

JQuery

A Biblioteca do Programador JavaScript

Maurício Samy Silva



Novatec



CAPÍTULO 1

O que é jQuery?

Neste capítulo, faremos uma introdução à biblioteca jQuery, relatando suas origens, finalidades e destinação. Faremos um breve relato histórico de sua evolução, examinando as motivações que resultaram em sua criação e descreveremos algumas de suas possibilidades dando uma idéia do seu potencial e mostrando a força e poder da programação com jQuery.

1.1 Definições e conceitos

1.1.1 O que é jQuery?

jQuery é uma biblioteca JavaScript criada por John Resig e disponibilizada como software livre e aberto, ou seja, de emprego e uso regido segundo licença conforme as regras estabelecidas pelo MIT – Massachusetts Institute of Technology ou pelo GPL – GNU General Public License. Isso, resumidamente, significa que você pode usar a biblioteca gratuitamente tanto em desenvolvimento de projetos pessoais como comerciais. Para maiores detalhes sobre esses tipos de licença consulte os seguintes endereços na internet:

- http://pt.wikipedia.org/wiki/Mit_license
- http://pt.wikipedia.org/wiki/GNU_General_Public_License

John Resig no prefácio do livro *jQuery in Action* diz, traduzido para o português, o seguinte:

“O foco principal da biblioteca jQuery é a simplicidade. Por que submeter os desenvolvedores ao martírio de escrever longos e complexos códigos para criar simples efeitos?”

E, sem dúvida, John Resig estava em um momento de rara inspiração quando assim escreveu, pois isso muito bem resume jQuery: uma maneira simples e fácil de escrever JavaScript colocada ao alcance não só de programadores experientes como também de designers e desenvolvedores com pouco conhecimento de programação.

E o que torna o estudo e entendimento de jQuery mais fascinante ainda é o fato de que você, não precisa ser um profundo conhecedor de JavaScript, por mais estranho que isso possa parecer, pois se trata de uma biblioteca criada com base nesta programação.

Simplicidade é a palavra-chave que resume e norteia o desenvolvimento com jQuery. Linhas e mais linhas de programação JavaScript escritas para obter um determinado efeito em uma página são substituídas por apenas uma dezena de linhas escritas com sintaxe jQuery. Intrincados e às vezes confusos códigos JavaScript destinados a selecionar um determinado elemento HTML componente da árvore do documento são substituídos por uma linha de código. Você mesmo, ao longo da leitura desse livro irá constatar como jQuery consegue a proeza de criar extraordinários efeitos com uma simplicidade impressionante, colocando o desenvolvimento de scripts ao seu alcance e dispor imediatos, sem exigir profundos conhecimentos de programação.

1.1.1.1 Histórico

No dia 22 de agosto de 2005 John Resig, cuja foto é mostrada na figura 1.1, um desenvolvedor americano profundo conhecedor de JavaScript atuando na Corporação Mozilla e autor do livro ***Pro JavaScript Techniques*** escreveu em seu blog um artigo relatando sua frustração com a maneira verbosa de se escrever JavaScript para obter os resultados pretendidos.



Figura 1.1 – John Resig o criador da jQuery.

Naquele artigo publicou alguns exemplos no quais propunha o uso de seletores CSS com o objetivo de simplificar e dar maior versatilidade ao código. Escreveu, então, que aquela ainda não era uma forma definitiva do que tinha em mente, mas iria aperfeiçoar e testar suas propostas. O nome ainda não existia mais ali foi lançada a idéia que resultaria em jQuery.

Aproximadamente cinco meses após a publicação do artigo em seu blog John Resig apresenta publicamente os resultados de seus estudos no BarCampNYC - Wrap up realizado no dia 14 de janeiro de 2006 em uma palestra sob o título “jQuery a nova onda para JavaScript”.

O ano de 2006 marcou ainda a criação do primeiro plug-in, o lançamento de uma versão não obstrutiva de LightBox usando a biblioteca jQuery, o lançamento das versões 1.0, 1.0.1, 1.0.2, 1.0.3 e 1.0.4, o lançamento da versão XML da biblioteca e o primeiro teste público de criação com jQuery.

Em 2007 foram lançadas as versões 1.1, 1.1.1, 1.1.2, 1.1.3a, 1.1.3, 1.1.3.1, 1.1.4, 1.2 e 1.2.1, no dia 11 do mês de março aconteceu o primeiro encontro jQuery.

Em 2008 até a data em que escrevi esse livro foram lançadas as versões 1.2.2, 1.2.3, 1.2.4, 1.2.5 e 1.2.6.

1.1.2 Para que serve jQuery?

jQuery se destina a adicionar interatividade e dinamismo às páginas web, incrementando de forma progressiva e não obstrutiva a usabilidade, a acessibilidade e o design, enriquecendo a experiência do usuário.

Use jQuery em sua página para:

- adicionar efeitos visuais e animações;
- acessar e manipular o DOM;
- buscar informações no servidor sem necessidade de recarregar a página;
- prover interatividade;
- alterar conteúdos;
- modificar apresentação e estilização;
- simplificar tarefas específicas de JavaScript;
- realizar outras tarefas relacionadas às descritas.

1.1.3 jQuery em conformidade com os Padrões Web

jQuery foi criada com a preocupação de ser uma biblioteca em conformidade com os Padrões Web, ou seja, compatível com qualquer sistema operacional, navegador e com suporte total para as CSS 3. Sim, é isso mesmo, você pode usar todos os poderosos seletores previstos nas CSS 3.

Evidentemente isso não significa que todo código escrito com uso de jQuery resulta em um documento válido segundo a ótica dos Padrões Web. A biblioteca foi criada e está em acordo com as diretrizes do W3C, mas cabe a você, desenvolvedor, escrever seus scripts em conformidade.

Se você pretende escrever em conformidade com os Padrões Web adote como regra básica de desenvolvimento de scripts a filosofia de que jQuery se destina a adicionar interatividade e dinamismo ***incrementando de forma progressiva e não obstrutiva*** a usabilidade, a acessibilidade e o design com o propósito de enriquecer a experiência do usuário.

Permita-me estabelecer a seguinte comparação:

Por não dispor de verba suficiente Fulano compra um carro, modelo o mais simples da linha, e alguns meses depois, tendo sobrado uma verba, resolve investir no carro equipando-o com alguns acessórios como, por exemplo, insulfilm, som, alarme, protetor solar e alguns outros mais, mudando o aspecto inicial do carro e fazendo-o parecer um modelo intermediário da linha. Passado algum tempo mais e tendo economizado uma boa quantia Fulano resolve melhorar o carro instalando calotas de aço escovado, GPS, TV digital, frigobar, MP3 e alguns outros acessórios mais sofisticados conseguindo um visual de carro top de linha.

Fulano é o desenvolvedor, o carro mais simples é a página web sem scripts, o carro top de linha é a página web incrementada com jQuery, as duas etapas de colocação de acessórios representam o incremento progressivo e o fato de Fulano (e ninguém em sã consciência) não ter mandado retirar o motor do carro para colocar um jarro de flores representa o incremento não obstrutivo.

O carro top de linha do Fulano continuará sendo um carro com todas as suas funcionalidades, cumprindo rigorosamente tudo para o qual foi projetado quando saiu da fábrica mesmo que dele sejam retirados todos os acessórios. Com scripts em geral, e jQuery no nosso caso, acontece rigorosamente o mesmo. Seu documento estará em conformidade com os Padrões Web se ele continuar usável e funcional caso os scripts parem de funcionar. Um belíssimo menu de navegação, com efeitos ultra sofisticados não valerá nada se não for acessível sem o script que o faz funcionar. Ele pode tornar-se esteticamente horroroso, mas deve cumprir sua finalidade quando não sustentado pelo script.

Algumas técnicas para preservar a acessibilidade aos conteúdos incrementados com jQuery são básicas e as abordaremos dando destaque a elas quando oportuno. Outras situações de preservação de acessibilidade por estarem inseridas no contexto de um desenvolvimento particular não têm solução em uma técnica pré-estabelecida e dependem da criatividade e capacidade do desenvolvedor em resolver situações específicas.

Sempre que for caso desenvolveremos os exemplos deste livro de forma não obstrutiva, contudo como dito anteriormente, quando esta questão depender de um contexto particular passaremos ao largo desta questão entendendo que sua avaliação e solução depende de cada caso.

1.1.4 Características da biblioteca jQuery

jQuery é uma biblioteca JavaScript que possui as seguintes características:

- uso de seletores CSS para localizar elementos componentes da estrutura de marcação HTML da página;
- arquitetura compatível com instalação de plug-ins e extensões em geral;
- indiferença às inconsistências de renderização entre navegadores;
- capaz de interação implícita isto é, não há necessidade de construção de loops para localização de elementos no documento;
- admite programação encadeada, ou seja, cada método retorna um objeto.
- é extensível, pois admite criação e inserção de novas funcionalidades na biblioteca existente.

Não se preocupe se algumas das características descritas soem sem sentido para você. Com o prosseguimento da leitura os conceitos ficarão claros e entendidos. O fato é que tais características possibilitaram a criação de uma biblioteca bastante compacta e ao mesmo tempo poderosa o bastante para oferecer funcionalidades que tornam o processo de desenvolvimento igualmente compacto e extremamente simples.

Por ser distribuída como software livre jQuery tem o apoio e o envolvimento de uma considerável comunidade. Desenvolvedores do mundo todo têm contribuído em larga escala com novas idéias, scripts, plug-ins, extensões e toda sorte de implementações, com a finalidade de incrementar não só a biblioteca como as técnicas de desenvolvimento jQuery.

1.1.5 Como instalar jQuery

A biblioteca jQuery nada mais é do que um arquivo JavaScript (arquivo tipo texto puro gravado com a extensão .js, como por exemplo: `meu_arquivo.js`) que deverá ser referenciado na página web onde serão aplicados efeitos. Ou, dito de uma forma menos técnica: a página web onde serão aplicados os efeitos deverá “chamar” a biblioteca. E é somente isso. Você não precisa instalar nada. Basta fazer o download gratuito do arquivo e chamá-lo na(s) página(s).

Uma página ou documento “chama” um arquivo de script fazendo uso do elemento `script` e seus atributos `type` e `src` colocado na seção `head` do documento.

A versão mais recente da biblioteca está no arquivo `jquery-1.2.6.js` (adiante veremos como e onde fazer o download do arquivo). Então, um documento que use a biblioteca deverá ter marcado em sua seção `head` o seguinte elemento `script` entre outros elementos próprios de cada caso.

```
...
<head>
...
<script type="text/javascript" src="/caminho/jquery-1.2.6.js"></script> <!-- esta
    linha chama a biblioteca jQuery -->
</head>
...
```

O valor do atributo `src` indica o caminho (diretório) no qual se encontra gravado o arquivo da biblioteca.

A partir daqui é imperativo que você tenha gravado no seu HD a última versão da biblioteca para poder acompanhar os exercícios desse livro e fazer funcionar seus experimentos com jQuery, que, tenho certeza, irão fasciná-lo.

Entre no site oficial <http://jquery.com> mostrado na figura 1.1 e faça o download da biblioteca jQuery.

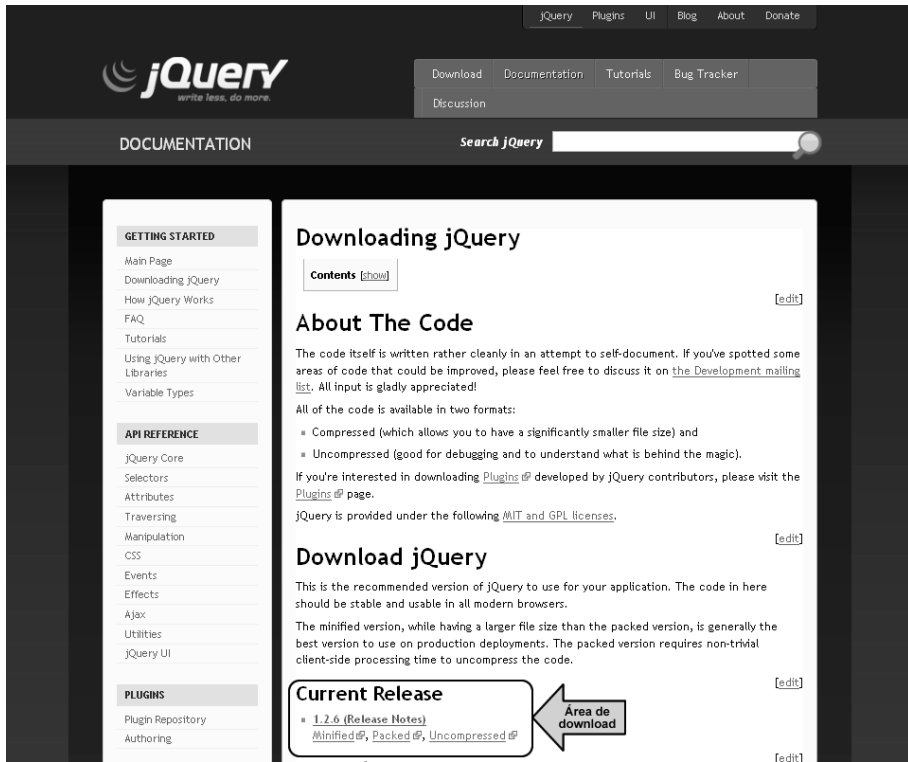


Figura 1.2 – Site oficial da biblioteca jQuery.

Observe na área de download em destaque na figura 1.1 que existem diferentes tipos de apresentação da biblioteca para download e um link para um documento hospedado no próprio site, contendo um relatório das modificações introduzidas na atual versão.

A apresentação denominada “Uncompressed” – `jquery-1.2.6.js` – relacionada na segunda posição da lista apresentada na figura 1.1 anterior destina-se ao uso em testes, estudos e desenvolvimento da biblioteca. Trata-se de um arquivo no qual o texto do script foi escrito com espaço entre cada linha de código, amplamente comentado o que facilita a leitura e compreensão do arquivo. Esta é a apresentação recomendada para você baixar e fazer uso nos estudos e acompanhamento dos experimentos desenvolvidos neste livro. O tamanho deste arquivo é de 97kB.

As duas outras apresentações são para uso em produção. A apresentação denominada “Packed” – `jquery-1.2.6.min.js` – difere da anterior por terem sido removidos todos os comentários e espaçamentos entre linhas e declarações, tornando o código difícil de ler para humanos, pois se transforma em uma seqüência corrida de código, sem quebras de linha. Esta é a apresentação recomendada para ser usada no desenvolvimento de sites. A vantagem sobre a versão anterior é que se trata do

mesmo arquivo com tamanho reduzido para 30kB.

Finalmente, a apresentação denominada “Minified and Gzipped” – `jquery-1.2.6.pack.js` – é uma compressão da biblioteca resultando em um arquivo com tamanho igual a 16kB. O uso dessa apresentação depende da capacidade de descompressão de arquivos JavaScript e está fora do escopo deste livro relatar como isso funciona. Use esta apresentação se você sabe como funciona compressão/descompressão de arquivos JavaScript.

A partir daqui, para testar os exemplos do livro, consideraremos que você baixou e gravou no seu HD a apresentação “Uncompressed” – `jquery-1.2.6.js` – da biblioteca para ser chamada pelas páginas de estudos conforme explicado anteriormente.

1.1.6 jQuery na prática

Para que um script consiga imprimir dinamismo, alterar comportamentos ou apresentação, no todo ou em partes de uma página web ele precisa de um método de acesso à árvore do documento com a finalidade de encontrar o elemento HTML no qual será vinculado o comportamento.

Para exemplificar vamos supor que em uma página web existe um botão que ao ser clicado muda a cor de um cabeçalho de verde para vermelha.

- Solução com JavaScript inline:

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
</style>
</head>
<body>
  <h1 id="cor">Cabeçalho muda de cor</h1>
  <button type="button" onclick = "document.getElementById('cor').style.
  color='#FF0000';">
    Vermelha
  </button>
...
```

[arquivo-1.1.6a.html]

A técnica usada nessa solução consiste em inserir script dentro de marcação HTML (inline) e é uma prática ultrapassada que infelizmente ainda é largamente empregada por muitos desenvolvedores. Essa solução contraria um princípio fundamental dos Padrões Web que preconiza separação total entre

estrutura de marcação com HTML, apresentação com CSS e comportamento com scripts. Cada código no seu arquivo correspondente e separado. Evite usar essa solução.

- Solução com função JavaScript:

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
</style>
<script type="text/javascript">
  function mudaCor() {
    document.getElementById('cor').style.color = '#FF0000';
  }
</script>
</head>
<body>
  <h1 id="cor">Cabeçalho muda de cor</h1>
  <button type="button" onclick = "mudaCor();">
    Vermelha
  </button>
...

```

[arquivo-1.1.6b.html]

A técnica usada nessa solução é uma melhoria da solução anterior e consiste em definir-se uma função dentro da seção `head` do documento que pode ser chamada por vários botões dentro de um mesmo documento permitindo ampliar o uso do script para além de um botão. Uma variante desta solução consiste em colocar a função em um arquivo externo e linkar o arquivo ao documento. Isso permite usar a função em vários botões dentro de vários documentos. Essa solução continua misturando estrutura com comportamento e tal como a anterior deve ser evitada.

- Solução com JavaScript fora da marcação:

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
</style>
<script type="text/javascript">
  window.onload = function() {
    document.getElementById('btn-vermelha').onclick = mudaCor;
  };

```

```

function mudaCor() {
    document.getElementById('cor').style.color = '#FF0000';
};
</script>
</head>
<body>
    <h1 id="cor">Cabeçalho muda de cor</h1>
    <button type="button" >Vermelha</button>
...

```

[arquivo-1.1.6c.html]

Agora sim! A marcação HTML está isenta de código JavaScript pois incorporamos nosso script na seção head do documento. Resta agora colocar o script em um arquivo externo separado e linkar para a(s) página(s). Essa é uma boa solução sob o ponto de vista da separação do comportamento, marcação, estilização.

- Solução com jQuery:

```

...
<head>
...
<script type="text/javascript" src="../jquery-1.2.6.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $('#btn-vermelha').click(function() {
            $('#cor').css('color', '#FF0000');
        });
    });
</script>
</head>
<body>
    <h1 id="cor">Cabeçalho muda de cor</h1>
    <button type="button" >Vermelha</button>
...

```

[arquivo-1.1.6d.html]

Ao contrário das soluções tradicionais com JavaScript, como vimos anteriormente, o uso da biblioteca jQuery não permite misturar script com marcação HTML. Você é obrigado a inserir seu script incorporado na seção head do documento ou escrevê-lo em um arquivo separado e linkar para os documentos onde serão utilizados. Como regra geral adote as mesmas diretrizes que regem a vinculação de folhas de estilo, isto é, se o seu script serve a mais de uma página adote a solução de linkar, se não, adote a solução de incorporar na seção head do documento.

Os quatro exemplos aqui apresentados tiveram por objetivo único mostrar na prática um efeito bem simples criado de três maneiras diferentes com JavaScript e à maneira jQuery. Caso você não saiba nada de JavaScript não haverá grandes prejuízos para sua aprendizagem, simplesmente consulte com atenção os códigos e vá em frente. Mas, tanto você quanto aqueles familiarizados com JavaScript não deixem de abrir cada uma das páginas contendo os scripts, clicar no botão lá existente e olhar no código fonte da página. Os arquivos das páginas estão no site do livro disponíveis para consulta on-line e também para download.

1.1.6.1 *Evento* window.onload

JavaScript é uma técnica de programação que funciona percorrendo e buscando seus alvos (elementos da marcação) na árvore do documento, ou no DOM. Dito de outra forma: um script só consegue executar sua ação se todo o documento já tiver sido carregado. Os elementos da marcação de uma página só existem depois que a página é carregada, ou, mais precisamente, vão existindo à medida que a página vai sendo carregada e na seqüência em que aparecem na marcação a partir da declaração do DOCTYPE até a tag de fechamento do elemento `html`.

Na prática isso significa que se você inserir na marcação de uma página um script que se refira a um elemento `h1`, por exemplo, em local acima daquele no qual foi escrito o elemento `h1`, o seu script não vai funcionar, porque o script será carregado na página antes do carregamento do elemento `h1`.

Observe, a seguir, a transcrição do terceiro exemplo mostrado no item anterior no qual todo JavaScript foi retirado da marcação e passado para a seção `head` do documento.

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
</style>
<script type="text/javascript">
  window.onload = function() {
    document.getElementById('btn-vermelha').onclick = mudaCor;
  };
  function mudaCor() {
    document.getElementById('cor').style.color = '#FF0000';
  };
</script>
</head>
<body>
  <h1 id="cor">Cabeçalho muda de cor</h1>
```

```
<button type="button" >Vermelha</button>
...
```

Vamos alterar ligeiramente o script anterior reescrevendo-o conforme mostrado a seguir.

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
</style>
<script type="text/javascript">
  document.getElementById('btn-vermelha').onclick = mudaCor;
  function mudaCor() {
    document.getElementById('cor').style.color = '#FF0000';
  }
</script>
</head>
<body>
<h1 id="cor">Cabeçalho muda de cor</h1>
<button type="button" >Vermelha</button>
...
```

[arquivo-1.1.6.1a.html]

Observe que o que fizemos foi retirar do script a declaração que diz:

```
window.onload = function() {
...faça isso...
}
```

Mesmo para quem não conhece nada de JavaScript a declaração é auto-explicativa, pois `window.onload`, traduzindo para o português significa: depois que a janela (documento) for carregada faça isso. Faça isso é a função `mudaCor()`.

Agora, com a retirada que fizemos o script (Faça isso...) não espera a página ser carregada e em consequência não funciona e o botão não troca a cor do cabeçalho, conforme você pode constatar no arquivo existente no site do livro. Isso acontece pela razão explicada anteriormente o script foi escrito antes do elemento `h1` e não foi dito para ele esperar a página ser carregada.

Se tirarmos o script da seção `head` do documento e posicionarmos ele depois dos elementos envolvidos o script funciona normalmente. Façamos assim:

```
...
<head>
...
<style type="text/css" media="all">
  h1 { color:#090; }
```

```

</style>
</head>
<body>
  <h1 id="cor">Cabeçalho muda de cor</h1>
  <button type="button" >Vermelha</button>
<script type="text/javascript">
  document.getElementById('btn-vermelha').onclick = mudaCor;
  function mudaCor() {
    document.getElementById('cor').style.color = '#FF0000';
  }
</script>
...

```

[arquivo-1.1.6.1b.html]

E tudo volta a funcionar. Veja nos arquivos existentes no site do livro as páginas mostrando as duas situações descritas anteriormente.

Conclusão

Para possibilitar a retirada de seus scripts da marcação HTML JavaScript dispõe da declaração `window.onload` que atua como uma espécie de aviso para a função definida na declaração entre em ação (ou comece a “rodar”) somente após a página ter sido completamente carregada.

Existem outros métodos que cumprem a mesma finalidade e oferecem outras funcionalidades, como por exemplo, permitir declarar várias funções, mas não é do escopo desse livro aprofundar a linguagem JavaScript.

1.1.6.2 Método `ready()`

Tal como vimos para JavaScript, jQuery que é baseada nessa linguagem também precisa que seus scripts conheçam a árvore do documento para poder funcionar.

E a sintaxe jQuery para o equivalente ao `window.onload` e todas as suas variantes é mostrada a seguir.

```

$(document).ready(function() {
  ...faça isso...
});

```

Esta é conhecida como **sintaxe formal** para escrita do método `ready()`.

Que significa o seguinte: “quando o documento estiver pronto faça isso”. Faça isso é o script a executar.

Você pode omitir o parâmetro passado para a função construtora `$()` e escrever

com a seguinte sintaxe:

```
$.ready(function() {  
    ...faça isso...  
});
```

Convém ressaltar que o método jQuery `read()` oferece duas vantagens sobre seu equivalente JavaScript:

- O script está pronto para funcionar tão logo a árvore do documento tenha sido carregada. Não é preciso, como ocorre com JavaScript, que todos os componentes da página, tais como, imagens, folhas de estilo, animações e mídias em geral tenham sido carregadas. Basta que a estrutura de marcação da página esteja disponível e o script já pode funcionar.
- Não há variações funcionais para o método e pelo fato de jQuery não admitir mistura de script com marcação usamos a sintaxe mostrada para servir de container aos nossos scripts.

Existe outra sintaxe alternativa à sintaxe formal mostrada anteriormente para o método `ready()`, chamada de **sintaxe abreviada** que é a seguinte:

```
$ (function() {  
    ...faça isso...  
});
```

Nos códigos desenvolvidos nesse livro adotaremos a sintaxe formal, pois consideramos que apesar de ser mais extensa oferece melhor legibilidade, sendo em consequência mais fácil de ser visualizada.

1.1.7 Fundamentos jQuery

A finalidade do uso de jQuery é controlar comportamento de toda ou partes de uma página web. Sabemos que uma página web nada mais é do que marcação HTML. Então é lícito concluir que o princípio de funcionamento de jQuery fundamenta-se na sua capacidade de encontrar os elementos HTML que constituem a página.

1.1.7.1 Construtor jQuery

Inicialmente queremos apresentá-lo ao encarregado de encontrar elementos HTML em um documento:

```
$(  
    ...  
);
```

Tecnicamente trata-se do que chamamos em linguagem de programação de **construtor**. O construtor `$()` ou ainda a função construtora `$()` estará presente em

todos os scripts que você escrever, portanto decore sua sintaxe, um sinal de cifrão seguido de parêntesis, (o que convenhamos não é tão difícil assim) desde já.

Outras bibliotecas JavaScript também usam a função `$()`. Isso pode causar conflitos e mau funcionamento de scripts quando se usa mais de uma biblioteca no mesmo documento. Nesses casos existem métodos para evitar conflitos e um deles é usar a sintaxe alternativa que é mostrada a seguir.

```
jQuery()
```

Nos códigos desenvolvidos nesse livro adotaremos a sintaxe: `$()`

Simplicidade é a filosofia que orienta o desenvolvimento de jQuery desde sua criação. Observe os códigos a seguir. Você, mesmo não conhecendo nada de jQuery, seria capaz de concluir a finalidade de cada uma das linhas do código a seguir?

```
$('#h1');  
$('#p');  
$('#span');  
$('#table');
```

Bem simples não é mesmo? Estamos instruindo nosso construtor a encontrar **todos** os elementos `h1`, `p`, `span` e `table` respectivamente.

jQuery ao contrário de JavaScript não requer loops para construir arrays quando busca elementos no DOM. O construtor `$()` armazena automaticamente em um objeto array tudo que encontra.

Encontrar **todos** os elementos de um determinado tipo não me parece muito útil. Seria bem melhor se eu pudesse encontrar uma ocorrência específica de um elemento.

Por exemplo: quero encontrar o terceiro parágrafo do documento. Bem, neste caso uma solução seria marcar o terceiro parágrafo com o atributo `id` para identificar sua ocorrência.

```
<p id="xpto">Texto do terceiro parágrafo</p>
```

e escrever o construtor assim:

```
$('#xpto')
```

Você conhece CSS? Não? Aconselho com muita ênfase a começar a aprender hoje mesmo, sob pena de ficar ultrapassado nas técnicas de desenvolvimento web. jQuery não é exceção à regra e faz amplo emprego de seletores CSS. Acredite, adota os poderosos seletores CSS 3 para localizar elementos no DOM. E não se preocupe, o uso de seletores CSS em jQuery nada tem a ver com suporte pelos navegadores. Use

e abuse desses seletores que seus scripts funcionarão em qualquer navegador.

Observe o código a seguir.

```
$(‘body p:nth-child(3)’)
```

Aqui o construtor está usando um seletor CSS 3 que tem como alvo o terceiro filho parágrafo do elemento `body`. Simples e sem necessidade de atribuir um identificador ao terceiro parágrafo como fizemos anteriormente.

A verdade é que jQuery usa amplamente seletores CSS e assim sendo é pré-requisito para bem desenvolver jQuery o conhecimento profundo desses seletores. No apêndice XX você encontra um guia de consulta aos seletores CSS 3 que irá ajudá-lo a acompanhar os exemplos desse livro.

1.1.7.2 Encadeamento jQuery

Um conceito importante no desenvolvimento com jQuery é o encadeamento. Observe a linha de código a seguir.

```
$(‘h1’).css(‘color’, ‘#FF0000’).fadeOut().addClass(‘xpto’)
```

Não se preocupe se o código parecer confuso ou ininteligível, logo você estará entendendo. Vamos ler o que está escrito.

“Construtor, encontre todos os elementos `h1` no documento e aplique a regra CSS que muda a cor do texto para vermelha (`#FF0000`), a seguir aplique um efeito de esmaecimento (`fadeOut`) no texto dos `h1` e adicione a classe denominada `xpto` nos elementos `h1`.”

Denomina-se encadeamento a característica de se poder encadear diversos métodos em uma declaração. Isso é possível porque jQuery foi criado de modo a que cada método retorne um objeto pronto para receber outro método que por sua vez retornará outro objeto e assim por diante permitindo ao desenvolvedor construir uma cadeia infinita de objetos e métodos. Em JavaScript tradicional não existe o encadeamento o que obriga o uso de múltiplas declarações para se conseguir um efeito que em jQuery se consegue com uma linha de código apenas.

1.1.7.3 Funções utilitárias

Servir como inspetor e selecionador de componentes do DOM, conforme vimos anteriormente, não é a única atribuição da função `$()`. jQuery prevê um conjunto de funções chamadas utilitárias que usa o sinal `$` como um identificador tal qual qualquer outro identificador previsto em JavaScript. A sintaxe para as funções utilitárias é mostrada no exemplo a seguir.

```
$.browser;
```

ou com a opção de uso do identificador jQuery

```
jQuery.browser
```

O exemplo mostra uma função para identificar o tipo de navegador do usuário que é equivalente à função `navigator.userAgent` do JavaScript.

É muito pouco provável que você use uma função utilitária no desenvolvimento de scripts para funcionar em uma página web. Elas têm sua utilidade no desenvolvimento e extensão da biblioteca jQuery, como por exemplo, na criação de plug-ins.

1.1.7.4 Conflitos com outras bibliotecas

Sem dúvida a invenção de bibliotecas revigorou o uso de JavaScript no desenvolvimento de páginas web, pois além de tornar o processo de criação bem mais simples, forneceu mecanismos que possibilitam criar códigos não obstrutivos e em consequência sem prejuízos para usabilidade e acessibilidade da página. jQuery não detém exclusividade no campo das bibliotecas JavaScript, pelo contrário, muitas bibliotecas surgiram nos últimos tempos entre elas: Prototype, MochiKit, MooTools, Yahoo User Interface (YUI), DOMAssistant.

O identificador `$` também não é exclusividade de jQuery e outras bibliotecas fazem uso dele. Se um documento usa mais de uma biblioteca é provável que ocorram conflitos com diferentes bibliotecas tentando interpretar um mesmo identificador e estabelecendo-se uma tremenda confusão.

O sinal `$` é um pseudônimo, no jargão técnico diz-se “alias”, para o identificador da biblioteca. O nosso identificador é jQuery, assim quando usamos essa biblioteca `$` é o pseudônimo de jQuery e é as duas sintaxes mostradas a seguir se equivalem.

```
$(
```

ou

```
jQuery()
```

Usar `jQuery()` elimina o risco de conflitos, mas obriga o desenvolvedor a abrir mão de escrever com a forma abreviada do pseudônimo.

Felizmente, para evitar conflitos com outras bibliotecas, sem abrir mão de uma forma abreviada existe a função `jQuery.noConflict()` que permite entre outras funcionalidades criar um pseudônimo personalizado para desenvolvimento. Trataremos desta função em [C2 S2.1]..