CAPÍTULO 12

jQuery

"O primeiro problema para todos, homens e mulheres, não é aprender, mas desaprender"
— Gloria Steinem

Por conta das dificuldades enfrentadas pelos programadores JavaScript para páginas Web, foi criada uma biblioteca que traz diversas funcionalidades voltadas à solução dos problemas mais difíceis de serem contornados com o uso do JavaScript puro.

A principal vantagem na adoção de uma biblioteca de JavaScript é permitir uma maior compatibilidade de um mesmo código com diversos navegadores. Uma maneira de se atingir esse objetivo é criando funções que verificam quaisquer características necessárias e permitam que o programador escreva um código único para todos os navegadores.

Além dessa vantagem, o jQuery, que é hoje a biblioteca padrão na programação front-end para Web, traz uma sintaxe mais "fluida" nas tarefas mais comuns ao programador que são: selecionar um elemento do documento e alterar suas características.

12.1 – JQUERY – A FUNÇÃO $

O jQuery é uma grande biblioteca que contém diversas funções que facilitam a vida do programador. A mais importante delas, que inicia a maioria dos códigos, é a função $.

Com ela é possível selecionar elementos com maior facilidade, maior compatibilidade, e com menos código. Por exemplo:

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

```
if (cabecalho.attachEvent) {
  cabecalho.attachEvent("onclick", function (event) {
    alert("Você clicou no cabeçalho, usuário do IE!");
  });
} else if (cabecalho.addEventListener) {
  cabecalho.addEventListener("click", function (event) {
    alert("Você clicou no cabeçalho!")
  }, false);
}

// jQuery
$("#cabecalho").click(function (event) {
  alert("Você clicou no cabeçalho!");
});
```

Note como a sintaxe do jQuery é bem menor, e a biblioteca se encarrega de encontrar o modo mais compatível possível para adicionar o evento ao elemento cujo id é `cabecalho`.

Existem diversas funções que o jQuery permite que utilizemos para alterar os elementos que selecionamos pela função **$**, e essas funções podem ser encadeadas, por exemplo:

```
$("#cabecalho").css({"margin-top": "20px", "color": "#333333"})
      .addClass("selecionado");
```

No código acima, primeiramente chamamos a função `$` e passamos como argumento uma String idêntica ao seletor CSS que utilizaríamos para selecionar o elemento de id `cabecalho`. Na sequência chamamos a função `css` e passamos um objeto como argumento, essa função adicionará ou alterará as informações desse objeto como propriedades de estilo do elemento que selecionamos com a função `$`. Em seguida chamamos mais uma função, a `addClass`, que vai adicionar o valor "selecionado" ao atributo `class` do elemento com o id "cabecalho".

Dessa maneira, é possível fazer muito mais com muito menos código, e ainda por cima de uma maneira que funciona em diversos navegadores.

12.2 – jQuery Selectors

Um dos maiores poderes do jQuery está na sua capacidade de selecionar elementos a partir de seletores CSS.

Como já aprendemos, exitem diversas formas de selecionarmos quais elementos ganharão determinado estilo. Infelizmente muitos desses seletores

não funcionam em todos os navegadores. Contudo, no jQuery, os temos todos à nossa disposição.

Por exemplo, se quisermos esconder todas as tags <td> filhas de um <tbody>, basta:

```
$('tbody td').hide();
```

Seletores mais comuns:

```
// pinta o fundo do formulario com id "form" de preto
$('#form').css('background', 'black');

// esconde todos os elementos com o atributo "class" igual a "headline"
$('.headline').hide();

// muda o texto de todos os parágrafos
$('p').text('alô :D');
```

Mais exemplos:

```
$('div > p:first'); // o primeiro elemento <p> imediatamente filho de um
<div>

$('input:hidden'); // todos os inputs invisíveis

$('input:selected'); // todas as checkboxes selecionadas

$('input[type=button]'); // todos os inputs com type="button"

$('td, th'); // todas as tds e ths
```

Lembre-se de que a função que chamamos após o seletor é aplicada para **todos** os elementos retornados. Veja:

```
// forma ineficiente
alert($('div').text() + $('p').text() + $('ul li').text());

// forma eficiente :D
alert($('div, p, ul li').text());
```

A função `text()` é chamada para todos os `<div>`s, `<p>`s, e `<li>`s filhos de `<ul>`s.

[Faça sua pergunta](#).

12.3 – Filtros customizados e por DOM

Existem diversos seletores herdados do css que servem para selecionar elementos baseados no DOM. Alguns deles são:

```
$('div > p'); // <p>s imediatamente filhos de <div>
$('p + p'); // <p>s imediatamente precedidos por outro <p>
$('div:first-child'); // um elemento <div> que seja o primeiro filho
$('div:last-child'); // um elemento <div> que seja o último filho
$('div > *:first-child'); // um elemento que seja o primeiro filho direto de
uma <div>
$('div > *:last-child'); // um elemento que seja o ultimo filho direto de uma
<div>
$('div p:nth(0)'); // o primeiro elemento <p> filho de uma <div>
$('div:empty'); // <div>s vazios
```

12.4 – Utilitário de iteração do jQuery

O jQuery traz também entre suas diversas funcionalidades, uma função que facilita a iteração em elementos de um Array com uma sintaxe mais agradável:

```
$("#menu-departamentos li").each(function (index, item) {
  alert(item.text());
});
```

A função `each` chamada logo após um seletor executa a função que passamos como argumento para cada um dos itens encontrados. Essa função precisa de dois argumentos. O primeiro será o "índice" do elemento atual na coleção (0 para o primeiro, 1 para o segundo e assim por diante), e o segundo será o próprio elemento.

Também é possível utilizar a função `each` do jQuery com qualquer Array:

```
var pessoas = ["João", "José", "Maria", "Antônio"];

$.each(pessoas, function(index, item) {
  alert(item);
})
```

Nesse caso, chamamos a função `each` diretamente após o `$`, pois essa implementação é um método do próprio objeto `$`. Passamos dois argumentos, o primeiro é o Array que queremos percorrer e o segundo a função que desejamos executar para cada um dos itens do Array.

12.5 – Características de execução

Para utilizarmos o jQuery em nossos projetos com maior segurança, devemos tomar alguns cuidados.

Importação

Antes de mais nada é necessário incluir o jQuery em nossa página. Só assim o navegador executará seu código para que possamos utilizar suas funcionalidades em nosso código.

Por isso é necessário que a tag `<script>` do jQuery seja a primeira de todas na ordem de nosso documento:

```
<script src="scripts/jquery.js"></script>
<!-- só podemos utilizar o jQuery após sua importação -->
<script src="scripts/meuscript.js"></script>
<script src="scripts/meuoutroscript.js"</script>
```

Executar somente após carregar

Como estamos constantemente selecionando elementos do documento e alterando suas características, é importante garantir que os elementos que pretendemos utilizar já tenham sido carregados pelo navegador.

A melhor maneira de garantir isso é somente executar nosso script após o término do carregamento total da página com a função `$` dessa maneira:

```
$(function() {
  $("#cabecalho").css({"background-color": "#000000"});
})
```

Essa função `$` que recebe uma função anônima como argumento garante que o código dentro dela só será executado ao fim do carregamento de todos os elementos da página.

12.6 - Mais produtos na home

Uma técnica comum de se implementar com JavaScript é a de permitir mais conteúdo ser mostrado na tela a partir de algum clique ou até ao se passar o mouse em cima.

Na nossa página, exibimos 6 produtos em cada painel de destaque. Poderíamos criar um botão para "Mostrar mais" produtos que exiba outros 6.

Para implementar, a maneira mais simples é inserir esses produtos adicionais no HTML e escondê-los com CSS usando `display:none`. Aí colocamos o botão de *Mostrar Mais* e, via JavaScript, exibimos quando o usuário clicar.

Carregamento de conteúdo com Ajax

No nosso exercício, vamos apenas esconder ou exibir o conteúdo usando CSS e JavaScript. Em alguns casos, pode ser interessante baixar conteúdo novo do servidor no momento do clique.

Esse tipo de página usa Ajax para requisitar novos dados ao servidor e inseri-los dinamicamente na página via JavaScript. Ajax e outras técnicas de JavaScript avançadas são tópicos do curso WD-47 da Formação Web da Caelum:

http://www.caelum.com.br/curso/wd47

12.7 - Exercícios: jQuery na home

1. Aumente a quantidade de produtos exibidos nos painéis da home para 12. Para isso, basta alterar o parâmetro na busca (ao lado do LIMIT).

Teste novamente a página e veja que são mostrados muitos produtos. Depois,

vamos esconder a metade e mostrar apenas se o usuário quiser ver.

2. Crie um `<button>` no final de cada DIV `painel`, logo após a lista `<ol>`. Esse será o botão responsável por exibir os produtos.

```
<button type="button">Mostra mais</button>
```

No estilos.css, esconda esse botão por padrão. Ele só vai ser exibido quando os produtos adicionais estiverem colapsados.

```
.painel button {
  display: none;
}
```

Repare como ainda não fizemos a funcionalidade em JavaScript para mostrar os produtos. Mas a página é usável e válida mesmo nesse caso. A ideia é que, na falta de JavaScript, todos os produtos sejam exibidos e o botão esteja escondido.

3. Implemente a funcionalidade de compactar o painel de produtos para mostrar apenas os 6 primeiros por padrão. Vamos fazer isso com CSS, através de uma nova classe `painel-compacto`.

São duas coisas: esconder os produtos a mais, e exibir o botão que vai fazer a funcionalidade.

```
.painel-compacto li:nth-child(n+7) {
  display: none;
}
.painel-compacto button {
  display: block;
}
```

Essa classe, claro, só vai fazer efeito se adicionarmos ela na página. Para testar, vá no div com classe `painel` e **adicione** a classe `painel-compacto` do lado.

4. Estamos sem JavaScript ainda na página. E, já que adicionamos a classe `painel-compacto` direto no HTML, quebramos a experiência do usuário nesse caso. Perceba que os produtos adicionais ficam escondidos e botão aparece.

Mas nada funciona! Péssima experiência.

Claro que, nesse caso, é porque não implementamos ainda. Mas imagine o cenário onde, mesmo com tudo implementado, o JavaScript não carrega, acontece um erro ou o usuário desabilitou.

5. Vamos implementar a funcionalidade em JavaScript. O primeiro passo é

remover a classe `painel-compacto` do HTML. Como ela é uma classe atrelada a funcionalidade JS, vamos adicioná-la com jQuery, apenas se o JS for executado.

Primeiro, vamos importar o jQuery na home. Inclua a seguinte linha **imediatamente antes** da importação do home.js:

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

Agora, no **home.js**, faça:

```
$('.novidades').addClass('painel-compacto');
```

O resultado visual parece o mesmo. Mas reflita sobre as implicações de progressive enhancement, essencial para um projeto de qualidade.

6. Ainda no **home.js**, implemente o evento de clique no botão. Ele deve remover a classe `painel-compacto`, fazendo o produto aparecer:

```
$('.novidades button').click(function() {
  $('.novidades').removeClass('painel-compacto');
});
```

Teste a funcionalidade.

7. (opcional) Implemente a mesma funcionalidade para o painel da direita, o **mais-vendidos**.

8. (opcional trabalhoso) Podemos estilizar o botão de mostrar mais produtos com regras CSS3 que aprendemos. Uma sugestão:

```
.painel button {
  /* posicionamento */
  float: right;
  margin-right: 10px;
  padding: 10px;

  /* estilo */
  background-color: #333;
  border: 0;
  border-radius: 4px;
  box-shadow: 1px 1px 3px rgba(30,30,30,0.5);
  color: white;
  font-size: 1em;
  text-decoration: none;
  text-shadow: 1px 0 1px black;

  /* animação*/
  transition: 0.3s;
}
.painel button:hover {
  background-color: #393939;
```

```
  box-shadow: 1px 0 20px rgba(200,200,120,0.9);
}
```

E até mais frescuras, se estiver disposto:

```
.painel button {
  color: white;
  position: relative;
  margin-bottom: 10px;
}
.painel button:after {
  /* elemento vazio */
  content: '';
  display: block;
  height: 0;
  width: 0;

  /* triangulo */
  border-top: 10px solid #333;
  border-left: 10px solid transparent;
  border-right: 10px solid transparent;

  /* posicionamento */
  position: absolute;
  top: 100%;
  left: 50%;
  margin-left: -5px;

  /* animação */
  transition: 0.3s;
}
.painel button:hover:after {
  border-top-color: #393939;
}
```

12.8 – O elemento output do HTML5

Na página de produto havíamos criado um input range para selecionar o tamanho da roupa. O problema é que não há feedback visual de qual valor está selecionado. Podemos então criar um outro elemento visual na página apenas para mostrar o valor atualmente selecionado no range.

Vamos usar JavaScript pra isso. Na prática, escutamos o evento **onchange** do range e toda vez que ele mudar de valor, pegamos o valor mais atual e copiamos pra esse novo elemento visual exibir.

Mas que tag usar pra representar esse elemento cujo valor é resultado do valor escolhido no range?

No HTML5, temos uma tag nova com valor semântico exato pra essa situação: o

<output>. Essa tag representa a saída de algum cálculo ou valor simples obtido a partir de um ou mais componentes de um formulário. Ele tem um atributo **for** que aponta de qual elemento saiu o valor.

```
<output for="tamanho" name="valortamanho">42</output>
```

Visualmente, é como se fosse um DIV simples. Depois vamos estilizar esse componente do jeito que quisermos com CSS. A grande sacada é o *valor semântico* da tag e o que ela representa.

O valor em si está como 42 estaticamente. O que precisamos é atualizar esse valor toda vez que o valor do input range mudar. Pra isso, precisamos de JavaScript. E vamos implementar com jQuery no exercício a seguir.

12.9 - Exercícios: mostrando tamanho do produto com jQuery

1. Na página **produto.php**, adicione o elemento output do HTML5 logo após o input range, ainda dentro do `fieldset` de escolha de tamanho.

```
<output for="tamanho" name="valortamanho">42</output>
```

Repare que esse elemento não tem visual específico e também não atualiza seu valor sozinho. Vamos implementar isso via JavaScript, usando jQuery.

2. O preenchimento inicial e atualização do valor no output deve ser feita via JavaScript. É bastante simples: quando o input range mudar de valor (evento `oninput`), pegamos seu valor e jogamos no output.

Para escrever o JavaScript, você pode criar um novo arquivo **produto.js** e importá-lo na página. Ou, como o código é bem pequeno, se preferir, pode escrever direto

num elemento `<script>` na página.

O importante é **importar o jQuery antes do nosso código**, como fizemos antes:

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

O nosso código é:

```
$('[name=tamanho]').on('input', function(){
  $('[name=valortamanho]').val(this.value);
});
```

Teste o funcionamento no slider no range, veja se o output atualiza de valor corretamente.

> **IE10**
>
> Para suportar o IE10, precisamos colocar o evento `onchange`. O correto no HTML5 seria usar o evento `oninput`, que até funciona melhor nos browsers modernos. Se quiser suportar os dois no jQuery, podemos atrelar dois eventos à mesma função de uma só vez:
>
> ```
> $('[name=tamanho]').on('change input', function(){ ...
> ```
>
> Além disso, como o elemento `output` não é corretamente reconhecido pelo navegador, alterar a propriedade `value` dele não vai ter o resultado esperado. Para o nosso código funcionar nele, precisamos mexer diretamente no texto do elemento:
>
> ```
> $('[name=valortamanho]').text(this.value);
> ```

3. Estilize o `output` para ter um design mais ajustado a nossa página de produto:

```
.tamanhos output {
  display: inline-block;
  height: 44px;
  width: 44px;

  line-height: 44px;
  text-align: center;

  border: 3px solid #666;
  border-radius: 50%;
  color: #555;
}
```

12.10 – Plugins jQuery

Além de usar os componentes JavaScript que vêm prontos no Bootstrap, podemos baixar outros prontos. São plugins feitos para o jQuery ou para o Bootstrap que trazem novas funcionalidades.

A grande riqueza do jQuery é justo sua vasta gama de plugins disponíveis. Há até um diretório no site deles:

http://plugins.jquery.com/

Cada plugin é um arquivo JavaScript que você inclui na página e adiciona uma funcionalidade específica. Muitos exigem que escrevamos um pouco de código pra configurar seu uso; outros são mais plug and play.

Você vai precisar consultar a documentação do plugin específico quando for usar.

12.11 – Exercícios: plugin

1. Um plugin que podemos usar na nossa página é **máscaras numéricas** para digitar em campos como CPF ou CEP. Isso ajuda bastante o usuário.

Para usar esse plugin, basta invocar seu arquivo JavaScript no final da página do checkout, logo após a chamada do jQuery e do bootstrap.js:

```
<script src="js/inputmask-plugin.js"></script>
```

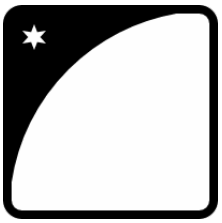
2. Cada campo que for usar uma máscara numérica precisa definir o atributo **data-mask** com o formato a ser usado.

No `<input>` do CPF, adicione o atributo:

```
data-mask="999.999.999-99"
```

3. Nos campos do número do cartão com código de verificação, podemos usar:

```
data-mask="9999 9999 9999 9999 - 999"
```



Ruby, Agile, Mobile, Web e .NET.

Faça curso com **quem escreveu essa apostila**.

Consulte as vantagens do curso *Desenvolvimento Web com HTML, CSS e JavaScript*.

12.12 – Exercícios opcionais

1. Refaça os exercícios da home da validação da busca e do banner usando jQuery. Tente ver como o código fica bem menor e mais legível.

CAPÍTULO ANTERIOR:

Bootstrap e formulários HTML5

PRÓXIMO CAPÍTULO:

Integrações com serviços Web

Facebook

Newsletter

Casa do Código

Twitter