JQuery

A Biblioteca do Programador JavaScript

2ª Edição
Revisada e ampliada

Maurício Samy Silva

Novatec

CAPÍTULO 1

O que é jQuery?

Neste capítulo, será apresentada a biblioteca jQuery, relatando-se suas origens, finalidades e destinação. Será feito um breve relato histórico de sua evolução, examinando as motivações que resultaram em sua criação, e serão descritas algumas de suas possibilidades, dando uma ideia do seu potencial e mostrando a força e poder da programação JavaScript com o uso da biblioteca 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) e pelo GPL (GNU General Public License). Isto, resumidamente, significa que você pode usar a biblioteca gratuitamente tanto para desenvolver 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 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?".

Sem dúvida, ele estava em um momento de rara inspiração quando assim escreveu, pois soube resumir muito bem jQuery. É uma maneira simples e fácil de escrever JavaScript colocada ao alcance não só de programadores experientes, mas também de designers e desenvolvedores com pouco conhecimento de programação.

E o que torna o estudo e entendimento dos conceitos básicos 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 para ser usada com base nessa 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 simples efeito em um objeto são substituídas por apenas algumas, 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 um simples método jQuery. Você mesmo, ao longo da leitura deste livro, irá constatar como jQuery consegue a proeza de criar extraordinários efeitos com uma simplicidade impressionante, colocando o desenvolvimento de scripts a seu alcance e dispor imediatos, sem exigir profundos conhecimentos de programação.

1.1.1.1 Histórico

Em 22 de agosto de 2005, John Resig, cuja foto é mostrada na figura 1.1, um desenvolvedor americano profundo conhecedor de JavaScript, que atua na Corporação Mozilla e é 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.

Nesse 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 essa, ainda, não era a forma definitiva do que tinha em mente, mas iria aperfeiçoar e testar suas propostas. O nome ainda não existia, mas nessa ocasião foi lançada a ideia que traria como resultado a biblioteca jQuery.

Aproximadamente cinco meses após a publicação do artigo em seu blog, John Resig apresentou publicamente os resultados de seus estudos em uma palestra intitulada "jQuery, a nova onda para JavaScript", proferida no BarCampNYC – Wrap Up, no dia 14 de janeiro de 2006.

O ano de 2006 marcou a criação do primeiro plug-in para a biblioteca, o lançamento de uma versão não obstrutiva de LightBox usando a biblioteca jQuery. Nesse ano, ocorreram, ainda, o lançamento das versões 1.0, 1.0.1, 1.0.2, 1.0.3 e 1.04, da versão XML da biblioteca e o primeiro conteste público de criação com jQuery.

Em 2007, lançaram-se 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 de março ocorreu o primeiro encontro jQuery.

Em 2008, lançaram-se as versões 1.2.2, 1.2.3, 1.2.4, 1.2.5 e 1.2.6.

Em 2009, lançaram-se as versões 1.3, 1.3.1, 1.3.2

Em 2010, até a data em que escrevi este livro, foram lançadas as versões 1.4, 1.4.1 e 1.4.2.

1.1.2 Para que serve jQuery?

jQuery destina-se a adicionar interatividade e dinamismo às páginas web, proporcionando ao desenvolvedor funcionalidades necessárias à criação de scripts que visem a incrementar, 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 (fora do escopo deste livro);
- 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 e navegador, além de oferecer suporte total para as CSS 3. Sim, é isso mesmo, você pode usar todos os poderosos seletores previstos nas CSS 3 sem se preocupar se este ou aquele navegador suporta o seletor para as CSS. Esclarecendo: a sintaxe do seletor segue a sintaxe prevista nas CSS 3, mas quem usa os seletores é a biblioteca, com a finalidade de selecionar elementos no DOM e não estilizar.

Evidentemente, isso não significa que todo código escrito com uso de jQuery resulta em um documento válido segundo os Padrões Web. A biblioteca foi criada e está de 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.

Leia a analogia a seguir.

Por não dispor de verba suficiente, Fulano compra um carro, o modelo mais simples da linha, e alguns meses depois, tendo sobrado uma verba, resolve investir no carro equipando-o com alguns acessórios, como insulfilm, som, alarme, protetor solar e alguns outros itens, mudando o aspecto inicial do carro e fazendo-o parecer um modelo intermediário da linha. Passado algum tempo e tendo economizado uma boa quantia, Fulano resolve melhorar o carro instalando calotas de aço escovado, GPS, TV digital, frigobar, MP3 e 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 de Fulano continuará sendo um carro com todas as suas funcionalidades, cumprindo rigorosamente todas para as quais foi projetado quando saiu da fábrica, mesmo que dele sejam retirados todos os acessórios. Com scripts em geral, e jQuery no nosso caso, acontece a mesma situação. Seu documento estará em conformidade com os Padrões Web se continuar usável e funcional caso os scripts parem de funcionar. Um belíssimo menu de navegação, com efeitos ultrassofisticados, não valerá nada se não for acessível sem o script que o faz funcionar. 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 serão abordadas em momento oportuno. Outras situações de preservação de acessibilidade por estarem inseridas no contexto de um desenvolvimento particular não têm solução em uma técnica preestabelecida e dependem da criatividade e capacidade de o desenvolvedor resolver situações específicas.

Sempre que for o caso, os exemplos deste livro serão desenvolvidos de forma não obstrutiva, contudo, como dito anteriormente, quando a obstrução depender de um contexto particular, será entendido que sua avaliação e solução dependem de cada caso.

1.1.4 Características da biblioteca jQuery

jQuery é uma biblioteca JavaScript que possui as seguintes características:

- utiliza seletores CSS para localizar elementos componentes da estrutura de marcação HTML da página;
- possui arquitetura compatível com instalação de plug-ins e extensões em geral;
- é indiferente às inconsistências de renderização entre navegadores;
- é 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 compreensíveis. 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 ideias, scripts, plug-ins, extensões e toda sorte de implementações, com a finalidade de incrementar não só a biblioteca, mas também as técnicas de desenvolvimento jQuery.

1.1.5 Como instalar jQuery

A biblioteca jQuery nada mais é que um arquivo JavaScript (arquivo do tipo texto puro gravado com a extensão `.js`, por exemplo, `meu_arquivo.js`) que deverá ser linkado à página web onde serão aplicados efeitos, ou seja, a citada página web deverá "chamar" a biblioteca. É somente isso. Você não precisará instalar nada.

Existem duas opções para linkar, ou "chamar", a biblioteca jQuery em uma página web:

- **A partir de um arquivo local**: hospedando a biblioteca localmente no seu HD ou no servidor remoto no qual o site será hospedado e linkar a página para o endereço no qual foi hospedada.
- **A partir de um arquivo remoto público**: o Google e a Microsoft mantêm em seus sites as diferentes versões da biblioteca para acesso público. Assim, a página pode ser linkada para o endereço de hospedagem da biblioteca nesses servidores.

Vejamos a seguir cada uma dessas opções.

1.1.5.1 A partir de um arquivo local

Basta fazer o download gratuito do arquivo da biblioteca, conforme mostrado a seguir, e linká-la na(s) página(s).

Uma página ou documento linka para um arquivo de script fazendo uso do elemento `script` e seus atributos `type` e `src` colocados na seção `head` do documento.

A versão mais recente da biblioteca, na época em que escrevi este livro, emcontra-se no arquivo `jquery-1.4.2.js` e adiante você verá como e onde fazer o download dele. É muito provável que na ocasião em que você estiver lendo este tópico já exista uma versão mais recente e é esta que você deverá baixar para seus estudos e desenvolvimentos. 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-1.4.2.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.

Entre no site oficial `http://jquery.com`, vá para a página de download conforme mostrado na figura 1.2 e faça o download da última versão da biblioteca jQuery.

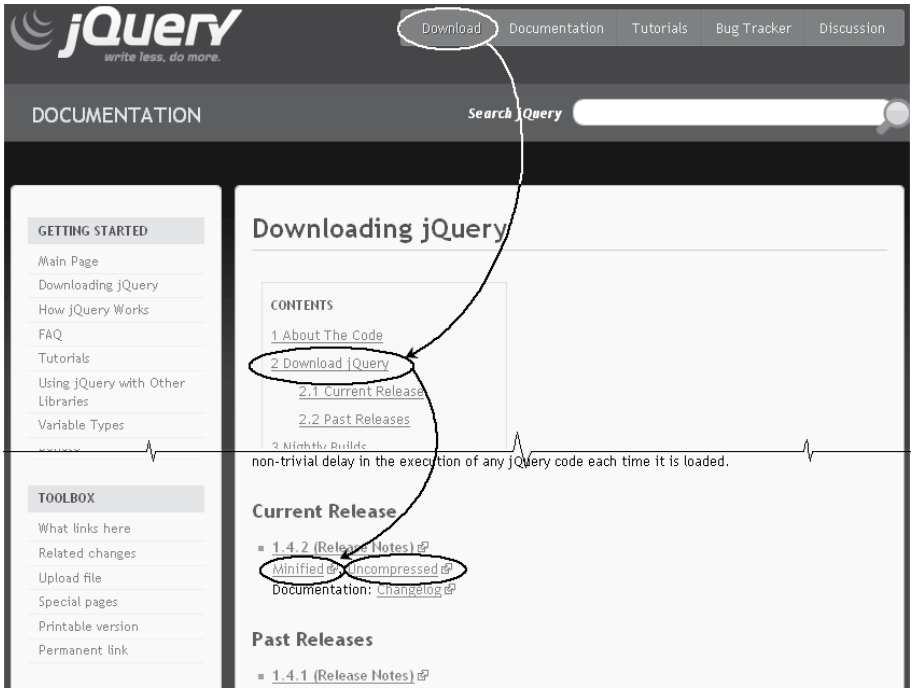

Figura 1.2 – Site oficial da biblioteca jQuery.

Observe, na área de download mostrada na figura 1.2, que existem dois 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-1.4.2.js` – 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, análise e entendimento do código contido no arquivo. Esta é a apresentação recomendada para você baixar e fazer uso nos estudos e acompanhamento dos experimentos desenvolvidos neste livro. O tamanho desse arquivo é de 155KB.

A apresentação denominada "Minified" – `jquery-1.4.2.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 e mais compacto, pois se transforma em uma sequência 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 24KB.

1.1.5.2 A partir de um arquivo remoto

Servidores do Google

O Google disponibiliza para uso público uma área denominada "API AJAX de bibliotecas do Google" (`http://code.google.com/intl/pt-BR/apis/ajaxlibs/`), que, segundo ele, é uma rede de distribuição de conteúdo e arquitetura de carregamento das bibliotecas JavaScript de código aberto mais populares.

Não só a biblioteca jQuery, mas também outras bibliotecas JavaScript de código aberto, como jQuery UI, Prototype, Mootools, YUI, Dojo, estão disponíveis nos seus servidores para serem linkadas para páginas web.

Há duas opções para se linkar a biblioteca: a primeira opção faz uso do elemento `script`, como se mostrou para a biblioteca instalada localmente, usando o endereço de hospedagem remota. Observe o código a seguir que exemplifica essa opção.

```
...
<head>
...
```

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"
   type="text/javascript"></script>
</head>
...
```

Nessa opção, você linka para uma das versões da biblioteca. No exemplo mostrado, está-se linkando para a versão 1.4.2 "minified" (compactada).

A segunda opção faz uso do método `google.load()`. Trata-se de um método desenvolvido pelo Google que permite carregar as versões da biblioteca de forma mais flexível do que a mostrada anteriormente. Essa forma de linkar foi denominada pelo Google de "`Criação de versão com o google.load()`" e, segundo ele, permite que o seu aplicativo especifique a versão desejada com a precisão necessária. Observe o código a seguir que exemplifica essa opção.

```
1. ...
2. <head>
3. ...
4. <script src="http://www.google.com/jsapi"  type="text/javascript"></script>
5. <script type="text/javascript">
 6. // Carrega jQuery
7.  google.load("jquery", "1.4.2");
8. </script>
9. </head>
10. ...
```

Código comentado:

Linha(s)	Descrição
Linha 4	Carrega uma biblioteca desenvolvida pelo Google. Destina-se a definir métodos e funções necessárias para carregar as bibliotecas da API do Google.
Linha 7	Carrega uma das bibliotecas da API. No exemplo mostrado, consta a biblioteca jQuery em sua versão 1.4.2.

Note que o método `google.load()` mostrado na linha 7 do script anterior admite dois parâmetros. O primeiro deles é o nome da biblioteca a carregar e o segundo, a sua versão. Por padrão, a biblioteca carregada está na sua opção compacta.

É possível especificar um terceiro parâmetro, opcional, caso se queira carregar a opção não compacta da biblioteca como mostrado a seguir.

```
1. ...
2. <head>
3. ...
4. <script src="http://www.google.com/jsapi"  type="text/javascript"></script>
5. <script type="text/javascript">
```

```
6.  // Carrega jQuery
7.  google.load("jquery", "1.4.2", {uncompressed: true});
8. </script>
9. </head>
10. ...
```

A versatilidade do método `google.load()` vai além do que se acaba de ver. Ele permite carregar dinamicamente a versão atual da biblioteca ou mesmo a versão mais atual de um determinado conjunto de versões. Veja alguns exemplos para esclarecer essa forma dinâmica de carregar a biblioteca.

Observe o exemplo a seguir.

```
1. ...
2. <head>
3. ...
4. <script src="http://www.google.com/jsapi"  type="text/javascript"></script>
5. <script type="text/javascript">
6.  // Carrega jQuery
7.  google.load("jquery", "1.3");
8. </script>
9. </head>
10. ...
```

Conforme se viu, em 2009 foram lançadas as versões 1.3, 1.3.1 e 1.3.2 da biblioteca. A linha 7 do código do exemplo mostrado insere a versão 1.3 da biblioteca em uma página web à época do lançamento da versão 1.3. Algum tempo depois, por ocasião do lançamento da versão 1.3.1, a mesma página carregará a versão 1.3.1. Da mesma forma, passado algum tempo e lançada a versão 1.3.2, a mesma página carregará a versão 1.3.2.

Depois das três versões da série 1.3.x, foi lançada a versão 1.4 e nossa página, marcada com a linha de código do exemplo mostrado, carregará para sempre a versão 1.3.2, ou seja, a última versão da série 1.3.x.

Observe outro exemplo.

```
1. ...
2. <head>
3. ...
4. <script src="http://www.google.com/jsapi"  type="text/javascript"></script>
5. <script type="text/javascript">
6.  // Carrega jQuery
7.  google.load("jquery", "1");
8. </script>
9. </head>
10. ...
```

Nesse caso, a cada lançamento da biblioteca em suas versões começando com o número 1, a biblioteca irá sendo atualizada na página até a versão imediatamente anterior ao lançamento de uma suposta versão 2. No dia em que escrevi este parágrafo, o código mostrado carregará a versão 1.4.2, que é a mais atual na presente data.

Podemos dizer que o método `google.load()` funciona com a especificação da versão, podendo esta ser feita à maneira como se escrevem caracteres curinga em buscas, ou seja, conforme a escrita da versão o carregamento da biblioteca se faz tanto para uma versão fixa como para uma versão sempre atualizada.

Na documentação do Google sobre a "API AJAX de bibliotecas do Google", em `http://code.google.com/intl/pt-BR/apis/ajaxlibs/documentation/index.html#jquery`, encontra-se uma lista das versões disponíveis, contudo tal lista nem sempre está atualizada. Por exemplo, à época que escrevi este livro, a lista parava na versão 1.3.2. A prática tem demonstrado que tão logo uma versão é lançada, ela é inserida e disponibilizada na API. Em caso de dúvida, construa uma página de teste e link a versão que você pretende usar, testando a seguir se ela funciona.

Servidores da Microsoft

A Microsoft disponibiliza a biblioteca jQuery, a partir da versão 1.3.2, para uso público, nas suas versões "uncompressed" e "mini". Veja a documentação no site da Microsoft em `http://www.asp.net/ajax/cdn`. Conforme consta daquela documentação, para linkar a biblioteca à uma página web usando os servidores da Microsoft, utiliza-se o código mostrado a seguir.

```
...
<head>
...
<script src="http://ajax.microsoft.com/ajax/jQuery/jquery-1.4.2.min.js"
   type="text/javascript"></script>
</head>
...
```

A partir daqui, para testar os exemplos do livro, supõe-se que você baixou e gravou em seu HD a última versão da biblioteca, para ser chamada pelas páginas de estudos conforme explicado anteriormente, ou, ainda, inseriu nas páginas um link para um dos endereços onde se encontra uma versão pública da biblioteca, quer usando linkagem direta, quer usando linkagem com o método `google.load()`, quer linkando para os servidores da Microsoft.

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, 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, imagine uma página web na qual existe um botão que, ao ser clicado, muda a cor de um cabeçalho, de verde para vermelha.

Nos exemplos constantes dos arquivos disponíveis para consulta ou download, você irá verificar o funcionamento dos scripts, interagindo com a página que contém o exemplo, clicando um botão ou agindo com o mouse. Para repetir o funcionamento do script, recarregue a página (em ambiente Windows, tecle F5).

▶ **Solução com JavaScript in-line**:

```
...
<head>
...
<style type="text/css" media="all">
   h1 {color:#090;} /* cor verde para o cabeçalho */
</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 (in-line), sendo uma prática ultrapassada e, infelizmente, ainda amplamente 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 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 dessa 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" id="btn-vermelha">Vermelha</button>
...
```

[arquivo-1.1.6c.html]

Agora, sim! A marcação HTML está isenta de código JavaScript, pois se incorporou o script à 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 e estilização.

▶ **Solução com jQuery**:

Caso você nada conheça de jQuery, não se intimide com o script a seguir. Limite-se a observar com atenção cada linha do código que seu conhecimento de JavaScript lhe dará uma noção bem próxima de seu funcionamento. Com o prosseguimento da leitura, você entenderá a finalidade de cada linha do script:

```
...
<head>
...
<style type="text/css" media="all">
    h1 {color:#090;}
</style>
<script type="text/javascript" src="../jquery-1.4.2.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" id="btn-vermelha">Vermelha</button>
...
```

[arquivo-1.1.6d.html]

Ao contrário das soluções tradicionais com JavaScript, como as mostradas anteriormente, o uso da biblioteca jQuery não permite misturar script com marcação HTML. Não é previsto nem existe uma sintaxe para jQuery in-line. 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 seu script serve a mais de uma página, adote a solução de linkar, senão, a solução de incorporar na seção `head` do documento. Mas lembre-se que mesmo para uso em uma só página, em scripts que demandem tempo de carregamento não desprezível, a solução de linkar é melhor, pois neste caso o script vai para o cache da máquina do usuário.

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. Ainda que seus conhecimentos básicos de JavaScript sejam limitados, não se intimide, 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, constatar seu funcionamento 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.

Toda página na qual se pretende fazer funcionar um script jQuery deverá estar linkada para o arquivo da biblioteca baixado do site jQuery.

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, ou seja, 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 sequência 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`, seu script não irá funcionar, porque será carregado na página antes do carregamento do elemento `h1`.

Observe, a seguir, a transcrição do terceiro exemplo mostrado no item 1.1.6 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" id="btn-vermelha">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" id="btn-vermelha">Vermelha</button>
...
```

 **[arquivo-1.1.6.1a.html]**

Observe que o que se fez foi retirar do script a declaração que diz:

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

Mesmo para quem pouco conhece de JavaScript, a declaração é autoexplicativa, pois `window.onload`, traduzindo para o português, significa: depois que o documento for carregado, faça isso. Faça isso é a função `mudaCor()`.

Agora, com a retirada que se fez, o script (`...faça isso...`) não esperará a página ser carregada e, consequentemente, não funcionará e o botão não trocará a cor do cabeçalho, conforme se pode constatar no arquivo existente no site do livro. Isso ocorre pela razão explicada anteriormente: o script foi escrito antes dos elementos `h1` e `button` com seus ids `cor` e `btn-vermelha` constantes do script, ou seja, quando o script é carregado, ainda não existem os ids de que precisa para funcionar.

Se tirar o script da seção `head` do documento e posicioná-lo depois dos elementos envolvidos, funcionará normalmente. Faça 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" id="btn-vermelha">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 voltará 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, a linguagem JavaScript dispõe da declaração `window.onload` que atua como uma espécie de aviso para que a função 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 permitir declarar várias funções, mas não é o escopo deste livro aprofundar a linguagem JavaScript.

1.1.6.2 Método `ready()`

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

Na sintaxe jQuery, o equivalente ao `window.onload` e todas as suas variantes é mostrado a seguir.

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

Essa notação 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 `document` passado para a função jQuery construtora `$()` e escrever com a seguinte sintaxe:

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

O método jQuery `ready()` oferece duas vantagens sobre seu equivalente JavaScript:

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

Existe uma 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 neste livro, será adotada a sintaxe formal, pois se considera que, apesar de ser mais extensa, oferece melhor legibilidade, sendo em consequência mais fácil de ser visualizada. Em seus desenvolvimentos reais, sinta-se à vontade para usar a sintaxe abreviada que, certamente, reduz o trabalho de digitação.

1.1.7 Fundamentos jQuery

A finalidade do uso de jQuery é controlar o comportamento de toda ou partes de uma página web. Sabe-se que uma página web nada mais é do que marcação HTML. Então, é lícito concluir que o princípio de funcionamento de jQuery fundamenta-se em sua capacidade de encontrar os elementos HTML que constituem a página e a estes anexar seus métodos.

1.1.7.1 Construtor jQuery

Inicialmente, veja o encarregado de encontrar elementos HTML em um documento:

```
$()
```

Tecnicamente, trata-se do que se denomina, em linguagem de programação, de construtor. O construtor `$()` ou, ainda, a função construtora `$()` estará presente em todos os scripts que você escrever, portanto decore desde já sua sintaxe: um sinal de cifrão seguido de parênteses (o que, convenha, não é tão complicado de decorar).

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 mostrada a seguir. Outros métodos para evitar conflitos serão apresentados posteriormente.

```
jQuery()
```

Nos códigos desenvolvidos neste livro, será adotada 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? Você está instruindo seu 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 documento. O construtor `$()` armazena automaticamente em um objeto array tudo que encontra.

Encontrar todos os elementos de um determinado tipo não parece muito útil. Seria bem melhor se você 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? Ainda não? É essencial 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. Como se disse anteriormente, adota os poderosos seletores CSS 3 para localizar elementos no documento. E não se preocupe, pois o uso de seletores CSS em jQuery em nada se relaciona 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 parágrafo descendente do elemento `body`. Trata-se de uma busca simples, sem necessidade de atribuir um identificador ao terceiro parágrafo como se fez 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 A, você encontra um guia de consulta aos seletores que irá ajudá-lo a acompanhar os exemplos deste livro.

1.1.7.2 Encadeamento jQuery

Um conceito importante da biblioteca jQuery é o encadeamento. Observe a linha de código a seguir:

```
$('div').children('p').fadeOut().addClass('xpto')
```

Não se preocupe se o código parecer confuso ou ininteligível, pois logo você estará entendendo-o. O código diz o seguinte:

> "Construtor, encontre todos os elementos parágrafos que sejam filhos dos elementos `div`, neles aplique um efeito de esmaecimento (fadeOut) e adicione a classe `xpto`".

Denomina-se encadeamento a característica de se poder encadear diversos métodos em uma declaração. Isso é possível porque se criou jQuery 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 encadeamento como o projetado para jQuery, 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 visto 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 equivale 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 de extensões para a biblioteca jQuery, como a 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) e 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 enorme confusão.

O sinal `$` é um pseudônimo e, no jargão técnico, diz-se "alias" para o identificador da biblioteca. O identificador utilizado foi jQuery, assim, ao usar essa biblioteca, `$` é o pseudônimo de `jQuery` e as duas sintaxes mostradas a seguir são equivalentes:

```
$()
```

e

```
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 prescindir de uma forma abreviada, existe a função `jQuery.noConflict()` que permite, entre outras funcionalidades, criar um pseudônimo personalizado para desenvolvimento. Tal função será abordada no capítulo 2, em [C2 S2.1].