Capítulo 1

Introdução a JavaScript

JavaScript é a linguagem de programação da Web. A ampla maioria dos sites modernos usa JavaScript e todos os navegadores modernos – em computadores de mesa, consoles de jogos, tablets e smartphones – incluem interpretadores JavaScript, tornando-a a linguagem de programação mais onipresente da história. JavaScript faz parte da tríade de tecnologias que todos os desenvolvedores Web devem conhecer: HTML, para especificar o conteúdo de páginas Web; CSS, para especificar a apresentação dessas páginas; e JavaScript, para especificar o comportamento delas. Este livro o ajudará a dominar a linguagem.

Se você já conhece outras linguagens de programação, talvez ajude saber que JavaScript é uma linguagem de alto nível, dinâmica, interpretada e não tipada, conveniente para estilos de programação orientados a objetos e funcionais. A sintaxe de JavaScript é derivada da linguagem Java, das funções de primeira classe de Scheme e da herança baseada em protótipos de Self. Mas não é preciso conhecer essas linguagens nem estar familiarizado com esses termos para utilizar este livro e aprender JavaScript.

Na verdade, o nome "JavaScript" é um pouco enganoso. A não ser pela semelhança sintática superficial, JavaScript é completamente diferente da linguagem de programação Java. E JavaScript já deixou para trás suas raízes como linguagem de script há muito tempo, tornando-se uma linguagem de uso geral robusta e eficiente. A versão mais recente da linguagem (veja o quadro) define novos recursos para desenvolvimento de software em grande escala.

JavaScript: nomes e versões

JavaScript foi criada na Netscape na fase inicial da Web e, tecnicamente, "JavaScript" é marca registrada, licenciada pela Sun Microsystems (agora Oracle), usada para descrever a implementação da linguagem pelo Netscape (agora Mozilla). A Netscape enviou a linguagem para a ECMA – European Computer Manufacturer's Association – para padronização e, devido a questões relacionadas à marca registrada, a versão padronizada manteve o nome estranho "ECMAScript". Pelos mesmos motivos ligados à marca registrada, a versão da Microsoft da linguagem é formalmente conhecida como "JScript". Na prática, quase todo mundo chama a linguagem de JavaScript. Este livro usa o nome "ECMAScript" apenas para se referir ao padrão da linguagem.

Na última década, todos os navegadores Web implementaram a versão 3 do padrão ECMAScript e não havia necessidade de se pensar em números de versão: o padrão da linguagem era estável e as implementações dos navegadores eram, na maioria, interoperáveis. Recentemente, uma importante nova versão da linguagem foi definida como ECMAScript versão 5 e, quando este livro estava sendo produzido, os navegadores estavam começando a implementá-la. Este livro aborda todos os novos recursos da ECMAScript 5, assim como todos os recursos consagrados da ECMAScript 3. Às vezes, você vai ver essas versões da linguagem abreviadas como ES3 e ES5, assim como às vezes vai ver o nome JavaScript abreviado como JS.

Quando falamos da linguagem em si, os únicos números de versão relevantes são ECMAScript versões 3 ou 5. (A versão 4 da ECMAScript esteve em desenvolvimento por anos, mas se mostrou ambiciosa demais e nunca foi lançada.) Contudo, às vezes você também vai ver um número de versão de JavaScript, como JavaScript 1.5 ou JavaScript 1.8. Esses são números da versão do Mozilla: a versão 1.5 é basicamente a ECMAScript 3 e as versões posteriores incluem extensões não padronizadas da linguagem (consulte o Capítulo 11). Por fim, também existem números de versão vinculados a interpretadores ou "engines" de JavaScript específicos. O Google chama seu interpretador JavaScript de V8, por exemplo, e quando este livro estava sendo produzido a versão corrente do mecanismo V8 era a 3.0.

Para ser útil, toda linguagem deve ter ou uma plataforma, ou biblioteca padrão, ou API de funções para fazer coisas como entrada e saída básicas. A linguagem JavaScript básica define uma API mínima para trabalhar com texto, arrays, datas e expressões regulares, mas não inclui funcionalidade alguma de entrada ou saída. Entrada e saída (assim como recursos mais sofisticados, como conexão em rede, armazenamento e gráficos) são responsabilidade do "ambiente hospedeiro" dentro do qual JavaScript está incorporada. Normalmente, esse ambiente hospedeiro é um navegador Web (apesar de que iremos ver duas utilizações de JavaScript sem um navegador Web, no Capítulo 12). A Parte I deste livro aborda a linguagem em si e sua API interna mínima. A Parte II explica como JavaScript é usada em navegadores Web e aborda as amplas APIs baseadas em navegador, geralmente conhecidas como "JavaScript do lado do cliente".

A Parte III é a seção de referência da API básica. Você pode ler sobre a API de manipulação de arrays de JavaScript procurando por "Array" nessa parte do livro, por exemplo. A Parte IV é a seção de re-

ferência de JavaScript do lado do cliente. Você pode procurar por "Canvas" nessa parte do livro para ler sobre a API gráfica definida pelo elemento `<canvas>` de HTML5, por exemplo.

Este livro abrange inicialmente os fundamentos de nível mais baixo e depois os amplia para abstrações mais avançadas e de nível mais alto. Os capítulos se destinam a serem lidos mais ou menos em ordem. Porém, aprender uma nova linguagem de programação nunca é um processo linear, e a descrição de uma linguagem também não é linear: cada recurso da linguagem se relaciona a outros recursos e este livro está repleto de referências cruzadas – às vezes para trás e às vezes à frente do material que você ainda não leu. Este capítulo faz um giro rápido pela linguagem básica e pela API do lado do cliente, apresentando recursos importantes que tornarão mais fácil entender o tratamento aprofundado dos capítulos seguintes.

Explorando JavaScript

Ao se aprender uma nova linguagem de programação é importante testar os exemplos do livro e, então, modificá-los e testá-los novamente para avaliar seu entendimento da linguagem. Para isso, você precisa de um interpretador de JavaScript. Felizmente, todo navegador Web contém um interpretador de JavaScript e, se você está lendo este livro, provavelmente já tem mais de um navegador Web instalado em seu computador.

Ainda neste capítulo, vamos ver que é possível incorporar código JavaScript entre marcas `<script>` em arquivos HTML e, quando o navegador carregar o arquivo, vai executar o código. Felizmente, contudo, não é preciso fazer isso sempre que você quiser testar trechos simples de código JavaScript. Impulsionados pela poderosa e inovadora extensão Firebug do Firefox (ilustrada na Figura 1-1 e disponível para download no endereço *http://getfirebug.com/*), todos os navegadores Web atuais contêm ferramentas para desenvolvedores Web que são indispensáveis para depuração, experimentação e aprendizado. Normalmente, essas ferramentas podem ser encontradas no menu Ferramentas do navegador, sob nomes como "Desenvolvedor Web" ou "Console da Web". (O Firefox contém um "Console da Web" interno, mas quando este livro estava sendo produzido, a extensão Firebug era melhor.) Frequentemente, é possível ativar um console com um toque de tecla, como F12 ou Ctrl-Shift-J. Muitas vezes, essas ferramentas de console aparecem como painéis na parte superior ou inferior da janela do navegador, mas alguns permitem abri-las como janelas separadas (conforme ilustrado na Figura 1-1), o que costuma ser bastante conveniente.

Um painel ou janela típica de "ferramentas para o desenvolvedor" contém várias guias que permitem inspecionar coisas como a estrutura de documentos HTML, estilos CSS, pedidos da rede, etc. Uma das guias é um "console JavaScript" que permite digitar linhas de código JavaScript e testá-las. Essa é uma maneira especialmente fácil de estudar JavaScript e recomendo que você a utilize ao ler este livro.

Existe uma API de console simples, implementada de forma portável pelos navegadores modernos. Você pode usar a função `console.log()` para exibir texto na console. Isso muitas vezes é surpreendentemente útil ao se fazer depuração, sendo que alguns exemplos deste livro (mesmo na seção de linguagem básica) utilizam `console.log()` para produzir saída simples. Uma maneira semelhante, porém mais invasiva, de exibir saída ou mensagens de depuração é passar uma string de texto para a função `alert()`, a qual as exibe em uma caixa de diálogo modal.

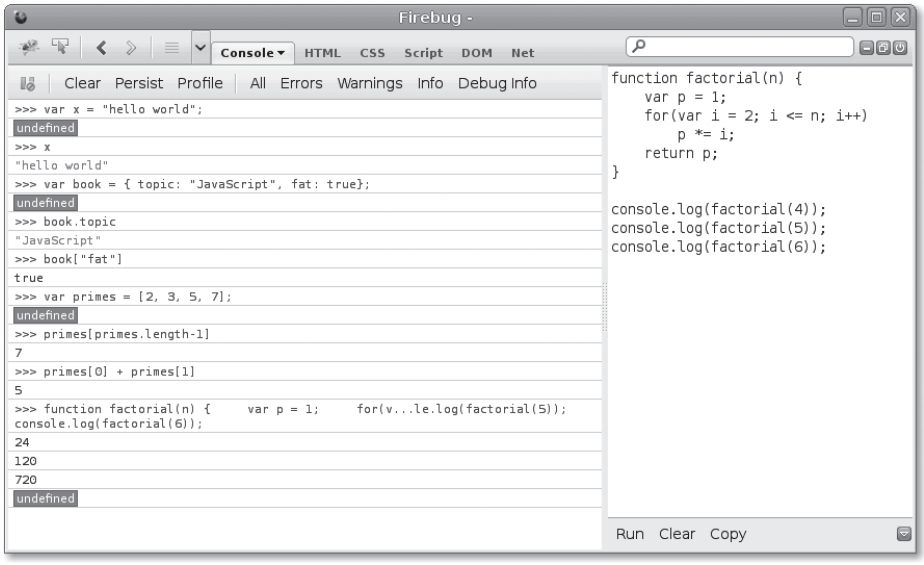

Figura 1-1 Console de depuração Firebug do Firefox.

1.1 JavaScript básica

Esta seção faz um giro pela linguagem JavaScript e também pela Parte I deste livro. Após este capítulo introdutório, entraremos no nível mais baixo de JavaScript: o Capítulo 2, *Estrutura léxica*, explica coisas como comentários em JavaScript, pontos e vírgulas e o conjunto de caracteres Unicode. O Capítulo 3, *Tipos, valores e variáveis*, começa a ficar mais interessante: ele explica as variáveis de JavaScript e os valores que podem ser atribuídos a elas. Aqui está um exemplo de código para ilustrar os destaques desses dois capítulos:

```
// Tudo que vem após barras normais duplas é um comentário em linguagem natural.
// Leia os comentários atentamente: eles explicam o código JavaScript.

// variável é um nome simbólico para um valor.
// As variáveis são declaradas com a palavra-chave var:
var x;                          // Declara uma variável chamada x.

// Valores podem ser atribuídos às variáveis com o sinal =
x = 0;                          // Agora a variável x tem o valor 0
x                               // => 0: Uma variável é avaliada com seu valor.

// JavaScript aceita vários tipos de valores
x = 1;                          // Números.
x = 0.01;                       // Apenas um tipo Number para inteiros e reais.
x = "hello world";              // Strings de texto entre aspas.
x = 'JavaScript';               // Apóstrofos também delimitam strings.
x = true;                       // Valores booleanos.
x = false;                      // O outro valor booleano.
```

```
x = null;                  // Null é um valor especial que significa "nenhum valor".
x = undefined;             // Undefined é como null.
```

Dois outros *tipos* muito importantes que programas em JavaScript podem manipular são objetos e arrays. Esses são os temas do Capítulo 6, *Objetos*, e do Capítulo 7, *Arrays*, mas são tão importantes que você vai vê-los muitas vezes antes de chegar a esses capítulos.

```
// O tipo de dados mais importante de JavaScript é o objeto.
// Um objeto é uma coleção de pares nome/valor ou uma string para mapa de valores.
var book = {               // Objetos são colocados entre chaves.
    topic: "JavaScript",   // A propriedade "topic" tem o valor "JavaScript".
    fat: true              // A propriedade "fat" tem o valor true.
};                         // A chave marca o fim do objeto.

// Acesse as propriedades de um objeto com . ou []:
book.topic                 // => "JavaScript"
book["fat"]                // => true: outro modo de acessar valores de propriedade.
book.author = "Flanagan";  // Crie novas propriedades por meio de atribuição.
book.contents = {};        // {} é um objeto vazio sem qualquer propriedade.

// JavaScript também aceita arrays (listas indexadas numericamente) de valores.
var primes = [2, 3, 5, 7]; // Um array de 4 valores, delimitados com [ e ].
primes[0]                  // => 2: o primeiro elemento (índice 0) do array.
primes.lenght              // => 4: quantidade de elementos no array.
primes[primes.lenght-1]    // => 7: o último elemento do array.
primes[4] = 9;             // Adiciona um novo elemento por meio de atribuição.
primes[4] = 11;            // Ou altera um elemento existente por meio de atribuição.
var empty = [];            // [] é um array vazio, sem qualquer elemento.
empty.lenght               // => 0

// Os arrays e objetos podem conter outros arrays e objetos:
var points = [             // Um array com 2 elementos.
    {x:0, y:0},            // Cada elemento é um objeto.
    {x:1, y:1}
];
var data = {               // Um objeto com 2 propriedades
  trial1: [[1,2], [3,4]],  // O valor de cada propriedade é um array.
  trial2: [[2,3], [4,5]],  // Os elementos dos arrays são arrays.
};
```

A sintaxe ilustrada anteriormente para listar elementos de array entre chaves ou para mapear nomes de propriedade de objeto em valores de propriedade entre colchetes é conhecida como *expressão inicializadora* e é apenas um dos assuntos do Capítulo 4, *Expressões e operadores*. Uma *expressão* é uma frase em JavaScript que pode ser *avaliada* para produzir um valor. O uso de . e [] para se referir ao valor de uma propriedade de objeto ou a um elemento de array é uma expressão, por exemplo. Talvez você tenha notado no código anterior que, quando uma expressão aparece sozinha em uma linha, o comentário que se segue começa com uma seta (=>) e o valor da expressão. Essa é uma convenção que você vai ver por todo o livro.

Uma das maneiras mais comuns de formar expressões em JavaScript é usar *operadores*, como segue:

```
// Os operadores atuam sobre os valores (operandos) para produzir um novo valor.
// Os operadores aritméticos são os mais comuns:
3 + 2                      // => 5: adição
```

```
3 - 2                          // => 1: subtração
3 * 2                          // => 6: multiplicação
3 / 2                          // => 1.5: divisão
points[1].x - points[0].x      // => 1: operandos mais complicados também funcionam
"3" + "2"                      // => "32": + soma números, ou concatena strings

// JavaScript define alguns operadores aritméticos de forma abreviada
var count = 0;                 // Define uma variável
count++;                       // Incrementa a variável
count--;                       // Decrementa a variável
count += 2;                    // Soma 2: o mesmo que count = count + 2;
count *= 3;                    // Multiplica por 3: o mesmo que count = count * 3;
count                          // => 6: nomes de variáveis também são expressões.

// Os operadores de igualdade e relacionais testam se dois valores são iguais,
// desiguais, menores que, maiores que, etc. São avaliados como verdadeiros ou falsos.
var x = 2, y = 3;              // Esses sinais = são atribuições e não testes
                               // de igualdade.
x == y                         // => falso: igualdade
x != y                         // => verdadeiro: desigualdade
x < y                          // => verdadeiro: menor que
x <= y                         // => verdadeiro: menor ou igual a
x > y                          // => falso: maior que
x >= y                         // => falso: maior ou igual a
"two" == "three"               // => falso: as duas strings são diferentes
"two" > "three"                // => verdadeiro: "tw" é alfabeticamente maior do que "th"
false == (x > y)               // => verdadeiro: falso é igual a falso

// Os operadores lógicos combinam ou invertem valores booleanos
(x == 2) && (y == 3)           // => verdadeiro: as duas comparações são verdadeiras. &&
                               // é E
(x > 3) || (y < 3)             // => falso: nenhuma das comparações é verdadeira. || é OU
!(x == y)                      // => verdadeiro: ! inverte um valor booleano
```

Se as frases em JavaScript são expressões, então as sentenças completas são *instruções*, as quais são o tema do Capítulo 5, *Instruções*. No código anterior, as linhas que terminam com ponto e vírgula são instruções. (No código a seguir, você vai ver instruções de várias linhas que não terminam com ponto e vírgula.) Na verdade há muita sobreposição entre instruções e expressões. Em linhas gerais, uma expressão é algo que calcula um valor, mas não *faz* nada: ela não altera o estado do programa de modo algum. As instruções, por outro lado, não têm um valor (ou não têm um valor com que nos preocupemos), mas alteram o estado. Você viu declarações de variável e instruções de atribuição anteriormente. A outra categoria abrangente de instrução são as *estruturas de controle*, como as condicionais e os laços. Exemplos aparecerão a seguir, após abordarmos as funções.

Uma *função* é um bloco de código JavaScript nomeado e parametrizado que você define uma vez e, então, pode chamar repetidamente. As funções não serão abordadas formalmente até o Capítulo 8, *Funções*, mas, assim como os objetos e arrays, você vai vê-las muitas vezes antes de chegar a esse capítulo. Aqui estão alguns exemplos simples:

```
// As funções são blocos de código JavaScript parametrizados que podemos chamar.
function plus1(x) {       // Define uma função chamada "plus1", com o parâmetro "x"
return x+1;               // Retorna um valor uma unidade maior do que o que foi passado
}                         // As funções são incluídas entre chaves
```

```
plus1(y)              // => 4: y é 3; portanto, essa chamada retorna 3+1

var square = function(x) {      // As funções são valores e podem ser atribuídas a
                      // variáveis
    return x*x;                 // Calcula o valor da função
};                    // Um ponto e vírgula marca o fim da atribuição.

square(plus(y))       // => 16: chama duas funções em uma única expressão
```

Quando combinamos funções com objetos, obtemos *métodos*:

```
// Quando funções recebem as propriedades de um objeto, as
// chamamos de "métodos". Todos os objetos de JavaScript têm métodos:
var a = [];                         // Cria um array vazio
a.push(1,2,3);                      // O método push() adiciona elementos em um array
a.reverse();                        // Outro método: inverte a ordem dos elementos

// Também podemos definir nossos próprios métodos. A palavra-chave "this" se refere ao
// objeto no qual o método é definido: neste caso, o array de pontos anterior.
points.dist = function() {          // Define um método para calcular a distância entre
                                    // pontos
    var p1 = this[0];               // Primeiro elemento do array que chamamos
    var p2 = this[1];               // Segundo elemento do objeto "this"
    var a = p2.x-p1.x;              // Diferença em coordenadas X
    var b = p2.y-p1.y;              // Diferença em coordenadas Y
    return Math.sqrt(a*a +          // O teorema de Pitágoras
        b*b);                       // Math.sqrt() calcula a raiz quadrada
};
points.dist()                       // => 1,414: distância entre nossos 2 pontos
```

Agora, conforme prometido, aqui estão algumas funções cujos corpos demonstram instruções de estruturas de controle JavaScript comuns:

```
// As instruções JavaScript incluem condicionais e laços que usam a sintaxe
// das linguagens C, C++, Java e outras.
function abs(x) {                   // Uma função para calcular o valor absoluto
    if (x >= 0) {                   // A instrução if...
        return x;                   // executa este código, se a comparação for
                                    // verdadeira.
    }                               // Este é o fim da cláusula if.
    else {                          // A cláusula opcional else executa seu código se
        return -x;                  // a comparação for falsa.
    }                               // Chaves são opcionais quando há 1 instrução por
                                    // cláusula.
}                                   // Observe as instruções return aninhadas dentro de
                                    // if/else.

function factorial(n) {             // Uma função para calcular fatoriais
    var product = 1;                // Começa com o produto de 1
    while(n > 1) {                  // Repete as instruções que estão em {}, enquanto a
                                    // expressão em () for verdadeira
        product *= n;               // Atalho para product = product * n;
        n--;                        // Atalho para n = n - 1
    }                               // Fim do laço
    return product;                 // Retorna o produto
}
factorial(4)                        // => 24: 1*4*3*2
```

```
function factorial2(n) {          // Outra versão, usando um laço diferente
    var i, product = 1;           // Começa com 1
    for(i=2; i <= n; i++)         // Incrementa i automaticamente, de 2 até n
        product *= i;             // Faz isso a cada vez. {} não é necessário para laços
                                  // de 1 linha
    return product;               // Retorna o fatorial
}
factorial2(5)                     // => 120: 1*2*3*4*5
```

JavaScript é uma linguagem orientada a objetos, mas é bastante diferente da maioria. O Capítulo 9, *Classes e módulos*, aborda com detalhes a programação orientada a objetos em JavaScript, com muitos exemplos, sendo um dos capítulos mais longos do livro. Aqui está um exemplo muito simples que demonstra como definir uma classe JavaScript para representar pontos geométricos bidimensionais. Os objetos que são instâncias dessa classe têm um único método chamado `r()` que calcula a distância do ponto a partir da origem:

```
// Define uma função construtora para inicializar um novo objeto Point
function Point(x,y) {        // Por convenção, as construtoras começam com letras
                             // maiúsculas
    this.x = x;              // A palavra-chave this é o novo objeto que está sendo
                             // inicializado
    this.y = y;              // Armazena os argumentos da função como propriedades do
                             // objeto
}                            // Nenhum return é necessário

// Usa uma função construtora com a palavra-chave "new" para criar instâncias
var p = new Point(1, 1);     // O ponto geométrico (1,1)

// Define métodos para objetos Point atribuindo-os ao objeto
// prototype associado à função construtora.
Point.prototype.r = function() {
    return Math.sqrt(        // Retorna a raiz quadrada de x² + y²
        this.x * this.x +    // Este é o objeto Point no qual o método...
        this.y * this.y      //...é chamado.
    );
};

// Agora o objeto Point b (e todos os futuros objetos Point) herda o método r()
p.r()                        // => 1,414...
```

O Capítulo 9 é o clímax da Parte I e os capítulos posteriores resumem outros pontos e encerram nossa exploração da linguagem básica. O Capítulo 10, *Comparação de padrões com expressões regulares*, explica a gramática das expressões regulares e demonstra como utilizá-las na comparação de padrões textuais. O Capítulo 11, *Subconjuntos e extensões de JavaScript*, aborda os subconjuntos e as extensões de JavaScript básica. Por fim, antes de mergulharmos em JavaScript do lado do cliente em navegadores Web, o Capítulo 12, *JavaScript do lado do servidor*, apresenta duas maneiras de usar JavaScript fora dos navegadores.

1.2 JavaScript do lado do cliente

JavaScript do lado do cliente não apresenta o problema de referência cruzada não linear no mesmo grau que a linguagem básica exibe, sendo que é possível aprender a usar JavaScript em navegadores Web em uma sequência bastante linear. Porém, você provavelmente está lendo este livro para

aprender JavaScript do lado do cliente e a Parte II está muito longe daqui; portanto, esta seção é um esboço rápido das técnicas básicas de programação no lado do cliente, seguida por um exemplo detalhado.

O Capítulo 13, *JavaScript em navegadores Web*, é o primeiro capítulo da Parte II e explica em detalhes como trabalhar com JavaScript em navegadores Web. O mais importante que você vai aprender nesse capítulo é que pode incorporar código JavaScript em arquivos HTML usando a marca <script>:

```
<html>
<head>
<script src="library.js"></script> <!-- inclui uma biblioteca de código JavaScript -->
</head>
<body>
<p>This is a paragraph of HTML</p>
<script>
// E este é um código JavaScript do lado do cliente
// literalmente incorporado no arquivo HTML
</script>
<p>Here is more HTML.</p>
</body>
</html>
```

O Capítulo 14, *O objeto Window*, explica técnicas de scripts no navegador Web e aborda algumas funções globais importantes de JavaScript do lado do cliente. Por exemplo:

```
<script>
function moveon() {
    // Exibe uma caixa de diálogo modal para fazer uma pergunta ao usuário
    var answer = confirm("Ready to move on?");
    // Se ele clicou no botão "OK", faz o navegador carregar uma nova página
    if (answer) window.location = "http://google.com";
}
// Executa a função definida acima por 1 minuto (60.000 milissegundos) a partir de agora.
setTimeout(moveon, 60000);
</script>
```

Note que o código do exemplo no lado do cliente mostrado nesta seção aparece em trechos mais longos do que os exemplos da linguagem básica anteriormente no capítulo. Esses exemplos não devem ser digitados em uma janela de console do Firebug (ou similar). Em vez disso, você pode incorporá-los em um arquivo HTML e testá-los, carregando-os em seu navegador Web. O código anterior, por exemplo, funciona como um arquivo HTML independente.

O Capítulo 15, *Escrevendo scripts de documentos*, trata do que é realmente JavaScript do lado do cliente, fazendo scripts de conteúdo de documentos HTML. Ele mostra como se seleciona elementos HTML específicos dentro de um documento, como se define os atributos HTML desses elementos, como se altera o conteúdo desses elementos e como se adiciona novos elementos no documento. A função a seguir demonstra diversas dessas técnicas básicas de pesquisa e modificação de documentos:

```
// Exibe uma mensagem em uma seção de saída de depuração especial do documento.
// Se o documento não contém esta seção, cria uma.
function debug(msg) {
    // Localiza a seção de depuração do documento, examinando os atributos de
    // identificação HTML
    var log = document.getElementById("debuglog");
```

```
    // Se não existe elemento algum com a identificação "debuglog", cria um.
    if (!log) {
        log = document.createElement("div"); // Cria um novo elemento <div>
        log.id = "debuglog";                 // Define o atributo de identificação HTML
                                             // nele
        log.innerHTML = "<h1>Debug Log</h1>";  // Define o conteúdo inicial
        document.body.appendChild(log);        // Adiciona-o no final do documento
    }

    // Agora, coloca a mensagem em seu próprio <pre> e a anexa no log
    var pre = document.createElement("pre");   // Cria uma marca <pre>
    var text = document.createTextNode(msg);   // Coloca a msg em um nó de texto
    pre.appendChild(text);                     // Adiciona o texto no <pre>
    log.appendChild(pre);                      // Adiciona <pre> no log
}
```

O Capítulo 15 mostra como JavaScript pode fazer scripts de elementos HTML que definem conteúdo da Web. O Capítulo 16, *Scripts de CSS*, mostra como você pode usar JavaScript com os estilos CSS que definem a apresentação desse conteúdo. Normalmente isso é feito com o atributo `style` ou `class` dos elementos HTML:

```
function hide(e, reflow) { // Oculta o elemento e faz script de seu estilo
    if (reflow) {                          // Se o 2º argumento é verdadeiro
        e.style.display = "none"           // oculta o elemento e utiliza seu espaço
    }
    else {                                 // Caso contrário
        e.style.visibility = "hidden";     // torna e invisível, mas deixa seu espaço
    }
}

function highlight(e) {          // Destaca e, definindo uma classe CSS
    // Basta definir ou anexar no atributo da classe HTML.
    // Isso presume que uma folha de estilos CSS já define a classe "hilite"
    if (!e.className) e.className = "hilite";
    else e.className += " hilite";
}
```

JavaScript nos permite fazer scripts do conteúdo HTML e da apresentação CSS de documentos em navegadores Web, mas também nos permite definir o comportamento desses documentos com *rotinas de tratamento de evento*. Uma rotina de tratamento de evento é uma função JavaScript que registramos no navegador e que este chama quando ocorre algum tipo de evento especificado. O evento de interesse pode ser um clique de mouse ou um pressionamento de tecla (ou, em um smartphone, pode ser um gesto de algum tipo, feito com dois dedos). Ou então, uma rotina de tratamento de evento pode ser ativada quando o navegador termina de carregar um documento, quando o usuário redimensiona a janela do navegador ou quando o usuário insere dados em um elemento de formulário HTML. O Capítulo 17, *Tratando eventos*, explica como se define e registra rotinas de tratamento de eventos e como o navegador as chama quando ocorrem eventos.

O modo mais simples de definir rotinas de tratamento de evento é com atributos HTML que começam com "on". A rotina de tratamento "onclick" é especialmente útil quando se está escrevendo programas de teste simples. Suponha que você tenha digitado as funções `debug()` e `hide()` anteriores e salvo em arquivos chamados *debug.js* e *hide.js*. Você poderia escrever um arquivo de teste simples em HTML usando elementos `<button>` com atributos da rotina de tratamento de evento `onclick`:

```
<script src="debug.js"></script>
<script src="hide.js"></script>
Hello
<button onclick="hide(this,true); debug('hide button 1');">Hide1</button>
<button onclick="hide(this); debug('hide button 2');">Hide2</button>
World
```

Aqui está um código JavaScript do lado do cliente que utiliza eventos. Ele registra uma rotina de tratamento de evento para o importante evento "load" e também demonstra uma maneira mais sofisticada de registrar funções de rotina de tratamento para eventos "click":

```
// O evento "load" ocorre quando um documento está totalmente carregado. Normalmente,
// precisamos esperar por esse evento antes de começarmos a executar nosso código
// JavaScript.
window.onload = function() {    // Executa esta função quando o documento for carregado
    // Localiza todas as marcas <img> no documento
    var images = document.getElementsByTagName("img");

    // Faz um laço por elas, adicionando uma rotina de tratamento para eventos "click" em
    // cada uma para que clicar na imagem a oculte.
    for(var i = 0; i < images.length; i++) {
        var image = images[i];
        if (image.addEventListener) // Outro modo de registrar uma rotina de
                                    // tratamento
            image.addEventListener("click", hide, false);
        else                        // Para compatibilidade com o IE8 e anteriores
            image.attachEvent("onclick", hide);
    }

    // Esta é a função de rotina para tratamento de evento registrada anteriormente
    function hide(event) { event.target.style.visibility = "hidden"; }
};
```

Os capítulos 15, 16 e 17 explicam como usar JavaScript para fazer scripts do conteúdo (HTML), da apresentação (CSS) e do comportamento (tratamento de eventos) de páginas Web. As APIs descritas nesses capítulos são um pouco complexas e, até recentemente, cheias de incompatibilidades com os navegadores. Por esses motivos, a maioria dos programadores JavaScript do lado do cliente optam por usar uma biblioteca ou estrutura do lado do cliente para simplificar as tarefas básicas de programação. A biblioteca mais popular é a jQuery, o tema do Capítulo 19, *A biblioteca jQuery*. A biblioteca jQuery define uma API engenhosa e fácil de usar para fazer scripts do conteúdo, da apresentação e do comportamento de documentos. Ela foi completamente testada e funciona em todos os principais navegadores, inclusive nos antigos, como o IE6.

É fácil identificar um código jQuery, pois ele utiliza frequentemente uma função chamada $(). Aqui está a função `debug()` utilizada anteriormente, reescrita com jQuery:

```
function debug(msg) {
    var log = $("#debuglog");              // Localiza o elemento para exibir a msg.
    if (log.length == 0) {                 // Se ele ainda não existe, cria-o...
        log = $("<div id='debuglog'><h1>Debug Log</h1></div>");
        log.appendTo(document.body);       // e o insere no final do corpo.
    }
    log.append($("<pre/>").text(msg));     // Coloca a msg em <pre> e anexa no log.
}
```

Os quatro capítulos da Parte II descritos até aqui foram todos sobre páginas Web. Outros quatro capítulos mudam o enfoque para *aplicativos* Web. Esses capítulos não falam sobre o uso de navegadores Web para exibir documentos com conteúdo, apresentação e comportamento em scripts. Em vez disso, falam sobre o uso de navegadores Web como plataformas de aplicativo e descrevem as APIs fornecidas pelos navegadores modernos para suportar aplicativos Web sofisticados do lado do cliente. O Capítulo 18, *Scripts HTTP*, explica como se faz requisições HTTP em scripts JavaScript – um tipo de API de ligação em rede. O Capítulo 20, *Armazenamento no lado do cliente*, descreve mecanismos para armazenar dados – e até aplicativos inteiros – no lado do cliente para usar em sessões de navegação futuras. O Capítulo 21, *Mídia e gráficos em scripts*, aborda uma API do lado do cliente para desenhar elementos gráficos em uma marca `<canvas>` da HTML. E, por fim, o Capítulo 22, *APIs de HTML5*, aborda várias APIs de aplicativo Web novas, especificadas pela HTML5 ou relacionadas a ela. Conexão em rede, armazenamento, gráficos: esses são serviços de sistema operacional que estão sendo fornecidos pelos navegadores Web, definindo um novo ambiente de aplicativo independente de plataforma. Se você está visando navegadores que aceitam essas novas APIs, esse é um bom momento para ser um programador JavaScript do lado do cliente. Não há exemplos de código desses quatro últimos capítulos aqui, mas o longo exemplo a seguir utiliza algumas dessas novas APIs.

1.2.1 Exemplo: uma calculadora de empréstimos em JavaScript

Este capítulo termina com um longo exemplo que reúne muitas dessas técnicas e mostra como são os programas JavaScript do lado do cliente reais (mais HTML e CSS). O Exemplo 1-1 lista o código do aplicativo de calculadora de pagamento de empréstimos simples ilustrada na Figura 1-2.

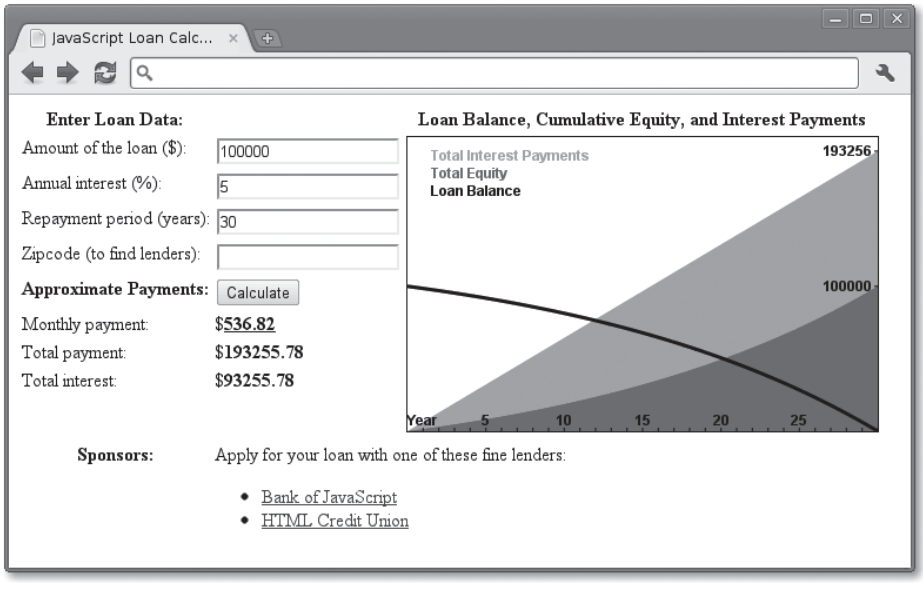

Figura 1-2 Um aplicativo Web de calculadora de empréstimos.

Vale a pena examinar o Exemplo 1-1 atentamente. Não espere compreender tudo, mas o código está bastante comentado e você será capaz de pelo menos entender a ideia geral de seu funcionamento. O

exemplo demonstra vários recursos da linguagem JavaScript básica e também importantes técnicas de JavaScript do lado do cliente:

- Como localizar elementos em um documento.
- Como obter entrada do usuário a partir de elementos de entrada de formulários.
- Como definir o conteúdo HTML de elementos do documento.
- Como armazenar dados no navegador.
- Como fazer requisições HTTP em scripts.
- Como desenhar gráficos com o elemento `<canvas>`.

Exemplo 1-1 Uma calculadora de empréstimos em JavaScript

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Loan Calculator</title>
<style> /* Esta é uma folha de estilos CSS: ela adiciona estilo na saída do programa */
.output { font-weight: bold; }               /* Valores calculados em negrito */
#payment { text-decoration: underline; }     /* Para elemento com id="payment" */
#graph { border: solid black 1px; }          /* O gráfico tem borda simples */
th, td { vertical-align: top; }              /* Não centraliza células da tabela */
</style>
</head>
<body>
<!--
  Esta é uma tabela HTML com elementos <input> que permitem ao usuário inserir dados e
  elementos <span> nos quais o programa pode exibir seus resultados. Esses elementos têm
  identificações como "interest" e "years". Essas identificações são usadas no código
  JavaScript que vem após a tabela. Note que alguns dos elementos de entrada definem
  rotinas de tratamento de evento "onchange" ou "onclick". Elas especificam strings de
  código JavaScript a ser executado quando o usuário insere dados ou dá um clique.
-->
<table>
  <tr><th>Enter Loan Data:</th>
    <td></td>
    <th>Loan Balance, Cumulative Equity, and Interest Payments</th></tr>
  <tr><td>Amount of the loan ($):</td>
      <td><input id="amount" onchange="calculate();"></td>
      <td rowspan=8>
        <canvas id="graph" width="400" height="250"></canvas></td></tr>
  <tr><td>Annual interest (%):</td>
      <td><input id="apr" onchange="calculate();"></td></tr>
  <tr><td>Repayment period (years):</td>
      <td><input id="years" onchange="calculate();"></td>
  <tr><td>Zipcode (to find lenders):</td>
      <td><input id="zipcode" onchange="calculate();"></td>
  <tr><th>Approximate Payments:</th>
      <td><button onclick="calculate();">Calculate</button></td></tr>
  <tr><td>Monthly payment:</td>
      <td>$<span class="output" id="payment"></span></td></tr>
  <tr><td>Total payment:</td>
      <td>$<span class="output" id="total"></span></td></tr>
```

```
  <tr><td>Total interest:</td>
      <td>$<span class="output" id="totalinterest"></span></td></tr>
  <tr><th>Sponsors:</th><td colspan=2>
     Apply for your loan with one of these fine lenders:
     <div id="lenders"></div></td></tr>
</table>

<!-- O restante deste exemplo é código JavaScript na marca <script> a seguir -->
<!-- Normalmente, este script ficaria na marca <head> do documento acima, mas -->
<!-- é mais fácil entendê-lo aqui, depois de ter visto seu contexto HTML. -->
<script>
"use strict"; // Usa o modo restrito da ECMAScript 5 nos navegadores que o suportam

/*
 * Este script define a função calculate() chamada pelas rotinas de tratamento de evento
 * no código HTML acima. A função lê valores de elementos <input>, calcula
 * as informações de pagamento de empréstimo, exibe o resultado em elementos <span>.
 * Também salva os dados do usuário, exibe links para financeiras e desenha um gráfico.
 */
function calculate() {
    // Pesquisa os elementos de entrada e saída no documento
    var amount = document.getElementById("amount");
    var apr = document.getElementById("apr");
    var years = document.getElementById("years");
    var zipcode = document.getElementById("zipcode");
    var payment = document.getElementById("payment");
    var total = document.getElementById("total");
    var totalinterest = document.getElementById("totalinterest");

    // Obtém a entrada do usuário através dos elementos de entrada. Presume que tudo isso
    // é válido.
    // Converte os juros de porcentagem para decimais e converte de taxa
    // anual para taxa mensal. Converte o período de pagamento em anos
    // para o número de pagamentos mensais.
    var principal = parseFloat(amount.value);
    var interest = parseFloat(apr.value) / 100 / 12;
    var payments = parseFloat(years.value) * 12;

    // Agora calcula o valor do pagamento mensal.
    var x = Math.pow(1 + interest, payments); // Math.pow() calcula potências
    var monthly = (principal*x*interest)/(x-1);

    // Se o resultado é um número finito, a entrada do usuário estava correta e
    // temos resultados significativos para exibir
    if (isFinite(monthly)) {
        // Preenche os campos de saída, arredondando para 2 casas decimais
        payment.innerHTML = monthly.toFixed(2);
        total.innerHTML = (monthly * payments).toFixed(2);
        totalinterest.innerHTML = ((monthly*payments)-principal).toFixed(2);

        // Salva a entrada do usuário para que possamos recuperá-la na próxima vez que
        // ele visitar
        save(amount.value, apr.value, years.value, zipcode.value);
```

```
        // Anúncio: localiza e exibe financeiras locais, mas ignora erros de rede
        try {  // Captura quaisquer erros que ocorram dentro destas chaves
            getLenders(amount.value, apr.value, years.value, zipcode.value);
        }
        catch(e) { /* E ignora esses erros */ }

        // Por fim, traça o gráfico do saldo devedor, dos juros e dos pagamentos do
           capital
        chart(principal, interest, monthly, payments);
    }
    else {
        // O resultado foi Not-a-Number ou infinito, o que significa que a entrada
        // estava incompleta ou era inválida. Apaga qualquer saída exibida anteriormente.
        payment.innerHTML = ""; // Apaga o conteúdo desses elementos
        total.innerHTML = ""
        totalinterest.innerHTML = "";
        chart();     // Sem argumentos, apaga o gráfico
    }
}

// Salva a entrada do usuário como propriedades do objeto localStorage. Essas
// propriedades ainda existirão quando o usuário visitar no futuro
// Esse recurso de armazenamento não vai funcionar em alguns navegadores (o Firefox, por
// exemplo), se você executar o exemplo a partir de um arquivo local:// URL. Contudo,
// funciona com HTTP.
function save(amount, apr, years, zipcode) {
    if (window.localStorage) {  // Só faz isso se o navegador suportar
        localStorage.loan_amount = amount;
        localStorage.loan_apr = apr;
        localStorage.loan_years = years;
        localStorage.loan_zipcode = zipcode;
    }
}

// Tenta restaurar os campos de entrada automaticamente quando o documento é carregado
// pela primeira vez.
window.onload = function() {
    // Se o navegador suporta localStorage e temos alguns dados armazenados
    if (window.localStorage && localStorage.loan_amount) {
        document.getElementById("amount").value = localStorage.loan_amount;
        document.getElementById("apr").value = localStorage.loan_apr;
        document.getElementById("years").value = localStorage.loan_years;
        document.getElementById("zipcode").value = localStorage.loan_zipcode;
    }
};

// Passa a entrada do usuário para um script no lado do servidor que (teoricamente) pode
// retornar
// uma lista de links para financeiras locais interessadas em fazer empréstimos. Este
// exemplo não contém uma implementação real desse serviço de busca de financeiras. Mas
// se o serviço existisse, essa função funcionaria com ele.
function getLenders(amount, apr, years, zipcode) {
    // Se o navegador não suporta o objeto XMLHttpRequest, não faz nada
    if (!window.XMLHttpRequest) return;
```

```
    // Localiza o elemento para exibir a lista de financeiras
    var ad = document.getElementById("lenders");
    if (!ad) return; // Encerra se não há ponto de saída
    // Codifica a entrada do usuário como parâmetros de consulta em um URL
    var url = "getLenders.php" +                 // Url do serviço mais
        "?amt=" + encodeURIComponent(amount) +           // dados do usuário na string
                                                         // de consulta
        "&apr=" + encodeURIComponent(apr) +
        "&yrs=" + encodeURIComponent(years) +
        "&zip=" + encodeURIComponent(zipcode);

    // Busca o conteúdo desse URL usando o objeto XMLHttpRequest
    var req = new XMLHttpRequest(); // Inicia um novo pedido
    req.open("GET", url);           // Um pedido GET da HTTP para o url
    req.send(null);                 // Envia o pedido sem corpo

    // Antes de retornar, registra uma função de rotina de tratamento de evento que será
    // chamada em um momento posterior, quando a resposta do servidor de HTTP chegar.
    // Esse tipo de programação assíncrona é muito comum em JavaScript do lado do
    // cliente.
    req.onreadystatechange = function() {
        if (req.readyState == 4 && req.status == 200) {
            // Se chegamos até aqui, obtivemos uma resposta HTTP válida e completa
            var response = req.responseText;    // Resposta HTTP como string
            var lenders = JSON.parse(response); // Analisa em um array JS

            // Converte o array de objetos lender em uma string HTML
            var list = "";
            for(var i = 0; i < lenders.length; i++) {
                list += "<li><a href='" + lenders[i].url + "'>" +
                lenders[i].name + "</a>";
            }

            // Exibe o código HTML no elemento acima.
            ad.innerHTML = "<ul>" + list + "</ul>";
        }
    }
}

// Faz o gráfico do saldo devedor mensal, dos juros e do capital em um elemento <canvas>
// da HTML.
// Se for chamado sem argumentos, basta apagar qualquer gráfico desenhado anteriormente.
function chart(principal, interest, monthly, payments) {
    var graph = document.getElementById("graph");        // Obtém a marca <canvas>
    graph.width = graph.width;      // Mágica para apagar e redefinir o elemento
                                // canvas
    // Se chamamos sem argumentos ou se esse navegador não suporta
    // elementos gráficos em um elemento <canvas>, basta retornar agora.
    if (arguments.length == 0 || !graph.getContext) return;

    // Obtém o objeto "contexto" de <canvas> que define a API de desenho
    var g = graph.getContext("2d"); // Todo desenho é feito com esse objeto
    var width = graph.width, height = graph.height;      // Obtém o tamanho da tela de
                                        // desenho
```

```
// Essas funções convertem números de pagamento e valores monetários em pixels
function paymentToX(n) { return n * width/payments; }
function amountToY(a) { return height-(a * height/(monthly*payments*1.05));}

// Os pagamentos são uma linha reta de (0,0) a (payments, monthly*payments)
g.moveTo(paymentToX(0), amountToY(0));      // Começa no canto inferior esquerdo
g.lineTo(paymentToX(payments),              // Desenha até o canto superior direito
         amountToY(monthly*payments));
g.lineTo(paymentToX(payments), amountToY(0));        // Para baixo, até o canto
                                 // inferior direito
g.closePath();                           // E volta ao início
g.fillStyle = "#f88";                    // Vermelho-claro
g.fill();                           // Preenche o triângulo
g.font = "bold 12px sans-serif";         // Define uma fonte
g.fillText("Total Interest Payments", 20,20);        // Desenha texto na legenda

// O capital acumulado não é linear e é mais complicado de representar no gráfico
var equity = 0;
g.beginPath();                           // Inicia uma nova figura
g.moveTo(paymentToX(0), amountToY(0));              // começando no canto inferior
                                                    // esquerdo
for(var p = 1; p <= payments; p++) {
    // Para cada pagamento, descobre quanto é o juro
    var thisMonthsInterest = (principal-equity)*interest;
    equity += (monthly - thisMonthsInterest);      // O resto vai para o capital
    g.lineTo(paymentToX(p),amountToY(equity));     // Linha até este ponto
}
g.lineTo(paymentToX(payments), amountToY(0));      // Linha de volta para o eixo X
g.closePath();                                     // E volta para o ponto inicial
g.fillStyle = "green";                             // Agora usa tinta verde
g.fill();                                          // E preenche a área sob a curva
g.fillText("Total Equity", 20,35);                 // Rotula em verde

// Faz laço novamente, como acima, mas representa o saldo devedor como uma linha
// preta grossa no gráfico
var bal = principal;
g.beginPath();
g.moveTo(paymentToX(0),amountToY(bal));
for(var p = 1; p <= payments; p++) {
    var thisMonthsInterest = bal*interest;
    bal -= (monthly - thisMonthsInterest);    // O resto vai para o capital
    g.lineTo(paymentToX(p),amountToY(bal));   // Desenha a linha até esse ponto
}
g.lineWidth = 3;                              // Usa uma linha grossa
g.stroke();                                   // Desenha a curva do saldo
g.fillStyle = "black";                        // Troca para texto preto
g.fillText("Loan Balance", 20,50);            // Entrada da legenda

// Agora faz marcações anuais e os números de ano no eixo X
g.textAlign="center";                                 // Centraliza o texto nas
                                                      // marcas
var y = amountToY(0);                                 // Coordenada Y do eixo X
for(var year=1; year*12 <= payments; year++) {        // Para cada ano
    var x = paymentToX(year*12);                      // Calcula a posição da marca
    g.fillRect(x-0.5,y-3,1,3);                        // Desenha a marca
    if (year == 1) g.fillText("Year", x, y-5);        // Rotula o eixo
```

```
            if (year % 5 == 0 && year*12 !== payments)         // Numera cada 5 anos
                g.fillText(String(year), x, y-5);
        }

        // Marca valores de pagamento ao longo da margem direita
        g.textAlign = "right";              // Alinha o texto à direita
        g.textBaseline = "middle";          // Centraliza verticalmente
        var ticks = [monthly*payments, principal];   // Os dois pontos que marcaremos
        var rightEdge = paymentToX(payments);        // Coordenada X do eixo Y
        for(var i = 0; i < ticks.length; i++) {      // Para cada um dos 2 pontos
            var y = amountToY(ticks[i]);    // Calcula a posição Y da marca
            g.fillRect(rightEdge-3, y-0.5, 3,1);     // Desenha a marcação
            g.fillText(String(ticks[i].toFixed(0)), // E a rotula.
                    rightEdge-5, y);
        }
    }
    </script>
    </body>
    </html>
```