Aula 17: Eventos

Nesta aula você verá o conceito de evento na linguagem JavaScript. Veremos quais os modelos de tratamento de eventos, como associar trechos de código às suas ocorrências e muitas outras coisas mais sobre esta importante forma de programação dinâmica.

Objetivos:
Nesta aula você aprenderá:
- o que são eventos
- quais os modelos de tratamento de eventos
- como fazer a associação de código a eventos
- quais os tipos de evento
- como usar a palavra chave `this`
- como usar objeto `Event`

Pré-requisitos:
As aulas 12 a 16 são importantes para esta aula.

1. Definindo e Caracterizando Eventos

O código JavaScript na tag `<script>` é executado apenas uma vez quando a página é lida pelo navegador. Um programa que usa apenas scripts definidos desta forma não pode responder dinamicamente às atitudes do usuário. Programas dinâmicos devem poder responder a certas ações do usuário ou a acontecimentos decorrentes da exibição da página no navegador.

Um **evento** é um acontecimento iniciado por alguma atitude do usuário (o movimentar do mouse, o pressionar de uma tecla, o clicar de um botão, o envio de um formulário, etc.) ou pelo próprio funcionamento do navegador (o fim do carregamento de uma página para a exibição, o não conseguir carregar uma imagem etc.).

Todo evento envolve uma ação e um objeto (ou trecho da página) que sofre esta ação. Por exemplo, quando o usuário pressiona o botão do mouse e o cursor está sobre uma imagem, dizemos que esta imagem sofreu o evento de "click" do mouse.

Numa linguagem orientada a eventos (como JavaScript) é

possível definir trechos de código que serão executados quando ocorrer um determinado evento. Associar um código a um evento é chamado de "capturar" um evento. Chamamos de "tratamento" de um evento à execução do código associado a ele.

1.1. Modelos de Tratamento a Eventos

Infelizmente, a forma de capturar e tratar um evento em JavaScript é uma das coisas mais difíceis da linguagem. Esta dificuldade existe porque os navegadores mais utilizados (Netscape e Explorer) implementaram modelos de tratamento a eventos bem diferentes entre si. Se por um lado, com o aparecimento da versão 6 do Netscape, os dois modelos se aproximaram um pouco, por outro lado agora há três versões de modelo de tratamento de eventos.

A tradução literal de **layer** seria camada. O objeto `Layer` só está disponível no Navigator. Ele serve para o posicionamento de elementos. Todo o elemento que está posicionado de maneira não relativa à posição da página na tela é considerado posicionado em uma **camada (layer)** separada do resto do documento.

A diferença começa na definição de quais objetos podem sofrer eventos. No Explorer e no Netscape 6 qualquer objeto pode sofrer eventos. Mas até a versão 4.7 do Nestcape eles estavam restritos aos seguintes objetos: imagens, links, janelas (`window`), `document`, `layer`, formulários e seus elementos.

A principal diferença entre os três modelos diz respeito a quem tem a preferência no tratamento do evento no caso de mais de um objeto que ocupe a região da página que sofreu o evento. Estas diferenças serão discutidas com mais detalhes na segunda seção desta aula.

A versão 1.2 incorporou o objeto `event` que permitiu o tratamento de um grande número de novos eventos e uma flexibilidade maior para tratá-los. Veremos como utilizar este objeto também na seção 2 desta aula.

Outras questões envolvendo tratamento de eventos serão vistas nas seções 1.3, 1.5 e 1.6 a seguir:

- atributo de uma tag HTML associada ao evento;
- usando os atributos `FOR` e `EVENT` em `<script>`;
- propriedade de um objeto JavaScript; e
- propriedade do objeto `Event`.

1.2. Tipos de Eventos

A linguagem suporta diversos tipos de eventos, cada tipo é gerado por circunstâncias diferentes. Por exemplo, o evento gerado quando o usuário "clica" um botão é diferente do

gerado pelo carregamento de uma página no *browser*. Também pode ocorrer que objetos diferentes gerem eventos diferentes sobre as mesmas circunstâncias. Se o usuário "clicar" com o mouse em um botão, é gerado o evento `onClick`, mas o mouse ao ser clicado sobre um texto não gera qualquer evento.

Todos os nomes de atributos iniciam com "on". O uso da primeira letra depois do "on" em maiúscula é uma convenção comumente usada para a descrição dos atributos em arquivos HTML.

Como na forma *client-side* da linguagem os eventos se originam de elementos HTML (como botões, imagens etc.) são definidos como atributos destes elementos. Para permitir a manipulação de evento nesta forma, as várias tags possíveis de responderem a eventos passaram a ter atributos específicos (que depois foram incorporados ao padrão HTML 4.0).

Embora HTML seja insensível ao uso de maiúsculas ou minúsculas, o que é denominado "case-insensitive" , esta convenção auxilia a identificar estes atributos como um conjunto diferenciado dos demais da linguagem, que geralmente seriam escritos em maiúsculas ou minúsculas.

O que distingue o tipo dos eventos é principalmente os atributos que eles "disparam". Cada elemento possível de incluir em uma página HTML contém um determinado conjunto de eventos associados. Para definir o código de tratamento de um evento é necessário definir um atributo na tag relativa ao elemento. Para cada tipo de evento há um atributo específico. Na tabela 17.1 a seguir são mostrados alguns destes atributos e o evento a que estão associados.

Tabela 17.1 - Atributos e sua descrição

Atributo	Descrição
Eventos do mouse	
`onClick`	Clique do mouse sobre o elemento.
`onDblClick`	Clique duas vezes seguidas na mesma posição.
`onMouseMove`	Mouse se move sobre o elemento.
`onMouseOver`	O cursor do mouse está sobre o elemento.
`onMouseOut`	Mouse não está mais sobre o elemento (saiu).
`onMouseDown`	Botão do mouse pressionado sobre o elemento.
`onMouseUp`	Botão do mouse é solto sobre o elemento.
Eventos associados a elementos de formulários	
`onChange`	Modificação do conteúdo do elemento.
`onSelect`	Seleção, desseleção ou ativação de um item.
Eventos da página ou janela	
`onLoad`	Acabou de ser carregada, também associado a imagens.
`onFocus`	Fica ativada, ganha o foco de entrada de dados. Também associado a elementos de texto.
`onBlur`	Perde o "foco" de entrada, fica desativado, também associado a elementos de texto.
`onUnload`	Foi descarregado.
`onResize`	A janela foi redimensionada pelo usuário.
Eventos associados a imagens	
`onError`	Ocorreu um erro ao carregar imagem.
`onAbort`	Carregamento da imagem foi interrompido.

Desseleção, é um termo usado aqui no sentido de **desfazer** a operação de **seleção**!

Eventos de formulário	
onSubmit	Formulário enviado (submetido).
onReset	Requisitou o "reset" (limpar o conteúdo).
Eventos do teclado	
onKeyDown	Ocorre quando uma tecla é apertada pelo usuário.
onKeyPress	Ocorre quando uma tecla é apertada e solta.
onKeyUp	Ocorre quando uma tecla é solta pelo usuário.

É importante notar que nem todos os atributos se aplicam a qualquer elemento e que o mesmo atributo pode fazer sentido para mais de um forma de interação. Por exemplo, `onLoad` geralmente está associado ao carregamento da página, mas também pode ser acionado pelo carregamento de uma imagem. Os eventos do mouse, dependendo da plataforma e da versão do *browser* usado, podem funcionar com links, imagens, elementos de formulário e documentos. A melhor forma de saber qual atributo faz sentido para cada tipo de elemento é consultar um guia de referência. Na WWW é possível obter um guia no endereço:

`ftp://ftp12.ba.best.com/pub/dgoodman/NS4Map.zip`

1.3. Associando Eventos a Elementos HTML

Os códigos de tratamento de eventos são expressos como valores de atributos HTML, na forma de *strings* que contêm um ou mais comandos JavaScript. Se houver mais de um comando eles devem ser separados por ponto e vírgula. Por exemplo, o posicionamento do cursor do mouse (evento `onMouseOver`) sobre um link pode disparar a abertura de uma janela de mensagem. A seguir vemos como associar o evento `onMouseOver` a um link HTML:

```
<A HREF="http://www.cederj.gov.br/"
  onMouseOver="alert('Entre aqui!');">
Click aqui!
</A>
```

A mesma sintaxe do exemplo anterior pode ser usada para os demais **eventos do mouse**. Se você não entendeu bem a diferença entre eles pode ser uma boa forma de auxiliar este entendimento experimentar as diferenças de comportamento com cada um dos outros eventos do mouse da tabela 17.1. Esta também é uma forma de verificar se determinado evento está disponível para aquele elemento HTML em determinada versão de browser ou plataforma.

A string que dispara o evento pode conter chamadas a funções definidas pelo programador, como nas linhas abaixo onde é suposto que `calculaTotal()` seja uma função já

definida em algum lugar do código.

```
<FORM NAME=formucompras>
  <INPUT TYPE=button VALUE="Total"
  NAME="calcula"
  onClick="calculaTotal()">
</FORM>
```

A um mesmo elemento HTML pode estar associado também diversos eventos e cada um disparar uma ação diferente. As linhas de código abaixo chamam diversas funções, uma para cada movimento diferente que o usuário fizer com o mouse sobre os links. Supõem-se que estes links estejam sobre diversas imagens na página, cada uma delas com o nome fornecido ao *Array* `nomeImagem`.

```
<HTML>
  <HEAD>
  <TITLE>Diversos Eventos</TITLE>

  <SCRIPT LANGUAGE="javascript">
  nomeImagem=new Array("nome1",
                       "nome2",
                       "nome3");
  function entrei(n)
  {
    document.saida.texto.value=
    nomeImagem[n];
  }
  function sai( )
  {
    document.saida.texto.value=" saiu ";
  }
  function nada()
  {
  }
  function clica( )
  {
  document.saida.texto.value=" clicou ";
  }
  </SCRIPT>
  </HEAD>

  <BODY BGCOLOR="salmon">
    <CENTER>
    <H1>Diversos Eventos</H1>
    <A href="javascript:nada()"
       onClick="clica()",
       onMouseOver="entrei(0)",
       onMouseOut="sai()">
    <IMG SRC="image1.gif" BORDER=0></A>

    <A href="javascript:nada()"
       onClick="clica()"
       onMouseOver="entrei(1)"
       onMouseOut="sai()">
```

```
    <IMG SRC="image2.gif" BORDER=0></A>

    <A href="javascript:nada()"
       onClick="clica()"
       onMouseOver="entrei(2)"
       onMouseOut="sai()">
    <IMG SRC="image3.gif" BORDER=0></A>

    <FORM NAME=saida>
     <INPUT TYPE=TEXT NAME=texto>
    </FORM>

    </CENTER>
  </BODY>
</HTML>
```

Embora você possa incluir qualquer número de comandos JavaScript dentro de um único evento, uma técnica comum nos casos onde mais de dois comandos são ativados é definí-los como um função entre as *tags* `<SCRIPT> ... </SCRIPT>`, e invocar esta função no atributo do evento.

1.4. A Palavra Reservada `this`

Funções e métodos já foram usados por você diversas vezes no decorrer deste módulo de JavaScript, não? Você está bem seguro da diferença entre ambos? Métodos não são nada mais que funções JavaScript invocadas através de um objeto e por isso mesmo tem uma propriedade muito importante: o **objeto que o chamou**. Este objeto permanece armazenado no corpo do método no valor da palavra-chave `this.`

Por exemplo, um método qualquer, que suponhamos seja chamado `met`, se você o **invocar** através de um objeto, `obj,` quando você o executar através `obj.met(),` o método pode se referir ao próprio objeto `obj` através da palavra reservada `this.`

Mas isto não quer dizer que dentro de funções (que não sejam métodos) `this` não tenha valor. Muitas vezes ao **invocar** uma função você está **invocando** um método do objeto global. Dentro destas funções, a palavra `this` se refere a este objeto. Vejamos um exemplo mais concreto. Suponha que os seguintes trechos tenham substituído os correspondentes no exemplo da seção anterior.

```
....
function sai()
{
  document.saida.texto.value="sai "+this;
}
```

```
.....
function clica(n)
{
document.saida.texto.value="clicou "+n;
}...

  // e antes de cada tag IMG trocar por:
  <A href="javascript:nd()"
     onClick="clica(this)",
     onMouseOver="entrei(0)",
     onMouseOut="sai(0)">
```

Se você visualizar esta nova versão verá como o conteúdo de `this` muda em cada contexto.

Resumindo, `this` serve para referenciar o objeto corrente. Seu valor é definido logo após a criação do objeto (pelo operador `new`), mas seu significado depende do contexto onde é usado.

1.5. Manipulando Eventos na Tag <SCRIPT>

Esta forma só está disponível no Internet Explorer. Ela envolve o uso dos atributos FOR e EVENT na tag <SCRIPT> para especificar o código relacionado com um evento. O nome deste evento será atribuído a EVENT. O elemento HTML no qual o evento ocorre terá seu nome atribuído a FOR. Usando esta forma a linha:

```
<INPUT TYPE=TEXT NAME=texto VALUE=""
onClick="clicou">
```

poderia ser reescrita como:

```
<INPUT TYPE=TEXT NAME=texto Value="">
<SCRIPT FOR=texto EVENT=onClick>
   VALUE="clicou";
</SCRIPT>
```

Mas como esta técnica não é aceita em todos os demais navegadores, não é recomendável. Ela é apresentada aqui apenas para que você conheça todas as formas possíveis de manipulação de eventos comentadas anteriormente.

1.6. Eventos como Propriedades do Objeto

Embora a forma mais freqüente de manipular eventos seja defini-los como atributos da *tag*, como já comentamos, esta não é uma única forma. Nas versões 1.1 e posteriores de JavaScript eles podem ser definidos explicitamente como

funções a serem usadas como propriedades de objetos HTML.

Por exemplo, considere o seguinte evento associado a um botão descrito na forma de atributo HTML:

```
<INPUT TYPE="button" NAME="b1"
       VALUE="Aperte"
       onClick="alert('Valeu!')";>
```

Pode-se ter o mesmo efeito com a seguinte linha de código JavaScript:

```
document.forms[0].b1.onclick=function()
{alert('Valeu!');}
```

Cada objeto da linguagem que representa um elemento HTML tem propriedades que correspondem aos atributos do elemento. Se você atribuir uma função para uma destas propriedades, essa função será usada como uma rotina de tratamento de evento e será invocada automaticamente pelo sistema sempre que o evento correspondente ocorrer. No caso de atributos, a convenção usada é misturar letras minúsculas e maiúsculas na definição do nome do atributo. Mas, nesta forma, como JavaScript é *case-sensitive*, os eventos devem ser expressos exclusivamente em minúsculas (`onclick`, `ondblclick`, `onload`, `onmouseover` etc.).

Esta forma de manipular os eventos tem a vantagem de reduzir a necessidade de misturar HTML e JavaScript, facilitando a manutenção do código. As funções também não precisam ser fixas como os atributos dos códigos HTML, que são definidos apenas uma vez quando o código é criado. As propriedades de JavaScript podem mudar a qualquer hora, o que pode ser muito útil em programas interativos complexos.

Mas, como estas criações do objeto *`Event`* foram feitas a partir de modelos independentes, as propriedades dos objetos são quase completamente incompatíveis.

2. O Objeto `Event`

Uma das deficiências das formas de tratamento a eventos discutidas até agora é que não há maneira de se obter detalhes sobre o evento que ocorreu. Muito provavelmente quando capturamos o pressionar do botão do mouse (`onClick`) vamos querer saber qual o botão foi pressionado. Quando capturamos o pressionar de uma tecla (`onKeyPress`) vamos querer saber, com certeza, qual tecla foi pressionada. Para dar informações adicionais como estas foi definido a partir da versão 1.2 do JavaScript o objeto *Event*. Infelizmente, as propriedades deste objeto variam bastante dependendo do navegador e sua versão. Mais adiante mostramos as propriedades deste objeto indicando a versão onde é válida: Netscape 4 (referenciada como N4), Explorer 4 (referenciada

como IE) e Netscape 6 (referenciada como N6) .

O objeto `Event` é criado automaticamente pelo navegador. No exemplo a seguir, o objeto `Event` é passado como parâmetro para a função `elefante()` que é chamada quando o botão do mouse for pressionado:

```
<img src="elephant.jpg"
     onMouseDown="elefante(event)">
```

A tabela 17.2 descreve as propriedades de `Event` e a qual versão de *browser* elas são aplicadas.

17.2 - Propriedades do objeto `Event`

Propriedade	Descrição
`target`	Destino do evento (N4 e N6).
`srcElement`	Fonte do evento (IE).
`type`	Tipo do evento, uma string contendo o nome do tipo (todas as versões).
`which`	Botão do mouse (1, 2, 3) ou código ASCII da tecla (N4 e N6).
`button`	Botão do mouse (1 ou 2)(IE e N6).
`keyCode`	Código Unicode do caracter correspondente à tecla (IE e N6).
`modifiers`	Uma máscara binária identificando se as teclas Shift, Control ou ALT foram pressionadas (N4 e N6).
`altKey`, `ctrlKey` e `shiftKey`	Valores booleanos individuais identificando as teclas correspondentes (IE e N6).
`pageX`, `pageY`	Posição do mouse dentro da página (N4 e N6).
`clientX`, `clientY`	Posição do mouse dentro da página (IE).
`screenX`, `screenY`	Posição do mouse em relação ao vídeo (todas as versões).

Além destas diferenças, a forma como o objeto `Event` é disponibilizado para o programa também é diferente. No Netscape ele é passado como argumento para a função de tratamento do evento. No Explorer, o objeto `Event` é armazenado em uma variável global denominada `event`.

2.1 Propagação do Evento

Uma característica importante do modelo de tratamento de eventos envolve a noção de **propagação de eventos**. Propagação significa que um evento não ocorre simplesmente em um objeto e então "morre", ao invés disso ele pode se

propagar para outros objetos que estejam na área afetada. Por exemplo, ao invés de ter várias funções de manipulação de eventos, uma para cada elemento de um formulário, pode-se simplesmente ter uma função genérica em um objeto no topo de hierarquia como o objeto `Document`. Um evento que ocorra dentro de um elemento de um formulário, também ocorreu no formulário e também ocorreu no documento. É possível então capturar o evento no documento e depois propagá-lo para o elemento onde ele também ocorreu ou capturar o evento no elemento e depois propagá-lo para o formulário.

Pela última linha do parágrafo anterior podemos perceber que a propagação de um evento pode se dar em duas direções diferentes: do topo da hierarquia para a base ou da base para o topo. Esta diferença é o que caracteriza os três modelos de captura de eventos implementados pelos *browsers* mais populares.

A tradução literal destes dois modelos seria "capturando" eventos e "borbulhando" eventos.

O modelo do Netscape 4.x é chamado "**event capturing**". Nele os objetos `Window` e `Document` têm métodos que permitem a eles requisitar a chance de tratar eventos antes que sejam tratados no objeto onde se originaram. As funções de tratamento podem optar por propagar ou não o evento.

No modelo do Explorer, chamado "**event bubbling**", o evento é sempre capturado pelo elemento que efetivamente recebe o evento para posteriormente ir como uma bolha se expandindo ("borbulhando") por cada elemento seguinte na hierarquia, ou até que uma das funções de tratamento decida que ele não vai mais se propagar.

O modelo implementado pela Netscape a partir da versão 6, chamado de "**event path**", permite que a propagação seja tratada de uma forma ou de outra de acordo com a conveniência do programa.

O que há em comum entre os três modelos é a possibilidade de propagar um evento até os objetos `Window` e `Document`.

2.2. Event Capturing

No Navigator 4.x os objetos `Window`, `Document` e `Layer` podem requisitar a oportunidade de tratar os eventos antes que sejam processados pelo elemento que os gerou. Tal requisição é feita com o uso do método `captureEvents()` definido nos objetos `Windows`,

Document e Layer.

O argumento deste método é uma combinação de constantes definidas como propriedades de Event. Essas propriedades são combinadas usando o operador "|". As linhas abaixo, fazem com que, os eventos correspondentes a movimentos de pressionar e soltar o botão do mouse sejam examinados pelo objeto Window antes de serem tratados pelo objeto de origem:

```
document.captureEvents(Event.MOUSEDOWN |
                       Event.MOUSEUP);
```

Depois de feita esta requisição de captura dos eventos, o programa deve registrar o tratamento para eles:

```
document.onmousedown=function aperta(ev)
{
  alert("Botão apertado !");
};
document.onmouseup=function solta(ev)
{
  alert("Botão solto !");
};
```

Quando uma destas funções recebe o evento, decide o que ocorre a seguir: se o evento capturado é tratado parando de se propagar ou se é passado adiante. O comportamento padrão é a não propagação do evento. Para que ele se propague é necessário invocar o routeEvent() definido no objeto Window, Document ou Layer. Este método passa o evento para o próximo objeto Document ou Layer na hierarquia que tenha usado captureEvent() (caso exista), ou então para o objeto que efetivamente sofreu o evento:

```
document.onmousedown=function ratoDoc(ev)
{
  alert("rotina do documento !");
  routeEvent(ev);
};
```

Uma alternativa à chamada de "routeEvent();" é simplesmente passar o objeto Event para o método "handleEvent();" do objeto que você desejar que o receba. Este método passa o evento para a função apropriada. As linhas abaixo mostram um exemplo completo de como pode-se usar a forma do Netscape 4.x de propagação de eventos:

```
<HTML>
<HEAD>
   <TITLE>Evento no Netscape 4.x </TITLE>
</HEAD>
<BODY bgcolor=lightcyan>
<H1>Evento no Netscape </H1>
```

```
<P>
   <img src="seuemb.gif" name="fig1" >
   <img src="nota.gif" name="fig2"
   width=150 height=150>
</P>
<SCRIPT>
function ratImg(ev)
{
  alert("na imagem!");
}
function ratDoc(ev)
{
  if (ev.target.name=="fig2")
    routeEvent(ev);
  else
    alert("documento!");
}
if((navigator.appName=="Netscape")&&
   (navigator.appVersion.charAt(0)=="4"))
{
  document.captureEvents(Event.MOUSEDOWN);
  document.onmousedown=ratDoc;
  document.images[0].onmousedown=ratImg;
  document.images[1].onmousedown=ratImg;
}
else
  alert("para uso no Netscape 4.x");
</SCRIPT>
</BODY>
</HTML>
```

Resumindo, no Netscape é possível forçar que todos eventos de um determinado tipo sejam direcionados apenas para uma função associada ao evento no objeto `Document`. Para isso utiliza-se a função `document.captureEvents()`. Esta função não pode ser utilizada no Explorer, causando erro na página, pois o tipo de constante passada como parâmetro para o método não está definida (assim como o próprio método). Uma forma de tratar a execução para determinado tipo de *browser* é fazer o tratamento prévio mostrado no exemplo anterior.

2.3. Event Bubbling

No Explorer o modelo anterior é invertido. Ao invés de capturar o evento pelo maior nível na hierarquia e então ir até o objeto de origem, o tratamento do evento começa pelo objeto que o originou e então se dirige ("borbulha") para os demais na hierarquia. Assim se o mouse é pressionado sobre uma imagem no IE4, o código associado ao `onmousedown` na imagem é invocado. A função de tratamento deste evento do `Document` é invocado, e depois desta a do objeto

`Window`. Qualquer uma destas funções pode impedir que o evento passe para o próximo nível definido a propriedade `cancelBuble` do objeto `Event` (que como já comentamos é uma variável global) como `true`.

Esta forma é bem mais simples que a do Netscape. Há um senão neste modelo: nem todos os tipos de eventos "borbulham". Para compreender isso é necessário introduzir os conceitos de evento "bruto" e evento "semântico". O primeiro tipo de evento representa os que simplesmente reportam um evento do mouse ou teclado. Como o passar do mouse sobre algo, o pressionar do mouse, soltar o botão do mouse, tirar o mouse de algo, apertar uma tecla do teclado, soltar uma tecla, ativar ou desativar um elemento.

Semântico caracteriza algo que resulta de um significado diferente das palavras ou atitudes.

Os eventos semânticos por sua vez interpretam a atitude do usuário e lhe atribuem certo significado. Por exemplo o `submit` ou o `reset` de um formulário fazem diversas ações e unem diversas coisas à atitude do usuário.

As regras de propagação seguem esta divisão em categorias de eventos. Os semânticos não se propagam, mas os de outro tipo sim. Isso faz sentido, pois é a fonte do evento que define completamente seu sentido e será, obviamente, o melhor lugar para processá-lo. Por exemplo, não seria o próprio formulário o melhor lugar para processá-lo (submetê-lo ou "resetá-lo") e validá-lo?

As linhas de código abaixo exemplificam este modelo de tratamento de eventos. Nelas também é feito o tratamento prévio para identificação do tipo de *browser*.

```
<HTML>
<HEAD>
  <TITLE>Evento no Explorer</TITLE>
</HEAD>
<BODY bgcolor=lightcyan>
<H1>Evento no Explorer</H1>
<P>
  <img src="seuemb.gif" name="fig1" >
  <img src="nota.gif"   name="fig2"
       width=150 height=150>
</P>
<SCRIPT>
function ratImg(ev)
{
  alert("imagem!");
  event.cancelBubble = true;
  //apagando linha acima para de propagar
}
function ratDoc(ev)
{
  alert("documento!");
```

```
}
if(navigator.appName.indexOf("Explorer")!=-1)
{
  document.images[0].onmousedown=ratImg;
  document.images[1].onmousedown=ratImg;
  document.onmousedown = ratDoc;
}
else
  alert("para uso no Explorer");
</SCRIPT>
</BODY>
</HTML>
```

2.4. Event Path

A Netscape optou por introduzir um novo modelo de tratamento de eventos a partir da versão 6 do seu navegador. O objetivo desta mudança foi tornar o navegador mais compatível com a padronização proposta pelo W3C.

Neste novo modelo não existem mais as funções `routeEvent()` e `captureEvents()` pois não são mais necessárias. Todo evento se propaga a menos que se diga algo em contrário atribuindo o valor `true` à propriedade `cancelBuble` do objeto `Event`. Apesar disso, o Netscape não adotou o modelo do Explorer. A primeira grande diferença é que não é mais possível capturar um evento simplesmente atribuindo um valor a uma propriedade do objeto. A captura deve ser feita através da invocação do método `addEventListener`, definido para todos os objetos. Esta função espera 3 parâmetros como mostrado a seguir:

`addEventListener(tipo,função,capture)`

Onde:

`tipo` - é uma *string* com o tipo do evento ("`mouseup`", "`click`" etc),

`função` - é o nome da função (sem parênteses ou parâmetros), e

`capture` - é um booleano que indica o método de propagação (`true` = capture, `false` = bubble)

Na verdade, o programa pode optar se a propagação do evento vai se dar na ordem de *capture* ou na ordem do *bubbling*. Quem vai indicar isso é o terceiro parâmetro da função `addEventListener()`. As linhas de código abaixo exemplificam este modelo de tratamento de eventos.

```
<HTML>
<HEAD>
<TITLE> Netscape 6.x - Só na Imagem
</TITLE>
```

```
</HEAD>
<BODY bgcolor=lightblue>
<CENTER>
<H1> Netscape - tratado na imagem</H1>
<P> <img src="seuemb.gif" name="fig1"
     width="150" height="150"></P>
</CENTER>
<SCRIPT> // Evento sem propagação
function ratImg(ev)
{
    alert("rotina da imagem !");
    ev.cancelBubble = true;
}
function ratDoc(ev)
{
     alert("rotina do documento !");
}
if ((navigator.appName == "Netscape") &&
    (navigator.appVersion.charAt(0)>"4"))
{
     document.addEventListener
          ("mousedown", ratDoc, false);
     document.images[0].addEventListener
          ("mousedown", ratImg, true);
}
else
     alert("Esta página só funcionará " +
     "corretamente no Netscape 6.x");
</script>
</BODY>
</HTML>
```

Exercícios:

1. Bata o código que está sendo discutido em `Diversos Eventos` na seção 1.3 desta aula, usando no lugar de `image1.gif, image2.gif` e `image3.gif` quaisquer imagens que você queira. Também defina seus nomes adequadamente no *Array* `nomeImagem` do mesmo código.
Agora faça a página ser carregada na forma original das funções definidas. Depois modifique o código para:

- incluir 6 imagens;
- alterar o texto da função de saída, para escrever nome da imagem da qual esta saindo ("saindo de `nomedaimagem`"); e
- altere também o texto da função associado ao evento "click" para também escrever o nome da imagem na qual está clicando.

Verifique como ficou seu documento em cada alteração.

2. Faça no exercício anterior as modificações sugeridas na

seção 1.4. Verifique o conteúdo de `this` visualizando novamente a página e fazendo com que todos os tipos de evento ocorram.

3. (Só faça se você dispuser do Netscape) Utilize as linhas de código do final da seção 2.2 para entender a forma de propagação de eventos lá discutidas. Após identificar os métodos comentados nesta seção, procure entender cada linha de código. Para isso, bata o exemplo e visualize o resultado que terá abrindo uma página com o código em um navegador. Use quaisquer duas figuras que você queira (ou tenha disponível) trocando o nome das suas figuras pelo das exemplificadas no código. "Clique" sobre as imagens e sobre o fundo do documento para ver o que acontece.
Agora responda:

- por que o que acontece quando você clica o mouse na primeira figura é diferente do que ocorre quando clica na segunda?
- o que você faria para o evento se propagar para as imagens?

Verifique se você acertou suas respostas fazendo a alteração correspondente no código e vendo se realmente funciona.

4. (Só faça se você dispuser do Explorer) Utilize as linhas de código da seção 2.3 para entender a forma de propagação de eventos lá discutidas. Após identificar o que foi comentado na seção, procure entender cada linha de código. Para isso, bata o exemplo e visualize o resultado que terá abrindo uma página com o código em um navegador. Use quaisquer duas figuras que você queira trocando o nome das suas figuras pelo das exemplificadas no código. "Clique" sobre as imagens e sobre o fundo do documento para ver o que acontece.
Agora responda: o que você faria para o evento se propagar apenas sobre uma das figuras?
Verifique se você acertou sua resposta fazendo a alteração correspondente no código e vendo se realmente funciona.

Resumo:

Nesta aula você conheceu uma das coisas mais importantes na programação dinâmica: os eventos. Viu como eles podem ser utilizados para dar interatividade as suas páginas e aprendeu os diversos tipos e modelos de tratamento de eventos. Na próxima aula veremos uma das principais aplicações de JavaScript: o tratamento de formulários.

Auto-avaliação:

Quantos conceitos novos e quantas novas possibilidades! Não se preocupe se não ficar muito claro, este assunto é realmente complexo e você só ficará seguro após ter sido um criador de algumas páginas ou pelo menos ter usado cada um dos tipos de eventos algumas vezes. Retorne a esta aula sempre que precisar usar eventos.