



Secrets of the JavaScript Ninja

SEGUNDA EDIÇÃO

John Resig
Bear Bibeault
Josip Maras



TRIPULAÇÃO

Folha de dicas ES6

Literals de modelo embute expressões em strings: ``$ { ninja} ``.

Variáveis com escopo de bloco:

- Use as novas `let` palavra-chave para criar variáveis de nível de bloco: `let ninja = "Yoshi"`.
- Use as novas `const` palavra-chave para criar variáveis constantes de nível de bloco cujo valor não pode ser reatribuído a um valor completamente novo: `const ninja = "Yoshi"`.

Parâmetros de função:

- **Parâmetros de descanso** crie uma matriz de argumentos que não foram correspondidos aos parâmetros:

```
função multiMax (primeiro, ... restante) {/"..."/}multiMax(2, 3, 4, 5); // primeiro: 2;  
restantes: [3, 4, 5]
```

- **Parâmetros padrão** especifique os valores de parâmetro padrão que são usados se nenhum valor for fornecido durante a invocação:

```
função do (ninja, action = "skulk") { retornar ninja + "" + ação;} do ("Fuma"); // "Fuma skulk"
```

Operadores de spread expanda uma expressão onde vários itens são necessários: `[... itens, 3, 4, 5]`.

Funções de seta vamos criar funções com menos confusão sintática. Eles não têm seus próprios esta parâmetro. Em vez disso, eles o herdam do contexto em que foram criados:

```
valores const = [0, 3, 2, 5, 7, 4, 8, 1];  
valores.sort ((v1, v2) => v1 - v2); / * OR * / valores.sort ((v1, v2) => {return v1 - v2;}); value.forEach (value => console.log (value));
```

Geradores gerar sequências de valores por solicitação. Uma vez que um valor é gerado, o gerador suspende sua execução sem bloqueio. Usar produzem para gerar valores:

```
função * IdGenerator () {  
  deixe id = 0;  
  enquanto (verdadeiro) { rendimento ++ eu ia; }  
}
```

Promessas são marcadores de posição para o resultado de um cálculo. Uma promessa é uma garantia de que eventualmente saberemos o resultado de alguns cálculos. A promessa pode ser bem-sucedida ou falhar e, depois de cumprida, não haverá mais mudanças:

- Crie uma nova promessa com nova promessa `((resolver, rejeitar) => {})` ;.
- Ligar para resolver função para resolver explicitamente uma promessa. Ligar para rejeitar função para rejeitar explicitamente uma promessa. Uma promessa é rejeitada implicitamente se ocorrer um erro.
- O objeto de promessa tem um `então` método que retorna uma promessa e recebe dois retornos de chamada, um retorno de sucesso e um retorno de falha:

```
myPromise.then (val => console.log ("Sucesso"), err => console.log ("Erro"));
```

- Corrente em um pegar método para detectar falhas de promessa: `myPromise.catch (e => alert (e))` ;.

(continua na contracapa)

Elogios pela Primeira Edição

Finalmente, de um mestre, um livro que oferece o que um aspirante a desenvolvedor de JavaScript precisa para aprender a arte de criar um JavaScript eficaz para vários navegadores.

- Glenn Stokol, desenvolvedor sênior de currículo principal,
Oracle Corporation

Consistente com o lema da jQuery, "Escreva menos, faça mais".

- André Roberge, Université Saint-Anne

Técnicas interessantes e originais.

- Scott Sauyet, Four Winds Software

Leia este livro, e você não vai mais inserir cegamente um trecho de código e se maravilhar com a forma como ele funciona - você entenderá "por que" funciona.

- Joe Litton, desenvolvedor de software colaborativo, JoeLitton.net

Ajudará você a elevar seu JavaScript ao reino dos mestres.

- Christopher Haupt, greenstack.com

As coisas que os ninjas precisam saber.

- Chad Davis, autor de *Struts 2 em ação*

Leitura obrigatória para qualquer JavaScript Master.

- John J. Ryan III, Princigration LLC

Este livro é obrigatório para qualquer programador JS sério. Seu conhecimento do idioma se expandirá dramaticamente.

- S., leitor da Amazon

Segredos do JavaScript Ninja,
Segunda edição

JOHN RESIG
BEAR BİBEAULT
e JOSIP MARAS



MANN I NG
Shelter Island

Para informações online e pedidos deste e de outros livros da Manning, visite www.manning.com. O editor oferece descontos neste livro quando solicitado em quantidade. Para mais informações por favor entre em contato


Departamento de Vendas Especiais
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964 Email:
orders@manning.com

© 2016 por Manning Publications Co. Todos os direitos reservados.

Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação ou transmitida, em qualquer forma ou por meio eletrônico, mecânico, fotocópia ou outro, sem a permissão prévia por escrito do editor.

Muitas das designações usadas por fabricantes e vendedores para distinguir seus produtos são reivindicadas como marcas registradas. Onde essas designações aparecem no livro, e a Manning Publications estava ciente de uma reivindicação de marca registrada, as designações foram impressas em maiúsculas iniciais ou todas em maiúsculas.

☺ Reconhecendo a importância de preservar o que foi escrito, é política de Manning fazer com que os livros que publicamos sejam impressos em papel sem ácido, e fazemos o possível para esse fim. Reconhecendo também nossa responsabilidade de conservar os recursos de nosso planeta, os livros de Manning são impressos em papel que é pelo menos 15% reciclado e processado sem o uso de cloro elementar.

 Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

| | |
|------------------------------------|--------------------------------|
| Editor de desenvolvimento: | Dan Maharry |
| Editor de desenvolvimento técnico: | Gregor Zurowski |
| Editor de resenhas: | Ozren Harlovic |
| Editor de projeto: | Tiffany Taylor |
| Editor de cópia: | Sharon Wilkey |
| Revisor: | Alyson Brener |
| Revisores técnicos: | Mathias Bynens, Jon Borgman |
| Tipógrafo: | Gordan Salinovic |
| Designer de capa: | Marija Tudor |

ISBN 9781617292859

Impresso nos Estados Unidos da América

1 2 3 4 5 6 7 8 9 10 - EBM - 21 20 19 18 17 16

conteúdo

introdução do autor xi
agradecimentos xiii
sobre este livro xv
sobre a ilustração da capa xxi

P ARTE 1 W ARMANDO 1

1 *JavaScript está em toda parte 3*

1.1 Compreendendo a linguagem JavaScript 4

Como o JavaScript evoluirá? 6. Transpilers nos dão acesso ao JavaScript de amanhã, hoje 6

1.2 Compreendendo o navegador 7

1.3 Usando as melhores práticas atuais 8

Depurando 9. Teste 9. Análise de desempenho 10

1.4 Aumentando a capacidade de transferência 10

1.5 Resumo 11

2 *Construindo a página no tempo de execução 13*

2.1 A visão geral do ciclo de vida 14

2.2 A fase de construção da página 17

Analisando o HTML e construindo o DOM 18 . Executando o código JavaScript 20

2.3 Tratamento de eventos 23

Visão geral do tratamento de eventos 24 . Registrando manipuladores de eventos 25 Gerenciando eventos 27

2.4 Resumo 29

2.5 Exercícios 29

PARTE 2 U FUNÇÕES DE ENTENDIMENTO 31

3

Funções de primeira classe para o novato: definições e argumentos 33

3.1 O que há com a diferença funcional? 34

Funciona como objetos de primeira classe 35 . Funções de retorno de chamada 36

3.2 Diversão com funções como objetos 40

Armazenando funções 40 . Funções de auto-memorização 42

3.3 Definindo funções 44

Declarações de funções e expressões de funções 45 . Funções de seta 50

3.4 Argumentos e parâmetros de função 52

Parâmetros de descanso 54 . Parâmetros padrão 55

3.5 Resumo 58

3.6 Exercícios 59

4

Funções para o jornalista: função de compreensão

4.1 Usando parâmetros de função implícita 62

O parâmetro de argumentos 62 . O parâmetro this: introduzindo o contexto da função 67

4.2 Chamando funções 67

Invocação como uma função 68 . Invocação como método 69 Invocação como construtor 72 . Chamada com os métodos apply e call 77

4.3 Resolvendo o problema de contextos de função 83

Usando funções de seta para contornar contextos de função 83 Usando o método bind 86

4.4 Resumo 88

4.5 Exercícios 88

5 *Funções para o mestre: fechamentos e escopos 91*

5.1 Compreendendo os fechamentos 92

5.2 Colocando fechamentos para funcionar 95

Imitando variáveis privadas 95 . Usando fechamentos com retornos de chamada 96

5.3 Acompanhamento da execução do código com contextos de execução 99

5.4 Acompanhar identificadores com ambientes lexicais 103

Aninhamento de código 103 . Aninhamento de código e ambientes lexicais 104

5.5 Noções básicas sobre tipos de variáveis JavaScript 106

Mutabilidade variável 107 . Palavras-chave de definição de variável e ambientes lexicais 109 . Registrando identificadores em ambientes lexicais 113

5.6 Explorando como funcionam os fechamentos 117

Revisitando; imitando variáveis privadas com fechamentos 117 . Advertência de variáveis privadas 121 . Revisitando o exemplo de fechamentos e retornos de chamada 122

5,7 Resumo 124

5,8 Exercícios 124

6 *Funções para o futuro: geradores e promessas 126*

6.1 Tornando nosso código assíncrono elegante com geradores e promessas 127

6.2 Trabalhando com funções do gerador 129

Controlando o gerador através do objeto iterador 130 . Usando geradores 133 . Comunicando-se com um gerador 136 Explorando geradores sob o capô 139

6.3 Trabalho com promessas 146

Compreendendo os problemas com retornos de chamada simples 147 . Mergulhando nas promessas 149 . Rejeitando promessas 152 . Criando nossa primeira promessa do mundo real 154 . Encadeando promessas 155 Esperando por uma série de promessas 156

6.4 Combinando geradores e promessas 158

Olhando para o futuro - a função assíncrona 161

6,5 Resumo 162

6.6 Exercícios 163

PARTE 3 DIGGING EM OBJETOS E FORTIFICANDO

SEU CÓDIGO 165

7

Orientação do objeto com protótipos 167

7.1 Compreendendo os protótipos 168

7.2 Construção de objetos e protótipos 171

Propriedades da instância 173 . Efeitos colaterais da natureza dinâmica do JavaScript 176 . Digitação de objetos por meio de construtores 179

7.3 Atingindo herança 181

O problema de substituir a propriedade do construtor 184 . A instância do operador 187

7.4 Usando “classes” JavaScript no ES6 190

Usando a palavra-chave class 190 . Implementando herança 193

7.5 Resumo 195

7.6 Exercícios 196

8

Controlando o acesso a objetos 199

8.1 Controle de acesso às propriedades com getters e setters 200

Definindo getters e setters 202 . Usando getters e setters para validar valores de propriedade 207 . Usando getters e setters para definir propriedades computadas 208

8.2 Usando proxies para controlar o acesso 210

Usando proxies para registro 214 . Usando proxies para medir o desempenho 215 . Usando proxies para preencher automaticamente propriedades 217 Usando proxies para implementar índices de array negativos 218 Custos de desempenho de proxies 220

8.3 Resumo 221

8.4 Exercícios 222

9

Lidando com coleções 224

9.1 Arrays 225

Criação de matrizes 225 . Adicionando e removendo itens em qualquer extremidade de uma matriz 227 . Adicionar e remover itens em qualquer local da matriz 230 . Operações comuns em matrizes 232 . Reutilizando funções de array integradas 242

9.2 Mapas 244

Não use objetos como mapas 245 . Criando nosso primeiro mapa 247 Iterando sobre mapas 250

9.3 Conjuntos 251

Criando nosso primeiro conjunto 252 . União de conjuntos 253 . Interseção de conjuntos 255 . Diferença de conjuntos 255

9.4 Resumo 256

9.5 Exercícios 256

10

Wrangling expressões regulares 259

10.1 Por que as expressões regulares são 260

10.2 Um atualizador de expressão regular 261

Expressões regulares explicadas 261 . Termos e operadores 263

10.3 Compilando expressões regulares 267

10.4 Capturando segmentos correspondentes 269

Executando capturas simples 269 . Correspondência usando expressões globais 271 . Referenciando capturas 272 . Grupos de não captura 273

10.5 Substituindo usando funções 274

10.6 Resolvendo problemas comuns com expressões regulares 276

Novas linhas correspondentes 277 . Unicode 277 correspondente . Caracteres de escape correspondentes 278

10.7 Resumo 279

10.8 Exercícios 280

11

Técnicas de modularização de código 282

11.1 Modularizando código em JavaScript pré-ES6 283

Usando objetos, encerramentos e funções imediatas para especificar módulos 284
Modularizando aplicativos JavaScript com AMD e CommonJS 290

11.2 Módulos ES6 294

Funcionalidade de exportação e importação 294

11.3 Resumo 300

11.4 Exercícios 301

P ARTE 4 B RECONHECIMENTO DE ROWSER 303

12

Trabalhando no DOM 305

12.1 Injetando HTML no DOM 306

Convertendo HTML em DOM 307 . Inserindo elementos no documento 311

12.2 Usando atributos e propriedades DOM 313

12.3 Dores de cabeça de atributo de estilo 315

Onde estão meus estilos? 315. Nomenclatura de propriedades de estilo 318 Buscando estilos computados 319. Convertendo valores de pixel 322 Medindo alturas e larguras 323

12.4 Minimizando layout thrashing 327

12.5 Resumo 330

12.6 Exercícios 330

13

Sobrevivendo eventos 332

13.1 Mergulho no loop de eventos 333

Um exemplo com apenas macrotarefas 336. Um exemplo com macro e microtarefas 339

13.2 Temporizadores de domesticação: tempos limite e intervalos 344

Execução do cronômetro dentro do loop de evento 345. Lidando com processamento computacionalmente caro 350

13.3 Trabalhando com eventos 353

Propagando eventos através da DOM 354. Eventos personalizados 360

13.4 Resumo 364

13.5 Exercícios 364

14

Desenvolvimento de estratégias para vários navegadores 367

14.1 Considerações entre navegadores 368

14.2 As cinco principais questões de desenvolvimento 370

Bugs e diferenças do navegador 371. Correções de bug do navegador 371 Código externo e marcação 373. Regressões 376

14.3 Estratégias de implementação 378

Correções seguras para navegadores diferentes 378. Detecção de recursos e polyfills 379 Problemas não testáveis do navegador 381

14.4 Reduzindo suposições 383

14.5 Resumo 384

14.6 Exercícios 385

apêndice A Recursos ES6 adicionais 387

Apêndice B Armando com teste e depuração 392 Respostas de

apêndice C exercícios 411

índice 433

introdução do autor

É incrível pensar no quanto o mundo do JavaScript mudou desde que comecei a escrever *Segredos do JavaScript Ninja* em 2008. O mundo em que escrevemos JavaScript agora, embora ainda esteja amplamente centrado no navegador, é quase irreconhecível.

A popularidade do JavaScript como uma linguagem de plataforma cruzada cheia de recursos explodiu. Node.js é uma plataforma formidável contra a qual inúmeros aplicativos de produção são desenvolvidos. Os desenvolvedores estão, na verdade, escrevendo aplicativos em uma linguagem - JavaScript - que podem ser executados em um navegador, em um servidor e até mesmo em um aplicativo nativo em um dispositivo móvel.

É mais importante agora, do que nunca, que o conhecimento de um desenvolvedor da linguagem JavaScript esteja no auge. Ter uma compreensão fundamental da linguagem e das maneiras pelas quais ela pode ser melhor escrita permitirá que você crie aplicativos que podem funcionar em praticamente qualquer plataforma, o que é uma afirmação de que poucas outras linguagens podem se gabar legitimamente.

Ao contrário das eras anteriores no crescimento do JavaScript, não houve um crescimento igual nas incompatibilidades de plataforma. Antes, você ficava salivando só de pensar em usar os novos recursos mais básicos do navegador e, ainda assim, ficava bloqueado por navegadores desatualizados que tinham participação de mercado excessiva. Entramos em uma época harmoniosa em que a maioria dos usuários está em navegadores atualizados rapidamente que competem para ser a plataforma mais compatível com os padrões. Os fornecedores de navegadores até se esforçam para desenvolver recursos voltados especificamente para desenvolvedores, na esperança de tornar suas vidas mais fáceis.

As ferramentas que temos agora, fornecidas pelos navegadores e pela comunidade de código aberto, estão anos-luz à frente das práticas antigas. Temos uma infinidade de estruturas de teste para escolher, a capacidade de fazer testes de integração contínua, gerar relatórios de cobertura de código, teste de desempenho em dispositivos móveis reais em todo o mundo e até mesmo carregar navegadores virtuais automaticamente em qualquer plataforma para testar.

A primeira edição do livro se beneficiou tremendamente da visão de desenvolvimento fornecida por Bear Bibeault. Esta edição recebeu ajuda substancial de Josip Maras para explorar os conceitos por trás do ECMAScript 6 e 7, mergulhar nas melhores práticas de teste e entender as técnicas empregadas por estruturas JavaScript populares.

Tudo isso é uma longa maneira de dizer: a maneira como escrevemos JavaScript mudou substancialmente. Felizmente, este livro está aqui para ajudá-lo a se manter atualizado sobre as práticas recomendadas atuais. Não apenas isso, mas ajudará a melhorar a forma como você pensa sobre suas práticas de desenvolvimento como um todo para garantir que você estará pronto para escrever JavaScript no futuro.

J OHN R ESIG

agradecimentos

O número de pessoas envolvidas na escrita de um livro surpreenderia a maioria das pessoas. Foi necessário um esforço colaborativo por parte de muitos colaboradores com uma variedade de talentos para trazer o volume que você está segurando (ou e-book que está lendo na tela) em prática.

A equipe da Manning trabalhou incansavelmente conosco para garantir que este livro atingisse o nível de qualidade que esperávamos e agradecemos a eles por seus esforços. Sem eles, este livro não teria sido possível. Os “créditos finais” deste livro incluem não apenas nosso editor, Marjan Bace, e o editor Dan Maharry, mas também os seguintes colaboradores: Ozren Harlovic, Gregor Zurowski, Kevin Sullivan, Janet Vail, Tiffany Taylor, Sharon Wilkey, Alyson Brener e Gordan Salinovic.

Não se pode dizer o suficiente para agradecer aos nossos revisores que ajudaram a moldar a forma final do livro, desde a captura de erros de digitação simples até a correção de erros de terminologia e código, e ajuda na organização dos capítulos do livro. Cada passagem por um ciclo de revisão acabou melhorando muito o produto final. Por reservar um tempo para revisar o livro, gostaríamos de agradecer a Jacob Andresen, Tidjani Belmansour, Francesco Bianchi, Matthew Halverson, Becky Huett, Daniel Lamb, Michael Lund, Kariem Ali Elkoush, Elyse Kolker Gordon, Christopher Haupt, Mike Hatfield, Gerd Klevesaat, Alex Lucas, Arun Noronha, Adam Scheller, David Starkey e Gregor Zurowski.

Agradecimentos especiais a Mathias Bynens e Jon Borgman, que atuaram como revisores técnicos do livro. Além de verificar cada amostra de código de exemplo em vários ambientes, eles também ofereceram contribuições inestimáveis para a precisão técnica do texto, localizaram informações que estavam faltando originalmente e se mantiveram a par das rápidas mudanças no suporte a JavaScript e HTML5 nos navegadores .

John Resig

Gostaria de agradecer a meus pais por seu apoio e incentivo constantes ao longo dos anos. Eles me forneceram os recursos e ferramentas de que eu precisava para despertar meu interesse inicial em programação - e eles têm me incentivado desde então.

Bear Bibeault

O elenco de personagens que eu gostaria de agradecer por esta sétima tentativa tem uma longa lista de "suspeitos do costume", incluindo, mais uma vez, os membros e a equipe do coderanch.com (antigo JavaRanch). Sem meu envolvimento no CodeRanch, eu nunca teria tido a oportunidade de começar a escrever em primeiro lugar, então agradeço sinceramente a Paul Wheaton e Kathy Sierra por começarem tudo, bem como colegas de equipe que me encorajaram e apoiaram, incluindo (mas certamente não limitado a) Eric Pascarello, Ernest Friedman-Hill, Andrew Monkhouse, Jeanne Boyarsky, Bert Bates e Max Habibi.

Meu marido, Jay, e meus cachorros, Ursinho e Cozmo, recebem os agradecimentos calorosos de sempre por tolerar a presença sombria que compartilhava sua casa e raramente erguia os olhos do teclado, exceto para xingar Word, os navegadores, minha falta de dedos gordos de habilidades de digitação, ou qualquer outra coisa que atraiu minha ira enquanto eu estava trabalhando neste projeto.

Por fim, gostaria de agradecer aos meus co-autores, John Resig e Josip Maras, sem os quais este projeto não existiria.

Josip Maras

Os maiores agradecimentos vão para minha esposa, Josipa, por suportar todas as horas gastas para escrever este livro.

Também gostaria de agradecer a Maja Stula, Darko Stipanicev, Ivica Crnkovic, Jan Carlson e Bert Bates: todos eles pela orientação e conselhos úteis, e alguns deles por serem tolerantes em minhas atribuições de "trabalho diário" conforme os prazos dos livros se aproximavam .

Por fim, gostaria de agradecer ao resto da minha família - Jere, dois Marijas, Vitomir e Zdenka - por sempre estarem disponíveis para mim.

sobre este livro

JavaScript é importante. Nem sempre foi assim, mas agora é verdade. JavaScript se tornou uma das linguagens de programação mais importantes e mais amplamente usadas hoje.

Espera-se que os aplicativos da Web ofereçam aos usuários uma experiência de interface de usuário rica e, sem JavaScript, você pode muito bem mostrar fotos de gatinhos. Mais do que nunca, os desenvolvedores da web precisam ter uma compreensão sólida da linguagem que dá vida aos aplicativos da web.

E como suco de laranja e café da manhã, JavaScript não é mais apenas para navegadores. A linguagem há muito derrubou as paredes do navegador e está sendo usada no servidor graças ao Node.js, em dispositivos desktop e móveis por meio de plataformas como Apache Cordova, e até mesmo em dispositivos embarcados com Espruino e Tessel.

Embora este livro se concentre principalmente em JavaScript executado no navegador, os fundamentos da linguagem apresentada neste livro são aplicáveis em todo o quadro. Compreender verdadeiramente os conceitos e aprender várias dicas e truques farão de você um desenvolvedor JavaScript completo.

Com cada vez mais desenvolvedores usando JavaScript em um mundo cada vez mais JavaScript, é mais importante do que nunca entender seus fundamentos para que você possa se tornar um ninja especialista na linguagem.

Público

Se você não está familiarizado com JavaScript, este provavelmente não deve ser seu primeiro livro. Mesmo se for, não se preocupe muito; tentamos apresentar os conceitos fundamentais do JavaScript de uma forma que seja compreensível mesmo para iniciantes. Mas para ser honesto,

este livro provavelmente será mais adequado para pessoas que já conhecem um pouco de JavaScript e desejam aprofundar sua compreensão de JavaScript como uma linguagem, bem como do navegador como o ambiente no qual o código JavaScript é executado.

Roteiro

Este livro é organizado para levá-lo de um aprendiz a um ninja em quatro partes.

A Parte 1 apresenta o tópico e prepara o cenário para que você possa progredir facilmente pelo resto do livro:

- O Capítulo 1 apresenta a linguagem JavaScript e seus recursos mais importantes, enquanto sugere as melhores práticas atuais que devemos seguir ao desenvolver aplicativos, incluindo testes e análise de desempenho.
- Como nossa exploração de JavaScript é feita no contexto de navegadores, no capítulo 2 definiremos o cenário, introduzindo o ciclo de vida dos aplicativos da Web do lado do cliente. Isso nos ajudará a entender a função do JavaScript no processo de desenvolvimento de aplicativos da web.

A Parte 2 concentra-se em um dos pilares do JavaScript: funções. Estudaremos por que as funções são tão importantes em JavaScript, os diferentes tipos de funções, bem como os detalhes essenciais de definir e invocar funções. Colocaremos um foco especial em um novo tipo de função - funções geradoras - que são especialmente úteis ao lidar com código assíncrono:

- O Capítulo 3 inicia nossa incursão nos fundamentos da linguagem, começando, talvez para sua surpresa, com um exame completo do *função* conforme definido por JavaScript. Embora você possa ter esperado o *objeto* para ser o alvo de nosso primeiro foco, uma compreensão sólida da função e do JavaScript como uma linguagem funcional, começa nossa transformação de codificadores comuns de JavaScript para ninjas de JavaScript!
- Continuamos esse encadeamento funcional no capítulo 4, explorando o mecanismo exato de invocar funções, bem como os meandros dos parâmetros de função implícitos.
- Ainda não concluindo as funções, no capítulo 5 levamos nossa discussão para o próximo nível, estudando dois conceitos intimamente relacionados: *escopos* e *fechamentos*. Um conceito-chave na programação funcional, os encerramentos nos permitem exercer um controle refinado sobre o escopo dos objetos que declaramos e criamos em nossos programas. O controle desses escopos é o fator chave para escrever um código digno de um ninja. Mesmo que você pare de ler depois deste capítulo (mas esperamos que não pare), você será um desenvolvedor de JavaScript muito melhor do que quando começou.
- Concluimos nossa exploração de funções no capítulo 6, dando uma olhada em um tipo completamente novo de função (*funções do gerador*) e um novo tipo de objeto (*promessas*) que nos ajudam a lidar com valores assíncronos. Também mostraremos como combinar geradores e promessas para obter elegância ao lidar com código assíncrono.

A Parte 3 trata do segundo pilar do JavaScript: objetos. Exploraremos exaustivamente a orientação a objetos em JavaScript e estudaremos como proteger o acesso a objetos e como lidar com coleções e expressões regulares:

- Os objetos são finalmente tratados no capítulo 7, onde aprendemos exatamente como funciona o sabor ligeiramente estranho do JavaScript de orientação a objetos. Também apresentaremos uma nova adição ao JavaScript: classes, que, bem no fundo, podem não ser exatamente o que você espera.
- Continuaremos nossa exploração de objetos no capítulo 8, onde estudaremos diferentes técnicas para proteger o acesso a nossos objetos.
- No capítulo 9, colocaremos um foco especial em diferentes tipos de coleções que existem em JavaScript; em arrays, que fazem parte do JavaScript desde seu início; e em mapas e conjuntos, que são uma adição recente ao JavaScript.
- O Capítulo 10 enfoca as expressões regulares, um recurso frequentemente esquecido da linguagem que pode fazer o trabalho de dezenas de linhas de código quando usado corretamente. Aprenderemos como construir e usar expressões regulares e como resolver alguns problemas recorrentes de maneira elegante, usando expressões regulares e os métodos que funcionam com elas.
- No capítulo 11, aprenderemos diferentes técnicas para organizar nosso código em módulos: segmentos menores e relativamente fracamente acoplados que melhoram a estrutura e a organização de nosso código.

Finalmente, a parte 4 encerra o livro estudando como o JavaScript interage com nossas páginas da web e como os eventos são processados pelo navegador. Terminaremos o livro examinando um tópico importante, o desenvolvimento entre navegadores:

- O Capítulo 12 explora como podemos modificar dinamicamente nossas páginas por meio de APIs de manipulação de DOM e como podemos lidar com atributos, propriedades e estilos de elementos, bem como algumas considerações importantes de desempenho.
- O Capítulo 13 discute a importância do modelo de execução de thread único do JavaScript e as consequências que esse modelo tem no loop de eventos. Também aprenderemos como temporizadores e intervalos funcionam e como podemos usá-los para melhorar o desempenho percebido de nossos aplicativos da web.
- O Capítulo 14 conclui o livro examinando as cinco principais preocupações de desenvolvimento com relação a esses problemas entre navegadores: diferenças de navegador, bugs e correções de bugs, código externo e marcação, recursos ausentes e regressões. Estratégias como simulação de recursos e detecção de objetos são discutidas detalhadamente para nos ajudar a lidar com esses desafios entre navegadores.

Convenções de código

Todo o código-fonte nas listagens ou no texto está em um fonte de largura fixa como esta para separá-lo do texto comum. Nomes de métodos e funções, propriedades, elementos XML e atributos no texto também são apresentados nesta mesma fonte.

Em alguns casos, o código-fonte original foi reformatado para caber nas páginas. Em geral, o código original foi escrito com as limitações de largura da página em mente, mas às vezes você pode encontrar uma pequena diferença de formatação entre o código do livro e o fornecido no download do código-fonte. Em alguns casos raros, onde longas linhas não puderam ser reformatadas sem alterar seu significado, as listas de livros contêm marcadores de continuação de linha.

As anotações de código acompanham muitas das listagens; estes destacam conceitos importantes.

Downloads de código

O código-fonte de todos os exemplos de trabalho neste livro (junto com alguns extras que nunca foram incluídos no texto) está disponível para download na página do livro em <https://manning.com/books/secrets-of-the-javascript-ninja-segunda-edição>.

Os exemplos de código para este livro são organizados por capítulo, com pastas separadas para cada capítulo. O layout está pronto para ser servido por um servidor web local, como o servidor Apache HTTP. Descompacte o código baixado em uma pasta de sua escolha e torne essa pasta a raiz do documento do aplicativo.

Com algumas exceções, a maioria dos exemplos não requer a presença de um servidor web e pode ser carregada diretamente em um navegador para execução, se desejar.

Autor Online

Os autores e a Manning Publications convidam você para o fórum do livro, administrado pela Manning Publications, onde você pode fazer comentários sobre o livro, fazer perguntas técnicas e receber ajuda dos autores e de outros usuários. Para acessar e assinar o fórum, aponte seu navegador para <https://manning.com/books/secrets-of-the-javascript-ninja-second-edition> e clique no link Autor Online. Esta página fornece informações sobre como entrar no fórum depois de registrado, que tipo de ajuda está disponível e as regras de conduta do fórum.

O compromisso de Manning com nossos leitores é fornecer um espaço onde um diálogo significativo entre leitores individuais e entre leitores e autores possa ocorrer. Não é um compromisso com qualquer quantidade específica de participação por parte dos autores, cuja contribuição para o fórum do livro permanece voluntária (e gratuita). Sugerimos que você tente fazer algumas perguntas desafiadoras aos autores, para que o interesse deles não se desvie!

O fórum do autor online e os arquivos das discussões anteriores estarão acessíveis no site da editora, desde que o livro esteja sendo impresso.

Sobre os autores



John Resig é engenheiro da Khan Academy e criador da biblioteca jQuery JavaScript.

Além da primeira edição do

Segredos do JavaScript Ninja, ele também é o autor do livro *Técnicas de JavaScript Pro*.

John desenvolveu um abrangente banco de dados de impressão de blocos de madeira japonês e um mecanismo de busca de imagens: Ukiyo-e.org. Ele é membro do conselho da Sociedade de Arte Japonesa da América e é um visitante

Pesquisador da Ritsumeikan University trabalhando no estudo de Ukiyo-e.

John está localizado em Brooklyn, NY.



Bear Bibeault vem escrevendo software há mais de três décadas, começando com um programa Tic-Tac-Toe escrito em um supercomputador Control Data Cyber por meio de um teletipo de 100 baud. Tendo dois diplomas em engenharia elétrica, Bear deveria estar projetando antenas ou algo parecido, mas desde seu primeiro emprego na Digital Equipment Corporation, ele sempre foi muito mais fascinado por programação.

Bear também atuou em empresas como Dragon Systems, Works.com, Spreadfast, Logitech, Caringo e mais de um punhado de outras. Bear até serviu nas forças armadas dos Estados Unidos, liderando e treinando um pelotão de soldados de infantaria antitanques - habilidades que são úteis durante as reuniões do scrum. "Isso é *Sargento* Tenha paciência com você, estagiário! "

Bear é atualmente um desenvolvedor front-end sênior para um provedor líder de software de armazenamento de objetos que fornece escalabilidade de armazenamento massivo e proteção de conteúdo.

Além da primeira edição deste livro, Bear também é autor de vários outros livros de Manning, incluindo *jQuery em ação* (primeira, segunda e terceira edições), *Ajax na prática*, e *Protótipo e Scriptaculous em ação*; e ele tem sido um revisor técnico de muitos dos livros focados na web "Use a Cabeça" da O'Reilly Publishing, como *Head First Ajax*, *Head Rush Ajax*, e *Use primeiro Servlets e JSP*.

Além de seu trabalho diário, Bear também escreve livros (duh!), Dirige uma pequena empresa que cria aplicativos da web e oferece outros serviços de mídia (mas não videografia de casamento - nunca, jamais, videografia de casamento) e ajuda em CodeRanch.com como um "marechal" (moderador uber).

Quando não está plantado na frente de um computador, Bear gosta de cozinhar *grande* comida (o que explica o tamanho de seu jeans), mergulhe em fotografia e vídeo, ande em sua Yamaha V-Star e use camisetas com estampas tropicais.

Ele trabalha e mora em Austin, Texas, uma cidade que ele ama, exceto pelo trânsito e os motoristas completamente insanos.



Josip Maras é pesquisador de pós-doutorado na faculdade de engenharia elétrica, engenharia mecânica e arquitetura naval da Universidade de Split, Croácia. Ele tem um PhD em engenharia de software, com a tese "Automatizando o Reuso no Desenvolvimento de Aplicativos Web", que entre outras coisas incluiu a implementação de um interpretador JavaScript em JavaScript. Durante sua pesquisa, ele publicou mais de uma dúzia de artigos em jornais e conferências científicas, a maioria lidando com análise de programas de aplicativos da Web do lado do cliente.

Quando não está fazendo pesquisa, Josip passa seu tempo ensinando desenvolvimento para web, análise e design de sistemas e desenvolvimento para Windows (algumas centenas de alunos nos últimos seis anos). Ele também possui uma pequena empresa de desenvolvimento de software.

Em seu tempo livre, Josip gosta de ler, correr longas e, se o tempo permitir, nadar no Adriático.

sobre a ilustração da capa

A figura na capa do *Segredos do JavaScript Ninja*, segunda edição tem como legenda "Ator Noh, Samurai" de uma impressão em xilogravura de um artista japonês desconhecido de meados do século XIX. Derivado da palavra japonesa para *talento* ou *habilidade*, Noh é uma forma de drama musical clássico japonês que é apresentada desde o século XIV. Muitos personagens são mascarados, com homens desempenhando papéis masculinos e femininos. O samurai, uma figura heróica no Japão por centenas de anos, era freqüentemente apresentado nas apresentações, e nesta gravura o artista representa com grande habilidade a beleza do traje e a ferocidade do samurai.

Samurais e ninjas eram guerreiros que se destacavam na arte da guerra japonesa, conhecidos por sua bravura e astúcia. Samurais eram soldados de elite, homens bem-educados que sabiam ler e escrever, além de lutar, e estavam vinculados a um rígido código de honra chamado Bushido (O Caminho do Guerreiro), que era transmitido oralmente de geração em geração, começando no século 10. Recrutados na aristocracia e nas classes altas, análogos aos cavaleiros europeus, os samurais iam para a batalha em grandes formações, usando armaduras elaboradas e vestidos coloridos destinados a impressionar e intimidar. Os ninjas foram escolhidos por suas habilidades nas artes marciais, e não por sua posição social ou educação. Vestidos de preto e com o rosto coberto, eles foram enviados em missões sozinhos ou em pequenos grupos para atacar o inimigo com subterfúgio e furtividade, usar qualquer tática para garantir o sucesso; seu único código era o sigilo.

A ilustração da capa é de um conjunto de três gravuras japonesas de propriedade por muitos anos de um editor da Manning, e quando estávamos procurando por um ninja para a capa deste livro, o

Impressão impressionante de samurai chamou nossa atenção e foi selecionada por seus detalhes intrincados, cores vibrantes e representação vívida de um guerreiro feroz pronto para atacar - e vencer.

Em uma época em que é difícil distinguir um livro de computador do outro, Manning celebra a inventividade e a iniciativa do setor de informática com capas de livros baseadas em ilustrações de duzentos anos que retratam a rica diversidade de trajes tradicionais de todo o mundo , trazido de volta à vida por estampas como esta.

Aquecendo

Tua parte do livro definirá o cenário para seu treinamento ninja de JavaScript. No capítulo 1, veremos o estado atual do JavaScript e exploraremos alguns dos ambientes nos quais o código JavaScript pode ser executado. Colocaremos um foco especial no ambiente onde tudo começou - *o navegador*—E discutiremos algumas das melhores práticas ao desenvolver aplicativos JavaScript.

Como nossa exploração de JavaScript será feita no contexto de navegadores, no capítulo 2 ensinaremos a você o ciclo de vida dos aplicativos da Web do lado do cliente e como a execução do código JavaScript se encaixa nesse ciclo de vida.

Quando terminar esta parte do livro, você estará pronto para embarcar em seu treinamento como um ninja JavaScript!

JavaScript está em todo lugar



Este capítulo cobre

- Os principais recursos da linguagem JavaScript
- Os principais itens de um motor JavaScript
- Três melhores práticas no desenvolvimento de JavaScript

Vamos falar sobre Bob. Depois de passar alguns anos aprendendo como criar aplicativos de desktop em C ++, ele se formou como desenvolvedor de software no início dos anos 2000 e então saiu para o mundo inteiro. Nesse ponto, a web tinha acabado de começar e todos queriam ser a próxima Amazon. Então, a primeira coisa que ele fez foi aprender desenvolvimento web.

Ele aprendeu um pouco de PHP para que pudesse gerar páginas da web dinamicamente, que normalmente ele borrifava com JavaScript para obter funcionalidades complexas, como validação de formulário e até mesmo relógios dinâmicos na página! Alguns anos depois, os smartphones se tornaram uma coisa, então, antecipando a abertura de um grande mercado, Bob foi em frente e aprendeu Objective-C e Java para desenvolver aplicativos móveis que rodam em iOS e Android.

Ao longo dos anos, Bob criou muitos aplicativos de sucesso que precisam ser mantidos e ampliados. Infelizmente, saltar diariamente entre todas essas diferentes linguagens de programação e estruturas de aplicativos realmente começou a desgastar o pobre Bob.

Agora vamos falar sobre Ann. Dois anos atrás, Ann se formou em desenvolvimento de software, com especialização em aplicativos baseados na web e na nuvem. Ela criou alguns aplicativos da web de médio porte com base em estruturas modernas de model-view-controller (MVC), junto com os aplicativos móveis que os acompanham que rodam em iOS e Android. Ela criou um aplicativo de desktop que roda em Linux, Windows e SO

X, e até começou a construir uma versão sem servidor desse aplicativo totalmente baseada na nuvem. E *tudo o que ela fez foi escrito em JavaScript*.

Isso é extraordinário! O que Bob levou 10 anos e 5 idiomas para fazer, Ann conseguiu em 2 anos e em *apenas um idioma*. Ao longo da história da computação, é raro que um determinado conjunto de conhecimentos seja facilmente transferível e útil em tantos domínios diferentes.

O que começou como um humilde projeto de 10 dias em 1995 é agora uma das linguagens de programação mais amplamente usadas no mundo. JavaScript é literalmente *em toda parte*, graças a mecanismos JavaScript mais poderosos e à introdução de estruturas como Node, Apache Cordova, Ionic e Electron, que levaram a linguagem além da humilde página da web. E, como o HTML, a própria linguagem agora está recebendo atualizações há muito atrasadas com o objetivo de tornar o JavaScript ainda mais adequado para o desenvolvimento de aplicativos modernos.

Neste livro, vamos garantir que você saiba tudo o que precisa saber sobre JavaScript para que, seja como Ann ou Bob, você possa desenvolver todos os tipos de aplicativos em um campo verde ou marrom.

O que são Babel e Traceur, e por que eles são importantes

para os desenvolvedores de JavaScript de hoje?

Você sabe?

Quais são as partes principais do JavaScript de qualquer navegador da web

API usada por aplicativos da web?

1,1 *Compreender a linguagem JavaScript*

À medida que avançam em suas carreiras, muitos programadores de JavaScript como Bob e Ann chegam ao ponto em que estão usando ativamente o vasto número de elementos que formam a linguagem. Em muitos casos, entretanto, essas habilidades não podem ser levadas além dos níveis fundamentais. Nossa suposição é que isso geralmente ocorre porque o JavaScript, usando uma sintaxe semelhante a C, tem uma semelhança superficial com outras linguagens semelhantes a C (como C # e Java) e, portanto, deixa a impressão de familiaridade.

As pessoas geralmente acham que, se conhecerem C # ou Java, já terão um conhecimento bastante sólido de como o JavaScript funciona. Mas é uma armadilha! Quando comparado a outras linguagens convencionais, JavaScript é muito mais *funcionalmente* orientado. Alguns conceitos de JavaScript diferem fundamentalmente daqueles da maioria das outras linguagens.

Essas diferenças incluem o seguinte:

- *Funções são objetos de primeira classe*— Em JavaScript, as funções coexistem e podem ser tratadas como qualquer outro objeto JavaScript. Eles podem ser criados por meio de literais, referenciados por variáveis, passados como argumentos de função e até mesmo retornados como valores de retorno de função. Dedicamos grande parte do capítulo 3 para explorar alguns dos benefícios maravilhosos que funções como objetos de primeira classe trazem para nosso código JavaScript.
- *Fechamentos de funções*— O conceito de encerramentos de função é geralmente mal compreendido, mas ao mesmo tempo exemplifica fundamental e irrevogavelmente a importância das funções para o JavaScript. Por enquanto, basta saber que uma função é *um fechamento quando ele mantém ativamente (“fecha”) as variáveis externas usadas em seu corpo*. Não se preocupe por enquanto se você não vê os muitos benefícios dos fechamentos; vamos ter certeza de que tudo está claro como cristal no capítulo 5. Além dos encerramentos, exploraremos completamente os muitos aspectos das próprias funções nos capítulos 3 e 4, bem como os escopos de identificador no capítulo 5.
- *Âmbitos* - Até recentemente, o JavaScript não tinha variáveis de nível de bloco (como em outras linguagens semelhantes a C); em vez disso, tivemos que confiar apenas em variáveis globais e variáveis de nível de função.
- *Orientação a objetos baseada em protótipo* - Ao contrário de outras linguagens de programação convencionais (como C #, Java e Ruby), que usam orientação a objetos baseada em classe, JavaScript usa protótipos. Frequentemente, quando os desenvolvedores vêm para o JavaScript de linguagens baseadas em classe (como Java), eles tentam usar JavaScript como se fosse Java, essencialmente escrevendo o código baseado em classe do Java usando a sintaxe do JavaScript. Então, por algum motivo, eles ficam surpresos quando os resultados diferem do que eles esperam. É por isso que nos aprofundaremos em protótipos, como funciona a objetororientação baseada em protótipos e como é implementada em JavaScript.

JavaScript consiste em um relacionamento próximo entre objetos e protótipos, funções e encerramentos. Compreender as fortes relações entre esses conceitos pode melhorar muito sua capacidade de programação JavaScript, dando a você uma base sólida para qualquer tipo de desenvolvimento de aplicativo, independentemente de seu código JavaScript ser executado em uma página da web, em um aplicativo de desktop ou em um aplicativo móvel , ou no servidor.

Além desses conceitos fundamentais, outros recursos de JavaScript podem ajudá-lo a escrever um código mais elegante e mais eficiente. Alguns desses são recursos que desenvolvedores experientes como Bob reconhecerão de outras linguagens, como Java e C ++. Em particular, nos concentramos no seguinte:

- *Geradores*, que são funções que podem gerar vários valores por solicitação e podem suspender sua execução entre solicitações
- *Promessas*, que nos dá melhor controle sobre o código assíncrono
- *Proxies*, que nos permitem controlar o acesso a certos objetos
- *Métodos avançados de array*, que tornam o código de manipulação de array muito mais elegante

- *Mapas*, que podemos usar para criar coleções de dicionário; e *conjuntos*, que nos permitem lidar com coleções de itens exclusivos
- *Expressões regulares*, que nos permite simplificar o que de outra forma seriam partes complicadas de código
- *Módulos*, que podemos usar para quebrar o código em partes menores e relativamente autocontidas que tornam os projetos mais gerenciáveis

Ter um profundo conhecimento dos fundamentos e aprender como usar os recursos avançados de linguagem da melhor maneira possível pode elevar seu código a níveis mais altos. Aperfeiçoar suas habilidades para unir esses conceitos e recursos lhe dará um nível de compreensão que coloca a criação de qualquer tipo de aplicativo JavaScript ao seu alcance.

1.1.1 *Como o JavaScript evoluirá?*

O comitê ECMAScript, responsável pela padronização da linguagem, acaba de finalizar a versão ES7 / ES2016 do JavaScript. A versão ES7 é uma atualização relativamente pequena para o JavaScript (pelo menos, quando comparada ao ES6), porque a meta do comitê daqui para frente é se concentrar em mudanças incrementais menores e anuais no idioma.

Neste livro, exploramos completamente o ES6, mas também nos concentramos nos recursos do ES7, como a nova função assíncrona, que o ajudará a lidar com o código assíncrono (discutido no capítulo 6).



NOTA Quando cobrimos os recursos de JavaScript definidos no ES6 / ES2015 ou ES7 / ES2016, você verá esses ícones ao lado de um link para informações sobre se eles são suportados pelo seu navegador.

Atualizações incrementais anuais da especificação do idioma são excelentes notícias, mas isso não significa que os desenvolvedores da web terão acesso instantâneo aos novos recursos depois que a especificação for lançada. O código JavaScript deve ser executado por um mecanismo JavaScript, portanto, muitas vezes ficamos esperando impacientemente por atualizações em nossos mecanismos JavaScript favoritos que incorporam esses novos e interessantes recursos.

Infelizmente, embora os desenvolvedores do mecanismo de JavaScript estejam tentando se manter atualizados e melhorando o tempo todo, sempre há uma chance de você encontrar recursos que está morrendo de vontade de usar, mas que ainda não têm suporte.

Felizmente, você pode acompanhar o estado do suporte a recursos nos vários navegadores por meio das listas em <https://kangax.github.io/compat-table/es6/> , [http://kangax.github.io/mesa/compativel / es2016plus /](http://kangax.github.io/mesa/compativel/es2016plus/) , e <https://kangax.github.io/compat-table/esnext/> .

1.1.2 *Transpilers nos dão acesso ao JavaScript de amanhã hoje*

Devido aos rápidos ciclos de lançamento dos navegadores, geralmente não temos que esperar muito para que um recurso JavaScript seja compatível. Mas o que acontece se quisermos aproveitar

todos os benefícios dos mais novos recursos de JavaScript, mas são reféns de uma dura realidade: os usuários de nossos aplicativos da web ainda podem estar usando navegadores mais antigos?

Uma resposta para este problema é usar *transpiladores* ("Transformação + compilação"), ferramentas que pegam código JavaScript de ponta e o transformam em código equivalente (ou, se isso não for possível, semelhante) que funciona corretamente na maioria dos navegadores atuais.

Os transpiladores mais populares de hoje são Traceur (<https://github.com/google/traceur-compiler>) e Babel (<https://babeljs.io/>) Configurar-los é fácil; basta seguir um dos tutoriais, como <https://github.com/google/traceur-compiler/wiki/Getting-Started> ou <http://babeljs.io/docs/setup/> .

Neste livro, colocamos um foco especial na execução de código JavaScript no navegador. Para usar a plataforma do navegador de forma eficaz, você precisa sujar as mãos e estudar o funcionamento interno dos navegadores. Vamos começar!

1,2 Compreender o navegador

Atualmente, os aplicativos JavaScript podem ser executados em muitos ambientes. Mas o ambiente a partir do qual tudo começou, o ambiente a partir do qual todos os outros ambientes tiveram ideias e o ambiente no qual nos concentraremos, é o *navegador*. O navegador fornece vários conceitos e APIs para explorar completamente; veja a figura 1.1.

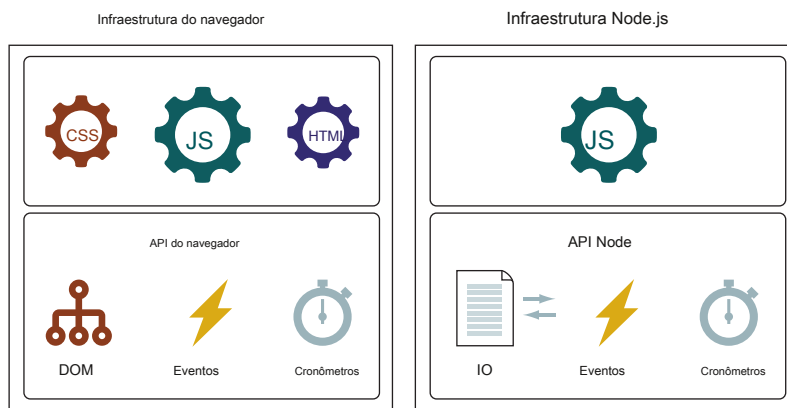


Figura 1.1 Os aplicativos da web do lado do cliente dependem da infraestrutura fornecida pelo navegador. Vamos nos concentrar particularmente no DOM, eventos, temporizadores e APIs do navegador.

Vamos nos concentrar no seguinte:

- *O Document Object Model (DOM)* - O DOM é uma representação estruturada da IU de um aplicativo da web do lado do cliente que é, pelo menos inicialmente, construído a partir do código HTML de um aplicativo da web. Para desenvolver ótimos aplicativos, você precisa não apenas ter um conhecimento profundo da mecânica básica do JavaScript, mas também estudar como o DOM é construído (capítulo 2) e como escrever um código eficaz que manipule o DOM (capítulo 12). Isso colocará a criação de interfaces de usuário avançadas e altamente dinâmicas ao seu alcance.

- *Eventos*—A grande maioria dos aplicativos JavaScript são *orientado por eventos* aplicativos, o que significa que a maior parte do código é executada no contexto de uma resposta a um determinado evento. Os exemplos de eventos incluem eventos de rede, cronômetros e eventos gerados pelo usuário, como cliques, movimentos do mouse, pressionamentos de teclado e assim por diante. Por esse motivo, exploraremos completamente os mecanismos por trás dos eventos no capítulo 13. Prestaremos atenção especial a *temporizadores*, que são frequentemente um mistério, mas vamos lidar com tarefas complexas de codificação, como cálculos de longa duração e animações suaves.
- *API do navegador*—Para nos ajudar a interagir com o mundo, o navegador oferece uma API que nos permite acessar informações sobre dispositivos, armazenar dados localmente ou nos comunicar com servidores remotos. Exploraremos algumas dessas APIs ao longo do livro.

Aperfeiçoar suas habilidades de programação JavaScript e alcançar um conhecimento profundo das APIs oferecidas pelo navegador o levará longe. Mas mais cedo, ao invés de mais tarde, você vai se deparar com *os navegadores* e seus vários problemas e inconsistências. Em um mundo perfeito, todos os navegadores estariam livres de bugs e ofereceriam suporte aos padrões da web de maneira consistente; infelizmente, não vivemos nesse mundo.

A qualidade dos navegadores melhorou muito ultimamente, mas todos eles ainda têm alguns bugs, APIs ausentes e peculiaridades específicas do navegador com as quais precisamos lidar. O desenvolvimento de uma estratégia abrangente para lidar com esses problemas do navegador e se familiarizar intimamente com suas diferenças e peculiaridades pode ser quase tão importante quanto a proficiência no próprio JavaScript.

Quando estamos escrevendo aplicativos de navegador ou bibliotecas JavaScript para serem usados neles, escolher quais navegadores dar suporte é uma consideração importante. Gostaríamos de oferecer suporte a todos eles, mas as limitações nos recursos de desenvolvimento e teste ditam o contrário. Por esse motivo, exploraremos exaustivamente as estratégias de desenvolvimento entre navegadores no capítulo 14.

O desenvolvimento de código eficaz para vários navegadores pode depender significativamente da habilidade e experiência dos desenvolvedores. Este livro tem como objetivo aumentar esse nível de habilidade, então vamos começar examinando as melhores práticas atuais.

1,3

Usando as melhores práticas atuais

O domínio da linguagem JavaScript e uma compreensão dos problemas de codificação entre navegadores são partes importantes para se tornar um desenvolvedor especialista de aplicativos da Web, mas não são o quadro completo. Para entrar nas grandes ligas, você também precisa exibir as características que dezenas de desenvolvedores anteriores comprovaram serem benéficas para o desenvolvimento de código de qualidade. Essas características são conhecidas como *Melhores Práticas*, e, além do domínio da linguagem, incluem elementos como

- Habilidades de depuração
- Testando
- Análise de desempenho

É de vital importância aderir a essas práticas ao codificar e as usaremos ao longo do livro. Vamos examinar alguns deles a seguir.

1.3.1 *Depurando*

Depurar JavaScript costumava significar usar alerta para verificar o valor das variáveis. Felizmente, a capacidade de depurar código JavaScript melhorou drasticamente, em grande parte por causa da popularidade da extensão do desenvolvedor Firebug para Firefox. Ferramentas semelhantes foram desenvolvidas para todos os principais navegadores:

- *Firebug*—A popular extensão de desenvolvedor para Firefox que fez a bola rolar (<http://getfirebug.com/>)
- *Chrome DevTools*—Desenvolvido pela equipe do Chrome e usado no Chrome e no Opera
- *Ferramentas de desenvolvedor do Firefox*—Uma ferramenta incluída no Firefox, desenvolvida pela equipe do Firefox
- *Ferramentas de desenvolvedor F12*—Incluído no Internet Explorer e Microsoft Edge
- *Inspetor WebKit*—Usado pelo Safari

Como você pode ver, todos os principais navegadores oferecem ferramentas de desenvolvedor que podemos usar para depurar nossos aplicativos da web. Os dias de usar alertas JavaScript para depuração já se foram!

Todas essas ferramentas são baseadas em ideias semelhantes, que foram introduzidas principalmente pelo Firebug, de modo que oferecem funcionalidades semelhantes: explorar o DOM, depurar JavaScript, editar estilos CSS, rastrear eventos de rede e assim por diante. Qualquer um deles fará um bom trabalho; use aquele oferecido pelo navegador de sua escolha ou no navegador no qual você está investigando bugs.

Além disso, você pode usar alguns deles, como o Chrome Dev Tools, para depurar outros tipos de aplicativos, como aplicativos Node.js. (Apresentaremos algumas técnicas de depuração no Apêndice B.)

1.3.2 *Testando*

Ao longo deste livro, aplicaremos técnicas de teste para garantir que o código de exemplo opere conforme pretendido e para servir como exemplos de como testar o código em geral. A principal ferramenta que usaremos para teste é um afirmar função, cujo propósito é afirmar que uma premissa é verdadeira ou falsa. Ao especificar asserções, podemos verificar se o código está se comportando conforme o esperado.

A forma geral desta função é a seguinte:

afirmar (condição, mensagem);

O primeiro parâmetro é uma condição que deveria ser verdadeira, e o segundo é uma mensagem que será exibida se não for.

Considere isto, por exemplo:

```
assert (a === 1, "Desastre! a não é 1!");
```

Se o valor da variável `uma` não é igual a 1, a afirmação falha e a mensagem um tanto excessivamente dramática é exibida.

NOTA O afirmar função não é um recurso padrão da linguagem, então vamos implementá-la nós mesmos no apêndice B.

1.3.3 *Análise de desempenho*

Outra prática importante é a análise de desempenho. Os motores JavaScript deram passos surpreendentes no desempenho do JavaScript, mas isso não é desculpa para escrever código desleixado e ineficiente.

Usaremos um código como o seguinte posteriormente neste livro para coletar informações de desempenho:

```
console.time("Minha operação");  
  
para (var n = 0; n < maxCount; n++) {  
    /* realizar a operação a ser medida */  
}  
  
console.timeEnd("Minha operação");
```

Inicia o cronômetro

Executa a operação várias vezes

Para o cronômetro

Aqui, colocamos entre parênteses a execução do código a ser medido com duas chamadas para o `Time` e `timeEnd` métodos do embutido `console` objeto.

Antes que a operação comece a ser executada, a chamada para `console.time` inicia um cronômetro com um nome (neste caso, `Minha operação`). Em seguida, executamos o código no `for` loop um certo número de vezes (neste caso, `maxCount` vezes). Como uma única operação do código acontece muito rapidamente para ser medida de maneira confiável, precisamos executar o código muitas vezes para obter um valor mensurável. Frequentemente, essa contagem pode chegar a dezenas de milhares, ou mesmo milhões, dependendo da natureza do código que está sendo medido. Uma pequena tentativa e erro nos permite escolher um valor razoável.

Quando a operação termina, chamamos o `console.timeEnd` método com o mesmo nome. Isso faz com que o navegador exiba o tempo decorrido desde que o cronômetro foi iniciado.

Essas técnicas de prática recomendada, junto com outras que você aprenderá ao longo do caminho, irão aprimorar muito o seu desenvolvimento de JavaScript. O desenvolvimento de aplicativos com os recursos restritos que um navegador oferece, juntamente com o mundo cada vez mais complexo da capacidade e compatibilidade do navegador, requer um conjunto robusto e completo de habilidades.

1,4 *Aumentando a capacidade de transferência de habilidades*

Quando Bob estava aprendendo desenvolvimento web pela primeira vez, cada navegador tinha sua própria maneira de interpretar estilos de script e IU, pregando que seu jeito era o melhor e fazendo todo desenvolvedor ranger de frustração. Felizmente, as guerras dos navegadores terminaram com a padronização de HTML, CSS, a API DOM e o JavaScript e o foco do desenvolvedor voltando-se para aplicativos JavaScript entre navegadores eficazes. Na verdade, este foco em

tratar sites como aplicativos levou a muitas ideias, ferramentas e técnicas que passaram de aplicativos de desktop para aplicativos da web. E agora, essa transferência de conhecimento e ferramentas aconteceu novamente, à medida que ideias, ferramentas e técnicas originadas no desenvolvimento da Web no cliente também permearam outros domínios de aplicação.

Alcançar um entendimento profundo dos princípios fundamentais do JavaScript com o conhecimento das principais APIs pode, portanto, torná-lo um desenvolvedor mais versátil. Usando os navegadores e Node.js (um ambiente derivado do navegador), você pode desenvolver quase qualquer tipo de aplicativo imaginável:

- *Aplicativos de desktop*, usando, por exemplo, NW.js (<http://nwjs.io/>) ou elêtron (<http://electron.atom.io/>)
Essas tecnologias geralmente envolvem o navegador para que possamos construir UIs de desktop com HTML, CSS e JavaScript padrão (dessa forma, podemos confiar em nosso JavaScript básico e conhecimento do navegador), com suporte adicional que torna possível interagir com o sistema de arquivos. Podemos construir aplicativos de desktop verdadeiramente independentes de plataforma que tenham a mesma aparência no Windows, Mac e Linux.
- *Aplicativos móveis com estruturas*, como Apache Cordova (<https://cordova.apache.org/>) Semelhante aos aplicativos de desktop desenvolvidos com tecnologias da web, as estruturas para aplicativos móveis usam um navegador empacotado, mas com APIs adicionais específicas da plataforma que nos permitem interagir com a plataforma móvel.
- *Aplicativos do lado do servidor e aplicativos para dispositivos incorporados com Node.js*, um ambiente derivado do navegador que usa muitos dos mesmos princípios básicos do navegador. Por exemplo, Node.js executa código JavaScript e depende de eventos.

Ann não sabe o quão sortuda ela é (embora Bob tenha uma boa ideia). Não importa se ela precisa construir um aplicativo de desktop padrão, um aplicativo móvel, um aplicativo do lado do servidor ou mesmo um aplicativo incorporado - todos esses tipos de aplicativos compartilham alguns dos mesmos princípios básicos dos aplicativos da web do lado do cliente padrão. Ao entender como a mecânica principal do JavaScript funciona e compreender as APIs principais fornecidas pelos navegadores (como eventos, que também têm muito em comum com os mecanismos fornecidos pelo Node.js), ela pode impulsionar suas habilidades de desenvolvimento em todo o quadro. Como você pode. No processo, você se tornará um desenvolvedor mais versátil e ganhará o conhecimento e a compreensão para lidar com uma ampla variedade de problemas.

1,5 Resumo

- Os aplicativos da Web do lado do cliente estão entre os mais populares hoje, e os conceitos, ferramentas e técnicas antes usados apenas para seu desenvolvimento permearam outros domínios de aplicativo. Compreender os fundamentos dos aplicativos da web do lado do cliente o ajudará a desenvolver aplicativos para uma ampla variedade de domínios.

- Para melhorar suas habilidades de desenvolvimento, você precisa obter um conhecimento profundo da mecânica central do JavaScript, bem como da infraestrutura fornecida pelos navegadores.
- Este livro enfoca os principais mecanismos do JavaScript, como funções, encerramentos de função e protótipos, bem como novos recursos do JavaScript, como geradores, promessas, proxies, mapas, conjuntos e módulos.
- JavaScript pode ser executado em um grande número de ambientes, mas o ambiente onde tudo começou, e o ambiente no qual nos concentraremos, é o navegador.
- Além do JavaScript, exploraremos os aspectos internos do navegador, como o DOM (uma representação estruturada da IU da página da web) e eventos, porque os aplicativos da web do lado do cliente são aplicativos orientados a eventos.
- Faremos essa exploração com as melhores práticas em mente: depuração, teste e análise de desempenho.

Prédio

a página em tempo de execução

Este capítulo cobre

- Etapas no ciclo de vida de um aplicativo da web
- Processando código HTML para produzir uma página da web
- Ordem de execução do código JavaScript
- Alcançando interatividade com eventos
- O loop de eventos

Nossa exploração de JavaScript é realizada no contexto de aplicativos da Web do lado do cliente e do navegador como o mecanismo que executa o código JavaScript. Para ter uma base sólida a partir da qual continuar explorando JavaScript como linguagem e o navegador como plataforma, primeiro temos que entender o ciclo de vida completo do aplicativo da Web e, especialmente, como nosso código JavaScript se encaixa nesse ciclo de vida.

Neste capítulo, exploraremos completamente o ciclo de vida dos aplicativos da Web do lado do cliente, desde o momento em que a página é solicitada, por meio de várias interações realizadas pelo usuário, até o fechamento da página. Primeiro, vamos explorar como a página é construída processando o código HTML. Em seguida, vamos nos concentrar na execução do código JavaScript, que adiciona a dinâmica muito necessária às nossas páginas. E finalmente

veremos como os eventos são tratados para desenvolver aplicativos interativos que respondem às ações dos usuários.

Durante esse processo, exploraremos alguns conceitos fundamentais de aplicativos da web, como o DOM (uma representação estruturada de uma página da web) e o loop de eventos (determina como os eventos são tratados pelos aplicativos). Vamos mergulhar!

O navegador sempre constrói a página exatamente de acordo
para o HTML fornecido?

Você sabe?

Quantos eventos um aplicativo da web pode manipular de uma vez? Por que os navegadores
devem usar uma fila de eventos para processar eventos?

2,1

A visão geral do ciclo de vida

O ciclo de vida de um aplicativo da Web típico do lado do cliente começa com o usuário digitando uma URL na barra de endereços do navegador ou clicando em um link. Digamos que desejamos pesquisar um termo e acessar a página inicial do Google. Nós digitamos o URL www.google.com, conforme mostrado no canto superior esquerdo na figura 2.1.

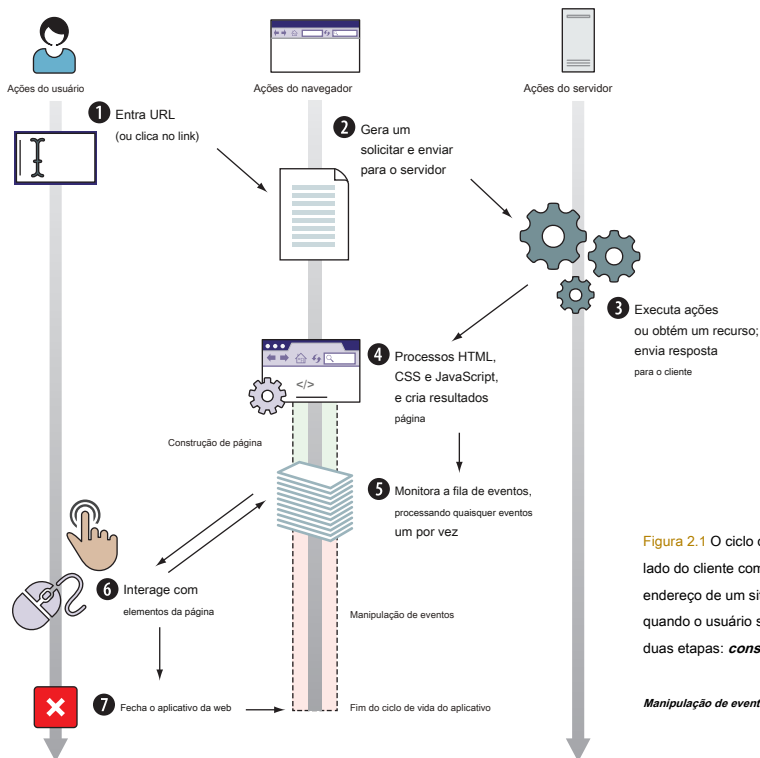


Figura 2.1 O ciclo de vida de um aplicativo da web do lado do cliente começa com o usuário especificando o endereço de um site (ou clicando em um link) e termina quando o usuário sai da página da web. É composto de duas etapas: **construção de página** e

Manipulação de eventos.

Em nome do usuário, o navegador formula uma solicitação que é enviada a um servidor **C**, que processa o pedido **D** e formula uma resposta que geralmente é composta de Código HTML, CSS e JavaScript. No momento em que o navegador recebe esta resposta **E** é quando nosso aplicativo da web do lado do cliente realmente começa a ganhar vida.

Como os aplicativos da web do lado do cliente são aplicativos de interface gráfica do usuário (GUI), seu ciclo de vida segue fases semelhantes a outros aplicativos de GUI (pense em aplicativos de desktop padrão ou aplicativos móveis) e é realizado nas duas etapas a seguir:

- 1 *Construção de página*—Configure a interface do usuário.
- 2 *Manipulação de eventos*—Digite um loop **F** esperando que os eventos ocorram **G**, e iniciar invok-manipuladores de eventos.

O ciclo de vida do aplicativo termina quando o usuário fecha ou sai da página da web **H**.

Agora, vamos ver um exemplo de aplicativo da web com uma interface de usuário simples que reage ao usuário ações: sempre que um usuário move o mouse ou clica na página, uma mensagem é exibida. Usaremos esse aplicativo ao longo do capítulo.

Listagem 2.1 Pequeno aplicativo da web com uma GUI reagindo a eventos

```
<!DOCTYPE html>
<html>
  <head>
    <title> Ciclo de vida do aplicativo da web </title>
    <style>
      # primeira {cor: verde;}
      # segundo {cor: vermelho;}
    </style>
  </head>
  <body>
    <ul id = "first"> </ul>

    <script>
      function addMessage (elemento, mensagem) {
        var messageElement = document.createElement ("li"); messageElement.textContent =
          mensagem;
        element.appendChild (messageElement);
      }

      var primeiro = document.getElementById ("primeiro"); addMessage (primeiro,
        "Carregamento de página");
    </script>

    <ul id = "second"> </ul>

    <script>
      document.body.addEventListener ("mousemove", function () {
        var segundo = document.getElementById ("segundo"); addMessage (segundo, "Evento:
          mousemove");
      });

      document.body.addEventListener ("click", function () {
```

Define uma função
que adiciona uma
mensagem a um elemento

Anexa mousemove
manipulador de eventos para o corpo

Anexa evento de clique
manipulador para o corpo

```

        var segundo = document.getElementById ("segundo"); addMessage (segundo, "Evento:
        clique");
    });
</script>
</body>
</html>

```

A Listagem 2.1 primeiro define duas regras CSS, # primeiro e # segundo, que especificam a cor do texto para os elementos com os IDs primeiro e segundo (para que possamos distinguir facilmente entre eles). Continuamos definindo um elemento de lista com o id primeiro:

```
<ul id = "first"> </ul>
```

Então, definimos um adicionar mensagem função que, quando chamada, cria um novo elemento de item de lista, define seu conteúdo de texto e o anexa a um elemento existente:

```

function addMessage (elemento, mensagem) {
    var messageElement = document.createElement ("li"); messageElement.textContent =
    mensagem;
    elemento.appendChild (messageElement);
}

```

Seguimos isso usando o integrado getElementById método para buscar um elemento com o ID primeiro do documento e adicionando uma mensagem a ele que nos notifica que a página está carregando:

```

var primeiro = document.getElementById ("primeiro"); addMessage (primeiro,
"Carregamento de página");

```

Em seguida, definimos outro elemento de lista, agora com o atributo ID segundo:

```
<ul id = "second"> </ul>
```

Finalmente, anexamos dois manipuladores de eventos ao corpo da página da web. Começamos com o mousemove manipulador de eventos, que é executado toda vez que o usuário move o mouse e adiciona uma mensagem " Evento: mousemove " ao segundo elemento de lista chamando o adicionar mensagem função:

```

document.body.addEventListener ("mousemove", function () {
    var segundo = document.getElementById ("segundo"); addMessage (segundo, "Evento:
    mousemove");
});

```

Também registramos um clique manipulador de eventos, que, sempre que o usuário clica na página, registra uma mensagem " Evento: clique em ", também para o segundo elemento da lista:

```

document.body.addEventListener ("click", function () {
    var segundo = document.getElementById ("segundo"); addMessage (segundo, "Evento:
    clique");
});

```


O resultado da execução e interação com este aplicativo é mostrado na figura 2.2.

Usaremos este aplicativo de exemplo para explorar e ilustrar as diferenças entre as diferentes fases do ciclo de vida do aplicativo da web. Vamos começar com a fase de criação de página.

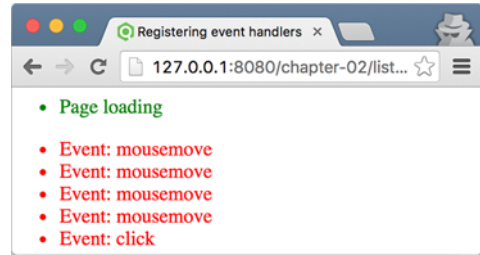


Figura 2.2 Quando o código da listagem 2.1 é executado, as mensagens são registradas dependendo das ações do usuário.

2,2

A fase de construção da página

Antes que um aplicativo da web possa ser interagido ou até mesmo exibido, a página deve ser construído a partir das informações em sua resposta recebida do servidor (geralmente código HTML, CSS e JavaScript). O objetivo desta fase de construção de página é configurar a UI de um aplicativo e isso é feito em duas etapas distintas:

- 1 Analisando o HTML e construindo o Document Object Model (DOM)
- 2 Executando código JavaScript

A etapa 1 é realizada quando o navegador está processando nós HTML, e a etapa 2 é realizada sempre que um tipo especial de elemento HTML - o roteiro elemento (que contém ou se refere ao código JavaScript) - é encontrado. Durante a fase de construção da página, o navegador pode alternar entre essas duas etapas quantas vezes forem necessárias, conforme mostrado na figura 2.3.

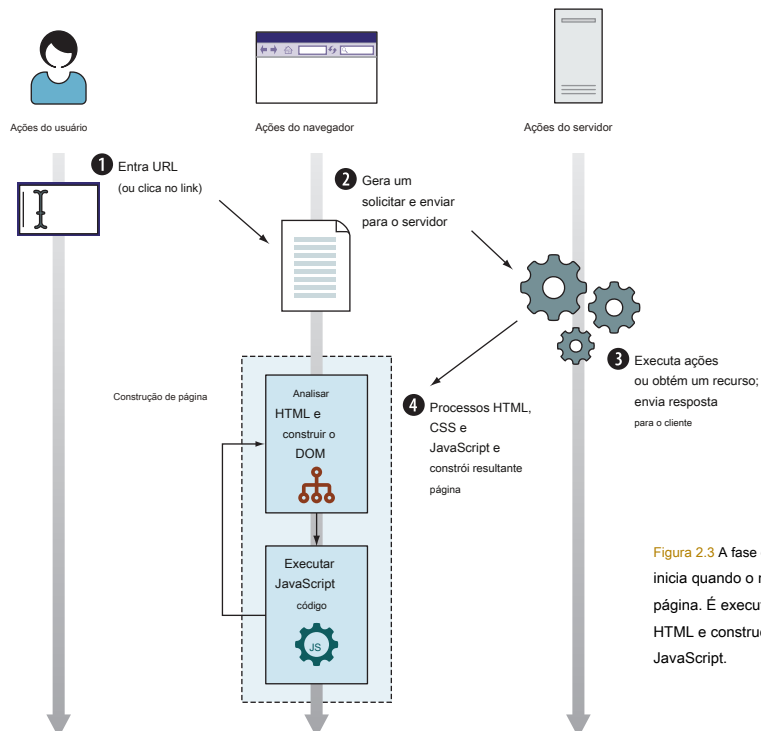


Figura 2.3 A fase de construção da página inicia quando o navegador recebe o código da página. É executado em duas etapas: análise do HTML e construção do DOM e execução do código JavaScript.

2.2.1 Analisando o HTML e construindo o DOM

A fase de construção da página começa com o navegador recebendo o código HTML, que é usado como base sobre a qual o navegador constrói a IU da página. O navegador faz isso analisando o código HTML, um elemento HTML por vez, e construindo um DOM, uma representação estruturada da página HTML na qual cada elemento HTML é representado como um nó. Por exemplo, a figura 2.4 mostra o DOM da página de exemplo que foi construída até o primeiro roteiro elemento é alcançado.

Observe como os nós na figura 2.4 são organizados de tal forma que cada nó, exceto o primeiro (a raiz **html** nó **B**) tem exatamente um dos pais. Por exemplo, o cabeça nó **C** tem o **html** nó **B** como seu pai. Ao mesmo tempo, um nó pode ter qualquer número de filhos. Por exemplo, o **html** nó **B** tem dois filhos: o cabeça nó **C** e a corpo nó **H**. Filhos do mesmo elemento são chamados *irmãos*. (O cabeça nó **C** e a corpo nó **H** são irmãos.)

É importante enfatizar que, embora o HTML e o DOM sejam intimamente vinculados, com o DOM sendo construído a partir de HTML, eles não são o mesmo. Você deve pensar no código HTML como um *planta* o navegador segue ao construir o DOM inicial - a IU - da página. O navegador pode até corrigir problemas que

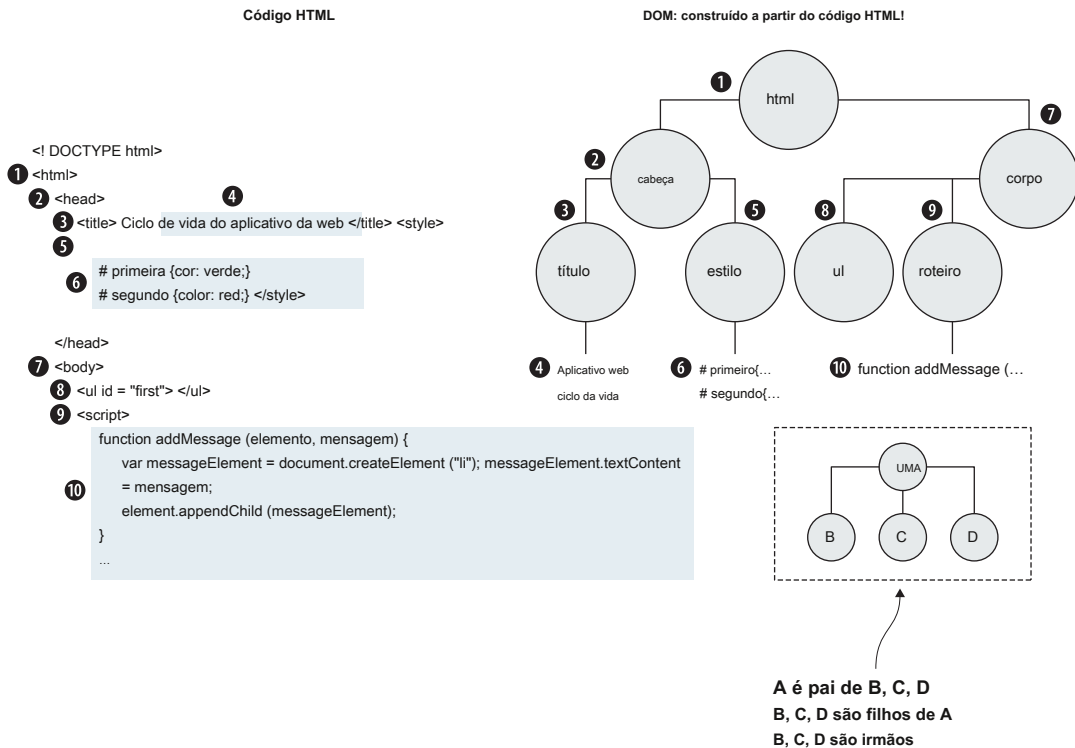


Figura 2.4 No momento em que o navegador encontra o primeiro **roteiro** elemento, ele já criou um DOM com vários elementos HTML (os nós à direita).

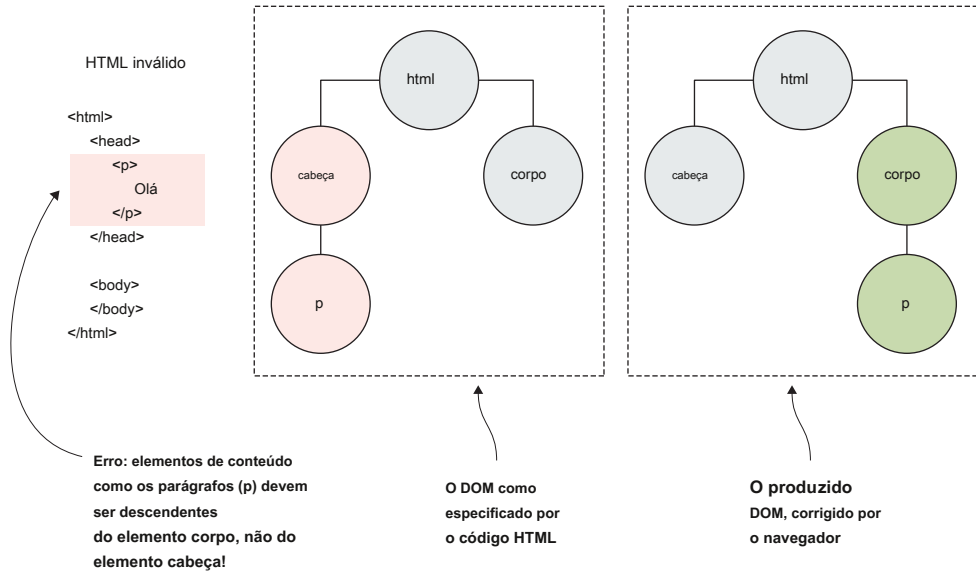


Figura 2.5 Um exemplo de HTML inválido corrigido pelo navegador

encontra com este blueprint para criar um DOM válido. Vamos considerar o exemplo mostrado na figura 2.5.

A Figura 2.5 dá um exemplo simples de código HTML incorreto no qual um elemento de parágrafo é colocado no cabeça elemento. A intenção do cabeça elemento é usado para fornecer informações gerais da página: por exemplo, o título da página, codificações de caracteres e estilos e scripts externos. Não se destina a definir o conteúdo da página, como neste exemplo. Por ser um erro, o navegador o corrige silenciosamente, construindo o DOM correto (à direita na figura 2.5), no qual o elemento de parágrafo é colocado no corpo elemento, onde o conteúdo da página deveria estar.

Especificação HTML e especificação DOM

A versão atual do HTML é HTML5, cuja especificação está disponível em <https://html.spec.whatwg.org/>. Se você precisar de algo mais legível, recomendamos o guia HTML5 da Mozilla, disponível em <https://developer.mozilla.org/en-US/docs/Guide/HTML/HTML5>.

O DOM, por outro lado, está evoluindo um pouco mais devagar. A versão atual é DOM3, cuja especificação está disponível em <https://dom.spec.whatwg.org/>. Novamente, a Mozilla preparou um relatório que pode ser encontrado em https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.

Durante a construção da página, o navegador pode encontrar um tipo especial de elemento HTML, o roteiro elemento, que é usado para incluir o código JavaScript. Quando isso acontece, o navegador pausa a construção do DOM a partir do código HTML e começa a executar o código JavaScript.

2.2.2 *Executando código JavaScript*

Todo o código JavaScript contido no roteiro elemento é executado pelo mecanismo JavaScript do navegador; por exemplo, Spidermonkey do Firefox, Chrome e Opera's V8, ou Edge's (IE's) Chakra. Como o objetivo principal do código JavaScript é fornecer dinamismo à página, o navegador fornece uma API por meio de um objeto global que pode ser usado pelo mecanismo JavaScript para interagir e modificar a página.

OBJETOS GLOBAIS EM JAVASCRIPT

O principal objeto global que o navegador expõe ao mecanismo JavaScript é o `global` objeto, que representa a janela na qual uma página está contida. O `global` objeto é o objeto global por meio do qual todos os outros objetos globais, variáveis globais (mesmo aquelas definidas pelo usuário) e APIs do navegador são acessíveis. Uma das propriedades mais importantes do `global` objeto é o `document`, que representa o DOM da página atual. Ao usar este objeto, o código JavaScript pode alterar o DOM da página em qualquer grau, modificando ou removendo elementos existentes e até mesmo criando e inserindo novos.

Vejamos um snippet de código da listagem 2.1:

```
var primeiro = document.getElementById ("primeiro");
```

Este exemplo usa o `global document` objeto para selecionar um elemento com o ID `primeiro` do DOM e atribui-lo a uma variável `primeiro`. Podemos então usar o código JavaScript para fazer todos os tipos de modificações nesse elemento, como alterar seu conteúdo textual, modificar seus atributos, criar e adicionar dinamicamente novos filhos a ele e até mesmo remover o elemento do DOM.

APIs de navegador

Ao longo do livro, usamos uma série de objetos e funções embutidos no navegador (por exemplo, `window` e `document`). Infelizmente, cobrir tudo que é suportado pelo navegador está além do escopo de um livro sobre JavaScript. Felizmente, a Mozilla novamente nos apoia com <https://developer.mozilla.org/en-US/docs/Web/API>, onde você pode encontrar o status atual das interfaces de API da web.

Com esse entendimento básico dos objetos globais fornecidos pelo navegador, vamos examinar dois tipos diferentes de código JavaScript que definem exatamente quando esse código é executado.

DIFERENTES TIPOS DE CÓDIGO JAVASCRIPT

Diferenciamos amplamente dois tipos diferentes de código JavaScript: *código global* e *código de função*. A listagem a seguir o ajudará a entender as diferenças entre esses dois tipos de código.

Listagem 2.2 Código JavaScript global e de função

```
<script>
  function addMessage (elemento, mensagem) {
    var messageElement = document.createElement ("li"); messageElement.textContent =
    mensagem;
    element.appendChild (messageElement);
  }

  var primeiro = document.getElementById ("primeiro"); addMessage (primeiro,
  "Carregamento de página");
</script>
```

Código de função é o código contido em uma função.

Código global é o código fora das funções.

A principal diferença entre esses dois tipos de código é o seu posicionamento: o código contido em uma função é chamado *código de função*, enquanto o código colocado fora de todas as funções é chamado *código global*.

Esses dois tipos de código também diferem em sua execução (você verá algumas diferenças adicionais posteriormente, especialmente no capítulo 5). O código global é executado automaticamente pelo mecanismo JavaScript (mais sobre isso em breve) de maneira direta, linha por linha, conforme é encontrado. Por exemplo, na listagem 2.2, as partes do código global que definem o adicionar mensagem função usar o embutido getElementById método para buscar o elemento com ID primeiro e ligue para o adicionar mensagem função; eles são executados um após o outro conforme são encontrados, conforme mostrado na figura 2.6.

Por outro lado, o código da função, para ser executado, deve ser chamado por outra coisa: ou pelo código global (por exemplo, o adicionar mensagem chamada de função no código global causa a execução do adicionar mensagem código de função), por alguma outra função ou pelo navegador (mais sobre isso em breve).

EXECUÇÃO DO CÓDIGO JAVASCRIPT NA FASE DE CONSTRUÇÃO DE PÁGINA

Quando o navegador atinge o roteiro na fase de construção da página, ele pausa a construção do DOM com base no código HTML e começa a executar o código JavaScript.

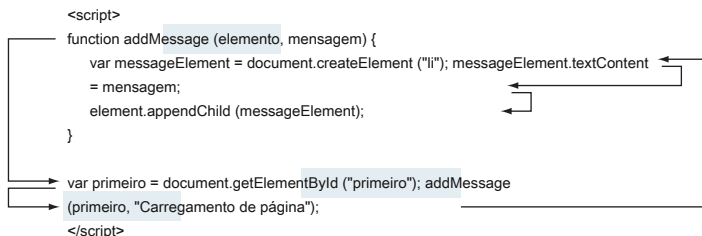


Figura 2.6 Fluxo de execução do programa ao executar o código JavaScript

Isso significa executar o código JavaScript global contido no roteiro elemento (e funções chamadas pelo código global também são executadas). Vamos voltar ao exemplo da listagem 2.1.

A Figura 2.7 mostra o estado do DOM depois que o código JavaScript global foi executado. Vamos caminhar lentamente por sua execução. Primeiro uma função adicionar mensagem é definido:

```
function addMessage (elemento, mensagem) {
    var messageElement = document.createElement ("li"); messageElement.textContent =
    mensagem;
    element.appendChild (messageElement);
}
```

Em seguida, um elemento existente é buscado no DOM usando o global documento objeto e seu getElementById método:

```
var primeiro = document.getElementById ("primeiro");
```

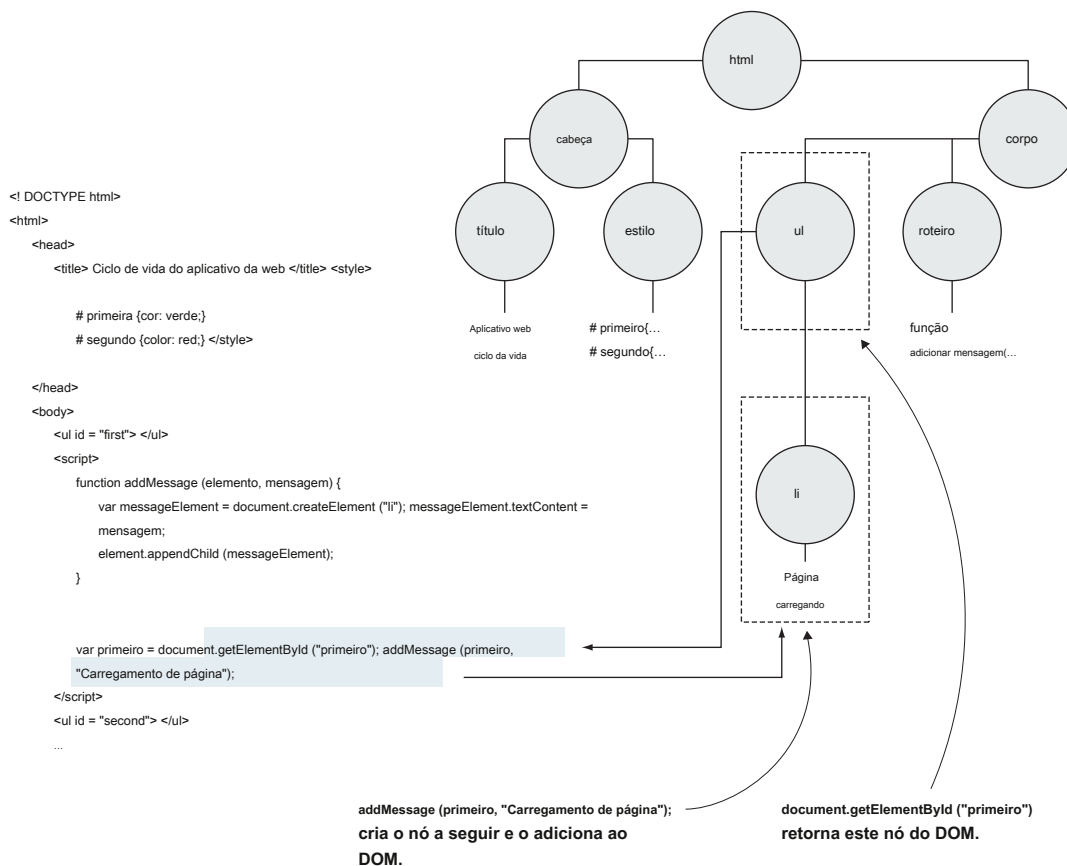


Figura 2.7 O DOM da página após a execução do código JavaScript contido no **roteiro** elemento

Isso é seguido por uma chamada para o adicionar mensagem função

```
addMessage (primeiro, "Carregamento de página");
```

que causa a criação de um novo li elemento, a modificação de seu conteúdo de texto e, finalmente, sua inserção no DOM.

Neste exemplo, o código JavaScript modifica o DOM atual criando um novo elemento e inserindo-o no DOM. Mas, em geral, o código JavaScript pode modificar o DOM em qualquer grau: ele pode criar novos nós e modificar ou remover nós DOM existentes. Mas também há algumas coisas que ele não pode fazer, como selecionar e modificar elementos que ainda não foram criados. Por exemplo, não podemos selecionar e modificar o ul elemento com o ID segundo, porque esse elemento é encontrado após o atual roteiro nó e ainda não foi alcançado e criado. Essa é uma das razões pelas quais as pessoas tendem a colocar seus roteiro elementos na parte inferior da página. Dessa forma, não precisamos nos preocupar se um determinado elemento HTML foi atingido.

Depois que o mecanismo de JavaScript executa a última linha do código JavaScript no roteiro elemento (na figura 2.5, isso significa retornar do adicionar mensagem), o navegador sai do modo de execução JavaScript e continua construindo nós DOM processando o código HTML restante. Se, durante esse processamento, o navegador encontrar novamente um roteiro , a criação do DOM a partir do código HTML é novamente pausada e o tempo de execução do JavaScript começa a executar o código JavaScript contido. É importante observar que o estado global do aplicativo JavaScript persiste enquanto isso. Todas as variáveis globais definidas pelo usuário criadas durante a execução do código JavaScript em um roteiro elemento são normalmente acessíveis ao código JavaScript em outro roteiro elementos Isso acontece porque o global janela O objeto, que armazena todas as variáveis JavaScript globais, está ativo e acessível durante todo o ciclo de vida da página.

Essas duas etapas

- 1 Construindo o DOM a partir de HTML
- 2 Executando código JavaScript

são repetidos enquanto houver HTML elementos para processar e código JavaScript a ser executado.

Finalmente, quando o navegador fica sem elementos HTML para processar, a fase de construção da página está concluída. O navegador então passa para a segunda parte do ciclo de vida do aplicativo da web: *Manipulação de eventos*.

2,3

Manipulação de eventos

Os aplicativos da Web do lado do cliente são aplicativos GUI, o que significa que eles reagem a diferentes tipos de eventos: movimentos do mouse, cliques, pressionamentos do teclado e assim por diante. Por esse motivo, o código JavaScript executado durante a fase de construção da página, além de influenciar o estado global do aplicativo e modificar o DOM, também pode registrar ouvintes (ou manipuladores) de eventos: funções que são executadas pelo navegador quando ocorre um evento. Com esses manipuladores de eventos, nós fornecemos interatividade para nossos aplicativos. Mas antes de examinar mais de perto o registro de manipuladores de eventos, vamos examinar as idéias gerais por trás do tratamento de eventos.

2.3.1 Visão geral do tratamento de eventos

O ambiente de execução do navegador é, em sua essência, baseado na ideia de que apenas uma única parte do código pode ser executada de uma vez: o chamado *single-threaded* modelo de execução. Pense em uma fila no banco. Todos ficam em uma única fila e têm que esperar sua vez de serem “processados” pelos caixas. Mas com JavaScript, apenas 1 a janela do caixa está aberta! Os clientes (eventos) são processados apenas um de cada vez, quando chega a sua vez. Basta uma pessoa que ache apropriado fazer seu planejamento financeiro para todo o ano fiscal enquanto está na janela do caixa (todos nós encontramos com ela!) Para obstruir o trabalho.

Sempre que ocorre um evento, o navegador deve executar a função de manipulador de eventos associada. Mas não há garantia de que temos usuários extremamente pacientes que sempre esperarão um período de tempo apropriado antes de acionar outro evento. Por esse motivo, o navegador precisa de uma maneira de rastrear os eventos que ocorreram, mas ainda não foram processados. Para fazer isso, o navegador usa um *fila de eventos*, conforme mostrado na figura 2.8.

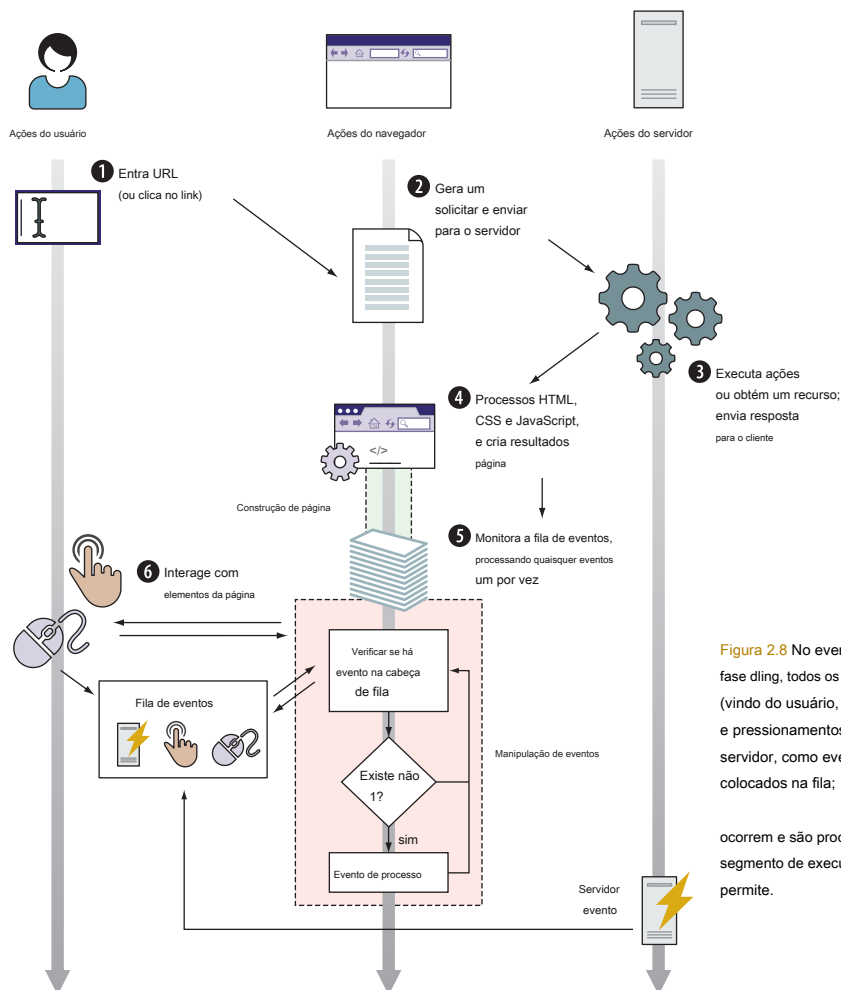


Figura 2.8 No evento-handling, todos os eventos (vindo do usuário, como cliques do mouse e pressionamentos de tecla, ou vindo do servidor, como eventos Ajax) são colocados na fila;

ocorrem e são processados como o único segmento de execução permite.

Todos os eventos gerados (não importa se são gerados pelo usuário, como movimentos do mouse ou pressionamentos de tecla, ou gerados pelo servidor, como eventos Ajax) são colocados na mesma fila de eventos, na ordem em que são detectado pelo navegador. Conforme mostrado no meio da figura 2.8, o processo de manipulação de eventos pode então ser descrito com um fluxograma simples:

- O navegador verifica o início da fila de eventos. Se não houver
- eventos, o navegador continua verificando.
- Se houver um evento no topo da fila de eventos, o navegador o pega e executa o manipulador associado (se houver). Durante essa execução, o restante dos eventos aguardam pacientemente na fila de eventos pelo seu turno de processamento.

Como apenas um evento é tratado por vez, temos que ser extremamente cuidadosos sobre a quantidade de tempo necessária para tratar os eventos; escrever manipuladores de eventos que levam muito tempo para executar leva a aplicativos da web que não respondem! (Não se preocupe se isso soar um pouco vago; voltaremos ao loop de eventos no capítulo 13 e veremos exatamente como isso afeta o desempenho percebido dos aplicativos da web).

É importante observar que o mecanismo do navegador que coloca eventos *para* a fila é externa às fases de criação de página e tratamento de eventos. O processamento necessário para determinar quando os eventos ocorreram e que os empurra para a fila de eventos não participa do thread que está *tratando* os eventos.

EVENTOS SÃO ASSÍNCRONOS

Os eventos, quando acontecem, podem ocorrer em momentos imprevisíveis e em uma ordem imprevisível (é difícil forçar os usuários a pressionar teclas ou clicar em alguma ordem específica). Dizemos que o tratamento de eventos e, portanto, a invocação de suas funções de tratamento, é *assíncrono*.

Os seguintes tipos de eventos podem ocorrer, entre outros:

- Eventos do navegador, como quando uma página termina de carregar ou quando deve ser descarregada. Eventos de rede, como respostas vindas do servidor (eventos Ajax, eventos do lado do servidor)
- Eventos do usuário, como cliques do mouse, movimentos do mouse e pressionamentos de tecla. Eventos de cronômetro, como quando um tempo limite expira ou um intervalo é disparado

A grande maioria do código é executada como resultado de tais eventos!

O conceito de manipulação de eventos é central para os aplicativos da Web e é algo que você verá repetidas vezes ao longo dos exemplos neste livro: O código é configurado com antecedência para ser executado posteriormente. Exceto pelo código global, a grande maioria do código que colocamos em uma página será executada como resultado de algum evento.

Antes que os eventos possam ser manipulados, nosso código deve notificar o navegador de que estamos interessados em manipular eventos específicos. Vejamos como registrar manipuladores de eventos.

2.3.2 Registrando manipuladores de eventos

Como já mencionamos, os manipuladores de eventos são funções que desejamos executar sempre que ocorrer um evento específico. Para que isso aconteça, temos que notificar o navegador

que estamos interessados em um evento. Isso é chamado *registro de manipulador de eventos*. Em aplicativos da web do lado do cliente, existem duas maneiras de registrar eventos:

- Atribuindo funções a propriedades especiais
- Usando o integrado `addEventListener` método

Por exemplo, escrever o código a seguir atribui uma função ao especial carregando propriedade do janela objeto:

```
window.onload = function () {};
```

Um manipulador de eventos para o carregar evento (quando o DOM está pronto e totalmente construído) é registrado. (Não se preocupe se a notação no lado direito do operador de atribuição parecer um pouco estranha; falaremos muito sobre as funções nos capítulos posteriores.) Da mesma forma, se quisermos registrar um manipulador para o clique evento no corpo do documento, podemos escrever algo neste sentido:

```
document.body.onclick = function () {};
```

Atribuir funções a propriedades especiais é uma maneira fácil e direta de registrar manipuladores de eventos, e provavelmente você já se deparou com isso. Mas não recomendamos que você registre manipuladores de eventos dessa maneira, porque fazer isso tem uma desvantagem: só é possível registrar um manipulador de função para um determinado evento. Isso significa que é fácil sobrescrever funções de manipulador de eventos anteriores, o que pode ser um pouco frustrante. Felizmente, há uma alternativa: o `addEventListener` método nos permite registrar quantas funções de manipulador de eventos forem necessárias. Para mostrar um exemplo, a listagem a seguir retorna a um trecho do exemplo da listagem 2.1.

Listagem 2.3 Registrando manipuladores de eventos

```
<script>
  document.body.addEventListener("mousemove", function () {
    var segundo = document.getElementById("segundo"); addMessage(segundo, "Evento:
    mousemove");
  });

  document.body.addEventListener("click", function () {
    var segundo = document.getElementById("segundo"); addmessage(segundo, "Evento:
    clique");
  });
</script>
```

← Registra um manipulador para a mousemove evento

← Registra um manipulador para o evento de clique

Este exemplo usa o integrado `addEventListener` método em um elemento HTML para especificar o tipo de evento (`mousemove` ou clique) e a função de manipulador de eventos. Isso significa que sempre que o mouse é movido sobre a página, o navegador chama uma função que adiciona uma mensagem, "Evento: mousemove ", para o elemento da lista com o ID segundo (uma mensagem semelhante, "Evento: clique em ", é adicionado ao mesmo elemento sempre que o corpo é clicado).

Agora que você sabe como configurar manipuladores de eventos, vamos relembrar o fluxograma simples que você viu antes e dar uma olhada mais de perto em como os eventos são tratados.