caso contrário um valor y.

Conheça a sintaxe desse tipo de condição:

```
nomeDaVariável = (condição) ? valorSeVerdadeiro : valorSeFalso;
```

Agora observe um exemplo para uma melhor compreensão:

```
var fruta = "Maçã";
var resposta = (fruta == "Maçã") ? "A fruta é a Maçã." : "Não sei qual é a fruta.";
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

No exemplo acima fizemos o seguinte processo: inicialmente temos uma variável denominada fruta com o valor Maçã; em seguida, a variável resposta com um valor condicional, ou seja, se o valor da variável fruta for igual a Maçã, teremos o valor **A fruta é a Maçã**, caso contrário, teremos **Não sei qual é a fruta**.

Condições

Declarações condicionais no JavaScript, assim como em outras linguagens, são utilizadas para executar diferentes ações baseadas em determinadas condições.

Declarações condicionais

Durante o desenvolvimento dos nossos scripts, sentimos a necessidade de executar determinadas ações baseados em uma determinada condição, ou seja, se uma condição for verdadeira (true) será executada a ação X, caso contrário, se for falsa (false), a ação Y.

Em JavaScript temos as seguintes declarações condicionais:

Declaração if

Utilizamos essa declaração quando necessitamos que somente uma parte do código seja executada e quando uma determinada condição for verdadeira.

Declaração if...else

Utilizamos essa declaração para que somente uma parte do código seja executada quando determinada condição for verdadeira e, outra parte, quando a condição for falsa.

Declaração if...else if...else

Utilizamos essa declaração quando desejamos selecionar um bloco de código entre vários outros.

Declaração switch

Utilizamos também essa declaração quando desejamos selecionar um bloco de código entre vários outros. É uma forma mais prática para alguns casos.

Declaração if

Sintaxe

```
if (condição) {
 //código a ser executado quando a condição for
verdadeira.
}
```

Exemplo

O exemplo abaixo exibirá o texto **Olá brazuca!** se a variável **pais** for igual a Brasil.

```
var pais = "Brasil";

if(pais == "Brasil") {
 alert("Olá brazuca!");
}
```

No exemplo acima, a declaração if (se) verifica se a variável *pais* é igual a Brasil. Como a condição é verdadeira, ou seja, a variável *pais* tem o valor Brasil, será exibido o texto **Olá brazuca!** via **alert()**.

Declaração if...else

Sintaxe

```
if (condição) {
 //código a ser executado quando a condição for
verdadeira.
} else {
 //código a ser executado quando a condição for
falsa.
}
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Exemplo

O exemplo abaixo exibirá o texto **Você não é um brazuca!** se a variável *pais* não for igual a Brasil.

```
var pais = "China";

if(pais == "Brasil") {
 alert("Olá brazuca!");
} else {
 alert("Você não é um brazuca!");
}
```

No exemplo acima, a declaração if (se) verifica se a variável *pais* é igual a Brasil. Como a condição é falsa, ou seja, a variável *pais* não tem o valor Brasil e sim China, será exibido o texto **Você não é um brazuca!** via alert().

Declaração if...else if...else

Sintaxe

```
if (condição) {
 //código a ser executado quando a condição for
verdadeira.
} else if (2º condição) {
 //código a ser executado quando a 2º condição for
verdadeira.
} else {
 //código a ser executado quando a primeira e a
segunda condição forem falsas.
}
```

Exemplo

```
var pais = "China";
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
if(pais == "Brasil") {
 alert("Olá brazuca!");
} else if(pais == "China") {
 alert("Olá chinês!");
} else {
 alert("Você não é um brazuca e nem chinês!");
}
```

No exemplo acima, realizamos duas verificações: perguntamos se a variável *pais* tem o valor Brasil (a primeira). Caso verdadeiro, exibiremos o texto **Olá brazuca!**; ou perguntamos se a mesma variável tem o valor China (a segunda). Neste caso, exibiremos o texto **Olá chinês!**.

Caso a variável não tenha nenhum dos dois valores - Brasil ou China - exibirá o texto **Você não é um brazuca e nem chinês!**

Declaração switch

Sintaxe

```
switch(expressão) {
 case valor 1:
  //código a ser executado se a expressão = valor
1;
  break
 case valor 2:
  //código a ser executado se a expressão = valor
2;
  break
 default:
  //código a ser executado se a expressão for
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
diferente do valor 1 e valor 2;
}
```

Exemplo

O exemplo abaixo dirá ao usuário o estado em que ele reside. Observe:

```
var estado = "MG";

switch(estado) {
  case "MG":
    alert("Você mora em Minas Gerais.");
    break
  case "SP":
    alert("Você mora em São Paulo.");
    break
  case "ES":
    alert("Você mora no Espírito Santo.");
    break
  default:
    alert("Onde moras?");
}
```

Vamos compreender o exemplo. Temos a variável *estado* com o valor **MG**. Logo após, iniciamos nossa declaração condicional com o switch, passando a ele como parâmetro a expressão (variável) **estado**.

Então, utilizamos o case (caso). Para isso é necessário verificar o seguinte: se o valor da variável for **MG** exibimos um texto, se for **SP** exibimos outro, e assim por diante; se o valor da variável não coincidir com nenhum dos valores nos cases, exibimos um texto padrão (default).

Algumas considerações a cerca do switch.

- A expressão na maioria das vezes será uma variável;
- O valor da expressão é comparado com os valores de cada caso (case) da estrutura. Caso ocorra um valor semelhante, o código a ele associado é executado;
- O break previne que o próximo caso (case) seja executado automaticamente;
- O valor padrão (default) é utilizado caso nenhum dos valores dos cases coincidam com o valor da expressão.

Loops

As declarações de looping em JavaScript, assim como em outras linguagens, tem por objetivo a execução de um mesmo bloco de código por uma determinada quantidade de vezes ou enquanto uma condição for verdadeira.

Neste tópico conheceremos as declarações de looping:

Declaração for

Looping que percorre determinado bloco de código por uma quantidade especificada de vezes.

Declaração while

Looping que percorre determinado bloco de código se e enquanto a condição for verdadeira.

Declaração do...while

Esse loop é uma variável do loop while. Ele sempre executará um bloco de código uma vez e o repetirá enquanto a condição for verdadeira.
Mesmo que uma condição seja falsa, esse loop ocorrerá uma vez pois a condição é verificada após a execução.

Declaração break e continue

São declarações especiais para se trabalhar com looping.
O break interrompe o looping e prossegue com o restante do código (se houver).
O continue pára o loop atual e continua com o próximo valor.

Você entenderá melhor nos exemplos abaixo.

Loop For

Sintaxe

```
for (var = valorInicial; var <= valorFinal; var =
var + incremento) {
 //código a ser executado.
}
```

Exemplo

```
var numero = 1;

for (numero = 1; numero <= 5; numero++) {
 alert("O número atual é: " + numero);
}
```

No exemplo acima temos a variável *numero* que inicia com 1 e em seguida um *loop for* que funciona da seguinte maneira: temos o valor inicial (1) e o valor final (5) para a variável *numero*, seguido de um incremento (++). Isso significa dizer que o loop iniciará com a variável *numero* com o valor 1 e finalizará quando ela alcançar um valor menor ou igual a 5.

O código a ser executado será uma mensagem de alerta dizendo ao usuário o número atual do looping.

Loop while

Sintaxe

```
while (var <= valorFinal) {
 //código a ser executado.
}
```

Exemplo

```
var numero = 1;
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
while (numero <= 5) {
 alert("O número atual é: " + numero);
 numero = numero + 1;
}
```

O exemplo do *loop while* é semelhante ao do *loop for*. O resultado de ambos são os mesmos, porém eles se diferenciam pelas suas características.

No exemplo do *while*, começamos informando o valor final (5) da variável *numero,* em seguida, o código a ser executado e, por fim, o incremento da variável em questão (*numero*).

Loop do...while

Sintaxe

```
do {
 //código a ser executado.
} while (var <= valorFinal);
```

Exemplo

```
var numero = 1;

do {
 alert("O número atual é: " + numero);
 numero = numero + 1;
} while (numero < 1).
```

No exemplo acima, temos a condição (*numero* < 1) indicando que a variável *numero* é menor do que 1, ou seja, 0. Logo, o loop não deveria ser executado.

Entretanto, como expliquei anteriormente, o loop *do* verifica a condição após ser executado e, devido a isso, ocorreu a execução do loop.

Break e Continue

Exemplo - break

```
var numero = 95;

for (numero = 95; numero < 150; numero++) {
 if(numero == 100) {
  break;
 }
 alert("Número: " + numero);
}
```

Neste exemplo, o loop *for* deveria ocorrer até o número 149 (*numero* < 150). Porém, utilizamos à condição *if* para detectar quando a variável *numero* estiver no número 100.

Quando isto ocorrer, utilizamos o comando *break* para interromper o loop, assim, chegaremos somente até o número 100.

Exemplo - continue

```
var idade = 20;

for (idade = 20; idade <= 25; idade++) {
 if(idade == 24) {
  continue;
 }
 alert("Você tem " + idade + " anos ?");
}
```

Conforme dito anteriormente, o comando *continue* interrompe o loop em um determinado momento e continua com o mesmo.

No exemplo acima, o loop *for* deveria rodar do número 20 até o 25, porém o comando *continue* interrompe a exibição do número 24.

Funções

As funções são blocos de códigos reutilizáveis, uma verdadeira "mão na roda". Elas são executadas através de algum evento ou quando são invocadas.

Você pode chamar uma função de qualquer lugar da página ou até mesmo de outras páginas contanto que o script esteja em um arquivo externo (.js).

As funções podem conter parâmetros ou não e, quando eles existem, são informados à função quando a chamamos.

Sintaxes

Sintaxe - sem parâmetros

```
function nomeDaFuncao() {
 //códigos referente à função.
}
```

Sintaxe - com parâmetros

```
function nomeDaFuncao(varivel 1, variavel 2, variavel 3,
...) {  //códigos referente à função.
}
```

Exemplos

Veja alguns exemplos para uma melhor compreensão.

Exemplo - sem parâmetros

```
function boasVindas() {
 alert("Olá, seja bem vindo.\nEspero que goste do nosso
site.");
}

// Um exemplo para chamá-la.
```


Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

```
<a ... onclick="javascript:boasVindas();">Chamar a Função</a>
```

Essa função é bem simples e, ao ser chamada, exibirá uma mensagem **alert()**.

Ela é invocada através do evento *onclick* inserido na tag *a* (link).

Exemplo - com parâmetros

```
function boasVindas(nome) {
 alert("Olá " + nome + ", seja bem vindo.\nEspero que goste do nosso site.");
 window.location.href = "http://www.projetoiluminati.com";
}

// Um exemplo para utilizá-la.
<input type="button" value="Mensagem de boas vindas" onclick="boasVindas(document.nomeForm.nomeCampo.value);" />
```

Acima temos o exemplo de uma função com parâmetro.

Esta mensagem pode ser personalizada uma vez que o parâmetro passado a ela é o nome de algum usuário. Após, temos uma forma de redirecionar o usuário para um site específico, no exemplo, o portal Projeto iluminati.

Para chamarmos a função utilizamos um campo de formulário do tipo texto (text) para que o usuário possa digitar o seu nome.

Ao clicar no botão intitulado **Mensagem de boas vindas** é invocada a função *boasVindas*, passando a ela o nome digitado.

Para enviarmos o nome digitado à função fazemos assim: *document.nomeForm.nomeCampo.value.*

Interpretamos essa declaração da seguinte forma:

document é igual ao documento, ou seja, a página em si; *nomeForm* é igual ao nome do *form*, concedido a ele através do atributo *name* da tag *form*; *nomeCampo* é o nome do campo no qual o usuário informou o seu nome; já o *nomeCampo.value* é o valor contido no campo, no caso, o nome informado.

A declaração return

Quando queremos que uma função retorne algum valor, utilizamos a declaração *return*, a qual tem o papel de especificar o valor retornado pela função.

Exemplo

```
function calculo(valor1,valor2) {
 total = valor1 * valor2;
 return total;
}

// Chamando a função através do alert().
alert(calculo(58,10));
```

No exemplo acima, chamamos a função *calculo* fornecendo a ela dois parâmetros - valor1 (58) e valor2 (10).

Nessa função temos a variável *total* que multiplica os dois parâmetros informados.

Algumas importantes considerações sobre as funções:

- A palavra *function* sempre deve ser escrita com letras minúsculas;
- O nome da função pode conter letras minúsculas e/ou maiúsculas;
- As chaves { } indicam o início e o término da função;
- Mesmo que uma função não contenha parâmetros, os parentêses ( ) devem ser incluídos após o nome da função.

Método getElementById

Provavelmente o método mais utilizado no DOM seja o *getElementById*. Sua função simplesmente retorna o elemento cujo atributo ID foi especificado. Caso o elemento não exista ou o ID informado seja incorreto ele retornará *null*.

É importante lembrarmos que somente um objeto pode conter um ID específico, ou seja, não podem existir dois objetos em uma página com o mesmo valor de um atributo ID.

Considere o seguinte campo de um formulário:

```
<input type="text" id="nomeContato" name="nomeContato" ...
```

Para acessarmos o valor digitado nesse campo de texto em JavaScript procederemos assim:

```
document.nomeForm.nomeContato.value
```

Ou seja, precisamos declarar *document* mais o nome do formulário em que o campo se localizada e o seu nome espeficado no atributo name.

Se utilizarmos o método *getElementById*, podemos acessá-lo assim:

```
document.getElementById('nomeContato').value
```

Assim, ao invés de informarmos toda a localização do objeto, dizemos apenas o seu atributo ID ao método *getElementById*.

Por exemplo: imagine uma pessoa solicitando seu endereço e, ao invés de você informar rua, bairro, cidade, estado, você informar apenas o seu CEP e o seu número.

Na página **Método: getElementById** existem dois scripts de exemplos, um em JavaScript e outro utilizando o método *getElementById* do DOM.

Detalhe importante: *getElementById* é diferente de getelementbyid ou getElementbyId e por aí vai. Caso você escreva o nome errado será retornado um erro. Por exemplo: document.getelementByid não é uma função.

Método innerHTML

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

Interno é o equivalente de inner em inglês. Então innerHTML quer dizer o HTML interno de algo, sendo esse último um objeto, ou seja, uma tag.

Assim, para alterarmos ou inserirmos conteúdo ou uma marcação HTML em um objeto, utilizamos o método innerHTML.

Imaginemos uma tag div (ou qualquer outra) em sua marcação HTML e que essa tag esteja com o atributo ID definido como box. Então, podemos utilizar o método **getElementById** para acessá-la e usufruir do método innerHTML para manipular seu conteúdo ou marcação HTML.

Vejamos um exemplo de como alterar o conteúdo de uma tag:

```
var obj = document.getElementById("box");
obj.innerHTML = "Este será o novo conteúdo.";
```

Vejamos um exemplo de como alterar a marcação HTML de uma tag:

```
var obj = document.getElementById("box");
obj.innerHTML = "<h1>Esta é a nova marcação HTML desta
tag.</h1>";
```

Utilizando o operador de atribuição += podemos acrescentar conteúdo ao conteúdo já existente de uma tag como marcação HTML.

Observe esta marcação HTML:

```
<div id="box">
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
sed diam nonummy nibh euismod tincidunt.</p>
</div>
```

Vamos supor que queremos inserir um título de primeiro nível (h1) acima do parágrafo existente na marcação acima. Para tal procedimento faremos assim:

```
var obj = document.getElementById("box");
var novaMarcacao = "<h1>Título deste parágrafo.</h1>";
obj.innerHTML = novaMarcacao += obj.innerHTML;
```

Introdução ao AJAX

Uma sigla, diversas novidades e inúmeras possibilidades que podem ser realizadas com essa metodologia de desenvolvimento para a web.

Como havia dito anteriormente **Ajax** ou **A**synchronous **J**avaScript **A**nd **X**ML não é uma tecnologia, e sim um metodologia, a qual utiliza-se de vários recursos/tecnologias que listaremos abaixo:

- marcação e apresentação de conteúdo, utilizando XHTML e CSS, seguindo os padrões da web;
- interação e apresentação dinâmica através do DOM;
- troca de dados e manipulação desses através da XML e do XSLT;
- recuperação de dados de forma assíncrona utilizando-se o XMLHttpRequest;
- JavaScript com a função de integrar todos esses ingredientes.

Jesse James Garrett diretor do departamento de estratégia de experiência do usuário e sócio-fundador da **Adaptive Path** foi o criador da sigla Ajax e ninguém mais do que o **Google** foi uma das empresas responsáveis em mostrar ao mundo e aos desenvolvedores as inúmeras possibilidades que essa metodologia pode oferecer aos aplicativos e ao usuário quando interagem com tais aplicativos. Essas demonstrações vieram em produtos como: **Gmail**, **Google Suggest**, **Google Maps**, dentre outros.

Quando interagimos com aplicativos baseados no Ajax, não precisamos esperar que a página se recarregue a cada interação que fazemos com ela. À medida que vamos utilizando seus recursos, a interface vai se moldando às nossas necessidades, baseando-se em nossas ações e requisições perante a ela.

Nas unidades seguintes deste capítulo falaremos de forma mais prática para que você compreenda e reconheça o Ajax.

Conhecendo Ajax

Imagine o seguinte cenário: você irá realizar um cadastro e, para se cadastrar, você precisará escolher um login para utilização do sistema em questão. Então, você escolhe o login, preenche o restante dos dados e envia o formulário. Como resposta você terá provavelmente a seguinte mensagem: *o login informado já está em uso por outro usuário, por favor, escolha outro*. E lá vai você mais uma vez, tendo em alguns casos que ver a mesma mensagem diversas vezes, o que pode até fazer você desistir do cansativo processo de escolha de login.

Se tal processo fosse baseado em Ajax poderia conter duas ou mais possibilidades, por exemplo: informar ao usuário que tal login já existe quando ele terminar de informá-lo e sair do campo e/ou disponibilizar um botão para que tal consulta seja feita sem que a página se recarregue e sem tornar o processo cansativo e demorado.

Vejamos esse exemplo utilizado no Gmail:

Mas, como tudo isso ocorre? Como desenvolver tais recursos?

Bom, vamos ver como acontece essa recuperação de dados de forma assíncrona e posteriormente aprenderemos como desenvolvê-la baseado no Ajax.

O método clássico de interação com uma interface ocorre da seguinte maneira: ocasionamos uma ação através de algum evento (com o mouse, o teclado, dentre outros) e tal ação é enviada ao servidor que irá interpretá-la e retornará algum dado referente a tal ação.

Vamos compreender isto melhor baseando-se no exemplo do login que citamos anteriormente.

O usuário efetua uma ação – escolhe o login, preenche o restante dos dados e envia o formulário – tal ação é enviada ao servidor que retornará uma mensagem ao usuário baseando-se nos dados por ele informados, ou seja, será informado se o cadastro foi realizado, ou se ocorreu algum erro que deverá ser corrigido para conclusão do mesmo.

O método de interação com uma interface baseado no Ajax ocorre da seguinte maneira: alguma ação é ocasionada através de algum evento (com o mouse, o teclado, dentre outros) e uma função é associada a tal ação. Essa função irá comunicar-se com o servidor sem que o formulário

seja enviado e informará ao usuário a mensagem que foi retornada referente à ação que ele ocasionou.

Resumindo, você não precisa submeter o formulário para comunicar-se com o servidor, basta utilizar o método XMLHttpRequest (falaremos melhor sobre ele a seguir) para se comunicar e interagir com o servidor através de algum evento sem que a página seja recarregada.

A imagem abaixo ilustra os modelos apresentados nos parágrafos acima.

O objeto XMLHttpRequest

O objeto XMLHttpRequest foi primeiramente implementado pela Microsoft no Internet Explorer como um objeto ActiveX chamado XMLHTTP. Logo após, outros browsers (Mozilla, Safari, entre outros) seguiram o mesmo caminho, implementando o XMLHttpRequest em uma classe que suporta as propriedades e os métodos do objeto ActiveX da Microsoft.

Como você pode perceber, outros browsers implementaram o objeto XMLHttpRequest de forma nativa, mas a Microsoft, em seu navegador (Internet Explorer), o implementou como um objeto ActiveX, apesar de ter sido sua a idéia.

Isso significa dizer que precisaremos verificar qual o objeto que o browser trabalha, XMLHTTP ou XMLHttpRequest.

Conheceremos agora as propriedades, métodos e eventos do objeto XMLHttpRequest e, no capítulo seguinte (onde desenvolveremos o projeto final), veremos como instanciar tal objeto para trabalharmos com ele.

Propriedades do objeto XMLHttpRequest	
readyState	A requisição se apresenta em 4 (quatro) estágios; ambos representando por um número. • 0 - não inicializado; • 1 - carregamento;

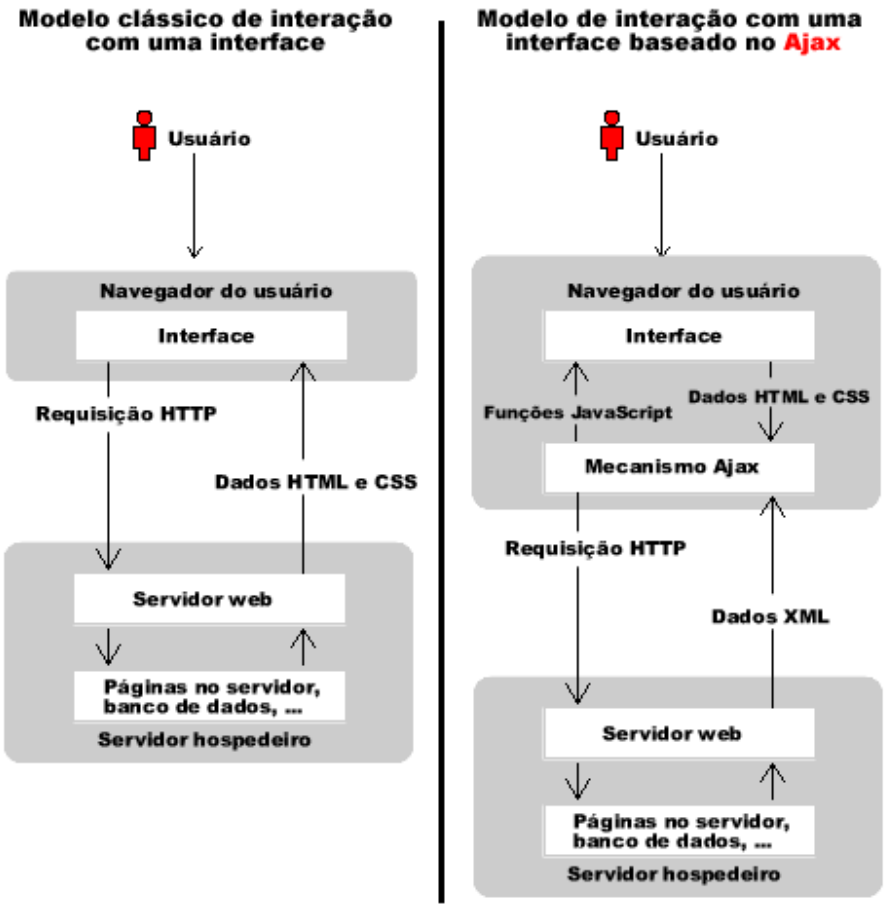
Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

	• 2 - carregado; • 3 - interativo; • 4 - completo.
status	Código númerico do status HTTP retornado pelo servidor web.
statusText	Texto associado ao código númerico do status HTTP. Por exemplo: 200 significa "OK" e 404 significa "Página não encontrada".
responseText	String que contém os dados retornados pelo servidor web.
responseXML	Se o servidor web retornar um documento XML, lhe permitindo acessá-lo através de funções JavaScript utilizando o DOM.

Métodos do objeto XMLHttpRequest	
open(método, url, síncrono, usuário, senha)	Inicia uma nova requisição, onde: • método - Requisição HTTP, na maiorida das vezes "GET", ou "POST"; • url - endereço da URL que será requisitada no servidor web; • síncrono - se o método trabalhará de forma síncrona ou assíncrona; o valor padrão é true - assíncrona; • usuário e senha - se o servidor web necessitar de uma autenticação.
setRequestHeader(nome, valor)	Informa um cabeçalho (header) para a requisição.
send(dados)	Envia a requisição. Enviando opcionalmente os dados.
abort()	Aborta uma requisição em atividade.
getResponseHeader(nome)	Retorna o valor do cabeçalho

Instrutor: Alann Jaksnon (filosofo@pop.com.br)
http://www.projetoiluminati.com/

	(header) informado.
getAllResponseHeaders()	Retorna uma string contendo todos os cabeçalhos (header).
Eventos do objeto XMLHttpRequest	
onreadystatechange	Elevando a cada mudança da propriedade readyState.



Compreendendo AJAX

Repare nesta função abaixo:

```
function openAjax() {
    var ajax;
    try {
        ajax = new XMLHttpRequest();
    } catch(ee) {
        try {
            ajax = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
            try {
                ajax = new ActiveXObject("Microsoft.XMLHTTP");
            } catch(E) {
                ajax = false;
            }
        }
    }
    return ajax;
}
```

Essa função nomeada como **openAjax()** é a responsável em instanciar o objeto **XMLHttpRequest** e essa instância do objeto será atribuída a variável **ajax**.

Repare que através do **try / catch** estamos verificando qual o método suportado pelo browser: **XMLHttpRequest**, presente no Mozila, Firefox, Safari, entre outros; ou **XMLHTTP**, utilizado no Internet Explorer como um objeto **ActiveX**.

Função que chama o AJAX

Esta é uma função padrão em JavaScript que vai chamar o AJAX em segundo plano. Vamos aos comentários em verde.

```
//função que vai chamar o AJAX
function NomeFuncao(){
	//é definida uma variável para receber o método XML do ajax
	var HTTP = openAJAX();
	//é atribuído o método(GET ou POST) de envio dos dados e indicado qual a
	página que receberá os dados
	HTTP.open("POST","pagina.asp",true);
	//é atibuído o cabeçalho para saber qual o método de codificação da página
	HTTP.setRequestHeader("Content-Type", "application/x-www-form-
	urlencoded");
	//é verificado o status da comunicação com a página requisitada, atribuindo
	uma função
	HTTP.onreadystatechange = function(){
		//é verificado qual o estado atual;
		if (HTTP.readyState == 1){ //carregamento
			//o campo com “id” apontada receberá uma imagem e a palavra
			carregando...
			document.getElementById("local").innerHTML = “<IMG
			SRC=’carregando.gif’> Carregando...”;
		} //fecha if-readyState
		//se estado = 4, então processo completo e a “id” referenciada pelo
		<DIV> ou <SPAN> receberá os dados da página requisitada
		if (HTTP.readyState == 4){ //completo
			if (HTTP.status == 200){ //sucesso
				//o campo com “id” apontada receberá os dados da pagina
				requisitada pelo AJAX
				document.getElementById("local").innerHTML =
				HTTP.responseText;
			} //fecha if-status
		} //fecha if-readyState
	} //fecha onreadystatechange = function()
	//parâmetros que serão enviados à pagina requisitada
	HTTP.send("parametro="+document.getElementById('id').value);
} //fecha função principal
```

Referências

- http://www.imasters.com.br
- http://www.linhadecodigo.com.br
- http://www.ajaxpatterns.org
- http://dev2dev.bea.com/pub/a/2005/08/ajax_introduction.html
- http://www.elcio.com.br
- http://pt.wikipedia.org/wiki/AJAX
- http://jairelton.com/tutoriais/ajax/ajax.html