



Introdução ao jQuery

FELIPPE ALEX SCHEIDT



Introdução ao jQuery

FELIPPE ALEX SCHEIDT

Felippe Alex Scheidt

Introdução ao jQuery

1ª Edição

Editora Itacaiúnas
2015

Copyright © 2015, Felippe Alex Scheidt

Capa: Joelson Nascimento

Editoração eletrônica: Editora Itacaiúnas

S318i

Scheidt, Felipe Alex

Introdução ao jQuery / Felipe Alex Scheidt 1.ed. – Foz do Iguaçu: Editora Itacaiúnas, 2015.

Ebook: PDF

ISBN 978-85-68154-24-3

1. jQuery (Biblioteca JavaScript 2. JavaScript – linguagem de programação. 3. Ciência da computação I. Título.

CDD-000

CDU- 004.43

Dados Internacionais de

Catálogo na Publicação (CIP)

O conteúdo desta obra é de responsabilidade do autor, proprietário do Direito Autoral.

SUMÁRIO

1 Introdução

1.1 O que é JavaScript?

1.2 O que é jQuery?

1.3 Vantagens do jQuery

1.4 O que é desenvolvimento web?

1.5 O que é o DOM?

2 Introdução ao jQuery

2.1 Começando com jQuery

2.2 Seletores

2.3 Manipulação de estilo

[2.4 Modificando o conteúdo](#)

[2.5 Tratamento de eventos](#)

[Lista de atividades](#)

[3 Seletores](#)

[3.1 Seletores básicos](#)

[3.2 Seleccionando elementos por seus atributos](#)

[3.3 Seleccionando elementos pelo conteúdo](#)

[3.4 Seleccionando elementos pela hierarquia](#)

[3.5 Seleccionando elementos pelo status](#)

[3.6 Uso de filtros](#)

[3.7 Animações e efeitos especiais](#)

[3.8 Funções de busca](#)

[Lista de atividades](#)

[4 Navegação e inserção de elementos no DOM](#)

[4.1 Criando novos elementos com jQuery](#)

[4.2 Removendo elementos](#)

[4.3 Navegando no DOM](#)

[4.4 Manipulação dinâmica do DOM](#)

[4.5 Validação e máscaras](#)

[Lista de atividades](#)

[5 jQuery UI](#)

[5.1 O que é jQuery UI](#)

[5.2 Principais componentes](#)

[Referências](#)

[*OceanofPDF.com*](#)

1 Introdução

Este livro apresenta os conceitos fundamentais para aqueles interessados na programação de *scripts* para páginas web. O foco apresentado aqui consiste em explorar e investigar os principais métodos e recursos disponibilizados na API (*application program interface*) do jQuery, reconhecida hoje como a principal biblioteca para desenvolvimento de interfaces e *scripts*. A didática elaborada para este conteúdo segue a lógica do aprendizado por exemplo. Apresenta-se uma breve explicação do conceito, seguido de código fonte onde o aluno poderá testar imediatamente em seu ambiente de desenvolvimento e verificar os resultados encontrados. Nas próximas seções exploraremos conceitos essenciais para o entendimento dos conceitos que permeiam o desenvolvimento web.

1.1 O que é JavaScript?

JavaScript é uma linguagem de programação dinâmica usada nos navegadores web para realizar processamento no lado do cliente, o que inclui: manipular a interface gráfica alterando propriedades de CSS e do HTML; fazer comunicação com o servidor de acordo com ações e eventos disparados pelo usuário; receber dados do servidor e apresentá-los ao usuário. As possibilidades de programação com JavaScript são inúmeras e junto com o CSS e HTML formam a pilha de tecnologias usadas no desenvolvimento de interfaces para o usuário. Compreender estas tecnologias é essencial para quem pretende se dedicar a esta área de conhecimento. Apesar de o JavaScript ser uma linguagem praticamente obrigatória de ser aprendida nesse contexto, nos últimos anos tem ganhado bastante força na comunidade de desenvolvedores o uso de bibliotecas escritas em JavaScript que abstraem e simplificam funcionalidades típicas encontradas em *web sites*. O uso dessas bibliotecas facilitou muito a aprendizagem de comandos e operações básicas para manipular a interface do usuário. Uma dessas bibliotecas, o jQuery, ganhou destaque mundial e hoje é a principal biblioteca utilizada para desenvolvimento de programação

do lado cliente. Portanto, este livro busca explorar as principais funcionalidades dessa biblioteca e apresentá-la ao público interessado em dar os primeiros passos nessa tecnologia.

1.2 O que é jQuery?

O jQuery é uma biblioteca JavaScript gratuita que oferece um conjunto de funcionalidades que simplificam bastante a programação JavaScript, deixando-a mais intuitiva, clara e fácil. O jQuery é construído sobre a linhagem JavaScript, ou seja, tudo o que é feito com jQuery pode ser feito com JavaScript. É importante destacar que o jQuery não é uma linguagem, na verdade ele é JavaScript sendo interpretado pelo navegador web como código JavaScript e respeitando a mesma sintaxe do JavaScript.

1.3 Vantagens do jQuery

O jQuery simplifica bastante o desenvolvimento de código JavaScript. Sua API é intuitiva e de fácil leitura, o que torna a curva de aprendizado da linguagem mais suave. O jQuery implementa uma única API que executa em qualquer *browser* moderno – *até mesmo no Internet Explorer 6* – sem a necessidade de *hacks* em JavaScript para adaptar o código para cada navegador, pois o jQuery faz esse trabalho por baixo do pano de forma transparente ao programador. Isso economiza um tempo considerável e libera o programador para concentrar-se nas principais funcionalidades de implementação do site. Além disso, os usuários da API se beneficiam também com a liberação de novas versões do jQuery, pois a cada novo *release* seu site é automaticamente atualizado, bastando apenas a substituição do arquivo de importação do jQuery. Agora, se você escreve todo o seu código em JavaScript, cabe ao próprio desenvolvedor o trabalho de atualizar e manter as funcionalidades do seu site, verificando questões de compatibilidade entre as últimas versões de navegadores.

A seguir veremos exemplos comparativos de códigos JavaScript e o equivalente usando jQuery.

Obtendo uma referência a um elemento pelo seu ID:

```
// JavaScript
var menu = document.getElementById("menu");

// jQuery
var menu = $("#menu");
```

Acessando todos os parágrafos da página e modificando a cor de fundo para amarelo:

```
//JavaScript
var ps = document.getElementsByTagName("p");
for(i=0;i<ps.length;i++)
    ps[i].style.backgroundColor = "yellow";

//jQuery
$("p").css("background-color","yellow");
```

Associando um evento do tipo *click* ao botão #bto:

```
// JavaScript
function aviso(){
    alert("botão acionado!");
}

function adicionaEvento() {
    var el = document.getElementById("bto");
    el.addEventListener("click", aviso, false);
}

document.addEventListener("DOMContentLoaded",
adicionaEvento, false);

//jQuery
$("#bto").click(function(){
    alert("botão acionado!");
});
```

Como é possível ver nos exemplos acima, o jQuery consegue executar a mesma funcionalidade do JavaScript com um número bem menor de linhas de comando e ainda consegue deixar mais claro e legível o código. Em resumo, o jQuery possibilita:

- manipular facilmente o conteúdo de uma página web;
- trabalhar com eventos de maneira bastante clara e intuitiva;
- adicionar efeitos especiais;
- manipular um conjunto de elementos de uma página com apenas uma linha de código;
- utilizar uma grande quantidade de *plugins* disponíveis para diversos tipos de necessidades de desenvolvimento web, tais como: menus, slides, animações, gráficos, etc.

A filosofia do jQuery sintetiza o que observamos nos exemplos acima, ou seja: "*escreva menos, faça mais*". A economia de linhas de código é ainda mais significativa em sites profissionais, quando arquivos *scripts* chegam à casa dos milhares de linhas de código.

1.4 O que é desenvolvimento web?

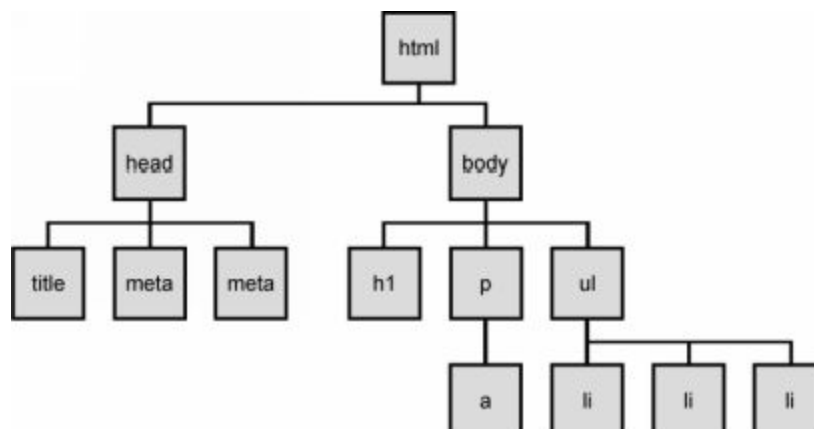
Desenvolvimento web é matéria fundamental dentro dos cursos de informática, devido principalmente ao amplo acesso que o mundo todo vem ganhando a internet. Pessoas que acessam a internet navegam em web sites, que são escritos em basicamente 3 linguagens fundamentais: (1) HTML, (2) CSS e (3) JavaScript. Estas 3 tecnologias são as linguagens básicas que todo navegador web ou *browser* compreende. Programadores que desejam desenvolver sites precisam adquirir letramento nestas linguagens. Todo código que executa no *browser* é definido como camada cliente. Entretanto, o desenvolvimento de sistemas web dinâmicos – aqueles que alteram seus conteúdos de acordo com as ações do usuário – exige ainda o domínio de outras linguagens que executam no lado do servidor ou na camada servidor. Linguagens tipicamente utilizadas são: (1) PHP, (2) Java, (3) Ruby, (4) Python, (5) DotNet, entre tantas outros.

1.5 O que é o DOM?

DOM ou *Document Object Model* é um padrão ou forma de representar objetos HTML que todos os navegadores web implementam. Ele é uma representação hierárquica das tags HTML, onde cada tag tem um nodo pai, podendo também ter um ou mais nodos filhos. Considere o código a seguir:

```
<html>
<head>
  <title>DOM</title>
  <meta charset="utf-8"/>
  <meta name="keywords"
    content="HTML,CSS,jQuery">
</head>
<body>
  <h1>Título principal</h1>
  <p>Um parágrafo descrevendo o conteúdo dessa página e uma
<a href="#">lista</a> com os principais tópicos abordados:</p>
  <ul>
    <li>Desenvolvimento web</li>
    <li>JavaScript</li>
    <li>jQuery</li>
  </ul>
</body>
</html>
```

Quando o navegador web lê o código acima, ele na verdade cria uma representação desse código usando uma árvore de elementos hierárquicos:



Após o término da leitura do código, o navegador constrói o seu modelo que representa essa página, chamada de DOM. É a partir do DOM que o jQuery e JavaScript executam suas funções e permitem que o script navegue e localize elementos HTML. O propósito do DOM é oferecer uma interface programática para que *scripts* JavaScript possam adicionar, remover, substituir, modificar e monitorar os elementos HTML.

Pontos a serem observados:

1. A tag <html> é nó raiz, pois não possui um nodo pai.
2. Os nodos <title>, <meta>s, <h1>, <a> e s são nodos folhas, pois não possuem nodos filhos.

Com esse modelo fica fácil percorrer a árvore e localizar elementos. Por exemplo, tomando como referencial a tag *body* podemos inferir que:

1. o elemento <head> é irmão do elemento <body>
2. os elementos <h1>, <p> e são filhos diretos da tag <body>
3. o elemento <html> é pai da tag <body>
4. o elemento <a> e as três tags são descendentes indiretos de <body>

2 Introdução ao jQuery

Neste capítulo veremos as principais funcionalidades do jQuery, como, por exemplo, o uso de seletores, funções para alterar aparência dos elementos HTML, como trabalhar com eventos e manipular o conteúdo das *tags*.

2.1 Começando com jQuery

O primeiro passo é fazer download da biblioteca do jQuery no site <http://jquery.com/>. Para este livro usarei a versão *uncompressed* do jQuery. Esta versão é geralmente utilizada para a fase de desenvolvimento de *sites*, pois permite lermos o código do jQuery caso haja necessidade. A versão *compressed* é uma versão compactada, com remoção dos espaços em branco. Deve ser utilizada quando o *site* é publicado, pois seu tamanho é menor, o que torna o *download* da página mais rápido.

Com o arquivo baixado (jquery-1.x.x.js ou jquery-2.x.x.js) vamos renomeá-lo para *jquery.js* para manter a simplicidade dos exemplos. Para incorporar o jQuery em sua página HTML precisamos fazer o *import* da biblioteca jquery.js com o seguinte comando:

```
<script src="jquery.js"></script>
```

É importante salientar que o arquivo jquery.js deve estar na mesma pasta onde está o seu arquivo HTML. Caso contrário, o comando acima precisará ser adaptado para conter o caminho do arquivo. Sugiro que siga este passo a passo até conseguir confirmação de que o jQuery está sendo executado corretamente.

Outra alternativa ao *download* da biblioteca é usar um servidor CDN (*content discovery network*). Com o CDN podemos carregar a biblioteca do jQuery diretamente da internet a partir dos servidores que hospedam o *script*, sem a necessidade de baixar o arquivo .js para a pasta do seu projeto ou site. Observe o código a seguir:

```
<script  
src="http://code.jquery.com/jquery-2.0.3.min.js">  
</script>
```

Nesse caso, a biblioteca do jQuery será baixada do site `code.jquery.com`. A biblioteca é apenas um arquivo .js. A vantagem do uso do CDN é que os servidores estão distribuídos por várias regiões do mundo, o que torna mais eficiente o seu *download* por usuários de diferentes países. Recomenda-se usar CDN quando o site já estiver pronto para ser publicado na internet.

Usando o jQuery

O jQuery possui apenas uma função de entrada que você precisará chamá-la toda vez que necessitar algum recurso da API. O nome dessa função é jQuery. Porém, numa página web é comum chamarmos esta função dezenas ou centenas de vezes. Pensando nisso foi criado um alias para esta função, através do símbolo \$, a fim de economizar código. Observe o código abaixo. As duas linhas são equivalentes, a única diferença é que a primeira usa a função jQuery e a segunda usa o seu alias.

```
<script>  
alert('versão do jQuery: ' + jQuery().jquery);  
alert('versão do jQuery: ' + $.jquery);  
</script>
```

A partir de agora sempre que usarmos o símbolo \$() estamos na verdade chamando a principal função da API que é a função jQuery().

Para garantir que todo o código HTML e CSS tenha sido baixado é uma boa prática usarmos a função *ready* do jQuery para garantir que nosso código poderá contar com todos os elementos HTML carregados na árvore DOM. Esse é um recurso essencial ao inicializar o jQuery, pois quando usamos o jQuery é quase certo que acessaremos o HTML ou CSS a partir do jQuery

e, portanto, se o arquivo ainda não foi carregado totalmente, poderíamos ter um erro no jQuery. Então o que fazemos é forçar o jQuery a aguardar o *download* completo do arquivo HTML e CSS antes de realizar algum processamento. Esse comportamento é garantido pela função *ready*. Assim, depois de carregada a página, podemos chamar o método `alert()` para imprimir uma mensagem:

```
<script>
$(document).ready(function(){
    alert('jQuery ok');
});
</script>
```

Se uma janela de alerta apareceu no seu navegador, então você executou seu exemplo jQuery.

Em que local devo colocar o script do jQuery? Durante os exercícios propostos neste livro, você pode colocar o seu código `<script>` dentro da tag `<head>`. Entretanto à medida que você avança no desenvolvimento de sites, o seu código jQuery deve ser colocado em um arquivo externo e nomeado com a extensão `js`. Esse arquivo será importado através da tag *script*, por exemplo:

```
<script src="meuScript.js"></script>
```

No exemplo acima, o seu código ficará localizado dentro do arquivo `meuScript.js`.

2.2 Seletores

Antes de realizar alguma operação no jQuery, precisamos selecionar quais são os elementos ou *tags* alvos, aos quais serão aplicadas as funções do jQuery. Para selecionarmos essas *tags* ou elementos, usamos seletores que possuem uma sintaxe semelhante ao CSS. O operador `$()` é uma função especial do jQuery que permite selecionar qualquer elemento existente na

árvore DOM. Passamos como parâmetro ao `$()` o nome da *tag*, classe ou id que desejamos selecionar. Vejamos alguns exemplos:

Como selecionar todas as *tags* `<p>` de uma página? Basta passar como parâmetro para a função `$()` o nome da *tag*, sem os parênteses `<>`:

```
$("p")
```

Como selecionar todas as *tags* cuja classe é "destaque"? Passamos como parâmetro o nome da classe CSS, lembrando que toda classe começa com "."

```
$(".destaque")
```

Como selecionar a *tag* cujo id é "menu"? Passando o nome do id da *tag* como parâmetro. Todo id inicia-se com o sinal de *hash* ou cerquilha (`#`). Pela especificação do HTML, o id não pode se repetir, portanto só poderá haver uma única *tag* com um determinado id. No exemplo abaixo, a *tag* cujo id é "menu" deve ser única, ou seja, não pode haver outro elemento com esse mesmo id. É muito importante que essa regra seja respeitada para o funcionamento correto do jQuery e CSS.

```
$("#menu")
```

Como selecionar apenas os *links* que estão dentro do menu cujo id é "menu"? Usando o mesmo conceito do CSS, definindo primeiro o elemento menu, seguido de um espaço e do nome da *tag*. A leitura que se faz do seletor abaixo é: retorna todas as *tags* `<a>` contidas no elemento cujo id é "menu".

```
$("#menu a")
```

De fato, o exemplo anterior pode ser ampliado para selecionar qualquer elemento na árvore DOM. Por exemplo, suponha que precisemos selecionar apenas as *spans* que estão contidas num parágrafo localizado dentro de uma div que está localizada na *tag* header:

```
$("#header div p span")
```

Como podemos ver no exemplo acima, os seletores do jQuery são bastante poderosos para permitir encontrar qualquer elemento dentro da árvore DOM. No próximo capítulo veremos como customizar ainda mais os seletores.

A função seletora do jQuery permite a construção de seletores bem mais sofisticados, como busca por atributos, pseudo-classes, entre outros.

2.3 Manipulação de estilo

O jQuery oferece diversos métodos para manipulação do estilo de um elemento HTML. A seguir veremos alguns exemplos, como adicionar e remover classes do CSS, adicionar uma regra CSS, exibir e ocultar um elemento e modificar atributos de uma *tag*.

Função `css()`

A função `css()` permite definir uma ou mais propriedades css para um elemento. No exemplo abaixo definimos a propriedade *background-color* com valor `#CCC` para todos os parágrafos (`p`).

```
$(document).ready(function(){  
    $("p").css("background-color", "#CCC");  
});
```

Além de definir uma propriedade CSS, podemos obter o valor da propriedade CSS que já foi aplicada ao elemento. No exemplo abaixo, quando usamos a função `css()` passando apenas um parâmetro, ela

funcionará como um método `get()` e retornará o valor daquela propriedade, neste caso, o tamanho da fonte:

```
$(document).ready(function){  
    var fontSize = $("p").css("font-size");  
    alert(fontSize);  
});
```

Também é possível alterar mais de uma propriedade CSS em uma só chamada da função `css()`. Basta passar como parâmetro para a função um conjunto de propriedades-valor no formato:

```
{"propriedade":"valor", "propriedade":"valor", ...}
```

Abaixo temos um exemplo de código que aplica uma cor de fundo `#CCC` e, ao mesmo tempo, define uma borda de cor vermelha para um elemento cuja classe é `"two"`

```
<script>  
    $(document).ready(function){  
        $(".two").css(  
            {  
                "background-color":"#CCC",  
                "border":"2px solid red"  
            }  
        );  
    });  
</script>
```

Uma breve pausa aqui!

Uma típica declaração ou comando do jQuery é composto de 4 partes:

1. a função jQuery ou seu alias
2. o(s) seletor(es)
3. a(s) ação(ões)
4. o(s) parâmetro(s)

Por exemplo, considere a declaração:

```
$("p").css("background-color", "#CCC");
```

podemos dividi-la nas seguintes partes:

Função jQuery	Seletor	Ação	Parâmetros
\$	("p")	.css()	("background-color", "#CCC");

A função jQuery sempre chamamos; o seletor permite definir quais elementos serão selecionados; a ação aplica algum método ou função do jQuery sobre os elementos selecionados; os parâmetros definem as propriedades que serão aplicadas na ação.

Ok! Voltando à manipulação de estilos com jQuery.

Função addClass()

A função addClass permite adicionar uma classe CSS a um determinado elemento. É importante destacar que essa classe deve existir no arquivo CSS ou na seção *style*, caso contrário a função não terá nenhum efeito. No exemplo abaixo, todos os parágrafos da página terão associado à classe "destaque":

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <title></title>
  <style>
    .destaque{
      color: red;
      background-color: #F6D8CE;
    }
  </style>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
```

```
    $("p").addClass("destaque");
  });
</script>
</head>
<body>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
  eiusmod.</p>
</body>
</html>
```

Observe que a função `addClass` recebe o nome da classe, sem o sinal de ponto na frente.

A função `addClass` também permite que sejam adicionadas várias classes de uma só vez, por exemplo:

```
<script>
  $(document).ready(function(){
    $("p").addClass("destaque aviso");
  });
</script>
```

Nesse caso, as classes *destaque* e *aviso* são adicionadas ao mesmo tempo no parágrafo.

Função `hasClass()`

Esta função permite verificar se determinado elemento possui uma classe CSS:

```
<script>
$(document).ready(function(){
  var possuiDestaque = $("#pi").hasClass("destaque");
  if(possuiDestaque)
    alert("parágrafo possui destaque");
  else
    alert("classe destaque não detectada");
});
</script>
```

Nesse exemplo, a função *hasClass* retorna *true* caso a classe já esteja associada ao elemento *#pi*, caso contrário, retorna *false*.

Função `removeClass()`

A função *removeClass* faz exatamente o oposto da função *addClass*, ou seja, remove uma classe de determinado elemento:

```
<script>
$(document).ready(function(){
  $("p").removeClass("destaque");
});
</script>
```

No exemplo, removemos a classe *destaque* do elemento `<p>`. Se a classe não estiver associada a `<p>` então o código acima não terá nenhum efeito.

Função `toggleClass()`

A operação de adicionar e remover classes CSS é algo bem comum no jQuery. Um exemplo típico é quando o usuário clica num elemento, e precisamos dar um destaque para o mesmo, como mudar sua cor de fundo. Nesse caso, poderíamos adicionar uma classe CSS que realiza a formatação de destaque, e quando o usuário clica novamente no elemento temos que remover essa classe CSS. Só que isso pode ser redundante e trabalhoso, pois teríamos que fazer um `if` para testar se a classe existe ou não, para antes decidir se usaremos o método `addClass()` ou `removeClass()`. A função `toggleClass()` faz esse controle sem a necessidade de `if`, em uma só linha. Por exemplo:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <title></title>
  <style>
```

```
.destaque{
  color: red;
  background-color: #F6D8CE;
}
</style>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
  $("#manchete").click(function(){
    $(this).toggleClass("destaque");
  });
});
</script>
</head>
<body>
  <p id="manchete">Lorem ipsum dolor sit amet, consectetur
  adipisicing elit, sed do eiusmod
  tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
  minim veniam, quis nostrud exercitation ullamco laboris nisi ut
  aliquip ex ea commodo.</p>
</body>
</html>
```

Nesse exemplo usamos a função `click()` para identificar o evento de clique do mouse (já falaremos mais sobre essa função), e em seguida chamamos `toggleClass` passando o nome da classe CSS. O que o `toggleClass` faz é verificar se o elemento já possui associado a classe "destaque". Caso ele já possua a classe, então `toggleClass` remove a classe. Se não possui ainda a classe, então `toggleClass` adiciona a classe. Portanto, o controle se a classe está associada ou não é feito inteiramente pelo método, deixando o programador livre dessa responsabilidade. Sempre que possível, opte por `toggleClass` ao invés de usar `addClass` e `removeClass`, pois você estará deixando seu código mais legível e conciso.

Pausa para alguns comentários!

Funções anônimas

Em vários exemplos acima usamos um recurso do JavaScript (JS) chamado função anônima. Observe quando usamos a função *ready()*. O que foi passado como parâmetro? Veja o código abaixo:

```
$(document).ready(function){  
    alert('esta é uma função anônima');  
});
```

A função *ready()* está recebendo uma função como parâmetro. O escopo dessa função existe apenas dentro da chamada da função *ready()*. Essa função não possui um nome ou identificador. Seu propósito é existir apenas quando a função *ready()* for chamada. Esse tipo de função é chamada de função anônima. O uso de função anônima é bem comum no jQuery, mas não necessariamente obrigatório. O mesmo código acima poderia ser escrito como:

```
<script>  
$(document).ready(teste);  
  
function teste(){  
    alert("esta é uma função normal do JS");  
}  
</script>
```

Veja que usamos aqui uma função identificada, cujo nome é *teste*. Ela possui uma declaração completa e pode ser chamada de outras partes do código, enquanto que uma função anônima não pode ser chamada. Por isso, na hora de criar suas funções, pense se você precisará usar essa função em outras partes do seu código, pois caso precise, então será melhor atribuir um nome à função para poder reaproveitá-la em outros trechos de código. Nesse caso evite funções anônimas.

Referência *this*

Se você prestou atenção ao exemplo do *toggleClass*, deve ter se perguntado: o que seria a palavra *this*? Vamos considerar outro exemplo que associa

uma função que monitora eventos do tipo click em todos os parágrafos da página:

```
<script>
  $(document).ready(function(){
    $("p").click(function(){
      $(this).toggleClass("destaque");
    });
  });
</script>
```

Nesse exemplo, toda vez que houver um clique num parágrafo, a classe destaque será adicionada ao parágrafo caso ainda não exista, ou removida caso já esteja presente. Mas e o *this* nessa história toda? Observe que no momento que escrevemos o código jQuery, não temos como saber quantos parágrafos existem na página. É provável que exista mais de um parágrafo, ou até dezenas ou centenas. Portanto, se rodarmos o código da seguinte maneira:

```
<script>
  $(document).ready(function(){
    $("p").click(function(){
      $("p").toggleClass("destaque");
    });
  });
</script>
```

estariamos fazendo um método que, quando recebesse um clique em p, aplicaria a classe destaque em todos os parágrafos! Mas não é isso que queremos. O objetivo é aplicar apenas no p que recebeu o clique. E é exatamente isso que o comando *this* faz: definir que o <p> que terá o *toggleClass* aplicado é apenas aquele que recebeu o clique. A referência *this* deixa o código mais genérico, pois podemos trabalhar com uma grande quantidade de elementos, sem sabermos, a princípio, determinar qual é o elemento. Guarde bem esse conceito, pois ele aparecerá muitas e muitas vezes daqui para frente, junto com as funções anônimas.

Ok! Voltando ao tópico de manipulação de estilos.

Função `hide()`

Esta função esconde/oculta um elemento. No exemplo abaixo quando o usuário clicar no parágrafo, o mesmo é ocultado. O usuário não enxergará mais esse parágrafo, entretanto ele continua presente na página e poderá tornar-se visível usando a função `show()`.

```
$('#p').click(function(){  
    $(this).hide();  
});
```

Inspecionando o parágrafo depois de ocultado, vemos que na verdade a função `hide()` apenas define a propriedade *display* do CSS para *none*:

```
<body>  
  <p style="display: none;">...</p>  
</body>
```

Função `show()`

A função `show()` faz o oposto de `hide()`. Se um elemento está escondido, podemos torná-lo visível chamando a função `show()`.

```
$('#botao').click(function(){  
    $('#p').show();  
});
```

O que na prática consiste em modificar a propriedade *display* do CSS, de *none* para *block*.

Função `attr()`

A função `attr()` permite alterar um atributo de um elemento. No exemplo abaixo usamos a função `attr` para alterar a largura da imagem toda vez que o

mouse passa sobre ela. Quando sai de cima da imagem, alteramos a largura para o tamanho original:

```
<script>
$(document).ready(function(){
  $("#photo").mouseover(function(){
    $(this).attr("width",650);
  });
  $("#photo").mouseout(function(){
    $(this).attr("width",620);
  });
});
</script>
```

2.4 Modificando o conteúdo

As funções `text()`, `html()` e `val()` permitem obter e alterar o conteúdo de um elemento. A seguir veremos a diferença entre cada um.

Função `text()`

Esta função permite pegar o texto dentro de uma tag ou colocar um outro texto dentro da mesma. Abaixo temos um exemplo de uma função que copia o texto do segundo parágrafo para o primeiro. Veja que o comando `$("#p2").text()` usa a função `text()` sem passar nenhum parâmetro. Quando não passamos parâmetro, estamos dizendo ao jQuery que queremos que ele retorne o conteúdo daquele elemento. Logo na sequência, o conteúdo de `p2` é salvo na variável `p2Texto`. Na próxima linha, `$("#p1").text(p2Texto)`, estamos usando a mesma função `text()`, mas agora passando um parâmetro. Aqui a função `text()` se comporta de maneira diferente, pois quando definimos um parâmetro o jQuery entende que desejamos alterar o conteúdo do elemento para aquele especificado no argumento do método. Então é nessa linha que o conteúdo de `p2` é copiado para `p1`.

```

<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#copiar").click(function(){
        // Salva conteúdo de p2 em uma variavel:
        var p2Texto = $("#p2").text();
        // Atribui a p1 o conteúdo de p2:
        $("#p1").text(p2Texto);
      });
    });
  </script>
</head>
<body>
  <p id="p1">Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.</p>

  <button id="copiar">Copy p2 -> p1</button>

  <p id="p2">Duis <b>aute</b> irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur.</p>

</body>
</html>

```

Porém, preste atenção no resultado final dessa cópia. É possível afirmar que o método `copy` fielmente o conteúdo? Se observarmos bem, pode-se ver que em p2 temos uma palavra em negrito (aute). Quando o conteúdo é copiado, a palavra em negrito, ou melhor, a tag `<b>` não é copiada, apenas o seu texto. Essa é a principal característica da função `text()`, copiar apenas o texto, retirando todo o código html. Se você precisa copiar o código HTML também, precisamos usar a função `html()` descrita a seguir.

Função html()

Similar a função text(), a função html() permite copiar tanto o texto contido dentro da *tag* como também todo o código html. No exemplo a seguir, o conteúdo do segundo parágrafo é copiado para o primeiro parágrafo, substituindo seu conteúdo original.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <title></title>
  <style>
    b{color:red;}
    i{color:blue;}
  </style>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#copiar").click(function(){
        var p2_conteudo = $("#p2").html();
        $("#p1").html(p2_conteudo);
      });
    });
  </script>
</head>
<body>
  <p id="p1">Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.</p>

  <button id="copiar">Copy p2 -> p1</button>

  <p id="p2">Duis <b>aute</b> irure <i>dolor</i> in reprehenderit
in voluptate velit esse
cillum dolore eu fugiat nulla pariatur.</p>
</body>
</html>
```

Nesse exemplo, o conteúdo de p2, incluindo o código HTML interno, será copiado para p1, ficando ambos exatamente idênticos.

Função val()

A função val() é utilizada para as *tags* input e select. Ela permite definir um valor para a *input* ou retornar o valor contido dentro da input. A função val() funciona como get, quando não passamos nenhum valor como parâmetro. Por outro lado, quando o parâmetro é especificado, a função val() funciona como um método set(). Esse comportamento se repete em várias outras funções do jQuery.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#confirmar").click(function(){
        var novaCor = $("#nome").val();
        $("body").css("background-color",novaCor);
      });
    });
  </script>
</head>
<body>
  <b>Digite o nome de uma cor:</b>
  <input type="text" id="nome" name="nome">
  <button id="confirmar">Aplicar</button>
</body>
</html>
```

Nesse caso, usamos a função val() como *get* para obter o valor digitado pelo usuário na input #nome e em seguida aplicamos esse valor como cor de fundo na *tag* body.

Função is()

A função is() permite testar o estado de um elemento. Por exemplo, suponha que precisemos verificar se uma *checkbox* está marcada ou não, e nesse caso exibir ou esconder uma div:

```
<script>
$(document).ready(function(){
  $("#exibir").click(function(){
    if($(this).is(":checked")){
      $(".infoBox").show();
    }else{
      $(".infoBox").hide();
    }
  });
});
</script>
```

A função is() sempre retorna true ou false de acordo com o seletor passado como parâmetro. No exemplo, o seletor é ":checked" e só faz sentido usá-lo em *tags* do tipo checkbox.

Função toggle()

Existem diversos cenários onde os elementos HTML oscilam entre dois estados como, por exemplo, oculto e visível. Pensando na facilidade de programar tal situação, o jQuery implementa a função toggle() que faz em uma só linha o que seria necessário fazer em duas ou mais com as funções hide() e show(). Vejamos um exemplo:

```
<script>
$(document).ready(function(){
  $("#exibir").click(function(){
    $(".infoBox").toggle();
  });
});
```

```
</script>
```

Nesse exemplo, independente da sequência de cliques, a função `toggle` possui internamente um *if* que testa se o elemento "infoBox" está visível ou não. Caso o elemento esteja visível, a função `toggle` altera seu estado para oculto e, caso esteja invisível, altera seu estado para visível. Compare com o exemplo da função `is()` e veja como a programação fica muito mais enxuta.

2.5 Tratamento de eventos

Existem diversos tipos de eventos que ocorrem numa página web durante a interação do usuário. Por exemplo, quando o usuário move o mouse, clica sobre um elemento, seleciona um texto, digita um texto, seleciona uma opção, etc. O jQuery oferece um vasto arsenal de funções para lidar com esses eventos, possibilitando capturá-los para disparar rotinas de tratamento de eventos associados a uma ação desejada. Por exemplo, toda vez que o usuário passa o mouse sobre uma imagem, essa mesma poderia ser aumentada para facilitar a visualização de seus detalhes. Esse é um exemplo que envolve a programação de algumas etapas, tais como: associar o evento do "mouse sobre" a aquela imagem específica; monitorar quando esse evento é disparado; tomar uma ação para realizar o resultado desejado. Todas essas etapas são realizadas com o tratamento de eventos que o jQuery oferece em sua API.

Evento click

Talvez um dos eventos mais comuns. Ocorre quando o usuário efetua um clique do mouse sobre algum elemento. Vejamos um exemplo de como associar um evento de click a um botão HTML:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
```

```

</style></style>
<script>
    $(document).ready(function(){
        $("#botao").click(function(){
            alert("O botão foi clicado");
        });
    });
</script>
</head>
<body>
    <button id="botao">Clique aqui!</button>
</body>
</html>

```

Quando o usuário clicar no botão, o evento click já estará sendo monitorado pelo jQuery. Além disso, associamos uma função anônima, passada como parâmetro para a função click, que será executada. É nesse ponto que colocamos a ação que pretendemos realizar quando tal evento ocorrer. Por exemplo, vamos supor que toda vez que o click é realizado, a função `toggleClass` é chamada para adicionar ou remover a classe *black* na *tag body*:

```

<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8"/>
    <script src="jquery.js"></script>
    <style>
        .black{
            background-color: #000;
            color: #fff;
        }
    </style>
    <script>
        $(document).ready(function(){
            $("#botao").click(function(){
                $("body").toggleClass("black");
            });
        });
    </script>

```

```
});  
</script>  
</head>  
<body>  
  <button id="botao">Clique aqui!</button>  
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna  
aliqua.</p>  
</body>  
</html>
```

Evento mouseover

Ocorre quando o usuário passa o cursor do mouse sobre determinado elemento. Veja o exemplo a seguir, que modifica a cor de fundo de uma div com id #texto toda vez que o mouse passa sobre a mesma:

```
<script>  
$(document).ready(function(){  
  $("#texto").mouseover(function(){  
    $(this).css("background-color","red");  
  });  
});  
</script>
```

Observe o uso do *this*. É comum o seu uso quando se trabalha com eventos, pois geralmente quando capturamos um evento sobre um determinado elemento, o próprio elemento é o alvo da ação que se realizará logo a seguir. Portanto, nesse caso o *this* se refere ao elemento #texto, que é a div.

Evento change

Este evento pode ser bastante útil para capturarmos quando o estado de um elemento é alterado. Considere que precisamos alterar o *src* de uma imagem de acordo com a opção selecionada em um *select*. Usando o evento *change* fica fácil fazer isso:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#foto").hide();
      $("#modelos").change(function(){
        $("#foto").attr("src",$(this).val());
        $("#foto").show();
      });
    });
  </script>
</head>
<body>
  Selecione o modelo de carro:
  <select id="modelos">
    <option></option>
    <option value="palio.png">Palio</option>
    <option value="corsa.png">Corsa</option>
    <option value="gol.png">Gol</option>
  </select>
  <img src="" id="foto"/>
</body>
</html>
```

Nesse exemplo, a função *change* só é disparada quando o *select* tem seu valor modificado. Assim, basta usar a função *attr()* para modificar o atributo *src* da imagem.

Evento keypress

O evento *keypress* pertence à outra categoria de eventos associados a ações que o usuário realiza através do uso do teclado. A função *keypress* permite monitorar quando uma tecla é pressionada e assim realizar algum tipo de ação. Considere o código a seguir:

```

<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#nome").keypress(function(evento){
        $("#exibe").append("Código ascii da tecla digitada: " +
evento.keyCode + "<br>");
      });
    });
  </script>
</head>
<body>
  Digite seu nome: <input type="text" id="nome"/>
  <div id="exibe">
  </div>
</body>
</html>

```

Nesse exemplo é associado um tratador de eventos do tipo "tecla pressionada" ao input #nome. Assim, toda vez que a tecla é pressionada esse tratador é chamado. Nesse caso, a função de tratamento recebe como parâmetro um objeto chamado "evento" que possui informações sobre o evento da tecla pressionada. Com isso é possível saber exatamente qual foi a tecla pressionada através do atributo *keyCode*. Sabendo disso podemos realizar alguma lógica de decisão como, por exemplo, verificar a tecla digitada pelo usuário e alterar a propriedade CSS da tag *body*:

```

<script>
$(document).ready(function(){
  $("#nome").keypress(function(evento){
    console.log(evento.keyCode);
    // se usuário digita "a" muda cor para azul
    if(evento.keyCode == 97){
      $("body").css("background-color", "blue");
    }
  });
});

```

```
    }  
    // se digita "r" muda cor para vermelho  
    else if(evento.keyCode == 114){  
        $("body").css("background-color", "red");  
    }  
});  
});  
</script>
```

Evitando o envio do formulário

Toda vez que um usuário pressiona a tecla *enter* em um campo dentro de um formulário, a ação padrão do navegador web é enviar o formulário para o servidor. Essa ação pode ser indesejável, principalmente quando o usuário comete o erro de teclar *enter* antes mesmo de completar o preenchimento do formulário. A fim de evitar tal situação, o jQuery oferece uma função que previne o envio do formulário, deixando que o controle de envio do mesmo fique na mão do programador, por exemplo:

```
<script>  
$(document).ready(function(){  
    $("#formulario input[type='text']").keypress(  
        function(evento){  
            // se usuário pressionou tecla enter  
            if(evento.keyCode == 13){  
                // anulando o envio do formulário:  
                evento.preventDefault();  
            }  
        });  
});  
</script>
```

Com a função *preventDefault()* cancelamos o evento de submissão do formulário. Essa mesma lógica pode ser utilizada para realizar validações no formulário e impedir que o mesmo seja enviado caso existam campos com valores inválidos.

Função on()

Outra forma de associar um tratador de eventos a um elemento é através da função on(). De fato, essa função é a forma mais recomendada para se adicionar eventos a um elemento. No exemplo a seguir temos uma input com o tratamento de eventos do tipo *keyup*. Toda vez que uma tecla é liberada, o conteúdo da *input* é copiado para a div #exibe :

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    input{width: 80%; color: green;}
    #exibe{
      border: 1px solid #CCC;
      padding: 20px; margin: 20px;
    }
  </style>
  <script>
    $(document).ready(function(){
      $("#nome").on("keyup",function(e){
        $("#exibe").text($(this).val());
      });
    });
  </script>
</head>
<body>
  Digite algo: <input type="text" id="nome"/>
  <div id="exibe">
  </div>
</body>
</html>
```

Algumas vezes trabalhamos com a criação dinâmica de elementos. Nesse caso, como os elementos são criados depois do código jQuery ter sido executado, a função on() não consegue associar eventos a estes novos

elementos. Sendo assim, temos que fazer a delegação de eventos. Observe o código abaixo. Primeiro, declaramos a região de seleção da função *on()*, neste caso, o elemento *body*. Quando um *keyup* é acionado no elemento cuja classe é "*fieldName*", executamos a função anônima. Deste modo, podemos criar quantos campos "*fieldName*" quisermos, que automaticamente teremos um tratamento de evento associado a este elemento recém criado:

```
<script>
$(document).ready(function(){
  $("body").on("keyup",".fieldName",function(e){
    $("#exibe").text($(this).val());
  });
});
</script>
```

Função off()

Esta função possui o efeito contrário à função *on()*. Ela permite *desassociar* um evento de um elemento. No exemplo abaixo, o elemento *#nome* terá cancelado todos os eventos a ele associados.

```
<script>
$(document).ready(function(){
  $("#desliga").click(function(){
    $("#nome").off();
  });
});
</script>
```

Caso queira cancelar apenas um tipo de evento, deverá especificar qual o evento, passando como parâmetro para a função *off*:

```
$("#nome").off("click");
```

Função trigger()

Esta função permite disparar de modo manual um evento. No código abaixo, a input "#changeBody", quando clicada, altera a formatação da body. Observe que, na última linha da função ready, disparamos um clique usando a função trigger, tendo como alvo o checkbox. Com isso, quando a página for carregada automaticamente, o *checkbox* aparecerá marcado:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#changeBody").click(function(){
        $("body").css({
          "background-color":"black",
          "color":"white"
        });
      });
      $("#changeBody").trigger("click");
    });
  </script>
</head>
<body>
  <input type="checkbox" id="changeBody"/>
  Clique aqui para alterar a body
</body>
</html>
```

Lista de eventos

A tabela a seguir apresenta uma lista resumida de eventos separados por tipos de eventos: (1) eventos de mouse; (2) eventos de teclado e (3) eventos do *browser*. Todos esses eventos podem ser mapeados no jQuery e realizado o devido tratamento.

FUNÇÃO	DESCRIÇÃO
--------	-----------

Eventos de mouse	
.change()	este evento ocorre toda vez que o valor do elemento é alterado
.click()	ocorre quando um click do mouse é detectado num elemento
.dblclick()	evento de clique duplo sobre um elemento
.focus()	ocorre quando um elemento recebe o foco
.hover()	ocorre quando o mouse passa sobre um elemento
.mousedown()	evento disparado quando o botão do mouse é apertado
.mousemove()	evento é acionado toda vez que o mouse se movimento dentro de um elemento
.mouseout()	evento é acionado toda vez que o mouse sai de cima de um elemento
.mouseover()	evento é acionado toda vez que o mouse passa sobre o elemento
.mouseup()	evento é acionado toda vez que o botão do mouse é clicado e liberado
Eventos de teclado	
.keydown()	evento acionado quando uma tecla é apertada
.keyup()	evento acionado quando uma tecla é liberada
.keypress()	evento acionado quando uma tecla é pressionada
Eventos do browser	
.ready()	evento acionado indicando que todo o código HTML foi lido e a árvore DOM está pronta
.resize()	evento acionado quando o tamanho da janela do browser é alterada

Função `bind()` e `unbind()`

A função `bind()` permite associar um "ouvinte" de eventos a um elemento. Por exemplo, no código abaixo definimos um botão "Esconder" que, quando clicado, oculta todas as imagens da página. Para mapear o *click*, que é o gatilho dessa ação, usamos a função `bind`, passando dois parâmetros: o nome do evento e uma função anônima que contém a lógica para esconder as imagens:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <title></title>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#esconder").bind("click",function(){
        $("img").hide();
        alert("imagens ocultadas");
      });
      $("#remover").bind("click",function(){
        $("#esconder").unbind("click");
        alert("botão esconder foi desabilitado");
      });
    });
  </script>
</head>
<body>
  <button id="esconder">Esconder imagens</button>
  <button id="remover">Remove evento</button>
  
  
</body>
</html>
```

Nesse exemplo também usamos a função *unbind*. É o que acontece quando o botão remover é clicado. Aqui fazemos um *unbind* do evento *click* no botão esconder. A função *unbind* remove o ouvinte de um determinado evento. Isso é útil quando precisamos desabilitar uma funcionalidade associada a um evento. A função *bind* é, na verdade, uma função *proxy*, que por "baixo dos panos" chama a função *on()*. Portanto, o mais indicado é usar a função *on()* ao invés da função *bind()*;

Lista de atividades

1. Considere o código HTML abaixo e escreva a função seletora para:
 - a. Selecionar todas as tags
 - b. Selecionar apenas as universidades cuja classe é "b"
 - c. Selecionar a lista de universidades do Paraná

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <title>Lista de universidades</title>
</head>
<body>
  <h2>Lista de universidades federais</h2>
  <span class="estado">Rio Grande do Sul</span>
  <ul id="lista_rs">
    <li class="a">UFRGS</li>
    <li class="b">UNIPAMPA</li>
    <li class="b">UFPEL</li>
  </ul>
  <span class="estado">Paraná</span>
  <ul id="lista_pr">
    <li class="b">UNILA</li>
    <li class="b">UFPR</li>
    <li class="c">UTFPR</li>
  </ul>
</body>
</html>
```

2. Implemente no jQuery a tela abaixo, onde o usuário pode digitar algo, e clicar no botão "alterar". Quando o botão é clicado, o texto digitado é copiado para dentro da div, substituindo o "Texto inicial da div".

Digite algo:

Texto inicial da div

Veja o que acontece depois que o usuário digita algo e clica no botão alterar:

Digite algo:

Algo novo para a div!

3. Continue o exercício 2, colocando agora uma caixa de seleção com 3 opções de cores (azul, amarelo e laranja). Quando o usuário selecionar uma cor, esta deverá ser aplicada como cor de fundo da div.

Digite algo:

Selecione uma cor:

Texto inicial da div

Azul
Amarelo
Laranja

Depois de selecionada a cor, o resultado será:

Digite algo:

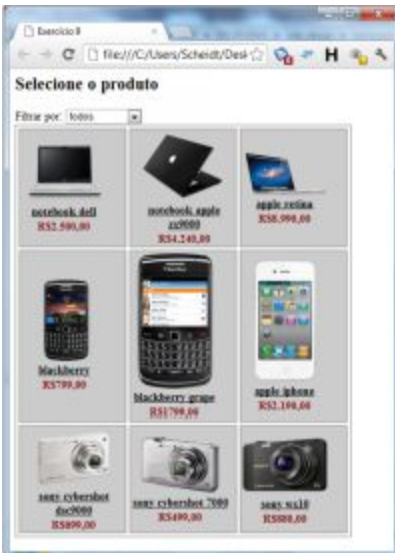
Selecione uma cor: Laranja ▾

Texto inicial da div

4. Faça uma tabela contendo 6 carros que, ao clique do usuário, seleciona a célula marcando-a com fundo vermelho. Se clicar novamente na célula, a célula é desmarcada voltando a sua cor inicial (#CCC).



5. Elabore a página abaixo que, quando o usuário seleciona uma categoria na *select* filtrar, apenas os produtos daquele tipo são exibidos e o restante é ocultado. Faça 3 categorias: notebooks, câmeras e smartphones.



3 Seletores

No capítulo anterior vimos formas de selecionar elementos do DOM usando classe, id e por tipo de tag. Entretanto, o jQuery fornece uma variedade muito maior de seletores de elementos e conhecê-los é essencial para tirar todo o proveito do jQuery. No jQuery existem muitas formas diferentes de selecionar um ou vários elementos e este poderoso recurso é uma das principais vantagens do jQuery. A maioria dos seletores do jQuery usa uma sintaxe semelhante àquela usada no CSS, o que facilita sua compreensão para aqueles que já são familiarizados com o CSS.

3.1 Seletores básicos

Os seletores básicos do jQuery são aqueles mais comumente utilizados. Para os exemplos a seguir, considere o seguinte código HTML:

```
<div id="site">
  <h1>Principais manchetes do dia</h1>
  <p class="destaque">
    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Duis aute
    irure dolor in reprehenderit <span>in voluptate velit</span> esse
    cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat.
  </p>
  <h2 class="titulo">Escolha sua opção</h2>
  <ul id="menu">
    <li><a href="faq.pdf">FAQ</a></li>
    <li><a href="mapa.jpg">Ver mapa</a></li>
    <li><a href="mundo.jpg">Mapa mundo</a></li>
  </ul>
  <h3 class="titulo">Mais conteúdos</h3>
  <span>Acesse:
    <a href="http://www.loremdotcolor.com">aqui</a>
```

```
</span>  
</div>
```

1 Seleciona por ID

Seleciona o elemento cujo id é "menu". O sinal de # é usado para indicar que se trata de um id.

```
$("#menu")
```

A aplicação do seletor acima retorna o elemento :

```
<ul id="menu">  
  <li><a href="faq.pdf">FAQ</a></li>  
  <li><a href="mapa.jpg">Ver mapa</a></li>  
  <li><a href="mundo.jpg">Mapa mundo</a></li>  
</ul>
```

2 Seleciona por classe

Seleciona todos os elementos que possuem uma determinada classe. O uso do ponto como primeiro caractere caracteriza que estamos selecionando elementos pela classe. No exemplo abaixo, estamos selecionando todos os elementos que possuem a classe título associada:

```
$(".título")
```

O resultado do seletor por classe serão os dois elementos abaixo:

```
<h2 class="título">Escolha sua opção</h2>  
<h3 class="título">Mais conteúdos</h3>
```

3 Seleciona por nome da tag

Selecione todos os elementos da página que sejam uma span:

```
$("#span")
```

Esse seletor retornará dois elementos span de acordo com o código exemplo:

```
<span>in voluptate velit</span>  
<span>Acesse:  
  <a href="http://www.loremdotcolor.com">aqui</a>  
</span>
```

4 Seleciona por classe e tag

Selecione apenas os parágrafos cuja classe é destaque:

```
$("#p.destaque")
```

O resultado será:

```
<p class="destaque">  
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis aute  
  irure dolor in reprehenderit <span>in voluptate velit</span> esse  
  cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat.  
</p>
```

5 Seleciona múltiplos elementos

Selecione de uma só vez todas as tags do tipo h1, h2 e h3:

```
$("#h1, h2, h3")
```

O resultado será:

```
<h1>Principais manchetes do dia</h1>
<h2 class="titulo">Escolha sua opção</h2>
<h3 class="titulo">Mais conteúdos</h3>
```

3.2 Selecionando elementos por seus atributos

Neste tipo de seletor o elemento é procurado pelo valor de seu atributo. Considere o código HTML a seguir que usaremos para demonstrar o resultado para cada tipo de seletor:

```
<div id="site">
  Estado origem:
  <input type="text" value="PR"/>

  Estado destino:
  <input type="text" value="RS"/>

  Estado nascimento:
  <input type="text" value="TO"/>

  Escolha um destino:
  <input class="pl_destino" type="text"/>

  Escolha um idioma:
  <input class="pl_idioma" type="text"/>

  Email:
  <input class="validate_and_return" type="text"/>

  <a href="info.pdf">Arquivo</a>
  <a href="index.html">Voltar</a>
</div>
```

6 Seleciona elemento se *value* for igual a determinado valor

Por exemplo, vamos selecionar todas as inputs cujo valor do atributo *value* seja PR:

```
$("#input[value='PR']")
```

O resultado do seletor será:

```
<input type="text" value="PR"/>
```

7 Seleciona se *value* for diferente de um determinado valor

Por exemplo, precisamos selecionar todas as inputs cujos valores sejam *diferentes* de PR:

```
$("#input[value!='PR']")
```

O resultado será todos os inputs com valor diferente de "PR", inclusive as inputs sem valores pré-definidos:

```
<input type="text" value="RS"/>  
<input type="text" value="TO"/>  
<input class="pl_destino" type="text"/>  
<input class="pl_idioma" type="text"/>  
<input class="validate_and_return" type="text"/>
```

8 Seleciona de acordo com o final do valor do atributo

Por exemplo, vamos selecionar todas as inputs cujo atributo *class* termina com a string "idioma".

```
$("input[class$='idioma']")
```

O resultado será apenas 1 elemento selecionado:

```
<input class="pl_idioma" type="text"/>
```

Seleciona todos os links que apontam para um arquivo pdf:

```
$("a[href$='.pdf']")
```

O resultado será:

```
<a href="info.pdf">Arquivo</a>
```

9 Seleciona de acordo com o início do valor do atributo

Vamos selecionar todas as input cujo atributo *class* começa com a string "pl". O sinal ^ indica que o valor deve iniciar pela string "pl":

```
$("input[class^='pl']")
```

O resultado será duas inputs selecionadas:

```
<input class="pl_destino" type="text"/>  
<input class="pl_idioma" type="text"/>
```

10 Seleciona elemento se contém uma substring de valor

Por exemplo, vamos seleccionar todas as inputs cujo atributo *class* contém a substring "and" em qualquer posição:

```
$("input[class*='and']")
```

O seletor encontra 1 tag:

```
<input class="validate_and_return" type="text"/>
```

11 Selecciona se existir um determinado atributo

Por exemplo, vamos seleccionar apenas as inputs que possuem o atributo *value* definido:

```
$("input[value]")
```

No exemplo, foram encontradas 3 tags com value definido:

```
<input type="text" value="PR"/>
<input type="text" value="RS"/>
<input type="text" value="TO"/>
```

3.3 Seleccionando elementos pelo conteúdo

Estes seletores permitem encontrar elementos de acordo com seus conteúdos. Considere o seguinte código HTML para ilustração do resultado retornado para cada seletor:

```
<section id="site">
  <h1>Principais manchetes do dia</h1>
  <p class="destaque">Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempo incididunt ut labore et dolore
```

magna aliqua. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

<div>

Verifique sua opção de escolha:

</div>

<h3 class="titulo">Mais conteúdos</h3>

<div>Acesse:

aqui

</div>

<div id="erro"></div>

</section>

12 Seleciona elemento se contém determinada palavra

Por exemplo, selecione todos os parágrafos que contém a palavra "tempo":

```
$("#p:contains('tempo')")
```

O retorno do seletor será o elemento:

<p class="destaque">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempo incididunt ut labore et dolore magna aliqua. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>

13 Seleciona o elemento que contém uma determinada tag

Por exemplo, vamos selecionar todas as divs que possuem a tag span:

```
$("#div:has(span)")
```

O resultado é a seleção de uma div:

```
<div>
  <span>Verifique sua opção de escolha:</span>
</div>
```

14 Seleciona elementos vazios

Permite selecionar apenas aquelas divs que não possuem conteúdo ou elementos filhos:

```
$("div:empty")
```

O resultado é a div#erro:

```
<div id="erro"></div>
```

15 Seleciona elementos com conteúdo

Por exemplo, seleciona apenas aquelas divs que possuem algum conteúdo ou tag filha:

```
$("div:parent")
```

O retorno do seletor será o elemento:

```
<div>
  <span>Verifique sua opção de escolha:</span>
</div>
<div>Acesse:
  <a href="www.loremipsundotcolor.com">aqui</a>
</div>
```

3.4 Selecionando elementos pela hierarquia

Permitem selecionar elementos de acordo com a posição hierárquica do elemento na árvore DOM. Considere o seguinte código HTML para ilustração do resultado retornado para cada seletor:

```
<div id="site">
<button>voltar</button>
<form>
  <span>Salvar login? </span>
  <input type="checkbox">
  <a href="#">Avaliar custo</a>

  <label for="nome">Nome:</label>
  <input type="text" id="nome"/>
  <label for="cpf">CPF:</label>
  <input type="text" id="cpf"/>

  <button>Cadastrar</button>
  <input type="text" id="email"/>
  <ul>
    <li><a href="#">solicitar ajuda</a></li>
    <li><a href="#">verificar compra</a></li>
    <li><a href="#">confirmar reserva</a></li>
  </ul>
</form>
</div>
```

16 Seleciona por ancestral direto ou indireto

No exemplo, a seguir desejamos selecionar todos os *links* que estão contidos dentro do formulário, sejam filhos diretos ou indiretos:

```
$("form a")
```

O retorno do seletor são 4 links:

```
<a href="#">Avaliar custo</a>  
<a href="#">solicitar ajuda</a>  
<a href="#">verificar compra</a>  
<a href="#">confirmar reserva</a>
```

17 Seleciona por ancestral direto

No exemplo, queremos selecionar apenas os links que são filhos diretos do form:

```
$("form > a")
```

Existe apenas um link associado diretamente ao form. Os outros 3 links são filhos do elemento .

```
<a href="#">Avaliar custo</a>
```

18 Seleciona elemento adjacente

No exemplo, gostaríamos de selecionar todas as input que são adjacentes (que estão ao lado) de um elemento do tipo label:

```
$("label + input")
```

O resultado será as seguintes inputs:

```
<input type="text" id="nome"/>  
<input type="text" id="cpf"/>
```

19 Seleciona elemento irmão

Seleciona todos os inputs que possuem como irmão um button:

```
$("button ~ input")
```

O resultado será:

```
<input type="text" id="email"/>
```

É importante observar que esse seletor seleciona apenas os elementos que são irmãos de button, mas somente a partir dele, não considerando os elementos que antecedem ao button. Ambos devem possuir o mesmo ancestral direto (*form*) em comum.

3.5 Selecionando elementos pelo status

São seletores do jQuery que permitem selecionar determinados elementos de acordo com seu estado, tais como: marcado, selecionado, desabilitado e com o foco do usuário. Considere o código a seguir para demonstração do uso de cada tipo de seletor:

```
<div id="site">
<button>voltar</button>
<form>
  <span>Salvar login? </span>
  <input type="checkbox" checked>

  Número protocolo:
  <input type="text" id="proto" value="910021"
disabled="disable">

  <input type="hidden" id="user_id" value="1949"/>

  País destino:
  <select>
```

```
<option>Brasil</option>
<option selected>Paraguai</option>
<option>Argentina</option>
</select>

<label for="nome">Nome:</label>
<input type="text" id="nome"/>
<label for="cpf">CPF:</label>
<input type="text" id="cpf"/>
</form>
</div>
```

20 Seleciona elemento marcado

Este seletor permite localizar apenas os elementos que foram ou estão marcados. No exemplo, temos um seletor que seleciona apenas as inputs marcadas:

```
$("input:checked")
```

O retorno do seletor será:

```
<input type="checkbox" checked>
```

21 Seleciona elemento que está selecionado

Por exemplo, se quisermos encontrar a opção de um select que foi selecionada pelo usuário, poderíamos definir o seguinte seletor:

```
$("option:selected")
```

O retorno do seletor será o elemento:

```
<option selected>Paraguai</option>
```

22 Seleciona elemento que possui o foco

No exemplo, procuramos apenas a input que possui o foco do usuário, isto é, aquela que tem o cursor de tela no momento atual:

```
$("input:focus")
```

23 Seleciona elemento que está desabilitado

No exemplo, selecionamos apenas as inputs que estão desabilitadas:

```
$("input:disabled")
```

O retorno do seletor será o elemento:

```
<input type="text" id="proto" value="910021" disabled="disable">
```

24 Seleciona elementos visíveis

Este seletor permite selecionar apenas as *tags* visíveis. No exemplo, definimos que apenas as inputs visíveis devem ser selecionadas:

```
$("input:visible")
```

O retorno do seletor será:

```
<input type="checkbox" checked>  
<input type="text" id="proto" value="910021" disabled="disable">  
<input type="text" id="nome"/>
```

```
<input type="text" id="cpf"/>
```

25 Seleciona elementos escondidos

No exemplo, buscamos apenas as inputs que estão ocultas:

```
$("input:hidden")
```

O retorno do seletor será:

```
<input type="hidden" id="user_id" value="1949"/>
```

3.6 Uso de filtros

Filtros atuam em conjunto com os seletores, possibilitando um ajuste ainda precisa definição de critérios para seleção de elementos. Filtros removem determinados elementos, mantendo apenas aqueles necessários.

A tabela a seguir lista alguns filtros e sua descrição:

FILTRO	DESCRIÇÃO
:first	Seleciona o primeiro elemento
:last	Seleciona o último elemento
:even	Seleciona apenas os elementos pares
:odd	Seleciona apenas os elementos ímpares
:eq(n)	Seleciona o enésimo elemento
:gt(n)	Seleciona apenas os elementos cujo index é maior que n
:lt(n)	Seleciona apenas os elementos cujo index é menor que n
:header	Seleciona apenas elementos do tipo título (H1, H2, etc.)
:animated	Seleciona todos os elementos que possuem animação
:not(selector)	Um filtro que nega a condição especificada e, portanto,

seleciona apenas os elementos que não condizem com o critério

A seguir veremos a aplicação de alguns deles. Para cada um dos exemplos abaixo considere o seguinte código HTML:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
</head>
<body>
  <h4>Itens mais elogiados</h4>
  <ul>
    <li>Item A</li>
    <li>Item B</li>
    <li>Item C</li>
    <li>Item D</li>
  </ul>
  <h4>Itens com mais reclamações</h4>
  <ol>
    <li>Item P</li>
    <li>Item L</li>
    <li>Item Q</li>
    <li>Item W</li>
  </ol>
</body>
</html>
```

Filtro $eq(n)$

Seleciona o elemento cujo índice é n , dentro de um conjunto de elementos. No exemplo abaixo, selecionamos o elemento `<li>` cujo índice é igual a 1:

```
<script>
$(document).ready(function(){
  $("li:eq(1)").css("color", "red");
});
```

```
});  
</script>
```

O elemento selecionado será:

```
<li>Item B</li>
```

No jQuery e JavaScript o índice começa em 0. Portanto, a numeração do índice de cada elemento `<li>` seria:

```
<ul>  
  <li>Item A</li> <!-- INDICE 0 -->  
  <li>Item B</li> <!-- INDICE 1 -->  
  <li>Item C</li> <!-- INDICE 2 -->  
  <li>Item D</li> <!-- INDICE 3 -->  
</ul>
```

O índice nada mais é do que a posição de determinado elemento dentro de um vetor de elementos. Como o jQuery trabalha na maior parte do tempo com vetores, quando chamamos `eq(n)` estamos na verdade acessando a posição `n` de um vetor que contém vários objetos que foram selecionados de acordo com o seletor especificado. Por exemplo, se executo o comando:

```
var el = $("li")
```

o jQuery na verdade retornará um vetor de elementos `li`, onde:

```
el[0] == <li>Item A</li>
```

```
el[1] == <li>Item B</li>
```

```
...
```

```
el[7] == <li>Item W</li>
```

Filtro `gt(n)`

Similar ao filtro `eq()`, porém seleciona todos elementos cujo índice seja maior que `n` (*greater than n*). No exemplo abaixo, selecionamos os elementos `<li>` cujos índices são maiores que 3:

```
<script>
```

```
$(document).ready(function(){  
  $("li:gt(3)").css("color","blue");  
});  
</script>
```

Os elementos selecionados serão:

```
<li>Item P</li>  
<li>Item L</li>  
<li>Item Q</li>  
<li>Item W</li>
```

Filtro lt(n)

Similar ao filtro gt(), porém seleciona apenas os elementos cujo índice seja menor que n (*less than* n). No exemplo abaixo, selecionamos os elementos cujos índices são menores que 3:

```
<script>  
$(document).ready(function){  
  $("li:lt(3)").css("color","blue");  
});  
</script>
```

Os elementos selecionados serão:

```
<li>Item A</li>  
<li>Item B</li>  
<li>Item C</li>
```

Filtro not()

O filtro *not* permite negar um seletor, fazendo com que apenas os elementos que não estejam na negação sejam selecionados. No exemplo abaixo é solicitado que sejam selecionados todos os elementos que não estejam dentro de um .

```
<script>
```

```
$(document).ready(function(){
    $("li:not(ul li)").css("color","red");
});
</script>
```

Os elementos selecionados nesse caso serão apenas aqueles dentro da tag `<ol>`:

```
<li>Item P</li>
<li>Item L</li>
<li>Item Q</li>
<li>Item W</li>
```

3.7 Animações e efeitos especiais

O jQuery possui diversas funções que permitem realizar animações básicas com elementos HTML. Estas animações operam sobre propriedades do CSS variando seus valores de modo a criar um efeito de transição entre dois estados como, por exemplo, entre um estado visível e outro oculto.

Funções `fadeIn()` e `fadeOut()`

Tais funções permitem efetuar a animação entre dois estados: 1. totalmente visível e 2. totalmente invisível. Ambas as funções operam sobre a propriedade *opacity* do CSS. A diferença é que a função `fadeIn()` ajusta o nível de opacidade de 0 para 1, de modo gradual, o que simula o efeito de aparecimento. Já a função `fadeOut()` faz exatamente o oposto: ela ajusta a opacidade de 1 para 0, de modo gradual, simulando o efeito de desaparecimento.

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8"/>
    <script src="jquery.js"></script>
    <script>
```

```

$(document).ready(function(){
    $("#desaparecer").click(function(){
        $("#foto").fadeOut();
    });
    $("#aparecer").click(function(){
        $("#foto").fadeIn();
    });
});
</script>
</head>
<body>
    <button id="desaparecer">Desaparecer</button>
    <button id="aparecer">Aparecer</button><br>
    
</body>
</html>

```

Função fadeTo()

Permite especificar a opacidade ou transparência de um elemento entre uma escala de 0 a 1 , onde 0 é totalmente transparente e 1 totalmente visível. No código abaixo, a função fadeTo recebe dois parâmetros: (1) a velocidade(slow / fast) que ocorrerá a animação e (2) o nível da opacidade(entre 0 e 1).

```

<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8"/>
    <script src="jquery.js"></script>
    <script>
        $(document).ready(function(){
            // reduz em 50% a opacidade:
            $("#foto").click(function(){
                $(this).fadeTo("slow",0.5);
            });
        });
    </script>

```

```
</script>
</head>
<body>
  
</body>
</html>
```

Funções `slideUp()` e `slideDown()`

A função `slideUp()` esconde um elemento diminuindo o seu tamanho até desaparecer. A função `slideDown()` faz o contrário, torna o elemento visível aumentando seu tamanho progressivamente até atingir o tamanho original. No exemplo abaixo, a imagem `#foto` é submetida à função `slideUp()` ou `slideDown()` sempre que um dos botões é pressionado:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    #foto{
      border: 1px solid black;
    }
  </style>
  <script>
    $(document).ready(function(){
      // reduz em 50% a opacidade:
      $("#slideup").click(function(){
        $("#foto").slideUp("slow");
      });
      $("#slidedown").click(function(){
        $("#foto").slideDown("slow");
      });
    });
  </script>
</head>
```

```
<body>
  <button id="slideup">SlideUp</button>
  <button id="slidedown">SlideDown</button><br>
  
</body>
</html>
```

Função animate

Permite criar animações sobre propriedades numéricas do CSS. No exemplo a seguir, define-se uma div "barra" que quando clicado no botão "começar" tem sua propriedade *width* aumentada de 30px para 100% da tela. Essa animação tem duração de 5000 milissegundos (ou 5 segundos):

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    #barra{
      margin-top:20px; background-color:lightgreen;
      border:1px solid black; width:30px;
      height:25px;
    }
  </style>
  <script>
    $(document).ready(function(){
      $("#start").click(function(){
        $("#barra").animate({width:"100%"},5000);
      });
    });
  </script>
</head>
<body>
  <button id="start">Começar</button>
  <div id="barra"></div>
```

```
</body>
</html>
```

Para animar duas ou mais propriedades CSS simultaneamente, basta passar como parâmetro para a função `animate()` um conjunto de propriedade-valor entre chaves. Veja o exemplo a seguir, onde a animação opera sobre o `width` e `height` do elemento "barra":

```
<script>
$(document).ready(function(){
  $("#start").click(function(){
    $("#barra").animate(
      {width:"100%",height:"200px"},5000);
  });
});
</script>
```

Função Delay

A função *delay* permite definir um atraso entre uma animação e outra. No exemplo a seguir, a imagem "foto" é primeiramente escondida com a função `slideUp()`. A seguir, a função `delay(2000)` faz uma pausa de 2 segundos antes de executar a função `fadeIn()`.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#start").click(function(){
        $("#foto").slideUp(1000);
        $("#foto").delay(2000);
        $("#foto").fadeIn();
      });
    });
  </script>
```

```
</script>
</head>
<body>
  <button id="start">Começar!</button><br>
  
</body>
</html>
```

3.8 Funções de busca

Função each()

A função `each` permite navegar através de cada um dos elementos do DOM, verificando ou alterando suas propriedades de modo individual se assim necessário. Considere o código a seguir, que permite ao usuário digitar um nome, e à medida que as teclas são digitadas, a função `each` é chamada para percorrer todas as `<li>`s verificando se o conteúdo da `<li>` possui o texto informado pelo usuário:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>Função Each</title>
  <style>
    .off{background-color:#CCC;}
    .on{background-color:red;}
    li{
      border-radius:4px; padding: 8px;
      background-color:navy;
      color:white;display:inline;
    }
    ul{list-style: none;}
    input.busca{
      margin-left: 40px;width: 350px;
      height: 25px;font-size: 16px;
```

```
padding: 4px;
}
</style>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
    $("#filtro").keyup(function(){
        var termo = $('#filtro').val().toLowerCase();

        $('ul.nomes li').each(function(index){
            $(this).addClass('off');
            $(this).removeClass('on');

            var li_texto = $(this).text().toLowerCase();
            if(li_texto.search(termo) != -1
                && termo.length>1){
                $(this).removeClass('off');
                $(this).addClass('on');
            }
        });
    });
});
</script>
</head>
<body>
<input type="text" id="filtro" class="busca"/>
<ul class="nomes">
    <li>Adriana</li>
    <li>Alice</li>
    <li>Fabiana</li>
    <li>Fabiano</li>
    <li>Fernando</li>
    <li>Fabrício</li>
    <li>Marcos</li>
    <li>Maria</li>
    <li>Marta</li>
</ul>
```

```
</body>
</html>
```

O seletor :contains()

Este seletor permite selecionar elementos que contenham uma determinada palavra ou texto. No exemplo a seguir, o seletor procura por todos os parágrafos que contenham a *string* "library" e logo em seguida aplica no texto a cor vermelha. Neste caso, apenas o primeiro parágrafo será selecionado.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("p:contains('library')").css("color","red");
    });
  </script>
</head>
<body>
  <p>
    One important thing to know is that jQuery is just a JavaScript
    library. All the power of jQuery is accessed via JavaScript, so having
    a strong grasp of
    JavaScript is essential for understanding, structuring, and
    debugging your code.
  </p>
  <p>
    While working with jQuery regularly can, over time, improve your
    proficiency with JavaScript, it can be hard to get started writing
    jQuery without a working knowledge of JavaScript's built-in
    constructs and syntax.
  </p>
</body>
```

```
</html>
```

Encadeamento de funções

O jQuery permite que sejam chamadas funções de objetos de modo encadeado, isto é, cada chamada de função opera sobre o resultado retornado pela função anterior. Para exemplificar essa característica, vejamos o código a seguir:

```
$(document).ready(function(){  
    var nav = $("nav.main");  
    var lista = nav.children("ul");  
    lista.css("color","red")  
    lista.children(":first").css("background-color","yellow");  
});
```

O código acima obtém a referência a uma lista contida dentro do elemento nav e modifica seu CSS para deixar o texto na cor vermelha. Logo em seguida, obtém o primeiro elemento filho e aplica a cor amarelo como cor de fundo. Este mesmo código poderia ser reescrito numa única declaração usando chamada encadeada de funções:

```
$("nav.main").children("ul").css("color","red").children(":first").css("background-color","yellow");
```

Função data

Com o uso dos atributos data-* do HTML5 podemos adicionar dados e informações aos elementos. Para acessar estes dados, o jQuery oferece a função data(). Considere o exemplo a seguir:

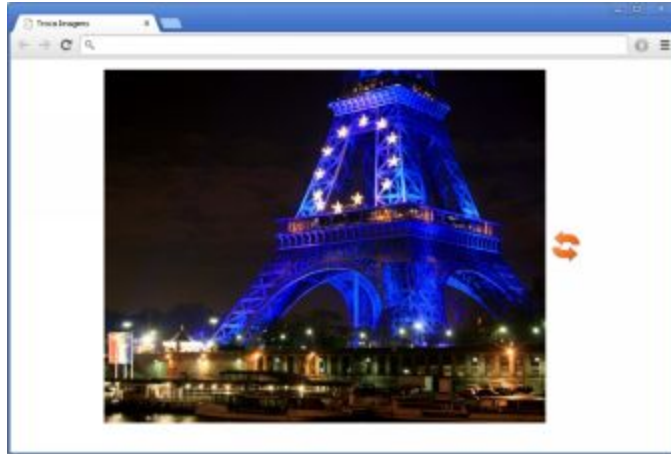
```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8"/>  
    <title>Função Data</title>  
    <script src="jquery.js"></script>
```

```
<script>
$(document).ready(function(){
    var id = $("li:first").data("id");
    var classe = $("li:first").data("classe");
    alert("Informações: ID="+id+"; classe="+classe);
});
</script>
</head>
<body>
<ul class="nomes">
    <li data-id="1100" data-classe="CFT">Adriana</li>
    <li data-id="1200" data-classe="CFT">Alice</li>
    <li data-id="1300" data-classe="ECN">Fabiana</li>
    <li data-id="1400" data-classe="PRI">Fabiano</li>
</ul>
</body>
</html>
```

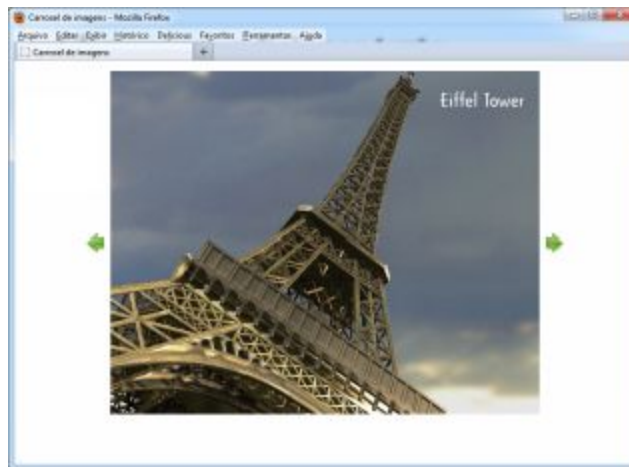
Com a função `data()` foram acessados os atributos " `data-id` " e " `data-classe` " do primeiro elemento `<li>`.

Lista de atividades

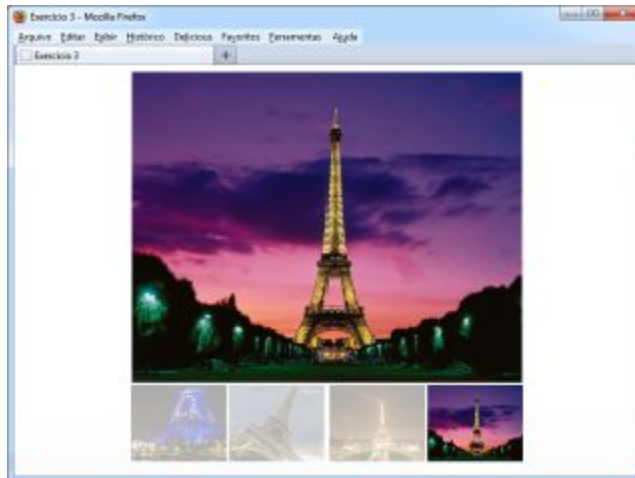
1. Faça uma página com 2 imagens. Quando o usuário clicar no ícone trocar, deverá alterar a imagem principal para outra. Se ocorrer um novo clique, deve voltar a imagem original. Faça com que a cada novo clique as imagens sejam trocadas, assim sucessivamente. Dica: use `attr` para alterar a imagem, e `click` no ícone para efetuar a troca da imagem. Use também `fadeIn()` para criar o efeito de transição entre uma imagem e outra.



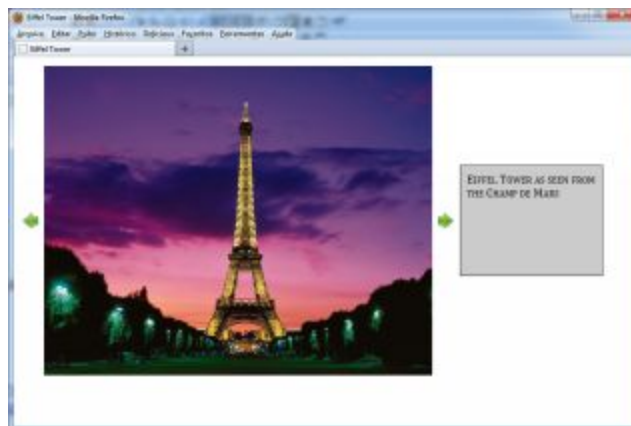
2. Faça uma página com 4 imagens. Quando o usuário clicar nas setas laterais, a imagem central deve ser alterada pela próxima imagem. Dica: use `attr` para alterar a imagem central, e `click` nas setas para avançar ou retornar. Use também `fadeIn()` para criar o efeito de transição entre uma imagem e outra.



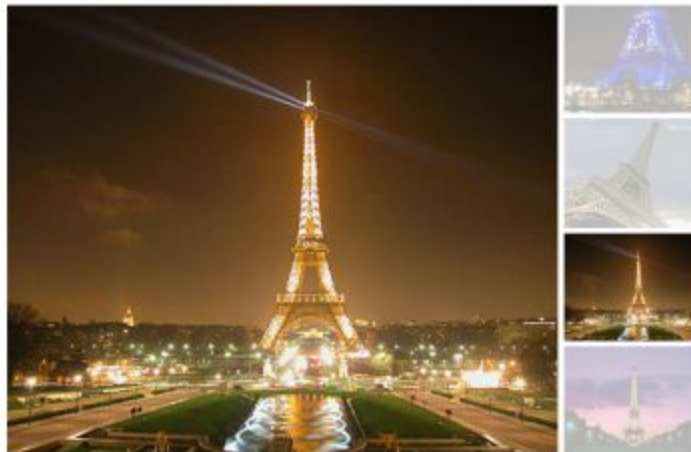
3. Faça uma página que ao clicar na imagem em miniatura, a mesma é expandida para o quadro principal. Observe que quando o clique é feito na miniatura, a mesma fica "ativada" enquanto as outras miniaturas ficam desativadas. Dica: use `fadeTo()` para acertar o nível de opacidade desejado.



4. Faça a página abaixo. Quando o usuário clicar no botão avançar ou retornar, a caixa cinza deve ser preenchida com um texto informativo sobre a imagem.



5. Utilize o evento mouseover para quando o mouse passar em cima da imagem de miniatura, a imagem grande seja alterada.



OceanofPDF.com

4 Navegação e inserção de elementos no DOM

4.1 Criando novos elementos com jQuery

Para criar um novo elemento HTML usando jQuery basta chamar a função `jQuery` ou seu alias `$` e passar uma string como parâmetro. Por exemplo, vamos criar um novo parágrafo e associar uma classe CSS, para depois anexá-lo ao documento HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Criando elementos</title>
  <meta charset="utf-8"/>
  <style>
    body{
      padding-top: 20px;
    }
    p.destaque{
      border: 1px solid red;
    }
  </style>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#criar").click(function(){
        // cria o parágrafo:
        var p = $("<p>Novo parágrafo criado!</p>");
        // associa a classe destaque ao parágrafo:
        p.addClass("destaque");
        // anexa o parágrafo:
```

```
    $("#site").append(p);  
  });  
});  
</script>  
</head>  
<body>  
<div id="site">  
  <button id="criar">Criar parágrafo</button>  
</div>  
</body>  
</html>
```

Vamos detalhar o código acima. O comando abaixo cria o parágrafo e salva este elemento numa variável p:

```
var p = $("<p>Novo parágrafo criado!</p>");
```

A seguir associamos a classe destaque ao parágrafo recém criado. Até aqui este parágrafo ainda não existe no DOM:

```
p.addClass("destaque");
```

O último passo consiste em anexar o parágrafo à árvore DOM. Para isso precisamos escolher um nó pai e anexá-lo nesse ponto. O comando abaixo faz exatamente isso, escolhemos o nó pai, no caso a div #site e usamos a função append() para anexar o parágrafo à div:

```
$("#site").append(p);
```

Criando um elemento customizado

À medida que você se familiariza com o funcionamento do jQuery para criar novos elementos, talvez você comece a se perguntar sobre formas de ter mais controle sobre a criação de novos elementos. Por exemplo, como posso definir um id, um texto para o elemento recém criado ou como associar um evento? A função jQuery permite fazer isso:

```
<script>
```

```

$(document).ready(function){
  $("#criar").click(function(){
    // cria um botão customizado
    var botao = $("<button>",{
      id:'alerta',
      text:'clique aqui',
      click: function(){
        alert("botão foi clicado!");
      }
    });
    $("#site").append(botao);
  });
}
</script>

```

Na função jQuery, especificamos as características do novo elemento usando propriedades no formato chave/valor. No exemplo acima, id é a chave e 'alerta' é o valor, portanto {id:'valor'} representa a definição de um atributo. Cada par chave/valor deve ser separada por vírgula.

4.2 Removendo elementos

Para remover um elemento do documento, basta utilizar a função remove. Vejamos um exemplo:

```

<script>
$(document).ready(function){
  $("p").dblclick(function(){
    $(this).remove();
  });
}
</script>

```

Associamos um evento ao parágrafo, no caso o clique duplo, e quando o mesmo ocorre, removemos o parágrafo específico.

4.3 Navegando no DOM

O jQuery oferece diversas funções que facilitam a navegação no DOM. Com a chamada de algumas funções é possível obter os nodos filhos de um elemento, acessar o elemento mais próximo, encontrar o próximo e o elemento anterior, dentre outras possibilidades.

A função `parent()`

Esta função retorna o elemento pai de um dado elemento. Considere o exemplo abaixo:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    div.destaque{background-color:lightblue;}
  </style>
  <script>
    $(document).ready(function(){
      $(".seleciona").click(function(){
        $(this).parent().toggleClass("destaque");
      });
    });
  </script>
</head>
<body>
  <h4>Escolha seu destino</h4>
  <div class="destino">
    <input class="seleciona" type="checkbox">
    <span>Porto Alegre</span>
  </div>
  <div class="destino">
    <input class="seleciona" type="checkbox">
    <span>Florianópolis</span>
  </div>
</body>
</html>
```

```
</div>
<div class="destino">
  <input class="seleciona" type="checkbox">
  <span>Curitiba</span>
</div>
</body>
</html>
```

Toda vez que um click ocorrer na input "seleciona", a função `parent()` busca o ancestral direto da input, que no exemplo é a div "destino".

A função `children()`

Esta função retorna todos os nodos filhos de um elemento.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    span.destaque{color:red}
  </style>
  <script>
    $(document).ready(function(){
      $(".destino").bind("mouseover mouseout",
        function(){
          $(this).children().toggleClass("destaque");
        });
    });
  </script>
</head>
<body>
  <h4>Escolha seu destino</h4>
  <div class="destino">
    <span>Porto Alegre</span>
    <span><b>|+| Mais informações</b></span>
```

```
</div>
<div class="destino">
  <span>Florianópolis</span>
  <span><b>|+| Mais informações</b></span>
</div>
</body>
</html>
```

Nesse exemplo, associamos dois eventos "mouseover" e "mouseout" às divs com classe ".destino". Quando o evento é disparado, todos os nodos filhos, no caso , e são formatados com a classe "destaque". Essa função permite em uma só linha de comando atingir todos os nodos filhos de uma só vez, o que pode ser bastante útil quando não sabemos exatamente quantos nodos filhos existem.

A função next()

Esta função encontra o próximo elemento a partir de um dado elemento. Por exemplo, vamos criar um botão que ao ser clicado, seleciona o próximo elemento de uma lista:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    li.destaque{background-color:yellow}
  </style>
  <script>
$(document).ready(function(){
  var li_atual = $("#lista li:first");
  $("#prox").click(function(){
    $("li").removeClass("destaque");
    $(li_atual).addClass("destaque");
    li_atual = $(li_atual).next();
  });
});
```

```

});
</script>
</head>
<body>
  <h4>Melhores destinos</h4>
  <button id="prox">Próximo >></button>
  <ul id="lista">
    <li>Paris</li>
    <li>Santiago</li>
    <li>Londres</li>
    <li>Buenos Aires</li>
  </ul>
</body>
</html>

```

Nesse exemplo começamos inicializando a variável `li_atual` apontando para o elemento "`<li>Paris</li>`". Quando executamos `$(li_atual).next()` é equivalente à chamada `$("#<li>Paris</li>").next()`. A função `next()` verifica se existe uma `<li>` que esteja adjacente à direita da `li_atual`, que no caso é "`<li>Santiago</li>`". Sendo assim, a `li_atual` é reconfigurada para apontar para "`<li>Santiago</li>`". Se houver novamente um *click*, o mesmo processamento ocorrerá, somente parando quando chegar em `<li>Buenos Aires</li>` pois nesse ponto não existe um elemento `next()`, ou seja, algum elemento adjacente à direita. Essa função é bastante interessante, pois não precisamos saber exatamente quantos elementos existem, basta chamar a função `next` para navegar ao próximo elemento. A condição de parada é quando `next()` retorna *null*, ou seja, quando não existem mais elementos próximos a serem visitados.

A função `closest()`

Esta função localiza o elemento mais próximo de um determinado elemento de acordo com o parâmetro especificado. Considere o exemplo a seguir:

```

<!DOCTYPE html>
<html lang="pt">
<head>

```

```

<meta charset="UTF-8"/>
<script src="jquery.js"></script>
<style>
  div.destaque{background-color:yellow}
</style>
<script>
$(document).ready(function(){
  $(".seleciona").click(function(){
    $(this).closest("div").toggleClass("destaque");
  });
});
</script>
</head>
<body>
<h4>Escolha seu destino</h4>
<div class="destino">
  <table>
    <tr>
      <td>
        <input class="seleciona" type="checkbox">
      </td>
      <td>Paris</td>
    </tr>
  </table>
</div>
</body>
</html>

```

Observe que a função *closest* recebe como parâmetro o elemento alvo de sua busca. Nesse caso, estamos interessados em localizar a div "mais próxima" da checkbox. Essa div pode ser um ancestral direto (nodo pai) ou um ancestral indireto, como é o caso do exemplo (um ancestral de quarto grau td->tr->table->div). A função *next* vai gradualmente subindo na árvore DOM até encontrar a div mais próxima. Quando localizá-la, adiciona a classe destaque na mesma, caso ainda não possua.

Compare a função *closest* com a função *parent*. De fato, a função *closest* pode trazer o mesmo resultado da função *parent*, entretanto a função *parent*

não consegue obter os mesmos resultados da função `closest`, pois *parent* sempre retorna o ancestral direto, enquanto que `closest` pode retornar o ancestral direto ou indireto.

4.4 Manipulação dinâmica do DOM

Funções para alteração dinâmica do DOM, seja removendo, adicionando ou alterando a hierarquia de elementos da árvore do *document object model*.

A função `wrap()`

Esta função envolve um elemento com outra *tag*. No código abaixo, toda vez que o *button* recebe um clique, a função `wrap()` envolve o mesmo com uma `div`:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <style>
    div{border: 1px solid black;padding:20px;}
  </style>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#bto").click(function(){
        $(this).wrap("<div></div>");
      });
    });
  </script>
</head>
<body>
  <button id="bto">Wrap this!</button>
</body>
</html>
```

O elemento div será criado dinamicamente e adicionado ao DOM, ao entorno do button nesse caso.

Funções insertBefore e insertAfter

A função insertBefore permite anexar um elemento no DOM, exatamente antes de um determinado elemento. A função insertAfter faz o oposto, ela permite anexar um elemento logo depois de um determinado elemento. No exemplo a seguir, temos dois botões. O primeiro insere um elemento p, criado dinamicamente, antes do elemento #menu. O segundo botão insere um elemento p, depois do elemento #menu:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <style>
    div, p{border:1px solid black;margin:10px;}
    #menu{background-color:lightgreen;}
  </style>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#insertB").click(function(){
        $("<p>elemento inserido antes</p>").insertBefore("#menu");
      });
      $("#insertA").click(function(){
        $("<p>elemento inserido depois</p>").insertAfter("#menu");
      });
    });
  </script>
</head>
<body>
  <button id="insertB">InsertBefore P</button>
  <button id="insertA">InsertAfter P</button>
  <h1>Título principal</h1>
  <div id="menu">Porções: arroz, batata frita</div>
```

```
<div id="artigos">Conheça mais sobre culinária</div>
</body>
</html>
```

Função replaceWith

Esta função substitui um elemento por outro. No exemplo a seguir, quando o button #trocar é pressionado, a div "#content" é substituída por um parágrafo. O novo elemento é passado como parâmetro na função replaceWith. Depois que o replaceWith é executado, a div content não existirá mais na árvore DOM. O replaceWith pode ser visto também como a execução de duas operações: (1) remove a div e (2) adiciona o novo elemento.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <style>
    div,p{border:1px solid black;
      margin:10px;background-color:yellow;
    }
    p{background-color:lightgreen;}
  </style>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function(){
      $("#trocar").click(function(){
        $("#content").replaceWith("<p>Novo conteúdo!</p>");
      });
    });
  </script>
</head>
<body>
  <button id="trocar">ReplaceWith p</button>
```

```
<div id="content">Conteúdo da div sobre alguma coisa importante.</div>
</body>
</html>
```

A função clone()

Esta função, como o próprio nome já diz, faz um clone ou uma cópia de determinado elemento. Às vezes, criar seu próprio elemento usando apenas o jQuery pode ser muito trabalhoso, ainda mais se este elemento possui diversos nodos filhos. É o caso do código abaixo, onde sempre que o usuário clica no botão "Novo campo", uma nova linha na tabela é inserida. Se fossemos criar esta linha manualmente, precisaríamos criar 3 inputs, depois anexá-las a 3 <td>s e depois anexá-las a <tr>. Isso acaba sendo muito trabalhoso. A alternativa aqui é usar a função clone, onde apenas especificamos o elemento a ser copiado e a função retorna um novo elemento, independente do elemento original. Depois disso, basta anexá-lo num ponto da árvore DOM:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    tr:nth-child(even){
      background-color: blue;
    }
  </style>
  <script>
    var newTR;
    $(document).ready(function(){
      $("#cloneButton").click(function(){
        newTR = $("#notas tbody tr:first").clone();
        newTR.children().children().val("");
        $("#notas tbody").append(newTR);
      });
    });
  </script>
</html>
```

```

});
});
</script>
</head>
<body>
<button id="cloneButton">Novo campo</button>
<table id="notas">
  <thead>
    <tr>
      <th>Nome</th>
      <th>Nota 1</th>
      <th>Nota 2</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td><input type="text" name="nome"/></td>
      <td><input type="number" name="n1" step="any"
        min="0" max="10"/>
      </td>
      <td><input type="number" name="n2" step="any"
        min="0" max="10"/>
      </td>
    </tr>
  </tbody>
</table>
</body>
</html>

```

Uma observação importante quando se trabalha com a função clone: cuidado ao clonar elementos que possuem um id. Neste caso, ou você não usa id, optando pelo uso de classes, ou antes de anexar o elemento clonado você altera o id desse elemento. Lembre-se que esta é uma regra importante do HTML e que se, não respeitada, poderá comprometer a funcionalidade do jQuery e do JavaScript e o seu script não terá o resultado esperado.

A função load()

Esta função carrega dados do servidor e os anexa diretamente em um elemento HTML. No exemplo abaixo, a função load() carrega o conteúdo do arquivo "roteiros.txt" e o anexa ao elemento #lista. A função *load* é um tipo de chamada ajax. Para conseguir testar esta função é necessário que a mesma esteja dentro de um servidor web como, por exemplo, o apache.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    ul{font-family:monospace;}
    button{display:block;margin: 20px 0px;}
  </style>
  <script>
    $(document).ready(function(){
      $("#prox").click(function(){
        $("#lista").load("roteiros.txt");
      });
    });
  </script>
</head>
<body>
  <h4>Melhores destinos</h4>
  <button id="prox">Carregar lista</button>
  <ul id="lista">
  </ul>
</body>
</html>
```

A função ajax()

Esta função permite gerar requisições do tipo GET ou POST para um servidor de forma assíncrona, isto é, *sem a necessidade de bloquear a*

interface do usuário até o retorno da resposta. Para fazer uma chamada assíncrona precisamos definir alguns parâmetros:

url : define o endereço da página ou arquivo a ser requisitado.

context : define o elemento que será vinculado à requisição, geralmente aquele que receberá os dados retornados pelo servidor.

type : define qual o método de requisição que será utilizado, geralmente GET ou POST.

success : define a função callback, que é executada após a resposta do servidor.

Abaixo temos um exemplo de requisição assíncrona que solicita o arquivo `tabela.txt`, sem bloquear a navegação do usuário. Caso a resposta seja confirmada (com sucesso), uma função anônima é declarada, cujo parâmetro `data` encapsula os dados retornados pelo servidor. Nesse caso, a operação se completa fazendo um `append()` desses dados ao elemento `#lista`.

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <style>
    ul{font-family:monospace;}
    button{display:block;margin: 20px 0px;}
  </style>
  <script>
$(document).ready(function(){
  $("#carregar").click(function(){
    $.ajax({
      url: "tabela.txt",
      context: $("#lista"),
      type: "GET",
      success: function(data){
        this.append(data);
      }
    });
  });
});
```

```
    }  
    });  
    });  
    });  
</script>  
</head>  
<body>  
    <h4>Melhores destinos</h4>  
    <button id="carregar">Carregar lista</button>  
    <ul id="lista">  
    </ul>  
</body>  
</html>
```

Para conseguir testar esta função é necessário que a mesma esteja dentro de um servidor web como, por exemplo, o apache.

4.5 Validação e máscaras

Validação é uma importante etapa durante o processamento e envio de formulários do HTML. Garantir que os campos estejam preenchidos de forma correta e com dados correspondentes ao tipo solicitado são tarefas cotidianas de todo desenvolvedor front-end. Felizmente, o jQuery e seus plugins oferecem uma ampla possibilidade de controlar e alertar ao usuário sobre inconsistências e erros. A seguir veremos alguns exemplos.

O primeiro passo é fazer o download do plugin jquery.validate no seguinte link: <http://ajax.aspnetcdn.com/ajax/jquery.validate/1.13.1/jquery.validate.js>

Salve o arquivo jquery.validate.js na mesma pasta onde se encontra o seu arquivo html e o arquivo jquery.js. Com isso, podemos escrever o código a seguir:

```
<!DOCTYPE html>  
<html lang="pt">  
<head>  
    <meta charset="UTF-8"/>  
    <style>
```

```

.error{color:red}
</style>
<script src="jquery.js"></script>
<script src="jquery.validate.js"></script>
<script>
$( "document" ).ready(function(){
    $( "#formulario" ).validate();
});
</script>
</head>
<body>
<form id="formulario" action="cadastro.php">
Nome:
<input type="text" name="nome" class="required"/>
<br>
CPF:
<input type="text" name="cpf" class="required"/>
<br>
<input type="submit" value="enviar"/>
</form>
</body>
</html>

```

Para fazer a validação dos campos do formulário, precisamos:

1. Atribuir um id para o formulário.
2. Aplicar o método `validate()` no formulário.
3. Em cada campo do formulário que necessita validação, colocar a classe `"required"`.

A classe *required* é o tipo de validação mais simples. Com ela a verificação é feita apenas no caso do campo estar vazio. Existem outras possibilidades mais avançadas, como validação de email, url, datas e números. Observe o código a seguir:

```

<!DOCTYPE html>
<html lang="pt">
<head>
<meta charset="UTF-8"/>

```

```

<style>
.error{color:red}
</style>
<script src="jquery.js"></script>
<script src="jquery.validate.js"></script>
<script>
$("document").ready(function(){
  $("input[type='text']").attr("autocomplete","off");
  $("#formulario").validate();
});
</script>
</head>
<body>
  <form id="formulario" action="cadastro.php">
    Nome:
    <input type="text" name="nome" class="required"/><br>
    CPF:
    <input type="text" name="cpf" class="required"/><br>
    Email:
    <input type="text" name="email" class="required email"/> <br>
    Site:
    <input type="text" name="site" class="required url"/><br>
    Data:
    <input type="text" name="data" class="required date"/><br>
    Nota:
    <input type="text" name="nota" class="required number"/>
    <br>
    Idade:
    <input type="text" name="idade" class="required digits"/><br>
    <input type="submit" value="enviar"/>
  </form>
</body>
</html>

```

Veja que para aperfeiçoar o tipo de validação, basta adicionar uma nova classe ao campo. No exemplo da validação de email, apenas adicionamos a classe email ao atributo *class*. Automaticamente o jQuery passa a validar o campo dentro da regra de sintaxe válida de emails. O mesmo ocorre para os

campos site, data, nota e idade. O detalhe entre nota e idade é que o primeiro será validado como um número, seja inteiro ou real. Já a idade será validada apenas como números inteiros.

Uso de máscaras

É comum em formulários existir o que chamamos de máscaras, que são campos cuja formatação de entrada de dados é pré-definida. Campos que costumam fazer uso de máscara são, por exemplo, cep, cpf e telefone. Além disso, o uso de máscaras facilita a entrada de dados para o usuário.

O primeiro passo para usar máscaras no jQuery é baixar o *plugin Mask*: <http://plugins.jquery.com/mask/>

Dentro do arquivo zip baixado, localize o arquivo jquery.mask.js, copie-o para dentro da mesma pasta do arquivo html e do jquery.js.

Com isso, podemos partir para o seguinte código que aplica uma máscara para os campos cep, cpf e telefone:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8"/>
  <script src="jquery.js"></script>
  <script src="jquery.mask.js"></script>
  <script>
    $("document").ready(function(){
      $('#cep').mask('00000-000');
      $('#fone').mask('(00) 0000-0000');
      $('#cpf').mask('000.000.000-00', {reverse: true});
    });
  </script>
</head>
<body>
  <form id="formulario" action="cadastro.php">
    Telefone: <input type="text" id="fone"/><br>
    CEP: <input type="text" id="cep"/><br>
```

```
CPF: <input type="text" id="cpf"/><br>
<input type="submit" value="enviar"/>
</form>
</body>
</html>
```

Observe que o único trabalho é chamar a função *mask()* para cada uma das inputs, passando como parâmetro o padrão de formatação da máscara, sendo que 0 significa um dígito curinga que representa qualquer número entre 0 e 9 .

Lista de atividades

1. Crie um formulário que aceita como entrada a url de uma imagem e ao clicar enviar, insere a imagem numa div. Dica: use *append*.

Galeria de fotos



2. Utilize a função *append()* para que toda vez que o usuário clicar no botão (+), adicione um novo input do tipo text dentro do formulário. A figura abaixo a direita mostra o resultado, depois que o usuário apertou duas vezes no botão (+).

Formulário

Nome:

Item 1: 

Formulário

Nome:

Item 1: 

Item 2:

Item 3:

3. Faça uma tabela dinâmica, de cadastros de produtos. Toda vez que o botão "Novo produto" é clicado, uma nova linha é adicionada na tabela:

Novo produto

Produto	Preço	Qtde.
Furadeira ▾	digite o preço	quantidade

Veja o exemplo do que ocorre depois que o usuário clicou 3 vezes em novo produto:

Novo produto

Produto	Preço	Qtde.
Furadeira ▾	digite o preço	quantidade
Furadeira ▾	digite o preço	quantidade
Furadeira ▾	digite o preço	quantidade
Furadeira ▾	digite o preço	quantidade

4. Faça um script que permita percorrer uma lista toda vez que o botão "Próximo" é clicado. A lista começa em Paris e termina em Buenos Aires.



5. O botão próximo agora permite mostrar a imagem associada a cada roteiro. Dica: armazene o nome do arquivo da imagem no atributo data-value em cada tag .



5 jQuery UI

5.1 O que é jQuery UI

jQuery UI (*user interface*) é uma biblioteca complementar ao jQuery que utiliza a base de funcionalidades deste para ampliar o conjunto de efeitos especiais, componentes gráficos, temas de layout e estilo com propósito de facilitar a construção de sites profissionais. O jQuery UI é composto de duas partes: código JavaScript e folhas de estilo CSS. Ele possui uma série de componentes como calendários, menus, sliders, caixas de diálogos, entre outros. Neste capítulo veremos alguns desses elementos.

5.2 Principais componentes

A função `autocomplete()`

A função `autocomplete` possibilita oferecer um conjunto de valores para um usuário à medida que o mesmo digita um termo num campo de entrada de dados. Por exemplo, se o usuário digita o termo "ac", a função `autocomplete` busca na sua fonte de dados por strings que contenham como substring esse termo, e retorna todas estas sugestões para o usuário poder selecionar uma dessas. À medida que o usuário digita mais caracteres, a busca vai se reduzindo e sendo filtrada. Vejamos o código a seguir:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="utf-8">
  <title>Autocomplete</title>
```

```
<link rel="stylesheet"
href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-
ui.css">
<script src="//code.jquery.com/jquery-1.10.2.js"></script>
<script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
<script>
$(document).ready(function() {
  var listaDeAlunos = [
    "Amanda","Angelica","Carlos","Carolina",
    "Cristina","Daniel","Daniela","Eduardo",
    "Emanuele","Emanuel","Fabricio","Fabiana",
    "Fatima","Fernando","Gisele","Gabriel"
  ];
  $( "#aluno" ).autocomplete({
    source: listaDeAlunos
  });
});
</script>
</head>
<body>
<div class="ui-widget">
  <label for="aluno">Localizar aluno: </label>
  <input id="aluno">
</div>
</body>
</html>
```

Primeiro observe que estamos importando 3 arquivos essenciais para o funcionamento do script, que são:

jqueryui.js : arquivo JavaScript da biblioteca jQuery UI.

jquery-ui.css : arquivo de estilos usado pela biblioteca jQuery UI.

jquery-1.10.2.js: arquivo com a biblioteca jQuery.

Nesse exemplo usamos o CDN do jQuery, o que significa que esses arquivos não precisam ser baixados para a mesma pasta do arquivo .html. Quando o navegador abrir o arquivo html, estes 3 arquivos serão automaticamente baixados pelo *browser*.

Continuando no exemplo, precisamos agora definir o *source* do autocomplete, isto é, o local onde se encontra os dados que alimentarão a busca. Para isso, foi definido um vetor JavaScript chamado *listaDeAlunos* que contém os nomes dos alunos. Em seguida é vinculada a função *autocomplete* à input "#aluno". Nesse momento a input passa a ser um componente autocomplete e automaticamente um ouvinte de eventos de teclado é associado à mesma. Por fim, é passado por parâmetro o *source* do autocomplete, ou seja, onde serão buscados os valores do autocomplete, no caso na própria variável *listaDeAlunos*.

O resultado será:

Localizar aluno:

Fabricio
Fabiana
Fatima
Fernando

A função *datepicker()*

Com esta função conseguimos criar um calendário no HTML. Este tipo de componente é muito útil para facilitar a seleção de datas pelo usuário. Até a versão 4 do HTML, a única opção de gerar um calendário era via JavaScript.

Data de pagamento:

April 2015

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

O jQuery UI torna muito fácil a tarefa de incorporar um calendário. Vejamos o código:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="utf-8">
  <title>Datepicker</title>
  <link rel="stylesheet" href="jquery-ui/themes/redmond/jquery-
ui.css">
  <script src="jquery-ui/jquery.js"></script>
  <script src="jquery-ui/jquery-ui.js"></script>
  <script>
    $(document).ready(function() {
      $( "#calendario" ).datepicker();
    });
  </script>
</head>
<body>
  <p>Data de pagamento:
    <input type="text" id="calendario">
  </p>
</body>
</html>
```

Basta chamar a função *datepicker()* sobre a input *"#calendario"*. Automaticamente ela passa a ser um *datepicker*.

A função *dialog()*

Esta função permite criar uma caixa de diálogo do tipo modal, onde o restante do site fica em segundo plano e a única opção do usuário é interagir com esta caixa. Considere o seguinte código:

```
<!DOCTYPE html>
<html lang="pt">
<head>
```

```

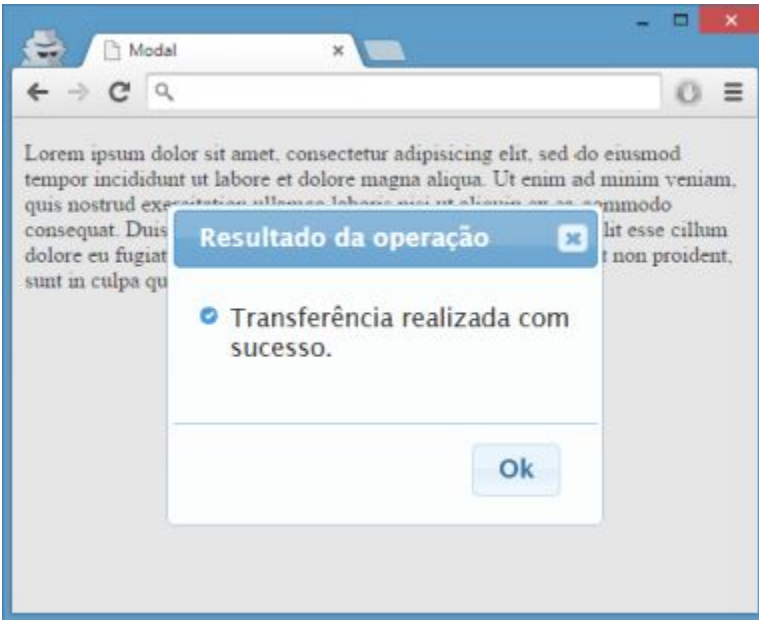
<meta charset="utf-8">
<title>Modal</title>
<link rel="stylesheet" href="jquery-ui/themes/redmond/jquery-
ui.css">
<script src="jquery-ui/jquery.js"></script>
<script src="jquery-ui/jquery-ui.js"></script>
<script>
$(document).ready(function() {
    $("#caixaModal").dialog({
        modal: true,
        buttons: {
            Ok: function() {
                $( this ).dialog("close");
            }
        }
    });
});
</script>
</head>
<body>
<div id="caixaModal" title="Resultado da operação">
    <p>
        <span class="ui-icon ui-icon-circle-check" style="float:left;
margin:0 7px 50px 0;"></span>
        Transferência realizada com sucesso.
    </p>
</div>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.
</p>
</body>
</html>

```

Para criar a modal, basta chamar a função *dialog()* sobre o elemento que será exibido como caixa de mensagem. No caso, usamos a div #caixaModal. É necessário passar como parâmetro para a função *dialog* a

propriedade *modal* com valor *true*, e definir a propriedade *buttons*, especificando os botões que serão anexados na caixa, que neste caso será apenas o botão *ok*. Uma função anônima é associada ao botão, definindo que quando este é executado a caixa de dialogo é fechada.

O resultado do código:



A função tabs()

Esta função cria um componente com várias abas onde a navegação é facilitada, pois diversos conteúdos podem ser colocados na mesma região da tela, sendo alternados por abas intercaladas. Considere o seguinte código:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="utf-8">
  <title>Tabs</title>
  <link rel="stylesheet" href="jquery-ui/themes/redmond/jquery-
  ui.css">
  <script src="jquery-ui/jquery.js"></script>
  <script src="jquery-ui/jquery-ui.js"></script>
```

```

<script>
$(document).ready(function() {
    $( "#tabs" ).tabs();
});
</script>
</head>
<body>

<div id="tabs">
    <ul>
        <li><a href="#tabs1">Horários</a></li>
        <li><a href="#tabs2">Tarifas</a></li>
        <li><a href="#tabs3">Roteiros</a></li>
    </ul>
    <div id="tabs1">
        <p>Horários de funcionamento: das 08h-12h e 14h-18h</p>
    </div>
    <div id="tabs2">
        <p>Valores do ingresso: R$12,00 para turistas e R$7,00 para
        moradores da região </p>
    </div>
    <div id="tabs3">
        <p>Saídas a partir da Argentina, Brasil e Paraguai.</p>
    </div>
</div>
</body>
</html>

```

O resultado do código acima:



A função menu()

Esta função transforma uma lista simples ou encadeada em um componente do tipo menu, com a possibilidade de criar submenus para cada entrada do menu. Veja o exemplo abaixo:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="utf-8">
  <title>Menu</title>
  <link rel="stylesheet" href="jquery-ui/themes/redmond/jquery-
ui.css">
  <script src="jquery-ui/jquery.js"></script>
  <script src="jquery-ui/jquery-ui.js"></script>
  <script>
    $(document).ready(function() {
      $("#opcoes").menu();
    });
  </script>
  <style>
    .ui-menu { width: 150px; }
  </style>
</head>
<body>
```

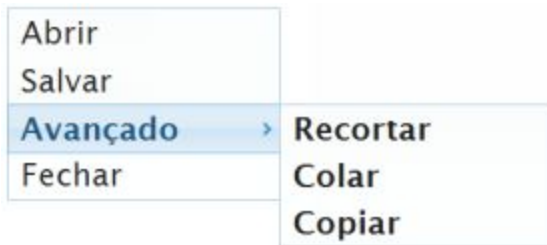
```

<ul id="opcoes">
  <li>Abrir</li>
  <li>Salvar</li>
  <li>Avançado
    <ul>
      <li>Recortar</li>
      <li>Colar</li>
      <li>Copiar</li>
    </ul>
  </li>
  <li>Fechar</li>
</ul>
</body>
</html>

```

Como é possível observar, o menu nada mais é do que uma lista identificada por um id. No jQuery, basta chamar a função `menu()` para a lista `#opcoes`.

Resultado do código:



A função `accordion()`

A função *accordion* permite criar um efeito sanfona entre várias divs. Uma div começa visível, e à medida que o usuário clica em outras divs, o *accordion* fecha uma e expande outra. Vejamos um exemplo:

```

<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="utf-8">
  <title>Accordion</title>

```

```

<link rel="stylesheet" href="jquery-ui/themes/redmond/jquery-
ui.css">
<script src="jquery-ui/jquery.js"></script>
<script src="jquery-ui/jquery-ui.js"></script>
<script>
$(document).ready(function() {
$( "#accordion" ).accordion(
{heightStyle: "content"}
);
});
</script>
</head>
<body>
<div id="accordion">
<h3>Turismo</h3>
<div>

<p style="margin-left:160px;">

```

As cataratas é um conjunto de cerca de 275 quedas de água no Rio Iguaçu localizada entre o Parque Nacional do Iguaçu Paraná no Brasil 20%, e o Parque Nacional Iguazú em Misiones, na Argentina 80%, na fronteira entre os dois países.

```

</p>
</div>
<h3>Horários</h3>
<div>
<p>

```

Sed non urna. Donec et ante. Phasellus eu ligula. Vestibulum sit amet purus. Vivamus hendrerit, dolor at aliquet laoreet, mauris turpis porttitor.

```

</p>
</div>
<h3>Passeios</h3>
<div>
<p>

```

Nam enim risus, molestie et, porta ac, aliquam ac, risus. Quisque lobortis. Phasellus pellentesque purus in massa. Aenean

**in pede. Phasellus ac libero
ac tellus pellentesque semper.**

</p>

</div>

</div>

</body>

</html>



OceanofPDF.com

Referências

1. CASTLEDINE, E.; SHARKIE, C. *jQuery Novice to Ninja: New Kicks and Tricks*. Ed. Sitepoint, 2012.
2. CODY, L. *Dom Enlightenment*. Ed. O'Reilly, 2013.
3. DAYLEY, B. *jQuery and JavaScript: Phrasebook*. Ed. Addison-Wesley, 2013.
4. JQUERY. *jQuery Documentation API*. <http://api.jquery.com>
5. MACRAE, C. *Learning from jQuery*; Ed. Oreilly, 2013.
6. POLLOCK, J. *jQuery: A Beginner's Guide*. MC Graw Hill Education, 2014.