

R: A expressão regular / [^ abc]/ corresponde a uma das strings d, e, f - um personagem que não é *a*, *b*, ou *c* (opção b).

- 6 Qual das seguintes expressões regulares corresponde à string Olá?

R: As opções a, b e c correspondem. / Olá/ corresponde apenas à string exata Olá. /Olá/ corresponde também Olá ou Olá (o segundo eu é opcional). / Olá/, depois do primeiro eu, corresponde a qualquer número da letra eu.

- 7 A expressão regular / (cd) + (de) * / corresponde a qual das seguintes strings? R: As opções a, c, d e f estão corretas. / (cd) + (de) * / corresponde a uma ou mais ocorrências de CD seguido por qualquer número de ocorrências de de.

- 8 Em expressões regulares, podemos expressar alternativas com qual das seguintes? R: Usamos a barra vertical, |, para expressar alternativas em expressões regulares (opção c).

- 9 Na expressão regular / ([0-9]) 2 /, podemos fazer referência ao primeiro dígito correspondente com qual dos seguintes?

UMA: \ 1 (opção d)

- 10 A expressão regular / ([0-5]) 6 \ 1 / corresponderá a qual dos seguintes? R: Na expressão regular / ([0-5]) 6 \ 1 /, o primeiro caractere é um dígito de 0 a 5, o segundo caractere é o dígito 6 e o terceiro caractere é o primeiro dígito correspondente, então ambos 060 e 565 corresponder (opções a e d). A expressão regular / (?: ninja) - (truque)? - \ 1 / corresponderá a qual dos seguintes?

11

R: Na expressão regular / (?: ninja) - (truque)? - \ 1 /, o primeiro grupo

(?: ninja) não é de captura, enquanto o segundo grupo é de captura, o que é opcional (truque)?. Mas se este segundo grupo for encontrado, no final temos uma referência anterior a ele. Portanto, ninja- e truque-truque-ninja corresponder (opções a e c).

- 12 Qual é o resultado de executar " 012675 ".substituir (/ 0-5 / g, " a ")?

R: O código substitui todas as ocorrências de dígitos de 0 a 5 pela letra a, então aaa67a resultados (opção a).

Capítulo 11. Técnicas de modularização de código

- 1 Qual mecanismo habilita variáveis de módulo privadas no padrão de módulo?

R: No padrão do módulo, os fechamentos nos permitem ocultar os internos do módulo: Os métodos da API pública do módulo mantêm os internos do módulo vivos (opção b).

- 2 No código a seguir que usa módulos ES6, quais identificadores podem ser acessados se o módulo for importado?

```
const spy = "Yagyu"; comando de
função () {
    retorno geral + "ordena que você faça guerra!";
}
export const general = "Minamoto";
```

R: De fora do módulo, podemos acessar apenas o em geral identificador, porque é o único identificador que foi explicitamente exportado (opção c).

- 3 No código a seguir que usa módulos ES6, quais identificadores podem ser acessados quando o módulo é importado?

```
const ninja = "Yagyu"; comando de função
() {
    retorno geral + "ordena que você faça guerra!";
}
const general = "Minamoto";

exportar {ninja como espião};
```

R: De fora do módulo, podemos acessar apenas o espião identificador: este é o único identificador que foi exportado como um alias do ninja variável (opção a).

- 4 Quais das seguintes importações são permitidas?

```
// Arquivo: staff.js
const ninja = "Yagyu"; comando de função
() {
    retorno geral + "ordena que você faça guerra!";
}
const general = "Minamoto";

exportar {ninja como espião};
```

R: A primeira importação não é permitida porque o pessoal módulo não exporta ninja e em geral identificadores (opção a). A segunda importação é permitida porque importamos todo o módulo, que é acessível através do objeto Pessoal (opção b). A terceira importação também é permitida, pois importamos o exportado espião identificador (opção c).

- 5 Se tivermos o seguinte código de módulo, qual instrução importará o Ninja aula?

```
//Ninja.js
exportar classe padrão Ninja {
    skulk () {return "skulking"; }
}
```

R: A primeira importação é permitida: Importamos a exportação padrão (opção a). A segunda importação é permitida: Importamos todo o módulo (opção b). A terceira importação não é permitida porque não está sintaticamente correta (após o * deve vir a parte "as Name") (opção c).

Capítulo 12. Trabalhando o DOM

- 1 No código a seguir, qual das afirmações a seguir será aprovada?

```
<div id = "samurai"> </div>
<script>
    elemento const = document.querySelector ("# samurai");
```

```

assert (element.id === "samurai", "id da propriedade é samurai"); assert (element.getAttribute ("id") === "samurai",
    "id do atributo é samurai");

element.id = "novoSamurai";

assert (element.id === "newSamurai", "id da propriedade é newSamurai"); assert (element.getAttribute ("id") ===
    "newSamurai",
    "id do atributo é newSamurai");
</script>

```

R: Neste código, todas as afirmações são aprovadas. O eu ia atributo e o eu ia propriedade estão vinculadas; uma mudança em um deles se reflete no outro.

2 Dado o código a seguir, como podemos acessar o elemento largura da borda estilo

propriedade?

```

<div id = "element" style = "border-width: 1px;
                                estilo de borda: sólido; border-color: red ">

</div>
<script>
    elemento const = document.querySelector ("# elemento"); </script>

```

R: O expressão `element.border-width` não faz muito sentido. Ele calcula a diferença entre `element.border` e uma variável largura, o que definitivamente não é algo que queremos. A próxima opção, `element.getAttribute ("border-width")` ;; busca o atributo do elemento HTML, e não a propriedade de estilo. Finalmente, as duas últimas opções fornecem o valor de 1px (opções ce d).

3 Qual método integrado pode ter todos os estilos aplicados a um determinado elemento (estilos fornecidos pelo navegador, estilos aplicados por meio de folhas de estilo e propriedades definidas por meio do atributo style)?

R: Apenas a última opção, `getComputedStyle`, é um método integrado que pode ser usado para obter o estilo calculado de um determinado elemento HTML (opção c). Os outros três métodos não estão incluídos na API padrão.

4 Quando ocorre a eliminação do layout?

R: O descarte do layout ocorre quando nosso código executa uma série de leituras e gravações consecutivas no DOM, forçando sempre o navegador a recalculas as informações do layout. Isso leva a aplicativos da web mais lentos e menos responsivos.

Capítulo 13. Eventos de sobrevivência

1 Por que é importante que a adição de tarefas à fila de tarefas aconteça fora do loop de eventos?

R: Se o processo de adicionar tarefas à fila de tarefas fizer parte do loop de eventos, quaisquer eventos que ocorram enquanto o código JavaScript estiver sendo executado serão ignorados. Isso definitivamente seria uma má ideia.

- 2 Por que é importante que cada iteração do loop de eventos não leve muito mais do que cerca de 16 ms?

R: Para obter aplicativos de execução suave, o navegador tenta realizar a renderização cerca de 60 vezes por segundo. Como a renderização é realizada no final do loop de eventos, cada iteração não deve durar muito mais do que 16 ms, a menos que queiramos criar aplicativos lentos e irregulares.

- 3 Qual é o resultado da execução do código a seguir por 2 segundos?

```
setTimeout (function () {
    console.log ("Tempo limite");
}, 1000);

setInterval (function () {
    console.log ("Intervalo");
}, 500);
```

UMA: Intervalo Tempo Limite Intervalo Intervalo Intervalo (opção b). O definir-Intervalo método chama o manipulador com pelo menos o atraso fixo entre cada chamada, até que o intervalo seja explicitamente limpo. O setTimeout método, por outro lado, chama o retorno de chamada apenas uma vez, após o atraso especificado ter decorrido. Neste exemplo, primeiro o setInterval o retorno de chamada é disparado uma vez após 500 ms. Então o setTimeout retorno de chamada é invocado após 1000 ms, e outro setInterval

imediatamente depois. Nosso exame termina com mais dois setInterval invocações de retorno de chamada, uma a 1.500 ms e a outra a 2.000 ms.

- 4 Qual é o resultado da execução do código a seguir por 2 segundos?

```
const timeoutId = setTimeout (function () {
    console.log ("Tempo limite");
}, 1000);

setInterval (function () {
    console.log ("Intervalo");
}, 500);

clearTimeout (timeoutId);
```

UMA: Interval Interval Interval Interval (opção c). O setTimeout chamar-volta é liberada antes de ter a chance de disparar, portanto, neste caso, temos apenas quatro execuções do setInterval ligar de volta.

- 5 Qual é o resultado de executar o código a seguir e clicar no elemento com o ID interno?

```
<body>
  <div id = "outer">
    <div id = "inner"> </div>
  </div>
</script>

const innerElement = document.querySelector ("# inner"); const outerElement = document.querySelector ("# outer");
```

```

const bodyElement = document.querySelector ("body"); innerElement.addEventListener ("click",
function () {
    console.log ("Interno");
});

outerElement.addEventListener ("click", function () {
    console.log ("Externo");
}, verdade);

bodyElement.addEventListener ("click", function () {
    console.log ("Corpo");
});
<script>
</body>

```

UMA: Corpo Interno Externo (opção c). Os manipuladores de clique no innerElement e a bodyElement são registrados no modo bubbling, enquanto o manipulador de cliques no outerElement é registrado no modo de captura. Ao processar o evento, ele primeiro desce do topo e chama todos os manipuladores de eventos no modo de captura. A primeira mensagem será Exterior. Depois que a meta do evento é alcançada, no nosso caso, o elemento com ID interno, o borbulhar do evento acontece, e o evento borbulha. Portanto, a segunda mensagem será Interno, e o terceiro será Corpo.

Capítulo 14. Desenvolvimento de estratégias para vários navegadores

- 1 O que devemos levar em consideração ao decidir quais navegadores oferecer suporte? R: Ao decidir quais navegadores oferecer suporte, devemos pelo menos levar em consideração o seguinte:

- As expectativas e necessidades do público-alvo
- A participação de mercado do navegador
- A quantidade de esforço necessária para suportar o navegador Explique o

- 2 problema de IDs gananciosos.

R: Ao trabalhar com elementos de formulário, o navegador adiciona propriedades ao elemento de formulário para cada elemento descendente com um ID, para que possamos acessar esses elementos facilmente por meio do elemento de formulário. Infelizmente, isso pode substituir algumas das propriedades integradas do formulário, como ação ou enviar.

- 3 O que é detecção de recurso?

R: A detecção de recurso funciona determinando se um determinado objeto ou propriedade de objeto existe e, em caso afirmativo, presume que ele fornece a funcionalidade implícita. Em vez de testar se o usuário está usando um navegador específico e, em seguida, implementar soluções alternativas com base nessas informações, testamos se um determinado recurso funciona como deveria.

- 4 O que é um polyfill do navegador?

R: Se quisermos usar uma determinada funcionalidade que não é compatível com todos os navegadores-alvo, podemos usar a detecção de recursos. Se um navegador atual não suportar uma determinada funcionalidade, fornecemos nossa própria implementação, e isso é chamado de polyfilling.

Símbolos

`Character 388
 ^ Caracteres 263-264
 ! personagem 255
 ? personagem 265
 . personagem 263, 277
 () operadora 68
 * personagem 129, 264, 266 \
 personagem 262, 264,
 269, 316-317
 + operador 266, 273
 <operador 241
 => operador 50
 > operador 241
 >>> operadora 380
 | character 266, 427 \$
 character 263-264
 \$ {} sintaxe 388

UMA

acesso a objetos, controlando respostas
 de exercícios 422-424
 visão geral 210-213
 propriedades, acesso com getters e
 setters 200-210 definindo getters
 e
 setters 202-207
 usando getters e setters para
 definir computado
 propriedades 208-210
 usando getters e setters para
 validar propriedade
 valores 207-208

usando proxies
 para registro 214-215
 para medir
 desempenho 215-217
 visão geral 210-223
 custos de desempenho 220-223
 preencher automaticamente
 propriedades 217-218
 para implementar matriz negativa
 índices 218-220
 método de acesso 95-96
 atributo de ação 375
 Adicionar ponto de interrupção condicional
 opção 401
 método addEventListener
 26, 353-354, 356-357, 359,
 376-377, 411
 métodos avançados de matriz 5 agregando
 itens de matriz 241-242 método alert () 9

apelido
 objetos de argumentos como alias para
 parâmetros de função 65
 evitando 66-67
 propriedades de amálgama 322
 AMD (Módulo Assíncrono
 Definição) 291-292
 funções anônimas 39
 anti-recurso 374
 Apache Cordova 11
 APIs
 incongruente 383
 teste de desempenho 383
 aplicativos, para embutidos
 dispositivos 11
 aplique a armadilha 213, 216-217

método apply ()
 forçando o contexto da função em
 callbacks 83
 invocando funções com
 77-83
 argumentos
 parâmetros de função e
 52-60
 parâmetros padrão e
 55-60
 parâmetros de descanso e visão geral
 54-55 52
 fatiar 54
 parâmetros de argumentos 62-67
 como alias para parâmetros
 de função 65
 evitando aliases 66-67
 visão geral 413
 argument.length propriedade 63
 literais de matriz 225-226
 Objeto de matriz 225-226, 243
 Array.prototype.find
 método 379
 Array.prototype.push ()
 método 244
 matrizes 225-244
 adicionar e remover itens em qualquer
 localização da matriz 230-232 adicionar e
 remover itens em qualquer extremidade de
 227-229
 agregando itens da matriz
 241-242
 criando 225-227
 iterando mais de 232-233
 mapeamento 233-235

matrizes (*continua*)

- reutilizando array embutido
- funções 242-244**
- pesquisando 237-240
- classificação 39, 240-241
- teste de itens de matriz 235-236
- funções de seta 34, 44, 50-52, 72, 83-86, 160
- como palavra-chave 298-300
- método `assert ()` 9-10, 37-38, 108, 404-405
- método de asserção, teste
- código 404-405
- suposições, cross-browser
- estratégias 383-385
- asterisco 129
- função assíncrona 159-162
- eventos assíncronos 25
- tratamento assíncrono 152
- Módulo Assíncrono
- Definição. *Ver* AMD
- método `attachEvent` 376
- atributos, DOM 313-315
- módulos de aumento 287-290
- propriedades de preenchimento automático
- visão geral 200, 214
- usando proxies para 217-218 `await`
- palavra-chave 161

B

Babel 7

- caractere barra invertida 264
- backticks 388
- melhores práticas 8-10
- depuração 9**
- análise de desempenho 10
- testando 9-10
- método `bind ()`**
- corrigir o problema dos contextos de
- função com 86-90
- visão geral 83
- variáveis com escopo de bloco, usando `let`
- e `const` palavras-chave para
- especificar 111-113
- Propriedade booleana 176
- border-properties 322**
- propriedade `borderWidth` 317, 429 pontos de
- interrupção
- condicional 401-402
- definido 398-399
- navegador
- APIs 8, 20
- correções de bugs 371-372
- compatibilidade 8

visão geral 7-8

Veja também problemas de navegador não

- testáveis
- eventos do navegador 25
- modo borbulhante 431
- fase de borbulhamento 356
- página de construção em tempo de execução,
- exercícios de resposta 411
- funções de array integradas,
- reutilizando 242-244
- `<button>` elemento 79
- objeto de botão 78-79, 84-87 método
- 78 de `button.click ()`

C

- pilha de chamadas 100, 102
- método de `call ()` 68, 79-80, 82-83, 244
- funções de retorno de chamada
- visão geral 36-38
- classificação com comparador
- 39-40
- parâmetro de retorno de chamada 38
- callbacks**
- conceito de 36
- forçando o contexto da função
- em 83
- visão geral 160
- problemas com 147-149
- usando fechamentos com 96-99
- capturas 266
- capturando fase 356, 431 caractere
- circunflexo 263
- Folhas de estilo em cascata. *Ver* Bloco de captura
- de CSS 128, 138
- método `catch` 152-153, 156, 160 `catch`
- declaração 108
- encadeando promessas 155-156
- operador de classe de caracteres 263
- verificar a função 113
- elementos sem filhos 308
- Chrome DevTools 9, 102, 396-397
- classe palavra-chave 168, 190-191 classes,
- usando em ES6 190-197 implementação
- herança 193-197
- usando a palavra-chave `class` 190-193
- `className` argumento 269
- método `clearInterval`
- 122, 344-345
- Método `clearTimeout` 344
- função de clique 16, 26, 28, 78, 85

clientHeight propriedade 329

- `clientLeft` propriedade 329
- propriedade 329 do `clientTop`
- propriedade `clientWidth` 328-329
- operação `cloneNode` 305**
- acoplamento próximo 361
- fechamentos
- exemplo 122-125**
- respostas do exercício 416-417
- manter o controle de identificadores com
- léxico
- ambientes 103-106**
- visão geral 92-95
- variáveis privadas,
- imitando 95-96, 117-121
- monitorar a execução do código com
- contextos de execução 99-102
- usando com `callbacks` 96-99
- modularização de código
- técnicas. *Ver* módulos
- aninhamento de código 103-106
- `<col>` elemento 309
- `<colgroup>` elemento 309
- coleções**
- matrizes 225-244
- adicionar e remover itens**
- em qualquer matriz
- localização 230-232
- adicionar e remover itens**
- em qualquer extremidade
- de 227-229
- agregando matriz**
- itens 241-242
- criando 225-227**
- iterando mais de 232-233
- mapeamento 233-235
- reutilizando array embutido
- funções 242-244**
- pesquisando 237-240
- classificação 240-241
- testando itens de matriz 235-236 respostas
- de exercícios 424-426
- mapas 244-251
- criando 247-250**
- iterando mais de 250-251
- objetos como, não use
- 245-247**
- conjuntos 251-256
- criando 252-253**
- diferença de 255-256
- intersecção de 255
- união de 253-254
- propriedade de cor 317, 321
- CommonJS, módulos 292-293

comunicando, com
 geradores 136-139
 enviando valores como argumentos da
 função geradora
 136-137
 lançando exceções 138-139
 usando o próximo método para enviar valores
 para o gerador 137-138 comparador,
 classificando o retorno de chamada
 funções com 39-40
 computacionalmente caro
 processamento, loop de evento
 350-353
 estilos computados
 DOM 319-322
 visão geral 315
 variável `computedStyles` 321
 pontos de interrupção condicionais
 401-402
 chave configurável 185
 objeto de console 10
 método `console.assert()` 405
`console.log` método 103, 397
`console.time` método 10, 217
`console.timeEnd` método 10
`const` palavra-chave, especificando
 variáveis com escopo de bloco
 com 111-113
 variáveis `const` 107-109
 construir armadilha 213
 casos de teste construtivo 402 propriedade
 do construtor 173, 177,
 179-181, 183-184, 186, 193,
 246, 420-421
 construtores
 invocar funções como 72-77
 considerações de codificação
 para 76-77
 considerações para 76-77
 visão geral 72-74, 76
 valores de retorno 74-76
 digitação de objetos por meio da visão
 geral 179-181 172, 245
 contexto, para as funções 80, 83
`createElement` 309
 Cria o objeto 70
 Crockford, Douglas 287
 estratégias para vários navegadores
 correções de bug do navegador 371-372
 exercícios 385
 código externo e
 marcação 373-376
 Regras CSS 376
 DOM derrotando 374-375

encapsulamento 373-374
 trabalhando com existente
 código 374
 detecção de recurso 379-381
 corrige 378-379
 visão geral 8, 368-370
 polyfills 379-381
 reduzindo suposições
 383-385
 regressões 376-377
 navegador não testável
 edições 381-383
 Desempenho da API 383
 o navegador trava 382
 Efeitos de propriedade CSS 382 disparo
 de eventos 382
 ligações de manipulador de eventos 381
 APIs incongruentes 383
 Visão geral de CSS (Cascading Style
 Sheets) 306
 efeitos de propriedade 382
 regras 376
 eventos personalizados 360-364
 Ajax 361-362
 criando 362-364
 acoplamento solto 361

D

travessão 275
 Painel do depurador 395, código de
 depuração 399 397-402
 pontos de interrupção 398-399
 pontos de interrupção condicionais
 401-402
 registro 397-398
 entrando em função
 399-400
 declarações (função) 45-46
 casos de teste destrutivos 402
 exportações padrão, ES6
 módulos 297-298
 parâmetros padrão
 (função) 55-60
 definir função 291
 definição de interfaces de módulo 284 método de
 exclusão 249
 aplicativos de desktop 11
 desestruturação 387, 389-390
 Ferramentas do desenvolvedor, Firefox
 9, 393-395
 DevTools 9, 102, 396-397
 dicionários. *Ver* mapas
 difícil tratamento de erros 148

propriedade de exibição 323
 itens distintos 251
`<div>` elemento 268, 309, 314,
 316, 325
`doc.createDocumentFrag-`
 método de `ment()` 312
 manipulador de clique de documento
 355, 357
 objeto de documento 20, 22 Modelo de
 objeto de documento. *Ver*
 DOM
`document.getElementById`
 método 246
 Dojo toolkit 291
 caractere cifrão 264 DOM
 (objeto de documento
 Modelo) 305-331
 atributos e
 propriedades 313-315
 edifício 18-20
 estratégias para vários navegadores
 374-375
 respostas do exercício 428-429
 injetar HTML em 306-313 HTML
 wrapping 308-311
 inserir elementos em
 documento 311-313
 visão geral 307-311
 pré-processamento de HTML
 string de origem 307-308
 layout thrashing 327-331
 visão geral 7, 17
 propagando eventos
 até 354-360
 atributos de estilo 315-326
 estilos calculados 319-322
 convertendo pixel
 valores 322-323
 altura e largura
 propriedades 323
 nomeando 318-319
 visão geral 315-317
 atravessando usando
 geradores 135-136
 Painel do Explorador DOM 395
 fragmentos DOM 306, método 311
 DOM 313-315
 Especificação DOM 19
`DOMContentLoaded`
 manipulador 314
 feito propriedade 132
 caractere barra invertida dupla
 262, 264
 colchetes duplos 104

caractere de escape duplo 269
 natureza dinâmica do JavaScript,
 efeitos colaterais de 176-178

E

ECMAScript 6. *Ver* ES6 elem
 variável 98
 elipses 54, 309
 valores embutidos 269
 encapsulamento 373-374
 literais de objeto aprimorados
 387, 390-391
 chave enumerável 185
 enumerar armadilha 213
 Propriedade [[Ambiente]]
 106, 118
 erros, travando em cadeia
 promete 156
 ES6 (ECMAScript 6)
 desestruturação 389-390
 literais de objeto aprimorados
 390-391
 exportando e importando
 funcionalidade 294-302
 exportações padrão 297-298
 renomeando exportações e
 importações 298-302
 literais de modelo 387-388
 usando classes em 190-197
 implementação
 herança 193-197
 usando a palavra-chave *class*
 190–193
 caracteres de escape 278
 evento borbulhando 355-356,
 359-360, 364
 captura de eventos 355-356, 364 tratamento
 de eventos
 visão geral 14-15
 construção de página 23-29
 loop de eventos 152, 333, 335-344
 microtarefas 336-344
 microtarefas 339-344
 temporizadores 344-353
 computacionalmente caro
 processamento 350-353
 visão geral 345-350
 tempo limite e intervalos 350 fila de
 eventos 14, 24-25, 28 aplicativos orientados a
 eventos 8
 registro de manipulador de eventos 26
 propriedade *event.target* 359
 eventos
 eventos personalizados 360-364

Ajax 361-362
 criando 362-364
 acoplamento solto 361
 evento disparando 382
 associações do manipulador de eventos 381
 respostas do exercício 429-431
 exercícios 364-365
 propagando através de
 DOM 354-360
 Veja também loop de eventos
 a cada método 235
 exceções, jogando 138-139
 ponto de exclamação 255
 método *exec ()* 271, 275
 executando o estado 139
 execução 267
 contextos de execução
 execução do código de rastreamento
 com 99-102
 geradores de rastreamento
 usando 141-146
 função do executor 147, 151-152 respostas
 do exercício 411-431
 acesso a objetos,
 controlando 422-424
 página de construção na modularização do
 código 411 em tempo de execução
 técnicas 427-428
 coleções 424-426
 estratégias para vários navegadores 431
 DOM 428-429
 eventos 429-431
 funções
 fechamentos e escopos
 416-417
 definições e
 argumentos 411-413
 geradores e
 promessas 417-419
 invocação 413-416
 protótipos, orientação a objetos
 com 419-422
 expressões regulares 426-427
 esperar função 407
 rejeitando explicitamente as promessas 152 exportar
 palavra-chave 294–295, 297–
 298
 exporta propriedade 292
 exportações, renomeando 298-302
 expressões (função) 46-48
 estende a palavra-chave 194-195
 código externo e marcação,
 navegador cruzado
 estratégias 373-376
 Regras CSS 376

DOM derrotando 374-375
 encapsulamento 373-374
 trabalhando com existente
 código 374

F

F12 Developer Tools 9, 395 reprovam
 classe 405
 falha na função 108
 falha de retorno de chamada 147
 FastDom 329
 operador *fat-arrow* 50-51
 detecção de recursos, navegador cruzado
 estratégias 379-381
 método de filtro 238-239, 255, método de
 localização 237-238, 240, 244,
 379-380
 método *findIndex* 240
 Firebug 9, 393
 Ferramentas de desenvolvedor do Firefox
 9, 393-395
 objetos de primeira classe 5, 34-35, 38,
 44, 58, 160
 Fitzgerald, Michael 261
 para loop 10, 104, 110,
 112, 232–233
 para declaração 81
 loop *for-in* 185-186
 declarações *for-in* 213
 loop *for-of* 129-130, 132-133,
 136, 163, 418
 método *forEach ()* 81-82,
 233-234
<form> elemento 375
 parâmetro de fragmento 311
 Friedl, Jeffrey 261
 estado cumprido 150
 encerramentos de funções 5
 código de função 21, 99
 construtores de função 44, 72
 contextos de função
 corrigindo o problema de 83-90 usando o
 método de ligação 86-90 com as funções de
 seta 83-86 forçando em retornos de chamada
 80-83 visão geral 67, 100
 declarações de função 44
 contexto de execução de função 99
 expressões de função 44, 47 palavra-chave
 de função 45, 50,
 128-130, 162
 literal de função 44
 declaração de função *()* 49

programação funcional 36

funções

argumentos 52-60

parâmetros padrão e
55-60parâmetros de descanso e 54-55
fatiamento 54

como objetos de primeira classe 35-36

como módulos 284-285

como objetos 40-44

auto-memorável

funções 42-44

armazenamento de funções 40-42

callback 36-40

chamando funções antes de suas
declarações 115encerramentos, exercício
respostas 416-417

declarando 44

definindo 44-52

funções de seta 50-52

declarações de função
45-46

expressões de função 46-48

funções imediatas 48-50

corrigindo o problema dos contextos de
função 83-90

usando o método de ligação 86-90 com

geradores de funções de seta 83-86,

exercício

respostas 417-419

função implícita

parâmetros 62-67

parâmetro de argumentos
62-67

contexto de função 67

importância de nas funções

anônimas do JavaScript 39

conceito de callback 36

invocando 67, 72, 77-83 como

construtores 72-77

como métodos 69-72

considerações para 76-77

respostas do exercício 413-416

visão geral 68-69, 72, 76 com

aplicar e ligar

métodos 77-83

substituindo 115-116

parâmetros para 53, 67

promessas, exercício

respostas 417-419

escopos, respostas de exercícios
416-417

auto-lembrando 42

simulando métodos de array
com 242entrando em função
399-400

armazenando 42

G

g bandeira 262, 271

funções do gerador 44

geradores 129-146

combinando com

promete 158-164

comunicando-se com 136-139

enviando valores como gerador

função

argumentos 136-137

lançando exceções

138-139

usando o próximo método para enviar

valores em

gerador 137-138

controlando por meio de iterador

objeto 130-133

iterador de iteração 132

cedendo a outro

gerador 133

respostas do exercício 417-419

tornando o código assíncrono elegante com
127-129

visão geral 5

rastreamento usando execução

contextos 141-146

usando para gerar IDs

133-135

usando para atravessar DOM

135-136

obter a palavra-chave 185, 203

get trap 211-213, 215, 218, 220

getAttribute () método 313-315

getBoundingClientRect

propriedade 329

Método getComputedStyle

329, 429

Método getElementById

16, 21-22

getElementsByTagName

método 268

getPrototypeOf 213

função getter 185, 203 getters

e setters

acesso a propriedades

usando 200-210

definindo getters e

setters 202-207

usando getters e setters para
definir computado

propriedades 208-210

usando getters e setters para
validar propriedade

valores 207-208

visão geral 96

código global 21, 99

Goyvaerts, janeiro 261

suporte de navegador graduado 368

maior do que operador 241

operadores gananciosos 265

H

tratamento de eventos 25

tem o método 248-249, elemento

principal 253 18-19

propriedade de altura, DOM 323-326 internos

do módulo de ocultação 284 içamento 116

HTML

injetar no DOM 306-313 HTML

wrapping 308-311

inserir elementos em

documento 311-313

visão geral 307-311

pré-processamento de HTML

string de origem 307-308

análise de 18-20

elemento html 18, 372

painel HTML 393

eu

eu sinalizo 262

i variável 110-111

id atributo 314, 374, 429 id

valores 41, 134, 278 identificador

resolução 102

identificadores

registrando-se no léxico

ambientes 113-116

chamando funções antes

declarações 115

substituindo funções

115-116

processo 113-115

rastreamento com léxico

ambientes 103-106

IDEs (desenvolvimento integrado

ambientes) 393

IDs, gerando usando

geradores 133-135

IE (Internet Explorer) 395 IIFE
(invocado imediatamente
expressão de função) 49
funções imediatas 48-50
função implícita
parâmetros 62-67
parâmetros de argumentos 62-67
contexto de função 67
parâmetros implícitos 62
importar palavra-chave 294-296
anotação importante 317
importações, renomeando 298-302
método `indexOf` 239
herança 169
`input` # `action` element 375
propriedades da instância 173-176
operador instância
180, 187–190, 213, 421
ambiente de desenvolvimento integrado
ronments. *Ver* IDEs
interfaces, módulo 285-287
Internet Explorer. *Ver* Interseção do IE,
de conjuntos de intervalos de 255,
invocação de loop de evento 350,
exercício
respostas 413-416
invocando funções 67-83
como método 69-72
construtor 72-77
considerações de codificação
para 76-77
visão geral 72-74
valores de retorno 74-76
visão geral 68-69
com métodos de aplicação e
chamada 77-83
iterando sobre mapas 250-251 objeto
iterador, controlando
eradores até 130-133
iterador de iteração 132
cedendo a outro
gerador 133

J

Estrutura de teste Jasmine
407-409
JavaScript
melhores práticas 8-10
depuração 9
análise de desempenho 10
testando 9-10
aumentando a habilidade
transferibilidade 10-12

navegador 7-8
características do idioma
evolução de 6
visão geral 4-5
transpiladores 6-7
função `jQuery` 373
Método `JSON.parse` 155
método `juggle` () 79

K

método de chaves 251
Kleene, Stephen 261
objeto kuma 169-170

eu

funções lambda 44
layout thrashing, parâmetro de comprimento 64
do DOM 327-331
propriedade de comprimento 63, 225-227,
243-244, 380
menor que o operador 241
deixe a palavra-chave
visão geral 417
especificando com escopo de bloco
variáveis com 111-113
Levithan, Steven 261
ambientes lexicais
manter o controle de identificadores com
103-106
registrando identificadores
dentro de 113-116
chamando funções antes
declarações 115
substituindo funções
115-116
processo 113-115
 elemento 23, 405
ciclo de vida, construção de página 14-17
propriedade `location.href` 249–250
exploração madeireira
visão geral 397-398
usando proxies para 214-215
acoplamento solto 361

M

m bandeira 262
macrotarefas
loop de evento 336-344
visão geral 333-334
função `makeLoggable` 215

Construtor de mapa 248
função de mapa 234
matrizes de mapeamento 233-235
mapas 244–251
criando 247-250
iterando mais de 250-251
objetos como, não use 245-247 visão
geral 6
método `match` () 270-272
segmentos correspondentes, capturando
grupos de não captura 273
executando simples
captura 269
referenciando capturas 272
MDN (Mozilla Developer
Rede) 379
medição de desempenho, proxies
para 215-217
memoização 40
métodos, invocando funções
como 69-72
microtarefas
loop de eventos 339-344
visão geral 334, 336
minimizando suposições 383
aplicativos móveis, com
frameworks 11
parâmetro do módulo 288
padrão de módulo 286-287,
289-291, 293, 295
variável de módulo
286, 292-293, 295
`module.exports` object 293, 295 modules

ES6 294-302
respostas do exercício 427-428
no módulo pré-ES6 JavaScript 283-293
Asynchronous
Definição 291-292
módulos de aumento
287-290
CommonJS 292-293
funciona como 284-285
interfaces de módulo 285-287
evento `mouseenter` 382
evento `mouseleave` 382
evento `mousemove` 15-16,
27-28, 39
evento `mouseout` 382
evento `mouseover` 382
Mozilla Developer Network. *Ver*
MDN
cordas multilinha 387

N

exportação nomeada 296, índices de matriz negativa 298–299,
 implementando usando
 proxies 218–220
 eventos de rede 25
 novo operador 171–173
 novas linhas, correspondendo a 277
 próximo método 130–134, 136–138,
 140, 143–144, 158–160
 código de retorno de chamada não bloqueante 129
 geradores não bloqueantes 129
 grupos de não captura 273–274
 elemento não exibido 323
 variáveis não exportadas 296
 operadores não agudos 265
 variáveis não importadas 296
 modo não estrito 67–69, 205, 414

O

literais de objeto
 funções de seta e 85–86
 aprimorado 390–391
 métodos de objeto 192
 orientação do objeto, usando
 protótipos 419–422
 Object.defineProperty
 método 185–186, 193, 202,
 205–206
 Object.setPrototypeOf
 método 170
 objetos
 acesso, controlando 422–424 como
 mapas, não use construção de
 245–247, protótipos
 e 171–181
 propriedades da instância
 173–176
 digitação de objetos via
 construtores 179–181
 efeitos colaterais da dinâmica
 natureza de
 JavaScript 176–178
 funciona como 40–44
 auto-memorável
 funções 42–44
 armazenamento de funções 40–42
 propriedade offsetHeight 323, 329
 propriedade offsetLeft 329
 propriedade offsetParent 329
 propriedade offsetTop 329
 propriedade offsetWidth 323, 329

propriedade offsetX 329
 propriedade offsetY 329
 evento onerror 154
 onload evento 154
 onload propriedade 26
 <option> elemento 307–308
 exterior (pai) léxico
 ambiente 104
 substituindo funções 115–116

P

construção de página
 construção DOM 18–20
 tratamento de eventos 23–29
 exercícios 29
 Código JavaScript
 executando 21–23
 objetos globais 20
 tipos de 21
 ciclo de vida 14–17
 visão geral 14–15
 analisar HTML 18–20
 parâmetros, para as funções 53, 67
 método parseFloat 323
 subexpressão passiva
 273–274, 277
 variável padrão 262
 padrões 261
 estado pendente 149, 151–152
 desempenho
 medindo proxies para
 215–217
 visão geral 306
 caractere ponto 263, caractere 277
 barra vertical 266, valores de 427
 pixels, polyfills DOM 322–323 379–381
 método pop 227, 229
 propriedade de posição 323
 pré-compilar regular
 expressões 269
 pré-construção regular
 expressões 269
 personagem predefinido
 classes 265
 hash anterior 326
 valor principal 43
 variáveis privadas, imitando
 95–96
 Construtor de promessa
 146–147, 151
 Método Promise.all 157
 Método Promise.race 157

Método Promise.resolve () 342

promessas
 encadeamento 155–156
 combinando com
 geradores 158–164
 criando 154–155
 respostas do exercício 417–419
 tornando o código assíncrono elegante com
 127–129
 visão geral 149–152
 rejeitando 152–153
 esperando pelo número de
 156–158
 propriedades
 acesso a com getters e setters
 200–210
 definindo getters e
 setters 202–207
 usando getters e setters para
 definir computado
 propriedades 208–210
 usando getters e setters para
 validar propriedade
 valores 207–208
 DOM 313–315
 descritor de propriedade 185
 cadeia de protótipo 170, 175, 182
 métodos de protótipo 192
 propriedade de protótipo
 170, 173–174, 183
 objeto baseado em protótipo
 orientação 5
 protótipos
 alcançando herança
 181–190
 operador instância
 187–190
 problema de anular
 constructor
 propriedade 184–187
 instanciando usando referência
 para o construtor 179
 construção de objetos e
 171–181
 propriedades da instância
 173–176
 digitação de objetos via
 construtores 179–181
 efeitos colaterais da dinâmica
 natureza de
 JavaScript 176–178
 orientação de objeto usando
 417–422
 visão geral 168–171

protótipos (*continua*)

usando classes JavaScript no ES6

190-197

implementando

herança 193-197

usando a palavra-chave class

190-193

procuradores

custos de desempenho de 220-223 usando

para perfilagem 214-215 usando para

medição

desempenho 215-217

usando para preencher automaticamente

propriedades 217-218

usando para implementar índices de matriz

negativos 218-220

Construtor de proxy 210-211, 213

método push 227-229, 244

Q

Estrutura de teste QUnit

406-407

R

reconciliando propriedade

referências 178

reduzir método 242

referenciando capturas, capturando

segmentos correspondentes 272

Método RegExp () 262, regressões

269, navegador cruzado

estratégias 376-377

expressões regulares

benefícios de 260

capturando correspondência

segmentos 269-274

correspondência usando global

expressões 271

grupos de não captura 273

executando simples

captura 269

referenciando capturas

272-273

usando global

expressões 272

classes de personagens 263

compilando 267

fim da string 264

exemplos usando

correspondência de novas linhas 277

Unicode correspondente

caracteres 277-278

respostas do exercício 426-427

visão geral 260

substituindo usando funções 274

resolvendo problemas comuns

com

correspondência escapou

personagens 278

correspondência de novas linhas 277

Unicode 277 correspondente

termos e operadores

alternância (ou) 266

referências anteriores 266

começa e termina 264

escapando 263

correspondência exata 263

agrupamento 266

correspondência da classe de

personagens 263

personagem predefinido

classes 265

ocorrências repetidas 264

Expressão regular

construtor 426

rejeitar função 146-147,

150, 152

rejeitando promessas 152-153

operação removeChild 305

ocorrências repetidas, regulares

expressões e 264

método replace () 272, 274,

276, 321

objeto representativo 211

requer função 293

Requer JS 291

função de resolução 146-147,

149, 152

Modo de design responsivo 395 parâmetros

de descanso (função)

54-55

declaração de retorno 50, 52, 130,

140, 146, 312

valores de retorno, construtores

74-76

tempo de execução, página de construção em 411

S

escopos 5, 92, 416-417

elemento de script 17-18, 20, 22-23

propriedade scroll 329

scrollBy property 329

propriedade scrollByLines 329

propriedade scrollByPages 329

propriedade scrollHeight 329

Propriedade scrollIntoView 329

propriedade scrollLeft 329, 372

propriedade scrollTo 329

propriedade scrollTop 329, 372

propriedade scrollWidth 329

propriedade scrollY 329

procurando matrizes 237-240

<select> elemento 308-309

funções de auto-memorização 42-44

aplicativos do lado do servidor 11

definir palavra-chave 202-203

definir operador 263

set trap 211, 213, 215, 220 setAttribute

() método 314-315

método setInterval 98, 122,
344-345, 347, 349, 364, 430

setPrototypeOf 213

conjuntos 251-256

criando 252-253

diferença de 255-256

intersecção de 255

visão geral 251

união de 253-254

função setter 185

método setTimeout 344, 347,
350, 364, 430

método de deslocamento 227, 230

irmãos 18

elementos simples 266

simulando com base em classe

herança 190

execução single-threaded

modelo 24

propriedade de tamanho 249, 253

argumentos de corte 54

método de classificação 39, 50, 240

matrizes de classificação 39, 240-241

 elemento 268

método de emenda 231-232

colchetes 263

pilha 100

entrando na função 399-400 armazenar

variável 41

armazenamento de funções 40-42

modo estrito 66

interpolação de string 387

Objeto string 271, 274, 276 atributo

de estilo 315-317

e estilos computados 322 conversão de

valores de pixel 323 obtendo

propriedades da visão geral 317 315-316

<style> elemento 316

propriedade de estilo 315-317,

319-322, 331

método `style()` 322
atributos de estilo, DOM 315-326 estilos
 computados 319-322
 convertendo valores de pixel 322-323
 altura e largura
 propriedades 323-326
 nomeando 318-319
 visão geral 315-317
retorno de chamada de sucesso 147
super palavra-chave 195
Método da SuperClass 182
Estado de início suspenso 139 Estado de
rendimento suspenso 145

T

objeto alvo 210-211, 213,
 215-216, 219
propriedade alvo 354, 359, 364
elemento `<tbody>` 309
literais de modelo 387-388
casos de teste 402
função de teste 412
independência de teste 402
módulo de teste 283
repetibilidade de teste 402
testar a simplicidade 402
suítes de teste 406
método `test()` 269
testando
 itens de matriz 235-236
 visão geral 9
código de teste 402-410
 método de afirmação 404-405
 criando testes 402-404
 fundamentos de 404-410
 Jasmine 407-409
 medição de cobertura de código 410 QUnit
 406-407
elemento `textarea` 262
o método 146-147, 150-153,
 155, 160, 340
método de lançamento 138-139, 159
exceções de lançamento 138-139
método de tempo 10
tempos limite, loop de evento 350
método `timeEnd` 10
eventos de cronômetro 25
variável do temporizador 98

temporizadores, loop de evento 344-353
 caro computacionalmente
 processamento 350-353
 visão geral 345-350
método `toString` 247
Traceur 7
transformar propriedade 269-270
transpiladores 6-7
armadilhas 211
atravessando o DOM, usando
 geradores 135-136
bloco `try-catch` 138
declarações `try-catch` 103, 147,
 149, 153, 155, 161
atributo de tipo 378, 383 operador
`typeof` 56, 180, 213

U

u sinalizar 262
Capacidade de resposta da IU 395
`ul` element 23
string indefinida 56
Unicode, correspondendo a 277
união, dos conjuntos 253-254
promessa não resolvida 149
método `unshift` 227-230
problemas de navegador não testáveis
 381-383
 Desempenho da API 383
 o navegador trava 382
 Efeitos de propriedade CSS 382 disparo
 de eventos 382
 ligações de manipulador de eventos 381
 APIs incongruentes 383
recursos não testáveis 383
eventos do usuário 25

V

argumento de valor 319
chave de valor 185
métodos de valores 251
valores, enviando para
 gerador 137-138
palavra-chave `var`, variáveis 109-111
içamento variável 116
variáveis 106-116
 variáveis `const` 107-109
visão geral 106

registrando identificadores dentro de
 ambientes lexicais
 113-116
chamando funções antes
 declarações 115
substituindo funções
 115-116
processo 113-115
usando palavras-chave `let` e `const` para
 especificar o escopo do bloco
 variáveis 111-113
usando `var` palavra-chave 109-111
mutabilidade variável 107-109
propriedade de visibilidade 323

C

ferramentas de desenvolvedor da web 393-397
 Chrome DevTools 396-397
 F12 395
 Firebug 393
 Firefox 393-395
 WebKit Inspector 9, 396 WebKit
Inspector 9, 396 while loop 132, 134, 163,
 418 caractere de espaço em branco
 266, 269, 277
propriedade de largura 323-326,
 329, 378
objeto de janela 20, 23, 26,
 68, 414
`window.getComputedStyle()`
 método 319
chave gravável 185

X

Objeto XMLHttpRequest 154

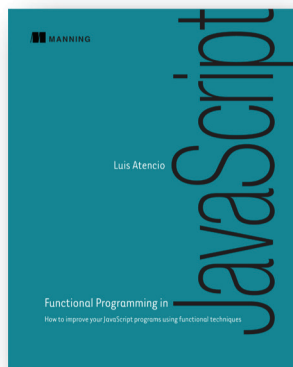
Y

bandeira y 262
expressão de rendimento 131, 137-138,
 140, 146, 160
palavra-chave de rendimento 128-129, 131, 145
rendimento * operador 133

Z

operador zero-fill shift right-shift 380

MAIS TÍTULOS DEMANNING



Programação Funcional em JavaScript

por Luis Atencio

ISBN: 9781617292828

272 páginas

\$ 44,99

Junho de 2016



Design de aplicativo JavaScript

Uma abordagem Build First

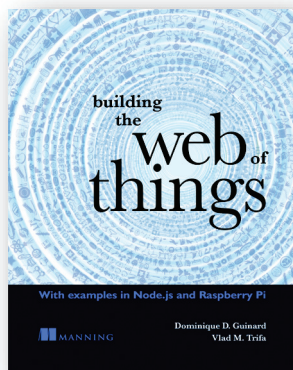
por Nicolas G. Bevacqua

ISBN: 9781617291951

344 páginas

\$ 39,99

Janeiro de 2015



Construindo a Teia das Coisas

Com exemplos em Node.js e Raspberry Pi

por Dominique D. Guinard e Vlad M. Trifa

ISBN: 9781617292682

344 páginas

\$ 34,99

Junho de 2016

Para obter informações sobre pedidos, acesse www.manning.com

Folha de dicas ES6 (continuação)

Aulas agir como um açúcar sintático em torno dos protótipos do JavaScript:

```
aula Pessoa {  
  construtor( nome) {this.name = nome; } dance () {return true; }  
  
}  
  
classe Ninja estende Pessoa {  
  construtor (nome, nível) {  
    super (nome);  
    this.level = level;  
  }  
  
  estático compare (ninja1, ninja2) {  
    retornar ninja1.level - ninja2.level;  
  }  
}
```

Proxies controlar o acesso a outros objetos. Ações personalizadas podem ser executadas quando um objeto é interagido (por exemplo, quando uma propriedade é lida ou uma função é chamada):

```
const p = novo Proxy ( alvo, {  
  obter: ( target, key) => {/ * Chamado quando propriedade acessada por meio de proxy * /},  
  definir: ( destino, chave, valor) => {/ * Chamado quando a propriedade definida por meio de proxy * /}});
```

Mapas são mapeamentos entre uma chave e um valor:

- novo mapa () cria um novo mapa.
- Use o definir método para adicionar um novo mapeamento, o obter método para buscar um mapeamento, o tem método para verificar se existe um mapeamento, e o excluir método para remover um mapeamento.

Jogos são coleções de itens exclusivos:

- novo conjunto() cria um novo conjunto.
- Use o adicionar método para adicionar um novo item, o excluir método para remover um item, e o Tamanho propriedade para verificar o número de itens em um conjunto.

para de os loops iteram sobre coleções e geradores.

Destruição extrai dados de objetos e matrizes:

- **const {nome: ninjaName} = ninja;**
- **const [firstNinja] = [" Yoshi "];**

Módulos são unidades maiores de código de organização que nos permitem dividir programas em clusters:

```
exportar classe Ninja {}; // Exportar um item  
exportação padrão class Ninja {} // Exportar exportação padrão {ninja}; // Exportar  
exportação de variáveis existentes {ninja como samurai}; // Renomear uma  
exportação  
  
importar Ninja de "Ninja.js"; // Importar uma importação de exportação padrão {ninja} de "Ninja.js"; //  
Importar exportações nomeadas  
importar * como Ninja de "Ninja.js"; // Importar todas as exportações nomeadas import {ninja como iNinja} de "Ninja.js"; //  
Importar com um novo nome
```

Segredos do JavaScript Ninja Segunda edição

Resig .Bibeault .Maras



Jtodo tipo de aplicativo, seja na web, no desktop, na nuvem ou em dispositivos móveis. Quando você

Se tornar um profissional de JavaScript, você tem um conjunto de habilidades poderoso que pode ser usado em todos esses domínios.

Segredos do JavaScript Ninja, segunda edição usa prático

exemplos para ilustrar claramente cada conceito e técnica principal. Esta edição completamente revisada mostra como fazer uma chave mestra

Conceitos de JavaScript, como funções, closures, objetos, prototipos e promessas. Abrange APIs como DOM, eventos, e temporizadores. Você descobrirá técnicas de práticas recomendadas, como teste e desenvolvimento em vários navegadores, todas ensinadas da perspectiva de profissionais qualificados de JavaScript.

O que há dentro

- Escrever código mais eficaz com funções, objetos e encerramentos
- Aprendendo a evitar armadilhas de aplicativos JavaScript
- Usando expressões regulares para escrever código de processamento de texto sucinto
- Gerenciando código assíncrono com promessas
- Totalmente revisado para cobrir os conceitos de ES6 e ES7

Você não precisa ser um ninja para ler este livro - apenas esteja disposto a se tornar um. Você está pronto?

John Resig é uma autoridade reconhecida em JavaScript e criadora da biblioteca jQuery. **Bear Bibeault** é um desenvolvedor web e autor da primeira edição, bem como co-autor de *Ajax na prática*, *protótipo* e *Scriptaculous em ação*, e *jQuery em ação*.

Josip Maras é pesquisador pós-doutorado e professor.

“
deve lopo...
técnicas poderosas para
melhore o seu JavaScript.”

- Becky Huett, Big Shovel Labs

Excelente e abrangente
visão sobre a magia das funções e
fechamentos para o
uso eficiente de JavaScript.”

- Gerd Klevesaat, Siemens

“
msuas habilidades essenciais para desenvolver
para o próximo nível.”

- David Starkey, Bluemix

“
Ajuda você a dominar as
técnicas fundamentais e usadas
de JavaScript moderno.”

- Christopher Haupt
New Relic Inc.

Para baixar seu e-book gratuito nos formatos PDF, ePub e Kindle,
os proprietários deste livro devem visitar
manning.com/books/secrets-of-the-javascript-ninja-second-edition

ISBN-13: 978-1-61729-285-9
ISBN-10: 1-61729-285-0



9 781617 292859