

O'REILLY®

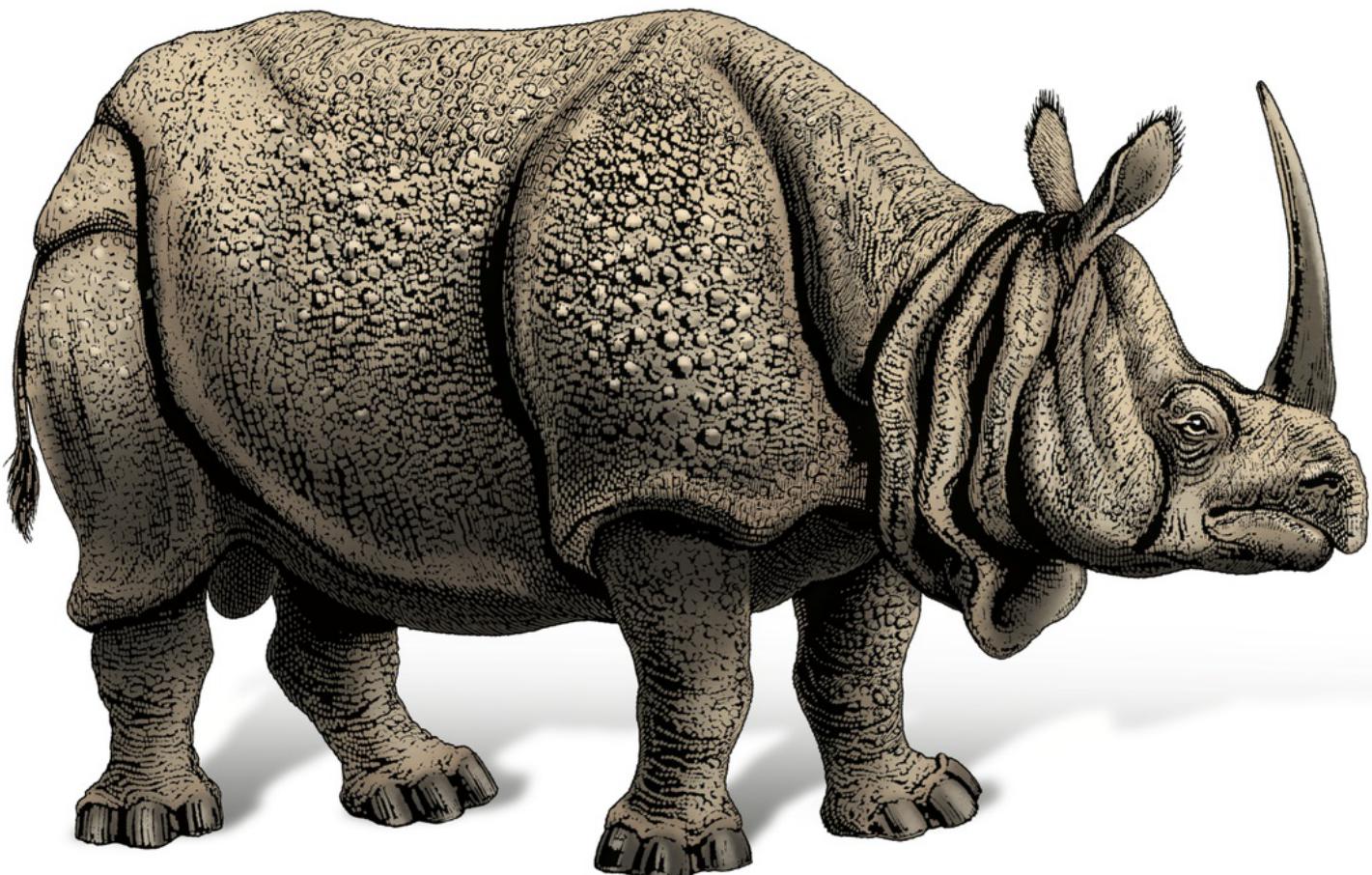
Erro ao traduzir esta página.

Seventh
Edition

JavaScript

The Definitive Guide

Master the World's Most-Used
Programming Language



David Flanagan

1. Prefácio
1.1 Convenções usadas neste livro
1.2 Código de exemplo
1.3 O'Reilly Online Learning
1.4 Como nos contatamos

2. Agradecimentos

3. Introdução ao JavaScript

3.1 Explorando JavaScript

3.2 Hello World

3.3 Um passeio de JavaScript

3.4 Exemplo: histogramas de frequência do personagem

3.5 Resumo

3.6 Estrutura lexical

3.7 O texto de um programa JavaScript

3.8 Comentários

3.9 Literais

3.10 Identificadores e palavras reservadas

3.11 Palavras reservadas

3.12 Unicode

3.13 Sequências de Escape Unicode

3.14 Normalização Unicode

f.2.6 Semicolons opcionaisg.2.7 Resumo4 tipos, valores e variáveissum.3.1 Visão geral e definiçõesb.3.2 númerosseu.3.2.1 literais inteirosii.3.2.2 Literais de ponto flutuanteiii.3.2.3 aritmética em javascript4.3.2.4 ponto flutuante binário eErros de arredondamentov. 3.2.5 inteiros de precisão arbitrária comBigintvi.3.2.6 Datas e horáriosc.3.3 Textoeu.3.3.1 Literais de cordasii.3.3.2 Sequências de fuga em literais de cordasiii.3.3.3 Trabalhando com strings4.3.3.4 Literais de modelov. 3.3.5 correspondência de padrõesd.3.4 valores booleanose.3.5 NULL e indefinidof.3.6 Símbolosg.3.7 O objeto global

h.3.8 valores primitivos imutáveis ??e mutávelReferências de objeto
3.9 Conversões de tipo
3.9.1 Conversões e igualdade
3.9.2 Conversões explícitas
3.9.3 Objeto de conversões primitivas
3.10 Declaração e atribuição variáveis
3.10.1 declarações com Let and Const
3.10.2 declarações variáveis ??com VAR
3.10.3 Atribuição de destruição
ok.3.11 Resumo
5. Expressões e operadores
4.1 Expressões primárias
4.2 Inicializadores de objeto e matriz
c.4.3 Expressões de definição de função
d.4.4 Expressões de acesso à propriedade
de
e.4.4.1 Acesso à propriedade condicional
e.4.5 Expressões de invocação
e.4.5.1 Invocação condicional
f.4.6 Expressões de criação de objetos
g.4.7 Visão geral do operador
e.4.7.1 Número de operandos

ii.4.7.2 Operando e tipo de resultadoiii.4.7.3 Efeitos colaterais do operador4.4.7.4 Precedência do operadorv. 4.7.5
Associatividade do operadorvi.4.7.6 Ordem de avaliaçãoh.4.8 Expressões aritméticaseu.4.8.1 O operador +ii.4.8.2
Operadores aritméticos unáriosiii.4.8.3 Operadores bitwiseeu.4.9 Expressões relacionaiseu.4.9.1 Operadores de
igualdade e desigualdadeii.4.9.2 Operadores de comparaçãoiii.4.9.3 O operador no4.4.9.4 A instância do
operadorj.4.10 Expressões lógicaseu.4.10.1 Lógico e (&&)ii.4.10.2 Lógico ou (||)iii.4.10.3 Lógico não (!)k.4.11
Expressões de atribuiçâoeu.4.11.1 Atribuição com operaçãool.4.12 Expressões de avaliaçãoe.4.12.1 Eval ()

ii.4.12.2 Global Eval ()iii.4.12.3 Eval rigoroso ()m.4.13 Operadores diversos
eu.4.13.1 O operador condicional (?)
:)ii.4.13.2 Primeiro definido (??)iii.4.13.3 O operador TIPEOF4.4.13.4 O operador de exclusão.
4.13.5 O operador aguardarvi.4.13.6 O operador vaziovii.4.13.7 O operador de vírgula (,)n.4.14 Resumo6. declarações
sum.5.1 declarações de expressãob.5.2 declarações compostas e vaziasc.5.3 Condicionais
eu.5.3.1 seii.5.3.2 else ifiii.5.3.3 Switchd.5.4 Loopseu.5.4.1 enquantoii.5.4.2 Do/whileiii.5.4.3 para

4.5.4.4 para/dev. 5.4.5 para/ine.5.5 saltoseu.5.5.1 Declarações rotuladasii.5.5.2 Breakiii.5.5.3 Continue4.5.5.4 Retornov. 5.5.5 Rendimentovi.5.5.6 Joguevii.5.5.7 Tente/Catch/Finalmentef.5.6 Declarações diversaseu.5.6.1 comii.5.6.2 Depuradoriii.5.6.3 ?Use rigoroso?g.5.7 declaraçõeseu.5.7.1 const, let e vari.5.7.2 Funçaoiii.5.7.3 Classe4.5.7.4 Importação e exportaçãoh.5.8 Resumo das declarações JavaScript7. Objetosum.6.1 Introdução aos objetos

b.6.2 Criando objetos
e.6.2.1 Literais de objeto
ii.6.2.2 Criando objetos com novo
iii.6.2.3 Protótipos
4.6.2.4
Object.Create ()
c.6.3 Propriedades de consulta e definição
e.6.3.1 Objetos como matrizes associativas
ii.6.3.2 Herança
iii.6.3.3 Erros de acesso à propriedade
d.6.4 Excluindo propriedade
e.6.5 Propriedades de teste
f.6.6 Propriedades de enumeração
e.6.6.1 Ordem de enumeração da propriedade
g.6.7 estendendo objetos
h.6.8 Objetos serializados
e.6.9 Métodos de objeto
e.6.9.1 O método ToString ()
ii.6.9.2 O método ToLocaleString ()
iii.6.9.3 o método ValueOf ()
4.6.9.4 O método ToJson ()
j.6.10 Sintaxe literal de objeto estendido
e.6.10.1 Propriedades abreviadas

ii.6.10.2 Nomes de propriedades computadasiii.6.10.3 Símbolos como nomes de propriedades4.6.10.4 Operador de espalhamentov. 6.10.5 Métodos de abreviaçãovi.6.10.6 Getters de propriedades e settersk.6.11 Resumo8. Matrizesum.7.1 Criando matrizeseu.7.1.1 Literais da matrizii.7.1.2 O operador de propagaçãooiii.7.1.3 O construtor Array ()4.7.1.4 Array.of ()v. 7.1.5 Array.From ()b.7.2 Elementos de matriz de leitura e escritac.7.3 Matrizes esparsasd.7.4 Comprimento da matrize.7.5 Adicionando e excluindo elementos de matrizf.7.6 Matrizes de iteraçãog.7.7 Matrizes multidimensionaish.7.8 Métodos de matrizeu.7.8.1 Métodos de iterador de matrizii.7.8.2 Matrizes achatadas com plano () eFlatmap ()

iii.7.8.3 Adicionando matrizes com concat ()
4.7.8.4 pilhas e filas com push (),pop (), shift () e não dividido ()
v. 7.8.5
subarrays with slice (), splice (),preench () e copywithin ()
vi.7.8.6 Pesquisa e classificação de
matrizesMétodos
vii.7.8.7 Array para conversões de string
viii.7.8.8 Funções de matriz estática
e.7.9 Objetos
semelhantes a matriz
j.7.10 Strings como matrizes
k.7.11 Resumo
9. funções
sum.8.1 Definindo funções
e.8.1.1 declarções de função
ii.8.1.2 Expressões de função
iii.8.1.3 Funções de seta
4.8.1.4 Funções aninhadas
b.8.2 Funções de invocaçâo
e.8.2.1 Invocação da função
ii.8.2.2 Invocação do método
iii.8.2.3 Invocação do
construtor
4.8.2.4 Invocação indireta

v. 8.2.5 Invocação de função implícita
c.8.3 Argumentos e parâmetros de função
e.8.3.1 parâmetros e padrões opcionais
ii.8.3.2 Parâmetros de descanso e variável-Listas de argumentos de comprimento
iii.8.3.3 O objeto de argumentos
iv.8.3.4 O operador de propagação para função Chamadas
v. 8.3.5 Argumentos de função de destruição em parâmetros
vi.8.3.6 Tipos de argumentos
d.8.4 Funções como valores
e.8.4.1 Definindo sua própria função Propriedade
f.8.5 Funções como namespaces
g.8.6 fechamentos
h.8.7 Propriedades, métodos e métodos da função Construtor
i.8.7.1 A propriedade de comprimento
ii.8.7.2 A propriedade Nome
iii.8.7.3 A propriedade do protótipo
iv.8.7.4 Os métodos Call () e Apply ()
v. 8.7.5 O método bind ()
vi.8.7.6 O método ToString ()

vii.8.7.7 O construtor function ()
h.8.8 Programação funcional
e.8.8.1 Matrizes de processamento com funções
ii.8.8.2 Funções de ordem superior
iii.8.8.3 Aplicação parcial de funções
4.8.8.4 MEMOIZAÇÃO
e.8.9 Resumo
10. Classes
sum.9.1 classes e protótipos
b.9.2 Classes e Construtores
e.9.2.1 Construtores, identidade de classe e
Instância de
ii.9.2.2 A propriedade do construtor
c.9.3 Classes com a palavra -chave da classe
e.9.3.1 Métodos estáticos
ii.9.3.2 Getters, Setters e outros métodos
Formas
iii.9.3.3 Campos públicos, privados e estáticos
4.9.3.4 Exemplo: uma classe de números complexos
d.9.4 Adicionando métodos às classes existentes
e.9.5 subclasses
e.9.5.1 subclasses e protótipos
ii.9.5.2 subclasses com extensões e super

iii.9.5.3 Delegação em vez de herança4.9.5.4 Hierarquias de classe e resumoClassesf.9.6 Resumo11.
Módulosum.10.1 Módulos com classes, objetos e fechamentoeu.10.1.1 Automatando baseado em
fechamentoModularidadeb.10.2 Módulos no nóeu.10.2.1 Exportações de nósii.10.2.2 Importações de nósiii.10.2.3
módulos no estilo de nó na webc.10.3 Módulos em ES6eu.10.3.1 ES6 Exportaçõesii.10.3.2 ES6
importaçõesiii.10.3.3 importações e exportações comRenomear4.10.3.4 Reexportsv. 10.3.5 Módulos JavaScript na
webvi.10.3.6 Importações dinâmicas com importação ()vii.10.3.7 Import.Meta.urld.10.4 Resumo12. A biblioteca
padrão JavaScript

um.11.1 conjuntos e mapaseu.11.1.1 A classe definidaii.11.1.2 A classe do mapaiii.11.1.3 Frawmap e fracob.11.2 Matrizes digitadas e dados binárioseu.11.2.1 Tipos de matriz digitadosii.11.2.2 Criando matrizes digitadasiii.11.2.3 Usando matrizes digitadas4.11.2.4 Métodos de matriz digitados ePropriedadesv. 11.2.5 DataView e Endianessc.11.3 Combinação de padrões com expressões regulareseu.11.3.1 Definindo expressões regularesii.11.3.2 Métodos de string para padrãoCorrespondênciiaiii.11.3.3 A classe Regexpd.11.4 datas e horárioseu.11.4.1 Timestampsii.11.4.2 Data aritméticaiii.11.4.3 Data de formatação e análiseCordase.11.5 Classes de errof.11.6 Serialização e análise de JSON JSON

eu.11.6.1 Customizações JSONg.11.7 A API de internacionalizaçõeou.11.7.1 Números de formataçãoi.11.7.2 Datas e horários de formataçãoi.11.7.3 Comparando stringsh.11.8 A API do consoleeu.11.8.1 Saída formatada com consoleeu.11.9 URL APIseu.11.9.1 Funções de URL do Legacyj.11.10 Timersk.11.11 Resumo13. Iteradores e geradoresum.12.1 como os iteradores funcionamb.12.2 Implementando objetos iteráveiseu.12.2.1 ?Fechando? um iterador: o retornoMétodoc.12.3 geradoreseu.12.3.1 Exemplos de geradoresii.12.3.2 Rendimento* e geradores recursivosd.12.4 Recursos avançados do geradoreu.12.4.1 O valor de retorno de um geradorFunçãoi.12.4.2 O valor de uma expressão de rendimento

iii.12.4.3 Os métodos de retorno () e arremesso () de um gerador
4.12.4.4 Uma nota final sobre geradores.
12.5 Resumo
14. JavaScript assíncrono
um.13.1 Programação assíncrona com retornos de chamada
e eu.13.1.1 Timers
ii.13.1.2 Eventos
iii.13.1.3 Eventos de rede
4.13.1.4 retornos de chamada e eventos no nós.
13.2 promessas
seu.13.2.1 Usando promessas
ii.13.2.2 Promessas de encadeamento
iii.13.2.3 Resolvendo promessas
4.13.2.4 Mais sobre promessas e erros
v. 13.2.5 promessas em paralelo
vi.13.2.6 Fazendo promessas
vii.13.2.7 Promessas em sequênciac
13.3 assíncrono e aguardareu
13.3.1 Aguardar expressões
ii.13.3.2 Funções assíncronas
iii.13.3.3 Aguardando várias promessas
4.13.3.4 Detalhes da implementação

d.13.4 iteração assíncronaeu.13.4.1 O loop for/waitii.13.4.2 Iteradores assíncronosiii.13.4.3 geradores assíncronos4.13.4.4 Implementando assíncronoIteradorese.13.5 Resumo15. Metaprogramaçãoum.14.1 Atributos da propriedadeb.14.2 Extensibilidade do objetc.14.3 O atributo do protótipod.14.4 Símbolos bem conhecidoseu.14.4.1 Symbol.iterator eSymbol.asyncIteratorii.14.4.2 Symbol.HasInsinStanceiii.14.4.3 Symbol.ToStringTag4.14.4.4 Symbol.Speciesv. 14.4.5 Symbol.iscoNcatsPreadablevi.14.4.6 Símbolos de correspondência de padrõesvii.14.4.7 Símbolo.Toprimitivoviii.14.4.8 Symbol.unscopablese.14.5 Tags de modelof.14.6 A API Refletir

g.14.7 Objetos de proxyeu.14.7.1 Invariantes de procuraçãoh.14.8 Resumo16. JavaScript em navegadores da webum.15.1 básicos de programação da webeu.15.1.1 JavaScript em tags HTML <Script>ii.15.1.2 O modelo de objeto de documentoiii.15.1.3 O objeto global na webNavegadores4.15.1.4 Os scripts compartilham um espaço para nomev. 15.1.5 Execução de programas JavaScriptvi.15.1.6 Entrada e saída do programavii.15.1.7 Erros do programaviii.15.1.8 O modelo de segurança da webb.15.2 Eventoseu.15.2.1 Categorias de eventosii.15.2.2 Manipuladores de eventos de registroiii.15.2.3 Invocação do manipulador de eventos4.15.2.4 Propagação de eventosv. 15.2.5 Cancelamento de eventosvi.15.2.6 Despacha eventos personalizadosc.15.3 Documentos de script

eu.15.3.1 Selecionando elementos do documentoii.15.3.2 Estrutura de documentos e travessiaiii.15.3.3 Atributos4.15.3.4 Conteúdo do elemento v. 15.3.5 Criando, inserindo e excluindo Nósvi.15.3.6 Exemplo: gerando uma tabela de Conteúdosd.15.4 CSS de scripte u.15.4.1 Classes CSSii.15.4.2 Estilos embutidosiii.15.4.3 Estilos computados4.15.4.4 folhas de estilo de scriptv. 15.4.5 Animações e eventos CSSe.15.5 Geometria de documentos e rolagemeu.15.5.1 Coordenadas de documentos e Coordenadas de viewportii.15.5.2 Consultando a geometria de um Elementoiii.15.5.3 Determinando o elemento em um Apontar4.15.5.4 Rolagemv. 15.5.5 Tamanho da viewport, tamanho de conteúdo e Posição de role

f.15.6 Componentes da Web
e.15.6.1 Usando componentes da Web
ii.15.6.2 Modelos HTML
iii.15.6.3 Elementos personalizados
4.15.6.4 Shadow Dom
v. 15.6.5 Exemplo: uma web <search-box>
Componente g.15.7 SVG: gráficos vetoriais escaláveis
e.15.7.1 SVG em HTML
ii.15.7.2 Scripts SVG
iii.15.7.3 Criando imagens SVG com JavaScript
h.15.8 Gráficos em A <Canvas>
e.15.8.1 caminhos e polígonos
ii.15.8.2 dimensões de tela e Coordenadas
iii.15.8.3 Atributos gráficos
4.15.8.4 Operações de desenho de tela
v. 15.8.5 Transformações do sistema de coordenadas
vi.15.8.6 recorte
vii.15.8.7 Manipulação de pixels
e.15.9 APIs de áudio
e.15.9.1 O construtor Audio ()

ii.15.9.2 A API Webaudioj.15.10 Localização, navegação e históriaeu.15.10.1 Carregando novos documentosii.15.10.2 História de navegaçãooiii.15.10.3 Gerenciamento de história comHashChange Events4.15.10.4 Gerenciamento de história compushState ()k.15.11 Rede de redeeu.15.11.1 Fetch ()ii.15.11.2 Eventos enviados pelo servidori.15.11.3 Websocketsl.15.12 Armazenamentoeu.15.12.1 LocalStorage e SessionStorageii.15.12.2 Cookiesiii.15.12.3 IndexedDBm.15.13 fios de trabalhador e mensagenseu.15.13.1 Objetos trabalhadoresii.15.13.2 O objeto global em trabalhadoresiii.15.13.3 Importar código para um trabalhador4.15.13.4 Modelo de execução do trabalhadov. 15.13.5 PostMessage (), Messageports,e Messagechannels

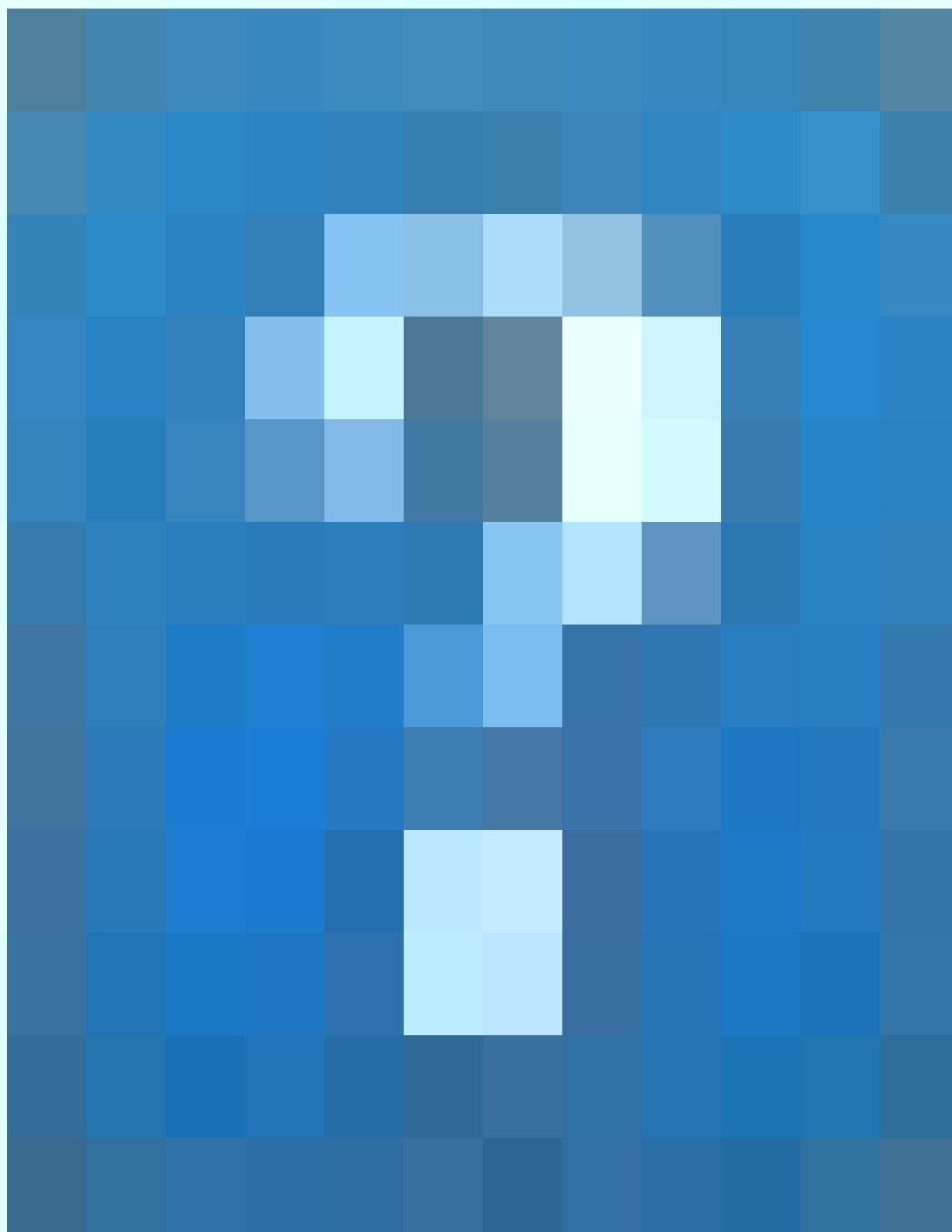
vi.15.13.6 mensagens de origem cruzada comPostMessage ()n.15.14 Exemplo: o conjunto Mandelbroto.15.15 Resumo e sugestões para maisLeituraeu.15.15.1 HTML e CSSii.15.15.2 Desempenhoiii.15.15.3 Segurança4.15.15.4 WebAssemblyv. 15.15.5 Mais documentos e janelasCaracterísticasvi.15.15.6 Eventosvii.15.15.7 Aplicativos da Web progressivos eTrabalhadores de serviçoviii.15.15.8 APIs de dispositivo móvelix.15.15.9 APIs bináriasx.15.15.10 APIs de mídiaxi.15.15.11 Criptografia e APIs relacionadas17. JavaScript do lado do servidor com núum.16.1 Programação do Nó básicoeu.16.1.1 Saída do consoleii.16.1.2 Argumentos da linha de comando eVariáveis ??de ambienteiii.16.1.3 Ciclo de vida do programa

4.16.1.4 Módulos de nós
v. 16.1.5 O gerenciador de pacotes do nó
b. 16.2 O nó é assíncrono por padrão.
c. 16.3 Buffers
d. 16.4 Eventos e EventEmitters
e. 16.5 fluxos
f. 16.5.1 Tubos
g. 16.5.2 iteração assíncrona
h. 16.5.3 Escrevendo para fluxos e manuseioBackpressure
i. 16.5.4 Lendo fluxos com eventos
f. 16.6 Process, CPU e detalhes do sistema operacional
g. 16.7 trabalhando com arquivos
h. 16.7.1 caminhos, descritores de arquivos e FileHandles
i. 16.7.2 Leitura de arquivos
j. 16.7.3 Escrevendo arquivos
k. 16.7.4 Operações de arquivos
l. 16.7.5 metadados do arquivo
m. 16.7.6 Trabalhando com diretórios
h. 16.8 clientes e servidores HTTP
u. 16.9 Servidores e clientes de rede não-HTTP
j. 16.10 Trabalhando com processos filhos

eu.16.10.1 Execsync () e ExecFilesync ()ii.16.10.2 EXEC () e EXECFILE ()iii.16.10.3 Spawn ()4.16.10.4 Fork ()k.16.11 tópicos dos trabalhadores seu.16.11.1 Criando trabalhadores e aprovação Mensagensii.16.11.2 A execução do trabalhador Ambienteiii.16.11.3 canais de comunicação e Messageports4.16.11.4 transferindo Messageports e Matrizes digitadas av. 16.11.5 Compartilhando matrizes digitadas entre Tópicos l.16.12 Resumo18. Ferramentas e extensões JavaScriptum.17.1 LING COM ESLINTb.17.2 Javascript Formating com mais bonito c.17.3 Teste de unidade com JESTd.17.4 Gerenciamento de pacotes com NPM e.17.5 Bundling de código f.17.6 Transpilação com Babelg.17.7 JSX: Expressões de marcação em JavaScript

h.17.8 Verificação do tipo com fluxo
e.17.8.1 Instalando e executando o fluxo
i.17.8.2 Usando anotações de tipo
iii.17.8.3 Tipos de classe
4.17.8.4 Tipos de objetos
v. 17.8.5 Aliases do tipo
vi.17.8.6 Tipos de matriz
vii.17.8.7 Outros tipos parametrizados
viii.17.8.8 Tipos somente leitura
ix.17.8.9 Tipos de funções
x.17.8.10 Tipos de sindicatos
xi.17.8.11 tipos enumerados
eSindicatos discriminados
e.17.9 Resumo
19. ÍNDICE

Louvor ao JavaScript: The Definitive Guide, Sétima Edição?Este livro é tudo o que você nunca soube que queria saber sobre JavaScript. Leve a qualidade do código JavaScript e a produtividade para o próximo nível. O conhecimento de David sobre a linguagem, seus meandros e petchas, é surpreendente e brilha neste verdadeiro Guia para a linguagem JavaScript. ?- Schalk Neethling, engenheiro sênior de front-end em MDN Web Docs?David Flanagan leva os leitores a uma visita guiada a JavaScript que fornecerá a eles uma imagem completa de recurso do idioma e seu ecossistema. ?- Sarah Wachs, desenvolvedor de front-end e mulheres que Código Berlin Lead?Qualquer desenvolvedor interessado em ser produtivo em bases de código desenvolvido durante toda a vida de JavaScript (incluindo os mais recentes recursos emergentes) serão bem servidos por um profundo e reflexivo Viaje por este livro abrangente e definitivo. ??Brian Sletten, presidente da Bosatsu Consulting



JavaScript: The Definitive Guide, Sétima EdiçãoPor David FlanaganCopyright © 2020 David Flanagan.Todos os direitos reservados.Impresso nos Estados Unidos da América.Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North,Sebastopol, CA 95472.Os livros de O'Reilly podem ser adquiridos para educação, negócios ou vendasuso promocional.Edições online também estão disponíveis para a maioria dos títulos(<http://oreilly.com>).Para mais informações, entre em contato com o nossoDepartamento de Vendas Corporativas/Institucionais: 800-998-9938 oucorporate@oreilly.com.Editor de aquisições: Jennifer PollockEditor de desenvolvimento: Angela RufinoEditor de produção: Deborah BakerCopyEditor: Holly Bauer ForsythRevisor: Piper Editorial, LLCIndexador: Judith McConvilleDesigner de interiores: David FutatoDesigner de capa: Karen Montgomery

Ilustrador: Rebecca DeMare
Junho de 1998: Terceira edição
Novembro de 2001: Quarta edição
Agosto de 2006:
Quinta Edição
Maio de 2011: Sexta edição
Maio de 2020: sétima edição
Histórico de revisão para a sétima
edição
2020-05-13: Primeira versão
Consulte <http://oreilly.com/catalog/errata.csp?isbn=9781491952023>
para Detalhes da liberação.
O O'Reilly Logo é uma marca registrada da O'Reilly Media, Inc.
JavaScript: The Definitive Guide, Sétima Edição, a imagem da capa, E vestidos comerciais relacionados são marcas comerciais da O'Reilly Media, Inc.
Enquanto o editor e os autores usaram esforços de boa fé para Verifique se as informações e instruções contidas neste trabalho são precisas, o editor e os autores renunciam a toda a responsabilidade por erros ou omissões, inclusive sem limitação responsabilidade por Danos resultantes do uso ou dependência deste trabalho.
Uso de informações e instruções contidas neste trabalho são por sua conta e risco. Se alguma amostra de código ou outra tecnologia, este trabalho contiver ou descreverá está sujeito a licenças de código aberto ou os direitos de propriedade intelectual de Outros, é sua responsabilidade garantir que seu uso esteja em conformidade com essas licenças e/ou direitos.

978-1-491-95202-3[Lsi]

DedicaçãoPara meus pais, Donna e Matt, com amor e gratidão.

PrefácioEste livro cobre a linguagem JavaScript e as APIs JavaScript implementado por navegadores da web e por nó.Eu escrevi para leitores comalguma experiência de programação anterior que deseja aprender JavaScript etambém para programadores que já usam JavaScript, mas querem levar seuscompreensão de um novo nível e realmente dominar o idioma.Meu objetivocom este livro é documentar a linguagem JavaScript de forma abrangente definitivamente e para fornecer uma introdução aprofundada ao máximoAPIs importantes do lado do cliente e do lado do servidor disponíveis para JavaScriptprogramas.Como resultado, este é um livro longo e detalhado.Minha esperança,No entanto, é que ele recompensará um estudo cuidadoso e que o tempo vocêPasse a leitura será facilmente recuperado na forma de maiorprodutividade de programação.As edições anteriores deste livro incluíram uma referência abrangentesação.Não sinto mais que faz sentido incluir esse material emformulário impresso quando é tão rápido e fácil de encontrar referência atualizadamaterial online.Se você precisar procurar algo relacionado ao núcleo ouJavaScript do lado do cliente, recomendo que você visite o site da MDN.EPara APIs do nó do lado do servidor, recomendo que você vá diretamente para a fonte consulte a documentação de referênciawww.Node.js.Convenções usadas neste livroEu uso as seguintes convenções tipográficas neste livro:

ítálico É usado para ênfase e para indicar o primeiro uso de um termo. Itálico é Também usado para endereços de email, URLs e nomes de arquivos. Largura constante É usado em todo o código JavaScript e listagens CSS e HTML, geralmente para qualquer coisa que você digitaria literalmente quando programação. Largura constante em itálico É usado ocasionalmente ao explicar a sintaxe do JavaScript. Largura constante em negrito Mostra comandos ou outro texto que deve ser digitado literalmente pelo usuário. OBSERVAÇÃO Esse elemento significa uma nota geral. IMPORTANTE Este elemento indica um aviso ou cautela. Código de exemplo Material suplementar (exemplos de código, exercícios, etc.) para este livro é Disponível para download em:

https://oreil.ly/javascript_defgd7Este livro está aqui para ajudá-lo a fazer seu trabalho. Em geral, se exemplo o código é oferecido com este livro, você pode usá-lo em seus programas e documentação. Você não precisa entrar em contato conosco para obter permissão, a menos que você está reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que use vários pedaços de código deste livro não requer permissão. Vendendo ou distribuindo exemplos de O'Reilly Os livros requerem permissão. Respondendo a uma pergunta citando isso Reserve e citando o código de exemplo não requer permissão. Incorporando uma quantidade significativa de código de exemplo deste livro em A documentação do seu produto requer permissão. Agradecemos, mas geralmente não exigem, atribuição. Uma atribuição Geralmente inclui o título, autor, editor e ISBN. Por exemplo: JavaScript: The Definitive Guide, Sétima Edição, de David Flanagan (O'Reilly). Copyright 2020 David Flanagan, 978-1-491-95202-3. Se você sentir que o uso de exemplos de código cai fora do uso justo ou de permissão dada acima, sinta-se à vontade para entrar em contato conosco em permissions@oreilly.com. O'Reilly Online Learning OBSERVAÇÃO Por mais de 40 anos, a O'Reilly Media forneceu tecnologia e negócios Treinamento, conhecimento e insight para ajudar as empresas a ter sucesso.

Nossa rede única de especialistas e inovadores compartilham seus conhecimentos e experiência através de livros, artigos e nossa plataforma de aprendizado on-line. A plataforma de aprendizado on-line da O'Reilly oferece acesso sob demanda ao vivo a cursos de treinamento, caminhos de aprendizagem detalhados, codificação interativa em ambientes e uma vasta coleção de texto e vídeo da O'Reilly e mais de 200 outros editores. Para mais informações, visite <http://oreilly.com>. Como nos contatar? Por favor, aborde os comentários e perguntas sobre este livro ao editor: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 800-998-9938 (nos Estados Unidos ou no Canadá), 707-829-0515 (internacional ou local), 707-829-0104 (fax). Temos uma página da web para este livro, onde listamos erros, exemplos e qualquer informação adicional. Você pode acessar esta página em https://oreil.ly/javascript_defgd7. Envie um email para bookquestions@oreilly.com para comentar ou perguntar técnico.

perguntas sobre este livro.Para notícias e mais informações sobre nossos livros e cursos, consulte nossoSite em <http://www.oreilly.com>.Encontre -nos no Facebook: <http://facebook.com/oreillySiga> -nos no Twitter: <http://twitter.com/oreillymedia>Assista ao YouTube: <http://www.youtube.com/oreillymedia>AgradecimentosMuitas pessoas ajudaram na criação deste livro.Eu gostaria deGraças à minha editora, Angela Rufino, por me manter no caminho certo e por elaPaciência com meus prazos perdidos.Obrigado também ao meu técnicoRevisores: Brian Sletten, Elisabeth Robson, Ethan Flanagan, Maximiliano Firtman, Sarah Wachs e Schalk Neethling.DelesComentários e sugestões fizeram deste um livro melhor.A equipe de produção de O'Reilly fez o seu bom trabalho habitual: KristenBrown gerenciou o processo de produção, Deborah Baker foi oA editora de produção, Rebecca DeMarest, atraiu as figuras, e JudyMcConville criou o índice.Editores, revisores e colaboradores de edições anteriores deste livroincluiu: Andrew Schulman, Angelo Sirigos, Aristóteles Pagaltzis, Brendan Eich, Christian Heilmann, Dan Shafer, Dave C. Mitchell, DebCameron, Douglas Crockford, Dr. Tankred Hirschmann, Dylan

Schiemann, Frank Willison, Geoff Stearns, Herman Venter, Jay Hodges, Jeff Yates, Joseph Kesselman, Ken Cooper, Larry Sullivan, Lynn Rollins, Neil Berkman, Mike Loukides, Nick Thompson, Norris Boyd, Paula Ferguson, Peter-Paul Koch, Philippe Le Hegaret, Raffaele Cecco, Richard Yaker, Sanders Kleinfeld, Scott Furman, Scott Isaacs, Shon Katzenberger, Terry Allen, Todd Ditchendorf, Vidur Aparao, Waldemar Horwat e Zachary Kessin. Escrever esta sétima edição me manteve longe da minha família para muitos tarde da noite. Meu amor a eles e meus agradecimentos por aguentar meus ausências. David Flanagan, março de 2020

Capítulo 1. Introdução a JavaScript

JavaScript é a linguagem de programação da web. A maioria dos sites usa JavaScript e todos os navegadores da web modernos - em desktops, tablets e telefones - incluindo intérpretes de JavaScript, fazendo JavaScript a linguagem de programação mais implantada da história.

Sobre Na última década, o Node.js permitiu a programação JavaScript fora de navegadores da web e o sucesso dramático do Node.js significa que JavaScript agora também é a linguagem de programação mais usada entre desenvolvedores de software.

Se você está começando do zero ou já está usando o JavaScript profissionalmente, este livro o ajudará a dominar o idioma.

Se você já está familiarizado com outras linguagens de programação, pode ajudá-lo a saber que o JavaScript é um de alto nível, dinâmico, interpretado linguagem de programação que é adequada para orientação de objetos e estilos de programação funcionais.

As variáveis ??do JavaScript não são criadas. Isso é a sintaxe é vagamente baseada em Java, mas os idiomas são de outra forma não relacionados. JavaScript deriva suas funções de primeira classe do esquema sua herança baseada em protótipo da linguagem pouco conhecida.

Mas você não precisa conhecer nenhum desses idiomas ou estar familiarizado com esses termos, usar este livro e aprender JavaScript.

O nome "JavaScript" é bastante enganador. Exceto por uma superficial semelhança sintática, JavaScript é completamente diferente do Java.

linguagem de programação. E JavaScript há muito tempo superou suas raízes em linguagem de script para se tornar um general robusto e eficiente. Linguagem de propósito adequada para engenharia e projetos graves de software com enormes bases de código. JavaScript: nomes, versões e modos. JavaScript foi criado no Netscape nos primeiros dias da web e tecnicamente, "JavaScript" é um marca registrada licenciada da Sun Microsystems (agora Oracle) usada para descrever o Netscape's (agora Mozilla's) implementação do idioma. Netscape enviou o idioma para padronização à ECMA - a Associação Europeia de Fabricante de Computadores - e por questões de marca registrada, o padronizado. A versão do idioma ficou presa com o nome desajeitado "Ecmascript". Na prática, todos apenas chamam o idioma JavaScript. Este livro usa o nome "ecmascript" e a abreviação "es" para consultar o padrão de idioma e as versões desse padrão. Na maior parte dos anos 2010, a versão 5 do padrão Ecmascript foi suportada por toda a webnavegadores. Este livro trata o ES5 como a linha de base da compatibilidade e não discute mais versões anteriores do idioma. O ES6 foi lançado em 2015 e adicionou novos recursos - incluindo aula e Sintaxe do módulo que alterou o JavaScript de uma linguagem de script em um grande, de uso geral idioma adequado para engenharia de software em larga escala. Desde o ES6, a especificação do ECMAScript tem mudado-se para uma cadência de liberação anual e versões do idioma - ES2016, ES2017, ES2018, ES2019 e ES2020 - agora são identificados por ano de lançamento. À medida que o JavaScript evoluiu, os designers de idiomas tentaram corrigir falhas no início (pré-ES5) versões. Para manter a compatibilidade com versões anteriores, não é possível remover recursos legados, não importa o quanto falho. Mas no ES5 e mais tarde, os programas podem optar pelo modo estrito de JavaScript no qual um número de erros de idioma inicial foram corrigidos. O mecanismo de opção é o ?UseStrict diretiva estrita ?descrita em §5.6.3. Essa seção também resume as diferenças entre o legado JavaScript e JavaScript rigoroso. No ES6 e mais tarde, o uso de novos recursos de linguagem frequentemente implicitamente invoca o modo rigoroso. Por exemplo, se você usar a palavra-chave da classe ES6 ou criar um módulo ES6, então todo o código dentro da classe ou módulo é automaticamente rigoroso, e os recursos antigos e defeituosos não são disponíveis nesses contextos. Este livro abordará os recursos legados do JavaScript, mas é cuidadoso para apontar que eles não estão disponíveis no modo rigoroso. Para ser útil, todo idioma deve ter uma plataforma ou biblioteca padrão, para realizar coisas como entrada e saída básicas. O principal JavaScript linguagem define uma API mínima para trabalhar com números, texto, Matrizes, conjuntos, mapas e assim por diante, mas não inclui nenhuma entrada ou saída funcionalidade. Entrada e saída (bem como recursos mais sofisticados, como networking, armazenamento e gráficos) são de responsabilidade do

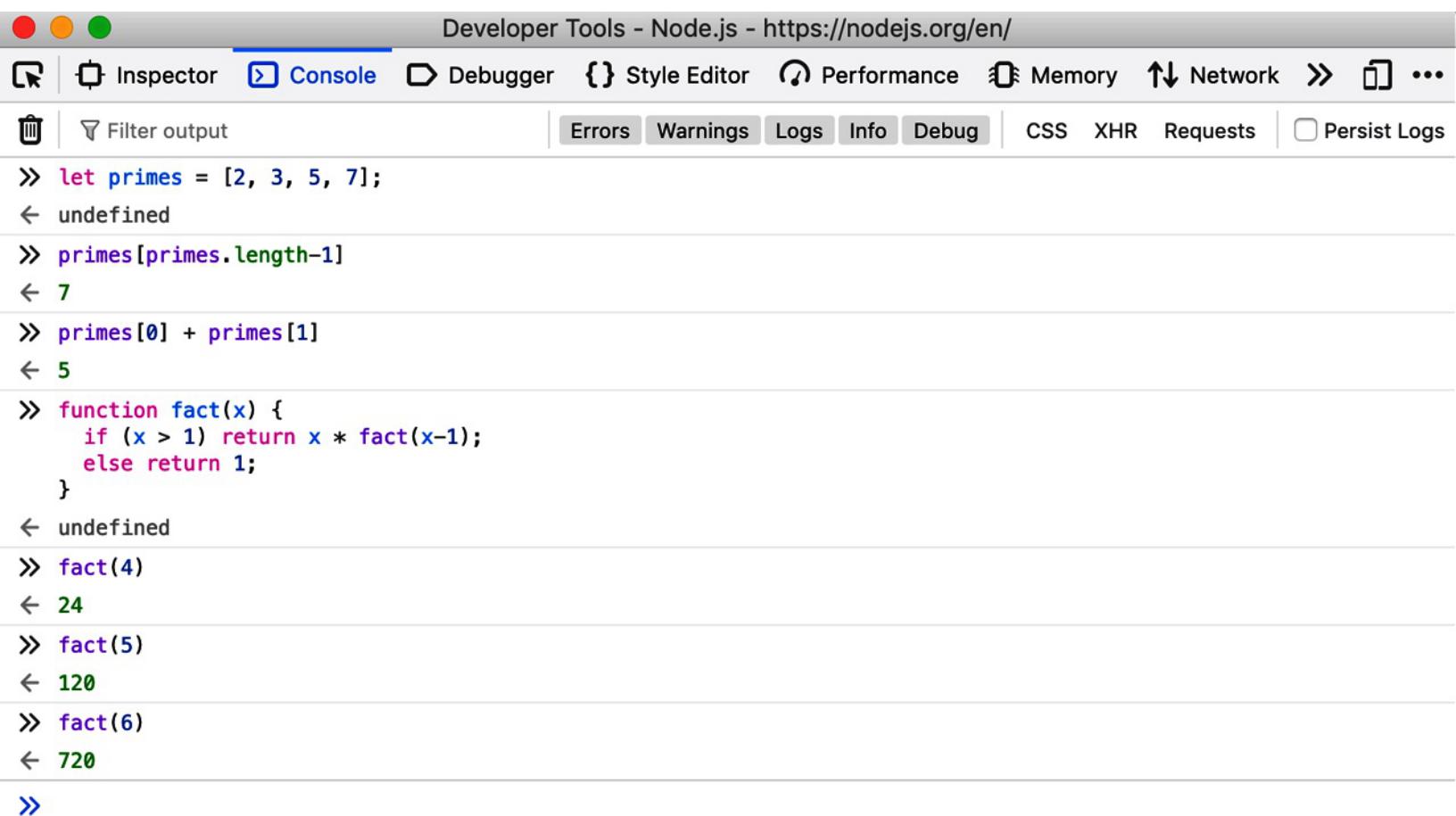
?Ambiente do host? dentro do qual o JavaScript está incorporado.O ambiente host original para JavaScript era um navegador da web, eEste ainda é o ambiente de execução mais comum para JavaScriptcódigo.O ambiente do navegador da web permite que o código JavaScript obtenhaentrada do mouse e teclado do usuário e fazendo httpsolicitações.E permite que o código JavaScript exiba saída para o usuáriocom HTML e CSS.Desde 2010, outro ambiente host está disponível para JavaScriptcódigo.Em vez de restringir o JavaScript para trabalhar com as APIsFornecido por um navegador da web, o nó dá acesso a JavaScript a todosistema operacional, permitindo que os programas JavaScript leiam e gravem arquivos,Envie e receba dados sobre a rede e faça e sirva HTTPPsolicitações.O nó é uma escolha popular para implementar servidores da web etambém uma ferramenta conveniente para escrever scripts simples de utilitário como alternativascripts de conchas.A maior parte deste livro está focada na própria linguagem JavaScript.Capítulo 11 documenta a biblioteca padrão JavaScript, capítulo 15Apresenta o ambiente do host do navegador da web e o capítulo 16Apresenta o ambiente do host do nó.Este livro abrange os fundamentos de baixo nível primeiro e depois se baseia nelesa abstrações mais avançadas e de nível superior.Os capítulos sãodenominados a ser lido mais ou menos em ordem.Mas aprender um novoA linguagem de programação nunca é um processo linear, e descrevendo uma linguagem também não é linear: cada recurso de idioma está relacionado a outrosRecursos, e este livro está cheio de referências cruzadas-às vezes

para trás e às vezes para a frente - para material relacionado. Esse O capitulo introdutorio faz uma primeira passagem rápida pelo idioma, a introdução dos principais recursos que facilitarão a compreensão do IN-Tratamento em profundidade nos capítulos a seguir. Se você já é um Praticando o programador JavaScript, você provavelmente pode pular este capítulo. (Embora você possa gostar de ler o Exemplo 1-1 no final do capitulo antes de seguir em frente.)

1.1 Explorando JavaScript Ao aprender uma nova linguagem de programação, é importante tentar o exemplos no livro, depois modifique -os e experimente -os novamente para testar seu entendimento da linguagem. Para fazer isso, você precisa de um javascript intérprete. A maneira mais fácil de experimentar algumas linhas de javascript é abrir o Ferramentas de desenvolvedor da web em seu navegador da web (com F12, Ctrl-Shift-i, ou Comando-opção-i) e selecione a guia Console. Você pode então digitar código no prompt e veja os resultados conforme você digita.

Desenvolvedor do navegador As ferramentas geralmente aparecem como painéis na parte inferior ou à direita do navegador janela, mas você geralmente pode destacá -los como janelas separadas (como Na figura 1-1), o que geralmente é bastante conveniente.

Figura 1-1.O JavaScript Console nas ferramentas de desenvolvedor do FirefoxOutra maneira de experimentar o código JavaScript é baixar e instalar o nóde <https://nodejs.org>.Uma vez que o nó é instalado no seu sistema, você pode simplesmente abrir uma janela do terminal e digitar nó para iniciar um



The screenshot shows the Firefox Developer Tools interface with the 'Console' tab selected. The title bar reads 'Developer Tools - Node.js - https://nodejs.org/en/'. The console output is as follows:

```
» let primes = [2, 3, 5, 7];
← undefined
» primes[primes.length-1]
← 7
» primes[0] + primes[1]
← 5
» function fact(x) {
  if (x > 1) return x * fact(x-1);
  else return 1;
}
← undefined
» fact(4)
← 24
» fact(5)
← 120
» fact(6)
← 720
»
```

sessão interativa de javascript como esta:\$ nodeBem -vindo ao Node.js v12.13.0.Digite ".help" para obter mais informações.> .Help. quebra às vezes você fica preso, isso o tira..Editor Enter o modo editor.Exit saia do repl.Help Imprima esta mensagem de ajuda.Lote Carregar JS de um arquivo na sessão Repl. Salvar todos os comandos avaliados nesta sessão repl para um arquivoPressione ^c para abortar a expressão atual, ^d para sair do repl> Seja x = 2, y = 3; indefinido> x + y5> (x === 2) && (y === 3) verdadeiro> (x > 3) ||(y < 3)falso1.2 Hello WorldQuando você estiver pronto para começar a experimentar pedaços mais longos de código, Esses ambientes interativos linha por linha podem não ser mais adequados, E você provavelmente preferirá escrever seu código em um editor de texto. De lá, você pode copiar e colar no console JavaScript ou em um nó sessão. Ou você pode salvar seu código em um arquivo (o nome do arquivo tradicional Extensão para o código JavaScript é .js) e, em seguida, execute o arquivo de javascript código com nó:\$ node snippet.jsSe você usar o nó de uma maneira não interativa como essa, não será

Imprima automaticamente o valor de todo o código que você executa, para que você tenha para fazer isso você mesmo. Você pode usar o console de função.log () para exibir texto e outros valores de JavaScript em sua janela do terminal ou em O console de ferramentas de desenvolvedor de um navegador. Então, por exemplo, se você criar umHello.js File contendo esta linha de código:console.log ("Hello World!"); e execute o arquivo com o node hello.js, você verá a mensagem "Olá mundo!" impresso. Se você quiser ver a mesma mensagem impressa no JavaScript console de um navegador da web, crie um novo arquivo chamado Hello.html e coloque este texto nele:<script src = "hello.js"> </script> Em seguida, carregue hello.html no seu navegador da web usando um arquivo: // url como este: Arquivo: //users/username/javascript/hello.html Abra a janela Ferramentas do desenvolvedor para ver a saudação no console.

1.3 Um passeio de JavaScript

Esta seção apresenta uma introdução rápida, através de exemplos de código, para a linguagem JavaScript. Após este capítulo introdutório, mergulhamos em JavaScript no nível mais baixo: o capítulo 2 explica coisas como JavaScript Comentários, semicolons e o conjunto de caracteres Unicode. Capítulo 3 começa para ficar mais interessante: explica as variáveis ?? JavaScript e os valores

Você pode atribuir a essas variáveis. Aqui está algum código de exemplo para ilustrar os destaques daqueles dois Capítulos:// Qualquer coisa após barras duplas é uma língua inglesa comentário.// Leia os comentários com cuidado: eles explicam o javascript código.// Uma variável é um nome simbólico para um valor.// As variáveis ?? são declaradas com a palavra -chave Let: deixe x; // Declare uma variável chamada x.// Os valores podem ser atribuídos a variáveis ?? com um signo x = 0; // agora a variável x tem o valor 0x // => 0: Uma variável avalia para seu valor.// javascript suporta vários tipos de valores x = 1; // números. x = 0,01; // números podem ser inteiros ou reais. x = "Hello World"; // seqüências de texto na citação Marcas. x = 'JavaScript'; // Marcas de cotação única também delimitam cordas. x = true; // Um ?? valor booleano. x = false; // O outro valor booleano. x = nulo; // null é um valor especial que significa "sem valor". x = indefinido; // indefinido é outro especial valor como nulo. Dois outros tipos muito importantes que os programas JavaScript podem manipular são objetos e matrizes. Estes são os sujeitos dos capítulos 6 e 7, mas eles são tão importantes que você os verá muitas vezes antes Você chega a esses capítulos:

```
// O Datatype mais importante do JavaScript é o objeto.// Um ??objeto é uma coleção de pares de nome/valor ou
uma string para valorizar mapa. Deixe o livro = {}// Objetos estão fechados em Curlyaparelho ortodôntico. Tópico:
"JavaScript", // the Property "tópico" tem valor "JavaScript." Edição: 7 // A propriedade "Edition" tem valor 7};// A cinta
encaracolada marca o fim do objeto.// Acesse as propriedades de um objeto com ou []:book.topic // =>
"JavaScript"livro ["edição"] // => 7: Outra maneira de acessar valores da propriedade. book.author = "flanagan";//
Crie novas propriedades por atribuição. book.contents = {};// {} é um objeto vazio sem propriedades.// Acesse
propriedades de acesso condicionalmente com?.(ES2020):book.contents?.ch01?.sect1 // => indefinido:
book.contents tem Nenhuma propriedade CH01.// JavaScript também suporta matrizes (numericamente
indexadas) dos valores: deixe os primos = [2, 3, 5, 7];// Uma matriz de 4 valores, delimitada com [e].primos [0] //
=> 2: o primeiro elemento (índice 0) da matriz. prima.length // => 4: quantos elementos na variedade. primos
[prime.length-1] // => 7: o último elemento da variedade. primos [4] = 9;// Adicione um novo elemento
por atribuição. primos [4] = 11;// ou alterar um elemento existente por atribuição. deixe vazio = [];// [] é uma matriz
vazia sem elementos. vazio.length // => 0// Matrizes e objetos podem conter outras matrizes e objetos:
```

Deixe pontos = [// Uma matriz com 2 elementos.{x: 0, y: 0}, // Cada elemento é um objeto.{x: 1, y: 1}]; Deixe dados = { // um objeto com 2 propriedadesTrial1: [[1,2], [3,4]], // O valor de cada propriedade é uma matriz.Trial2: [[2,3], [4,5]] // Os elementos das matrizes são matrizes.}; Comentário sintaxe em exemplos de códigoVocê deve ter notado no código anterior que alguns dos comentários começam com uma seta (=>). Eles mostram o valor produzido pelo código antes do comentário e são minha tentativa de imitar um ambiente interativo JavaScript, como um console de navegador da web em um livro impresso. Esses comentários também servem como uma afirmação, e eu escrevi uma ferramenta que testa o código para verificar que produz o valor especificado no comentário. Isso deve ajudar, espero, reduzir erros no livro. Existem dois estilos relacionados de comentário/afirmação. Se você vir um comentário do formulário // a == 42, ele significa que após o código antes do comentário executar, a variável A terá o valor 42. Se você vir um comentário do formulário //!, significa que o código na linha antes do comentário lança uma exceção (e o resto do comentário após o ponto de exclamação geralmente explica que tipo de exceção é lançada). Você verá esses comentários usados ?? ao longo do livro. A sintaxe ilustrada aqui para listar elementos de matriz no quadrado aparelhos ou mapeamento de nomes de propriedades para objetos para valores de propriedades dentro de brace curly é conhecido como expressão inicializadora, e é apenas um dos tópicos do capítulo 4. Uma expressão é uma frase de javascript que pode ser avaliada para produzir um valor. Por exemplo, o uso de e [] Para se referir ao valor de uma propriedade de objeto ou elemento de matriz, é uma expressão. Uma das maneiras mais comuns de formar expressões em JavaScript é

Use operadores:// Os operadores agem sobre valores (os operando) para produzir um novovalor// Os operadores aritméticos são alguns dos mais simples:3 + 2 // => 5: adição3 - 2 // => 1: Subtração3 * 2 // => 6: multiplicação3/2 // => 1.5: DivisãoPontos [1] .x - pontos [0] .x // => 1: operando mais complicadoTambém trabalhe "3" + "2" // => "32": + adiciona números,Concatenados// javascript define alguns operadores aritméticos abreviadosdeixe count = 0;// Defina uma variávelcontagem ++;// incremento a variávelcontar--;// diminuir a variávelcontagem += 2;// Adicione 2: o mesmo que count = count +2;contagem *= 3;// Multiplique por 3: o mesmo que count =contagem * 3;contagem // => 6: nomes de variáveis ??são expressões também// Operadores de igualdade e relacional testam se dois valores são iguais// desigual, menor que, maior que e assim por diante.Eles avaliam para verdadeiro ou falso.Seja x = 2, y = 3;// estes = sinais são tarefas,não testes de igualdade x === y // => false: igualdadex! == y // => true: desigualdadex <y // => true: menos o que x <= y // => true: menos o que x > y // => false: Greater-than x > y // => false: maior do que "dois" === "três" // => false: as duas strings são diferentes "dois" > "três" // => true: "tw" é alfabeticamente maior que "th"false === (x > y) // => true: false é igual a falso// Operadores lógicos combinam ou invertem valores booleanos

`(x === 2) && (y === 3) // => true: ambas as comparações são verdadeiro.&& é e(x> 3) ||(y <3) // => false: Nenhuma comparação é verdadeiro.||é ou! (x === y) // => true :!inverte um booleano valorSe as expressões JavaScript são como frases, as declarações JavaScript são como frases completas.Declaracões são o tópico do capítulo 5.`

Aproximadamente,Uma expressão é algo que calcula um valor, mas não fazQualquer coisa: não altera o estado do programa de forma alguma.Declaracões, emPor outro lado, não tem um valor, mas eles alteram o estado.Você temDeclaracões variáveis ??e declaracões de atribuiçao acima.O outroampla categoria de declaracão são estruturas de controle, como condicionaise loops.Você verá exemplos abaixo, depois de cobrirmos funções.Uma função é um bloco nomeado e parametrizado de código JavaScript queVocê define uma vez e pode invocar repetidamente.Funçõesnão são cobertos formalmente até o capítulo 8, mas como objetos e matrizes,Você os verá muitas vezes antes de chegar a esse capítulo.Aqui estãoAlguns exemplos simples:// As funções são blocos parametrizados de código JavaScript quePodemos invocar.função plus1 (x) { // define uma função chamada "plus1"com o parâmetro "X"retornar x + 1;// retorna um valor maior queO valor passou em} // As funções são fechadas em Curlyaparelho ortodônticoPlus1 (y) // => 4: y é 3, então esteRetornos de invocação 3+1Deixe Square = function (x) { // As funções são valores e podem seratribuído a vars

retornar x * x;// Calcule o valor da função};// semicolon marca o fim da atribuição.quadrado (mais1 (y)) // => 16:
Invoca duas funções em uma expressãoNo ES6 e mais tarde, há uma sintaxe abreviada para definir funções.Esta
sintaxe concisa usa => para separar a lista de argumentos do corpo de função, então as funções definidas dessa
maneira são conhecidas como seta funções.As funções de seta são mais comumente usadas quando você
quer passar uma função sem nome como argumento para outra função.O código anterior se parece com isso
quando reescrito para usar funções de seta:const Plus1 = x => x + 1;// A entrada X mapeia para a saída x + 1const
square = x => x * x;// A entrada X mapeia para a saída x * xPlus1 (y) // => 4: A invocação da função é o
mesmo quadrado (mais1 (y)) // => 16Quando usamos funções com objetos, obtemos métodos:// quando as funções
são atribuídas às propriedades de um objeto, nós chamamos// eles "métodos".Todos os objetos JavaScript
(incluindo matrizes) tem métodos:deixe A = [];// Crie uma matriz vaziaA.push (1,2,3);// o método push () adiciona
elementos para uma matriz. reverse ()// Outro método: reverte a ordem dos elementos// Também podemos definir
nossos próprios métodos.A palavra -chave "isto" refere-se ao objeto// no qual o método é definido: neste caso, os
pontosArray de antes.pontos.dist = function () {// define um método para calcular

distância entre pontosSeja p1 = this [0];// Primeiro elemento da matriz somosinvocouSeja P2 = este [1];// segundo elemento do "isto"objetoSeja a = p2.x-p1.x;// diferença em coordenadas xSeja b = p2.y-p1.y;// diferença nas coordenadas yRetornar Math.sqrt (A*A + // O Teorema Pitagóricob*b);// math.sqrt () calcula o quadradoraiz};Points.dist () // => Math.sqrt (2): DistânciaEntre nossos 2 pontosAgora, como prometido, aqui estão algumas funções cujos corpos demonstramDeclarações de estrutura de controle JavaScript comum:// declarações JavaScript incluem condicionais e loops usandoa sintaxe// de C, C ++, Java e outros idiomas.função abs (x) {// uma função para calcular ovalor absoluto.if (x> = 0) {// a instrução if ...retornar x;// executa este código se oA comparação é verdadeira.} // Este é o fim do IFcláusula.else {// a cláusula opcional maisexecuta seu código seretornar -x;// A comparação é falsa.} // aparelho encaracolado opcional quando 1declaração por cláusula.} // Nota As declarações de retorno aninhadasdentro se/else.ABS (-10) === ABS (10) // => trueFunção Sum (Array) {// Calcule a soma dos elementosde uma matrizdeixe soma = 0;// Comece com uma soma inicial de 0.para (deixe x de matriz) {// loop sobre a matriz, atribuindo cadaelemento para x.

```
soma += x;// Adicione o valor do elemento aos soma.} // Este é o fim do loop.soma de retorno;// retorna a
soma.}soma (primos) => 28: soma dos primeiros 5Prima 2+3+5+7+11função fatorial (n) {// uma função para
calcularfatoriaisdeixe o produto = 1;// Comece com um produto de 1while (n> 1) {// repita declarações em {}
whileexpr in () é verdadeiroproduto *= n;// atalho para produto = produto* n;n--;// atalho para n = n - 1} // Fim do
loopproduto de retorno;// retorna o produto}Fatorial (4) => 24: 1*4*3*2função fatorial2 (n) {// outra versão usando
umloop diferenteDeixe eu, produto = 1;// Comece com 1para (i = 2; i <= n; i ++) // incremento automaticamente i
de2 até nproduto *= i;// Faça isso cada vez.{} nãoé necessário para loops de 1 lineproduto de retorno;// retorna o
fatorial}Fatorial2 (5) => 120: 1*2*3*4*5JavaScript suporta um estilo de programação orientado a objetos, mas
ésignificativamente diferente da programação "clássica" orientada a objetosidiomas.O capítulo 9 abrange a
programação orientada a objetos emJavaScript em detalhes, com muitos exemplos.Aqui está um muito
simplesexemplo que demonstra como definir uma classe JavaScript para representarPontos geométricos
2D.Objetos que são casos desta classe têm um
```

método único, denominado distância (), que calcula a distância doponto da origem:classe Point {// por convenção, os nomes da classe sãocapitalizado.construtor (x, y) {// função construtoralnicialize novas instâncias.this.x = x;// essa palavra -chave é o novo objetosendo inicializado.this.y = y;// armazenar argumentos de função comoPropriedades do objeto.} // Nenhum retorno é necessário emFunções do construtor.distance () {// Método para calcular a distância deorigem para apontar.Retornar Math.Sqrt (/ Retornar a raiz quadrada de $x^2 +y^2$.this.x * this.x + // refere -se ao pontoobjeto em qualthis.y * this.y // o método de distância éinvocado.);}}// Use a função do construtor Point () com "novo" para criarObjetos pontuaisSeja p = novo ponto (1, 1); // O ponto geométrico (1,1).// Agora use um método do objeto Point Pp.distance () // => math.sqrt2Este passeio introdutório da sintaxe fundamental de JavaScript eOs recursos termina aqui, mas o livro continua com independenteCapítulos que cobrem recursos adicionais do idioma:Capítulo 10, módulos

Mostra como o código JavaScript em um arquivo ou script pode usar JavaScript funções e classes definidas em outros arquivos ou scripts. Capítulo 11, a biblioteca padrão JavaScript Cobre as funções e classes internas que estão disponíveis para todos os Programas JavaScript. Isso inclui estutores de dados importantes como Mapas e conjuntos, uma classe de expressão regular para padrão textual Combinando, funções para serializar estruturas de dados de JavaScript em muito mais. Capítulo 12, iteradores e geradores Explica como funciona o for/de loop e como você pode fazer o seu próprios aulas iterable com para/of. Também sobre o gerador funções e a declaração de rendimento. Capítulo 13, JavaScript assíncrono Este capítulo é uma exploração aprofundada de assíncrono Programação em JavaScript, cobrindo retornos de chamada e eventos, APIs baseadas em promessas e as palavras-chave assíncronas e aguardam. Embora a linguagem principal JavaScript não seja assíncrona, APIs assíncronas são o padrão nos navegadores da web e no nó, e este capítulo explica as técnicas para trabalhar com elas APIs. Capítulo 14, Metaprogramação Apresenta vários recursos avançados do JavaScript que podem ser de interesse para programadores que escrevem bibliotecas de código para outros Programadores JavaScript para usar. Capítulo 15, JavaScript em navegadores da Web Apresenta o ambiente do host do navegador da web, explica como a web Os navegadores executam o código JavaScript e abrange o mais importante de As muitas APIs definidas pelos navegadores da Web. Este é de longe o mais longo

Capítulo no livro.Capítulo 16, JavaScript do lado do servidor com nóApresenta o ambiente do host do nó, cobrindo o fundamental modelo de programação e as estruturas de dados e APIs que são mais importantes para entender.Capítulo 17, Ferramentas e extensões JavaScriptCobre ferramentas e extensões de idiomas que valem a pena saber sobrePorque eles são amplamente utilizados e podem torná-lo um mais produtivo programador.1.4 Exemplo: Frequência do personagemHistogramasEste capítulo termina com um programa JavaScript curto, mas não trivial.Exemplo 1-1 é um programa de nó que lê texto da entrada padrão, calcula um histograma de frequência de caracteres desse texto e depois imprime o histograma.Você pode invocar o programa como este para Analise a frequência do caractere de seu próprio código -fonte:\$ node charfreq.js <charfreq.jsT: ##### 11.22%E: ##### 10,15%R: ##### 6,68%S: ##### 6,44%A: ##### 6,16%N: ##### 5,81%O: ##### 5,45%I: ##### 4,54%H: ##### 4,07%C: ### 3,36%L: ### 3,20%U: ### 3,08%/: ### 2,88%

Este exemplo usa vários recursos avançados de JavaScript e é destinado a demonstrar quais programas JavaScript do mundo real podem parecer como. Você não deve esperar entender todo o código ainda, mas estou garantindo que tudo isso será explicado nos capítulos a seguir.

Exemplo 1-1. Histogramas de frequência de caracteres de computação com JavaScript

```
*** Este programa de nó lê o texto da entrada padrão, calcula a frequência* de cada letra nesse texto e exibe um histograma da maioria* de caracteres usados ?? com frequência. Requer o nó 12 ou superior acorrer.** Em um ambiente do tipo Unix, você pode invocar o programa como esse:  
* node charfreq.js <corpus.txt*** Esta classe estende Mapa para que o método get () retorne o especificado// valor em vez de nulo quando a chave não está no mapa. A classe DefaultMap estende o mapa {construtor (defaultValue) {super();// Invoca a Superclass construtor this.defaultValue = defaultValue;// lembre -se do valor padrão}Get (key) {if (this.has (key)) {// se a chave já está no mapa, retorna super.get (chave); // retorna seu valor da superclass.}outro {retornar this.defaultValue;// de outra forma retorna o valor padrão}}
```

```
}// Esta classe calcula e exibe histogramas de frequênciacartaclasse Histograma {construtor ()  
{this.LetterCounts = new DefaultMap (0); // mapa decartas para contarthis.TotalLetters = 0; // Quantosletras em  
tudo} // Esta função atualiza o histograma com as letras detexto.add (text) // remova o espaço em branco do texto  
e converta paramaiúsculatext = text.replace (/s/g, "").toUpperCase (); // agora percorre os personagens do  
textopara (deixe o caráter do texto) {Let count = this.letterCounts.get (caractere); // Obtenha uma contagem  
antigathis.letterCounts.set (caractere, contagem+1); // Incrementá -lothis.TotalLetters ++;} // converte o histograma  
em uma string que exibe um ASCIIgráficotestring () // converte o mapa em uma matriz de matrizes [chave]deixe  
entradas = [... this.letterCounts]; // Classifique a matriz por contagem e depois em ordem alfabeticaentre.sort ((a,  
b) => // uma função paraDefina a ordem de classificação.if (a [1] === b [1]) // se as contagens são iguaisretornar A  
[0] <b [0]?-1: 1; // organizaralfabeticamente.} else // se as contagens
```

```
diferirretornar b [1] - a [1];// classificar por maiorcontar.}});// converte as contagens em porcentagenspara (deixe a  
entrada de entradas) {entrada [1] = entrada [1] / this.ToTalleters*100;}// soltar qualquer entradas inferiores a  
1%entradas = entradas.Filter (Entrada => Entrada [1]> = 1);// agora converte cada entrada em uma linha de  
textoDeixe linhas = entradas.map (([l, n]) => `\$ {l}: \$ {"#". Repita (math.round (n))\$ {n.toFixed (2)%}`);// e retorne as  
linhas concatenadas, separadas porpersonagens newline.retorno linhas.Join ("\ n");});// Esta função Async (Promise  
Returning) cria um histogramaobjeto,// lê de forma assíncrona pedaços de texto da entrada padrão eadiciona  
esses pedaços a// o histograma.Quando chega ao final do fluxo, eleRetorna este histogramaFunção assíncreada  
histogramfromstdin () {process.stdin.setEncoding ("UTF-8");// Leia Unicodecordas, não bytesSeja histograma =  
novo histograma ();para aguardar (let chunk of process.stdin) {histogram.add (pedaço);}histograma de retorno;}//  
Esta linha final de código é o principal corpo do programa.
```

// Faz um objeto histograma a partir de entrada padrão, depois imprime o histograma.histogramfromstdin (). Então
(histograma => {console.log (histograma.toString ())});1.5 ResumoEste livro explica JavaScript de baixo para
cima. Isso significa que nós Comece com detalhes de baixo nível, como comentários, identificadores, variáveis
??tipos; Em seguida, construa expressões, declarações, objetos e funções; e Em seguida, cubra abstrações de
idiomas de alto nível, como classes e módulos. EU leve a palavra definitiva no título deste livro a sério, e o Capítulos
próximos explicam o idioma em um nível de detalhe que pode parecer desanimador no começo. O verdadeiro
domínio do JavaScript requer uma compreensão dos detalhes, no entanto, e espero que você faça Hora de ler a capa
deste livro para capa. Mas por favor não sinta que você Precisa fazer isso em sua primeira leitura. Se você se sentir
sentindo Empolto em uma seção, basta pular para a próxima. Você pode voltar a dominar os detalhes depois de ter
um conhecimento prático da linguagem como um todo.

Capítulo 2. Estrutura lexicalA estrutura lexical de uma linguagem de programação é o conjunto de regras elementares que especificam como você escreve programas nessa linguagem. É a sintaxe de nível mais baixo de um idioma: especifica o que os nomes de variáveis parecem, os caracteres delimitadores para comentários e como uma declaração do programa é separada da próxima, por exemplo. Este curto capítulo documenta a estrutura lexical do JavaScript.

Isto capas:Sensibilidade ao caso, espaços e quebras de linhaComentáriosLiteraisIdentificadores e palavras reservadasUnicodeSemicolons opcionais

2.1 O texto de um programa JavaScript é uma linguagem sensível ao caso. Isso significa que a linguagem Palavras-chave, variáveis, nomes de funções e outros identificadores devem sempre ser digitados com uma capitalização consistente das letras. O tempo A palavra-chave, por exemplo, deve ser digitada "while", não "enquanto" ou "ENQUANTO." Da mesma forma, online, online, online e online são quatro nomes variáveis distintos.

O JavaScript ignora espaços que aparecem entre os tokens nos programas. Para Na maioria das vezes, JavaScript também ignora quebras de linha (mas veja §2.6 para uma exceção). Porque você pode usar espaços e linhas de novo livremente em seus programas, você pode formatar e recuar seus programas em um elegante maneira consistente que facilita a leitura e o entendimento do código. Além do personagem espacial regular (\ U0020), JavaScript também reconhece guias, características de controle ASCII variadas e vários Caracteres do Espaço Unicode como espaço em branco. JavaScript reconhece Newlines, retornos de transporte e uma sequência de retorno/alimentação de linha de transporte como Terminadores de linha.

2.2 Comentários

O JavaScript suporta dois estilos de comentários. Qualquer texto entre // e o fim de uma linha é tratado como um comentário e é ignorado por JavaScript. Qualquer texto entre os caracteres /* e */ também é tratado como um comentário; Esses comentários podem abranger várias linhas, mas podem não ser aninhados. As seguintes linhas de código são todos os comentários legais de JavaScript:

```
// Este é um comentário de linha única.  
/* Este também é um comentário */  
// e aqui está outro comentário.  
/** Este é um comentário de várias linhas. Os personagens extras * em o início de * Cada linha não é uma parte necessária da sintaxe; eles apenas Parece legal! */
```

2.3 Literais

Um literal é um valor de dados que aparece diretamente em um programa. O A seguir, todos os literais:
12 // o número doze
1.2 // o ponto número um dois
"Hello World" // uma série de texto
'Oi' // outra string
verdadeiro // um valor booleano
false // o outro valor booleano
nulo // ausência de um objeto
Detalhes completos sobre literais numéricos e de cordas aparecem no capítulo 3.2.4 Identificadores e palavras reservadas
Um identificador é simplesmente um nome. Em JavaScript, os identificadores são usados ?? para constantes de nome, variáveis, propriedades, funções e classes e para Forneça rótulos para determinados loops no código JavaScript. Um javascriptIdentificador deve começar com uma carta, um sublinhado (_) ou um sinal de dólar (\$). Caracteres subsequentes podem ser cartas, dígitos, sublinhados ou dólarSinais. (Os dígitos não são permitidos como o primeiro caractere para que JavaScript pode facilmente distinguir identificadores dos números.) Estes são todos legaisIdentificadores: e umy_variable_namev13_fictício\$ str Como qualquer idioma, JavaScript se reserva certos identificadores para uso pelolinguagem em si. Essas "palavras reservadas" não podem ser usadas como regularidentificadores. Eles estão listados na próxima seção.

2.4.1 Palavras reservadasAs seguintes palavras fazem parte da linguagem JavaScript.Muitos deestes (como se, enquanto e para) são palavras -chave reservadas que devemnão ser usado como nomes de constantes, variáveis, funções ou classes(embora todos possam ser usados ??como nomes de propriedades dentro de umobjeto).Outros (como de, de, obter e se estabelecer) são usados ??em limitadoscontextos sem ambiguidade sintática e são perfeitamente legais comoidentificadores.Ainda outras palavras -chave (como Let) não podem ser totalmente reservadaspara manter a compatibilidade com versões anteriores com programas mais antigos e, portanto,Existem regras complexas que governam quando podem ser usadas comoidentificadores e quando não podem.(vamos ser usados ??como um nome de variávelse declarado com var fora de uma aula, por exemplo, mas não se declaradodentro de uma aula ou com const.) O curso mais simples é evitar usarqualquer uma dessas palavras como identificadores, exceto para, conjunto e alvo,que são seguros de usar e já estão em uso comum.Como a exportação const obtém o alvo nulovazioAsync continuam se estende se dissoenquantoAguarde o depurador False Import Return Throwcomquebrar o padrão finalmente no set truecolheitaExcluir casos, por exemplo, de tentativa estáticaCatch Do de Let Super TypeofClasse mais funçãoJavaScript também se reserva ou restringe o uso de certas palavras -chave que sãoatualmente não usado pelo idioma, mas isso pode ser usado no futuroversões:Enum implementos Pacote de interface Protegido Private

públicoPor razões históricas, argumentos e avaliação não são permitidos como identificadores em certas circunstâncias e são melhor evitados inteiramente.2.5 UnicodeOs programas JavaScript são escritos usando o conjunto de caracteres Unicode e Você pode usar qualquer caractere unicode em cordas e comentários.Para Portabilidade e facilidade de edição, é comum usar apenas letras ASCII e dígitos em identificadores.Mas esta é apenas uma convenção de programação,e o idioma permite letras, dígitos e ideografias unicode (mas não emojis) em identificadores.Isso significa que os programadores podem usar símbolos e palavras matemáticas de idiomas não ingleses como Constantes e variáveis:const ? = 3,14;const sí = true;2.5.1 Sequências de Escape UnicodeAlguns hardware e software de computador não podem exibir, entrar ou processar corretamente o conjunto completo de caracteres Unicode.Para apoiar programadores e sistemas usando tecnologia mais antiga, o JavaScript define Sequências de fuga que nos permitem escrever caracteres unicode usando apenas Caracteres ASCII.Essas fugas unicode começam com os personagens \ ue são seguidos por exatamente quatro dígitos hexadecimais (usando letras maiúsculas ou minúsculas a - f) ou por um a seis dígitos hexadecimais fechado dentro de aparelhos encaracolados.Essas escapadas de unicode podem aparecer em Javascript String literais, literais regulares de expressão e identificadores (mas

não em palavras -chave do idioma).O Unicode escapa para o personagem "é".Por exemplo, é \ u00e9;Aqui estão três maneiras diferentes de escrever umNome da variável que inclui este personagem:Deixe Café = 1;// Defina uma variável usando um caractere unicodecaf \ u00e9 // => 1;Acesse a variável usando uma fugaSequênciacaf \ u {e9} // => 1;Outra forma da mesma fugaSequênciaAs primeiras versões do JavaScript suportaram apenas a fuga de quatro dígitosSequência.A versão com aparelho encaracolado foi introduzido em ES6 paraapoiar melhor os pontos de codepates unicode que requerem mais de 16 bits, comoComo emoji:console.log ("\" u {1f600}");// imprime um rosto sorridente emojiO Unicode Escapes também pode aparecer nos comentários, mas desde os comentáriossão ignorados, eles são simplesmente tratados como caracteres ASCII nesse contextoe não interpretado como unicode.2.5.2 Normalização unicodeSe você usa caracteres não-ASCII em seus programas JavaScript, vocêdeve estar ciente de que o Unicode permite mais de uma maneira de codificar omesmo personagem.A string "é", por exemplo, pode ser codificada como ocaractere unicode único \ u00e9 ou como um ASCII regular ?e? seguidopelo sotaque agudo combinando Mark \ U0301.Essas duas codificaçõesnormalmente parece exatamente o mesmo quando exibido por um editor de texto, masEles têm codificações binárias diferentes, o que significa que são consideradasDiferente por JavaScript, que pode levar a programas muito confusos:

const Café = 1; // Esta constante é nomeada "Café" const Café = 2; // Esta constante é diferente: "Café" \u{e9} // O padrão Unicode define a codificação preferida para todos os personagens específicos de um procedimento de normalização para converter texto em uma forma canônica adequada para comparações. JavaScript assume que o código-fonte está interpretando já foi normalizada e não faz nenhuma normalização por conta própria. Se você planeja usar caracteres Unicode em seu código JavaScript, você deve garantir que seu editor ou algum outro programa que executa a normalização Unicode do seu código-fonte para prevenir você de acabar com um resultado visualmente indistinguível de outros identificadores.

2.6 Semicolons opcionais

Como muitas linguagens de programação, o JavaScript usa o semicolon (;) para separar declarações (consulte o capítulo 5) um do outro. Isso é importante para deixar claro o significado do seu código: sem um separador, o final de uma declaração pode parecer o começo de uma declaração no próximo, ou vice-versa. Em JavaScript, você geralmente pode omitir o semicolon entre duas declarações se essas declarações forem escritas em linhas separadas. (Você também pode omitir um ponto e vírgula no final de um programa ou se o próximo token no programa for uma cinta curta fechada:). Muitos programadores JavaScript (e o código neste livro) usam semicolons para marcar explicitamente os fins das declarações, mesmo onde elas não estão obrigatórios. Outro estilo é omitir semicolons sempre que possível, usando-os apenas nas poucas situações que os exigem. Qualquer estilo que você

Escolha, existem alguns detalhes que você deve entender sobre opcionais Semicolons em JavaScript. Considere o seguinte código. Desde que as duas declarações aparecem em linhas separadas, o primeiro semicolon pode ser omitido:`a = 3;b = 4;` Escrito da seguinte forma, no entanto, é necessário o primeiro ponto de vírgula:`a = 3;b = 4;` Observe que o JavaScript não trata todas as quebras de linha como um semicolon: Normalmente trata as quebras de linha como semicolons apenas se não conseguir analisar o código sem adicionar um semicolon implícito. Mais formalmente (e com três exceções descritas um pouco mais tarde), JavaScript trata uma quebra de linha como um semicolon se o próximo personagem não espacial não pode ser interpretado como uma continuação da declaração atual. Considere o seguinte código: deixe um `um = 3`; `console.log(a)`. JavaScript interpreta este código como este: deixe um `a = 3`; `console.log(a)`. JavaScript trata a primeira ruptura da primeira linha como um semicolon porque não é possível analisar o código, deixe um `A` sem um semicolon. O segundo poderia ficar sozinho como a afirmação `A;`, mas JavaScript não trata o

Segunda linha quebra como um semicolon porque pode continuar analisando oDeclaracão mais longa a = 3;.Essas regras de rescisão de declaração levam a alguns casos surpreendentes.EsseO código parece duas declarações separadas separadas com uma nova linha:Deixe y = x + f(a+b) .ToString ()Mas os parênteses na segunda linha de código podem ser interpretados como umInvocação de função de F a partir da primeira linha, e JavaScript interpretaO código como este:Seja y = x + f (a + b) .ToString ();É mais provável que não, essa não é a interpretação pretendida peloautor do código.Para trabalhar como duas declarações separadas, umO semicolon explícito é necessário neste caso.Em geral, se uma declaração começa com (, [, /, +, ou -, há uma chanceque isso poderia ser interpretado como uma continuação da declaração antes.Declaracões que começam com /, +e - são bastante raras na prática, masdeclarações começando com (e [não são incomuns, pelo menos emAlguns estilos de programação JavaScript.Alguns programadores gostam de colocarum semicolon defensivo no início de qualquer declaração para quecontinuará a funcionar corretamente, mesmo que a declaração antes sejaModificado e um semicolon de terminaçao anteriormente removido:Seja x = 0 // semicolon omitido aqui; [x, x+1, x+2] .foreach (console.log) // defensivo;mantém issodeclaração separada

Existem três exceções à regra geral que o JavaScript interpreta. A linha quebra como semicolons quando não pode analisar a segunda linha como uma continuação da declaração na primeira linha. A primeira exceção envolve o retorno, jogue, rendimento, quebra e continue declarações (consulte o Capítulo 5). Essas declarações geralmente permanecem sozinhas, mas às vezes, eles são seguidos por um identificador ou expressão. Se uma linha quebra aparece após qualquer uma dessas palavras (antes de qualquer outro tokens), o JavaScript sempre interpretará essa quebra de linha como um semicolon. Para exemplo, se você escrever: retornar verdadeiro; JavaScript pressupõe que você quis dizer: retornar verdadeiro; No entanto, você provavelmente quis dizer: retornar true; Isso significa que você não deve inserir uma quebra de linha entre o retorno, quebre ou continue e a expressão que segue a palavra-chave. Se você insere uma quebra de linha, é provável que seu código falhe de uma maneira que é difícil de depurar. A segunda exceção envolve os operadores ++ e - (§4.8). Esses operadores podem ser operadores de prefixo que aparecem antes de uma expressão ou operadores pós-fix que aparecem depois de uma expressão. Se você quiser usar qualquer um desses operadores como operadores pós-fix, eles devem aparecer no

Mesma linha da expressão a que se aplicam. A terceira exceção envolve funções definidas usando a sintaxe concisa de ?Arrow?: a própria seta => deve aparecer na mesma linha que a lista de parâmetros. 2.7 Resumo Este capítulo mostrou como os programas JavaScript são escritos nivél mais baixo. O próximo capítulo nos leva um passo mais alto e apresenta os tipos e valores primitivos (números, strings e assim por diante) que servem como unidades básicas de computação para programas JavaScript.

Capítulo 3. Tipos, valores e Variáveis

Os programas de computador funcionam manipulando valores, como o número 3.14 ou o texto "Hello World". Os tipos de valores que podem ser representados e manipulados em uma linguagem de programação são conhecidos como tipos, e uma das características mais fundamentais de uma linguagem de programação é o conjunto de tipos que ela suporta. Quando um programa precisa manter um valor para uso futuro, ele atribui o valor a (ou "loja") o valor em uma variável. Variáveis têm nomes e permitem o uso desses nomes em nossos programas para se referir a valores. A maneira como as variáveis trabalham é outra característica fundamental de qualquer programação linguagem. Este capítulo explica tipos, valores e variáveis em JavaScript. Começa com uma visão geral e algumas definições.

3.1 Visão geral e definições

Os tipos de JavaScript podem ser divididos em duas categorias: tipos primitivos e tipos de objetos. Os tipos primitivos de JavaScript incluem números, seqüências de seqüências de texto (conhecido como Strings) e valores da verdade booleana (conhecidos como booleanos). Uma parte significativa deste capítulo é dedicada a um detalhado explicação dos tipos numéricos (§3.2) e string (§3.3) em JavaScript. Booleanos são cobertos em §3.4. Os valores especiais de JavaScript nulos e indefinidos são primitivos.

valores, mas não são números, cordas ou booleanos. Cada valor é normalmente considerado o único membro de seu próprio tipo especial. §3.5 tem mais sobre nulo e indefinido. ES6 adiciona um novo especial-Tipo de propósito, conhecido como símbolo, que permite a definição de linguagemExtensões sem prejudicar a compatibilidade atrasada. Símbolos são Coberto brevemente no §3.6. Qualquer valor de javascript que não seja um número, uma corda, um booleano, umSímbolo, nulo ou indefinido é um objeto. Um objeto (isto é, um membro do objeto de tipo) é uma coleção de propriedades onde cada uma propriedade tem um nome e um valor (um valor primitivo ou outro objeto). Um objeto muito especial, o objeto global, é coberto no §3.7. Mas a cobertura mais geral e mais detalhada dos objetos está no capítulo 6. Um objeto JavaScript comum é uma coleção não ordenada de nomevalores. O idioma também define um tipo especial de objeto, conhecido como umArray, que representa uma coleção ordenada de valores numerados. OA linguagem JavaScript inclui sintaxe especial para trabalhar com matrizes,e as matrizes têm algum comportamento especial que os distingue de objetos comuns. Matrizes são o assunto do capítulo 7. Além de objetos e matrizes básicos, JavaScript define uma série de Outros tipos de objetos úteis. Um objeto definido representa um conjunto de valores. UMO objeto de mapa representa um mapeamento de chaves para valores. Vários ?digitados tipos de matriz ?facilitam operações em matrizes de bytes e outros bináriosdados. O tipo regexp representa padrões textuais e permiteCombinação sofisticada, pesquisando e substituindo operações em strings. O tipo de data representa datas e horários e suporta rudimentardata aritmética. Erro e seus subtipos representam erros que podem surgir

Ao executar o código JavaScript.Todos esses tipos são cobertos em Capítulo 11.JavaScript difere de idiomas mais estáticos nessas funções eAs aulas não fazem apenas parte da sintaxe da linguagem: elas são elas mesmasValores que podem ser manipulados por programas JavaScript.Como qualquerValor de javascript que não é um valor primitivo, funções e classes são um tipo especializado de objeto.Eles são cobertos em detalhes nos capítulos 8e 9.O intérprete JavaScript realiza uma coleção automática de lixo para Gerenciamento de memória.Isso significa que um programador JavaScript geralmente não precisa se preocupar com destruição ou desalocação de objetos ou outros valores.Quando um valor não é mais acessível - quando um programa não tem mais maneira de se referir a ele - o intérprete sabe disso e nunca pode ser usado novamente e recupera automaticamente a memória que foi ocupando.(Os programadores JavaScript às vezes precisam cuidar de garantir que os valores não permaneçam inadvertidamente alcançáveis ??- ePortanto, não reclamável - mais do que o necessário.)O JavaScript suporta um estilo de programação orientado a objetos.Vagamente,Isso significa que, em vez de ter funções definidas globalmente para operar em valores de vários tipos, os próprios tipos definem métodos para trabalhar com valores.Para classificar os elementos de uma matriz A, por exemplo,Não passamos uma função de classificação ().Em vez disso, invocamos o Método de Sort () de A:a.sort ()// A versão orientada ao objeto (a).

A definição do método é abordada no capítulo 9. Tecnicamente, é apenasObjetos JavaScript que têm métodos.Mas números, cordas, booleanos,e os valores de símbolo se comportam como se tivessem métodos.Em JavaScript,nulo e indefinido são os únicos valores que os métodos não podem serinvocou.Os tipos de objetos de JavaScript são mutáveis ??e seus tipos primitivos sãoimutável.Um valor de um tipo mutável pode mudar: um javascriptO programa pode alterar os valores das propriedades do objeto e dos elementos da matriz.Números, booleanos, símbolos, nulos e indefinidos são imutáveis- nem faz sentido falar sobre mudar o valor de umnúmero, por exemplo.Strings podem ser pensadas como matrizes de personagens,E você pode esperar que eles sejam mutáveis.Em JavaScript, no entanto,Strings são imutáveis: você pode acessar o texto em qualquer índice de uma string,Mas o JavaScript não fornece como alterar o texto de uma string existente.As diferenças entre valores mutáveis ??e imutáveis ??são exploradosMais adiante em §3.8.JavaScript converte liberalmente valores de um tipo para outro.Se ao programa espera uma string, por exemplo, e você dá um número, vaiConverta automaticamente o número em uma string para você.E se você usar umvalor não-booleano em que é esperado um booleano, o JavaScript será convertidode acordo.As regras para conversão de valor são explicadas no §3.9.As regras de conversão de valor liberal do Javascript afetam sua definição deigualdade e o operador de igualdade == executa conversões de tipodescrita em §3.9.1.(Na prática, no entanto, o operador de igualdade == édescontinuado a favor do estrito operador de igualdade ===, que não fazDigite conversões.Consulte §4.9.1 para saber mais sobre os dois operadores.)

Constantes e variáveis ??permitem que você use nomes para se referir a valores em seus programas. Constantes são declaradas com const e variáveis ??são declaradas com LET (ou com var no código JavaScript mais antigo). JavaScript Constantes e variáveis ??não são criadas: as declarações não especificam o tipo de valores serão atribuídos. Declaração e atribuição variáveis são cobertas em §3.10. Como você pode ver nesta longa introdução, este é um abrangente capítulo que explica muitos detalhes fundamentais sobre como os dados são representados e manipulados em JavaScript. Começaremos mergulhando certos detalhes dos números e texto do JavaScript.

3.2 números

O número numérico principal do JavaScript, número, é usado para representar inteiros e aproximar números reais. JavaScript representa números usando o formato de ponto flutuante de 64 bits definido pelo IEEE754 padrão, o que significa que pode representar números tão grandes quanto $\pm 1,7976931348623157 \times 10^{308}$ e tão pequenos quanto $\pm 5 \times 10^{-324$. O formato do número JavaScript permite que você represente exatamente todos os inteiros entre -9.007.199.254.740.992 (-2) e 9.007.199.254.740.992 (2), inclusive. Se você usar valores inteiros maiores, isso, você pode perder precisão nos dígitos à direita. Nota, no entanto, que certas operações em JavaScript (como indexação de matrizes e os operadores bitwise descritos no capítulo 4) são realizadas com 32 bits inteiros. Se você precisar representar exatamente números inteiros maiores, consulte §3.2.5. Quando um número aparece diretamente em um programa JavaScript, ele é chamado de 1308-3245353

literal numérico. JavaScript suporta literais numéricos em vários formatos, conforme descrito nas seções a seguir. Observe que qualquer literal numérico pode ser precedido por um sinal menos (-) para tornar o número negativo.

3.2.1 literais inteiros Em um programa JavaScript, um número inteiro base-10 é escrito como uma sequência de dígitos. Por exemplo: 0310000000

Além dos literais inteiros da Base-10, JavaScript reconhece Valores hexadecimais (Base-16). Um literal hexadecimal começa com 0x ou 0X, seguido por uma sequência de dígitos hexadecimais. Um dígito hexadecimal é um dos dígitos de 0 a 9 ou as letras a (ou A) a f (ou F), que representam os valores 10 a 15. Aqui estão exemplos de literais inteiros hexadecimais:

```
0xff // => 255: (15*16 + 15)
0xbadcafe // => 195939070
No ES6 e mais tarde, você também pode expressar números inteiros em binário (base 2) ou octal (base 8) usando os prefixos 0b e 0o (ou 0B e 0O) em vez de 0x:
0B10101 // => 21: (1*16 + 0*8 + 1*4 + 0*2 + 1*1)
0o377 // => 255: (3*64 + 7*8 + 7*1)
```

3.2.2 Literais de ponto flutuante Os literais de ponto flutuante podem ter um ponto decimal; Eles usam o tradicional

Sintaxe para números reais.Um valor real é representado como parte integrante do número, seguido por um ponto decimal e a parte fracionária do número.Os literais de ponto flutuante também podem ser representados usando exponencialNotação: um número real seguido pela letra e (ou E), seguida por um sinal opcional Plus ou Minus, seguido de um expoente inteiro.Esse formato representa o número real multiplicado por 10 ao poder da expoente.Mais sucintamente, a sintaxe é:[dígitos] [. dígitos] [(e | E) [(+|-)] dígitos]Por exemplo:3.142345.6789.333333333333333336.02E23 // 6,02 × 10²³1.4738223E-32 // 1.4738223 × 10⁻³²Separadores em literais numéricosVocê pode usar sublinhados em literais numéricos para dividir literais longos em pedaços que são mais fáceis de ler:Seja bilhão = 1_000_000_000;// ressalta os milhares de separadores.deixe bytes = 0x89_ab_cd_ef;// como um separador de bytes.Deixe bits = 0b0001_1101_0111;// como um separador de mordisba.Deixe a fração = 0,123_456_789;// funciona na parte fracionária também.No momento da redação deste artigo no início de 2020, sublinhamentos em literais numéricos ainda não estão formalmente padronizados como parte do JavaScript.Mas eles estão nos estágios avançados da padronização do processo e são implementados por todos os principais navegadores e por nós.

3.2.3 aritmética em javascriptOs programas JavaScript funcionam com números usando os operadores aritméticos.que o idioma fornece.Isso inclui + para adição, para subtração, * para multiplicação, / para divisão e % para módulo(restante após a divisão).O ES2016 adiciona ** para exponenciação.CompletoDetalhes sobre esses e outros operadores podem ser encontrados no capítulo 4.Além desses operadores aritméticos básicos, JavaScript suporta operações matemáticas mais complexas através de um conjunto de funções e Constantes definidas como propriedades do objeto de matemática:Math.pow (2,53) // => 9007199254740992: 2 para potência 53Math.Round (.6) // => 1,0: arredondado para o mais próximoMath.ceil (.6) // => 1.0: arredondado para um número inteiroMath.floor (.6) // => 0,0: Recupera para um número inteiroMath.abs (-5) // => 5: valor absolutoMath.max (x, y, z) // retorna o maior argumentoMath.min (x, y, z) // retorna o menor argumentoMath.random () // pseudo-random número x onde $0 \leq x < 1$, Math.pi // ?: circunferência de um círculo /diâmetroMath.e // e: a base do naturallogaritmoMath.sqrt (3) // => $3^{0.5}$: a raiz quadrada de 3Math.pow (3, 1/3) // => $3^{(1/3)}$: a raiz do cubo de 3Math.sin (0) // Trigonometria: também Math.cos, Math.atan, etc.Math.log (10) // Logaritmo natural de 10Math.log (100) /math.In10 // base 10 logaritmo de 100Math.log (512) /math.In2 // base 2 logaritmo de 512Math.exp (3) // math.e cubado

ES6 define mais funções no objeto de matemática:Math.cbrt (27) // => 3: raiz de cuboMath.hypot (3, 4) // => 5: raiz quadrada da soma dos quadrados de todos os argumentosMath.Log10 (100) // => 2: Logaritmo Base-10Math.log2 (1024) // => 10: LOGARITHM BASE-2Math.log1p (x) // log natural de (1+x); preciso para muito pequeno xMath.expm1 (x) // math.exp (x) -1; o inverso de Math.log1p ()Math.sign (x) // -1, 0 ou 1 para argumentos <, ==, ou > 0Math.imul (2,3) // => 6: multiplicação otimizada de 32 bitsInteirosMath.clz32 (0xf) // => 28: Número de zero bits líderes em um inteiro de 32 bitsMath.trunc (3.9) // => 3: converter em um número inteiro truncando parte fracionáriaMath.Fround (X) // Round to mais próximo do número de flutuação de 32 bitsMath.sinh (x) // seno hiperbólico.Também Math.Cosh (), Math.tanh ()Math.asinh (x) // arcsina hiperbólica.Também math.acosh (), Math.atanh ()A aritmética em JavaScript não levanta erros em casos de transbordamento, subfluxo, ou divisão por zero. Quando o resultado de uma operação numérica é maior que o maior número representável (estouro), o resultado é um valor especial do infinito, infinito. Da mesma forma, quando o absoluto o valor de um valor negativo se torna maior que o valor absoluto do maior número negativo representável, o resultado é o infinito negativo, -Infinitude. Os valores infinitos se comportam como seria de esperar: acrescentando, subtraindo, multiplicar ou dividindo-os por qualquer coisa resulta em um valor infinito (possivelmente com o sinal revertido). O fluxo ocorre quando o resultado de uma operação numérica está mais próxima de

zero que o menor número representável.Nesse caso, JavaScript retorna 0. Se o fluxo ocorrer de um número negativo, JavaScript retorna um valor especial conhecido como "zero negativo".Este valor é quaseCompletamente indistinguível de Zero e JavaScript regularesOs programadores raramente precisam detectá-lo.Divisão por zero não é um erro no JavaScript: simplesmente retorna o infinito ou infinidade negativa.Há uma exceção, no entanto: zero dividido porZero não tem um valor bem definido e o resultado desta operaçãoé o valor especial, não um número, Nan.Nan também surge se você tentarPara dividir o infinito pelo infinito, pegue a raiz quadrada de um número negativo,ou use operadores aritméticos com operandos não numéricos que não podem serconvertido em números.JavaScript predefine as constantes globais infinitas e a nan para manter oinfinito positivo e valor não um número, e esses valores também sãoDisponível como propriedades do objeto numérico:Infinito // um número positivo muito grande pararepresentarNúmero.positive_infinity // mesmo valor $1/0$ // => infinitoNúmero.max_value * 2 // => infinito;transbordamento-Infinity // um número negativo muito grande pararepresentarNúmero.negative_infinity // o mesmo valor $-1/0$ // => -infinity-Number.max_value * 2 // => -infinityNan // o valor não-numberNúmero.nan // o mesmo valor, escritoOutra maneira $0/0$ // => nanInfinito/infinito // => nan

Número.min_value/2 // => 0: subflow-Number.min_value/2 // => -0: Zero negativo-1/infinito // -> -0: também negativo 0-0// As seguintes propriedades de número são definidas no ES6Número.ParseInt () // O mesmo que o parseint global ()funçãoNúmero.parseFloat () // O mesmo que o parseFloat global ()funçãoNúmero.isnan (x) // é x o valor da nan?Número.isfinite (x) // é x um número e finito?Número.isinteger (x) // é x um número inteiro?Número.issafeInteger (x) // é x um número inteiro -(2 ** 53) <x <2 ** 53?Número.min_safe_integer // => -(2 ** 53 - 1)Número.max_safe_integer // => 2 ** 53 - 1Número.epsilon // => 2 **-52: menor diferença entre númerosO valor não-número de número tem um recurso incomum no JavaScript: ele faz não comparar igual a nenhum outro valor, inclusive a si mesmo. Isso significa isso você não pode escrever x === nan para determinar se o valor de um variável x é nan. Em vez disso, você deve escrever x! = X ou Número.isnan (x). Essas expressões serão verdadeiras se, e somente se, x tem o mesmo valor que a NAN constante global. A função global isNaN () é semelhante ao número.isnan (). Isso retorna verdadeiro se seu argumento for nan, ou se esse argumento é um não valor numérico que não pode ser convertido em um número. O relacionado função número.isfinite () retorna true se seu argumento for um número diferente da NAN, Infinito ou -Infinitude. O globalA função isfinite () retorna true se seu argumento for ou pode ser convertido em um número finito.

O valor zero negativo também é um tanto incomum. Compara igual(mesmo usando o teste estrito de igualdade de JavaScript) para zero positivo, que significa que os dois valores são quase indistinguíveis, exceto quando usado como divisor: Seja zero = 0; // zero regular deixa negz = -0; // Zero negativo zero === negz // => true: zero e zero negativo são igual 1/zero === 1/negz // => false: infinito e -infinity não são igual 3.2.4 Ponto flutuante binário e erros de arredondamento Existem infinitamente muitos números reais, mas apenas um número finito deles (18.437.736.874.454.810.627, para ser exato) podem ser representados Exatamente pelo formato de ponto flutuante JavaScript. Isso significa que quando você está trabalhando com números reais em JavaScript, a representação do número geralmente será uma aproximação do número real. A representação do ponto flutuante IEEE-754 usado por JavaScript (e praticamente qualquer outra linguagem de programação moderna) é um binário Representação, que pode representar exatamente frações como 1/2, 1/8, e 1/1024. Infelizmente, as frações que usamos mais comumente (especialmente quando realizar cálculos financeiros) são decimais Frações: 1/10, 1/100 e assim por diante. Ponto flutuante binário As representações não podem representar exatamente os números tão simples quanto 0,1. Os números de JavaScript têm muita precisão e podem aproximar 0,1 muito de perto. Mas o fato de esse número não pode ser representado Exatamente pode levar a problemas. Considere este código:

Seja $x = 0,3 - .2$ // trinta centavos menos 20 centavos
Seja $y = .2 - .1$ // vinte centavos menos 10 centavos
 $x === y // => false$: os dois valores não são o mesmo!
 $x === .1 // => false$: $.3 - .2$ não é igual a $.1$
 $y === .1 // => true$: $.2 - .1$ é igual a $.1$
Devido ao erro de arredondamento, a diferença entre as aproximações de $.3$ e $.2$ não é exatamente o mesmo que a diferença entre as aproximações de $.2$ e $.1$. É importante entender que isso
O problema não é específico para o JavaScript: afeta qualquer programação idioma que usa números de ponto flutuante binário.
Além disso, observe que os valores x e y no código mostrado aqui estão muito próximos um do outro em comparação ao valor correto. Os valores calculados são adequados para quase qualquer propósito;
O problema só surge quando tentamos comparar valores para igualdade.
Se essas aproximações de ponto flutuante forem problemáticas para o seu programa, considere o uso de números inteiros em escala. Por exemplo, você pode manipular valores monetários como centavos inteiros em vez de fracionários dólares.

3.2.5 Números inteiros de precisão arbitrária com bigint

Uma das características mais recentes do JavaScript, definida no ES2020, é um novo tipo numérico conhecido como bigint. No início de 2020, foi implementado em Chrome, Firefox, Edge e Node.js, e há uma implementação em andamento no Safari. Como o nome indica, Bigint é um tipo numérico cujos valores são inteiros. O tipo foi adicionado ao JavaScript principalmente para permitir a representação de números inteiros de 64 bits, que são necessários para a compatibilidade com muitas outras linguagens de programação.

e APIs. Mas valores de bigint podem ter milhares ou até milhões de dígitos, se você precisa trabalhar com números tão grandes. (Observação, No entanto, que as implementações do BIGINT não são adequadas para criptografia porque eles não tentam evitar ataques de tempo.) Os literais bigint são escritos como uma série de dígitos seguidos por um minúsculo `n`. Por padrão, estão na base 10, mas você pode usar o `0b`, `0o` e `0x` prefixos para bigints binários, octais e hexadecimais: `1234n` // um literal não-so-big bigint `0B111111n` // um bigint binário `0o7777n` // um bigint octal `0x8000000000000000n` // => `2n ** 63n`: um número inteiro de 64 bits. Você pode usar o `BigInt()` como uma função para converter regularmente Números de JavaScript ou Strings para valores Bigint: `BigInt(número.max_safe_integer)` // => `9007199254740991N`. Seja `String = "1" + "0" .repeat(100);` // 1 seguido por 100 zeros. `BigInt(string)` // => `10n ** 100n`: um Googol! Aritmética com valores bigint funciona como aritmética com regular Números de JavaScript, exceto que a divisão cai qualquer restante e é recunda (em direção a zero): `1000n + 2000n` // => `3000n` `3000N - 2000N` // => `1000N` `2000n * 3000n` // => `6000000n` `3000N / 997N` // => `3N`: O quociente é `33000n % 997n` // => `9n`: e o restante é `9(2n ** 131071n) - 1n` // Um ?? Mersenne Prime com 39457 decimal dígitos

Embora os operadores padrão +, -, *, /, % e ** trabalhem com Bigint, é importante entender que você não pode misturar operandos de tipo BIGINT com operando de números regulares. Isso pode parecer confuso em primeiro lugar, mas há uma boa razão para isso. Se um tipo numérico fosse mais geral que o outro, seria fácil definir aritmética em mistooperando para simplesmente retornar um valor do tipo mais geral. Mas também não é mais geral que o outro: Bigint pode representar extraordinariamente valores grandes, tornando-o mais geral que os números regulares. Mas Bigints só pode representar números inteiros, fazendo o tipo de número JavaScript comum mais geral. Não há como contornar esse problema, então JavaScript contaminam -o simplesmente não permitindo operandos mistos para a aritmética operadores. Operadores de comparação, por outro lado, trabalham com tipos numéricos mistos (mas consulte §3.9.1 para obter mais informações sobre a diferença entre == e ===):

```
1 < 2n // => true
2 > 1n // => true
0 == 0n // => true
0 === 0n // => false
```

O === verifica o tipo de igualdade com o bem os operadores bitwise (descritos em §4.8.3) geralmente funcionam com Bigint operando. Nenhuma das funções do objeto de matemática aceita bigint operando, no entanto.

3.2.6 Data e horários

JavaScript define uma classe de data simples para representar e manipular os números que representam datas e horários. JavaScript Data são objetos, mas também têm uma representação numérica como um

Timestamp que especifica o número de milissegundos decorridos desde 1 de janeiro de 1970: deixe timestamp = date.now ()// a hora atual como um timestamp (um número). deixe agora = new Date ()// a hora atual como uma data objeto. deixe ms = agora.getTime ()// converter em um milissegundo timestamp. vamos iso = agora.toISOString ()// converter em uma string em formato padrão. A classe de data e seus métodos são abordados em detalhes no §11.4. Mas nós veremos os objetos da data novamente em §3.9.3 quando examinarmos os detalhes de Conversões do tipo JavaScript.

3.3 Texto O tipo JavaScript para representar o texto é a string. Uma corda é uma sequência ordenada imutável de valores de 16 bits, cada um dos quais normalmente representa um caractere unicode. O comprimento de uma corda é o número de valores de 16 bits que ela contém. As cordas de JavaScript (e suas matrizes) usam zero-indexação baseada: o primeiro valor de 16 bits está na posição 0, o segundo em posição 1 e assim por diante. A sequência vazia é a sequência do comprimento 0. JavaScript não tem um tipo especial que represente um único elemento de uma corda. Para representar um único valor de 16 bits, basta usar uma string que tem um comprimento de 1. Personagens, pontos de código e strings de javascript JavaScript usa a codificação UTF-16 do conjunto de caracteres unicode, e as strings JavaScript são sequências de valores de 16 bits não assinados. Os caracteres unicode mais usados ??(aqueles do?Plano multilíngue básico?) têm pontos de código que se encaixam em 16 bits e podem ser representados por um elemento de uma corda. Caracteres unicode cujos pontos de código não se encaixam em 16 bits são codificados usando as regras de

UTF-16 como uma sequência (conhecida como "par substituta") de dois valores de 16 bits. Isso significa que um JavaScript String of Comprimento 2 (dois valores de 16 bits) pode representar apenas um único caractere unicode: deixe euro = "?"; Let Love = "?"; Euro.length // => 1: Este personagem tem um elemento de 16 bits Love.Length // => 2: UTF-16 A codificação de ? é "\ud83d\udc99" A maioria dos métodos de manipulação de cordas definidos pelo JavaScript opera em valores de 16 bits, não caracteres. Eles não tratam pares substitutos, especialmente, não realizam normalização da corda e nem mesmo verifique se uma string está bem formada UTF-16. No ES6, no entanto, as cordas são iteráveis, e se você usar o loop ou ... operador com uma corda, ele iterará os caracteres reais da string, não os valores de 16 bits.

3.3.1 Literais de cordas Para incluir uma string em um programa JavaScript, basta incluir os caracteres da string dentro de um par correspondente de aspas simples ou duplas ou backticks ('ou "ou`). Backticks podem estar contidos em strings delimitados por uma única citação ou personagens, e da mesma forma para as cordas delimitadas por citações duplas e backticks. Aqui estão exemplos de literais de cordas:

```
"" // a sequência vazia: tem zero caracteres
'Teste'
"3.14"
"nome = "myform"
"Você não prefere o livro de O'Reilly?"
"? é a proporção da circunferência de um círculo e seu raio"
``Ela disse 'oi' ``
```

Strings delimitadas com backticks são uma característica do ES6 e permitem expressões JavaScript a serem incorporadas (ou interpoladas em) string literal. Esta sintaxe de interpolação de expressão é coberta em §3.3.4. As versões originais do JavaScript exigiam que os literais fossem escritos

em uma única linha, e é comum ver o código JavaScript que cria strings longas, concatenando strings de linha única com o operador +. Com o ES5, no entanto, você pode quebrar uma corda literal em várias linhas portando cada linha, mas a última com uma barra de barra (\). Nem a barra de barriga nem o terminador de linha que o segue faz parte da string literal. Se você precisa incluir um personagem de nova linha em um citado ou duplo string literal, use a sequência de caracteres \n (documentada no próximo seção). A sintaxe do backtick ES6 permite que as strings sejam quebradas em várias linhas e, neste caso, os terminadores de linha fazem parte do string literal:// Uma string representando 2 linhas escritas em uma linha:'Duas \n lines'// Uma sequência de uma linha escrita em 3 linhas:"um\longo\linha"// Uma sequência de duas linhas escrita em duas linhas:`O personagem Newline no final desta linha está incluído literalmente nesta string` Observe que quando você usa citações únicas para delimitar suas cordas, você deve ter cuidado com as contrações e possessivos em inglês, como não pode ser O'Reilly's. Como o apóstrofo é o mesmo que o único personagem, você deve usar o personagem de barra de barragem (\) para "escapar" de qualquer apostóficos que aparecem em cordas de citação única (escapadas são explicadas na próxima seção). Na programação JavaScript do lado do cliente, o código JavaScript pode conter strings de código HTML e código HTML podem conter sequências de sequências de código JavaScript. Como o JavaScript, o HTML usa um único ou duplo

Citações para delimitar suas cordas. Assim, ao combinar JavaScript e HTML, é uma boa ideia usar um estilo de citações para JavaScript e o outro estilo para HTML. No exemplo seguinte, a string ?Obrigado! você "é citado único em uma expressão de JavaScript, que é então duas citadas dentro de um atributo HTML Eventyler:<button onclick = "alert ('obrigado')"> clique em mim </botão>3.3.2 Sequências de fuga em literais de cordas O caractere de barragem (\) tem um propósito especial em strings de javascript. Combinado com o personagem que o segue, ele representa um personagem! Isso não é representável de outra forma dentro da string. Por exemplo, \n é uma sequência de fuga que representa um personagem de nova linha. Outro exemplo, mencionado anteriormente, é a fuga, que representa o caractere de citação única (ou apóstrofe). Esta sequência de fuga é útil quando você precisa incluir um apóstrofe em um literal de corda que seja contido em citações únicas. Você pode ver por que eles são chamados Sequências de fuga: a barra de barriga permite que você escape do habitual interpretação do caractere de uma quitalha. Em vez de usá-lo para marcar o final da string, você a usa como um apóstrofo: 'Você está certo, não pode ser uma citação' A Tabela 3-1 lista as sequências de escape JavaScript e os personagens que eles representam. Três sequências de fuga são genéricas e podem ser usadas para representar qualquer caractere especificando seu código de caracteres unicode como um número hexadecimal. Por exemplo, a sequência \xa9 representa o símbolo de direitos autorais, que tem o unicode codificação dada pelo

Número hexadecimal A9. Da mesma forma, o \ u escape representa um caractere de unicode arbitrário especificado por quatro dígitos hexadecimais ou uma cinco dígitos quando os dígitos estão fechados em aparelhos encaracolados: \ u03c0 representa o caractere ?, por exemplo, e \ u {1f600} representa o emoji de "rosto sorridente". Tabela 3-1. Sequências de fuga de JavaScript Sequência Personagem representado\ O caractere nu (\ u0000)\ b Backspace (\ u0008)\ t Guia horizontal (\ u0009)\ n Newline (\ u000a)\ v Guia vertical (\ u000b)\ f Formulário de formulário (\ u000c)\ r Retorno de carregamento (\ u000d)\ "Citação dupla (\ u0022)\ 'Apóstrofo ou cotação única (\ u0027)\ \ Barragem (\ u005c)\ xnn O caractere unicode especificado pelos dois dígitos hexadecimais nn\ Unnnn O caractere unicode especificado pelos quatro dígitos hexadecimais nnnn\un} O caractere unicode especificado pelo codepoint n, onde n é um a seis dígitos hexadecimais entre 0 e 10ffff (ES6)

Se o caractere \ preceder qualquer personagem que não seja o mostrado em Tabela 3-1, a barra de barriga é simplesmente ignorada (embora versões futuras do idioma podem, é claro, definir novas sequências de fuga). Para exemplo, \# é o mesmo que #. Finalmente, como observado anteriormente, o ES5 permite um barragem antes de uma quebra de linha para quebrar uma corda literal em várias linhas.

3.3.3 Trabalhando com strings

Um dos recursos internos do JavaScript é a capacidade de concatenar cordas. Se você usar o operador + com números, ele os adicionará. Mas se você usa este operador em strings, ele se junta a eles anexando o segundo para o primeiro. Por exemplo:

```
msg = "Olá", "mundo"; // produz a corda "Olá, mundo"
```

Deixe cumprimentar = "Bem -vindo ao meu blog", "" + nome; As strings podem ser comparadas com o padrão == Equality e != Operadores de desigualdade: duas cordas são iguais se e somente se consistirem exatamente a mesma sequência de valores de 16 bits. Strings também podem ser comparados com os operadores <, <=, > e >=. Comparação de string é feita simplesmente comparando os valores de 16 bits. (Para localidade mais robusta-Comparação e classificação de strings, consulte §11.7.3.) Para determinar o comprimento de uma corda - o número de 16 bits que ela contém - use a propriedade de comprimento da string: S. Length

Além desta propriedade de comprimento, o JavaScript fornece uma API rica para trabalhar com strings: Seja s = "Olá, mundo"; // Comece com algum texto. // obtendo partes de uma corda. substring (1,4) // => "ell": o 2º, 3º e 4º caracteres. s.slice (1,4) // => "ell": a mesma coisa. slice (-3) // => "rld": últimos 3 caracteres. S.Split (",") // => ["Hello", "World"] : dividido em string delimiter // pesquisando uma string. S.IndexOF ("L") // => 2: Posição da primeira letra L. S.IndexOF ("L", 3) // => 3: posição de primeiro "L" em ou depois de 3. S.IndexOF ("zz") // => -1: s não inclui o Substring "ZZ". S.LastIndexOf ("L") // => 10: Posição da última letra L // funções de pesquisa booleana no ES6 e mais tardes. startswith ("inferno") // => true: a string começa com essa. endswith ("!") // => false: s não termina com isso. includes ("ou") // => true: s inclui substring "ou" // Criando versões modificadas de uma string. S.Replace ("LLO", "YA") // => "Heya, mundo". s.toLowerCase () // => "Olá, mundo". S.ToUppercase () // => "Olá, mundo". s.Normalize () // Unicode NFC Normalização: ES6. S.Normalize ("NFD") // NFD Normalização. Também "nfkc", "Nfkd" // Inspecionando caracteres individuais (16 bits) de uma string. S.Charat (0) // => "H": o primeiro caractere. S.Charat (S.Length-1) // => "D": o último caractere. S.charCodeAt (0) // => 72: número de 16 bits na posição especificada. S.CodePointAt (0) // => 72: ES6, trabalha para pontos CodePoints >

16 bits// Funções de preenchimento de string no ES2017 "x" .padstart (3) // => "x": adicione espaços à esquerda para um comprimento de 3 "X" .Padend (3) // => "x": adicione espaços à direita para um comprimento de 3 "x" .padstart (3, "*") // => "*** x": adicione estrelas à esquerda com um comprimento de 3 "x" .padend (3, "-") // => "x--": adicione traços à direita para um comprimento de 3// Funções de corte de espaço.Trim () é ES5; Outros ES2019 "teste" .Trim () // => "teste": remova os espaços no início e fim "teste" .Trimstart () // => "teste": remova os espaços à esquerda. Também trimleft "Teste" .TrimLeft () // => "teste": remova os espaços à esquerda. Também Trimright// métodos de string diversos.concat ("!") // => "Olá, mundo!": Apenas use + operador em vez disso "<>". Repita (5) // => "<> <> <> <> <>": concatenar cópias. ES6 Lembre -se de que as cordas são imutáveis ?? em JavaScript. Métodos como substituir () e touppercase () retornar novas strings: eles não modifique a sequência na qual eles são chamados. Strings também podem ser tratadas como matrizes somente leitura, e você pode acessar caracteres individuais (valores de 16 bits) de uma string usando quadradoSuportes em vez do método Charat (): Seja s = "Olá, mundo"; s [0] // => "h" s [s.Length-1] // => "D"

3.3.4 Literais de modelo No ES6 e posterior, os literais de cordas podem ser delimitados com backticks: Seja S = `Hello World`; Isso é mais do que apenas mais uma sintaxe literal de cordas, no entanto, porque esses literais de modelo podem incluir expressões arbitrárias de JavaScript. O valor final de uma string literal em backticks é calculado por avaliar quaisquer expressões incluídas, convertendo os valores daquele expressões para cordas e combinar as cordas calculadas com o Personagens literais dentro dos backticks: deixe o nome = "Bill"; deixe saudação = `Olá \${name}`; // saudação == "Olá Conta." Tudo entre o \${e a correspondência} é interpretado como um Expressão de JavaScript. Tudo fora do aparelho encaracolado é normal Texto literal de cordas. A expressão dentro do aparelho é avaliada e depois convertido em uma corda e inserido no modelo, substituindo o símbolo de dólar, os aparelhos encaracolados e tudo entre eles. Um modelo literal pode incluir qualquer número de expressões. Pode usar Qualquer um dos personagens de fuga que as cordas normais podem, e pode abranger Qualquer número de linhas, sem nenhuma escapada especial necessária. A seguir Modelo literal inclui quatro expressões JavaScript, um Unicode Escapesequência, e pelo menos quatro novas linhas (os valores de expressão podem incluir NEWLINES também): Deixe errorMessage = `\\ U2718 Falha no teste em \${fileName}: \${lineNumber}: \${exception.message}`

Stack Trace:\$ {expcion.stack}`;A barra de barriga no final da primeira linha aqui escapa do inicialnewline para que a string resultante comece com o unicode ?Personagem (\ U2718) em vez de uma nova linha.Literais de modelo marcadosUm recurso poderoso, mas menos comumente usado, dos literais de modelo é que,Se um nome de função (ou "tag") ocorre logo antes do backtick de abertura,então o texto e os valores das expressões dentro do modeloLiterais são passados ??para a função.O valor deste ?modelo marcadoliteral ?é o valor de retorno da função.Isso pode ser usado, para exemplo, para aplicar HTML ou SQL escapando para os valores antessubstituindo -os no texto.O ES6 possui uma função de tag embutida: string.raw ().Ele retorna o textoDentro de backticks sem nenhum processamento de backslash escapas:`\ n` .length // => 1: a string tem um únicopersonagem newlineString.raw` \ n` .length // => 2: um caractere de barriga e oCarta nObserve que, embora a parte da tag de um modelo marcado literal seja umFunção, não há parênteses usados ??em sua invocação.Neste mesmoCaso específico, os caracteres de backtick substituem o aberto e o fechamentoparênteses.A capacidade de definir suas próprias funções de tag de modelo é um poderoso

Recurso do JavaScript. Essas funções não precisam devolver strings e Eles podem ser usados ?? como construtores, como se definisse uma nova sintaxe literal para o idioma. Veremos um exemplo no §14.5.3.3.5 correspondência de padrões. JavaScript define um tipo de dados conhecido como expressão regular (ou RegExp) para descrever e corresponder padrões em seqüências de texto. Os regexps não são um dos tipos de dados fundamentais em JavaScript, mas Eles têm uma sintaxe literal como números e strings, então eles Às vezes parece que eles são fundamentais. A gramática de regularOs literais de expressão são complexos e a API que eles definem não é trivial. Eles estão documentados em detalhes no §11.3. Porque os regexps são poderoso e comumente usado para processamento de texto, no entanto, esta seção Fornece uma breve visão geral. O texto entre um par de barras constitui uma expressão regular literal. A segunda barra no par também pode ser seguida por um ou mais letras, que modificam o significado do padrão. Por exemplo: ^Html// corresponde às letras h t m l no início de uma corda/[1-9] [0-9]*// corresponde a um dígito diferente de zero, seguido por Qualquer número de dígitos \ b javascript \ b / i // corresponde a "javascript" como uma palavra, caso-insensível Objetos regexp definem vários métodos úteis e strings também ter métodos que aceitam argumentos regexp. Por exemplo: deixe texto = "Teste: 1, 2, 3"; // Texto da amostra deixa padrão = \ d+ / g // corresponde a todas as instâncias de um ou mais dígitos

Pattern.test (texto) // => True: Existe uma correspondênciatext.search (padrão) // => 9: posição do primeirocorrespondertext.match (padrão) // => ["1", "2", "3"] matrizde todas as partidastext.replace (padrão, "#") // => "teste: #, #, #"text.split (/d+/) // => ["", "1", "2", "3"] dividido em não "3.4 valores booleanosUm valor booleano representa a verdade ou a falsidade, dentro ou fora, sim ou não.Existem apenas dois valores possíveis desse tipo.As palavras reservadasVerdadeiro e falso avaliar para esses dois valores.Os valores booleanos são geralmente o resultado de comparações que você faz emSeus programas JavaScript.Por exemplo:a === 4Este código testa para ver se o valor da variável A é igual aonúmero 4. Se for, o resultado dessa comparação é o valor booleanoverdadeiro.Se A não for igual a 4, o resultado da comparação é falso.Os valores booleanos são comumente usados ??nas estruturas de controle de JavaScript.Por exemplo, a instrução if/else em JavaScript executa umação se um valor booleano for verdadeiro e outra ação se o valor forfalso.Você geralmente combina uma comparação que cria um booleanovalor diretamente com uma declaração que a usa.O resultado é assim:if (a === 4) {b = b + 1;

} outro {a = a + 1;}Este código verifica se A é igual a 4. Se sim, adiciona 1 a B; Caso contrário, adiciona 1 a a.Como discutiremos no §3.9, qualquer valor de JavaScript pode ser convertido em um valor booleano.Os seguintes valores se convertem para e, portanto, funcionamTipo, falso: indefinido nulo 0-0NaN "" // a corda vaziaTodos os outros valores, incluindo todos os objetos (e matrizes) se convertem e trabalhamTipo, verdadeiro.false, e os seis valores que se convertem para ele, às vezes são chamados valores falsamente, e todos os outros valores são chamados de verdade.A qualquer momento JavaScript espera um valor booleano, um valor falsamente funciona como falseE um valor verdadeiro funciona como verdadeiro.Como exemplo, suponha que a variável O seja mantida um objeto ou o valor nulo.Você pode testar explicitamente para ver se o não é nulo com uma declaração como esta:if (o! == NULL) ...O operador não equal! == compara o a nulo e avalia verdadeiro ou falso.Mas você pode omitir a comparação e, em vez disso,

Confie no fato de NULL ser falsamente e objetos são verdadeiros: se (o) ... No primeiro caso, o corpo do IF será executado apenas se O não for nulo. O segundo caso é menos rigoroso: ele executará o corpo do seSomente se O não for falso ou qualquer valor falsamente (como nulo ou indefinido). Qual afirmação se for apropriada para o seu programarealmente depende de quais valores você espera ser atribuído a o. Se vocêprecisa distinguir nulo de 0 e "", então você deve usar umcomparação explícita. Os valores booleanos têm um método `ToString()` que você pode usarconverte -os para as cordas "verdadeiras" ou "falsas", mas elas não têm nenhumOutros métodos úteis. Apesar da API trivial, existem três importantesoperadores booleanos. O operador `&&` executa o booleano e a operação. Ele avaliaum valor verdadeiro se e somente se ambos os seus operandos forem verdadeiros; Avalia para um valor falsamente de outra forma. O `||`operador é o booleano ouoperação: ele avalia um valor verdadeiro se um (ou ambos) de seusoperands é verdade e avalia um valor falsamente se ambos os operando forem falsidade. Finalmente, o unário`!` O operador executa o booleano nãoOperação: Avalia -se para TRUE se seu operando for falsamente e avaliarFalso se o seu operando for verdade. Por exemplo:
`if ((x == 0 && y == 0) ||! (z == 0)) { // x e y são zero ou z é diferente de zero}`

Detalhes completos sobre esses operadores estão no §4.10.3.5 NULL e indefinido. null é uma palavra-chave do idioma que avalia um valor especial que é geralmente usado para indicar a ausência de um valor. Usando o tipo de operador no NULL retorna a string "Objeto", indicando que o NULL pode ser pensado como um valor de objeto especial que indica "nenhum objeto". Em prática, no entanto, nulo é tipicamente considerado como o único membro de seu próprio tipo, e pode ser usado para indicar "sem valor" para números e strings e objetos. A maioria das linguagens de programação tem uma equivalente ao nulo de JavaScript: você pode estar familiarizado com ele como nulo, nil, ou nenhum. O JavaScript também possui um segundo valor que indica ausência de valor. O valor indefinido representa um tipo mais profundo de ausência. É o valor de variáveis ?? que não foram inicializadas e o valor que você obtém quando você consulta o valor de uma propriedade de objeto ou elemento de matriz que não existe. O valor indefinido também é o valor de retorno das funções que não retornem explicitamente um valor e o valor dos parâmetros de função para que nenhum argumento é aprovado. indefinido é um global predefinido constante (não uma palavra-chave de idioma como nula, embora isso não seja uma distinção importante na prática) que é inicializada para o valor indefinido. Se você aplicar o operador TIPEOF ao valor indefinido, ele retorna "indefinido", indicando que esse valor é o único membro de um tipo especial. Apesar dessas diferenças, nulos e indefinidos indicam um

ausência de valor e geralmente pode ser usada de forma intercambiável. A igualdade operador == considera -os iguais.(Use a igualdade estrita operador === para distinguir -los.) Ambos são valores falsamente: eles se comportam como false quando um valor booleano é necessário.Nem nulo nem undefined têm propriedades ou métodos.De fato, usando ou [] para Acesse uma propriedade ou método desses valores causa um TypeError.Considero undefined para representar um nível de sistema, inesperado, ou ausência de valor e nulo para representar um nível de programa,Ausência normal ou esperada de valor.Evito usar nulo e undefined quando posso, mas se eu precisar atribuir um desses valores a uma variável ou propriedade ou passar ou retornar um desses valores para ou de uma função, eu geralmente uso nulo.Alguns programadores se esforçam para evitar nulos inteiramente e use undefined em seu lugar onde puder.3.6 SímbolosOs símbolos foram introduzidos no ES6 para servir como nomes de propriedades que não são de corda.Para entender os símbolos, você precisa saber que o JavaScript's Tipo de objeto fundamental é uma coleção não ordenada de propriedades,onde cada propriedade tem um nome e um valor.Os nomes de propriedades são tipicamente (e até ES6, era exclusivamente) strings.Mas em ES6 e Mais tarde, os símbolos também podem servir a esse propósito:Seja strname = "Nome da string";// uma string para usar como um nome da propriedade e symname = símbolo ("propname");// um símbolo para usar como um nome da propriedade typeof strname // => "string": strname é uma corda typeof symname // => "símbolo": symname é

um símboloSeja o = {};// Crie um novo objetoo [strname] = 1;// Defina uma propriedade com umNome da stringo [symname] = 2;// Defina uma propriedade com umNome do síboloo [strname] // => 1: Acesse a string-propriedade nomeadao [symname] // => 2: Acesse o símbolo-propriedade nomeadaO tipo de símbolo não possui uma sintaxe literal.Para obter um síbolovalor, você chama a função Symbol ().Esta função nunca retornaO mesmo valor duas vezes, mesmo quando chamado com o mesmo argumento.Essesignifica que, se você chama Symbol () para obter um valor de símbolo, você podeuse com segurança esse valor como nome de propriedade para adicionar uma nova propriedade a umobjeto e não precisa se preocupar com o fato de você estar substituindo umPropriedade existente com o mesmo nome.Da mesma forma, se você usa simbólicosnomes de propriedades e não compartilham esses símbolos, você pode estar confianteque outros módulos de código em seu programa não accidentalmentesubstitua suas propriedades.Na prática, os símbolos servem como um mecanismo de extensão de linguagem.QuandoO ES6 introduziu os objetos for/of loop (§5.4.4) e iterável(Capítulo 12), precisava definir o método padrão que as classes poderiamimplementar para se tornar iterável.Mas padronizando qualquernome de string específico para este método de iterador teria quebradoCódigo existente, então um nome simbólico foi usado.Como veremos emCapítulo 12, Symbol.iterator é um valor de símbolo que pode ser usadocomo um nome de método para tornar um objeto iterável.A função Symbol () pega um argumento de string opcional e retorna

um valor único de símbolo. Se você fornecer um argumento de string, essa string irá ser incluída na saída do método `ToString()` do símbolo. Observe, no entanto, que chamando `símbolo()` duas vezes com a mesma corda produz dois valores de símbolo completamente diferentes. Seja `s = símbolo("sym_x"); s.ToString() // => "Symbol(sym_x)"`. `ToString()` é o único método interessante de instâncias de símbolos. Existem duas outras funções relacionadas ao símbolo que você deve conhecer, no entanto. Às vezes, ao usar símbolos, você deseja mantê-los privado para seu próprio código para que você tenha uma garantia de que suas propriedades nunca entrarão em conflito com as propriedades usadas por outro código. Outras vezes, no entanto, você pode querer definir um valor de símbolo e compartilhá-lo amplamente com outro código. Este seria o caso, por exemplo, se você fosse definindo algum tipo de extensão que você queria que outro código fosse capaz de participar, como no mecanismo de símbolo.iterator descrito anteriormente. Para servir este último caso de uso, JavaScript define um símbolo global registro. A função `symbol.for()` leva um argumento de string e retorna um valor de símbolo associado à string que você passa. Se não o símbolo já está associado a essa string, então um novo é criado e retornou; Caso contrário, o símbolo já existente é retornado. Que é, a função `símbolo.for()` é completamente diferente do `Symbol()`. Função: `Symbol()` nunca retorna o mesmo valor duas vezes, mas `symbol.for()` sempre retorna o mesmo valor quando chamado com a mesma corda. A string passada para `symbol.for()` aparece no saída de `toString()` para o símbolo retornado, e também pode ser

Recuperado chamando Symbol.keyFor () no símbolo retornado.Seja s = símbolo.for ("compartilhado");Seja t = símbolo.for ("compartilhado");s === t // => true.toString () // => "Símbolo (compartilhado)"Symbol.key para (t) // => "compartilhado"3.7 O objeto globalAs seções anteriores explicaram os tipos primitivos de JavaScript valores.Tipos de objetos - objetos, matrizes e funções - são cobertos em capítulos próprios mais tarde neste livro.Mas há um muito importanteValor do objeto que devemos cobrir agora.O objeto global é regularObjeto JavaScript que serve a um propósito muito importante: as propriedades deste objeto são os identificadores definidos globalmente que estão disponíveis para um programa JavaScript.Quando o intérprete JavaScript inicia (ou sempre que um navegador da web carrega uma nova página), ele cria um novo global objeto e fornece um conjunto inicial de propriedades que definem:Constantes globais como indefinidas, infinito e nanFunções globais como isNaN (), parseInt () (§3.9.2) e eval () (§4.12)Funções de construtor como date (), regexp (), string (), Object () e Array () (§3.9.2)Objetos globais como Math e Json (§6.8)As propriedades iniciais do objeto global não são palavras reservadas, mas elas merecem ser tratados como se fossem.Este capítulo já descreveu algumas dessas propriedades globais.A maioria dos outros será

coberto em outros lugares deste livro. No nó, o objeto global tem uma propriedade chamada global cujo valor é o próprio objeto global, então você sempre pode se referir ao objeto global pelo nome Global in Node Programs. Nos navegadores da web, o objeto da janela serve como objeto global para todos os Código JavaScript contido na janela do navegador que ele representa. Esse objeto de janela global possui uma propriedade de janela auto-referencial que pode ser usado para se referir ao objeto global. O objeto da janela define as Propriedades globais centrais, mas também define alguns outros globais que são específicos para navegadores da Web e JavaScript do lado do cliente. Trabalhador da webthreads (§15.13) têm um objeto global diferente da janela com que eles estão associados. O código em um trabalhador pode se referir ao seu global objeto como eu. O ES2020 finalmente define global, essa maneira padrão de se referir ao objeto global em qualquer contexto. No início de 2020, esse recurso foi implementado por todos os navegadores modernos e por nós.

3.8 valores primitivos imutáveis ??e Referências de objetos mutáveis

Existe uma diferença fundamental no JavaScript entre primitivovalores (indefinidos, nulos, booleanos, números e cordas) e objetos (incluindo matrizes e funções). Primitivos são imutáveis: Não há como mudar (ou "mutar") um valor primitivo. Isso é óbvio para números e booleanos - nem faz sentido

altere o valor de um número. Não é tão óbvio para as cordas, no entanto. Como as cordas são como matrizes de personagens, você pode esperar ser capazPara alterar o caractere em qualquer índice especificado. De fato, JavaScript nãoPermitir isso, e todos os métodos de string que parecem retornar uma string modificadaestão, de fato, retornando um novo valor de string. Por exemplo: Seja s = "olá"; // Comece com algum texto em minúsculas S.Touppercase (); // retorna "Hello", mas não altera Ss // => "Hello": a string original não temmudado Os primitivos também são comparados pelo valor: dois valores são os mesmos apenas seEles têm o mesmo valor. Isso parece circular para números, booleanos,nulo e indefinido: não há outra maneira de eles podercomparado. Novamente, no entanto, não é tão óbvio para as cordas. Se doisOs valores distintos de string são comparados, JavaScript os trata como igual se,e somente se eles tiverem o mesmo comprimento e se o personagem em cada índiceé o mesmo. Objetos são diferentes dos primitivos. Primeiro, eles são mutáveis ??- seuOs valores podem mudar: Seja o = {x: 1}; // Comece com um objeto O.x = 2; // sofre -o alterando o valor de umpropriedade O.Y = 3; // sofre novamente adicionando um novopropriedade Seja a = [1,2,3]; // matrizes também são mutáveisa [0] = 0; // altere o valor de um elemento de matriz a [3] = 4; // Adicione um novo elemento de matriz Os objetos não são comparados pelo valor: dois objetos distintos não são iguais

Mesmo que eles tenham as mesmas propriedades e valores. E dois distinosmatrizes não são iguais, mesmo que tenham os mesmos elementos no mesmoordem: Seja o = {x: 1}, p = {x: 1};// dois objetos com o mesmopropriedadeso === p // => false: objetos distintosnunca são iguaisSeja a = [], b = [];// duas matrizes distintas e vaziasa === b // => false: matrizes distintassão nunca igualOs objetos às vezes são chamados de tipos de referência para distingu -los deTipos primitivos de JavaScript. Usando essa terminologia, os valores dos objetos sãoreferências, e dizemos que os objetos são comparados por referência: doisOs valores dos objetos são os mesmos se e somente se eles se referirem ao mesmoobjeto subjacente.deixe A = [];// A variável A se refere a uma matriz vazia.Seja b = a;// Agora B refere -se à mesma matriz.b [0] = 1;// MATATE A matriz referida pela variável b.a [0] // => 1: A mudança também é visívelvariável a.a === b // => true: a e b referem -se ao mesmo objeto,Então eles são iguais.Como você pode ver neste código, atribuindo um objeto (ou matriz) a umA variável simplesmente atribui a referência: não cria uma nova cópia deo objeto.Se você quiser fazer uma nova cópia de um objeto ou matriz, vocêdeve copiar explicitamente as propriedades do objeto ou os elementos dovariedade.Este exemplo demonstra o uso de um loop for (§5.4.3):deixe A = ["A", "B", "C"];// Uma matriz que queremoscópiaSeja b = [];// Uma matriz distinta nós vamos

copiar empara (vamos i = 0; i < A.Length; i++) {// para cada índice de um []b [i] = a [i];// copie um elemento de umem b}Seja c = Array.From (B);// em ES6, copiar matrizescom Array.From ()Da mesma forma, se queremos comparar dois objetos ou matrizes distintos, nósdeve comparar suas propriedades ou elementos.Este código define uma funçãoPara comparar duas matrizes:função igualArrays (a, b) {if (a === b) retorna true;// idênticoMatrizes são iguaisif (a.Length! == B.Length) retornar FALSO;// Diferente-Matrizes de tamanho não são iguaispara (vamos i = 0; i < A.Length; i++) {// looptodos os elementosif (a [i]! == b [i]) retorna false;// se houverdiferem, matrizes não são iguais}retornar true;// De outra formaEles são iguais}3.9 Conversões de tipoO JavaScript é muito flexível sobre os tipos de valores necessários.Nós temosVi isso para booleanos: quando JavaScript espera um valor booleano, vocêpode fornecer um valor de qualquer tipo, e JavaScript o converterá comonecessário.Alguns valores (valores "verdadeiros") se convertem para verdadeiros e outros(Valores ?falsamente?) convertem para false.O mesmo vale para outros tipos: seJavaScript quer uma string, ele converterá qualquer valor que você der em umcorda.Se JavaScript quiser um número, tentará converter o valor que você

dê a um número (ou a `nan` se não puder realizar um significativo conversão). Alguns exemplos:
`10 + "Objetos" // => "10 Objetos"`: Número 10 convertido para uma string
`"7" * "4" // => 28`: Ambas as strings se convertem em números
Seja `n = 1 - "x"; // n == nan`; String "X" não pode ser convertida para um número
`n + "objetos" // => "NaN Objects"`: `nan` se converte para string "nan"
A Tabela 3-2 resume como os valores se convertem de um tipo para outro em JavaScript. Entradas em negrito na tabela destacam as conversões que você pode encontrar surpreendente. Células vazias indicam que nenhuma conversão é necessária e nenhum é realizado.
Tabela 3-2. Conversões do tipo JavaScript
Valor para string para número para booleano indefinido
"indefinido" "Nan" "false" "nulo" "0" "false" "verdadeiro" "true" "1" "falso" "0" "
(string vazia) "0" "false" "1.2" (não vazio, numérico) "1.2" "verdadeiro" "One" (não vazio, não numérico) "Nan" "verdadeiro" "0" "falso" "0" "

-0"0"falso1 (finito, diferente de zero)"1"verdadeiroInfinidade"Infinidade"verdadeiro-Infinidade"-Infinidade"verdadeiroNan"NaN"falso{} (qualquer objeto)Veja §3.9.3Veja §3.9.3verdadeiro[] (matriz vazia)""0verdadeiro[9] (um elemento numérico)"9"9verdadeiro['a'] (qualquer outra matriz)Use o método junção ()NaNverdadeirofunction () {} (qualquer função)Veja §3.9.3NanverdadeiroAs conversões primitivas para primitivas mostradas na tabela são relativamente direto.A conversão para booleana já foi discutida no §3.4.A conversão em strings é bem definida para todos os valores primitivos.A conversão em números é apenas um pouco mais complicada.Cordas que podem ser analisadasÀ medida que os números se convertem para esses números.Os espaços de liderança e trilha são permitidos, mas quaisquer caracteres de líder ou não -espaço que não seja parte de um literal numérico causa a conversão de corda em número produzir nan.Algumas conversões numéricas podem parecer surpreendentes: verdadeiro converte em 1, e false e a corda vazia converte para 0.A conversão de objeto para princípio é um pouco mais complicada, é o assunto do §3.9.3.3.9.1 Conversões e igualdade

O JavaScript possui dois operadores que testam se dois valores são iguais. O "operador estrito da igualdade", `==`, não considera seus operandos igual se eles não forem do mesmo tipo, e este é quase sempre o Operador direito para usar ao codificar. Mas porque JavaScript é tão flexível Com conversões de tipo, ele também define o operador `==` com um flexível Definição de igualdade. Todas as seguintes comparações são verdadeiras, para exemplo: `null == indefinido // => true`: esses dois valores são tratados como igual. `"0" == 0 // => true`: string se converte em um número antes de comparar. `0 == false // => true`: boolean converte em número antes de comparar. `"0" == false // => true`: ambos os operando se convertem para 0 Antes de comparar! §4.9.1 explica exatamente quais conversões são executadas pelo `==` operador para determinar se dois valores devem ser considerado igual. Lembre -se de que a conversibilidade de um valor para outro não implica igualdade desses dois valores. Se indefinido é usado onde um booleano O valor é esperado, por exemplo, será convertido para false. Mas isso faz não significa que indefinido `== false`. Operadores JavaScript e As declarações esperam valores de vários tipos e realizam conversões para Esse tipos. A declaração se converte indefinida a falsa, mas O operador `==` nunca tenta converter seus operandos em booleanos. 3.9.2 Conversões explícitas Embora o JavaScript execute muitas conversões de tipo automaticamente,

Às vezes, você pode precisar realizar uma conversão explícita, ou você pode preferir tornar as conversões explícitas para manter seu código mais claro. A maneira mais simples de realizar uma conversão de tipo explícita é usar o `toString()`, `Number()` e `String()`: `Número("3") // => 3` `String(false) // => "false"`; ou use `false.toString()`. Qualquer valor que não seja nulo ou indefinido tem um `toString()` método, e o resultado desse método geralmente é o mesmo retornado pela função `String()`. Como um aparte, observe que os `Boolean()`, `Number()` e `String()` As funções também podem ser invocadas - com novo - como construtor. Se você usa elas dessa maneira, você terá um objeto "wrapper" que se comporta exatamente como um valor primitivo de booleano, número ou string. Esses objetos de invólucro são umsobras históricas desde os primeiros dias de JavaScript, e há nunca realmente qualquer um bom motivo para usá-los. Certos operadores de JavaScript realizam conversões de tipo implícito e são às vezes usado explicitamente para fins de conversão de tipo. Se um operando do operador `+` é uma string, ele converte o outro em um corda. O operador `UNARY +` converte seu operando em um número. E o `UNARY!` operador converte seu operando em um booleano e nega. Esses fatos levam aos seguintes idiomas de conversão de tipo que você pode veja em algum código: `x + "" // => String(x)`

+x // => número (x)X-0 // => Número (x)!! x // => boolean (x): nota dupla!Formatação e análise de números são tarefas comuns no computadorprogramas e JavaScript possui funções e métodos especializados queForneça controle mais preciso sobre o número a cordas e a string-to-conversões numéricas.O método `toString ()` definido pela classe numérica aceita umArgumento opcional que especifica um radix ou base, para a conversão.SeVocê não especifica o argumento, a conversão é feita na base 10.No entanto, você também pode converter números em outras bases (entre 2 e36).Por exemplo:Seja n = 17;Seja binário = "0b" + n.toString (2);// binário == "0B10001"Seja octal = "0o" + n.toString (8);// octal == "0o21"Seja hex = "0x" + n.toString (16);// Hex == "0x11"Ao trabalhar com dados financeiros ou científicos, você pode quererconverter números em strings de maneiras que lhe dão controle sobre onúmero de locais decimais ou o número de dígitos significativos nosaída, ou você pode querer controlar se a notação exponencial éusado.A classe numérica define três métodos para esses tipos deconversões de número a cordas.`toFixed ()` converte um número em umstring com um número especificado de dígitos após o ponto decimal.Nuncausa notação exponencial.`toExponential ()` converte um número parauma string usando notação exponencial, com um dígito antes do decimalponto e um número especificado de dígitos após o ponto decimal (quesignifica que o número de dígitos significativos é maior que o valor

você especifica).toPrecision () converte um número em uma string com o número de dígitos significativos que você especificar. Usa notação exponencial se o número de dígitos significativos não é grande o suficiente para exibir o inteiro inteira do número. Observe que todos os três métodos ao redor dos dígitos ou almofada à direita com zeros, conforme apropriado. Considere o seguinte exemplo: Seja n = 123456.789; n.toFixed (0) // => "123457" n.toFixed (2) // => "123456.79" n.toFixed (5) // => "123456.78900" n.toExponential (1) // => "1.2e+5" n.toExponential (3) // => "1.235e+5" N.toPrecision (4) // => "1.235e+5" N.toPrecision (7) // => "123456.8" N.toPrecision (10) // => "123456.7890" Além dos métodos de formatação de números mostrados aqui, a classe Intl.NumberFormat define uma forma geral, internacionalizada de formatação de número. Veja §11.7.1 para obter detalhes. Se você passar uma string para a função de conversão número (), ela tenta analisar essa string como um número inteiro ou literal de ponto flutuante. Essa função funciona apenas para os números inteiros da base-10 e não permite caracteres à direita que não fazem parte do literal. O parseint () e as funções parsefloat () (essas são funções globais, não métodos de qualquer classe) são mais flexíveis. parseint () passa apenas números inteiros, enquanto Parsefloat () analisa números inteiros e pontos flutuantes. Se a string começa com "0x" ou "0X", parseint () o interpreta como um número hexadecimal. Parseint () e Parsefloat () ignoram espaço em branco, analisam o maior número possível de personagens numéricos, e

ignore qualquer coisa que se segue.Se o primeiro personagem não espacial não for parte de um literal numérico válido, eles retornam Nan:
parseint ("3 ratos cegos") // => 3
parseFloat ("3,14 metros") // => 3,14
parseint (" -12.34") // => -12
parseint ("0xff") // => 255
parseint ("0xff") // => 255
parseint (" -0xff") // => -255
parseFloat (. 1") // =>
0 parseint ("0,1") // => nan: os números inteiros não podem começar com ".parseFloat ("\$ 72,47") // => nan: os números não podem começar com "\$" parseint () aceita um segundo argumento opcional especificando o Radix (base) do número a ser analisado.Os valores legais estão entre 2 e 36. Por exemplo:
parseint ("11", 2) // => 3: (1*2 + 1)
parseint ("ff", 16) // => 255: (15*16 + 15)
parseint ("zz", 36) // => 1295: (35*36 + 35)
parseint ("077", 8) // => 63: (7*8 + 7)
parseint ("077", 10) // => 77: (7*10 + 7)3.9.3 Objeto de conversões primitivasAs seções anteriores explicaram como você pode converter explicitamente valores de um tipo para outro tipo e explicaram JavaScript's conversões implícitas de valores de um tipo primitivo para outro tipo primitivo.Esta seção abrange as regras complicadas que JavaScript usa para converter objetos em valores primitivos.É longo e obscuro, e se esta é a sua primeira leitura deste capítulo, você deve sentir

livre para pular a frente para o §3.10.Uma razão para a complexidade do objeto a princípio de JavaScriptconversões é que alguns tipos de objetos têm mais de um primitivorepresentação.Objetos de data, por exemplo, podem ser representados como stringsou como registro de data e hora numéricos.A especificação JavaScript define trêsAlgoritmos fundamentais para converter objetos em valores primitivos:preferência stringEste algoritmo retorna um valor primitivo, preferindo um valor de string,Se uma conversão para string for possível.número preferidoEste algoritmo retorna um valor primitivo, preferindo um número, seEssa conversão é possível.sem preferênciaEste algoritmo não expressa preferência sobre que tipo deo valor primitivo é desejado e as classes podem definir seus própriosconversões.Dos tipos de javascript embutidos, todos exceto a dataimplemente este algoritmo como número preferido.A classe de dataimplementa esse algoritmo como string de preferência.A implementação desses algoritmos de conversão de objeto para princípiosé explicado no final desta seção.Primeiro, no entanto, explicamos comoOs algoritmos são usados ??no JavaScript.Conversões objeto para booleanAs conversões de objeto para boolean são triviais: todos os objetos se convertem para true.Observe que esta conversão não requer o uso do objeto para-

algoritmos primitivos descritos e que literalmente se aplica a todos objetos, incluindo matrizes vazias e até o objeto Wrapper novoBooleano (falso). Conversões de objeto para corda Quando um objeto precisa ser convertido em uma string, JavaScript primeiro converte-o em um primitivo usando o algoritmo preferido de cordas, então converte o valor primitivo resultante em uma string, se necessário. Seguindo as regras na Tabela 3-2. Esse tipo de conversão acontece, por exemplo, se você passar um objeto para `Uma função interna que espera um argumento de string`, se você ligar `String ()` como uma função de conversão e quando você interpola objetos em literais de modelo (§3.3.4). Conversões de objeto para número Quando um objeto precisa ser convertido em um número, JavaScript primeiro converte-o em um valor primitivo usando o algoritmo de número preferido, então converte o valor primitivo resultante em um número, se necessário. Seguindo as regras na Tabela 3-2. Funções e métodos de javascript embutidos que esperam numéricos Argumentos convertem argumentos de objeto em números dessa maneira, e a maioria (veja as exceções a seguir) Operadores de JavaScript que esperam Operandos numéricos convertem objetos em números dessa maneira também. Conversões especiais de operadoras de casos Os operadores são abordados em detalhes no capítulo 4. Aqui, explicamos o

operadores de casos especiais que não usam o objeto básico para cordas eConversões de objeto para número descritas anteriormente.O operador + em JavaScript executa adição numérica e stringconcatenação.Se qualquer um de seus operandos for um objeto, JavaScript converteeles para valores primitivos usando o algoritmo de não preferência.Uma vez que temDois valores primitivos, ele verifica seus tipos.Se um argumento for umString, ele converte o outro em uma string e concatena as cordas.Caso contrário, ele converte os dois argumentos em números e os adiciona.Os == e! = Os operadores realizam testes de igualdade e desigualdades em ummaneira frouxa que permite conversões de tipo.Se um operando é um objeto eo outro é um valor primitivo, esses operadores convertem o objeto paraprinítivo usando o algoritmo sem preferência e compare os doisvalores primitivos.Finalmente, os operadores relacionais <, <=,> e> = compare a ordem deseus operandos e podem ser usados ??para comparar números e strings.SeQualquer um operando é um objeto, é convertido em um valor primitivo usando oAlgoritmo de número preferido.Observe, no entanto, que diferente do objeto para-Conversão de número, os valores primitivos retornados pelo número preferidoA conversão não é então convertida em números.Observe que a representação numérica dos objetos de data é significativamenteComparável com <e>, mas a representação da string não é.Para dataobjetos, o algoritmo de não preferência se converte em uma string, então o fatoEsse javascript usa o algoritmo de número preferido para esses operadoressignifica que podemos usá -los para comparar a ordem de dois objetos de data.

Os métodos `ToString()` e `ValueOf()` Todos os objetos herdam dois métodos de conversão que são usados ??por objeto a conversões primitivas e antes que possamos explicar a corda preferida, algoritmos de conversão de número preferido e de preferência, temos que Explique esses dois métodos. O primeiro método é `ToString()`, e seu trabalho é devolver uma string representação do objeto. O método padrão `toString()` faz não retornar um valor muito interessante (embora o achemos útil em §14.4.3): (`{x: 1, y: 2}`). `toString() // => "[objeto objeto]"` Muitas classes definem versões mais específicas do `ToString()` método. O método `toString()` da classe de matriz, por exemplo, converte cada elemento da matriz em uma string e se junta às strings resultantes juntamente com vírgulas no meio. O método `toString()` da classe de função converte funções definidas pelo usuário em strings de javascript código - fonte. A classe de data define um método `toString()` que retorna uma data e hora legíveis por humanos (e JavaScript) corda. A classe `regexp` define um método `toString()` que converte objetos `regexp` em uma string que se parece com um literal `regexp:[1,2,3]`. `ToString() // => "1,2,3"` (`function(x){f(x);}`). `toString() // => "function(x){f(x);} \d+/g.toString() // => "\d+/g"` Seja `d = nova data(2020,0,1); d.ToString() // => "Qua Jan 01 2020 00:00:00 GMT-0800 (Time padrão do Pacífico)"`

A outra função de conversão de objetos é chamada de `valueof()`. O trabalho de Este método é menos bem definido: deve converter um objeto para um valor primitivo que representa o objeto, se houver um valor primitivo existente. Objetos são valores compostos, e a maioria dos objetos não pode realmente ser representado por um único valor primitivo, portanto o valor padrão () o método simplesmente retorna o próprio objeto, em vez de retornar um primitivo. Classes de wrapper, como `string`, `número` e `booleano`, definem métodos `Valueof()` que simplesmente retornam o valor primitivo embrulhado. Matrizes, funções e expressões regulares simplesmente herdam o padrão método. Chamando `valueof()` para instâncias desses tipos simplesmente retorna o próprio objeto. A classe `data` define um método `ValueOf()` que retorna a data em sua representação interna: o número de milissegundos desde 1º de janeiro de 1970: Seja `D = New Date(2010, 0, 1); // 1 de janeiro de 2010, (Pacífico)` `D.Valueof() // => 1262332800000`

Algoritmos de conversão de objeto a princípio

Com os métodos `ToString()` e `ValueOf()` agora explique aproximadamente como os três objetos a princípio Os algoritmos funcionam (os detalhes completos são adiados até §14.4.7): O algoritmo preferido primeiro tenta o `toObject()` método. Se o método `for` definido e retornar um valor primitivo, então JavaScript usa esse valor primitivo (mesmo que não seja um corda!). Se `ToString()` não existir ou se retornar um objeto, então JavaScript tenta o método `ValueOf()`. Se isso o método existe e retorna um valor primitivo, depois JavaScript usa esse valor. Caso contrário, a conversão falha com um

TypeError.O algoritmo de número preferido funciona como a corda preferidaalgoritmo, exceto que tenta valueof () primeiro eToString () Segundo.O algoritmo sem preferência depende da classe doobjeto sendo convertido.Se o objeto for um objeto de data, entãoO JavaScript usa o algoritmo preferido.Para qualquer outroObjeto, JavaScript usa o algoritmo de número preferido.As regras descritas aqui são verdadeiras para todos os tipos de javascript embutidos esão as regras padrão para todas as classes que você se define.§14.4.7Explica como você pode definir sua própria conversão de objeto para primitivaAlgoritmos para as classes que você define.Antes de deixarmos este tópico, vale a pena notar que os detalhes doConversão de número preferido Explique por que as matrizes vazias se convertem para oNúmero 0 e matrizes de elementos únicos também podem se converter em números:Número ([] // => 0: Isso é inesperado!Número ([99]) // => 99: Sério?A conversão de objeto em número converte primeiro o objeto em um primitivo usando o algoritmo de número preferido e depois converte o resultantevalor primitivo para um número.O algoritmo do número preferido tentaValueof () primeiro e depois recorre ao ToString ().Mas a matrizClasse herda o método ValueOf () padrão, que não retorna umvalor primitivo.Então, quando tentamos converter uma matriz em um número, nósacaba invocando o método tostring () da matriz.Matrizes vaziasconverter para a string vazia.E a corda vazia se converte para onúmero 0. Uma matriz com um único elemento se converte para a mesma string

que esse elemento faz. Se uma matriz contiver um único número, que o número é convertido em uma string e depois volta a um número.

3.10 Declaração e atribuição variáveis

Uma das técnicas mais fundamentais da programação de computadores é o uso de nomes - ou identificadores - para representar valores. Vincular um nome a um valor nos dá uma maneira de se referir a esse valor e usá-lo nos programas que escrevemos. Quando fazemos isso, normalmente dizemos que somos atribuindo um valor a uma variável. O termo "variável" implica que novos valores podem ser atribuídos: que o valor associado à variável pode variar como nosso programa é executado. Se atribuirmos permanentemente um valor a um nome, então chamamos esse nome de constante em vez de uma variável. Antes de poder usar uma variável ou constante em um programa JavaScript, você deve declará-la. No ES6 e mais tarde, isso é feito com o Let and Const Palavras-chave, que explicaremos a seguir. Antes do ES6, variáveis eram declaradas com var, que é mais idiossincrático e é explicado mais tarde em esta seção.

3.10.1 declarações com Let and Const

No javascript moderno (ES6 e mais tarde), as variáveis são declaradas com o Deixe a palavra-chave, assim: deixe eu; deixe a soma; Você também pode declarar várias variáveis em uma única instrução Let:

Deixe eu, soma; É uma boa prática de programação atribuir um valor inicial ao seu variáveis ??quando você as declara, quando isso é possível: Deixe Message = "Hello"; Seja i = 0, j = 0, k = 0; Seja x = 2, y = x*x; // inicializadores podem usar anteriormente variáveis ??declaradas Se você não especificar um valor inicial para uma variável com o Letdeclaração, a variável é declarada, mas seu valor é indefinido atéSeu código atribui um valor a ele. Para declarar uma constante em vez de uma variável, use const em vez de let. Const funciona como Let, exceto que você deve inicializar a constante Quando você declara: const h0 = 74; // Hubble Constant (km/s/mpc) const C = 299792.458; // velocidade de luz no vácuo (km/s) const au = 1.496e8; // unidade astronômica: distância do Sol (km) Como o nome indica, as constantes não podem mudar seus valores e Qualquer tentativa de fazer isso faz com que um TypeError seja jogado. É uma convenção comum (mas não universal) declarar constantes Usando nomes com todas as letras maiúsculas, como H0 ou HTTP_NOT_FOUND como uma maneira de distingui -los das variáveis. Quando usar const

Existem duas escolas de pensamento sobre o uso da palavra -chave const.UmA abordagem é usar const apenas para valores que são fundamentalmente imutáveis, comoas constantes físicas mostradas, ou números de versão do programa, ou sequências de bytes usado para identificar tipos de arquivo, por exemplo.Outra abordagem reconhece que muitos deAs chamadas variáveis ??em nosso programa nunca mudam como nosso programacorre.Nesta abordagem, declaramos tudo com const, e então se encontrarmos issoNa verdade, queremos permitir que o valor varie, mudamos a declaração para deixar.Isso pode ajudar a prevenir bugs descartando mudanças accidentais para variáveis ??que nós não pretendia.Em uma abordagem, usamos const apenas para valores que não devem mudar.No outro,Usamos o const para qualquer valor que não mude.Eu prefiro o primeiroabordagem em meu próprio código.No capítulo 5, aprenderemos sobre o para, para/in e para/de loopDeclarações em JavaScript.Cada um desses loops inclui uma variável de loopIsso recebe um novo valor atribuído a ele em cada iteração do loop.O JavaScript nos permite declarar a variável de loop como parte do loopsintaxe em si, e essa é outra maneira comum de usar LET:para (vamos i = 0, len = data.length; i <len; i ++)console.log (dados [i]);para (Let Datum of Data) console.log (datum);para (deixe a propriedade no objeto) console.log (propriedade);Pode parecer surpreendente, mas você também pode usar o const para declarar o loop?Variáveis? para/in e para/de loops, desde que o corpo doO loop não reatribui um novo valor.Nesse caso, a declaração constestá apenas dizendo que o valor é constante durante a duração de um loopiteração:para (const Datum of Data) console.log (datum);

para (propriedade const em objeto) console.log (propriedade); Escopo variável e constante O escopo de uma variável é a região do seu código - fonte de programa em que é definido. Variáveis ?? e constantes declaradas com let e const são o bloqueio de blocos. Isso significa que elas são definidas apenas dentro do bloco de código no qual a instrução LET ou const aparece. A classe JavaScript e as definições de função são blocos, assim como os corpos de declarações IF/else, enquanto loops, loops e assim por diante. AGORAÇÃO falando, se uma variável ou constante for declarada dentro de um conjunto de aparelho encaracolado, então aqueles aparelhos encaracolados delimitam a região do código em que a variável ou constante é definida (embora é claro que não é legal para referenciar uma variável ou constante de linhas de código que executam antes da declaração LET ou const que declara a variável). Variáveis ?? e constantes declaradas como parte de um para, para/in ou para/de loop tem o corpo do loop como seu escopo, mesmo que ele tecnicamente aparecem fora dos aparelhos encaracolados. Quando uma declaração aparece no nível superior, fora de qualquer bloco de código, dizemos que é uma variável global ou constante e tem escopo global. No nó nos módulos JavaScript do lado do cliente (consulte o capítulo 10), o escopo de uma variável global é o arquivo em que é definido. No lado tradicional do cliente JavaScript, no entanto, o escopo de uma variável global é o documento HTML no qual é definido. Isto é: se um <script> declara uma variável global ou constante, essa variável ou constante é definida em todos os elementos <Script> nesse documento (ou pelo menos todos os scripts que executar após a instrução LET ou const executar).

Declaracões repetidasÉ um erro de sintaxe usar o mesmo nome com mais de um let ouDeclaracão const no mesmo escopo.É legal (embora seja melhor uma prticaevitado) para declarar uma nova variável com o mesmo nome em um aninhadoescopo:const x = 1;// declarar x como uma constante globalif (x === 1) {Seja x = 2;// dentro de um bloco x pode se referir a umvalor diferenteconsole.log (x);// imprime 2}console.log (x);// Imprima 1: estamos de volta ao globalescopo agoraSeja x = 3;// Erro!Erro de sintaxe tentando voltardeclarar xDeclaracões e tiposSe você está acostumado a idiomas estaticamente digitados como C ou Java, você podeachar que o objetivo principal das declaracões variáveis ??é especificar o tipo de valores que podem ser atribuídos a uma variável.Mas, como você temvisto, não há tipo associado à variável de JavaScriptdeclaracões.Uma variável JavaScript pode conter um valor de qualquer tipo.Parapor exemplo, é perfeitamente legal (mas geralmente mau estilo de programação) emJavascript para atribuir um número a uma variável e depois atribuir umstring para essa variável:deixe i = 10;i = "ten";3.10.2 declaracões variáveis ??com VAR2

Nas versões do JavaScript antes do ES6, a única maneira de declarar uma variável está com a palavra -chave VAR e não há como declarar constantes. A sintaxe do VAR é como a sintaxe de Let: var x; var dados = [], count = data.length; for (var i = 0; i < contagem; i++) console.log (dados [i]); Embora o VAR tenha e tenha a mesma sintaxe, existem importantes diferenças na maneira como eles funcionam: As variáveis ??declaradas com VAR não têm escopo de bloco. Em vez de, Eles são escopos no corpo da função que contém não importa o quanto profundamente aninhados eles estão dentro dessa função. Se você usar o VAR fora de um corpo de função, ele declara um global variável. Mas as variáveis ??globais declaradas com var diferem de Globals declarados com Let In de uma maneira importante. Globals declarados com VAR são implementados como propriedades do global objeto (§3.7). O objeto global pode ser referenciado como global. Então, se você escrever var x = 2; fora de uma função, é como se você escrevesse globalThis.x = 2;. Observação No entanto, que a analogia não é perfeita: as propriedades criadas com declarações globais de var Excluir operador (§4.13.4). Variáveis ??globais e constantes declarados com let e const não são propriedades do global objeto. Ao contrário das variáveis ??declaradas com LET, é legal declarar a mesma variável várias vezes com var. E porque var variáveis ??têm escopo de função em vez de escopo de bloco, é na verdade, comum para fazer esse tipo de redeclaração. A variável Eu sou frequentemente usado para valores inteiros, e especialmente como o Variável de índice de loops. Em uma função com múltiplos

Loops, é típico para cada um começar (var i = 0;.... porque o VAR não esconde essas variáveis ??para o loopcorpo, cada um desses loops é (inofensivamente) re-declaração e re-inicializando a mesma variável.Uma das características mais incomuns das declarações VAR éconhecido como iça.Quando uma variável é declarada com var, oa declaração é levantada (ou "içada") para o topo doFunção de anexo.A inicialização da variável permaneceonde você escreveu, mas a definição da variável se move parao topo da função.Portanto, as variáveis ??declaradas com var podem serUsado, sem erro, em qualquer lugar da função de anexo.Se oO código de inicialização ainda não foi executado, o valor doA variável pode ser indefinida, mas você não receberá um erro seVocê usa a variável antes de ser inicializada.(Este pode ser umfonte de bugs e é um dos malfeilhos importantes queDeixe corrigir: se você declarar uma variável com LET HABLE TENTEPara usá -lo antes da execução da declaração let, você receberá um verdadeiroerro em vez de apenas ver um valor indefinido.)Usando variáveis ??não declaradasNo modo rigoroso (§5.6.3), se você tentar usar uma variável não declarada, você receberá umErro de referência ao executar seu código.Fora do modo rigoroso, no entanto, se vocêatribua um valor a um nome que não foi declarado com let, const ou var,Você acabará criando uma nova variável global.Será um global, não importa agoraprofundamente aninhado nas funções e bloqueia seu código, o que é quase certamenteNão é o que você quer, é propenso a insetos e é uma das melhores razões para usar rigorosamentemodo!As variáveis ??globais criadas dessa maneira accidental são como variáveis ??globais declaradascom VAR: eles definem propriedades do objeto global.Mas, diferente das propriedadesDefinidos por declarações adequadas do VAR, essas propriedades podem ser excluídas com oExcluir operador (§4.13.4).

3.10.3 Atribuição de destruiçãoES6 implementa um tipo de declaração e atribuição compostasSintaxe conhecida como atribuição de destruição. Em uma destruiçãoatribuição, o valor no lado direito do sinal igual é um matriz ou objeto (um valor "estruturado"), e o lado esquerdo especifica um ou mais nomes variáveis ??usando uma sintaxe que imita a matriz eSintaxe literal do objeto. Quando ocorre uma tarefa de destruição, um ouMais valores são extraídos ("destruturados") do valor à direita e armazenado nas variáveis ??nomeadas à esquerda. DestruiçãoA atribuição talvez seja mais comumente usada para inicializar variáveis ??comoparte de uma declaração de declaração const, let ou var, mas também pode ser feito em expressões regulares de tarefas (com variáveis ??que têm já foi declarado). E, como veremos no §8.3.5, a destruição podeTambém seja usado ao definir os parâmetros para uma função. Aqui estão tarefas simples de destruição usando matrizes de valores: Seja $[x, y] = [1, 2]$; // o mesmo que let $x = 1$, $y = 2$; $[x, y] = [x+1, y+1]$; // o mesmo que $x = x + 1$, $y = y + 1$; $[x, y] = [y, x]$; // Troque o valor das duas variáveis $[x, y] // => [3, 2]$: o incrementado e trocado valores Observe como a atribuição de destruturação facilita o trabalho funções que retornam matrizes de valores: // converte $[x, y]$ coordenadas para $[r, \theta]$ coordenadas polares função topolar (x, y) {return [math.sqrt (x*x+y*y), math.atan2 (y, x)];} // Converter coordenadas polares para cartesianas

função tocartesian (r, teta) {retornar [r*math.cos (teta), r*math.sin (teta)];}Seja [r, teta] = topolar (1,0, 1,0);// r == Math.sqrt (2);Theta == Math.pi/4Seja [x, y] = tocartesiano (r, teta);// [x, y] == [1,0, 1,0]Vimos que variáveis ??e constantes podem ser declaradas como parte deVários de Javascript para loops.É possível usar variávelDestruir nesse contexto também.Aqui está um código que atravessa oNome/valor pares de todas as propriedades de um objeto e usa a destruiçãoatribuição para converter esses pares de matrizes de dois elementos emvariáveis ??individuais:Seja o = {x: 1, y: 2};// o objeto que vamos fazerfor (const [nome, valor] de object.entries (o)) {console.log (nome, valor);}// imprime "x 1" e "y 2"}O número de variáveis ??à esquerda de uma tarefa de destruição faznão precisa corresponder ao número de elementos de matriz à direita.ExtraVariáveis ??à esquerda são definidas como indefinidas e valores extras noà direita são ignorados.A lista de variáveis ??à esquerda pode incluir extravírgulas para pular certos valores à direita:vamos [x, y] = [1];// x == 1;y == indefinido[x, y] = [1,2,3];// x == 1;y == 2[, x , y] = [1,2,3,4];// x == 2;y == 4Se você deseja coletar todos os valores não utilizados ou restantes em um únicovariável ao destruir uma matriz, use três pontos (...) antes doÚltimo nome de variável no lado esquerdo:

vamos [x, ... y] = [1,2,3,4];// y == [2,3,4]Veremos três pontos usados ??dessa maneira novamente no §8.3.2, onde eles são usados para indicar que todos os argumentos de função restantes devem ser coletados em uma única matriz.A tarefa de destruição pode ser usada com matrizes aninhadas.Nesse caso,O lado esquerdo da tarefa deve parecer uma matriz aninhada literal:vamos [a, [b, c]] = [1, [2,2.5], 3];// a == 1;b == 2;c ==2.5Uma característica poderosa da destruição de matrizes é que ela realmente não requer uma matriz!Você pode usar qualquer objeto iterável (capítulo 12) no Lado Right Hands da tarefa;qualquer objeto que possa ser usado com um para/de loop (§5.4.4) também pode ser destruturado:vamos [primeiro, ... descansar] = "olá";// primeiro == "h";descanso ==["E", "L", "L", "O"]A tarefa de destruição também pode ser realizada quando a direita lado é um valor de objeto.Nesse caso, o lado esquerdo da tarefa Parece algo como um objeto literal: uma lista separada por vírgulas nomes variáveis ??dentro de aparelhos encaracolados:Seja transparente = {r: 0.0, g: 0,0, b: 0,0, a: 1,0}// A RGBACorSeja {r, g, b} = transparente;// r == 0,0;g == 0,0;b == 0.0O próximo exemplo copia as funções globais do objeto de matemática em Variáveis, que podem simplificar o código que faz muita trigonometria:

// O mesmo que const sin = math.sin, cos = math.cos, tan = math.tanconst {sin, cos, tan} = matemática;Observe no código aqui que o objeto de matemática tem muitas propriedades outrasdo que os três que são destruídos em variáveis ??individuais.Aqueles quenão são nomeados são simplesmente ignorados.Se o lado esquerdo desteA tarefa incluiu uma variável cujo nome não era uma propriedade deMatemática, essa variável seria simplesmente designada indefinida.Em cada um desses exemplos de destruição de objetos, escolhemosnomes variáveis ??que correspondem aos nomes de propriedades do objeto que somosdestruição.Isso mantém a sintaxe simples e fácil de entender, masnão é necessário.Cada um dos identificadores no lado esquerdo de umA atribuição de destruição de objetos também pode ser um par de separação de cólon deidentificadores, onde o primeiro é o nome da propriedade cujo valor éser atribuído e o segundo é o nome da variável para atribuí -la a:// O mesmo que const cossene = math.cos, tangente = math.tan;const {cos: cosseno, tan: tangente} = matemática;Acho que a sintaxe de destruição de objetos se torna muito complicada para serÚtil quando os nomes de variáveis ??e nomes de propriedades não são os mesmos,E eu tendem a evitar a abreviação neste caso.Se você optar por usá -lo,Lembre -se de que os nomes de propriedades estão sempre à esquerda do cólon, emambos os literais de objeto e à esquerda de um objeto destrutivoatribuição.A tarefa de destruição se torna ainda mais complicada quando éusado com objetos aninhados, ou matrizes de objetos, ou objetos de matrizes, masé legal:

deixe pontos = [{x: 1, y: 2}, {x: 3, y: 4}];// uma variedade de dois objetos de pontosSeja [{x: x1, y: y1}, {x: x2, y: y2}] = pontos;//Destruir em 4 variáveis.(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => trueOu, em vez de destruir uma variedade de objetos, poderíamos destruir um objeto de matrizes:Let Points = {P1: [1,2], P2: [3,4]};// um objeto com 2 adereços de matrizSeja {p1: [x1, y1], p2: [x2, y2]} = pontos;//Destruir em 4 VARs(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => trueSintaxe de destruição complexa como essa pode ser difícil de escrever e difícil paraLeia, e você pode estar melhor apenas escrevendo suas tarefas explicitamente com código tradicional como let x1 = pontos.p1 [0];.Compreendendo a destruição complexaSe você se encontrar trabalhando com código que usa atribuições complexas de destruição, há um útil regularidade que pode ajudá-lo a entender os casos complexos.Pense primeiro em um regular (único-valor) atribuição.Depois que a tarefa for concluída, você pode pegar o nome da variável da esquerda lado da tarefa e use -a como uma expressão em seu código, onde ele avaliará para qualquer valor que você atribuiu.O mesmo se aplica à atribuição de destruição.O lado esquerdo do lado de uma atribuição de destruição parece uma matriz literal ou um objeto literal (§6.2.1 e §6.10).Depois de A tarefa foi realizada, o lado esquerdo da mão funcionará como uma matriz válida literal ou objeto literal em outros lugares do seu código.Você pode verificar se escreveu uma tarefa de destruição corretamente porTentando usar o lado esquerdo do lado do campo de outra expressão de atribuição:// Comece com uma estrutura de dados e uma destruição complexadeixe pontos = [{x: 1, y: 2}, {x: 3, y: 4}];Seja [{x: x1, y: y1}, {x: x2, y: y2}] = pontos;// Verifique sua sintaxe de destruição de destruiçãoSeja pontos2 = [{x: x1, y: y1}, {x: x2, y: y2}];// pontos2 == pontos

3.11 ResumoAlguns pontos -chave a serem lembrados sobre este capítulo:Como escrever e manipular números e seqüências de texto emJavaScript.Como trabalhar com outros tipos primitivos de JavaScript:Booleanos, símbolos, nulos e indefinidos.As diferenças entre tipos primitivos imutáveis ??eTipos de referência mutáveis.Como o JavaScript converte valores implicitamente de um tipo paraOutro e como você pode fazê -lo explicitamente em seus programas.Como declarar e inicializar constantes e variáveis(inclusive com atribuição de destruição) e o lexicalEscopo das variáveis ??e constantes que você declara.1Este é o formato para números de tipo duplo em java, c ++ e mais modernolinguagens de programação.2Existem extensões de JavaScript, como TypeScript e Flow (§17.8), que permitem que tiposser especificado como parte de declarações variáveis ??com sintaxe como let x: número = 0;.

Capítulo 4. Expressões e Operadores

Este capítulo documenta expressões de javascript e os operadores com que muitas dessas expressões são construídas. Uma expressão é uma frase de JavaScript que pode ser avaliada para produzir um valor. Uma constante incorporada literalmente em seu programa é um tipo muito simples de expressão. Um nome de variável também é uma expressão simples que avalia para qualquer coisa. O valor foi atribuído a essa variável. Expressões complexas são construídas de expressões mais simples. Uma expressão de acesso à matriz, por exemplo, consiste em uma expressão que avalia a uma matriz seguida por um suporte quadrado aberto, uma expressão que avalia para um número inteiro e um feche o suporte quadrado. Esta nova expressão mais complexa avalia para o valor armazenado no índice especificado da matriz especificada. De forma similar, uma expressão de invocação de função consiste em uma expressão que avalia para um objeto de função e zero ou mais expressões adicionais que são usados como argumentos para a função. A maneira mais comum de construir uma expressão complexa a partir de mais simples expressões é com um operador. Um operador combina os valores de seu operando (geralmente dois deles) de alguma forma e avalia para um novo valor. O operador de multiplicação * é um exemplo simples. O expressão $x * y$ avalia o produto dos valores das expressões X e Y. Por simplicidade, às vezes dizemos que um operador retorna um valor em vez de "avaliar" um valor.

Este capítulo documenta todos os operadores de JavaScript, e também explica expressões (como indexação de matrizes e invocação de funções) que não usam operadores. Se você já conhece outra programação em linguagem que usa sintaxe no estilo C, você descobrirá que a sintaxe da maioria das expressões e operadores da JavaScript já estão familiarizados para você.

4.1 Expressões primárias

As expressões mais simples, conhecidas como expressões primárias, são aquelas que estão sozinhas - elas não incluem expressões mais simples. Primário Expressões em JavaScript são valores constantes ou literais, certos Palavras-chave do idioma e referências variáveis. Literais são valores constantes que são incorporados diretamente em seu programa. Eles se parecem com estes:

- 1.23 // um número literal
- "Olá" // uma string literal
- / padrão // uma expressão regular
- literal A sintaxe JavaScript para literais numéricos foi coberta no §3.2. Corda Os literais foram documentados no §3.3. A expressão regular da sintaxe literal foi introduzido no §3.3.5 e será documentado em detalhes no §11.3. Algumas das palavras reservadas de JavaScript são expressões primárias: verdadeiro // avalia para o valor verdadeiro
- booleano false // avalia o valor falso
- booleano null // avalia o valor nulo
- Isso // avalia o objeto "atual"

Aprendemos sobre verdadeiro, falso e nulo em §3.4 e §3.5. Diferente

as outras palavras -chave, isso não é uma constante - avalia para diferentes valores em diferentes locais do programa. A palavra -chave esta é usada em Programação orientada a objetos. Dentro do corpo de um método, este Avalia o objeto no qual o método foi invocado. Veja §4.5, Capítulo 8 (especialmente §8.2.2) e capítulo 9 para mais informações sobre isso. Finalmente, o terceiro tipo de expressão primária é uma referência a um variável, constante ou propriedade do objeto global: i // Avalia o valor da variável i. sum // avalia o valor da soma variável. indefinido // o valor da propriedade "indefinida" do objeto global Quando qualquer identificador aparece por si só em um programa, JavaScript assume É uma variável ou constante ou propriedade do objeto global e procure para cima seu valor. Se não houver variável com esse nome, uma tentativa de avaliar uma variável inexistente lança um referenceError.

4.2 Inicializadores de objeto e matriz

Inicializadores de objeto e matriz são expressões cujo valor é um recém-objeto criado ou matriz. Essas expressões de inicializador às vezes são chamados literais de objeto e literais de matriz. Ao contrário dos verdadeiros literais, no entanto, eles não são expressões primárias, porque incluem uma série de subexpressões que especificam valores de propriedade e elemento. Variedade Os inicializadores têm uma sintaxe um pouco mais simples, e começaremos com eles. Um inicializador de matriz é uma lista de expressões separada por vírgula contida dentro de colchetes. O valor de um inicializador de matriz é um recém-

matriz criada. Os elementos desta nova matriz são inicializados para oValores das expressões separadas por vírgula:[] // Uma matriz vazia: sem expressões dentro de colchetessignifica não elementos[1+2,3+4] // Uma matriz de 2 elementos. O primeiro elemento é 3, segundoé 7As expressões de elementos em um inicializador de matriz podem ser mesmasInicializadores, o que significa que essas expressões podem criar aninhadasMatrizes:Let Matrix = [[1,2,3], [4,5,6], [7,8,9]];As expressões de elemento em um inicializador de matriz são avaliadas a cada vezO inicializador da matriz é avaliado. Isso significa que o valor de uma matrizA expressão inicializadora pode ser diferente cada vez que é avaliada.Elementos indefinidos podem ser incluídos em uma matriz literal por simplesmenteomitindo um valor entre vírgulas.Por exemplo, a seguinte matrizContém cinco elementos, incluindo três elementos indefinidos:Seja SparsarArray = [1 ,,, , 5];Uma única vírgula à direita é permitida após a última expressão em uma matrizInicializador e não cria um elemento indefinido.No entanto, qualquer umExpressão de acesso à matriz para um índice após o da última expressãoserá necessariamente avaliado como indefinido.Expressões de inicializador de objetos são como expressões de inicializador de matriz, masOs suportes quadrados são substituídos por colchetes encaracolados, e cadaA subexpressão é prefixada com um nome de propriedade e um colón:

Seja p = {x: 2.3, y: -1.2}; // um objeto com 2 propriedades
Seja q = {} // um objeto vazio sem propriedades
q.x = 2.3; q.y = -1.2; // agora q tem as mesmas propriedades como p
No ES6, os literais de objetos têm uma sintaxe muito mais rica em recursos (você pode encontrar detalhes em §6.10). Os literais de objeto podem ser aninhados. Por exemplo:
Deixe Rectangle = {UpperLeft: {x: 2, y: 2}, LowerRight: {x: 4, y: 5}};
Veremos os inicializadores de objetos e matrizes nos capítulos 6 e 7.
4.3 Expressões de definição de função
Uma expressão de definição de função define uma função JavaScript, e o valor dessa expressão é a função recém-definida. Em certo sentido, uma expressão de definição da função é uma "função literal" da mesma maneira que um inicializador de objeto é um "objeto literal".
Uma definição de função A expressão normalmente consiste na função de palavra-chave seguida por uma lista separada por vírgula de zero ou mais identificadores (os nomes dos parâmetros) entre parênteses e um bloco de código JavaScript (o corpo da função) emparelhado encaracolado. Por exemplo:
// Esta função retorna o quadrado do valor passado para isto.
Deixe Square = function (x) {return x * x};
Uma expressão de definição de função também pode incluir um nome para o

função. As funções também podem ser definidas usando uma declaração de função em vez de uma expressão de função. E no ES6 e mais tarde, funçãoAs expressões podem usar uma nova sintaxe compacta "Função de seta". Detalhes sobre a definição da função estão no capítulo 8.4.4 Expressões de acesso à propriedade. Uma expressão de acesso à propriedade avalia o valor de um objeto propriedade ou um elemento de matriz. JavaScript define duas sintaxes para Acesso à propriedade: expressão.identificador [expressão] O primeiro estilo de acesso à propriedade é uma expressão seguida por um período e um identificador. A expressão especifica o objeto e o identificador especifica o nome da propriedade desejada. O segundo estilo de acesso à propriedade segue a primeira expressão (o objeto ou matriz) com outro expressão entre parênteses. Esta segunda expressão especifica o nome da propriedade desejada ou o índice do elemento de matriz desejado. Aqui estão alguns exemplos concretos: Seja `o = {x: 1, y: {z: 3}};`// um exemplo de objeto Seja `a = [0, 4, [5, 6]];`// um exemplo de matriz que contém um objeto `o.x` // => 1: Propriedade X de expressão `o.Y.Z` // => 3: Propriedade z da expressão `o.Y["x"]` // => 1: propriedade x do objeto `o[1]` // => 4: elemento no índice 1 da expressão `a[2][1]` // => 6: elemento no índice 1 da expressão `a[2]`

a [0] .x // => 1: Propriedade x de expressãoA [0]Com qualquer tipo de expressão de acesso à propriedade, a expressão antese .ou [é primeiro avaliado.Se o valor for nulo ou indefinido, oA expressão lança um TypeError, já que esses são os dois JavaScriptvalores que não podem ter propriedades.Se a expressão do objeto for seguidapor um ponto e um identificador, o valor da propriedade nomeada por issoO identificador é procurado e se torna o valor geral da expressão.Se a expressão do objeto for seguida por outra expressão no quadradoSuportes, essa segunda expressão é avaliada e convertida em uma corda.O valor geral da expressão é então o valor da propriedadenomeado por essa string.Em ambos os casos, se a propriedade nomeada nãoExistir, então o valor da expressão de acesso à propriedade é indefinido.A sintaxe .Identifier é a mais simples das duas opções de acesso à propriedade,mas observe que só pode ser usado quando a propriedade que você deseja acesso tem um nome que é um identificador legal e quando você sabe oNome quando você escreve o programa.Se o nome da propriedade incluirespaços ou caracteres de pontuação, ou quando é um número (para matrizes),Você deve usar a notação de suporte quadrado.Suportes quadrados também são usadosQuando o nome da propriedade não é estático, mas é o resultado de umComputação (consulte §6.3.1 para um exemplo).Objetos e suas propriedades são abordados em detalhes no capítulo 6 eMatrizes e seus elementos são abordados no capítulo 7.4.4.1 Acesso à propriedade condicionalO ES2020 adiciona dois novos tipos de expressões de acesso à propriedade:

expressão?.identificadorepressão?. [Expressão]Em JavaScript, os valores nulos e indefinidos são os únicos doisvalores que não têm propriedades.Em um acesso regular à propriedadeexpressão usando.ou [], você obtém um TypeError se a expressão noA esquerda avalia como nula ou indefinida.Você pode usar?.e ?.[]Sintaxe para proteger contra erros desse tipo.Considere a expressão A? .B.Se A é nulo ou indefinido, então oA expressão avalia para indefinida sem qualquer tentativa de acessar oPropriedade b.Se A é algum outro valor, então a? .B avalia para qualquer coisaA.B avaliaria (e se A não tivesse uma propriedade chamada b,então o valor será novamente indefinido).Esta forma de expressão de acesso à propriedade às vezes é chamada de ?opcionalencadeamento ?porque também funciona para acesso de propriedade mais longo? encadeado ?expressões como esta:Seja a = {b: null};a.b? .c.d // => indefinidoA é um objeto, então A.B é uma expressão de acesso à propriedade válida.Mas oO valor de A.B é nulo, então A.B.C lançaria um TypeError.Usando?.em vez de .Evitamos o TypeError, e A.B? .C avalia para indefinido.Isso significa que (A.B? .C) .D jogará um TypeError,Porque essa expressão tenta acessar uma propriedade do valorindefinido.Mas - e esta é uma parte muito importante de ?opcionalencadeamento ? - a.b? .c.d (sem parênteses) simplesmente avalia

indefinido e não apresenta um erro. Isso é porque propriedade de acesso com?.é "curto-circuito": se a subexpressão à esquerda de?. avalia como nulo ou indefinido, depois toda a expressão avalia imediatamente para indefinido sem mais propriedade tentativas de acesso. Obviamente, se A.B é um objeto, e se esse objeto não tiver propriedade denominada C, então A.B?.C.D lançará novamente um TypeError, e vamos que use outro acesso à propriedade condicional: Seja a = {b: {}}; a.b?.c?.d // => indefinido O acesso à propriedade condicional também é possível usando?.[] Em vez de ?. Na expressão a?.[B] [c], se o valor de a é nulo ou indefinido, então toda a expressão avalia imediatamente para indefinido, e as subexpressões B e C nunca são avaliadas. Se qualquer uma dessas expressões tem efeitos colaterais, o efeito colateral não terá ocorrido se A não estiver definido: deixe um:// opa, esquecemos de inicializar isso variável! deixe index = 0; tentar {a [index ++]; // lança TypeError} catch (e) {índice // => 1: O incremento ocorre antes que o TypeError seja jogado} a?.[index ++] // => indefinido: porque a é indefinido índice // => 1: não incrementado porque?.[] curtos circuitos A [index ++] //! TypeError: Não é possível indexar indefinidos.

Acesso à propriedade condicional com `com?.e?.` [] é um dos mais recentes recursos de JavaScript. No início de 2020, esta nova sintaxe é suportada nas versões atuais ou beta da maioria dos principais navegadores.

4.5 Expressões de invocação

Uma expressão de invocação é a sintaxe de JavaScript para chamadas (ou executando) uma função ou método. Começa com uma expressão de função que identifica a função a ser chamada. A expressão da função é seguida de um parêntese aberto, uma lista separada por vírgula de zero ou mais expressões de argumento e um parê de parênteses estreitos. Alguns exemplos:

- `f(0)` // f é a expressão da função;
- `0` é a expressão de argumento.
- `Math.max(x, y, z)` // `math.max` é a função; `x, y e z` são os argumentos.
- `a.sort()` // `a.sort` é a função; Não há argumentos.

Quando uma expressão de invocação é avaliada, a expressão da função é avaliada primeiro, e depois as expressões de argumento são avaliadas para produzir uma lista de valores de argumento. Se o valor da função não é uma função, um `TypeError` é jogado. Em seguida, o argumento Os valores são atribuídos, para os nomes de parâmetros especificados quando a função foi definida e, em seguida, o corpo da função é executado. Se a função usar uma declaração de retorno para retornar um valor, então isso O valor se torna o valor da expressão de invocação. Caso contrário, o O valor da expressão de invocação é indefinido. Detalhes completos na invocação de funções, incluindo uma explicação do que acontece quando O número de expressões de argumento não corresponde ao número de Os parâmetros na definição da função estão no capítulo 8.

Toda expressão de invocação inclui um par de parênteses e uma expressão antes dos parênteses abertos. Se essa expressão for uma propriedade de acesso, então a invocação é conhecida como um método de invocação. Nas invocações de método, o objeto ou a matriz que é o sujeito do acesso à propriedade se torna o valor dessa palavra-chave enquanto o corpo da função está sendo executado. Isso permite um objeto-paradigma de programação orientado no qual as funções (que chamamos "Métodos" quando usado dessa maneira) operam no objeto de que são papel. Veja o capítulo 9 para obter detalhes.

4.5.1 Invocação condicional

No ES2020, você também pode invocar uma função usando `?()`. Normalmente quando você invoca uma função, se a expressão à esquerda de os parênteses são nulos ou indefinidos ou qualquer outra não função, um `TypeError` é jogado. Com o novo `?()` Sintaxe de invocação, se a expressão à esquerda do `?()` avalia para nulo ou indefinido, então toda a expressão de invocação avalia para indefinido e não uma exceção é lançada. Objetos de matriz têm um método de classificação `()` que pode opcionalmente ser passado um argumento da função que define a ordem de classificação desejada para a matriz de elementos. Antes do ES2020, se você quisesse escrever um método como `classin()` que leva um argumento de função opcional, você normalmente usaria uma instrução `IF` para verificar se o argumento da função foi definido antes de invocá-lo no corpo do `IF`:
`Função quadrada (x, log) { // O segundo argumento é um`

função opcionalif (log) { // se a função opcional for passou log (x); // Invocar} retornar x * x; // retorna o quadrado do argumento} Com esta sintaxe de invocação condicional do ES2020, no entanto, você pode Basta escrever a invocação de funções usando?. () , Sabendo que A invocação só acontecerá se houver um valor a ser chamado: Função quadrada (x, log) { // O segundo argumento é um função opcional log?. (x); // Chame a função se houver um retornar x * x; // retorna o quadrado do argumento} Observe, no entanto, isso?. () Apenas verifica se o lado da esquerda é nulo ou indefinido. Não verifica que o valor é realmente um função. Então a função square () neste exemplo ainda jogaria Uma exceção se você passou dois números, por exemplo. Como expressões de acesso à propriedade condicional (§4.4.1), função invocação com?. () é curto-circuito: se o valor à esquerda de?. é nulo ou indefinido, então nenhuma das expressões de argumento Dentro dos parênteses são avaliados: Seja f = nulo, x = 0; tentar {f (x ++); // joga TypeError porque f é nulo} catch (e) {

x // => 1: x é incrementado antes da exceçãoé jogado}f?. (x++) // => indefinido: f é nulo, mas sem exceçãojogadox
// => 1: o incremento é ignorado por causa de curta circuíto Expressões de invocação condicional com?. ()
Funcionam tão bem paraMétodos como fazem para funções.Mas porque a invocação de método tambémenvolve
acesso à propriedade, vale a pena levar um momento para ter certeza deEntenda as diferenças entre as seguintes
expressões:o.m () // acesso regular à propriedade, invocação regularo?. .m () // Acesso à propriedade condicional,
invocação regularO.M?. () // Acesso regular à propriedade, Invocação condicionalNa primeira expressão, O deve
ser um objeto com uma propriedade M e oO valor dessa propriedade deve ser uma função.Na segunda expressão,
se oé nulo ou indefinido, então a expressão avalia para indefinido.Mas se O tiver algum outro valor, deve ter
umPropriedade m cujo valor é uma função.E na terceira expressão, Onão deve ser nulo ou indefinido.Se não tiver
uma propriedade m, ouSe o valor dessa propriedade for nulo, toda a expressãoAvalia como indefinido.Invocação
condicional com?. () É uma das características mais recentes deJavaScript.A partir dos primeiros meses de 2020,
esta nova sintaxe é suportadanas versões atuais ou beta da maioria dos principais navegadores.4.6 Expressões
de criação de objetos

Uma expressão de criação de objetos cria um novo objeto e invoca uma função (chamada de construtor) para inicializar as propriedades desse objeto. Expressões de criação de objetos são como expressões de invocação, exceto que elas são prefixadas com a palavra-chave `nova`: novo objeto () Novo ponto (2,3) Se nenhum argumento for passado para a função do construtor em um objeto Expressão da criação, o par vazio de parênteses pode ser omitido: novo objeto nova data O valor de uma expressão de criação de objetos é o objeto recém-criado. Os construtores são explicados em mais detalhes no capítulo 9.4.7 Visão geral do operador Os operadores são usados ?? para expressões aritméticas de JavaScript, comparação Expressões, expressões lógicas, expressões de atribuição e muito mais. A Tabela 4-1 resume os operadores e serve como um conveniente referência. Observe que a maioria dos operadores é representada por caracteres de pontuação como + e =. Alguns, no entanto, são representados por palavras-chave como `exclui` e `instanceof`. Os operadores de palavras-chave são operadores regulares, assim como os expressos com pontuação; eles simplesmente têm menos Sintaxe suíça. A Tabela 4-1 é organizada pela precedência do operador. Os operadores listados

Erro ao traduzir esta página.

Erro ao traduzir esta página.

`+=`, `-=`, `&=`, `^=`, `| =`, `<<=`, `>>=`, `>>>=`, Assim, Descarte o 1º operando, retornar 2ºL2 qualquer, qualquer ?
qualquer 4.7.1 Número de operandos Os operadores podem ser categorizados com base no número de operandos que eles Espera (sua arity). A maioria dos operadores de javascript, como a `*` multiplicação operador, são operadores binários que combinam duas expressões em uma expressão única e mais complexa. Ou seja, eles esperam dois operando. JavaScript também suporta vários operadores unários, que convertem uma expressão única em uma expressão única e mais complexa. O `-` operador na expressão `?x` é um operador unário que executa uma operação de negação no operando `x`. Finalmente, o JavaScript suporta um operador ternário, o operador condicional `?:`, que combina três expressões em uma única expressão. 4.7.2 Operando e tipo de resultado Alguns operadores trabalham em valores de qualquer tipo, mas a maioria espera um operando para ser de um tipo específico, e a maioria dos operadores retorna (ou avalia para) um valor de um tipo específico. A coluna Tipos na Tabela 4-1 especifica Tipos de operando (antes da seta) e tipo de resultado (após a seta) para os operadores. Os operadores JavaScript geralmente convertem o tipo (ver §3.9) de seu operando conforme necessário. O operador de multiplicação `*` espera numérico

operando, mas a expressão "3" * "5" é legal porque JavaScript pode converter os operandos em números. O valor dessa expressão é o número 15, não a corda "15", é claro. Lembre-se também disso: todo valor de JavaScript é "verdade" ou "falsidade", então os operadores que esperam booleanos funcionam com um operando de qualquer tipo. Alguns operadores se comportam de maneira diferente, dependendo do tipo de operando usado: ?? com eles. Mais notavelmente, o operador + adiciona numérico operando, mas concatena operandos de string. Da mesma forma, a comparação operadores como < executar comparação em numéricos ou alfabéticos. Enquanto isso, dependendo do tipo de operandos. As descrições individuais explicam suas dependências de tipo e especificam o que cada conversões que eles executam. Observe que os operadores de atribuição e alguns dos outros operadores estão listados na Tabela 4-1. Espere um operando do tipo LVAL. Ivalue é a termo histórico que significa uma expressão que pode aparecer legalmente no lado esquerdo de uma expressão de atribuição. Em JavaScript, variáveis, propriedades dos objetos e elementos das matrizes são Ivalues.

4.7.3 Efeitos colaterais do operador

Avaliar uma expressão simples como $2 * 3$ nunca afeta o estado de seu programa e qualquer computação futura que seu programa executa será afetado por essa avaliação. Algumas expressões, no entanto, têm efeitos colaterais e sua avaliação podem afetar o resultado do futuro. Avaliações. Os operadores de atribuição são o exemplo mais óbvio: se você atribui um valor a uma variável ou propriedade, que altera o valor de qualquer expressão que use essa variável ou propriedade. O ++ e -

Os operadores de incremento e decréscimos são semelhantes, pois realizam uma atribuição implícita. O operador de exclusão também tem efeitos colaterais: Excluir uma propriedade é como (mas não o mesmo que) atribuir para a propriedade. Nenhum outro operador JavaScript tem efeitos colaterais, mas a invocação de funções e as expressões de criação de objetos terão efeitos colaterais se algum dos operadores usados ?? no corpo da função ou do construtor têm efeitos colaterais.

4.7.4 Precedência do operador

Os operadores listados na Tabela 4-1 estão dispostos em ordem de alta precedência a baixa precedência, com linhas horizontais separando grupos de operadores no mesmo nível de precedência. A precedência do operador controla a ordem em que as operações são executadas. Operadores com maior precedência (mais próxima da parte superior da tabela) é realizada antes que os com menor precedência (mais próxima do fundo).

Considere a seguinte expressão: $w = x + y * z;$ O operador de multiplicação * tem uma precedência maior que a adição. Operador +, portanto a multiplicação é realizada antes da adição. Além disso, o operador de atribuição = tem a menor precedência, então a tarefa é realizada depois de todas as operações no lado direito estando concluídos.

A precedência do operador pode ser substituída pelo uso explícito de parênteses. Para forçar a adição no exemplo anterior a ser

executado primeiro, escreva:`w = (x + y)*z;`Observe que as expressões de acesso e invocação de propriedades têm maiorPrecedência do que qualquer um dos operadores listados na Tabela 4-1.Considere issoexpressão:// meu é um objeto com uma propriedade chamada funções cujoO valor é um// Matriz de funções.Invocamos o número da função X, passandoargumento// y, e então solicitamos o tipo de valor retornado.tipo de my.funções [x] (y)Embora o tipo de seja um dos operadores de maior prioridade, oTipoof Operação é realizada no resultado do acesso à propriedade,índice de matriz e invocação de funções, todos com maior prioridadedo que operadores.Na prática, se você não tiver certeza sobre a precedência de seuoperadores, a coisa mais simples a fazer é usar parênteses para fazer oOrdem de avaliação explícita.As regras que são importantes a saber sãoestes: multiplicação e divisão são realizadas antes da adição esubtração, e a atribuição tem muito baixa precedência e é quaseSempre executado por último.Quando novos operadores são adicionados ao JavaScript, eles nem sempre se encaixamnaturalmente nesse esquema de precedência.O ??Operador (§4.13.2) éMostrado na tabela como menor precedência que ||e &&, mas, de fato, seuPrecedência em relação a esses operadores não está definida e ES2020Requer que você use explicitamente parênteses se você misturar ??com qualquer um deles ||

ou `&&`. Da mesma forma, o novo operador de exponenciação não possui um precedência bem definida em relação ao operador de negação unário evocê deve usar parênteses ao combinar a negação com `Exp`.
4.7.5
Associatividade do operador Na Tabela 4-1, a coluna rotulou como específica a associatividade do operador. Um valor de L especifica a associativa da esquerda para a direita e um valor de R especifica a associativa da direita para a esquerda. A associatividade de um operador especifica a ordem em que operações da mesma precedências são realizadas. Associatividade da esquerda para a direita significa que as operações são realizadas da esquerda para a direita. Por exemplo, o operador de subtração tem associatividade da esquerda para a direita, então: $w = x - y - z$ é o mesmo que: $w = ((x - y) - z)$; Por outro lado, as seguintes expressões: $y = a^{**} b^{**} c$; $x = \sim -y$; $w = x = y = z$; P = a? B: C? D: E? F: G; são equivalentes a: $y = (a^{**} (b^{**} c))$; $x = \sim (-y)$;

w = (x = (y = z)); q = a? B: (c? D: (e? f: g)); Porque a exponenciação, unário, atribuição e condicional ternário Os operadores têm associativa do direito para a esquerda. 4.7.6 Ordem de avaliação Precedência e Associatividade do Operador Especifique a Ordem em que As operações são realizadas em uma expressão complexa, mas não Especifique a ordem em que as subexpressões são avaliadas. JavaScript Sempre avalia expressões em ordem estritamente da esquerda para a direita. No expressão $w = x + y * z$, por exemplo, a subexpressão w é avaliada primeiro, seguido por x , y e z . Então os valores de Y e Z são multiplicado, adicionado ao valor de x e atribuído à variável w . Propriedade especificada pela expressão w . Adicionando parênteses a expressões podem alterar a ordem relativa da multiplicação, adição e atribuição, mas não a ordem de avaliação da esquerda para a direita. A ordem de avaliação só faz a diferença se alguma das expressões ser avaliada tem efeitos colaterais que afetam o valor de outro expressão. Se a expressão x incrementos uma variável que é usada por expressão Z , então o fato de X ser avaliado antes de Z ser importante. 4.8 Expressões aritméticas Esta seção abrange os operadores que executam aritmética ou outra manipulações numéricas em seus operandos. A exponenciação, Os operadores de multiplicação, divisão e subtração são diretos

e são cobertos primeiro.O operador de adição recebe uma subseção própriaPorque também pode executar concatenação de string e tem alguns incomunsTipo Regras de conversão.Os operadores unários e os operadores bitwisetambém são cobertos por subseções próprias.A maioria desses operadores aritméticos (exceto como observado como segue) pode ser usado com bigint (ver §3.2.5) operando ou com números regulares, comodesde que você não misture os dois tipos.Os operadores aritméticos básicos são ** (exponenciação), *(multiplicação), / (divisão), % (Modulo: restante após a divisão), +(adição) e - (subtração).Como observado, discutiremos o operador +em uma seção própria.Os outros cinco operadores básicos simplesmente avaliam seus operando, converte os valores em números, se necessário, e depoisCalcule o poder, produto, quociente, restante ou diferença.Não-operando numéricos que não podem se converter em números convertidos para a nanvalor.Se qualquer um opera é (ou converter para) nan, o resultado da operação é (quase sempre) nan.O operador ** tem maior precedência que *, /e % (que por sua veztem maior precedência que + e -).Ao contrário dos outros operadores, **Funciona a direita para a esquerda, então $2^{**} 2^{**} 3$ é o mesmo que $2^{**} 8$, não $4^{**} 3$.Há uma ambiguidade natural em expressões como $-3^{**} 2$.Dependendo dea relativa precedência de unário menos e exponenciação, queA expressão pode significar $(-3)^{**} 2$ ou $-(3^{**} 2)$.Diferentes idiomaslidar com isso de maneira diferente e, em vez de escolher lados, JavaScript simplesmentetorna um erro de sintaxe omitir parênteses neste caso, forçando você aEscreva uma expressão inequívoca.** é a mais nova aritmética de JavaScript

Operador: foi adicionado ao idioma com ES2016. A função Math.pow () está disponível desde as primeiras versões de JavaScript, no entanto, ele executa exatamente a mesma operação que o operador **. O operador divide seu primeiro operando pelo segundo. Se você está acostumado a linguagens de programação que distinguem entre número inteiro e flutuante, você pode esperar obter um resultado inteiro quando você divide um número inteiro por outro. Em JavaScript, no entanto, todos os números são ponto flutuante, portanto, todas as operações de divisão têm resultados de ponto flutuante: 5/2 avalia para 2,5, não 2. Infinidade negativa, enquanto 0/0 avalia para NaN: nenhum desses casos levanta um erro. O operador % calcula o primeiro módulo de operando pelo segundo operando. Em outras palavras, ele retorna o restante após a divisão de número inteiro do primeiro operando pelo segundo operando. O sinal do resultado é o mesmo que o sinal do primeiro operando. Por exemplo, 5 % 2 avalia para 1 e -5 % 2 avalia para -1. Enquanto o operador do módulo é normalmente usado com operandos inteiros, ele também funciona para valores de ponto flutuante. Por exemplo, 6,5 % 2.1 Avalia para 0,2.

4.8.1 O operador + O operador binário + adiciona operandos numéricos ou concatena uma string:

`1 + 2 // => 3`"Olá" + "" + "lá" // => "Olá""1" + "2" // => "12"Quando os valores de ambos os operando são números ou são ambas as cordas,Então é óbvio o que o operador + faz.Em qualquer outro caso, no entanto,A conversão do tipo é necessária e a operação a ser realizadaDepende da conversão realizada.As regras de conversão para + dêprioridade à concatenação da string: se qualquer um dos operandos for uma string ouum objeto que se converte em uma string, o outro operando é convertido em umString e concatenação são realizadas.A adição é realizada apenas seNenhuma operando é semelhante a uma corda.Tecnicamente, o operador + se comporta assim:Se um de seus valores de operando for um objeto, ele o converte em umprimitivo usando o algoritmo de objeto a princípio descrito em§3.9.3.Os objetos de data são convertidos por seu toque ()método e todos os outros objetos são convertidos via `valueof ()`,Se esse método retornar um valor primitivo.No entanto, a maioriaObjetos não têm um método de valor útil (), então sãoconvertido via `ToString ()` também.Após a conversão de objeto para primitivo, se qualquer um operando for umstring, a outra é convertida em uma corda e a concatenação érealizado.Caso contrário, ambos os operando são convertidos em números (ou paraNan) e adição é realizada.Aqui estão alguns exemplos:`1 + 2 // => 3`: adição"`1`" + "`2`" // => "`12`": concatenação"`1`" + `2` // => "`12`": concatenação após número a-

`corda1 + {} // => "1 [objeto objeto]"`: concatenação após objeto para corda verdadeiro + verdadeiro // => 2: adição após boolean-to-number 2 + nulo // => 2: adição após o nulo se converte para 0 2 + indefinido // => nan: adição após convertidos indefinidos para NaN. Finalmente, é importante observar que quando o operador + é usado com strings e números, pode não ser associativa. Isto é, o resultado pode depender da ordem em que as operações são executadas. Por exemplo:

```
1 + 2 + "camundongos cegos" // => "3 ratos cegos"
1 + (2 + "camundongos cegos") // => "12 camundongos cegos"
```

A primeira linha não tem parênteses e o operador + tem da esquerda para a direita a associatividade, então os dois números são adicionados primeiro e sua soma é concatenada com a string. Na segunda linha, parênteses alteram isso.

Ordem de operações: o número 2 é concatenado com a string para produzir uma nova corda. Então o número 1 é concatenado com o novo string para produzir o resultado final.

4.8.2 Operadores aritméticos unários

Operadores unários modificam o valor de um único operando para produzir um novo valor. Em JavaScript, todos os operadores unários têm alta precedência e são todos bem associativos. Os operadores aritméticos unários descritos em esta seção (+, -, ++ e -) todos convertem seu único operando em um número, se necessário. Observe que os caracteres de pontuação + e - são usados como operadores unários e binários.

Os operadores aritméticos unários são os seguintes:
Unário mais (+) O operador unário mais converte seu operando em um número (ou emNaN) e retorna o valor convertido. Quando usado com um operando que já é um número, não faz nada. Este operador pode não ser usado com valores bigint, pois eles não podem ser convertidos em números regulares.
Unário menos (-) Quando - é usado como um operador unário, ele converte seu operando em um número, se necessário, e altera o sinal do resultado.
Incremento (++) O operador ++ incrementa (ou seja, adiciona 1 a) seu único operando, que deve ser um lvalue (uma variável, um elemento de uma matriz ou uma propriedade de um objeto). O operador converte seu operando em um número, adiciona 1 a esse número e atribui o valor incrementado de volta à variável, elemento ou propriedade. O valor de retorno do operador ++ depende da sua posição parentética no operando. Quando usado antes do operando, onde é conhecido como operador de pré-incremento, ele aumenta o operando e avalia o valor incrementado desse operando. Quando usado depois do operando, onde é conhecido como operador pós-incremento, ele incrementa seu operando, mas avalia o valor não incrementado de aquele operando. Considere a diferença entre essas duas linhas de código:
Seja i = 1, j = ++ i; // i é 1 e j é 2
Seja n = 1, m = n ++; // n é 1, m é 2

Observe que a expressão `x ++` nem sempre é a mesma que `x = x+1`. O operador `++` nunca executa concatenação de string: sempre converte seu operando em um número e o incrementa. Se `x` é a corda "1", `++ x` é o número 2, mas `x+1` é a string "11". Observe também que, devido ao semicolon automático do JavaScriptInserção, você não pode inserir uma quebra de linha entre o pós-incrementooperador e o operando que o precede. Se você fizer isso, JavaScripttratará o operando como uma declaração completa por si só e inserirá umSemicolon antes dele. Este operador, tanto em suas formas de pré e pós-incremento, é a maioria comumente usado para incrementar um contador que controla um loop(§5.4.4). Decremento (`-`) O operador espera um operando LValue. Converte o valor deo operando a um número, subtrai 1 e atribui o decrementadoValor de volta ao operando. Como o operador `++`, o valor de retorno de- depende de sua posição em relação ao operando. Quando usadoAntes do operando, ele diminui e retorna o decrementadovalor. Quando usado após o operando, diminui o operando, masRetorna o valor indecremido. Quando usado após seu operando, nãoA quebra de linha é permitida entre o operando e o operador.

4.8.3 Operadores bitwise

Os operadores bitwiseRepresentação binária de números. Embora eles não realizemOperações aritméticas tradicionais, elas são classificadas como aritméticaoperadores aqui porque operam em operandos numéricos e retornam umvalor numérico. Quatro desses operadores realizam álgebra booleana noBits individuais dos operandos, se comportando como se cada um pouco em cada operando

eram um valor booleano (1 = true, 0 = false). Os outros três bits operadores são usados ?? para mudar os bits para a esquerda e para a direita. Esses operadores não são comumente usado na programação JavaScript, e se você não estiver familiarizado com a representação binária de números inteiros, incluindo os dois complementos. Representação de números inteiros negativos, você provavelmente pode pulá-la para a seção. Os operadores bitwise esperam operando inteiros e se comportar. Os valores foram representados como números inteiros de 32 bits em vez de flutuação de 64 bits. Os valores pontuais. Esses operadores convertem seu operando em números, se necessário e depois coagir os valores numéricos a números inteiros de 32 bits, preservando qualquer parte fracionária e quaisquer bits além do 32º. A mudança. Os operadores exigem um operando do lado direito entre 0 e 31. Depois de convertendo este operando em um número inteiro de 32 bits não assinado, eles abandonam qualquer bit além do 5º, que produz um número no intervalo apropriado. Surpreendentemente, NaN, infinito e -infinity todos se convertem para 0. Quando usado como operando desses operadores bit -bit. Todos esses operadores bit new, exceto >>> podem ser usados ?? com regular operando numéricos ou com operando bigint (consulte §3.2.5). Bitwise e (&) O Operador executa uma operação booleana em cada bit de seus argumentos inteiros. Um pouco está definido no resultado apenas se o bit correspondente é definido nos dois operandos. Por exemplo, 0x1234 & 0x00FF Avalia para 0x0034. Bit nessa ou (|) O | O operador executa uma operação booleana ou operação em cada bit de seu

argumentos inteiros.Um pouco está definido no resultado se o bit correspondente está definido em um ou ambos os operandos.Por exemplo, 0x1234 |0x00FF Avalia para 0x12ff.Bitwise xor (^)O operador ^ realiza um exclusivo booleano ou operação em cada um pouco de seus argumentos inteiros.Exclusivo ou significa que também O operando um é verdadeiro ou o operando dois é verdadeiro, mas não ambos.Um pouco é definido no resultado desta operação se um bit correspondente for definido em um (mas não ambos) dos dois operandos.Por exemplo, 0xff00 ^ 0xf0f0Avalia para 0x0FF0.Bitwise não (~)O operador ~ é um operador unário que aparece antes de seu único operando inteiro.Opera revertendo todos os bits no operando.Por causa da maneira como os números inteiros assinados estão representados em JavaScript,Aplicar o operador ~ a um valor é equivalente a mudar seu sinal e subtrair 1. Por exemplo, ~ 0x0f avalia para 0xfffffffff0, ou -16.Mudança para a esquerda (<<)O operador << move todos os bits em seu primeiro operando para a esquerda pelo número de lugares especificados no segundo operando, que deve ser um número inteiro entre 0 e 31. Por exemplo, na operação A <<1, o primeiro bit (o pouco) de A se torna o segundo bit (os dois bits), o segundo bit de A se torna o terceiro, etc. Um zero é usado para o novo primeiro bit e os valores do 32º bit são perdidos.Mudando a posição deixada por uma posição é equivalente à multiplicação por 2, mudando duas posições são equivalentes a multiplicar por 4 e assim por diante.Para Exemplo, 7 << 2 avalia para 28.Mudar bem com o sinal (>>)

O operador `>>` move todos os bits em seu primeiro operando para a direita por um número de lugares especificados no segundo operando (um número inteiro entre 0 e 31). Bits que são deslocados para a direita são perdidos. Os bits preenchidos à esquerda dependem do bit de sinal do original operando, a fim de preservar o sinal do resultado. Se o primeiro operando é positivo, o resultado tem zeros colocados nos bits altos; se o primeiro operando é negativo, o resultado possui aqueles colocados na alta bits. Mudar um valor positivo para o lado, um lugar é equivalente a dividindo por 2 (descartando o restante), mudando para a direita dois lugares é equivalente à divisão inteira até 4, e assim por diante.

`7 >> 1` avalia 3, por exemplo, mas observe que `-7 >> 1` avalia como -4. Mudar à direita com preenchimento zero (`>>>`) O operador `>>>` é como o operador `>>`, exceto que os bits mudados para a esquerda são sempre zero, independentemente do sinal do primeiro operando. Isso é útil quando você deseja tratar 32 bits assinados valores como se fossem números inteiros não assinados. `?1 >> 4` avalia para -1, mas `?1 >>> 4` avalia para `0x0fffffff`, por exemplo. Isso é o único dos operadores JavaScript bit-new que não pode ser usado com valores bigint. Bigint não representa números negativos por definindo a parte alta da maneira que os números inteiros de 32 bits, e este operador faz sentido apenas para o complemento desses dois em particular representação.

4.9 Expressões relacionais

Esta seção descreve os operadores relacionais da JavaScript. Esses operadores testam um relacionamento (como "iguais", "menos que" ou "maior que") entre dois valores e retornam verdadeiro ou falso dependendo se esse relacionamento existe. Expressões relacionais sempre avaliam com um valor booleano, e esse valor é frequentemente usado para controlar o fluxo de execução do programa em se, enquanto e para.

Declarações (consulte o Capítulo 5). As subseções a seguir documentam os operadores de igualdade e desigualdade, os operadores de comparação e os outros dois operadores relacionais do JavaScript, dentro e a instância.

4.9.1 Operadores de igualdade e desigualdade

Os operadores `==` e `===` verificam se dois valores são iguais, usando duas definições diferentes de mesma. Ambos os operadores aceitam operando de qualquer tipo, e ambos retornam verdadeiro se seus operandos forem o mesmo e falso se forem diferentes. O operador `==` é conhecido como operador de igualdade (ou às vezes o operador de identidade), e ele verifica se seus dois operandos são "idênticos" usando uma definição estrita de mesma. O operador `==` é conhecido como operador de igualdade; isto verifica se seus dois operandos são "iguais" usando um mais relaxado. Definição de semelhança que permite conversões de tipo. O `!=` é o operador que testa exatamente o oposto do `==`. Os operadores de desigualdade retornam falso se os dois valores são iguais um ao outro de acordo com `==` e retornam verdadeiro de outra forma. O `!==` é o operador que retorna falso se os dois valores forem estritamente iguais um ao outro e retorna verdadeiro de outra forma. Como você verá em §4.10, o `!=` operador calcula a operação booleana e não. Isso faz isso fácil de lembrar disso! `!=` significa "não igual a" e "não estritamente igual a". O `=`, `==` e `===` operadores JavaScript suportam `=`, `==` e `===` operadores. Certifique-se de entender as diferenças entre estes operadores de igualdade, igualdade e igualdade rigorosa e tome cuidado para usar o correto ao codificar! Embora seja tentador ler os três operadores como "iguais", pode ajudar a reduzir a confusão se você

Leia "Gets" ou "é atribuído" para `=`, "é igual a" para `==`, e "é estritamente igual a" para `===`. O operador `==` é um recurso herdado do JavaScript e é amplamente considerado uma fonte de bugs. Você quase sempre deve usar `===` em vez de `==`, e `!=` em vez de `!=`. Como mencionado no §3.8, os objetos JavaScript são comparados por referência, não por valor. Um objeto é igual a si mesmo, mas não a nenhum outro objeto. Se dois objetos distintos têm o mesmo número de propriedades, com os mesmos nomes e valores, eles ainda não são iguais. Da mesma forma, duas matrizes que têm os mesmos elementos na mesma ordem não são iguais um ao outro.

O Operador de Igualdade Estrita `===` Avalia seus operandos e depois compara os dois valores da seguinte maneira, executando nenhuma conversão de tipo:

- Se os dois valores tiverem tipos diferentes, eles não são iguais.
- Se ambos os valores forem nulos ou ambos os valores forem indefinidos, eles são iguais.
- Se ambos os valores são o valor booleano verdadeiro ou ambos são o valor booleano Falso, eles são iguais.
- Se um ou ambos os valores forem `NaN`, eles não são iguais. (Isso é surpreendente, mas o valor `NaN` nunca é igual a nenhum outro valor, inclusive a si mesmo! Para verificar se um valor `x` é `NaN`, use `x === NaN`, ou a função global `isNaN()`.)
- Se ambos os valores forem números e têm o mesmo valor, eles são iguais.
- Se um valor for `0` e o outro é `-0`, eles também são iguais.
- Se ambos os valores forem strings e conter exatamente os mesmos 16 bits de valores (veja a barra lateral em §3.3) nas mesmas posições, eles são iguais.
- Se as cordas diferirem em comprimento ou conteúdo, elas não são iguais.

igual.Duas cordas podem ter o mesmo significado e o mesmoaparência visual, mas ainda ser codificada usando diferenteSequências de valores de 16 bits.JavaScript não executa o Unicodenormalização, e um par de cordas como essa não é considerado igual aos operadores `==` ou `==`.Se ambos os valores se referirem ao mesmo objeto, matriz ou função, elessão iguais.Se eles se referirem a objetos diferentes, eles não são iguais,Mesmo que ambos os objetos tenham propriedades idênticas.Igualdade com conversão de tipoO operador de igualdade `==` é como o operador estrito da igualdade, mas é menos rigoroso.Se os valores dos dois operandos não forem do mesmo tipo, eletenta algum tipo de conversões e tenta a comparação novamente:Se os dois valores tiverem o mesmo tipo, teste -os para rigorosamente igualdade como descrito anteriormente.Se eles são estritamente iguais, elessão iguais.Se eles não são estritamente iguais, não são iguais.Se os dois valores não tiverem o mesmo tipo, o operador `==`ainda pode considerá -los iguais.Ele usa as seguintes regras eDigite conversões para verificar a igualdade:Se um valor for nulo e o outro é indefinido,Eles são iguais.Se um valor é um número e o outro é uma string,Converta a string em um número e tente a comparaçãoNovamente, usando o valor convertido.Se qualquer um dos valores for verdadeiro, converta -o em 1 e tente ocomparação novamente.Se qualquer um dos valores for falso, converta -opara 0 e tente a comparação novamente.Se um valor é um objeto e o outro é um número oustring, converta o objeto em um primitivo usando o

algoritmo descrito no §3.9.3 e tente a comparação de novo. Um objeto é convertido em um valor primitivo por ou o método `ToString()` ou seu `valueOf()` método. As classes internas do JavaScript Core Tentativa de `Valueof()` Antes `ToString()` conversão, exceto a classe de data, que executa conversão de `toString()`. Quaisquer outras combinações de valores não são iguais. Como exemplo de teste de igualdade, considere a comparação: "1" == true // => true Esta expressão avalia para verdadeiro, indicando que estes muito diferentes Os valores de aparência são de fato iguais. O valor booleano true é o primeiro convertido para o número 1 e a comparação é feita novamente. Em seguida, o String "1" é convertida para o número 1. Como ambos os valores são agora o mesmo, a comparação retorna true.

4.9.2 Operadores de comparação Os operadores de comparação testam a ordem relativa (numérica ou alfabetico) de seus dois operandos: Menos que (<) O operador avalia como verdadeiro se seu primeiro operando for menor que o seu segundo operando; Caso contrário, ele avalia como falso. Maior que (>) O operador avalia como verdadeiro se seu primeiro operando for maior que seu segundo operando; Caso contrário, ele avalia como falso.

Menor ou igual (\leq) O operador \leq avalia para true se seu primeiro operando for menor que ou igual ao seu segundo operando; Caso contrário, ele avalia como falso. Maior ou igual (\geq) O operador \geq avalia para true se seu primeiro operando for maior ou igual ao seu segundo operando; Caso contrário, ele avalia para falso. Os operandos desses operadores de comparação podem ser de qualquer tipo. A comparação pode ser realizada apenas em números e cordas, no entanto, Portanto, operandos que não são números ou strings são convertidos. Comparação e conversão ocorrem da seguinte forma: Se um operando avaliar para um objeto, esse objeto é convertido em um valor primitivo, conforme descrito no final de §3.9.3; Se o método `valueof()` retornar um valor primitivo, esse valor é usado. Caso contrário, o valor de retorno de seu método `toString()` é usado. Se, após qualquer conversão de objeto para princípio necessário, ambos operando são strings, as duas cordas são comparadas, usando a ordem alfabética, onde a ordem alfabética é definida por ordem numérica dos valores de unicode de 16 bits que compõem as cordas. Se, após a conversão de objeto para princípio, pelo menos um operando é não é uma string, ambos os operando são convertidos em números e comparado numericamente. 0 e -0 são considerados iguais. O infinito é maior que qualquer número diferente de si mesmo e -O infinito é menor do que qualquer número que não seja. Se o operando é (ou converte para) nan, depois a comparação

O operador sempre retorna falso. Embora a aritmética Os operadores não permitem que os valores do BIGINT sejam misturados com regular Números, os operadores de comparação permitem comparações entre números e bigints. Lembre-se de que as cordas JavaScript são sequências de inteiro de 16 bits valores, e essa comparação de string é apenas uma comparação numérica de os valores nas duas cordas. A ordem de codificação numérica definida por Unicode pode não corresponder à ordem de agrupamento tradicional usada em qualquer idioma ou localidade particular. Observe em particular essa comparação de string é sensível ao caso, e todas as cartas de capital ASCII são ?menos do que? letras ascii minúsculas. Esta regra pode causar resultados confusos se você fizernão espere. Por exemplo, de acordo com o < operador, a string "Zoo" vem antes da string "Aardvark". Para um algoritmo de comparação de string mais robusto, tente o String.localeCompare () Método, que também toma localidade-Definições específicas de ordem alfabética. Para caso-Comparações insensíveis, você pode converter as seqüências para todas as minúsculas ou Toda a maçaneta usando string.toLowerCase () ou String.TOUUPPERCASE (). E, para um mais geral e melhor Ferramenta de comparação de string localizada, use a classe Intl.Collator descrita em §11.7.3. Tanto o operador + quanto os operadores de comparação se comportam de maneira diferente para operando numérico e string. + favorece as cordas: ele executa Concatenação se um operando for uma string. Os operadores de comparação números favoritos e executar apenas comparação de string se ambos os operando forem Strings:

`1 + 2 // => 3: adição.`"1" + "2" // => "12": concatenação.`"1" + 2 // => "12": 2 é convertido em "2".`11 <3 // => Falso: Comparação numérica.`"11" <"3" // => true: comparação de string.`"11" <3 // => False: Comparação numérica, "11" convertida para 11."um" <3 // => false: comparação numérica, "um" convertida para nan.Finalmente, observe que o <= (menor ou igual) e >= (maior que ou igual) operadores não confiam na igualdade ou em operadores estritos de igualdade para determinar se dois valores são "iguais".Em vez disso, o menos do que operador ou igual é simplesmente definido como "não maior que" e o operador maior do que ou igual é definido como "não menor que".O que a exceção ocorre quando um operando é (ou converte para) nan, em que caso, todos os quatro operadores de comparação retornam falsos.4.9.3 O operador no operator in espera um operando do lado esquerdo que é uma corda, símbolo ou valor que pode ser convertido em uma string.Espeta um operando do lado direito isso é um objeto.Ele avalia para verdadeiro se o valor do lado esquerdo for o nome de uma propriedade do objeto do lado direito.Por exemplo: deixe Point = {x: 1, y: 1}; // Defina um objeto "X" no ponto // => true: objeto tem propriedade chamada "X""Z" no ponto // => false: objeto não tem "z" propriedade."`ToString`" no ponto // => true: objeto herda método da `ToString` deixe dados = [7,8,9]; // Uma matriz com elementos (índices) 0, 1 e 2"0" em dados // => true: a matriz tem um elemento "0"

1 em dados // => true: os números são convertidos para cordas3 em dados // => false: nenhum elemento 34.9.4 A instância do operador A instância do operador espera um operando do lado esquerdo que é um objeto e um operando do lado direito que identifica uma classe de objetos. O operador avalia como verdadeiro se o objeto do lado esquerdo for uma instância da classe do lado direito e avalia o False, caso contrário. O capítulo 9 explica que, em JavaScript, as classes de objetos são definidas pelo construtor função que os inicializa. Assim, o operando do lado direito de instanceof deve ser uma função. Aqui estão exemplos: Seja d = new Date(); // Crie um novo objeto com a data () constructor instance of Date // => true: d foi criado com date () D instance of object // => true: Todos os objetos são instâncias de Object D instance of number // => false: D não é um objeto numérico Seja a = [1, 2, 3]; // Crie uma matriz com matriz literais uma instância de matriz // => true: a é uma matriz uma instância de objeto // => true: todas as matrizes são objetos uma instância de regexp // => false: as matrizes não são regulares expressões Observe que todos os objetos são instâncias de objeto. Instância desconsidera as "superclasses" ao decidir se um objeto é uma instância de uma classe. Se o operando do lado esquerdo da instância do objeto, a instância de retorna falsa. Se o lado da direita não for uma classe de objetos, ele lança um TypeError. Para entender como a instância do operador funciona, você deve entender a "cadeia de protótipos". Esta é a herança de JavaScript

mecanismo, e é descrito em §6.3.2. Para avaliar a expressão oInstância de F, JavaScript avalia F.Prototype e, em seguida, Procura esse valor na cadeia de protótipos de O. Se encontrar, então O éUma instância de f (ou de uma subclasse de f) e o operador retorna true. Se F. prototipo não for um dos valores na cadeia de protótipos de O, Então O não é uma instância de F e a instância de retorna falsa.

4.10 Expressões lógicas

Os operadores lógicos &&, || e! executar álgebra booleana e sãofrequentemente usado em conjunto com os operadores relacionais para combinar dois Expressões relacionais em uma expressão mais complexa. Esses Os operadores são descritos nas subseções a seguir. Para totalmenteentenda -os, você pode querer revisar o conceito de "verdade" e Valores falsamente? introduzidos no §3.4.4.10.1 lógico e (&&) O operador && pode ser entendido em três níveis diferentes. Nonível mais simples, quando usado com operandos booleanos, && executa o Booleano e operação nos dois valores: ele retorna true se e somenteSe seu primeiro operando e seu segundo operando forem verdadeiros. Se um ou ambos Destes operandos é falso, ele retorna falsa. && é frequentemente usado como uma conjunção para ingressar em duas expressões relacionais: x === 0 && y === 0 // true se, e somente se, x e y são Ambos 0

Expressões relacionais sempre avaliam como verdadeiro ou falso, então quando usado assim, o próprio operador `&&` retorna verdadeiro ou falso. Os operadores relacionais têm maior precedência que `&&` (`e ||`), então expressões como essas podem ser escritas com segurança sem parênteses. Mas `&&` não exige que seus operandos sejam valores booleanos. Lembre-se disso: todos os valores de JavaScript são "verdadeiros" ou "falsamente". (Veja §3.4 para obter detalhes. Os valores falsamente são falsos, nulos, indefinidos, 0, -0, `nan` e ""). Todos os outros valores, incluindo todos os objetos, são verdadeiros.) O segundo nível em que `&&` pode ser entendido é como um booleano e operador para valores verdadeiros e falsamente. Se ambos os operandos são verdadeiros, o operador retorna um valor verdadeiro. Caso contrário, um ou ambos os operandos devem ser falsamente, e o operador retorna um valor falsamente. Em JavaScript, qualquer expressão ou declaração que espera que um valor booleano funcione com um verdadeiro ou falsamente valor, então o fato de que `&&` nem sempre retorna verdadeiro ou falso não causará problemas práticos. Observe que esta descrição diz que o operador retorna ?um verdadeiro valor ?ou? um valor falsamente ?, mas não especifica o que é esse valor. Para isso, precisamos descrever `&&` no terceiro e último nível. Este operador começa avaliando seu primeiro operando, a expressão à sua esquerda. Se o valor à esquerda é falsamente, o valor de toda a expressão também deve ser falsy, então `&&` simplesmente retorna o valor à esquerda e nem mesmo avalia a expressão à direita. Por outro lado, se o valor à esquerda for verdadeiro, então o geral o valor da expressão depende do valor do lado direito. Se o valor à direita é verdadeiro, então o valor geral deve ser verdade,

e se o valor à direita for falsamente, o valor geral deve ser falsidade. Então, quando o valor à esquerda é verdadeiro, o operador de `&&` avalia e retorna o valor à direita: Seja `o = {x: 1};` Seja `p = nulo;` `O && o.x // => 1;` `o` é verdade, então retornar o valor de `O.x` `&& p.x // => null;` `p` é falsidade, então devolva e não avalie `P.X`. É importante entender que `&&` pode ou não avaliar seu direito operando lateral. Neste exemplo de código, a variável `p` é definida como nula e a expressão `P.X`, se avaliada, causaria um `TypeError`. Mas os usos de código `&&` de maneira idiomática para que o `P.X` seja avaliado apenas se `P` for verdade - não nula ou indefinida. O comportamento de `&&` às vezes é chamado de curto-circuito, e você pode às vezes ver o código que explora propositalmente esse comportamento para executar o código condicionalmente. Por exemplo, as duas linhas a seguir de JavaScript tem efeitos equivalentes:

```
if (a === b) stop(); // Invocar Stop()
Somente se A === B(a === b) && stop(); // Isso faz a mesma coisa
```

Em geral, você deve ter cuidado sempre que escrever uma expressão com efeitos colaterais (atribuições, incrementos, decrementos ou funções invocações) no lado direito de `&&`. Se esses efeitos colaterais ocorrem depende do valor do lado esquerdo. Apesar da maneira um tanto complexa que esse operador realmente funciona, é mais comumente usado como um simples operador de álgebra booleano que

Trabalha sobre valores verdadeiros e falsamente.4.10.2 Lógico ou (||)O ||operador executa o booleano ou operação em seus dois operando.Se um ou ambos os operandos é verdade, ele retornará um valor verdadeiro.Se Ambos os operandos são falsamente, retorna um valor falsamente.Embora o ||O operador é mais frequentemente usado simplesmente como booleano ou O operador, como o operador && tem um comportamento mais complexo.Começa Ao avaliar seu primeiro operando, a expressão à sua esquerda.Se o valor de Este primeiro operando é verdadeiro, é curto-circuito e retorno esse valor verdadeiro sem nunca avaliar a expressão à direita.Se, por outro Mão, o valor do primeiro operando é falsamente, então || avalia se use segundo operando e retorna o valor dessa expressão.Como no operador &&, você deve evitar operando do lado direito que inclua efeitos colaterais, a menos que você queira usar o fato de que o A expressão do lado direito não pode ser avaliada.Um uso idiomático deste operador é selecionar o primeiro valor verdadeiro em Um conjunto de alternativas:// Se MaxWidth for verdade, use isso.Caso contrário, procure um valor em // O objeto de preferências.Se isso não for verdade, use um constante codificada.Seja max = maxWidth || Preferências.MaxWidth || 500;Observe que se 0 for um valor legal para maxWidth, esse código não será Trabalhe corretamente, pois 0 é um valor falsamente.Veja o ??Operador (§4.13.2)

para uma alternativa. Antes do ES6, esse idioma é frequentemente usado em funções para fornecer padrãoValores para parâmetros:// copie as propriedades de O para P e retorne Pcópia da função (o, p) {P = P ||{}; // Se nenhum objeto foi aprovado para P, use um recémobjeto criado.// O corpo da função vai aqui}No ES6 e mais tarde, no entanto, esse truque não é mais necessário porque oO valor do parâmetro padrão pode ser simplesmente escrito na funçãoDefinição em si: função cópia (o, p = {}) {...}.4.10.3 Lógico não (!)O operador é um operador unário; É colocado antes de um único operando. Seu objetivo é inverter o valor booleano de seu operando. Por exemplo, seX é verdade ,! X avalia como falso. Se x é falsamente, então! X é verdadeiro. Ao contrário do && e ||operadores, !operador converte seu operando paraum valor booleano (usando as regras descritas no capítulo 3) antesinvertendo o valor convertido. Isso significa isso!sempre retorna verdadeiroou falso e que você pode converter qualquer valor x em seu equivalenteValor booleano aplicando este operador duas vezes: !! x (consulte §3.9.2). Como um operador unário ,!tem alta precedência e se liga firmemente. Se vocêdeseja inverter o valor de uma expressão como p && q, você precisa usarParênteses :! (P && q). Vale a pena notar duas leis de booleanos

Álgebra aqui que podemos expressar usando a sintaxe JavaScript:// Leis de Demorgan! $(p \&& q) === (! p || q)$ //
=> true: para todos os valores de p eq! $(p || q) === (! p \&& ! q)$ // => true: para todos os valores de p eq4.11
Expressões de atribuiçãoJavaScript usa o = operador para atribuir um valor a uma variável ou propriedade.Por exemplo:i = 0;// Defina a variável i como 0.O.x = 1;// Defina a propriedade X do objeto O como 1.O = operador espera que seu operando do lado esquerdo seja um LValue: uma variável ou propriedade de objeto (ou elemento de matriz).Espera seu operando do lado direito ser um valor arbitrário de qualquer tipo.O valor de uma tarefaExpressão é o valor do operando do lado direito.Como efeito colateral, o =O operador atribui o valor à direita à variável ou propriedade no lado esquerda para que as referências futuras à variável ou à propriedade avaliem para o valor.Embora as expressões de atribuição sejam geralmente bastante simples, você pode às vezes ver o valor de uma expressão de atribuição usada como parte de uma expressão maior.Por exemplo, você pode atribuir e testar um valor no mesmo expressão com código como este:(a = b) === 0Se você fizer isso, certifique -se de estar claro sobre a diferença entre o =

e === operadores!Observe que = tem muito baixa precedência e parênteses geralmente são necessários quando o valor de uma tarefa é usado em uma expressão maior.O operador de atribuição tem associatividade da direita para a esquerda, o que significa que quando vários operadores de atribuição aparecem em uma expressão, são avaliados da direita para a esquerda.Assim, você pode escrever código como este para Atribuir um único valor a várias variáveis:i = j = k = 0;// Inicialize 3 variáveis ??para 04.11.1 Atribuição com operaçãoAlém do operador normal = de atribuição, o JavaScript suporta um número de outros operadores de atribuição que fornecem atalhos por combinando atribuição com alguma outra operação.Por exemplo, o +=O operador executa adição e atribuição.A seguinte expressão:total += Salestax;é equivalente a este:TOTAL = TOTAL + SALESTAX;Como você pode esperar, o operador += funciona para números ou strings.Para operandos numéricos, ele executa adição e atribuição;para stringoperandos, executa concatenação e atribuição.Os operadores semelhantes incluem -=, *=, & = e assim por diante.A Tabela 4-2 os listados.

Tabela 4-2.Operadores de atribuiçãoOperadorExemploEquivalente
+= a += ba = a + b
-= a -= ba = a - b
*= a *= ba = a * b
/= a /= ba = a / b
%= a %= ba = a % b
**= a **= ba = a ** b
<<= a << ba = a << b
>>= a >> ba = a >> b
>>>= a >>> ba = a >>> b
&= a &= ba = a & b
|= a |= ba = a | b
^= a ^= ba = a ^ b
Na maioria dos casos, a expressão:
a op = b
Onde o OP é um operador, é equivalente à expressão:
a = a op b
Na primeira linha, a expressão A é avaliada uma vez.
No segundo, é avaliado duas vezes.
Os dois casos serão diferentes se um incluir o lado

efeitos como uma chamada de função ou um operador de incremento. A seguir Duas tarefas, por exemplo, não são as mesmas: dados [i ++] *= 2; dados [i ++] = dados [i ++] * 2; 4.12 Expressões de avaliação Como muitos idiomas interpretados, JavaScript tem a capacidade de interpretar strings de código -fonte JavaScript, avaliando -os para produzir um valor. JavaScript faz isso com a função global Eval (): Eval ("3+2") // => 5 A avaliação dinâmica de seqüências de código -fonte é uma linguagem poderosa Recurso que quase nunca é necessário na prática. Se você se encontrar Usando avaliar (), você deve pensar cuidadosamente sobre se você realmente precisa usá -lo. Em particular, avaliar () pode ser um buraco de segurança, e você nunca deve passar por qualquer string derivada da entrada do usuário para avaliar (). Com Uma linguagem tão complicada quanto JavaScript, não há como higienizar Entrada do usuário para tornar seguro o uso com avaliar (). Por causa disso Problemas de segurança, alguns servidores da web usam o HTTP ?Content-Security-POLÍTICA ?Cabeçalho para desativar avaliar () para um site inteiro. As subseções a seguir explicam o uso básico de avaliar () e Explique duas versões restritas que têm menos impacto no otimizador. É uma função ou um operador?

avaliar () é uma função, mas está incluído neste capítulo sobre expressões porque realmente deveria ter sido um operador. As versões mais antigas do idioma definiram uma função de avaliação () e desde então em seguida, designers de idiomas e escritores de intérpretes têm colocado restrições sobre ele que o tornam mais como operador. Interpretadores javascript modernos executam muita análise de código e otimização. De um modo geral, se uma função chama de avaliação (), o intérprete não pode otimizar que função. O problema de definir avaliar () em função é que ele pode receber outros nomes: Seja f = eval; Seja g = f; Se isso for permitido, o intérprete não pode ter certeza de quais funções chamam de avaliação (), para que não possa otimizar agressivamente. Esta questão poderia ter sido evitada se Eval () fosse um operador (e uma palavra reservada). Aprenderemos (em §4.12.2 e §4.12.3) sobre restrições colocadas em Eval () para fazê-lo mais como operador.

4.12.1 Eval ()

Eval () espera um argumento. Se você passar algum valor que não seja um String, simplesmente retorna esse valor. Se você passar por uma corda, ele tenta analisar a string como código JavaScript, lançando um SyntaxError se falhar. Se ele analisa com sucesso a string e avalia o código e retorna o valor da última expressão ou declaração na string ou undefined se a última expressão ou declaração não tivesse valor. Se o string avaliado lança uma exceção, que a exceção se propaga de volta para a chamada para avaliar (). A principal coisa sobre avaliar () (quando invocada assim) é que ele usa o ambiente variável do código que o chama. Isto é, ele olha para cima dos valores das variáveis ?? e define novas variáveis ?? e funções da mesma forma que o código local faz. Se uma função define uma variável local x e depois chama eval ("x"), ele obterá o valor do localvariável. Se ele chama de avaliar ("x = 1"), altera o valor do localvariável. E se a função chama avaliar ("var y = 3;"),

declara uma nova variável local y.Por outro lado, se avaliadoUsos de string let ou const, a variável ou constante declarada será local para a avaliação e não será definido na chamada ambiente.Da mesma forma, uma função pode declarar uma função local com código como este:avaliar ("função f () {return x+1;}");Se você ligar para EVEST () do código de nível superior, ele opera em variáveis ??globais e funções globais, é claro.Observe que a sequência de código que você passa para avaliar () deve fazer sintáticos.Sentir por conta própria: você não pode usar -lo para colar fragmentos de código em um função.Não faz sentido escrever avaliar ("retornar"), para exemplo, porque o retorno é apenas legal dentro das funções e o fato que a string avaliada usa o mesmo ambiente variável que oA função de chamada não faz parte dessa função.Se sua string faria sentido como um script independente (mesmo muito curto como x = 0), é legal passar para avaliar ().Caso SyntaxError.4.12.2 Global Eval ()É a capacidade de avaliar () alterar variáveis ??locais Problemático para otimizadores de JavaScript.Como solução alternativa, no entanto,Os intérpretes simplesmente fazem menos otimização em qualquer função que chamaEval ().Mas o que um intérprete de javascript deve fazer, no entanto, se um script define um alias para avaliar () e depois chama essa função por

Outro nome?A especificação JavaScript declara que quando quandoavaliar () é invocado por qualquer nome que não seja "avaliar", deve avaliar a string como se fosse o código global de nível superior.O código avaliado podeDefina novas variáveis ??globais ou funções globais, e pode definir globalvariáveis, mas não usarão ou modificarão nenhuma variável local para a chamadaFunção e, portanto, não interferirá nas otimizações locais.Uma "avaliação direta" é uma chamada para a função avaliação () com uma expressão que usa o nome exato e não qualificado "Eval" (que está começando a sentir como uma palavra reservada).Chamadas diretas para avaliar () usar a variável ambiente do contexto de chamada.Qualquer outra ligação - uma chamada indireta -usa o objeto global como seu ambiente variável e não pode ler,Escreva ou defina variáveis ??ou funções locais.(Tanto direto quanto indiretoAs chamadas podem definir novas variáveis ??apenas com var.Usos de let e constDentro de uma string avaliada, crie variáveis ??e constantes que são locaispara a avaliação e não altere o chamado ou ambiente global.)O código a seguir demonstra:const geval = avaliação;// usando outro nome fazUma avaliação globalSeja x = "global", y = "global";// duas variáveis ??globaisfunção f () {// Esta função faz umEval localSeja x = "local";// define uma variável localavaliar ("x += 'alterado';");// Conjuntos de avaliação direta Localvariávelretornar x;// retorno alterado localvariável}função g () {// Esta função faz umGlobal EvalSeja y = "local";// Uma variável local

geval ("y += 'alterado';");// Conjuntos de avaliação indiretamente variável global retornar y;// retorna local inalterado local variável}console.log (f (), x);// Variável local alterada: impressões "LocalChanged Global":console.log (g (), y);// variável global alterada: impressões "Local GlobalChanged":Observe que a capacidade de fazer uma avaliação global não é apenas uma acomodação para as necessidades do otimizador; é realmente um tremendo recurso útil que permite executar sequências de código como se estivessem scripts independentes de nível superior.Como observado no início desta seção, é raro precisar realmente avaliar uma sequência de código.Mas se você encontrar necessário, é mais provável que você queira fazer uma avaliação global do que um local eval.4.12.3 Eval rigoroso ()Modo rigoroso (ver §5.6.3) impõe restrições adicionais ao comportamento da função EVAL () e mesmo no uso do identificador "avaliar".Quando avaliar () é chamado do código de modo rigoroso ou quando a sequência de código a ser avaliado em si começa com uma diretiva de "uso rigoroso", entãoEval () faz uma avaliação local com um ambiente de variável privada.Esse significa que, no modo rigoroso, o código avaliado pode consultar e definir local variáveis, mas não pode definir novas variáveis ??ou funções no escopo local.Além disso, o modo rigoroso torna Eval () ainda mais parecido com o operador para efetivamente transformar "avaliar" em uma palavra reservada.Você não tem permissão para substituir a função Eval () com um novo valor.E você não é

permitido declarar uma variável, função, parâmetro de função ou capturaBloquear o parâmetro com o nome "Eval".

4.13 Operadores diversos

JavaScript suporta vários outros operadores diversos, descrito nas seções a seguir.

4.13.1 O operador condicional (? :)

O operador condicional é o único operador ternário (três operandos) em JavaScript e às vezes é realmente chamado de operador ternário. Este operador às vezes está escrito ?:, embora não apareçaAssim no código. Porque este operador tem três operandos, oPrimeiro vai antes do ?, O segundo vai entre o?e o: eO terceiro vai depois do :. É usado assim:`x > 0 ? x : -x` // o valor absoluto de xOs operandos do operador condicional podem ser de qualquer tipo. O primeiroOperando é avaliado e interpretado como um booleano. Se o valor doO primeiro operando é verdadeiro, então o segundo operando é avaliado e seu valor é retornado. Caso contrário, se o primeiro operando for falsamente, então o terceiroOperando é avaliado e seu valor é retornado. Apenas um dos segundos e terceiros operandos são avaliados;nunca os dois. Embora você possa obter resultados semelhantes usando a instrução IF (§5.3.1), O operador?: O operador geralmente fornece um atalho útil. Aqui está um típicouso, que verifica para ter certeza de que uma variável é definida (e tem umvalor significativo e verdadeiro) e usa -o se for ou fornece um valor padrão se

não:cumprimentando = "hello" + (nome de usuário? nome de usuário: "lá");Isso é equivalente a, mas mais compacto do que o seguinte sedeclarção:saudação = "olá";if (nome de usuário) {saudação += nome de usuário;} outro {saudação += "lá";}4.13.2 Primeiro definido (??)O primeiro operador definido ??avalia o seu primeiro operando definido: seSeu operando esquerdo não é nulo e não é indefinido, ele retorna esse valor.Caso contrário, ele retorna o valor do operando correto.Como o && e ||operadores, ??é curto-circuito: ele apenas avalia seu segundo operando seO primeiro operando avalia como nulo ou indefinido.Se a expressãoa não tem efeitos colaterais, então a expressão A ??B é equivalente a:(a! == null && a! == indefinido)?A: b?é uma alternativa útil a ||(§4.10.2) Quando você deseja selecionar oPrimeiro operando definido em vez do primeiro operando de verdade.Embora ||é nominalmente um lógico ou operador, também é usado idiomaticamente paraSelezione o primeiro operando não-falsy com código como este:// Se MaxWidth for verdade, use isso.Caso contrário, procure umvalor em// O objeto de preferências.Se isso não for verdade, use um

constante codificada.Seja max = maxWidth || Preferências.MaxWidth || 500;O problema com esse uso idiomático é que zero, a corda vazia e false são todos valores falsamente que podem ser perfeitamente válidos em algumas circunstâncias.Neste exemplo de código, se maxWidth for zero, esse valor será ignorado.Mas se mudarmos o operador para ??, acabamos com uma expressão em que zero é um valor válido:// Se a maxWidth for definida, use isso.Caso contrário, procure um valor em// O objeto de preferências.Se isso não estiver definido, use um constante codificada.Deixe Max = maxWidth ?? Preferências.MaxWidth ?? 500;Aqui estão mais exemplos mostrando como ?? funciona quando o primeiro operando é falsamente.Se esse operando é falsamente, mas definido, então ?? retorna.É somente quando o primeiro operando é "nulo" (ou seja, nulo ou undefined) que este operador avalia e retorna o segundo operando:Let Options = {Timeout: 0, Title: "", Verbose: false, n:nulo };options.timeout ?? 1000 // => 0: conforme definido no objeto options.title ?? "Untitled" // => "": conforme definido no objeto options.verbose ?? verdadeiro // => false: conforme definido no objeto options.quiet ?? false // => false: a propriedade não é definida options.n ?? 10 // => 10: A propriedade é nulaObserve que o tempo limite, título e expressões detalhadas aqui teriam valores diferentes se usássemos || em vez de ??.

O ??O operador é semelhante ao && e ||operadores, mas não tem maior precedência ou menor precedência do que eles. Se você usar Em uma expressão com qualquer um desses operadores, você deve usar explícito Parênteses para especificar qual operação você deseja executar primeiro:(A ?? B) ||c // ??Primeiro, depois ||A ??(B || C) // ||Primeiro, então ??A ??b ||c // SyntaxError: Parênteses são necessários O ??O operador é definido pelo ES2020 e, no início de 2020, é recentemente Suportado pelas versões atuais ou beta de todos os principais navegadores. Esse O operador é formalmente chamado de operadora de "coalescente nulo", mas evite esse termo porque este operador seleciona um de seus operandos, mas não os coalesce? de nenhuma maneira que eu possa ver.

4.13.3 O operador TIPEOF typeof é um operador unário que é colocado antes de seu único operando, que pode ser de qualquer tipo. Seu valor é uma string que especifica o tipo do operando. A Tabela 4-3 especifica o valor do operador do tipo de Qualquer valor JavaScript.

Tabela 4-3. Valores retornados pelo operador TypeOfxTipo de x indefinido "nulo" objeto "verdadeiro ou falso" Booleano "qualquer número ou nan" número"

qualquer bigint"bigint"qualquer string"corda"qualquer símbolo"símbolo"qualquer função"função"qualquer objeto de não função"objeto"Você pode usar o operador TIPEOF em uma expressão como esta:// Se o valor for uma string, embrulhe -o nas citações, caso contrário, converter(tipoof valor === "string")?"" + valor + """:value.ToString
()Observe que o tipo de retorna "objeto" se o valor do operando for nulo.Se Você deseja distinguir os objetos nulos, você terá que explicitamente testar para esse valor de caso especial.Embora as funções JavaScript sejam um tipo de objeto, o tipo de O operador considera as funções que são suficientemente diferentes que eles têm seu próprio valor de retorno.Porque o tipo de avalia para "objeto" para todos os valores de objeto e matrizAlém das funções, é útil apenas distinguir objetos de outros,Tipos primitivos.Para distinguir uma classe de objeto de outro, você deve usar outras técnicas, como a instânciaoperador (ver §4.9.4), o atributo de classe (ver §14.4.3) ou oPropriedade do construtor (ver §9.2.2 e §14.3).4.13.4 O operador de exclusão

Delete é um operador unário que tenta excluir a propriedade do objeto ou elemento da matriz especificado como seu operando. Como a tarefa, Incremento e operadores de decréscimo, o exclusão é normalmente usado para o seu efeito colateral da exclusão da propriedade e não para o valor que ele retorna. Alguns Exemplos: Seja `o = {x: 1, y: 2};`// Comece com um objeto excluir `o.x;`// Exclua uma de suas propriedades "x" em o // => false: a propriedade não existe mais. Seja `a = [1,2,3];`// Comece com uma matriz exclua um `[2];`// exclua o último elemento do array de `a` // => false: o elemento da matriz 2 não existe mais. A.Length // => 3: Observe que o comprimento da matriz não muda, no entanto observe que uma propriedade excluída ou elemento da matriz não é apenas definido como o valor indefinido. Quando uma propriedade é excluída, a propriedade deixa de existir. Tentando ler uma propriedade inexistente retorna indefinida. Mas você pode testar a existência real de uma propriedade com o operador ([§4.9.3](#)). A exclusão de um elemento de matriz deixa um "buraco" na matriz, não muda o comprimento da matriz. A matriz resultante é escassa ([§7.3](#)). Delete espera que seu operando seja um LValue. Se não é um lvalue, o operador não toma ação e retorna verdadeiro. Caso contrário, exclui tentativas de excluir o LValue especificado. Excluir retorna verdadeiro se for excluído com sucesso o LValue especificado. Nem todas as propriedades podem ser excluídas, no entanto: propriedades não confundíveis ([§14.1](#)) são imunes.

De exclusão. No modo rigoroso, a exclusão levanta um `síntaxe se seu operando for um identificador não qualificado, como uma variável, função ou funçãoParâmetro`: ele só funciona quando o operando é um acesso à propriedadeexpressão (§4.4). O modo rigoroso também especifica que o `delete` levanta um `TypeError` se solicitado a excluir qualquer não confundível (isto é, não lável)propriedade. Fora do modo rigoroso, nenhuma exceção ocorre nesses casos, e excluir simplesmente retorna `false` para indicar que o operando poderia não ser excluído. Aqui estão alguns exemplos de usos do operador de exclusão: Seja `o = {x: 1, y: 2};` excluir `o.x;`// excluir uma das propriedades do objeto; retorna verdadeiro. `typeof o.x;`// A propriedade não existe; retorna "indefinido". `excluir o.x;`// excluir uma propriedade inexistente; retorna verdadeiro. `excluir 1;`// Isso não faz sentido, mas apenas retorna verdadeiro.// não pode excluir uma variável; Retorna falsa, ou `syntaxerror` em modo rigoroso. `excluir o;`// Propriedade não lável: retorna falsa, ou `TypeError` em modo rigoroso. `excluir object.prototype;` Veremos o operador de exclusão novamente no §6.4.4.13.5 O operador aguardadoaguardar foi introduzido no ES2017 como uma maneira de tornar assíncronoProgramação mais natural em JavaScript. Você precisará ler

Capítulo 13 para entender este operador. Resumidamente, no entanto, aguardeespera um objeto de promessa (representando uma computação assíncrona) como o seu único operando, e faz com que seu programa se comportasse como se fosse Esperando a conclusão da computação assíncrona (mas faz isso sem realmente bloquear, e isso não impede outros assíncronos operações de proceder ao mesmo tempo). O valor do aguardarOperador é o valor de atendimento do objeto Promise. Importante, aguarda é apenas legal dentro de funções que foram declaradas assíncrono com a palavra -chave assíncrona. Novamente, veja o capítulo 13 para completodetalhes.

4.13.6 O operador vazioVoid é um operador unário que aparece antes de seu único operando, que pode ser de qualquer tipo. Este operador é incomum e com pouca frequência; isto Avalia seu operando, então descarta o valor e retorna indefinidos. Como o valor do operando é descartado, o uso do operador de vazios faz Senta apenas se o operando tiver efeitos colaterais. O operador vazio é tão obscuro que é difícil criar um Exemplo prático de seu uso. Um caso seria quando você quiser Defina uma função que não retorne nada, mas também usa a função de setaSintaxe de atalho (ver §8.1.3) onde o corpo da função é um único expressão que é avaliada e devolvida. Se você está avaliando a expressão apenas por seus efeitos colaterais e não deseja retornar seu valor, Então, o mais simples é usar aparelhos encaracolados ao redor do corpo da função. Mas, como alternativa, você também pode usar o operador de vazios neste caso: deixe o contador = 0;

const increment = () => void contador ++;incremento () // => indefinidocontador // => 14.13.7 O operador de vírgula (,)O operador de vírgula é um operador binário cujos operandos podem ser dequalquer tipo.Ele avalia seu operando esquerdo, avalia seu operando correto eEm seguida, retorna o valor do operando correto.Assim, a seguinte linha:i = 0, j = 1, k = 2;avalia para 2 e é basicamente equivalente a:i = 0;j = 1;k = 2;A expressão esquerda é sempre avaliada, mas seu valor é descartado,o que significa que só faz sentido usar o operador de vírgula quandoA expressão esquerda tem efeitos colaterais.A única situação em que oO operador de vírgula é comumente usado é com um loop (§5.4.3) que possuiVariáveis ??de loop múltiplas:// O primeiro vírgula abaixo faz parte da sintaxe do Letdeclaração// O segundo vírgula é o operador de vírgula: nos permite espremer2// expressões (i ++ e j--) em uma declaração (o loop for)que espera 1 para (vamos i = 0, j = 10; i <j; i ++, j--) {console.log (i+j);}4.14 Resumo

Este capítulo abrange uma grande variedade de tópicos, e há muitos material de referência aqui que você pode querer reler no futuro como você Continue a aprender JavaScript. Alguns pontos-chave a se lembrar, no entanto, são estes: Expressões são as frases de um programa JavaScript. Qualquer expressão pode ser avaliada em um valor JavaScript. Expressões também podem ter efeitos colaterais (como variável atribuição) além de produzir um valor. Expressões simples, como literais, referências variáveis ?? e os acessos à propriedade podem ser combinados com os operadores para produzir expressões maiores. JavaScript define operadores para a aritmética, comparações, lógica booleana, atribuição e manipulação de bits, juntamente com alguns operadores diversos, incluindo o ternário operador condicional. O operador JavaScript + é usado para adicionar números e cordas concatenadas. Os operadores lógicos && e || ter especial comportamento curto-circuito quando ?? e às vezes apenas avalia apenas um de seus argumentos. Idioms de javascript comum exigem que você Entenda o comportamento especial desses operadores.

Capítulo 5. Declarações

O capítulo 4 descreveu as expressões como frases de JavaScript. Por isso Analogia, declarações são frases ou comandos JavaScript. Assim como As frases em inglês são encerradas e separadas uma da outra com Períodos, as declarações de JavaScript são encerradas com semicolons (§2.6). Expressões são avaliadas para produzir um valor, mas as declarações são executadas para fazer algo acontecer. Uma maneira de "fazer algo acontecer" é avaliar uma expressão que tem efeitos colaterais. Expressões com efeitos colaterais, como atribuições e invocações da função, podem permanecer sozinhas como declarações e quando usadas O caminho são conhecidos como declarações de expressão. Uma categoria semelhante de declarações são as declarações de declaração que declaram novas variáveis e definir novas funções. Os programas JavaScript nada mais são do que uma sequência de declarações para executar. Por padrão, o intérprete JavaScript executa estes Declarações uma após a outra na ordem em que estão escritas. Outra maneira "Fazer algo acontecer" é alterar essa ordem padrão de execução, e JavaScript tem várias declarações ou estruturas de controle que fazem Apenas isso: Condicionais Declarações como se e switch que fazem o javascript intérprete executar ou pular outras declarações, dependendo do valor

de uma expressãoLoopsDeclarações como enquanto e para esse executar outras declaraçõeses repetidamenteSaltosDeclarações como quebrar, retornar e jogar que causam o intérprete para pular para outra parte do programaAs seções a seguir descrevem as várias declarações em JavaScripte explique sua sintaxe.Tabela 5-1, no final do capítulo, resume a sintaxe.Um programa JavaScript é simplesmente uma sequência de declarações, separadas uma da outra com semicolons, então uma vez você estãofamiliarizados com as declarações de JavaScript, você pode começar a escrever programas JavaScript.5.1 declarações de expressãoOs tipos mais simples de declarações em JavaScript são expressões que têm efeitos colaterais.Esse tipo de afirmação foi mostrado no capítulo 4.As declarações de atribuição são uma categoria principal de expressão de declarações.Por exemplo:saudação = "hello" + nome;i *= 3;Os operadores de incremento e decréscimos, ++ e -, estão relacionados a declarações de atribuição.Estes têm o efeito colateral de mudar um valor variável, como se uma tarefa tivesse sido realizada:

contador ++; O operador de exclusão tem o importante efeito colateral de excluir umaPropriedade do objeto. Assim, quase sempre é usado como uma afirmação, mas do que parte de uma expressão maior: excluir o.x; As chamadas de função são outra importante categoria de declarações de expressão. Para exemplo: console.log (DebugMessage); displayspinner () // uma função hipotética para exibir um Spinner em um aplicativo da web. Essas chamadas de função são expressões, mas têm efeitos colaterais que afetar o ambiente do host ou o estado do programa, e eles são usados ?? aqui com o de declarações. Se uma função não tiver efeitos colaterais, não há sentido em chamá-lo, a menos que faça parte de uma expressão maior ou de uma Declaração de atribuição. Por exemplo, você não apenas calcula umCosseno e descarte o resultado: Math.cos (x); Mas você pode calcular o valor e atribuir -lo a uma variável para uso futuro: cx = math.cos (x); Observe que cada linha de código em cada um desses exemplos é encerrada com um semicolon.

5.2 declarações compostas e vazias Assim como o operador de vírgula (§4.13.7) combina múltiplas expressões em uma única expressão, um bloco de declaração combina múltiplos declarações em uma única instrução composta. Um bloco de declaração é simplesmente uma sequência de declarações fechadas dentro de aparelhos encaracolados. Por isso, as linhas a seguir atuam como uma única declaração e podem ser usadas em qualquer lugar Esse JavaScript espera uma única declaração: {x = math.pi; cx = math.cos (x); console.log ("cos (?) =" + cx);} Há algumas coisas a serem observadas sobre este bloco de declaração. Primeiro, sim não terminar com um semicolon. As declarações primitivas dentro do bloco termina em semicolons, mas o próprio bloco não. Segundo, as linhas dentro do bloco é recuado em relação aos aparelhos encaracolados que incluem elas. Isso é opcional, mas facilita a leitura do código e entender. Assim como as expressões geralmente contêm subexpressões, muitos JavaScript As declarações contêm substanciais. Formalmente, a sintaxe JavaScript geralmente permite uma única substituição. Por exemplo, a sintaxe do loop while inclui uma única declaração que serve como o corpo do loop. Usando a Bloco de declaração, você pode colocar qualquer número de declarações dentro disso Substituição permitida única. Uma declaração composta permite que você use várias declarações onde A sintaxe do JavaScript espera uma única declaração. A declaração vazia é

o oposto: permite que você não inclua declarações onde alguém está esperado. A declaração vazia é assim:; O intérprete JavaScript não toma medidas quando executa um vazio de declaração. A declaração vazia é ocasionalmente útil quando você quiser para criar um loop que tenha um corpo vazio. Considere o seguinte para o loop (para loops será coberto no §5.4.3):// initialize uma matriz apara (vamos i = 0; i <a.Length; a [i ++] = 0); Neste loop, todo o trabalho é feito pela expressão a [i ++] = 0 e nenhum corpo de loop é necessário. A sintaxe JavaScript requer uma declaração como um corpo de loop, no entanto, uma declaração vazia - apenas um semicolon nu - é usado. Observe que a inclusão acidental de um ponto de vírgula após o direito parênteses de um loop for, enquanto o loop, ou se a declaração pode causar bugs frustrantes que são difíceis de detectar. Por exemplo, o seguinte código provavelmente não faz o que o autor pretendia: if ((a === 0) || (b === 0)); // opa! Esta linha faz nada... o = nulo; // e esta linha é sempre executado. Quando você usa intencionalmente a declaração vazia, é uma boa ideia comentar seu código de uma maneira que deixa claro que você está fazendo isso propósito. Por exemplo:

para (vamos i = 0; i <a.Length; a [i ++] = 0) / * vazio * /;5.3 CondicionaisDeclarações condicionais executam ou pularem outras declarações, dependendo deo valor de uma expressão especificada.Essas declarações são a decisãoPontos do seu código, e eles também são conhecidos como "ramos".Se você imagina um intérprete de javascript seguindo um caminho através do seuCódigo, as declarações condicionais são os lugares onde o código é ramificaem dois ou mais caminhos e o intérprete deve escolher qual caminhoseguir.As subseções a seguir explicam o condicional básico de JavaScript, olf/else declaração, e também o switch de capa, mais complicado,Declaração de Multiway Branch.5.3.1 seA declaração IF é a declaração de controle fundamental que permiteJavaScript para tomar decisões, ou, mais precisamente, para executar declaraçõescondicionalmente.Esta afirmação tem duas formas.O primeiro é:se (expressão)declaraçãoNesta forma, a expressão é avaliada.Se o valor resultante for verdade,a declaração é executada.Se a expressão for falsamente, a instrução não será executada.(Veja §3.4 para obter uma definição de valores verdadeiros e falsamente.) Por exemplo:se (nome de usuário == null) // se o nome de usuário for nulo ouindefinido,nome de usuário = "John Doe";// Defina

Ou da mesma forma:// Se o nome de usuário for nulo, indefinido, falso, 0, "" ou nan, dê um novo valorif (! Nome de usuário) nome de usuário = "John Doe";Observe que os parênteses ao redor da expressão são uma parte necessária da sintaxe para a instrução IF.A sintaxe JavaScript requer uma única declaração após a palavra -chave IF e expressão entre parênteses, mas você pode usar um bloco de declaração paraCombine várias declarações em uma.Portanto, a declaração se também podeParece isso:if (! endereço) {endereço = "";mensagem = "Especifique um endereço de correspondência.";}A segunda forma da declaração IF apresenta uma cláusula que é executada quando a expressão é falsa.Sua sintaxe é:se (expressão)Declaração1outroDeclaração2Esta forma da declaração executa a instrução1 se a expressão for Verdade e executa a declaração2 se a expressão for falsamente.Por exemplo:if (n === 1)console.log ("Você tem 1 nova mensagem.");outro

console.log (`você tem \$ {n} novas mensagens. Quando você tiver aninhado se as declarações com outras cláusulas, alguns cautelaé necessário para garantir que a cláusula elseidedeclaração.Considere as seguintes linhas:i = j = 1;k = 2;if (i === J)if (j === k)console.log ("eu igual a k");outroconsole.log ("Eu não igual a J");// ERRADO!!Neste exemplo, a instrução IF interna forma a instrução únicapermitido pela sintaxe da instrução IF Exterior.Infelizmente, énão claro (exceto da dica dada pelo recuo) que se omais vai com.E neste exemplo, o recuo está errado,Porque um intérprete de javascript realmente interpreta o anteriorExemplo como:if (i === j) {if (j === k)console.log ("eu igual a k");outroconsole.log ("Eu não igual a J");// opa!}A regra em JavaScript (como na maioria das linguagens de programação) é que porPadrão Uma cláusula else faz parte da instrução IF mais próxima.FazerEste exemplo menos ambíguo e mais fácil de ler, entender, manter,E Debug, você deve usar aparelhos encaracolados:

if (i === j) {if (j === k) {console.log ("eu igual a k");} } else {// que diferença a localização de uma cinta encaracolada faz! console.log ("Eu não igual a J");} Muitos programadores têm o hábito de envolver os corpos de se e else declarações (bem como outras declarações compostas, como enquanto loops) dentro de aparelhos encaracolados, mesmo quando o corpo consiste em apenas uma única declaração. Fazer isso de forma consistente pode impedir o tipo de problema acabou de ser mostrado, e eu aconselho você a adotar essa prática. Nesta Livro impresso, eu presto manter o código de exemplo verticalmente compacto, e nem sempre sigo meus próprios conselhos sobre esse assunto.

5.3.2 else if A instrução if/else avalia uma expressão e executa um dos duas peças de código, dependendo do resultado. Mas e quando você precisa executar uma das muitas peças de código? Uma maneira de fazer isso é com uma instrução else if. senão se não for realmente um javascript declaração, mas simplesmente um idioma de programação frequentemente usado que resulta Quando repetidos as declarações se/else são usadas:

```
if (n === 1) { // Execute o bloco de código nº 1} else if (n === 2) { // Executar o bloco de código #2} else if (n === 3) { // Executar o bloco de código #3} outro // Se tudo mais falhar, execute o bloco #4
```

}Não há nada de especial nesse código.É apenas uma série de se declarações, onde cada um seguindo se faz parte da cláusula de eliminação do de declaração anterior.Usando o else se o idioma for preferível emais legível do que, escrevendo essas declaraçõesFormulário equivalente e totalmente aninhado:
if (n === 1) {
 // Execute o bloco
 // de código nº 1
}
outro {
 if (n === 2) {
 // Executar o bloco de código #2
 }
 outro {
 if (n === 3) {
 // Executar o bloco de código #3
 }
 outro {
 // Se tudo mais falhar, execute o bloco #4
 }
 }
}5.3.3 SwitchUma declaração IF causa uma filial no fluxo da execução de um programa,E você pode usar o outro se o idioma para executar uma ramificação de várias via.Esta não é a melhor solução, no entanto, quando todos os galhos dependemno valor da mesma expressão.Nesse caso, é desperdiçadoAvalie repetidamente essa expressão em múltiplas declarações IF.A declaração do Switch lida exatamente com essa situação.O interruptor

A palavra -chave é seguida por uma expressão entre parênteses e um bloco deCódigo em aparelhos encaracolados:switch (expressão) {declarações}No entanto, a sintaxe completa de uma declaração de interruptor é mais complexa do queesse.Vários locais no bloco de código são rotulados com o casoPalavra -chave seguida por uma expressão e um colono. Quando um interruptorexecuta, ele calcula o valor da expressão e depois procura umrótulo de caso cuja expressão avalia o mesmo valor (ondeA semelhança é determinada pelo operador ==).Se encontrar um, começaexecutando o bloco de código na instrução rotulada pelo caso.Se issonão encontra um caso com um valor correspondente, ele procura uma declaraçãoPadrão rotulado:.Se não houver padrão: etiqueta, o interruptorA declaração pula o bloco de código completamente.Switch é uma declaração confusa para explicar;sua operação se tornaMuito mais claro com um exemplo.A seguinte declaração do interruptor éequivalente às declarações repetidas if/else mostradas no anteriorseção:Switch (n) {Caso 1: // Comece aqui se n === 1// Executar o bloco de código #1.quebrar;// Pare aquiCaso 2: // Comece aqui se n === 2// Executar o bloco de código #2.quebrar;// Pare aquiCaso 3: // Comece aqui se n === 3// Executar o bloco de código #3.quebrar;// Pare aqui

padrão: // Se tudo mais falhar ...// Executar o bloco de código #4.quebrar;// Pare aqui}Observe a palavra -chave quebrada usada no final de cada caso neste código.A declaração de quebra, descrita mais adiante neste capítulo, causa o intérprete para pular até o fim (ou "quebrar") do interruptordeclaração e continue com a declaração que a segue.O casocláusulas em uma declaração de interruptor especificam apenas o ponto de partida docódigo desejado;Eles não especificam nenhum ponto final.Na ausência deDeclarações de quebra, uma declaração de interruptor começa a executar seu bloco decódigo na etiqueta do caso que corresponde ao valor de sua expressão econtinua executando declarações até chegar ao final do bloco.Sobreocasiões raras, é útil escrever um código como esse que "cai"de uma etiqueta de caso para a seguinte, mas 99% do tempo você deve serCuidado para encerrar todos os casos com uma declaração de quebra.(Ao usarMudar dentro de uma função, no entanto, você pode usar uma declaração de retornoem vez de uma declaração de quebra.Ambos servem para encerrar o interruptordeclaração e impedir a execução de cair para o próximocaso.)Aqui está um exemplo mais realista da instrução Switch;ele converteUm valor para uma string de uma maneira que depende do tipo de valor:função convert (x) {Switch (tipo de x) {caso "número": // converte o número para umInteiro hexadecimalretornar x.toString (16);case "string": // retorna a string fechadaem citações

return "" + x + "";} padrão: // converte qualquer outro tipo da maneira usual retornar string (x);} } Observe que nos dois exemplos anteriores, as palavras -chave do caso são seguidas de literais de número e cordas, respectivamente. É assim que a declaração de switch é mais frequentemente usada na prática, mas observe que o padrão ECMAScript permite que cada caso seja seguido por um arbitrário expressão. A instrução Switch primeiro avalia a expressão que segue para alternar a palavra -chave e depois avaliar as expressões de caso, no ordenamento em que eles aparecem, até encontrar um valor que corresponda. O caso correspondente é determinado usando o operador de identidade ===, não o == Operador de igualdade, então as expressões devem corresponder sem qualquer tipo conversão. Porque nem todas as expressões de caso são avaliadas cada vez que a declaração de switch é executada, você deve evitar o uso de casos expressões que contêm efeitos colaterais, como chamadas de função ou tarefas. O curso mais seguro é simplesmente limitar seu caso expressões a expressões constantes. Como explicado anteriormente, se nenhuma das expressões de caso corresponder ao Expressão do interruptor, a declaração do interruptor começa a executar seu corpo. Na declaração chamada padrão:. Se não houver padrão: rótulo, a declaração do Switch pula completamente seu corpo. Observe que no 1

Exemplos mostrados, o padrão: rótulo aparece no final doMudar o corpo, seguindo todos os rótulos da caixa.Este é um lógico elugar comum para isso, mas pode realmente aparecer em qualquer lugar dentro docorpo da declaração.5.4 LoopsPara entender as declarações condicionais, imaginamos o JavaScriptintérprete seguindo um caminho de ramificação através do seu código -fonte.Odeclarações em loop são aquelas que dobram esse caminhoRepita partes do seu código.JavaScript tem cinco declarações de loop:Enquanto, faça/enquanto, para, para/de (e sua variante/aguardar),e para/in.As seguintes subseções explicam cada uma por sua vez.UmO uso comum para loops é iterar sobre os elementos de uma matriz.§7.6discute esse tipo de loop em detalhes e abrange métodos de loops especiaisdefinido pela classe da matriz.5.4.1 enquantoAssim como a declaração IF é condicional básico de JavaScript, o tempo todoA declaração é o loop básico de JavaScript.Tem a seguinte sintaxe:enquanto (expressão)declaraçãoPara executar um tempo, a declaração, o intérprete avalia primeiroexpressão.Se o valor da expressão for falsamente, então o intérpreteignora a afirmação que serve como o corpo do loop e segue paraA próxima declaração no programa.Se, por outro lado, a expressãoé verdade, o intérprete executa a declaração e repete, saltando

de volta ao topo do loop e avaliando a expressão novamente. Outra maneira de dizer isso é que o intérprete executa a declaração repetidamente enquanto a expressão é verdadeira. Observe que você pode criar um loop infinito com a sintaxe enquanto (true). Geralmente, você não deseja que o JavaScript execute exatamente o mesmo operação repetidamente. Em quase todos os loops, um ou mais as variáveis ??mudam com cada iteração do loop. Desde as variáveis Mudança, as ações executadas executando a declaração podem diferir cada vez através do loop. Além disso, se a variável em mudança ou Variáveis ??estão envolvidas na expressão, o valor da expressão pode ser diferente a cada vez através do loop. Isso é importante; de outra forma, uma expressão que começa a verdade nunca mudaria, e o loop nunca terminaria! Aqui está um exemplo de um loop de tempo que imprime os números de 0 a 9: deixe count = 0; while (contagem <10) {console.log (contagem); contagem ++;} Como você pode ver, a contagem de variáveis ??começa em 0 e é incrementada a cada vez que o corpo do loop corre. Uma vez que o loop execute 10 vezes, a expressão se torna falsa (ou seja, a contagem de variáveis ??é não mais de 10), o fim da declaração termina e o intérprete pode passar para a próxima declaração no programa. Muitos loops têm um contador de contra-variável. Os nomes da variável i, j e k são comumente usados como contadores de loop, embora você deva usar nomes mais descriptivos se tornar seu código mais fácil de entender.

5.4.2 Do/whileO loop do DO/While é como um loop de tempo, exceto que o loopA expressão é testada na parte inferior do loop e não na parte superior.Essesignifica que o corpo do loop é sempre executado pelo menos uma vez.OSintaxe é:fazerdeclaraçãoenquanto (expressão);O loop DO/enquanto é menos comum do que o seu primo -Na prática, é um tanto incomum ter certeza de que você quer umLoop para executar pelo menos uma vez.Aqui está um exemplo de um loop de fazer/while:função printArray (a) {Seja Len = A.Lengen, i = 0;if (len === 0) {console.log ("Array vazio");} outro {fazer {console.log (a [i]);} while (++ i <len);}}Existem algumas diferenças sintáticas entre o DO/enquantoloop e o loop comum.Primeiro, o loop do DO requer ambosfaça palavras -chave (para marcar o início do loop) e enquantopalavra -chave (para marcar o final e introduzir a condição do loop).Além disso, oO loop deve sempre ser encerrado com um ponto de vírgula.O loop whileNão precisa de um semicolon se o corpo do loop estiver fechado em aparelhos encaracolados.

5.4.3 paraA declaração for fornece uma construção em loop que geralmente é maisconveniente do que a declaração do tempo.A declaração for simplificaLoops que seguem um padrão comum.A maioria dos loops tem um contadorVariável de algum tipo.Esta variável é inicializada antes do início do loope é testado antes de cada iteração do loop.Finalmente, o balcãoA variável é incrementada ou atualizada no final do loopcorpo, pouco antes da variável ser testada novamente.Nesse tipo de loop, oinicialização, teste e atualização são os três cruciaismanipulações de uma variável de loop.A declaração for codifica cada um deessas três manipulações como expressão e fazem aquelasExpressões Uma parte explícita da sintaxe do loop:para (inicializar; teste; incremento)declaraçao inicializar, teste e incremento são três expressões (separadas porsemicolons) que são responsáveis ??por inicializar, testar eincrementando a variável de loop.Colocando todos eles na primeira linha doO loop facilita o entendimento do que um loop está fazendo eevita erros como esquecer de inicializar ou incrementar o loopvariável.A maneira mais simples de explicar como funciona um loop é para mostrar oequivaleente enquanto loop:inicializar;while (teste) {declaraçaoincremento;}2

Em outras palavras, a expressão inicializa é avaliada uma vez, antes do loop começar. Para ser útil, essa expressão deve ter efeitos colaterais (geralmente uma tarefa). JavaScript também permite a inicialização para ser um declaração de declaração variável para que você possa declarar e inicializar um contador de loop ao mesmo tempo. A expressão de teste é avaliada antecipadamente e controla se o corpo do loop é executado. Se o teste avalia a um valor verdadeiro, a afirmação que é o corpo do loop é executado. Finalmente, a expressão de incremento é avaliada de novo. Isso deve ser uma expressão com efeitos colaterais para ser útil. Geralmente, é uma expressão de atribuição ou usa o `++` ou `-` operadores. Podemos imprimir os números de 0 a 9 com um loop como o seguinte. Contrastá-lo com o equivalente enquanto o loop mostrado na Seção anterior:

```
for (let count = 0; contagem < 10; count++) {console.log(contagem);}
```

Loops podem se tornar muito mais complexos do que este exemplo simples, declaro, e às vezes várias variáveis ??mudam com cada iteração do loop. Esta situação é o único lugar que o operador de vírgula está comumente usado em JavaScript; Ele fornece uma maneira de combinar múltiplos inicializações e incrementos em uma única expressão adequada para uso em um loop:

```
let i = 0, j = 10;
for (i = 0, j = 10; i < 10; i++, j--) {
  soma += i * j;
}
```

Em todos os nossos exemplos de loop até agora, a variável de loop tem sido numérica. Isso é bastante comum, mas não é necessário. O código a seguir usa umPara o loop percorrer uma estrutura de dados da lista vinculada e retornar o último objeto na lista (ou seja, o primeiro objeto que não tem um próximo propriedade):

```
Função Tail (O) {  
    // Retornar o cauda de lista vinculada opera (<; o.next; o = o.next) / * vazio *; // atravessar enquanto o.Next é verdadeiro  
    return o;  
}  
Observe que este código não possui expressão inicializada. Qualquer um dos três Expressões podem ser omitidas de um loop para o loop, mas os dois semicolon são necessários. Se você omitir a expressão do teste, o loop se repete para sempre, e para (;;) é ?? outra maneira de escrever um loop infinito, como enquanto (verdadeiro).
```

5.4.4 para/de ES6 define uma nova declaração de loop: for/of. Este novo tipo de loop usa a palavra-chave for, mas é um tipo de loop completamente diferente do que é regular para loop. (Também é completamente diferente do mais velho para/em loop que descreveremos no §5.4.5.) O loop for/of trabalha com objetos iteráveis. Vamos explicar exatamente o que significa para um objeto ser iterável no capítulo 12, mas para este Capítulo, basta saber que matrizes, cordas, conjuntos e mapas são iteráveis: eles representam uma sequência ou conjunto de elementos que você pode fazer iterar usando um loop for/of.

Aqui, por exemplo, é como podemos usar para/de para fazer um loop através de elementos de uma variedade de números e calcule sua soma: Deixe dados = [1, 2, 3, 4, 5, 6, 7, 8, 9], soma = 0; para (Let of Data of Data) {soma += elemento;} soma // => 45 Superficialmente, a sintaxe parece regularmente para o loop: o paraA palavra -chave é seguida por parênteses que contêm detalhes sobre o que oLoop deve fazer. Nesse caso, os parênteses contêm uma variável declaração (ou, para variáveis ??que já foram declaradas, simplesmente o nome da variável) seguido pela palavra -chave e uma expressão que avalia um objeto iterável, como a matriz de dados em Este caso. Como em todos os loops, o corpo de um loop segue o Parênteses, normalmente dentro de aparelhos encaracolados. No código acabado de ser mostrado, o corpo do loop é executado uma vez para cada elemento da matriz de dados. Antes de cada execução do corpo do loop, o próximo elemento da matriz é atribuída à variável elemento. Os elementos da matriz são iterados em ordem do primeiro ao último. Matrizes são iterados "ao vivo" - as mudanças feitas durante a iteração podem afetar o resultado da iteração. Se modificarmos o código anterior para adicionar os dados da linha.push (soma); dentro do corpo do loop, então nós criarmos um loop infinito porque a iteração nunca pode atingir o último elemento da matriz. Para/de com objetos

Os objetos não são (por padrão) iterable.Tentando usar para/de umObjeto regular lança um TypeError em tempo de execução:Seja o = {x: 1, y: 2, z: 3};para (deixe o elemento de O) {}// lança TypeError porque O não é iterávelconsole.log (elemento);Se você deseja iterar através das propriedades de um objeto, você pode usá-lo loop for/in (introduzido no §5.4.5), ou uso para/de com oMétodo object.Keys ():Seja o = {x: 1, y: 2, z: 3};Let Keys = "";para (deixe k de object.keys (o)) {chaves += k;}chaves // => "xyz"Isso funciona porque object.keys () retorna uma variedade de propriedadesNomes para um objeto, e as matrizes são iteráveis ??com/of.Nota tambémque essa iteração das chaves de um objeto não é viva como a matrizExemplo acima foi - mudanças para o objeto O feito no corpo do loopnão terá efeito na iteração.Se você não se importa com as chaves de um objeto, você também pode iterar através dos valores correspondentes comoesse:deixe soma = 0;para (deixe v de object.values ??(o)) {soma += v;}soma // => 6

E se você estiver interessado nas chaves e nos valores de um objetoPropriedades, você pode usar para/de object.entries () eatribuição de destruição:deixe pars = "";para (vamos [k, v] de object.entries (o)) {pares + = k + v;}pares // => "x1y2z3"Object.Entries () retorna uma variedade de matrizes, onde cada um interiorArray representa um par de chaves/valor para uma propriedade do objeto.Nós usamosDesctrutamento de destruição neste exemplo de código para descompactar os internosMatrizes em duas variáveis ??individuais.Para/de stringsAs cordas são iteráveis ??de caractere por caractere no ES6:deixe frequência = {};para (deixe a carta de "Mississippi") {if (frequência [letra]) {frequência [letra] ++;} outro {frequência [letra] = 1;}}frequência // => {m: 1, i: 4, s: 4, p: 2}Observe que as strings são iteradas pelo Unicode CodePoint, não pelo UTF-16personagem.A string ?eu ??Tem um comprimento de 5 (porque os doisCada um dos caracteres emoji exige dois caracteres UTF-16 para representar).MasSe você itera essa corda com para/de, o corpo do loop executará trêsvezes, uma vez para cada um dos três pontos de código "eu", "?" e ". ?



Para/de com set e mapaAs classes de conjunto e mapa do ES6 embutidas são iteráveis. Quando você itera umDefina com/de, o corpo do loop é executado uma vez para cada elemento do conjunto. Você pode usar código como este para imprimir as palavras únicas em uma sequência de texto: deixe texto = "na na na na na na batman!"; deixe wordset = new Set (text.split ("")); deixe exclusivo = []; para (let Word of Wordset) {exclusivo.push (palavra);} exclusivo // => ["Na", "Na", "Batman!"] Os mapas são um caso interessante porque o iterador para um objeto de mapa faz não iterar as chaves do mapa ou os valores do mapa, mas pares de chave/valor. CadaTempo através da iteração, o iterador retorna uma matriz cujo primeiroO elemento é uma chave e cujo segundo elemento é o valor correspondente. Dado um mapa m, você pode iterar e destruir seus pares de chave/valor assim: Seja M = novo mapa ([[1, "One"]]); para (let [chave, valor] de m) {chave // ??=> 1 valor // => "um"} Iteração assíncrona com para/aguardarO ES2018 apresenta um novo tipo de iterador, conhecido como um assíncronoiterador, e uma variante no loop for/dePara/aguarda o loop que funciona com iteradores assíncronos.

Você precisará ler os capítulos 12 e 13 para entender opara/aguarda loop, mas aqui está como ele fica no código://
leia pedaços de um fluxo de maneira assíncrona elmprime -osFunção assíncreada printStream (stream) {para
aguardar (Let Chunk of Stream) {console.log (pedaço);}5.4.5 para/inA for/in loop se parece muito com um loop
para/de uma palavra -chavealterado para.o de, a for/in loop trabalha com qualquer objeto após o in.para/de loop é
novo no ES6, mas para/in fez parte do JavaScriptDesde o começo (e é por isso que tem o mais naturalSintaxe de
som).O para/in declaração circula através dos nomes de propriedades de umobjeto especificado.A sintaxe se
parece com a seguinte:para (variável no objeto)declaraçãoA variável normalmente nomeia uma variável, mas pode
ser uma declaração variávelou qualquer coisa adequada como o lado esquerdo de uma expressão de
atribuição.Objeto é uma expressão que avalia para um objeto.Como sempre, declaraçãoé a declaração ou bloco
de declaração que serve como o corpo do loop.E você pode usar um loop para/em um loop assim:

para (deixe p in o) { // atribui nomes de propriedades de o a variável pconsole.log (O [P]); // Imprima o valor de cada propriedade}Para executar uma declaração para/in, o intérprete JavaScript primeiroAvalia a expressão do objeto.Se avaliar para nulo ouundefined, o intérprete pula o loop e passa para o próximodeclaração.O intérprete agora executa o corpo do loop uma vez paracada propriedade enumerável do objeto.Antes de cada iteração, no entanto,O intérprete avalia a expressão variável e atribui o nome da propriedade (um valor de string) a ela.Observe que a variável no loop for/in pode ser um arbitrárioexpressão, desde que avalie para algo adequado para o lado esquerdo de uma tarefa.Esta expressão é avaliada a cada vez através doLoop, o que significa que pode avaliar de maneira diferente a cada vez.Para exemplo, você pode usar código como o seguinte para copiar os nomes de todosPropriedades do objeto em uma matriz:Seja o = {x: 1, y: 2, z: 3};deixe a = [], i = 0;para (a [i ++] em O) / * vazio * /;Matrizes JavaScript são simplesmente um tipo de objeto especializado, e matrizÍndices são propriedades de objetos que podem ser enumerados com um para/inlaço.Por exemplo, seguindo o código anterior com esta linhaenumera os índices de matriz 0, 1 e 2:para (deixe eu em a) console.log (i);

Acho que uma fonte comum de bugs em meu próprio código é o acidental uso de para/in com matrizes quando eu pretendia usar para/de. Quando Trabalhando com matrizes, você quase sempre deseja usar para/de vezde for/in. O loop for/in não enumera todas as propriedades de um objeto. Não enumera propriedades cujos nomes são símbolos. E das propriedades cujos nomes são strings, ele apenas se arrasta sobre opropriedades enumeráveis ??(ver §14.1). Os vários métodos internosDefinido pelo JavaScript Core não é enumerável. Todos os objetos têm umMétodo ToString (), por exemplo, mas o for/in loop nãoEnumere esta propriedade ToString. Além de métodos internos,Muitas outras propriedades dos objetos embutidos não são adequados. TodosPropriedades e métodos definidos pelo seu código são enumeráveis, porpadrão.(Você pode torná-los não-enumeráveis ??usando técnicas explicado no §14.1.) Propriedades herdadas enumeráveis ??(ver §6.3.2) também são enumeradas poro loop for/in. Isso significa que se você usar para/em loops e tambémUse o código que define propriedades herdadas por todos os objetos, entãoSeu loop pode não se comportar da maneira que você espera. Por esse motivo, muitosOs programadores preferem usar um loop for/of com object.keys ()em vez de um loop for/in. Se o corpo de um for/in loop excluir uma propriedade que ainda não foienumerado, essa propriedade não será enumerada. Se o corpo doLoop define novas propriedades no objeto, essas propriedades podem ou podemnão ser enumerado. Consulte §6.6.1 para obter mais informações sobre o pedido em

que para/em enumera as propriedades de um objeto.5.5 saltosOutra categoria de declarações de JavaScript são as declarações de salto.Como oO nome indica, eles fazem com que o intérprete JavaScript salte para um novoLocalização no código -fonte.A declaração de quebra faz oINTERPRETER SULT até o final de um loop ou outra declaração.continuarfaz com que o intérprete pule o resto do corpo de um loop e pule de voltaaté o topo de um loop para iniciar uma nova iteração.JavaScript permiteddeclarações a serem nomeadas ou rotuladas, e quebrar e continuar podemIdentifique o loop de destino ou outro rótulo de instrução.A declaração de retorno faz o intérprete saltar de uma funçãoinvocação de volta ao código que o invocou e também fornece o valorpara a invocação.A declaração de arremesso é uma espécie de retorno provisóriode uma função de gerador.A declaração de arremesso levanta, ou joga, umexceção e foi projetada para trabalhar com a tentativa/captura/finalmenteDeclaração, que estabelece um bloco de código de manuseio de exceção.Esseé um tipo complicado de declaração de salto: quando uma exceção é lançada,O intérprete salta para o manipulador de exceção de anexo mais próximo, quepode estar na mesma função ou na pilha de chamadas em uma invocaçãofunção.Detalhes sobre cada uma dessas declarações de salto estão nas seções queseguir.5.5.1 Declarações rotuladas

Qualquer declaração pode ser rotulada precedendo -a com um identificador e umcolon:Identificador: declaraçãoAo rotular uma declaração, você dá um nome que você pode usar para se referirem outros lugares do seu programa.Você pode rotular qualquer declaração, embora elaé útil apenas para rotular declarações que têm corpos, como loops econdiccionais.Ao dar um nome a um loop, você pode usar quebra eContinue as declarações dentro do corpo do loop para sair do loop ou paraSalte diretamente para o topo do loop para iniciar a próxima iteração.quebrare continuar são as únicas declarações JavaScript que usam a declaraçaoetiquetas;Eles são abordados nas seguintes subseções.Aqui está umExemplo de um rotulado enquanto loop e uma declaração continuada queusa o rótulo.MAINLOOP: while (token! == null) { // Código omitido ...Continue Mainloop;// pule para a próxima iteração doNomeado loop// mais código omitido ...}O identificador que você usa para rotular uma declaração pode ser qualquer javascript legalIdentificador que não é uma palavra reservada.O espaço para nome para rótulos édiferente do espaço para nome para variáveis ??e funções, então você podeUse o mesmo identificador que um rótulo de declaração e como uma variável ou função nome.Os rótulos de declaração são definidos apenas dentro da declaração para a qualEles se aplicam (e dentro de suas sub -subestimações, é claro).Uma declaração podenão tem o mesmo rótulo que uma declaração que o contém, mas doisAs declarações podem ter o mesmo rótulo, desde que nenhum esteja aninhado

dentro do outro. As declarações rotuladas podem ser rotuladas. Efetivamente, isso significa que qualquer declaração pode ter vários rótulos.

5.5.2 Break

A declaração de quebra, usada sozinha, causa o loop mais interno de fechamento ou alternar a instrução para sair imediatamente. Sua sintaxe é simples: `quebrar;` Porque causa um loop ou interruptor para sair, esta forma do intervalo A declaração é legal apenas se aparecer dentro de uma dessas declarações. Você já viu exemplos da declaração de quebra dentro de um Declaração de interruptor. Em loops, normalmente é usado para sair prematuramente Quando, por qualquer motivo, não há mais necessidade de completar o laço. Quando um loop tem condições complexas de terminação, é frequentemente mais fácil de implementar algumas dessas condições com declarações de quebra em vez de tentar expressá -los todos em uma única expressão de loop. O A seguir, o código pesquisa os elementos de uma matriz por um valor específico. O loop termina da maneira normal quando atinge o fim do variedade; termina com uma declaração de quebra se encontrar o que é Procurando na matriz:

```
para (vamos i = 0; i < A.Length; i++) {if (a [i] === Target) quebra;}
```

JavaScript também permite que a palavra-chave quebrada seja seguida por um Rótulo da declaração (apenas o identificador, sem colón):

Break LabelName; Quando o intervalo é usado com um rótulo, ele salta para o final ou termina, A declaração de anexo que possui o rótulo especificado. É um erro de sintaxe para usar o intervalo nesta forma se não houver uma declaração de anexo com a etiqueta especificada. Com esta forma da declaração de quebra, o nome nomeado A declaração não precisa ser um loop ou interruptor: o intervalo pode "sair de" qualquer declaração de anexo. Esta afirmação pode até ser um bloco de declaração agrupado em aparelhos encaracolados com o único objetivo de nomear o bloco com um rótulo. Uma nova linha não é permitida entre a palavra-chave quebrada e o LabelName. Isso é resultado da inserção automática de JavaScript de omitido semicolons: se você colocar um terminador de linha entre o intervalo Palavra-chave e o rótulo a seguir, JavaScript pressupõe que você pretende Use a forma simples e não marcada da declaração e trata a linha Terminator como um semicolon. (Veja §2.6.) Você precisa da forma rotulada da declaração de quebra quando quiser sair de uma declaração que não é o loop fechado mais próximo ou um trocar. O código a seguir demonstra:

```
Let Matrix = getData ()// Obtenha uma variedade 2D de números de um lugar// Agora some todos os números na matriz. Deixe Sum = 0, Sucesso = Falso;// Comece com uma declaração rotulada de que podemos sair se Erros ocorremComputeSum: if (Matrix){para (vamos x = 0; x < matrix.length; x++) {deixe a linha = matriz [x];if (! linha) quebrar computeSum;
```

para (vamos y = 0; y < row.length; y++) {deixe célula = linha [y];if (isnan (célula)) quebra computesum;soma += célula;}sucesso = true;}// As declarações de quebra saltam aqui.Se chegarmos aqui com sucesso == false// Então havia algo errado com a matriz que estávamosdado// caso contrário, a soma contém a soma de todas as células domatriz.Finalmente, observe que uma declaração de quebra, com ou sem rótulo, não podeTransferir controle através dos limites da função.Você não pode rotular umdeclaração de definição de função, por exemplo, e depois use esse rótulodentro da função.5.5.3 ContinueA declaração continua é semelhante à declaração de quebra.Em vez dede sair de um loop, no entanto, continue reinicia um loop no próximoiteração.A sintaxe da declaração continua é tão simples quanto aDeclaração de quebra:continuar;A declaração continua também pode ser usada com um rótulo:Continue LabelName;A declaração continua, em suas formas rotuladas e não marcadas, pode

ser usado apenas dentro do corpo de um loop. Usá-lo em qualquer outro lugar causa um erro de sintaxe. Quando a declaração continua é executada, a iteração atual do loop fechado é encerrado e a próxima iteração começa. Isso significa coisas diferentes para diferentes tipos de loops: Em um loop de tempo, a expressão especificada no início do loop é testado novamente e, se for verdade, o corpo do loop é executado a partir do topo. Em um loop de fazer/enquanto, a execução pula para o fundo do loop, onde a condição do loop é testada novamente antes de reiniciar o loop no topo. Em um loop for, a expressão de incremento é avaliada e a expressão do teste é testada novamente para determinar se outra iteração deve ser feita. Em um para/de ou para/em loop, o loop começa com o próximo valor iterado ou o próximo nome da propriedade sendo atribuído a uma variável especificada. Observe a diferença de comportamento da declaração continua no enquanto e para loops: um loop de tempo retorna diretamente à sua condição, mas um loop for primeiro avalia sua expressão de incremento e depois retorna à sua condição. Anteriormente, consideramos o comportamento do loop for termos de um "equivalente" enquanto loop. Porque continua a declaração se comporta de maneira diferente para esses dois loops, no entanto, não é realmente possível simular perfeitamente um loop com um pouco de loopszinho.

O exemplo a seguir mostra uma declaração de continuação não marcada sendo usado para pular o restante da iteração atual de um loop quando um erro ocorre: para (vamos $i = 0$; $i < \text{data.length}$; $i++$) { se ($\text{dados}[i]$) continuar; // não pode prosseguir com indefinidamente } $\text{dados} += \text{dados}[i]$ } Como a declaração de quebra, a declaração continua pode ser usada em seu formulário rotulado dentro de loops aninhados quando o loop a ser reiniciado não é o loop imediatamente fechado. Além disso, como na declaração de quebra, quebras de linha não são permitidas entre a declaração de continuação e suaLabelName.

5.5.4 Retorno Lembre -se de que as invocações de função são expressões e que todos elas têm valores. Uma declaração de retorno dentro de uma função especifica o valor das invocações dessa função. Aqui está a sintaxe de uma declaração de devolução: expressão de retorno; Uma declaração de retorno pode aparecer apenas dentro do corpo de uma função. Isto é um erro de sintaxe para aparecer em qualquer outro lugar. Quando o retorno é executada, a função que contém retorna o valor de expressão para seu chamador. Por exemplo: função quadrado (x) { return $x * x$; } // uma função que tem uma declaração de retorno

quadrado (2) // => 4Sem declaração de retorno, uma invocação de funções simplesmente executacada uma das declarações no corpo da função, por sua vez, até chegar aoFim da função e depois retorna ao seu chamador.Nesse caso, oA expressão de invocação é avaliada como indefinida.O retornoA declaração geralmente aparece como a última declaração em uma função, mas precisanão ser o último: uma função retorna ao seu chamador quando uma declaração de retorno éexecutado, mesmo que haja outras declarações restantes na funçãoocorpo.A declaração de retorno também pode ser usada sem expressão paraFaça a função retornar indefinida ao seu chamador.Por exemplo:Função DisplayObject (O) { // retorna imediatamente se o argumento for nulo ouundefined.se (! O) retornar; // O restante da função vai aqui ...}Por causa da inserção automática de semicolon do JavaScript (§2.6), vocênão pode incluir uma quebra de linha entre a palavra -chave de retorno e oexpressão que a segue.5.5.5 RendimentoA declaração de rendimento é muito parecida com a declaração de retorno, mas é usadaSomente nas funções do gerador ES6 (consulte §12.3) para produzir o próximo valorNa sequência gerada de valores sem realmente retornar:// uma função de gerador que gera uma variedade de números inteiros

Função* intervalo (de, a) {para (vamos i = de; i <= para; i++) {rendimento i;}}Para entender o rendimento, você deve entender os iteradores egeradores, que não serão cobertos até o capítulo 12. O rendimento éIncluído aqui para completar, no entanto.(Tecnicamente, porém,O rendimento é um operador e não uma declaração, conforme explicado no §12.4.2.)5.5.6 JogueUma exceção é um sinal que indica que algum tipo de excepcionalcondição ou erro ocorreu.Dar uma exceção é sinalizar talum erro ou condição excepcional.Pegar uma exceção é lidar com isso-para tomar as ações necessárias ou apropriadas para se recuperar dea exceção.Em JavaScript, as exceções são jogadas sempre que um tempo de execuçãoo erro ocorre e sempre que o programa joga explicitamente um usando oDeclaração de arremesso.Exceções são capturadas com oTente/Catch/Finalmente declaração, descrita no próximoseção.A declaração de arremesso tem a seguinte sintaxe:lançar expressão;A expressão pode avaliar um valor de qualquer tipo.Você pode jogar umnúmero que representa um código de erro ou uma string que contém um humanoMensagem de erro legível.A classe de erro e suas subclasses são usadasQuando o próprio intérprete JavaScript lança um erro, e você pode usar

eles também.Um objeto de erro tem uma propriedade de nome que especifica o tipo de erro e uma propriedade de mensagem que mantém a string passada para a função do construtor.Aqui está uma função de exemplo que lança um objeto de erro quando invocado com um argumento inválido:função fatorial (x) { // Se o argumento de entrada for inválido, faça uma exceção!se (x <0) lançar um novo erro ("x não deve ser negativo");// caso contrário, calcule um valor e retorne normalmente!deixe f;for (f = 1; x > 1; f *= x, x--) / *vazio */;retornar f;}Fatorial (4) // => 24Quando uma exceção é lançada, o intérprete JavaScript imediatamente interrompe a execução normal do programa e salta para a exceção mais próxima manipulador.Os manipuladores de exceção são escritos usando a cláusula de captura do Tente/Catch/Finalmente declaração, descrita no próximo seção.Se o bloco de código em que a exceção foi lançada não tem uma cláusula de captura associada, o intérprete verifica o próximo bloco de código mais alto para ver se ele tem um manipulador de exceção associado a ele.Isso continua até que um manipulador seja encontrado.Se uma exceção é lançada em uma função que não contém um Tente/Catch/Finalmente declaração para lidar com isso, a exceção propaga -se para o código que chamou a função.Desta maneira, Exceções se propagam através da estrutura lexical do JavaScript Métodos e subir a pilha de chamadas.Se nenhum manipulador de exceção for encontrado,A exceção é tratada como um erro e é relatada ao usuário.5.5.7

Tente/Catch/Finalmente

A declaração de tentativa/captura/finalmente é a exceção de JavaScriptmecanismo de manuseio.A cláusula de tentativa desta afirmação simplesmente defineO bloco de código cujas exceções devem ser tratadas.O bloco de tentativaé seguido por uma cláusula de captura, que é um bloco de declarações que sãoinvocado quando ocorre uma exceção em qualquer lugar dentro do bloco de tentativa.A cláusula de captura é seguida por um bloco finalmente contendoCódigo de limpeza que é garantido para ser executado, independentemente do queacontece no bloco de tentativa.Tanto a captura quanto finalmente os blocos sãoopcional, mas um bloco de tentativa deve ser acompanhado por pelo menos um dessesblocos.A tentativa, a captura e finalmente os blocos começam e terminam comaparelho encaracolado.Esses aparelhos são uma parte necessária da sintaxe e não podemSeja omitido, mesmo que uma cláusula contenha apenas uma única declaração.O código a seguir ilustra a sintaxe e o objetivo doTente/Catch/Finalmente declaração:tentar {}// normalmente, esse código é executado do topo do bloco parao fundo// sem problemas.Mas às vezes pode jogar umexceção,// seja diretamente, com uma declaração de arremesso, ouIndiretamente, ligando// Um ??método que lança uma exceção.}Catch (e) {}// As declarações neste bloco são executadas se, e somenteSe, a tentativa// Block lança uma exceção.Essas declarações podem usara variável local// e para se referir ao objeto de erro ou outro valor que foijogado// Este bloco pode lidar com a exceção de alguma forma, podeignore o// exceção fazendo nada, ou pode repensar oexceção com arremesso.

}finalmente {// Este bloco contém declarações que são sempre executadas, independentemente de// O que acontece no bloco de tentativa. Eles são executados se a tentativa// termina o bloco:// 1) normalmente, depois de atingir o fundo do bloco// 2) por causa de um intervalo, continue ou devolva a declaração// 3) com uma exceção que é tratada por uma cláusula acima// 4) com uma exceção não capturada que ainda é propagada}Observe que a palavra -chave de captura geralmente é seguida por um identificador em parênteses. Este identificador é como um parâmetro de função. Quando uma exceção é capturada, o valor associado à exceção (um erro ou objeto, por exemplo) é atribuído a este parâmetro. O identificador associado a uma cláusula de captura tem escopo de bloco - ela é definida apenas dentro do bloco de captura. Aqui está um exemplo realista da declaração de tentativa/captura. Ele usa o método Fatorial () definido na seção anterior e no cliente Métodos laterais de JavaScript Prompt () e alert () para entrada saída:tentar {// Peça ao usuário para inserir um númeroSeja n = número (prompt ("insira um número inteiro positivo", ""));// calcular o fatorial do número, assumindo a entrada é válidaSeja f = fatorial (n);// exibe o resultadoalerta (n + "!" + f);}

Catch (ex) {
// Se a entrada do usuário não foi válida, terminamos aqui em cima
alerta (ex);
// Diga ao usuário qual é o erro}
Este exemplo é uma declaração de tentativa/captura sem cláusula finalmente. Embora finalmente não seja usado com a mesma frequência que pode ser útil. No entanto, seu comportamento requer explicação adicional. O finalmente é garantido que a cláusula seja executada se alguma parte do bloco de tentativa forexecutado, independentemente de como o código no bloco de tentativa é concluído. Isso é geralmente usado para limpar após o código na cláusula de tentativa. No caso normal, o intérprete JavaScript chega ao final dotente blocos e depois prossegue para o bloco finalmente, que executa qualquer limpeza necessária. Se o intérprete deixou o bloco de tentativa por causa de uma declaração de retorno, continuação ou interrupção, o bloco finalmente é Executado antes que o intérprete salte para seu novo destino. Se ocorrer uma exceção no bloco de tentativa e há um associado Catch bloqueio para lidar com a exceção, o intérprete executa primeiro o Pegue o bloco e depois o bloco finalmente. Se não houver captura local bloco para lidar com a exceção, o intérprete executa primeiro o Finalmente bloqueie e depois pula para a captura mais próxima que contém cláusula. Se um bloco finalmente causar um salto com um retorno, continue, quebrar, ou fazer uma declaração, ou chamar um método que lança uma exceção, o intérprete abandona qualquer salto pendente e Executa o novo salto. Por exemplo, se uma cláusula finalmente lança um

exceção, essa exceção substitui qualquer exceção que estivesse no processode ser jogado.Se uma cláusula finalmente emitir uma declaração de devolução, oo método retorna normalmente, mesmo que uma exceção tenha sido lançada e tenha ainda não foi tratado.Tente e finalmente pode ser usado juntos sem uma cláusula de captura.Em Este caso, o bloco finalmente é simplesmente o código de limpeza que é garantido para ser executado, independentemente do que acontecer na tentativa bloquear.Lembre -se de que não podemos simular completamente um loop para um com um enquanto loop porque a declaração continua se comporta de maneira diferente para os dois loops.Se adicionarmos uma declaração de tentativa/finalmente, podemos escrever um enquanto loop que funciona como um loop e que as alças continuam declarando corretamente:// simular para (inicializar; teste; incremento) corpo; inicializar; while (teste) {tente {body;} finalmente {incremento;}} Observe, no entanto, que um corpo que contém uma declaração de quebra se comporta um pouco diferente (causando um incremento extra antes de sair) no enquanto o loop do que faz no loop for, mesmo com o finalmente cláusula, não é possível simular completamente o loop for com enquanto.Cláusulas de captura nuaOcasionalmente, você pode se encontrar usando uma cláusula de captura apenas para detectar e parar a propagação de Uma exceção, mesmo que você não se importe com o tipo ou o valor da exceção.Em ES2019E mais tarde, você pode omitir os parênteses e o identificador e usar a palavra -chave de captura nua nestecaso.Aqui está um exemplo:

// como json.parse (), mas retorne indefinido em vez de jogar um errofunção parsejson (s) {tentar {retornar json.parse (s);} pegar {// algo deu errado, mas não nos importamos com o que foiretornar indefinido;}}5.6 Declarações diversasEsta seção descreve as três declarações restantes de JavaScript- com, depurador e "use rigoroso".5.6.1 comA declaração com coma um bloco de código como se as propriedades de umObjeto especificado foi variável no escopo para esse código.Tem oApós a sintaxe:com (objeto)declaraçãoEsta afirmação cria um escopo temporário com as propriedades do objetocomo variáveis ??e, em seguida, executa a declaração dentro desse escopo.A declaração com comser considerado depreciado no modo não rigoroso: evite usá-lo sempre quepossível.O código JavaScript que usa é difícil de otimizar e éProvavelmente correr significativamente mais lentamente do que o código equivalente escritosem a declaração com.

O uso comum do com a declaração é facilitar o trabalho com hierarquias de objetos profundamente aninhados. Em JavaScript do lado do cliente, paraPor exemplo, você pode ter que digitar expressões como esta para acessar Elementos de um formulário HTML:`document.forms [0] .address.value`Se você precisar escrever expressões como essa várias vezes, você podeUse a declaração com com tratar as propriedades do objeto de forma como variáveis:`com (document.forms [0]) { // Acesso a elementos do formulário diretamente aqui. Por exemplo: name.value = ""; endereço.value = ""; email.value = "";`Isso reduz a quantidade de digitação que você precisa fazer: você não precisa maisPara prefixar cada nome da propriedade do formulário com `document.forms [0]`. Isso éTão simples, é claro, para evitar a declaração com e escrever oCódigo anterior como este:Seja `f = document.forms [0]; f.name.value = ""; f.address.value = ""; f.email.value = "";`Observe que se você usar const ou let ou var para declarar uma variável ouconstante dentro do corpo de uma declaração, cria um comumvariável e não define uma nova propriedade dentro do objeto especificado.

5.6.2 DepuradorA declaração do depurador normalmente não faz nada.Se, no entanto, aO programa depurador está disponível e está em execução, depois uma implementaçãopode (mas não é necessário) executar algum tipo de ação de depuração.EmPrática, esta afirmação age como um ponto de interrupção: execução de javascriptO código para e você pode usar o depurador para imprimir valores das variáveis,Examine a pilha de chamadas e assim por diante.Suponha, por exemplo, que você sejaobtendo uma exceção em sua função f () porque está sendo chamadacom um argumento indefinido, e você não consegue descobrir onde está essa chamadavindo de.Para ajudá -lo a depurar esse problema, você pode alterarf () para que comece assim:função f (o) {if (o === indefinido) depurador;// linha temporária parafins de depuração... // o resto da funçãovai aqui.)Agora, quando f () é chamado sem argumento, a execução vai parar eVocê pode usar o depurador para inspecionar a pilha de chamadas e descobrir ondeEsta chamada incorreta está vindo.Observe que não basta ter um depurador disponível: o depuradorA declaração não começará o depurador para você.Se você está usando uma webnavegador e ter o console de ferramentas de desenvolvedor aberto, no entanto, esteA declaração causará um ponto de interrupção.5.6.3 ?Use rigoroso?

"Use Strict" é uma diretiva introduzida no ES5. Diretivas não são declarações (mas estão próximas o suficiente para que "Uso Strict" esteja documentado aqui). Existem duas diferenças importantes entre o "uso" da diretiva rígida "Diretiva" e Declarações regulares: Não inclui nenhuma palavra-chave idioma: a diretiva é apenas uma declaração de expressão que consiste em uma string especial literal (em citações únicas ou duplas). Só pode aparecer no início de um script ou no início de um corpo da função, antes que quaisquer declarações reais aparecessem. O objetivo de uma diretiva "uso rigoroso" é indicar que o código deve seguir (no script ou função) é um código rigoroso. O nível superior (não função) Código de um script é um código rigoroso se o script tiver um "Use" da diretiva rigorosa "um corpo de função é um código rigoroso se for definido dentro do código rigoroso ou se possui uma diretiva "usar rigorosa". Código passado para o método avaliar () é um código rigoroso se avaliar () for chamado de rigoroso. Código ou se a sequência de código incluir uma diretiva "Use Strict". Em Além do código declarado explicitamente como rigoroso, qualquer código em uma classe ou corpo (capítulo 9) ou em um módulo ES6 (§10.3) é automaticamente rigoroso. Isso significa que se todo o seu código JavaScript for escrito com módulos, então tudo é automaticamente rigoroso, e você nunca precisará usar uma diretiva explícita "use rigorosa". O código rigoroso é executado no modo rigoroso. Modo rigoroso é um subconjunto restrito do idioma que corrige deficiências importantes da linguagem e fornece Verificação de erro mais forte e maior segurança. Porque o modo rigoroso é Não é o código JavaScript antigo e padrão que ainda usa o legado deficiente. Os recursos do idioma continuarão a funcionar corretamente. As diferenças

Entre o modo rigoroso e o modo não rigoroso, são os seguintes (o primeiro três são particularmente importantes): A declaração com com o modo não é permitido no modo rigoroso. No modo rigoroso, todas as variáveis ??devem ser declaradas: um ReferenceError é jogado se você atribuir um valor a um identificador essa não é uma variável declarada, função, parâmetro de função, Parâmetro da cláusula de captura, ou propriedade do objeto global. (Em modo não rigoroso, isso declara implicitamente uma variável global por adicionar uma nova propriedade ao objeto global.) No modo rigoroso, as funções invocadas como funções (e não como métodos) têm um valor desse valor indefinido. (Em não rigoroso modo, as funções invocadas como funções são sempre passadas o objeto global como esse valor.) Além disso, no modo estrito, quando uma função é invocada com Call () ou Apply () (§8.7.4), o valor é exatamente o valor passado como o primeiro argumento para ligar () ou aplique (). (No modo não estrito, nulos e os valores indefinidos são substituídos pelo objeto global e os valores não -objeto são convertidos em objetos.) No modo rigoroso, atribuições para propriedades não escritas e tentativas de criar novas propriedades em objetos não extensíveis jogam um TypeError. (No modo não rito, essas tentativas falham silenciosamente.) No modo rigoroso, o código passado para avaliar () não pode declarar variáveis ??ou definir funções no escopo do chamador como pode em modo não rigoroso. Em vez disso, as definições de variáveis ??e funções vivem em um novo escopo criado para o Eval (). Este escopo é descartado quando o Eval () retorna. No modo rigoroso, o objeto de argumentos (§8.3.3) é uma função mantém uma cópia estática dos valores passados ??para a função. Em não modo rigoroso, o objeto de argumentos tem comportamento "mágico" em

quais elementos da matriz e parâmetros de função nomeadosAmbos se referem ao mesmo valor.No modo rigoroso, um SyntaxError é jogado se a exclusãoO operador é seguido por um identificador não qualificado, como umParâmetro variável, função ou função.(No modo nãoEssa expressão de exclusão não faz nada e avalia parafalse.)No modo rigoroso, uma tentativa de excluir uma propriedade não configuráveljoga um TypeError.(No modo não rito, a tentativa falha eA expressão de exclusão avalia como falsa.)No modo rigoroso, é um erro de sintaxe para um objeto literal definir duas ou mais propriedades com o mesmo nome.(No modo não estrito, nenhum erro ocorre.)No modo rigoroso, é um erro de sintaxe para uma declaração de função paraTenha dois ou mais parâmetros com o mesmo nome.(Em nãomodo rigoroso, nenhum erro ocorre.)No modo rigoroso, literais inteiros octais (começando com um 0 que é não seguido por um x) não é permitido.(No modo não estrito, Algumas implementações permitem literais octais.)No modo rigoroso, os identificadores avaliam e os argumentos são tratados como palavras-chave, e você não tem permissão para alterar seu valor.Você não pode atribuir um valor a esses identificadores, declarar como variáveis, use -os como nomes de funções, use -os como nomes de parâmetros de função ou usá -los como identificador de um Catch bloco.No modo rigoroso, a capacidade de examinar a pilha de chamadas é restrito.argumentos.caller e argumentos.função de modo.Funções de modo rigoroso também têm chamador ePropriedades de argumentos que jogam TypeError quando lidas.

(Algumas implementações definem essas propriedades fora do padrão em funções não rigíveis.)

5.7 declarações

As palavras-chave const, let, var, function, classe, importar e exportação não são tecnicamente declarações, mas elas se parecem muito com declarações, e este livro refere-se informalmente a eles como declarações, então Eles merecem uma menção neste capítulo.

Essas palavras-chave são descritas com mais precisão como declarações do que declarações. Dissemos no início deste capítulo que declarações "Faça algo acontecer." As declarações servem para definir novos valores e dê a eles nomes que podemos usar para referir a esses valores. Eles Não fazem muito acontecer, mas fornecendo nomes para valores, eles, em um sentido importante, definem o significado das outras declarações em seu programa. Quando um programa é executado, são as expressões do programa que estão sendo avaliados e as declarações do programa que estão sendo executadas. Declarações em um programa não "correm" da mesma maneira: em vez disso, elas definem a estrutura do próprio programa. Vagamente, você pode pensar em declarações como as partes do programa que são processadas antes do código começar a ser executado. As declarações de JavaScript são usadas para definir constantes, variáveis, funções e classes e para importar e exportar valores entre módulos. As próximas subseções dão exemplos de todos esses tipos de declarações. Todos estão cobertos com muito mais detalhes em outros lugares em

este livro.5.7.1 const, let e varAs declarações Const, Let e ??VAR são cobertas em §3.10.Em ES6e mais tarde, Const declara constantes e deixe declara variáveis.AnteriorPara ES6, a palavra -chave VAR era a única maneira de declarar variáveis ??eNão havia como declarar constantes.Variáveis ??declaradas com var sãoescopo para a função contendo, em vez do bloco contendo.Esse pode ser uma fonte de bugs e, no javascript moderno, não há realmente nãoRazão para usar o VAR em vez de Let.const tau = 2*math.pi;Deixe o raio = 3;var circunferência = tau * raio;5.7.2 FunçãoA declaração de função é usada para definir funções, que sãocoberto em detalhes no capítulo 8. (Também vimos a função no §4.3,onde foi usado como parte de uma expressão de função em vez de umdeclaração de função.) Uma declaração de função se parece com a seguinte:Área de função (raio) {retornar math.pi * raio * raio;}Uma declaração de função cria um objeto de função e o atribui aoNome especificado - Área neste exemplo.Em outros lugares do nosso programa, nós pode se referir à função - e executar o código dentro dela - usando issonome.As declarações de função em qualquer bloco de código JavaScript são

processado antes que esse código seja executado, e os nomes de funções estão ligados aa função se observa em todo o bloco.Dizemos essa funçãoAs declarações são "içadas" porque é como se todas tivessem sido movidasaté o topo de qualquer escopo em que sejam definidos.O resultado é que código que chama uma função pode existir em seu programa antes do códigoque declara a função.§12.3 descreve um tipo especial de função conhecido como gerador.As declarações de gerador usam a palavra -chave da função, mas siga -a comum asterisco.§13.3 descreve funções assíncronas, que também sãodeclarado usando a palavra -chave da função, mas é prefixado com opalavra -chave assíncrona.5.7.3 ClasseNo ES6 e mais tarde, a declaração de classe cria uma nova classe e dáÉ um nome que podemos usar para referir a ele.As aulas são descritas em detalhes emCapítulo 9. Uma declaração simples de classe pode ser assim:

```
classe Circle {construtor (raio) {this.r = raio;}área () {return math.pi * this.r * this.r; }circunferênci () {return 2 * math.pi * this.r;}}
```

Ao contrário das funções, as declarações de classe não são içadas e você não pode usarUma classe declarada dessa maneira no código que aparece antes da declaração.5.7.4 Importação e exportaçãoAs declarações de importação e exportação são usadas juntas para fazer

valores definidos em um módulo de código JavaScript disponível em outromódulo.Um módulo é um arquivo de código JavaScript com seu próprio globalnamespace, completamente independente de todos os outros módulos.A únicacomo um valor (como função ou classe) definido em um módulo podeser usado em outro módulo é se o módulo definidor o exportarexportar e o uso do módulo importa com importação.Módulos sãoo assunto do capítulo 10 e a importação e exportação são cobertas emDetalhes em §10.3.Diretivas de importação são usadas para importar um ou mais valores de outroArquivo do código JavaScript e dê -lhes nomes no módulo atual.As diretrizes de importação vêm em algumas formas diferentes.Aqui estão algunsExemplos:círculo de importação de './geometry/circle.js';importar {pi, tau} de './geometry/constants.js';importar {magnitude como hipotenuse} de './vectors/utils.js';Os valores dentro de um módulo JavaScript são privados e não podem ser importadosem outros módulos, a menos que tenham sido exportados explicitamente.OA Diretiva de Exportação faz isso: declara que um ou mais valores definidosno módulo atual são exportados e, portanto, disponíveis para importaçãopor outros módulos.A diretiva de exportação tem mais variantes do que oA Diretiva de Importação faz.Aqui está um deles:// geometria/constants.jsconst pi = math.pi;const tau = 2 * pi;exportação {pi, tau};A palavra -chave de exportação às vezes é usada como modificador em outros

declarações, resultando em um tipo de declaração composta que define umconstante, variável, função ou classe e a exporta ao mesmo tempo.E quando um módulo exporta apenas um valor, isso geralmente é feitoCom o padrão de exportação de formulário especial:exportar const tau = 2 * math.pi;magnitude da função de exportação (x, y) {return math.sqrt (x*x + y*y);}exportar círculo de classe padrão { /* definição de classe omitidaaqui */ }5.8 Resumo das declarações JavaScriptEste capítulo introduziu cada uma das declarações da linguagem JavaScript,que estão resumidos na Tabela 5-1.Tabela 5-1.Declaração JavaScript SintaxeDeclaraçãoPropósitoquebrarSaia do loop ou interruptor mais íntimo ou do nome nomeadodeclaraçãocasoRotule uma declaração dentro de um interruptoraulaDeclarar uma aulaconstDeclarar e inicializar uma ou mais constantescontinuarComece a próxima iteração do loop mais interno ou o loop nomeadoDepuradorPonto de interrupção do depuradorpadrãoRotule a instrução padrão em um interruptorfaça/whileUma alternativa ao loop while

exportarDeclarar valores que podem ser importados para outros módulosparaUm loop fácil de usarpara/aguardarIteram de forma assíncrona os valores de um iterador assíncronopara/inEnumerar os nomes de propriedades de um objetopara/deEnumerar os valores de um objeto iterável, como uma matrizfunçãoDeclarar uma funçãoose/elseExecutar uma declaração ou outra dependendo de uma condiçãooimportarDeclarar nomes para valores definidos em outros módulosrótuloDê um nome para uso com quebra e continuaudeixarDeclare e inicialize uma ou mais variáveis ??escassas de blocos (Novosintaxe)retornarRetornar um valor de uma funçãootrocarnominal Multiway para Case ou Padrão: RótuloslançarJogue uma exceçãotente/capturar/finallyLidar com exceções e limpeza de código?Use rigoroso?Aplique restrições de modo rigoroso para script ou funçãovarDeclare e inicialize uma ou mais variáveis ??(sintaxe antiga)enquantoUma construção de loop básicocomEstender a cadeia de escopo (depreciada e proibida no modo rigoroso)colheitaFornecer um valor a ser iterado;usado apenas nas funções do gerador1O fato de as expressões de caso serem avaliadas em tempo de execução faz do JavaScriptDeclaração de interruptor muito diferente (e menos eficiente do que) a declaração de interruptor

de C, C ++ e Java. Nesses idiomas, as expressões de caso devem ser com tempo de compilação Constantes do mesmo tipo, e as declarações de switch geralmente podem se compilar até Tabelas de salto eficientes.² Quando considerarmos a declaração de continuação no §5.5.3, veremos que isso enquanto o loop é Não é um equivalente exato do loop for.

Capítulo 6. Objetos Os objetos são o tipo de dados mais fundamental do JavaScript, e você tem Já os vi muitas vezes nos capítulos que precedem este. Porque os objetos são muito importantes para a linguagem JavaScript, é importante que você entenda como eles funcionam em detalhes, e este capítulo fornece esse detalhe. Começa com uma visão geral formal dos objetos, então mergulhe em seções práticas sobre a criação de objetos e a consulta, Configuração, exclusão, teste e enumeração das propriedades dos objetos. Essas seções focadas na propriedade são seguidas por seções que explicam Como estender, serializar e definir métodos importantes em objetos. Finalmente, o capítulo termina com uma longa seção sobre novo objeto Sintaxe literal no ES6 e versões mais recentes do idioma.

6.1 Introdução aos objetos

Um objeto é um valor composto: agrupa vários valores (primitivos ou outros objetos) e permite armazenar e recuperar esses valores por nome. Um objeto é uma coleção não ordenada de propriedades, cada um dos quais tem um nome e um valor. Os nomes de propriedades são geralmente strings (embora, como veremos em §6.10.3, os nomes de propriedades também podem ser Símbolos), para que possamos dizer que os objetos mapeiam strings para valores. Esta string-O mapeamento de valor passa por vários nomes - você provavelmente já está familiarizado com a estrutura de dados fundamental sob o nome "Hash", "Hashtable", "Dictionary", ou "matriz associativa". Um objeto é mais do que um simples mapa de string a valor, no entanto. Além de manter

Seu próprio conjunto de propriedades, um objeto JavaScript também herda as propriedades de outro objeto, conhecido como seu "protótipo". Os métodos de um objeto são tipicamente propriedades herdadas, e essa "herança prototípica" é uma das principais características do JavaScript. Os objetos JavaScript são dinâmicos - as propriedades geralmente podem ser adicionadas e excluídas - mas eles podem ser usados para simular os objetos estáticos e estruturas de idiomas típicos estaticamente. Eles também podem ser usados para ignorar a parte do valor do mapeamento de sequência em valor) para representar conjuntos de cordas. Qualquer valor em JavaScript que não é uma string, um número, um símbolo, ou verdadeiro, falso, nulo ou indefinido é um objeto. E mesmo embora cordas, números e booleanos não são objetos, eles podem se comportar como objetos imutáveis. Lembre-se do §3.8 de que os objetos são mutáveis e manipulados por referência em vez de por valor. Se a variável `x` se referir a um objeto e o código `y = x;` é executado, a variável `y` mantém uma referência à mesma objeto, não uma cópia desse objeto. Quaisquer modificações feitas no objeto através da variável `y` também são visíveis através da variável `x`. As coisas mais comuns a ver com objetos são criar -los e definir, Consulta, excluir, teste e enumerar suas propriedades. Estes fundamentais operações são descritas nas seções de abertura deste capítulo. O Seções depois dessa cobertura tópicos mais avançados. Uma propriedade tem um nome e um valor. Um nome de propriedade pode ser qualquer string, incluindo a string vazia (ou qualquer símbolo), mas nenhum objeto pode

Tenha duas propriedades com o mesmo nome.O valor pode ser qualquerValor JavaScript, ou pode ser uma função Getter ou Setter (ou ambos).Aprenderemos sobre as funções Getter e Setter em §6.10.6.Às vezes é importante poder distinguir entre propriedadesdefinido diretamente em um objeto e aqueles que são herdados de umObjeto de protótipo.JavaScript usa o termo propriedade própria para se referir a nãopropriedades herdadas.Além de seu nome e valor, cada propriedade possui três propriedadesAtributos:O atributo gravável especifica se o valor doA propriedade pode ser definida.O atributo enumerável especifica se o nome da propriedadeé devolvido por um loop for/in.O atributo configurável especifica se a propriedade podeser excluído e se seus atributos podem ser alterados.Muitos dos objetos internos de JavaScript têm propriedades que são lidasSomente, não inebriante, ou não confundível.Por padrão, no entanto, todosAs propriedades dos objetos que você cria são graváveis, enumeráveis ??econfigurável.§14.1 explica técnicas para especificar não-defasaValores do atributo de propriedade para seus objetos.6.2 Criando objetosObjetos podem ser criados com literais de objeto, com a nova palavra -chave, ecom a função object.Create ().As subseções abaixo

descreva cada técnica.

6.2.1 Literais de objeto

A maneira mais fácil de criar um objeto é incluir um objeto literal em seu Código JavaScript. Na sua forma mais simples, um objeto literal é uma vírgula separada de nome separado pelo colón: pares de valor, fechados dentroaparelho encaracolado. Um nome de propriedade é um identificador JavaScript ou uma stringliteral (a corda vazia é permitida). Um valor de propriedade é qualquer javascriptexpressão; o valor da expressão (pode ser um valor primitivo ou um valor de objeto) torna -se o valor da propriedade. Aqui estão alguns Exemplos:

```
Seja vazio = {};// um objeto sem propriedades
deixe Point = {x: 0, y: 0};// dois numéricos
propriedades
Seja P2 = {x: Point.x, y: Point.y+1};// mais complexo
valores
Deixe o livro = {"Título principal": "JavaScript", // estas propriedades
Os nomes incluem espaços,
"Sub-título": "O Guia Definitivo", // e Hífens, então
Use literais de string.
para: "todos o público", // para é reservado, Mas sem citações.
autor: {}// o valor desse
propriedade é
FirstName: "David", // em si um objeto.
Sobrenome: "Flanagan"}; Uma vírgula à direita após a
última propriedade em um objeto literal é Legal e alguns estilos de programação incentivam o uso desses
```

vírgulas, portanto, é menos provável que você cause um erro de sintaxe se adicionar um novo propriedade no final do objeto literal em algum momento posterior. Um objeto literal é uma expressão que cria e inicializa um novo objeto distinto cada vez que é avaliado. O valor de cada propriedade é avaliado cada vez que o literal é avaliado. Isso significa que um único Objeto literal pode criar muitos novos objetos se aparecer dentro do corpo de um loop ou em uma função que é chamada repetidamente, e que a propriedade Os valores desses objetos podem diferir um do outro. Os literais do objeto mostrados aqui usam sintaxe simples que tem sido legal Desde as primeiras versões do JavaScript. Versões recentes da linguagem introduziu uma série de novos recursos literais de objeto, que são cobertos em §6.10.6.2.2 Criando objetos com novo O novo operador cria e inicializa um novo objeto. O novo A palavra -chave deve ser seguida por uma invocação de função. Uma função usada em Desta forma é chamado de construtor e serve para inicializar um recém -criado objeto. O JavaScript inclui construtores para seus tipos internos. Para exemplo: Seja o = new Object ()// Crie um objeto vazio: o mesmo que {}. deixe a = new Array ()// Crie uma matriz vazia: o mesmo que []. Seja d = new Date ()// Crie um objeto de data representando o horário atual Seja r = novo map ()// Crie um objeto de mapa para chave/valor mapeamento Além desses construtores embutidos, é comum definir seu

As funções próprias do construtor para inicializar objetos recém -criados.Fazendo issoestá coberto no capítulo 9.6.2.3 ProtótiposAntes de podermos cobrir a terceira técnica de criação de objetos, devemos fazer uma pausa por um momento para explicar protótipos.Quase todo objeto JavaScript tem um segundo objeto JavaScript associado a ele.Este segundo objeto é conhecido como protótipo, e o primeiro objeto herda as propriedades do protótipo.Todos os objetos criados por literais de objeto têm o mesmo objeto de protótipo,e podemos nos referir a este protótipo objeto no código JavaScript como Object.prototype.Objetos criados usando a nova palavra -chave e uma invocação do construtor Use o valor da propriedade do protótipo do construtor funciona como seu protótipo.Então o objeto criado por novo objeto () herda do object.prototype, assim como o objeto criado por {} faz.Da mesma forma, o objeto criado por novoArray () usa Array.prototype como seu protótipo e o objetoCriado por new date () usa date.prototype como seu protótipo.Isso pode ser confuso ao aprender o JavaScript pela primeira vez.Lembrar:Quase todos os objetos têm um protótipo, mas apenas um número relativamente pequeno de objetos têm uma propriedade de protótipo.São esses objetos compropriedades de protótipo que definem os protótipos para todos os outros objetos.Object.Prototype é um dos objetos raros que não possuem protótipo:Não herda nenhuma propriedade.Outros protótipos objetos são normalmente objetos que têm um protótipo.A maioria dos construtores embutidos (e a maioria

construtores definidos pelo usuário) têm um protótipo que herda Object.prototype. Por exemplo, Date.prototype herda propriedades do object.prototype, então um objeto de data criado por new Date() herda as propriedades de ambos Date.prototype e Object.prototype. Esta série vinculada de objetos de protótipo é conhecido como uma cadeia de protótipo. Uma explicação de como a herança da propriedade funciona é no §6.3.2. O capítulo 9 explica a conexão entre protótipos e construtores em mais detalhes: mostra como definir novas "classes" de objetos para escrever uma função construtora e definir sua propriedade de protótipo para o objeto de protótipo a ser usado pelas "instâncias" criadas com isso construtor. E aprenderemos a consultar (e até mudar) o protótipo de um objeto no §14.3.6.2.4 Object.Create(). Object.create() cria um novo objeto, usando seu primeiro argumento como o protótipo desse objeto: Seja o1 = object.create({x: 1, y: 2}); // O1 herda propriedades x e y. O1.x + o1.y // => 3 Você pode passar nulo para criar um novo objeto que não tenha um protótipo, mas se você fizer isso, o objeto recém-criado não herdará Qualquer coisa, nem mesmo métodos básicos como ToString() (o que significa Também não funcionará com o operador +): Seja o2 = object.create(nulo); // O2 herda não adereços ou métodos.

Se você deseja criar um objeto vazio comum (como o objeto retornadopor {} ou novo objeto()), aprove o object.prototype:Seja o3 = object.create (object.prototype);// o3 é como {} ounovo objeto ().A capacidade de criar um novo objeto com um protótipo arbitrário é um poderoso, e usaremos object.create () em várioslugares ao longo deste capítulo.(Object.create () também leva umsegundo argumento opcional que descreve as propriedades do novoobjeto.Este segundo argumento é um recurso avançado coberto no §14.1.)Um uso para object.create () é quando você deseja se protegerModificação não intencional (mas não maliciosa) de um objeto por uma bibliotecafunção que você não tem controle.Em vez de passar o objetoDiretamente para a função, você pode passar um objeto que herda dela.SeA função lê propriedades desse objeto, verá o herdadovalores.Se definir propriedades, no entanto, essas gravações não afetarão oobjeto original.Seja o = {x: "Não altere este valor"};Library.function (object.create (o));// se protege contramodificações accidentaisPara entender por que isso funciona, você precisa saber como são as propriedadesconsultado e definido em JavaScript.Estes são os tópicos da próxima seção.6.3 Propriedades de consulta e definiçãoPara obter o valor de uma propriedade, use o ponto(.) Ou suporte quadrado

([]) Os operadores descritos em §4.4.O lado esquerdo deve ser um expressão cujo valor é um objeto.Se estiver usando o operador de pontos, oO lado arete deve ser um identificador simples que nomeia a propriedade.SeUsando suportes quadrados, o valor dentro dos colchetes deve ser umexpressão que avalia a uma string que contém a propriedade desejada:Let Author = Book.author;// Obtenha a propriedade "Autor"do livro.Deixe o nome = Author.surname;// Obtenha a propriedade "Sobrenome"do autor.deixe title = livro ["título principal"];// Obtenha o "título principal"propriedade do livro.Para criar ou definir uma propriedade, use um ponto ou suportes quadrados como fariapara consultar a propriedade, mas coloque -os no lado esquerdo de umExpressão de atribuição:book.edition = 7;// Crie uma "edição"propriedade do livro.livro ["título principal"] = "ecmascript";// Alterar o "principaltítulo "Propriedade.Ao usar a notação de suporte quadrado, dissemos que a expressãoDentro dos suportes quadrados devem avaliar uma corda.Um mais precisodeclaração é que a expressão deve avaliar uma string ou um valor quepode ser convertido em uma corda ou em um símbolo (§6.10.3).No capítulo 7, paraPor exemplo, veremos que é comum usar números dentro do quadradoSuportes.6.3.1 Objetos como matrizes associativasConforme explicado na seção anterior, os dois JavaScript a seguir

Expressões têm o mesmo valor:`Object.PropertyObjeto ["Propriedade"]`A primeira sintaxe, usando o ponto e um identificador, é como a sintaxe usada para acessar um campo estático de uma estrutura ou objeto em C ou Java.O segundo Sintaxe, usando suportes quadrados e uma corda, parece acesso de matriz, mas uma matriz indexada por strings e não por números.Esse tipo de matriz é conhecida como uma matriz associativa (ou hash, mapa ou dicionário).Objetos JavaScript são matrizes associativas, e esta seção explica o porquê do porquê isso é importante.Em C, C ++, Java e idiomas fortemente digitados fortemente, um objeto pode ter apenas um número fixo de propriedades e os nomes destesAs propriedades devem ser definidas com antecedência.Como JavaScript é vagamente idioma digitado, esta regra não se aplica: um programa pode criar qualquer número de propriedades em qualquer objeto.Quando você usa o operador para acessar uma propriedade de um objeto, no entanto, o nome da propriedade é expresso como um identificador.Os identificadores devem ser digitados literalmente em seu Programa JavaScript;eles não são um tipo de dados, então não podem ser manipulados pelo programa.Por outro lado, quando você acessa uma propriedade de um objeto com o `[]`Notação de matriz, o nome da propriedade é expresso como uma string.Cordassão tipos de dados JavaScript, para que possam ser manipulados e criados enquanto um programa está em execução.Por exemplo, você pode escrever o seguinte Código em JavaScript:deixe `addr = "";`

para (vamos i = 0; i <4; i++) {addr + = cliente ['endereço \$ {i}`] + "\ n";}Este código lê e concatena o endereço0, endereço1,As propriedades de endereço2 e endereço3 do objeto do cliente.Este breve exemplo demonstra a flexibilidade de usar a notação de matrizPara acessar as propriedades de um objeto com expressões de string.Este código pode ser reescrito usando a notação de pontos, mas há casos em queSomente a notação da matriz serve.Suponha, por exemplo, que você sejaEscrever um programa que usa recursos de rede para calcular o atualValor dos investimentos no mercado de ações do usuário.O programa permite oUsuário para digitar o nome de cada estoque que eles possuem, bem como o número de ações de cada ação.Você pode usar um objeto chamado PortfolioPara manter essas informações.O objeto possui uma propriedade para cada estoque.O nome da propriedade é o nome do estoque e a propriedadeO valor é o número de ações dessas ações.Então, por exemplo, se um usuáriodetém 50 ações da IBM, a propriedade portfolio.ibmo valor 50.Parte deste programa pode ser uma função para adicionar um novo estoque aoPortfólio:função addstock (portfólio, nome da estoque, ações) {Portfolio [SockName] = ações;}Como o usuário insere nomes de estoque em tempo de execução, não há como vocêpode conhecer os nomes de propriedades antes do tempo.Já que você não pode saber o

nomes de propriedades Quando você escreve o programa, não há como você pode usar o operador para acessar as propriedades do objeto do portfólio. Você pode usar o operador [], no entanto, porque ele usa um valor de string (que é dinâmico e pode mudar em tempo de execução) em vez de um identificador (que é estático e deve ser codificado no programa) para citar a propriedade. No capítulo 5, introduzimos o loop for/in (e veremos novamente em breve, em §6.6). O poder desta declaração JavaScript fica claro quando você considera seu uso com matrizes associativas. Aqui está como você usaria -o ao calcular o valor total de um portfólio:

```
function computeValue (portfólio) {Deixe total = 0,0; para (deixe estoque no portfólio) { // para cada estoque em portfólio: Let ações = portfólio [estoque]; // Obtenha o número de ações Deixe o preço = getQuote (estoque); // Procure compartilhar preço Total += Ações * Preço; // Adicione o valor do estoque a valor total} retorno total; // retorna total valor.} Os objetos JavaScript são comumente usados ?? como matrizes associativas, como mostrado aqui, e é importante entender como isso funciona. Em ES6 e Mais tarde, no entanto, a classe de mapa descrita em §11.1.2 é frequentemente uma melhor escolha do que usar um objeto simples.
```

6.3.2 Herança

Os objetos JavaScript têm um conjunto de "próprias propriedades" e também herdam um conjunto de propriedades de seu objeto de protótipo. Para entender isso, nós devemos considerar o acesso à propriedade com mais detalhes. Os exemplos nessa seção usam a função `Object.create()` para criar objetos com protótipos especificados. Veremos no capítulo 9, no entanto, que toda vez que você cria uma instância de uma classe nova, você está criando um objeto que herda as propriedades de um objeto de protótipo. Suponha que você consulte a propriedade `x` no objeto `o`. Se `O` não tiver uma propriedade própria com esse nome, o protótipo do objeto de `O` é consultado para a propriedade `x`. Se o protótipo do objeto não tiver uma propriedade própria por esse nome, mas tem um protótipo em si, a consulta é realizada no protótipo do protótipo. Isso continua até que a propriedade `X` seja encontrada ou até que um objeto com um protótipo nulo seja pesquisado. Como você pode ver, o atributo do protótipo de um objeto cria uma cadeia ou lista vinculada onde as propriedades são herdadas: Seja `o = {};` // o herda métodos de objeto `Object.prototype`. `o.x = 1;` // e agora tem uma propriedade própria `x`. Seja `p = Object.create(o);` // `p` herda propriedades de `O` e `Object.prototype`. `p.y = 2;` // e tem uma propriedade própria `y`. Seja `q = Object.create(p);` // `q` herda propriedades de `p`, `o`, etc... `q.z = 3;` // ... `Object.prototype` e tem uma propriedade própria `z`. Seja `f = q.toString();` // `toString` é herdado de `Object.prototype`. `q.x + q.y` // => 3; `X` e `Y` são herdados de `o` e `p`.

Agora, suponha que você atribua à propriedade X do objeto o. Se o já possui uma propriedade própria (não-herdada) chamada X, depois a tarefa simplesmente altera o valor desta propriedade existente. Caso contrário, o A tarefa cria uma nova propriedade chamada x no objeto o. Se o anteriormente herdou a propriedade X, essa propriedade herdada é agora escondida pela propriedade própria recém-criada com o mesmo nome. A atribuição de propriedade examina a cadeia de protótipo apenas para determinar se a tarefa é permitida. Se o herdar uma propriedade somente leituraNomeado X, por exemplo, a atribuição não é permitida. (Detalhes sobre quando uma propriedade pode ser definida estão em §6.3.3.) Se a tarefa for permitido, no entanto, sempre cria ou define uma propriedade no originalObjeto e nunca modifica objetos na cadeia de protótipos. O fato dissoA herança ocorre ao consultar propriedades, mas não quando as definiré uma característica fundamental do JavaScript, porque nos permite seletivamente substituir propriedades herdadas: Seja unitcircle = {r: 1}; // um objeto para herdar de Seja c = object.create(unitcircle); // c herda a propriedade c.x = 1; c.y = 1; // c define duas propriedades próprias c.r = 2; // c substitui sua propriedade herdada unitcircle.r // => 1: o protótipo é afetado. Há uma exceção à regra de que uma tarefa de propriedade também falha ou cria ou define uma propriedade no objeto original. Se o herdar o Propriedade X, e essa propriedade é uma propriedade acessadora com um setterMétodo (ver §6.10.6), então esse método do setter é chamado em vez de

criando uma nova propriedade x em o. Observe, no entanto, que o método do setter é chamado ao objeto O, não no objeto de protótipo que define a propriedade, portanto, se o método do setter definir alguma propriedade, fará isso em o, e isso deixará a cadeia de protótipo não modificada.

6.3.3 Erros de acesso à propriedade

Expressões de acesso à propriedade nem sempre retornam ou definem um valor. Esseja seção explica as coisas que podem dar errado quando você consulta ou define uma propriedade. Não é um erro consultar uma propriedade que não existe. Se a propriedade x não é encontrada como uma propriedade própria ou uma propriedade herdada de O, o A expressão de acesso à propriedade O.x avalia a indefinida. Lembre -se disso. Nossobjeto de livro possui uma propriedade "subtítulo", mas não uma propriedade "legenda": book.subtitle // => indefinido: a propriedade não existe. É um erro, no entanto, tentar consultar uma propriedade de um objeto que não existe. Os valores nulos e indefinidos não têm propriedades. É um erro consultar propriedades desses valores. Continuando o exemplo anterior: Seja len = book.subtitle.length; //! TypeError: indefinido não tem comprimento. As expressões de acesso à propriedade falharão se o lado esquerdo do .é nulo ou indefinido. Então, ao escrever uma expressão como book.author.surname, você deve ter cuidado se não estiver certamente que o livro e o autor são realmente definidos. Aqui estão:

Duas maneiras de se proteger contra esse tipo de problema:// Uma técnica detalhada e explícitaSeja sobrenome = indefinido;if (livro) {if (book.author) {Sobrenome = book.author.surname;}}// Uma alternativa concisa e idiomática para obter sobrenome ou nuloou indefinidoSobrenome = book && book.author && book.author.surname;Para entender por que essa expressão idiomática funciona para prevenirExceções TypeError, você pode querer revisar o curto-circuitoComportamento do operador && em §4.10.1.Conforme descrito em §4.4.1, o ES2020 suporta acesso à propriedade condicionalcom?., que nos permite reescrever a tarefa anteriorexpressão como:Deixe o sobrenome = livro? .Author? .Surname;Tentar definir uma propriedade em nulo ou indefinido também causa umTypeError.Tentativas de definir propriedades em outros valores nem sempreTer sucesso, também: algumas propriedades são somente leitura e não podem ser definidas, eAlguns objetos não permitem a adição de novas propriedades.Em rigorosoModo (§5.6.3), um TypeError é jogado sempre que uma tentativa de definir umA propriedade falha.Fora do modo rigoroso, essas falhas geralmente são silenciosas.As regras que especificam quando uma atribuição de propriedade é bem -sucedida e quandoAs falhas são intuitivas, mas difíceis de expressar de forma concisa.Uma tentativa de definir

Uma propriedade P de um objeto O falha nessas circunstâncias: o tem uma propriedade própria P que é somente leitura: não é possível definir propriedades somente leitura. o tem uma propriedade herdada P que é somente leitura: não é possível esconder uma propriedade somente leitura herdada com um próprio propriedade de mesmo nome. o não possui uma propriedade própria p; O não herda um Propriedade P com um método de setter e o atributo extensível de O (Ver §14.2) é falso. Já que P ainda não existe em O e Se não houver um método Setter para chamar, P deverá ser adicionado a O. Mas se O não for extensível, nenhuma nova propriedade pode ser definida nele.

6.4 Excluindo propriedades

O operador de exclusão (§4.13.4) remove uma propriedade de um objeto. Isso é Operando único deve ser uma expressão de acesso à propriedade. Surpreendentemente, excluir não opera sobre o valor da propriedade, mas no Propriedade própria: excluir book.author;// O objeto do livro agora não tem Propriedade do autor. excluir livro ["título principal"];// agora não tem "principal" título ", também. O operador de exclusão exclui apenas propriedades próprias, não herdadas. (Para excluir uma propriedade herdada, você deve excluir -la do protótipo objeto no qual é definido. Fazer isso afeta todos os objetos que herda desse protótipo.)

Uma expressão de exclusão avalia para verdadeira se a exclusão for bem-sucedida ou se o delete não teve efeito (como excluir uma propriedade inexistente). Excluir também avalia como verdadeiro quando usado (sem sentido) com uma expressão que não é uma expressão de acesso à propriedade: Seja o = {x: 1}; // O tem propriedade própria x e herdaPropriedade ToStringExclua o.x // => true: exclui a propriedade xExclua o.x // => true: não faz nada (x não existe) Mas é verdade de qualquer maneira excluir o.toString // => true: não faz nada (a toString não é uma propriedade própria) exclua 1 // => true: bobagem, mas verdade de qualquer maneira Delete não remove propriedades que têm um atributo configurável falso. Certas propriedades de objetos embutidos não são confundíveis, assim como as propriedades do objeto global criado pela Declaração Variávele declaração de função. No modo rigoroso, tentando excluir um nãoA propriedade configurável causa um TypeError. No modo não estrito, exclua Simplesmente avalia o FALSE neste caso: // No modo rigoroso, todas essas deleções jogam TypeErrorEm vez de retornar falso excluir object.prototype // => false: propriedade não é configurável var x = 1; // declarar uma variável global exclua globalthis.x // => false: não é possível excluir issopropriedade função f () {} // declarar uma função global exclua globalthis.f // => false: Não é possível excluir issopropriedade também Ao excluir propriedades configuráveis ??do objeto global no não ritomodo, você pode omitir a referência ao objeto global e simplesmente Siga o operador Excluir com o nome da propriedade:

globalthis.x = 1;// Crie um global configurável propriedade (sem deixar ou var) exclua x // => true: esta propriedade pode ser excluída no modo rigoroso, no entanto, o Delete levanta um SyntaxError se o seu operando estiver um identificador não qualificado como x, e você deve ser explícito sobre o acesso à propriedade: excluir x;// SyntaxError no modo rigoroso exclui globalthis.x;// Isso funciona 6.5 Propriedades de teste Os objetos JavaScript podem ser pensados ?? como conjuntos de propriedades, e é frequentemente útil para poder testar a participação no set - para verificar se um objeto tem uma propriedade com um determinado nome. Você pode fazer isso com o operador, com o HasOwnProperty () e PropertyIsEnumerable () métodos, ou simplesmente consultando o próprio. Os exemplos mostrados aqui todos usam strings como nomes de propriedades, mas eles também trabalham com símbolos (§6.10.3). O operador IN espera um nome de propriedade no lado esquerdo e um objeto à sua direita. Ele retorna verdadeiro se o objeto tiver uma propriedade própria ou uma propriedade herdada por esse nome: Seja o = {x: 1}; "X" em o // => true: o tem uma propriedade própria "x" "y" em o // => false: o não tem uma propriedade "y" "ToString" em o // => true: o herda uma propriedade ToString

O método HasOwnProperty () de um objeto testa se issoO objeto possui uma propriedade própria com o nome fornecido. Retorna falsa paraPropriedades herdadas: Seja o = {x: 1}; O.hasOwnProperty ("x") // => true: o tem um próprioPropriedade xO.hasOwnProperty ("y") // => false: o não tem umPropriedade yO.hasOwnProperty ("ToString") // => false: ToString é umpropriedade herdadaO PropertyIsEnumerable () refina o teste HASOWNPROPERTY (). Ele retorna verdadeiro apenas se o nomeadoA propriedade é uma propriedade própria e seu atributo enumerável é verdadeiro. Certas propriedades embutidas não são enumeráveis. Propriedades criadas porO código JavaScript normal é enumerável, a menos que você tenha usado um dosTécnicas mostradas no §14.1 para torná-las que não são inebriantes. Seja o = {x: 1}; O.PropertyIsEnumerable ("x") // => true: o tem um propriopropriedade enumerable xO.PropertyIsEnumerable ("ToString") // => false: não um propriopropriedadeObject.prototype.propertyIsEnumerable ("toString") // => Falso: não enumerávelEm vez de usar o operador em In, muitas vezes é suficiente simplesmente consultara propriedade e o uso! == para garantir que não seja indefinido: Seja o = {x: 1}; o.x! == indefinido // => true: o tem uma propriedade xo.y! == indefinido // => false: o não tem umPropriedade yO.ToString! == indefinido // => true: o herda uma toque

propriedadeHá uma coisa que o operador pode fazer que a propriedade simplesA técnica de acesso mostrada aqui não pode fazer.pode distinguir entrepropriedades que não existem e propriedades que existem, mas foram definidas como indefinido.Considere este código:Seja o = {x: indefinido};// a propriedade está explicitamente definida como indefinido.o.x! == indefinido // => false: a propriedade existe, mas é indefinidoO.Y! == indefinido // => Falso: Propriedade nem mesmo existe"X" em o // => true: a propriedade existe"y" em o // => false: a propriedade não existeexcluir o.x;// Exclua a propriedade x"x" em o // => false: não existem mais6.6 Propriedades de enumeraçãoEm vez de testar a existência de propriedades individuais, nós às vezes queremos iterar ou obter uma lista de todas as propriedades de um objeto.Existem algumas maneiras diferentes de fazer isso.O loop for/in foi coberto no §5.4.5.Ele executa o corpo do loop uma vez para cada propriedade enumerável (própria ou herdada) do especificado objeto, atribuindo o nome da propriedade à variável loop.Embora os métodos que objetos herdam não são enumeráveis, mas as propriedades que seu código adiciona aos objetos é enumerável por padrão.Por exemplo:Seja o = {x: 1, y: 2, z: 3};// Três enumeráveis ??próprios propriedades

O.PropertyIsEnumerable ("ToString") // => Falso: não é enumerável para (vamos P in O) { // percorrer
propriedades console.log (P); // imprime x, y e z, mas não ToString} Para se proteger contra a enumeração de
propriedades herdadas com/in, você pode adicionar uma verificação explícita dentro do corpo do loop: para (vamos P in O) {if (!O.HasOwnProperty (P)) continuar; // Pular propriedades herdadas} para (vamos P in O) {if (typeof o [p]
== "function") continue; // Pule tudo Métodos} Como alternativa ao uso de um loop for/in, muitas vezes é mais fácil
obter um matriz de nomes de propriedades para um objeto e depois percorre essa matriz com um loop
para/de Existem quatro funções que você pode usar para obter um array de nomes de propriedades: Object.Keys ()
retorna uma matriz dos nomes das propriedades próprias enumeráveis ?? de um objeto. Não inclui propriedades não
enumeráveis, propriedades herdadas ou propriedades cujo nome é um símbolo (ver
§6.10.3). Object.getOwnPropertyNames () funciona como Object.Keys (), mas retorna uma variedade de nomes de
não-propriedades próprias também, desde que seus nomes sejam cordas.

`Object.getOwnPropertySymbols()` Retorna propriapropriedades cujos nomes são símbolos, sejam elesenumerável.`Reflete.ownkeys()` retorna todos os próprios nomes de propriedades, ambosenumerável e não enumerável, e símbolo e símbolo.(Veja §14.6.)Existem exemplos do uso de `object.keys()` com um para/deLoop em §6.7.6.6.1 Ordem de enumeração da propriedadeES6 define formalmente a ordem em que as próprias propriedades de umobjeto são enumerados.`Object.keys()`,`Object.getOwnPropertyNames()`,`Object.getOwnPropertySymbols()`,`Reflete.ownskeys()` e métodos relacionados, como`Json.Stringify()` todas as propriedades da lista na seguinte ordem,sujeito a suas próprias restrições adicionais sobre se eles listam nãopropriedades ou propriedades enumeráveis ??cujos nomes são strings ouSímbolos:Propriedades de string cujos nomes são inteiros não negativos sãoListado primeiro, em ordem numérica do menor ao maior.Esta regrasignifica que matrizes e objetos semelhantes a matrizes terão seu(propriedades enumeradas em ordem.Afinal, todas as propriedades que parecem índices de matriz estão listadas, todasAs propriedades restantes com nomes de strings estão listadas (incluindo(propriedades que parecem números negativos ou ponto flutuantenúmeros).Essas propriedades estão listadas na ordem em queEles foram adicionados ao objeto.Para propriedades definidas em um

objeto literal, essa ordem é a mesma ordem que eles aparecem no literal. Finalmente, as propriedades cujos nomes são objetos de símbolo são listados na ordem em que foram adicionados ao objeto. A ordem de enumeração para o loop `for/in` não é tão bem especificada como é para essas funções de enumeração, mas implementações normalmente enumerar as próprias propriedades na ordem que acabamos de descrever e depois viaje o protótipo. Cadeia de enumeração de propriedades na mesma ordem para cada objeto de protótipo. Observe, no entanto, que uma propriedade não será enumerada se uma propriedade nesse mesmo nome já foi enumerada, ou mesmo se uma propriedade não entusiasmada com o mesmo nome já foi considerado.

6.7 estendendo objetos

Uma operação comum em programas JavaScript está precisando copiar as propriedades de um objeto para outro objeto. É fácil fazer isso com código assim:

```
Deixe o destino = {x: 1}, fonte = {y: 2, z: 3}; para (deixe a chave do object.keys (fonte)) {Target [key] = fonte [chave];} Target // => {x: 1, y: 2, z: 3}
```

Mas porque esta é uma operação comum, vários JavaScript estruturas definidas funções de utilidade, geralmente nomeadas `estend ()`, para executar esta operação de cópia. Finalmente, no ES6, essa habilidade vem para a linguagem principal JavaScript na forma de `object.assign ()`.

`Object.assign ()` espera dois ou mais objetos como seus argumentos. Isto modifica e retorna o primeiro argumento, que é o objeto de destino, mas não altera o segundo ou nenhum argumento subsequente, que são os objetos de origem. Para cada objeto de origem, ele copia o próprio próprioPropriedades desse objeto (incluindo aqueles cujos nomes são símbolos) no objeto de destino. Ele processa os objetos de origem na lista de argumentos encomendar para que as propriedades na primeira fonte substituam as propriedades do mesmo nome no objeto de destino e propriedades na segunda fonte objeto (se houver um) substituir as propriedades com o mesmo nome no Objeto de primeira fonte. `Object.assign ()` copia as propriedades com propriedades comuns GET e Defina operações, portanto, se um objeto de origem tiver um método getter ou o alvoObjeto tem um método de setter, eles serão invocados durante a cópia, mas eles não serão copiados. Um motivo para atribuir propriedades de um objeto para outro é quando você tem um objeto que define valores padrão para muitas propriedades evocê deseja copiar essas propriedades padrão em outro objeto se uma propriedade com esse nome ainda não existe nesse objeto. Usando `Object.assign ()` ingenuamente não fará o que você deseja:
`Object.assign (O, padrões);`// substitui tudo em O com padrões Em vez disso, o que você pode fazer é criar um novo objeto, copiar os padrões nele e depois substituir esses padrões com as propriedades em O:
`o = object.assign ({}, padrões, o);`

Veremos em §6.10.4 que você também pode expressar este objeto Copiar e-Substituir a operação usando o ... Spread Operator como este:
o = {... padrão, ... o}; Também poderíamos evitar a sobrecarga da criação extra de objetos copiando escrevendo uma versão do object.assign () que copia propriedades apenas se estiverem faltando:// como object.assign (), mas não substitui propriedades// (e também não lida com propriedades de símbolo) função mescle (alvo, ... fontes) {para (deixe a fonte de fontes) {para (deixe a chave do object.Keys (fonte)) {if (! (chave no alvo)) { // isso é diferente de Object.assign () Target [key] = fonte [chave]; }}}}} alvo de retorno; } Object.assign ({x: 1}, {x: 2, y: 2}, {y: 3, z: 4}) // => {x:2, y: 3, z: 4} Merge ({x: 1}, {x: 2, y: 2}, {y: 3, z: 4}) // => {x:1, y: 2, z: 4} É simples escrever outros utilitários de manipulação de propriedades, como esta função mescle (). Uma função RESTRICT () pode excluir propriedades de um objeto se não aparecerem em outro objeto de modelo, por exemplo. Ou uma função subtract () pode remover toda apropriedade de um objeto de outro objeto.

6.8 Objetos serializadosA serialização do objeto é o processo de converter o estado de um objeto em um string a partir da qual pode ser restaurada posteriormente.As funções `JSON.stringify ()` e `JSON.parse ()` serializar e restaurar Objetos javascript.Essas funções usam o intercâmbio de dados JSON para formatar.JSON significa "notação de objeto JavaScript" e sua sintaxe é muito semelhante ao do objeto JavaScript e dos literais da matriz:Seja o = {x: 1, y: {z: [false, null, ""]}};// Defina um teste de objetoSeja s = json.Stringify (O);// s == '{"x": 1, "y": {"z": [False, Null, ""]}}'Seja p = json.parse (s);// p == {x: 1, y: {z: [false, nulo, ""]}}A sintaxe JSON é um subconjunto de sintaxe JavaScript, e não pode representar todos os valores JavaScript.Objetos, matrizes, cordas, números finitos, verdadeiro,Falso, e nulo são suportados e podem ser serializados e restaurados.Nan, infinito e -infinity são serializados para nulos.DataOs objetos são serializados para as seqüências de datas formatadas iso (ver `oDate.toJSON ()` função), mas `json.parse ()` deixa -os em formulário de string e não restaura o objeto Data original.Função,Regexp e objetos de erro e o valor indefinido não pode ser serializado ou restaurado.`JSON.stringify ()` serializa apenas as propriedades próprias enumeráveis ??de um objeto.Se um valor de propriedade não pode ser serializado, essa propriedade é simplesmente omitida da saída straciificada.`JSON.STRINGIFY ()` e `JSONPARSE ()` aceitam opcionalSegundo argumentos que podem ser usados ??para personalizar a serialização e/ou processo de restauração especificando uma lista de propriedades a serem serializadas, para exemplo, ou convertendo certos valores durante a serialização ou

Processo de Stringification. A documentação completa para essas funções é no §11.6.6.9 Métodos de objeto. Como discutido anteriormente, todos os objetos JavaScript (exceto aqueles explicitamente criados sem um protótipo) herdar propriedades de Object.prototype. Essas propriedades herdadas são principalmente métodos, e porque eles estão disponíveis universalmente, eles são de interesse particular para programadores JavaScript. Nós já vimos os métodos HASOwnProperty () e PROPERTY_IS_ENUMERABLE (), por exemplo. (E também já cobrimos alguns métodos estáticos definidos no construtor de objeto, como Object.Create () e Object.Keys ().) Esta seção explica um punhado de métodos de objetos universais que são definidos em Object.prototype, mas que devem ser substituídos por outras implementações mais especializadas. Nas seções a seguir, mostramos exemplos de definição desses métodos em um único objeto. Em Capítulo 9, você aprenderá como definir esses métodos de maneira mais geral para uma classe inteira de objetos.

6.9.1 O método ToString ()

O método toString () não leva argumentos; ele retorna uma string que, de alguma forma, representa o valor do objeto em que ele é chamado. JavaScript invoca esse método de um objeto sempre que precisar converter o objeto em uma string. Isso ocorre, por exemplo, quando você usa o operador + para concatenar uma string com um objeto ou quando você passa um objeto para um método que espera uma string.

O método padrão `toString()` não é muito informativo (embora seja útil para determinar a classe de um objeto, como veremos no §14.4.3). Por exemplo, a seguinte linha de código simplesmente avalia na string "[Objeto objeto]": Seja `s = {x: 1, y: 1} .toString();`// s == "[objeto objeto]" Porque esse método padrão não exibe muita informação útil. Muitas classes definem suas próprias versões do `ToString()`. Para exemplo, quando uma matriz é convertida em uma string, você obtém uma lista dos elementos da matriz, eles mesmos se converteram em uma corda, e quando uma função é convertida em uma string, você obtém o código -fonte para a função. Você pode definir seu próprio método `ToString()` como este: Deixe `Point = {x: 1,y: 2,tostring: function () {return `($ {this.x}, $ {this.y})`;}},String (ponto) // => "(1, 2)":` `tostring()` é usado para conversões de string.

6.9.2 O método `toLocaleString()`

Além do método BASIC `ToString()`, todos têm `toLocaleString()`. O objetivo deste método é retornar uma representação de string localizada do objeto. O padrão `toLocaleString()` definido pelo objeto não faz nenhuma localização em si: ele simplesmente chama `ToString()` e retorna esse valor. As classes de data e número definem versões personalizadas de

toLocaleString () que tentam formatar números, datas e vezes de acordo com as convenções locais. Array define um método `Tolocalestring ()` que funciona como `ToString ()`, exceto que formata os elementos da matriz chamando sua `toLocalestring ()` em vez de seus métodos `toString ()`. Você pode fazer a mesma coisa com um objeto de ponto como este:

```
Deixe Point = {x: 1000,y: 2000,tostring: function () {return `($ {this.x}, $ {this.y})`}; Tolocalestring: function () {retornar `($ {this.x.toLocalestring ()},$ {this.y.toLocalestring ()})`}};Point.ToString () // => "(1000, 2000)"Point.Tolocalestring () // => "(1.000, 2.000)": Notamilhares de separadoresAs aulas de internacionalização documentadas no §11.7 podem ser úteisAo implementar um método toLocalestring ().
```

6.9.3 o método `ValueOf ()`O método `ValueOf ()` é muito parecido com o método `ToString ()`, mas é chamado quando JavaScript precisa converter um objeto para alguns tipo primitivo que não seja uma string - normalmente, um número.Chamadas JavaScriptEste método automaticamente se um objeto for usado em um contexto em que umÉ necessário valor primitivo.O método padrão de `valueof ()` fazNada interessante, mas algumas das classes embutidas definem seus próprios

Erro ao traduzir esta página.

Erro ao traduzir esta página.

digite literalmente no seu código -fonte. Em vez disso, o nome da propriedade que você precisa é armazenado em uma variável ou é o valor de retorno de uma função que você invocar. Você não pode usar um objeto básico literal para esse tipo de propriedade. Em vez disso, você precisa criar um objeto e depois adicionar o desejo de propriedades como uma etapa extra:

```
const Property_name = "P1";function computEPropertyName () {return "p" + 2;}
```

Seja o = {};
o [property_name] = 1;
o [computePropertyName ()] = 2;
É muito mais simples configurar um objeto como este com um recurso ES6 Conhecido como propriedades computadas que permitem pegar os suportes quadrados do código anterior e mover -os diretamente para o objeto literal:

```
const Property_name = "P1";function computEPropertyName () {return "p" + 2;}
```

Seja p = {[Property_name]: 1,[ComputePropertyName ()]: 2};
p.p1 + p.p2 // => 3Com esta nova sintaxe, os parênteses quadrados delimitam um arbitrário Expressão de JavaScript. Essa expressão é avaliada e o resultado O valor (convertido em uma string, se necessário) é usado como o nome da propriedade. Uma situação em que você pode querer usar propriedades computadas é Quando você tem uma biblioteca de código JavaScript que espera ser passado objetos com um determinado conjunto de propriedades e os nomes daqueles

As propriedades são definidas como constantes nessa biblioteca. Se você está escrevendo código para criar os objetos que serão passados ?? para essa biblioteca, você pode hardcode os nomes de propriedades, mas você corre o risco de bugs se digitar o nome da propriedade errado em qualquer lugar e você arriscará a incompatibilidade da versão. Problemas se uma nova versão da biblioteca alterar a propriedade necessária. Em vez disso, você pode achar que torna seu código mais robusto para usar a sintaxe de propriedade computada com as constantes de nome da propriedade definido pela biblioteca.

6.10.3 Símbolos como nomes de propriedades

A sintaxe da propriedade computada permite um outro objeto muito importante: a característica literal. No ES6 e posterior, os nomes de propriedades podem ser strings ou símbolos. Se você atribuir um símbolo a uma variável ou constante, então você pode usar esse símbolo como um nome de propriedade usando a propriedade computada:

```
const extensao = "meu símbolo de extensão";
const o = {[extensão]: { / * Dados de extensão armazenados neste objeto * / }};
o[extensão].x = 0; // Isso não vai conflitar com outro
```

Propriedades de O. Conforme explicado no §3.6, os símbolos são valores opacos. Você não pode fazer qualquer coisa com eles além de usá-los como nomes de propriedades. Todo Símbolo é diferente de todos os outros símbolos, no entanto, o que significa que os símbolos são bons para criar nomes exclusivos de propriedades. Crie um novo símbolo chamando a função de fábrica de símbolo () . (Símbolos são valores primitivos, não objetos, então símbolo () não é um construtor)

função que você chama com o novo.) O valor retornado pelo símbolo ()não é igual a nenhum outro símbolo ou outro valor.Você pode passar uma cordapara symbol (), e essa string é usada quando seu símbolo é convertidopara uma string.Mas este é apenas um auxílio de depuração: dois símbolos criados comO mesmo argumento de string ainda é diferente um do outro.O ponto dos símbolos não é segurança, mas definir uma extensão seguraMecanismo para objetos JavaScript.Se você receber um objeto de terceiroscódigo que você não controla e precisa adicionar alguns dos seus própriospropriedades para esse objeto, mas querem ter certeza de que suas propriedades vãonão conflito com nenhuma propriedade que já possa existir no objeto,Você pode usar com segurança símbolos como nomes de propriedades.Se você fizer isso, vocêtambém pode estar confiante de que o código de terceiros não será accidentalmenteAltere suas propriedades simbolicamente nomeadas.(Esse código de terceiros poderia,Obviamente, use o object.getownPropertySymbols () para descobrirOs símbolos que você está usando e podem alterar ou excluir seupropriedades.É por isso que os símbolos não são um mecanismo de segurança.)6.10.4 Operador de espalhamentoNo ES2018 e mais tarde, você pode copiar as propriedades de um objeto existenteem um novo objeto usando o "operador de espalhamento" ... dentro de um objetaliteral:deixe a posição = {x: 0, y: 0};Let Dimensions = {Width: 100, Hight: 75};Deixe rect = {... posicionar, ... dimensões};rect.x + ret.y + ret.width + ret.height // => 175Neste código, as propriedades da posição e dimensões

Objetos são "espalhados" para o objeto Rect literal como se tivessem sido escrito literalmente dentro desses aparelhos encaracolados. Observe que isso ... sintaxe é frequentemente chamado de operador de spread, mas não é um verdadeiro operador de javascript em qualquer sentido. Em vez disso, é uma sintaxe de caso especial disponível apenas dentro de objetos literais. (Três pontos são usados ?? para outros propósitos em outros contextos de javascript, mas literais de objeto são o único contexto em que os três pontos causam esse tipo de interpolação de um objeto em outro.) Se o objeto que é espalhado é o objeto que ele está sendo espalhado para ambos, tenha uma propriedade com o mesmo nome, depois o valor dessa propriedade será o que vem por último: Seja o = {x: 1}; Seja p = {x: 0, ... o}; p.x // => 1: o valor do objeto o substitui o inicial valor. Seja q = {... o, x: 2}; q.x // => 2: O valor 2 substitui o valor anterior de o. Observe também que o operador de propagação apenas espalha as próprias propriedades de um objeto, não herdado: Seja o = Object.create ({x: 1}); // o herda a propriedade x. Seja p = {... o}; p.x // => indefinido. Finalmente, vale a pena notar que, embora o operador de propagação seja apenas três pequenos pontos em seu código, ele pode representar uma quantidade substancial de trabalho para o intérprete JavaScript. Se um objeto tem n propriedades, o processo de espalhar essas propriedades em outro objeto provavelmente será

uma operação O (n). Isso significa que se você se encontrar usando ...dentro de um loop ou função recursiva como uma maneira de acumular dados em um objeto grande, você pode estar escrevendo um algoritmo O (n) ineficiente que não será escalonado bem como n cresce.

6.10.5 Métodos de abreviação

Quando uma função é definida como uma propriedade de um objeto, chamamos de função um método (teremos muito mais a dizer sobre os métodos em Capítulos 8 e 9). Antes do ES6, você definiria um método em um objeto literal usando uma expressão de definição de função como:

```
Deixe Square = {área: function () {return this.side * this.side;}, lado: 10};
```

square.area () // => 100

No ES6, no entanto, a sintaxe literal do objeto (e também a definição de classe Sintaxe que veremos no capítulo 9) foi estendida para permitir um atalho onde a palavra-chave da função e o colón são omitidos, resultando em código como este:

```
Deixe Square = {área () {return this.side * this.side;}, lado: 10};
```

square.area () // => 100

Ambas as formas do código são equivalentes: ambos adicionam uma propriedade nomeada área para o objeto literal e ambos definem o valor dessa propriedade para o lado.

função especificada. A sintaxe abreviada deixa mais claro essa área () é um método e não uma propriedade de dados como o lado. Quando você escreve um método usando esta sintaxe abreviada, a propriedade de nome pode assumir qualquer um dos formulários que são legais em um objeto literal: Além de um identificador javascript regular, como a área de nome acima, Você também pode usar literais de cordas e nomes de propriedades computadas, que pode incluir nomes de propriedades de símbolo: const método_name = "m"; const símbolo = símbolo (); Deixe WeirdMethods = {"Método com espaços" (x) {return x + 1;}, [Method_name] (x) {return x + 2;}, [símbolo] (x) {return x + 3;}}; WeirdMethods ["Método com espaços"] (1) // => 2 WeirdMethods [Method_Name] (1) // => 3 WeirdMethods [símbolo] (1) // => 4 Usar um símbolo como nome de método não é tão estranho quanto parece. Ema fim de tornar um objeto iterável (para que possa ser usado com um para/deloop), você deve definir um método com o nome simbólicoSymbol.iterator, e há exemplos de fazer exatamente isso em Capítulo 12.6.10.6

Getters de propriedades e setters Todas as propriedades do objeto que discutimos até agora neste capítulo têm foram propriedades de dados com um nome e um valor comum. JavaScript também suporta propriedades de acessador, que não têm um único valor, mas Em vez disso, tenha um ou dois métodos de acessórios: um getter e/ou um setter.

Quando um programa consulta o valor de uma propriedade acessadora, JavaScriptInvoca o método getter (não transmitindo argumentos).O valor de retorno deEste método se torna o valor da expressão de acesso à propriedade.Quando um programa define o valor de uma propriedade acessadora, JavaScriptchama o método do setter, passando o valor do lado direito daatribuição.Este método é responsável por "cenário", em certo sentido,o valor da propriedade.O valor de retorno do método do setter é ignorado.Se uma propriedade tem um método getter e um setter, é uma leitura/gravaçãoopropriedade.Se possui apenas um método getter, é uma propriedade somente leitura.ESe possui apenas um método de setter, é uma propriedade somente de gravação (algo que não é possível com propriedades de dados) e tenta lê -lo sempreavaliar como indefinido.As propriedades do acessador podem ser definidas com uma extensão do objetoSintaxe literal (ao contrário das outras extensões ES6 que vimos aqui, Getterse os setters foram introduzidos no ES5):Seja o = {`// Uma propriedade de dados comumDataProp: Valor,`
`// Uma propriedade acessadora definida como um par de funções.obtenha o acessorProp () {return`
`this.DataProp;};SET ACDRESTORPROP (VALUE) {this.dataProp = value;};}As propriedades do acessador são definidas como um ou dois métodos cujo nome é o mesmo que o nome da propriedade.Estes parecem métodos comunsdefinido usando a taquigrafia ES6, exceto esse getter e setterAs definições são prefixadas com get ou set.(No ES6, você também pode usar`

Erro ao traduzir esta página.

exemplo.Javascript invoca essas funções como métodos do objeto em que estão definidos, o que significa que dentro do corpo da função, refere-se ao objeto de ponto p. Então, o método getter para a propriedade R pode se referir às propriedades X e Y como esta.x e this.y. Métodos e essa palavra-chave são abordados com mais detalhes no §8.2.2. As propriedades do acessador são herdadas, assim como as propriedades de dados, então você pode usar o objeto P definido acima como um protótipo para outros pontos. Você pode definir novos objetos suas próprias propriedades X e Y, e eles herdarão as propriedades R e Theta: Seja q = object.create(p); // um novo objeto que herda os getters e setters q.x = 3; q.Y = 4; // Crie propriedades de dados de Q q.r // => 5: o acessador herda as propriedades funcionam Q.Theta // => Math.atan2(4, 3) O código acima usa propriedades de acessador para definir uma API que fornece duas representações (coordenadas cartesianas e coordenadas polares) de um conjunto único de dados. Outros motivos para usar as propriedades do acessador incluem Verificação de sanidade da propriedade escrevendo e retornando valores diferentes em cada propriedade dizia: // Este objeto gera números de série crescentes estritamente const serialnum = { // Esta propriedade de dados contém o próximo número de série. // O _ no nome da propriedade sugere que é para somente uso interno. _n: 0, // retorna o valor atual e aumenta-lo obtemos a seguir () { return this._n++; } }

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Capítulo 7. MatrizesEste capítulo documenta as matrizes, um tipo de dados fundamental em javascripte na maioria das outras linguagens de programação.Uma matriz é uma ordemColeção de valores.Cada valor é chamado de elemento e cada elementotem uma posição numérica na matriz, conhecida como seu índice.JavaScriptMatrizes não são criadas: um elemento de matriz pode ser de qualquer tipo e diferenteElementos da mesma matriz podem ser de tipos diferentes.Elementos da matrizpode até ser objetos ou outras matrizes, o que permite criarestruturas de dados complexas, como matrizes de objetos e matrizes de matrizes.As matrizes JavaScript são baseadas em zero e usam índices de 32 bits: o índice deO primeiro elemento é 0 e o índice mais alto possível é 4294967294(2^{32}), para um tamanho máximo da matriz de 4.294.967.295 elementos.As matrizes de JavaScript são dinâmicas: elas crescem ou encolhem conforme necessário, eNão há necessidade de declarar um tamanho fixo para a matriz quando você a criaou realocá -lo quando o tamanho mudar.Matrizes de JavaScript podem serEsparsas: os elementos não precisam ter índices contíguos, e pode haverser lacunas.Cada matriz JavaScript possui uma propriedade de comprimento.Para não parseMatrizes, esta propriedade especifica o número de elementos na matriz.ParaMatrizes esparsas, o comprimento é maior que o índice mais alto de qualquer elemento.Matrizes de JavaScript são uma forma especializada de objeto JavaScript e ArrayOs índices são realmente pouco mais do que nomes de propriedades que sãointeiros.Falaremos mais sobre as especializações de matrizes em outros lugaresNeste capítulo.As implementações normalmente otimizam as matrizes para queO acesso a elementos de matriz numericamente indexados é geralmente significativamente³²

mais rápido que o acesso a propriedades regulares de objetos. Arrays herdam propriedades do Array.prototype, que define um rico conjunto de métodos de manipulação de matriz, cobertos em §7.8. A maioria destes métodos são genéricos, o que significa que eles funcionam corretamente não apenas para matrizes verdadeiras, mas para qualquer "objeto de matriz". Discutiremos como matrizObjetos em §7.9. Finalmente, as cordas JavaScript se comportam como matrizes de Personagens, e discutiremos isso no §7.10. O ES6 apresenta um conjunto de novas classes de matriz conhecidas coletivamente como ?digitadoMatrizes. Ao contrário das matrizes JavaScript regulares, as matrizes digitadas têm um fixo comprimento e um tipo de elemento numérico fixo. Eles oferecem alto desempenho e acesso no nível de byte a dados binários e são abordados no §11.2.7.1 Criando matrizes Existem várias maneiras de criar matrizes. As subseções a seguir Explique como criar matrizes com: Matriz literais O ... Spread Operator em um objeto iterável O construtor da matriz () Os métodos de fábrica do Array.Of () e Array.From () 7.1.1 Literais da matriz De longe, a maneira mais simples de criar uma matriz é com uma matriz literal, que é simplesmente uma lista separada por vírgula de elementos de matriz no quadrado Suportes. Por exemplo:

deixe vazio = [];// Uma matriz sem elementosSeja os primos = [2, 3, 5, 7, 11];// Uma matriz com 5 numéricoelementosSeja Misc = [1.1, verdadeiro, "A",];// 3 elementos de váriosTipos + vírgula à direitaOs valores em uma matriz literal não precisam ser constantes;eles podem serExpressões arbitrárias:deixe base = 1024;deixe tabela = [base, base+1, base+2, base+3];Literais da matriz podem conter literais de objetos ou outros literais da matriz:Seja b = [[1, {x: 1, y: 2}], [2, {x: 3, y: 4}]];Se uma matriz literal contiver várias vírgulas seguidas, sem valorEntre, a matriz é escassa (ver §7.3).Elementos de matriz para os quaisOs valores são omitidos não existem, mas parecem indefinidos se vocêConsulte -os:deixe count = [1, 3];// elementos nos índices 0 e 2. NãoElemento no índice 1deixe undefs = [,,];// uma matriz sem elementos, mas um comprimento de 2A sintaxe literal da matriz permite uma vírgula opcional, então [,,] tem umcomprimento de 2, não 3.7.1.2 O operador de propagaçãoNo ES6 e mais tarde, você pode usar o "operador de espalhamento" ..., para incluirOs elementos de uma matriz dentro de uma matriz literal:

Seja `a = [1, 2, 3];` Seja `b = [0, ... a, 4];` // `b == [0, 1, 2, 3, 4]` Os três pontos "espalham" a matriz A para que seus elementos se tornem elementos dentro da matriz literal que está sendo criada. É como se o... a foi substituído pelos elementos da matriz A, listados literalmente como parte da matriz fechada literal. (Observe que, embora nós os chamemos Três pontos um operador de espalhamento, este não é um verdadeiro operador porque pode ser usado apenas em literais de matriz e, como veremos mais adiante no livro, invocações da função.) O operador de spread é uma maneira conveniente de criar uma cópia (rasa) de uma matriz: Seja `original = [1, 2, 3];` Deixe copiar = [... original]; `cópia [0] = 0;` // modificar a cópia não altera o original original [0] // => 1 O operador de spread trabalha em qualquer objeto iterável. (Objetos iteráveis ?? são o que o loop for/de loop se destaca; Nós os vimos pela primeira vez no §5.4.4 e Veremos muito mais sobre eles no capítulo 12.) Strings são iteráveis, então Você pode usar um operador de spread para transformar qualquer string em uma matriz de strings de personagens: Deixe dígitos = [... "0123456789ABCDEF"]; `dígitos // => ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"]` Definir objetos (§11.1.1) são iteráveis, por isso é uma maneira fácil de remover duplicados de elementos de uma matriz é converter a matriz em um conjunto e depois

Converta imediatamente o conjunto em uma matriz usando o operador de espalhamento:Deixe letras = [... "Hello World"];[... novo conjunto (letras)] // => ["h", "e", "l", "o", "", "W", "R", "D"]7.1.3 O construtor Array ()Outra maneira de criar uma matriz é com o construtor Array ().VocêPode invocar este construtor de três maneiras distintas:Chame sem argumentos:deixe a = new Array ();Este método cria uma matriz vazia sem elementos e éequivalente à matriz literal [].Chame com um único argumento numérico, que especifica umcomprimento:deixe a = nova matriz (10);Esta técnica cria uma matriz com o comprimento especificado.EsseA forma do construtor Array () pode ser usada para pré -alocaruma matriz quando você souber com antecedênciaca quantos elementos vãoser necessário.Observe que nenhum valores é armazenado na matriz e oPropriedades do índice de matriz ?0?, ?1? e assim por diante nem são definidaspara a matriz.Especificarm explicitamente dois ou mais elementos de matriz ou um único nãoElemento numérico para a matriz:deixe a = nova matriz (5, 4, 3, 2, 1,

"Teste, teste"); Nesta forma, os argumentos do construtor se tornam os elementos da nova matriz. Usar uma matriz literal é quase sempre mais simples que esse uso do construtor da matriz () .7.1.4 Array.of () Quando a função do construtor da matriz () é invocada com um numérico argumento, ele usa esse argumento como um comprimento de matriz. Mas quando invoca com mais de um argumento numérico, trata esses argumentos como elementos para a matriz a ser criada. Isso significa que a matriz () Construtor não pode ser usado para criar uma matriz com um único numérico elemento. No ES6, a função da matriz.of () aborda esse problema: é um método de fábrica que cria e retorna uma nova matriz, usando seu argumento valores (independentemente de quantos deles existem) como a matrizElementos: Array.of () // => []; retorna uma matriz vazia sem argumentos Array.of (10) // => [10]; pode criar matrizes com um único argumento numérico Array.of (1,2,3) // => [1, 2, 3] 7.1.5 Array.From () Array.From é outro método de fábrica de matriz introduzido no ES6. Isto espera um objeto iterável ou semelhante a uma matriz como seu primeiro argumento e retorna uma nova matriz que contém os elementos desse objeto. Com um iterável Argumento, Array.From (iterable) funciona como o operador de espalhamento

[... iterable] Faz.Também é uma maneira simples de fazer uma cópia de umvariedade:Seja copy = Array.From (original);Array.from () também é importante porque define uma maneira de fazer umcópia de matriz verdadeira de um objeto semelhante a uma matriz.Objetos semelhantes a matrizes não sãoobjetos que têm uma propriedade de comprimento numérico e valores armazenados compropriedades cujos nomes são inteiros.Ao trabalhar comJavaScript do lado do cliente, os valores de retorno de alguns métodos do navegador da websão parecidos com a matriz e pode ser mais fácil trabalhar com eles se você primeiroconverte -os em verdadeiros matrizes:Deixe Truearray = Array.From (matriz);Array.From () também aceita um segundo argumento opcional.Se vocêPasse uma função como o segundo argumento, então como a nova matriz está sendoconstruído, cada elemento do objeto de origem será passado para ofunção que você especificar e o valor de retorno da função será armazenadona matriz em vez do valor original.(Isso é muito parecido com oMétodo de mapa de matriz () que será introduzido posteriormente no capítulo, masé mais eficiente para realizar o mapeamento enquanto a matriz está sendo construído que é para construir a matriz e depois mapeá -la para outra nova matriz.)7.2 Elementos de matriz de leitura e escrita\Você acessa um elemento de uma matriz usando o operador [].Uma referênciada matriz, deve aparecer à esquerda dos colchetes.Um arbitrárioexpressão que tem um valor inteiro não negativo deve estar dentro do

Suportes.Você pode usar esta sintaxe para ler e escrever o valor de um elemento de uma matriz.Assim, o seguinte são todos JavaScript legal declarações:deixe um = ["mundo"];// Comece com uma matriz de um elemento deixe o valor = a [0];// Leia o elemento 0a [1] = 3,14;// Escreva o elemento 1deixe i = 2;a [i] = 3;// Escreva elemento 2a [i + 1] = "Olá";// Escreva elemento 3a [a [i]] = a [0];// Leia os elementos 0 e 2, escrevaElemento 3O que é especial nas matrizes é que, quando você usa nomes de propriedades que são números inteiros não negativos inferiores a 2-1, a matriz automaticamenteMantém o valor da propriedade de comprimento para você.No anterior,Por exemplo, criamos uma matriz A com um único elemento.Nós entãovalores atribuídos nos índices 1, 2 e 3. A propriedade de comprimento doArray mudou como nós, então:A.Length // => 4Lembre -se de que as matrizes são um tipo especializado de objeto.O quadradoSuportes usados ??para acessar os elementos da matriz funcionam como o quadradoSuportes usados ??para acessar as propriedades do objeto.JavaScript converte o índice numérico de matriz que você especificar para uma string - o índice 1 se torna oString "1" - então usa essa string como nome de propriedade.Não há nadaEspecial sobre a conversão do índice de um número em uma string:Você também pode fazer isso com objetos regulares:Seja o = {};// Crie um objeto simpleso [1] = "um";// indexá-lo com um número inteiro32

o ["1"] // => "One"; nomes numéricos e de propriedades de strings são iguais É útil distinguir claramente um índice de matriz de um objeto nome da propriedade. Todos os índices são nomes de propriedades, mas apenas propriedade Os nomes inteiros entre 0 e 2-2 são índices. Todas as matrizes são Objetos, e você pode criar propriedades de qualquer nome neles. Se você usa Propriedades que são índices de matriz, no entanto, as matrizes têm o especial Comportamento de atualizar sua propriedade de comprimento, conforme necessário. Observe que você pode indexar uma matriz usando números negativos ou que não são inteiros. Quando você faz isso, o número é convertido em um String, e essa string é usada como o nome da propriedade. Já que o nome é Não é um número inteiro não negativo, é tratado como uma propriedade de objeto regular, não um índice de matriz. Além disso, se você indexar uma matriz com uma string que acontece com Seja um número inteiro não negativo, ele se comporta como um índice de matriz, não um objeto propriedade. O mesmo acontece se você usar um número de ponto flutuante que é o O mesmo que um número inteiro: a [-1.23] = true; // Isso cria uma propriedade chamada "-1,23" a ["1000"] = 0; // Este é o 1001º elemento da matriz a [1.000] = 1; // índice de matriz 1. O mesmo que a [1] = 1; O fato de os índices de matriz serem simplesmente um tipo especial de propriedade de objeto nome significa que as matrizes de javascript não têm noção de um ?fora de limites ?Erro. Quando você tenta consultar uma propriedade inexistente de qualquer Objeto, você não recebe um erro; Você simplesmente fica indefinido. Isso é justamente verdadeiro para matrizes quanto para objetos: deixe um = [verdadeiro, falso]; // Esta matriz tem elementos nos índices 0 e 132

a [2] // => indefinido;nenhum elemento nissoíndice.a [-1] // => indefinido;Sem propriedade com issonome.7.3 Matrizes esparsasUma matriz esparsa é aquela em que os elementos não têm contíguoíndices começando em 0. Normalmente, a propriedade de comprimento de uma matrizEspecifica o número de elementos na matriz.Se a matriz for escassa, oO valor da propriedade de comprimento é maior que o número de elementos.Matrizes esparsas podem ser criadas com o construtor Array () ou simplesmenteatribuindo a um índice de matriz maior que o comprimento da matriz atual.deixe a = nova matriz (5);// Sem elementos, mas A. o comprimento é 5.a = [];// Crie uma matriz sem elementos ecomprimento = 0.a [1000] = 0;// A atribuição adiciona um elemento, mas definecomprimento a 1001.Veremos mais tarde que você também pode fazer uma matriz escassa com oExcluir operador.Matrizes suficientemente escassos são tipicamente implementadas em ummaneira mais lenta e com eficiência de memória do que as densas matrizes, e olhandoelementos para cima em tal matriz levarão tanto tempo quanto regularPesquisa de propriedade do objeto.Observe que quando você omita um valor em uma matriz literal (usando repetidosvírgulas como em [1 , , 3]), a matriz resultante é escassa e o omitidoElementos simplesmente não existem:

Seja A1 = [,];// Esta matriz não tem elementos ecomprimento 1Seja A2 = [indefinido];// Esta matriz tem um indefinidoelemento0 em A1 // => false: A1 não tem elemento comÍNDICE 00 em A2 // => true: A2 tem o indefinidovalor no índice 0Entender matrizes esparsas é uma parte importante do entendimento doNatureza verdadeira das matrizes JavaScript.Na prática, no entanto, a maioria dos javascriptsMatrizes com quem você trabalhará não será escasso.E, se você tiver queTrabalhe com uma matriz esparsa, seu código provavelmente o tratará exatamente comotrataria uma matriz não par com elementos indefinidos.7.4 Comprimento da matrizCada matriz tem uma propriedade de comprimento, e é essa propriedade que fazMatrizes diferentes dos objetos JavaScript regulares.Para matrizes que sãoDenso (isto é, não escasso), a propriedade de comprimento especifica o número deelementos na matriz.Seu valor é mais do que o maior índice ema matriz:[] .Length // => 0: A matriz não tem elementos["A", "B", "C"]. Comprimento // => 3: O índice mais alto é 2, o comprimento é3Quando uma matriz é escassa, a propriedade de comprimento é maior que onúmero de elementos, e tudo o que podemos dizer sobre isso é que o comprimento éGarantido ser maior que o índice de todos os elementos da matriz.Ou,Em outras palavras, uma matriz (esparsa ou não) nunca terá um elementocujo índice é maior ou igual ao seu comprimento.A fim de

Erro ao traduzir esta página.

Erro ao traduzir esta página.

torna -se escasso.Como vimos acima, você também pode remover elementos do final de umMatriz simplesmente definindo a propriedade Length para o novo comprimento desejado.Finalmente, Splice () é o método de uso geral para inserção,excluir ou substituir elementos de matriz.Alterna a propriedade de comprimento e muda os elementos da matriz para índices mais altos ou mais baixos, conforme necessário.Ver§7.8 Para detalhes.7.6 Matrizes de iteraçãoA partir do ES6, a maneira mais fácil de percorrer cada um dos elementos de ummatriz (ou qualquer objeto iterável) é com o loop for/decoberto em detalhes em §5.4.4:Deixe letras = [... "Hello World"];// Uma variedade de letrasdeixe string = "";para (deixe a carta das letras) {string += letra;}String // => "Hello World";Remontizamos o texto originalO iterador de matriz embutido que o loop for/of usa retorna oelementos de uma matriz em ordem ascendente.Não tem comportamento especial paramatrizes esparsas e simplesmente retorna indefinidas para qualquer elementos de matrizque não existem.Se você deseja usar um loop for/de uma matriz e precisa saber oíndice de cada elemento da matriz, use o método de entradas () da matriz,

Juntamente com a tarefa de destruição, assim:deixe todos outros = "";para (vamos [índice, carta] de letters.entries ()) {if (índice % 2 === 0) todos os outros += letra;// cartas ematé índices}todos os outros // => "hlowrd"Outra boa maneira de iterar as matrizes é com a foreach ().Isso não é umnova forma do loop for, mas um método de matriz que oferece um funcionalabordagem para a iteração de matriz.Você passa uma função para o foreach ()Método de uma matriz, e foreach () invoca sua função uma vezcada elemento da matriz:Seja uppercase = "";letters.foreach (letra => {// Nota Sintaxe da função de setaaquiuppercase += letter.TouPercase ();});uppercase // => "Hello World"Como seria de esperar, foreach () itera a matriz em ordem, e elaNa verdade, passa o índice de matriz para sua função como um segundo argumento,o que é ocasionalmente útil.Ao contrário do loop for/of, oforeach () está ciente das matrizes esparsas e não invoca o seufunção para elementos que não estão lá.§7.8.1 documenta o método foreach () com mais detalhes.Essa seçãotambém abrange métodos relacionados, como map () e filtro () queRealize tipos especializados de iteração de matriz.

Você também pode fazer uma pancada nos elementos de uma matriz com um bom velho Modeld para Loop
(&5.4.3):Let vogais = "";para (vamos i = 0; i <letters.length; i ++) { // para cada índice na matrizDeixe a letra = letras [i];// Obtenha o elementoNesse índicese (/eiou].test(Letter)) { // use um regularteste de expressãovogais += letra; // se for umvogal, lembre -se disso}vogais // => "EOO"Em loops aninhados, ou outros contextos em que o desempenho é crítico, vocêàs vezes pode ver esse loop básico de iteração de matriz escrito para que o O comprimento da matriz é procurado apenas uma vez e não em cada iteração.Ambosdos seguintes formas de loop são idiomáticas, embora não particularmenteComum, e com intérpretes javascript modernos, não é de todo claroque eles têm algum impacto de desempenho:// salve o comprimento da matriz em uma variável localpara (vamos i = 0, len = letters.length; i <len; i ++) { // O corpo do loop permanece o mesmo}// itera para trás desde o final da matriz até o iníciopara (vamos i = letters.length-1; i> = 0; i--) { // O corpo do loop permanece o mesmo}Esses exemplos assumem que a matriz é densa e que todos os elementoscontêm dados válidos.Se não for esse o caso, você deve testar a matrizelementos antes de usá -los.Se você quiser pular indefinido e

Elementos inexistentes, você pode escrever: para (vamos i = 0; i < A.Length; i++) {if (a [i] === indefinido) continuar;// Saltar indefinido +elementos inexistentes// Corpo de loop aqui} 7.7 Matrizes multidimensionais JavaScript não suporta verdadeiras matrizes multidimensionais, mas você pode Aproxime -os com matrizes de matrizes. Para acessar um valor em uma matriz de matrizes, basta usar o operador [] duas vezes. Por exemplo, suponha que a matriz variável é uma matriz de matrizes de números. Cada elemento emMatrix [x] é uma variedade de números. Para acessar um número específico dentro dessa matriz, você escreveria matriz [x] [y]. Aqui está um exemplo concreto que usa uma matriz bidimensional como uma multiplicação: // Crie uma matriz multidimensional deixa tabela = nova matriz (10); // 10 linhas domes para (vamos i = 0; i < tabela.length; i++) {tabela [i] = nova matriz (10); // Cada linha tem 10 colunas} // inicialize a matriz for (let linha = 0; linha < tabela.length; row++) {for (let col = 0; col < tabela [linha] .Length; col++) {tabela [linha] [col] = linha * col;}} // Use a matriz multidimensional para calcular 5*7 Tabela [5] [7] // => 35

Erro ao traduzir esta página.

7.8.1 Métodos de iterador de matrizOs métodos descritos nesta seção iteram sobre as matrizes passando elementos de matriz, para uma função que você fornece, e eles fornecem maneiras convenientes de iterar, mapear, filtrar, testar e reduzir as matrizes.Antes de explicarmos os métodos em detalhes, no entanto, vale a pena fazer algumas generalizações sobre eles.Primeiro, todos esses métodos aceitam um funcionar como seu primeiro argumento e invocar essa função uma vez para cada elemento (ou alguns elementos) da matriz.Se a matriz for escassa, a função que você passa não é invocada para elementos inexistentes.Na maioria das vezes, a função que você fornece é invocada com três argumentos: o valor do elemento da matriz, o índice do elemento da matriz e a matriz em si.Freqüentemente, você só precisa do primeiro desses valores de argumento e pode ignorar o segundo e o terceiro valores.A maioria dos métodos iteradores descritos nas seguintes subseções Aceite um segundo argumento opcional.Se especificado, a função é invocada como se fosse um método deste segundo argumento.Isto é, o segundo argumento que você passa se torna o valor dessa palavra-chave dentro da função que você passa como o primeiro argumento.O valor de retorno da função que você passa geralmente é importante, mas métodos diferentes lidam com o valor de retorno de maneiras diferentes.Nenhum dos métodos descritos aqui modifica a matriz em que eles são invocados (embora a função que você passa possa modificar a matriz, é claro).Cada uma dessas funções é invocada com uma função como seu primeiro argumento, e é muito comum definir essa função embutida como parte da expressão de invocação de método em vez de usar uma função existente que

é definido em outro lugar.Função de seta Sintaxe (consulte §8.1.3) obrasparticularmente bem com esses métodos, e nós o usaremos nos exemplosso a seguir.Foreach ()O método foreach () itera através de uma matriz, invocando umfunção que você especifica para cada elemento.Como descrevemos, você passaa função como o primeiro argumento para foreach ().foreach () entãoInvoca sua função com três argumentos: o valor da matrizelemento, o índice do elemento da matriz e a própria matriz.Se você apenasse preocupar com o valor do elemento da matriz, você pode escrever uma função comApenas um parâmetro - os argumentos adicionais serão ignorados:deixe dados = [1,2,3,4,5], soma = 0;// calcular a soma dos elementos da matrizdata.foreach (value => {sum += value;});// soma ==15// agora incrementa cada elemento da matrizdata.foreach (function (v, i, a) {a [i] = v + 1;});// dados ==[2,3,4,5,6]Observe que a foreach () não fornece uma maneira de encerrar a iteraçãoAntes de todos os elementos serem passados ??para a função.Isto é, existeNenhum equivalente à declaração de quebra que você pode usar com um regular paralaço.MAPA()O método map () passa cada elemento da matriz em que éinvocado para a função que você especificar e retorna uma matriz contendo ovalores retornados por sua função.Por exemplo:

Seja `a = [1, 2, 3];a.map (x => x*x) // => [1, 4, 9]`: a função leva a entrada `x` e retorna `x*x`. A função que você passa para `map()` é invocada da mesma maneira que uma função passada para `foreach()`. Para o método `map()`, no entanto, a função que você passa deve retornar um valor. Observe que o `map()` retorna uma nova matriz: ele não modifica a matriz em que está invocada. Se essa matriz for esparsa, sua função não será chamada para os elementos ausentes, mas a matriz retornada será escassamente da mesma maneira que a matriz original: Ela terá o mesmo comprimento e os mesmos elementos ausentes.

O método `filter()` retorna uma matriz contendo um subconjunto de elementos da matriz em que é invocado. A função que você passa deve ser predicable: uma função que retorna verdadeiro ou falso. O predicable é chamado como para `foreach()` e `map()`. Se o retorno do valor é verdadeiro, ou um valor que se converte para verdadeiro, então o elemento é passado para o predicable: um membro do subconjunto e é adicionado à matriz que se tornará o valor de retorno. Exemplos:

Seja `a = [5, 4, 3, 2, 1];A.Filter (x => x < 3) // => [2, 1]`; valores menores que 3.

`a.Filter ((x, i) => i%2 === 0) // => [5, 3, 1]`; todos os outros valores.

Observe que o `filter()` pula os elementos ausentes em matrizes esparsas e que seu valor de retorno é sempre denso. Para fechar as lacunas em uma matriz esparsa, você pode fazer isso:

Seja `dense = Sparse.Filter(() => true)`;

Erro ao traduzir esta página.

Todos os elementos da matriz:Seja a = [1,2,3,4,5];a.every (x => x <10) // => true: Todos os valores são <10.a.avery (x => x % 2 === 0) // => false: nem todos os valores sãoaté.O método de alguns () é como o quantificador matemático "existe"?: ele retorna verdadeiro se houver pelo menos um elemento na matriz paraque o predicado retorna verdadeiro e retorna false se e somente se oRetornos predicados falsos para todos os elementos da matriz:Seja a = [1,2,3,4,5];A., (x => x%2 === 0) // => true;A tem alguns números uniformes.A. Alguma (isnan) // => false;A não tem não números.Observe que todos () e alguns () param de iterar os elementos da matriz comoAssim que eles sabem que valor retornar.alguns () retorna verdadeiro oPrimeira vez que seu predicado retorna <code> true </code> e apenas iteraDurante toda a matriz, se o seu predicado sempre retornar FALSE.todo () é o oposto: retorna falsa na primeira vez que vocêRetornos predicados falsos e apenas itera todos os elementos se o seuO predicado sempre retorna verdadeiro.Observe também que, por matemáticaConvenção, todo () retorna verdadeiro e alguns retornam falsos quandoinvocado em uma matriz vazia.Reduce () e ReduceRight ()Os métodos ReduCer () e ReduceRight () combinam oelementos de uma matriz, usando a função que você especifica, para produzir umvalor único.Esta é uma operação comum na programação funcional

e também passa pelos nomes "injetar" e "dobrar". Exemplos de ajudar a ilustrar como funciona: Seja $a = [1, 2, 3, 4, 5]$; $a.$ Reduce $((x, y) \Rightarrow x+y, 0) // \Rightarrow 15$; a soma dos valores da lista. $a.$ Reduce $((x, y) \Rightarrow x*y, 1) // \Rightarrow 120$; o produto dos valores da lista. $a.$ Reduce $((x, y) \Rightarrow (x > y) ? X : y) // \Rightarrow 5$; o maior dos valores. $a.$ Reduce () leva dois argumentos. O primeiro é a função que executa a operação de redução. A tarefa desta função de redução é de alguma forma combinar ou reduzir dois valores em um único valor e parar de retornar esse valor reduzido. Nos exemplos que mostramos aqui, as funções combinam dois valores adicionando-os, multiplicando-os e escolhendo o maior. O segundo argumento (opcional) é um valor inicial para passar para a função. As funções usadas com $a.$ Reduce () são diferentes das funções usadas com $a.$ foreach () e $a.$ map (). O valor familiar, índice e matriz de valores são passados assim como os segundo, terceiro e quarto argumentos. O argumento é o resultado acumulado da redução até agora. No primeiro chamado para a função, este primeiro argumento é o valor inicial que você passou como segundo argumento à função $a.$ reduce(). Nas chamadas subsequentes, é o valor retornado pela invocação anterior da função. No exemplo, a função de redução é chamada com os argumentos 0 e 1. Ele adiciona 1 ao 0 e retorna 1. É então chamado novamente com os argumentos 2 e 1 e retorna 3. Em seguida, ele calcula $3+3 = 6$, depois $6+4 = 10$ e finalmente $10+5 = 15$. Este valor final, 15, se torna o valor de retorno de $a.$ reduce().

Você deve ter notado que a terceira chamada para reduzir () neste exemploTem apenas um argumento: não há valor inicial especificado.Quandovocê chama Reduce () como este sem valor inicial, ele usa o primeiroelemento da matriz como o valor inicial.Issso significa que a primeira chamada paraa função de redução terá o primeiro e o segundo elementos de matriz comoseus primeiros e segundos argumentos.Nos exemplos de soma e produto, nospoderia ter omitido o argumento do valor inicial.Chamando Reduce () em uma matriz vazia sem argumento de valor inicialCausa um TypeError.Se você ligar com apenas um valor - uma matrizcom um elemento e nenhum valor inicial ou uma matriz vazia e um inicialvalor - simplesmente retorna esse valor sem nunca chamar ofunção de redução.ReduceRight () funciona como Reduce (), exceto queprocessa a matriz do índice mais alto para o mais baixo (da direita para a esquerda), mas simdo que do mais baixo ao mais alto.Você pode querer fazer isso se a reduçõoA operação tem associatividade da direita para a esquerda, por exemplo:// calcular $2^{(3^4)}$.A exponenciação tem o direito para a esquerdaprecedênciaseja a = [2, 3, 4];A. Reduteright ((ACC, Val) => Math.pow (val, acc)) // =>2.4178516392292583E+24Observe que nem reduz () nem reduteright () aceita umargumento opcional que especifica esse valor em que oA função de redução deve ser invocada.O argumento de valor inicial opcionaltorna seu lugar.Consulte o método function.bind () (§8.7.5) se vocêPrecisa da sua função de redução invocada como um método de um determinado

objeto. Os exemplos mostrados até agora foram numéricos por simplicidade, mas redução () e reduceRight () não se destinam apenas a cálculos matemáticos. Qualquer função que possa combinar dois valores (como dois objetos) em um valor do mesmo tipo pode ser usado como uma função de redução. Por outro lado, algoritmos expressos usando Reduções de matrizes podem rapidamente se tornar complexas e difíceis de entender. E você pode achar que é mais fácil de ler, escrever e raciocinar sobre o seu Código se você usar construções regulares de loops para processar suas matrizes.

7.8.2 Matrizes achatadas com plano () e plangmap ()

No ES2019, o método Flat () cria e retorna uma nova matriz que contém os mesmos elementos que a matriz é chamada, exceto que qualquer elemento que são próprios matrizes são "achatados" no retorno variedade. Por exemplo:

```
[1, [2, 3]]. flat () // => [1, 2, 3][1, [2, [3]]]. Quando chamado sem argumentos, plana () divide um nível de ninho. Elementos da matriz original que são próprios são achatados, Mas os elementos da matriz dessas matrizes não são achatados. Se você quiser Apoie mais níveis, passe um número para plana (): Seja a = [1, [2, [3, [4]]]]; a.flat (1) // => [1, 2, [3, [4]]]a.flat (2) // => [1, 2, 3, [4]]a.flat (3) // => [1, 2, 3, 4]a.flat (4) // => [1, 2, 3, 4]
```

O método flatmap () funciona como o método map () (ver "Map ()"), exceto que a matriz retornada é achata automaticamente como se passou para o plano (). Isto é, chamar A.flatmap (f) é o mesmo que (mas mais eficiente que) a.map (f). flat (): Deixe frases = ["Hello World", "The Definitive Guide"]; deixe palavras = phrases.flatmap (frase => phrase.split ("")); palavras // => ["Olá", "mundo", "o", "definitivo", "guia"]; Você pode pensar em Flatmap () como uma generalização de mapa () que permite cada elemento da matriz de entrada para mapear para qualquer número de elementos da matriz de saída. Em particular, Flatmap () permite que você mapear elementos de entrada para uma matriz vazia, que se acha para nada na matriz de saída: // mapear números não negativos para suas raízes quadradas [-2, -1, 1, 2] .flatmap (x => x <0? []: Math.sqrt (x)) // => [1, 2 ** 0,5] 7.8.3 Adicionando matrizes com concat () O método concat () cria e retorna uma nova matriz que contém os elementos da matriz original em que concat () foi invocado, seguido por cada um dos argumentos para concat (). Se algum deles argumentos é uma matriz, então são os elementos da matriz que são concatenados, não a própria matriz. Observe, no entanto, que concat () faz Não achar recursivamente matrizes de matrizes. concat () não modifica a matriz em que é invocada: Seja a = [1,2,3]; A.Concat (4, 5) // => [1,2,3,4,5] a.Concat ([4,5], [6,7]) // => [1,2,3,4,5,6,7]; Matrizes são

achatadoA.Concat (4, [5, [6,7]]) // => [1,2,3,4,5, [6,7]];mas não matriz aninhada sA matriz original é não modificada Observe que concat () faz uma nova cópia da matriz que ela é chamada. Em muitos casos, essa é a coisa certa a fazer, mas é um caro operação. Se você se encontrar escrevendo código como A = A.Concat (x), então você deve pensar em modificar sua matriz no lugar com push () ou splice () em vez de criar um novo. 7.8.4 pilhas e filas com push (), pop (), shift (), e não dividido () Os métodos push () e pop () permitem trabalhar com matrizes como se elas eram pilhas. O método push () anexa um ou mais novos elementos até o final de uma matriz e retorna o novo comprimento da matriz. Ao contrário do concat (), push () não aceita os argumentos da matriz. O método pop () faz o reverso: ele exclui o último elemento de uma matriz, diminui o comprimento da matriz e retorna o valor que ele removeu. Observação que ambos os métodos modificam a matriz no lugar. A combinação de push () e pop () permite que você use uma matriz JavaScript para implementar uma pilha de primeira entrada e saída. Por exemplo: deixe pilha = [] // Stack == [] Stack.push (1,2) // pilha == [1,2] Stack.pop () // pilha == [1]; Retorna 2 Stack.push (3) // Stack == [1,3] Stack.pop () // pilha == [1]; retorna 3 Stack.push ([4,5]) // Stack == [1, [4,5]] Stack.pop () // Stack == [1]; Retorna [4,5] Stack.pop () // pilha == []; retorna 1

O método push () não acha uma matriz que você passa para ele, mas se você quiser empurrar todos os elementos de uma matriz para outra matriz, você pode usar o operador de spread (§8.3.4) para achatá-lo explicitamente: a.push (... valores); Os métodos não -definidos () e shift () se comportam como push () e pop (), exceto que eles inserem e removem elementos do início de uma matriz e não do final. NETNIFF () adiciona um elemento ou elementos para o início da matriz, muda o existente elementos de matriz até índices mais altos para abrir espaço e retorna o novo comprimento da matriz. Shift () remove e retorna o primeiro elemento de uma matriz, mudando todos os elementos subsequentes para baixo para ocupar o espaço recém -vago no início da matriz. Você poderia usar o não definido () e shift () para implementar uma pilha, mas seria menos eficiente do que usar push () e pop () porque os elementos da matriz precisam ser deslocados para cima ou para baixo toda vez que um elemento é adicionado ou removido no início da matriz. Em vez disso, porém, você pode implementar uma estrutura de dados da fila usando push () para adicionar elementos no final de umArray e Shift () para removê-los do início da matriz: Seja q = [] // q == [] q.push (1,2) // q == [1,2] q.shift () // q == [2]; retorna 1 q.push (3) // q == [2,3] q.shift () // q == [3]; Retorna 2 q.shift () // q == []; retorna 3 Há uma característica do não -definido () que vale a pena chamar porque Você pode achar surpreendente. Ao passar vários argumentos para

deserto (), eles são inseridos de uma só vez, o que significa que eles terminamna matriz em uma ordem diferente da que seria se você inserissees um de cada vez:deixe A = [];// a == []A.UnShift (1) // A == [1]A.UnShift (2) // A == [2, 1]a = [];// a == []A.UnShift (1,2) // A == [1, 2]7.8.5 subarrays with slice (), splice (), preench () eCopyWithin ()Matrizes definem vários métodos que funcionam em regiões contíguas, ousubarrays ou "fatias" de uma matriz.As seções a seguir descrevemMétodos para extrair, substituir, preencher e copiar fatias.FATIAR()O método slice () retorna uma fatia, ou subarray, do especificadovariedade.Seus dois argumentos especificam o início e o final da fatia para serretornou.A matriz retornada contém o elemento especificado pelo primeiroargumento e todos os elementos subsequentes até, mas não incluindo, oelemento especificado pelo segundo argumento.Se apenas um argumento forEspecificado, a matriz retornada contém todos os elementos desde o inícioposição até o final da matriz.Se um argumento for negativo, éEspecifica um elemento de matriz em relação ao comprimento da matriz.Umargumento de ?1, por exemplo, especifica o último elemento na matriz eUm argumento de ?2 especifica o elemento antes disso.Observe queSlice () não modifica a matriz em que é invocada.Aqui estãoAlguns exemplos:

Seja a = [1,2,3,4,5];A.Slice (0,3);// retorna [1,2,3]A.Slice (3);// retorna [4,5]a.slice (1, -1);// retorna [2,3,4]a.slice (-3, -2);// retorna [3]Emenda ()Splice () é um método de uso geral para inserir ou removerelementos de uma matriz.Ao contrário do slice () e concat (),Splice () modifica a matriz em que é invocada.Observe queSplice () e Slice () têm nomes muito semelhantes, mas executamoperações substancialmente diferentes.Splice () pode excluir elementos de uma matriz, inserir novos elementosem uma matriz ou execute as duas operações ao mesmo tempo.Elementos dea matriz que vem após o ponto de inserção ou exclusão tem seus índices aumentaram ou diminuíram conforme necessário para que permaneçamcontíguo com o restante da matriz.O primeiro argumento a Splice ()especifica a posição da matriz na qual a inserção e/ou exclusão é paracomeçar.O segundo argumento especifica o número de elementos quedeve ser excluído de (emendado) da matriz.(Observe que isso éOutra diferença entre esses dois métodos.O segundo argumentoPara Slice () é uma posição final.O segundo argumento para Splice () éum comprimento.) Se este segundo argumento for omitido, todos os elementos da matriz deO elemento de partida para o final da matriz é removido.emenda ()Retorna uma matriz dos elementos excluídos, ou uma matriz vazia se nãoOs elementos foram excluídos.Por exemplo:Que a = [1,2,3,4,5,6,7,8];a.splice (4) // => [5,6,7,8];A agora é [1,2,3,4]

Erro ao traduzir esta página.

ÍNDICpreenchido.Se esse argumento for omitido, a matriz será preenchida desde o inícioíndice até o fim.Você pode especificar índices em relação ao final doArray passando números negativos, assim como você pode para Slice ()`.CopyWithin ()`copyWithin () copia uma fatia de uma matriz para uma nova posição dentroa matriz.Ele modifica a matriz no lugar e retorna a matriz modificada,Mas não mudará o comprimento da matriz.O primeiro argumentoEspecifica o índice de destino para o qual o primeiro elemento serácopiado.O segundo argumento especifica o índice do primeiro elemento paraser copiado.Se esse segundo argumento for omitido, 0 será usado.O terceiroO argumento especifica o fim da fatia dos elementos a serem copiados.Seomitido, o comprimento da matriz é usado.Elementos do índice de inícioAté, entre outros, o índice final será copiado.Você pode especificaríndices em relação ao final da matriz passando números negativos,Assim como você pode para Slice ():Seja a = [1,2,3,4,5];A.`Copywithin (1) // => [1,1,2,3,4]`: Copiar elementos da matrizup uma.`Copywithin (2, 3, 5) // => [1,1,3,4,4]`: Copie os últimos 2 elementospara indexado 2a.`Copywithin (0, -2) // => [4,4,3,4,4]`: compensações negativastrabalho também`copyWithin ()` é destinado a um método de alto desempenho que éparticularmente útil com matrizes digitadas (consulte §11.2).É modelado apósofunção `memmove ()` da biblioteca padrão C.Observe que a cópiafuncionará corretamente, mesmo que haja sobreposição entre a fonte e

regiões de destino.7.8.6 Métodos de pesquisa e classificação de matrizesMatrizes implementar indexof (), lastIndexOf () e inclui () métodos semelhantes aos métodos de mesmo nome decordas.Também existem métodos de classificação () e reverso () parareordenando os elementos de uma matriz.Esses métodos são descritos nosubseções a seguir.IndexOf () e LastIndexOf ()IndexOf () e LastIndexOf () pesquisam uma matriz por um elemento com um valor especificado e retorne o índice do primeiro elemento desse tipoencontrado, ou -1 se nenhum for encontrado.indexOf () pesquisa a matriz deComeçando ao fim, e LastIndexOf () pesquisas de ponta a começo:Seja a = [0,1,2,1,0];A.IndexOf (1) // => 1: a [1] é 1A.LastIndexOf (1) // => 3: a [3] é 1A.IndexOf (3) // => -1: nenhum elemento tem valor 3indexOf () e lastIndexOf () compararam seu argumento com oelementos de matriz usando o equivalente ao operador ===.Se sua matrizcontém objetos em vez de valores primitivos, esses métodos verificam para verSe duas referências se referem exatamente ao mesmo objeto.Se você quiserNa verdade, observe o conteúdo de um objeto, tente usar o método find ()com sua própria função de predicado personalizado.indexof () e lastIndexOf () tomam um segundo opcional

argumento que especifica o índice de matriz no qual iniciar a pesquisa. Se este argumento é omitido, `indexOf()` começa no início e `LastIndexOF()` começa no final. Valores negativos são permitidos para o segundo argumento e são tratados como um deslocamento do final do array, como são para o método `slice()`: um valor de `?1`, por exemplo, especifica o último elemento da matriz. A função a seguir procura uma matriz por um valor especificado e retorna uma matriz de todos os índices correspondentes. Isso demonstra como o segundo argumento para `indexOf()` pode ser usado para encontrar correspondências além do primeiro.// Encontra todas as ocorrências de um valor x em uma matriz A e retorna uma matriz de índices correspondentes função `findall(a, x)` { Deixe os resultados = [], // a matriz de índices Voltaremos Len = A.Length, // o comprimento da matriz para ser pesquisado pos = 0; // a posição para pesquisar while (pos < len) { // enquanto mais elementos para procurar... pos = a.IndexOf(x, pos); // Procurar if (pos === -1) quebra; // Se nada encontrado, estamos feito. resultados.push(POS); // Caso contrário, armazene o índice em variável de pos = pos + 1; // e inicie a próxima pesquisa em próximo elemento } } RETORNO DE RECURSOS; // Retornar a matriz de índices } Observe que as strings possuem métodos `indexOF()` e `LastIndexOF()`

que funcionam como esses métodos de matriz, exceto que um segundo negativoO argumento é tratado como zero.Inclui ()O ES2016 inclui () o método leva um único argumento eRetorna true se a matriz contiver esse valor ou false caso contrário.IstoNão informa o índice do valor, apenas se ele existe.OInclui () o método é efetivamente um teste de associação definido para matrizes.Observe, no entanto, que as matrizes não são uma representação eficiente para conjuntos,E se você estiver trabalhando com mais de alguns elementos, deve usarum objeto definido real (§11.1.1).O método inclui () é ligeiramente diferente do indexOF ()método de uma maneira importante.indexOf () testa a igualdade usando omesmo algoritmo que o operador ===considera o valor não-número de um número diferente de todos os outrosvalor, incluindo a si mesmo.inclui () usa uma versão ligeiramente diferente de igualdade que considera Nan igual a si mesmo.Isso significa issoindexOf () não detectará o valor da nan em uma matriz, masinclui () Will:Seja a = [1, verdadeiro, 3, nan];a.includes (verdadeiro) // => truea.includes (2) // => falsea.includes (nan) // => trueA.IndexOF (NAN) // => -1;Indexof não consegue encontrar nanORGANIZAR()Sort () classifica os elementos de uma matriz no lugar e retorna o classificado

variedade. Quando o Sort () é chamado sem argumentos, ele classifica a matriz elementos em ordem alfabética (convertendo temporariamente em stringsPara executar a comparação, se necessário): deixe um = ["banana", "cereja", "maçã"]; a.sort ()// a == ["maçã", "banana", "cereja"] Se uma matriz contiver elementos indefinidos, eles serão classificados até o final de a matriz. Para classificar uma matriz em alguma ordem que não seja alfabética, você deve passar uma função de comparação como argumento a classificar (). Esta função decide qual de seus dois argumentos deve aparecer primeiro no classificado variedade. Se o primeiro argumento deve aparecer antes do segundo, a função de comparação deve retornar um número menor que zero. Se o primeiro argumento deve aparecer depois do segundo na matriz classificada, a função deve retornar um número maior que zero. E se os dois valores são equivalentes (ou seja, se a ordem deles for irrelevante), a comparação função deve retornar 0. Então, por exemplo, para classificar elementos de matriz em ordem numérica e não alfabética, você pode fazer isso: Seja a = [33, 4, 1111, 222]; a.sort ()// a == [1111, 222, 33, 4]; ordem alfabética a.a.sort (função (a, b) { // passa uma função comparador retornar a-b; // retorna <0, 0 ou> 0, dependendo em ordem}); // a == [4, 33, 222, 1111]; numérico ordena. sort ((a, b) => b-a); // a == [1111, 222, 33, 4]; reverte ordem numérica Como outro exemplo de classificação de itens de matriz, você pode executar um caso-

tipo alfabético insensível em uma variedade de cordas passando por um função de comparação que converte seus dois argumentos em minúsculas(com o método tolowercase ()) antes de compará -los:deixe um = ["formiga", "bug", "gato", "cachorro"];a.sort ()// a == ["bug", "cachorro", "formiga", "gato"];caso-tipo sensível.a.sort (função (s, t) {deixe a = s.toLowerCase ();Seja B = T.toLowerCase ();if (a <b) retornar -1;se (a> b) retornar 1;retornar 0;});// a == ["Ant", "Bug", "Cat", "Dog"];Isensível ao casoorganizarREVERTER()O método reverso () reverte a ordem dos elementos de um matriz e retorna a matriz invertida.Faz isso no lugar;em outropalavras, ele não cria uma nova matriz com os elementos reorganizados, masEm vez disso, os reorganiza na matriz já existente:Seja a = [1,2,3];a. reverse ()// a == [3,2,1]7.8.7 Array para conversões de stringA classe da matriz define três métodos que podem converter matrizes paraStrings, que geralmente é algo que você pode fazer ao criar loge mensagens de erro.(Se você deseja salvar o conteúdo de uma matriz emforma textual para reutilização posterior, serialize a matriz comJson.stringify () [§6.8] em vez de usar os métodos descritos aqui.)

O método junção () converte todos os elementos de uma matriz em stringse concatena -os, retornando a string resultante.Você pode especificaruma sequênciacomparada que separa os elementos na sequênciaseja a = [1, 2, 3];A.Join () // => "1,2,3"A.Join ("") // => "1 2 3"A.Join ("") // => "123"Seja b = nova matriz (10);// Uma variedade de comprimento 10 com nãoelementosB.Join ("") // => "-----": uma sequênciacomparada de 9hífensO método junção () é o inverso da string.split ()Método, que cria uma matriz quebrando uma corda em pedaços.Matrizes, como todos os objetos JavaScript, possuem um método ToString ().ParaUma matriz, esse método funciona como o método junção () semArgumentos:[1,2,3] .ToString () // => "1,2,3"["A", "B", "C"]].ToString () // => "A, B, C"[1, [2, "C"]].ToString () // => "1,2, C"Observe que a saída não inclui colchetes ou qualquer outro tipode delimitador em torno do valor da matriz.toLocalestring () é a versão localizada do tostring ().Istoconverte cada elemento da matriz em uma string chamando oTOCALESTRING () Método do elemento, e entãoconcatena as seqüências resultantes usando um local específico (e

String de separador definida por implementação).7.8.8 Funções de matriz estáticaAlém dos métodos de matriz que já documentamos, a matrizA classe também define três funções estáticas que você pode invocar através doConstrutor de matrizes em vez de matrizes.Array.of () eArray.From () são métodos de fábrica para criar novas matrizes.Eles foram documentados em §7.1.4 e §7.1.5.A outra função de matriz estática é Array.isArray (), que é útil para determinar se um valor desconhecido é uma matriz ou não:`Array.isArray ([]) // => trueArray.isArray ({}) // => false`7.9 Objetos semelhantes a matrizComo vimos, as matrizes de javascript têm alguns recursos especiais que outros objetos não têm:A propriedade de comprimento é atualizada automaticamente como novaOs elementos são adicionados à lista.A configuração do comprimento para um valor menor trunca a matriz.Matrizes herdam métodos úteis do Array.prototype.Array.isArray () retorna true para matrizes.Estes são os recursos que tornam as matrizes JavaScript distintas de objetos.Mas eles não são os recursos essenciais que definem uma matriz.Isso é muitas vezes perfeitamente razoável para tratar qualquer objeto com um comprimento numérico

propriedade e propriedades inteiras não negativas correspondentes como uma espécie de variedade. Esses objetos "parecidos com a matriz" realmente aparecem ocasionalmente na prática, e embora você não possa invocar diretamente os métodos de matriz neles ou espere comportamento especial da propriedade de comprimento, você ainda pode iterar através deles com o mesmo código que você usaria para uma matriz verdadeira. Acontece que muitos algoritmos de matriz funcionam tão bem com objetos semelhantes a matrizes quanto eles fazem com matrizes reais. Isto é especialmente verdade se seus algoritmos tratam a matriz somente como leitura ou se eles pelo menos deixam o comprimento da matriz alterado. O código a seguir leva um objeto regular, adiciona propriedades para torná-lo um objeto semelhante a uma matriz e depois itera através dos "elementos" do pseudo-grays resultante:

```
deixe A = {};// Comece com um objeto vazio regular// Adicione propriedades para torná-lo "parecido com a matriz"
deixe i = 0; enquanto (i < 10) {a [i] = i * i; i ++;}A.Length = i;// agora itera através dele como se fosse uma matriz real
Deixe total = 0; para (vamos j = 0; j < A.Length; j++) {total += a [j];}
```

No JavaScript do lado do cliente, vários métodos para trabalhar com

Documentos HTML (como Document.querySelectorAll (), por exemplo) retorna objetos semelhantes a matrizes. Aqui está uma função que você pode usar para testar objetos que funcionam como matrizes:// Determine se o é um objeto semelhante a uma matriz.// Strings e funções têm propriedades numéricas de comprimento, mas não// excluído pelo teste TIPOOF. No JavaScript do lado do cliente, DOM Texto// os nós têm uma propriedade de comprimento numérico e pode precisar ser excluído// com um teste O.nodeType == 3. função isarraylike (o) {se (o && // o não for nulo, indefinido, etc. typeof o === "objeto" && / o é um objeto Número. isfinite (O.Length) && // O.Length é um número finito O.Length > = 0 && // O.Length não é não-negativo Número. isinteger (O.Length) && / O.Length é um inteiro O.Length <4294967295) { // O.Length <2^32 -1 retornar true; // então O é a matriz-como.} outro {retornar falso; // Caso contrário, é não. } } Veremos em uma seção posterior que as cordas se comportam como matrizes. No entanto, testes como este para objetos semelhantes a matrizes geralmente retornam falso para cordas - elas geralmente são melhor tratadas como cordas, não como matrizes. A maioria dos métodos de matriz JavaScript é definida propositadamente como genérico, então

que eles funcionam corretamente quando aplicados a objetos semelhantes a matrizes, além disso para as verdadeiras matrizes.Já que objetos semelhantes a matrizes não herdam `Array.prototype`, você não pode invocar métodos de matriz neles diretamente.Você pode invocar indiretamente usando a função `CALL`Método, no entanto (consulte §8.7.4 para obter detalhes):Seja `a = {"0": "a", "1": "b", "2": "c", comprimento: 3};`// Um objeto semelhante a uma matriz `Array.prototype.join.call (a, "+") // =>"A+B+C"``Array.prototype.map.call (a, x => x.TOUppercase ()) // =>["ABC"]``Array.prototype.slice.call (a, 0) // => ["a", "b", "c"]`: true cópia da matriz `Array.from (a) // => ["a", "b", "c"]`: cópia mais fácil da matrizA segunda linha deste código chama a fatia de matriz ()método em um objeto semelhante a uma matriz para copiar os elementos dissoObjeto em um verdadeiro objeto de matriz.Este é um truque idiomático que existe em muito código legado, mas agora é muito mais fácil de fazer com `Array.From ()`.7.10 Strings como matrizesStrings JavaScript se comportam como matrizes somente leitura do UTF-16 Unicode caracteres.Em vez de acessar caracteres individuais com oMétodo `Charat ()`, você pode usar colchetes:Seja `s = "teste";``S.Charat (0) // => "T"``s [1] // => "e"`

O operador TIPEOF ainda retorna "String" para Strings, é claro, e o método Array.isArray () retorna false se você passar uma corda. O principal benefício das sequências indexáveis é simplesmente que podemos substituir chamadas para charAt () com colchetes, que são mais concisos elegível e potencialmente mais eficiente. O fato de que as cordas se comportam como matrizes também significa, no entanto, que podemos aplicar a matriz genérica métodos para eles. Por exemplo: Array.prototype.join.call ("javascript", "") // => "j a v a aS c r i p t" Lembre -se de que as cordas são valores imutáveis, então quando são tratados como matrizes, são matrizes somente leitura. Métodos de matriz como push (), concat (), reverse () e splice () modificam uma matriz em Coloque e não trabalhe em cordas. Tentando modificar uma string usando um método de matriz, no entanto, não causa um erro: simplesmente falha silenciosamente.

7.11 Resumo

Este capítulo abordou as matrizes JavaScript em profundidade, incluindo:

- Detalhes sobre matrizes esparsas e objetos semelhantes a matrizes.
- Os principais pontos para tirar deste capítulo:

Literais de matrizes são escritos como listas de valores separadas por vírgulas dentro de colchetes. Elementos de matriz individuais são acessados especificando o índice de matriz desejado dentro de colchetes.

Erro ao traduzir esta página.

Capítulo 8. FunçõesEste capítulo abrange as funções JavaScript.Funções são fundamentaisBloco de construção para programas JavaScript e um recurso comum em quase todas as linguagens de programação.Você já pode estar familiarizado como conceito de uma função em um nome como sub -rotina ou procedimento.Uma função é um bloco de código JavaScript que é definido uma vez, mas pode ser executado ou invocado, várias vezes.As funções JavaScript são parametrizadas: uma definição de função pode incluir uma lista de identificadores, conhecidos como parâmetros, que funcionam como variáveis locais para o corpo da função.As invocações de função fornecem valores ou argumentos para os parâmetros da função.Funções geralmente usam seus valores de argumento para calcular um valor de retorno que se torna o valor da função-expressão de invocação.Além dos argumentos, cada invocação tem outro valor - o contexto de invocação - esse é o valor da palavra-chave `this`.Se uma função é atribuída a uma propriedade de um objeto, é conhecida como um método desse objeto.Quando uma função é invocada em ou através de um objeto, esse objeto é o contexto de invocação ou esse valor para o argumento `this`.Funções projetadas para inicializar um objeto recém-criado são construtores chamados.Os construtores foram descritos em §6.2 e serão cobertos novamente no capítulo 9.

Erro ao traduzir esta página.

Métodos. Esta sintaxe de definição de função foi abordada no §6.10.6. Observe que as funções também podem ser definidas com a função ()Construtor, que é o assunto do §8.7.7. Além disso, o JavaScript define alguns tipos especializados de funções. A função *define gerador* define geradores (consulte o capítulo 12) e a função assíncrona define funções assíncronas (consulte o Capítulo 13).

8.1.1 declarações de função

As declarações de função consistem na palavra -chave da função, seguida por esses componentes:

- Um identificador que nomeia a função. O nome é necessário.
- Parte das declarações de função: é usado como o nome de uma variável, e o objeto de função recém -definido é atribuído à variável.
- Um par de parênteses em torno de uma lista separada por vírgula de zero ou mais identificadores. Esses identificadores são os nomes de parâmetros para a função, e eles se comportam como variáveis ?? locais dentro do corpo da função.
- Um par de aparelhos encaracolados com zero ou mais declarações de JavaScript dentro. Essas declarações são o corpo da função: elas são executadas sempre que a função é invocada.

Aqui estão algumas declarações de função de exemplo:

```
// Imprima o nome e o valor de cada propriedade de O.
Retornar indefinido.
função printProps (o) {
    para (vamos P in O) {
        console.log ('$ {p}: $ {o [p]} \ n');
    }
}
```

}// calcula a distância entre os pontos cartesianos (x1, y1) e(x2, y2).distância da função (x1, y1, x2, y2) {Seja dx = x2 - x1;Seja dy = y2 - y1;retornar math.sqrt (dx*dx + dy*dy);}// uma função recursiva (que se chama) que calculafatoriais// Lembre -se que x!é o produto de x e tudo positivoInteiros menos do que isso.função fatorial (x) {if (x <= 1) retornar 1;retornar x * fatorial (x-1);}Uma das coisas importantes a entender sobre as declarações de funçãoé esse o nome da função se torna uma variável cujo valor é ofunção em si.As declarações de declaração de função são "içadas" ao topo do script, função ou bloqueio da anexo para que funções definidas emDessa forma, pode ser invocado do código que aparece antes da definição.Outra maneira de dizer isso é que todas as funções declaradas em um bloco do código JavaScript será definido em todo esse bloco, e eles irão ser definido antes que o intérprete JavaScript comece a executar qualquer um doso código nesse bloco.As funções distance () e fatorial () que descrevemos sãoprojetado para calcular um valor e eles usam o retorno para retornar esse valor para o chamador deles.A declaração de retorno faz com que a função pareexecutando e retornar o valor de sua expressão (se houver) ao chamador.Se a declaração de retorno não tiver uma expressão associada, o

Erro ao traduzir esta página.

retornar x*fato (x-1);}// As expressões de função também podem ser usadas como argumentos para outras funções:[3,2,1] .Sort (função (a, b) {return a-b;});// As expressões de função às vezes são definidas e imediatamente invocadas: Seja tensquared = (function (x) {return x*x;}) (10); Observe que o nome da função é opcional para funções definidas como expressões, e a maioria das expressões de função anterior que temos mostrado omite. Uma declaração de função realmente declara uma variável e atribui um objeto de função a ele. Uma expressão de função, por outro lado, não declara uma variável: cabe a você atribuir o recém-objeto de função definida para uma constante ou variável se você estiver indo para precisar consultar várias vezes. É uma boa prática usar const com expressões de função para que você não substitua acidentalmente seu função atribuindo novos valores. Um nome é permitido para funções, como a função factorial, que precisa consultar a si mesmos. Se uma expressão de função incluir um nome, o local de escopo da função para essa função incluirá uma ligação desse nome ao objeto de função. Na verdade, o nome da função se torna um local variável dentro da função. A maioria das funções definidas como expressões não precisa de nomes, o que torna sua definição mais compacta (embora não tão compacto quanto as funções de seta, descritas abaixo). Há uma diferença importante entre definir uma função f () com uma declaração de função e atribuição de uma função à variável f após criando isso como uma expressão. Quando você usa o formulário de declaração, os objetos de função são criados antes que o código que os contém comece a

executar, e as definições são içadas para que você possa chamar essas funções do código que aparece acima da instrução de definição. Isso não é verdade. Para funções definidas como expressões, no entanto: essas funções não existem até que a expressão que os define seja realmente avaliada. Além disso, para invocar uma função, você deve ser capaz de se referir a ela, e você não pode se referir a uma função definida como uma expressão até que seja atribuída a uma variável, portanto as funções definidas com expressões não podem ser invocadas antes de serem definidas.

8.1.3 Funções de seta
No ES6, você pode definir funções usando uma sintaxe particularmente compacta conhecida como "funções de seta". Esta sintaxe é uma reminiscência da notação matemática e usa um `=>` "seta" para separar a função dos parâmetros do corpo da função. A palavra-chave `function` não é usada, e, como as funções de seta são expressões em vez de declarações, também não há necessidade de um nome de função. A forma geral de uma função de seta é uma lista de parâmetros separados por vírgula entre parênteses, seguida pela seta `=>` seguida pelo corpo da função em curly brace orthodontic:

```
const sum = (x, y) => {return x + y;};
```

Mas as funções de seta suportam uma sintaxe ainda mais compacta. Se o corpo da função é uma única declaração de retorno, você pode omitir o `return` keyword, o semicolon que acompanha, e o encaracolado brace e escreva o corpo da função como a expressão cujo valor deve ser devolvido:

```
const sum = (x, y) => x + y;
```

Erro ao traduzir esta página.

Erro ao traduzir esta página.

função quadrado (x) {return x*x; }retornar math.sqrt (quadrado (a) + quadrado (b)); }O interessante sobre as funções aninhadas é o seu escopo variávelRegras: eles podem acessar os parâmetros e variáveis ??da função (oufunções) eles são aninhados dentro.No código mostrado aqui, por exemplo,A função interna Square () pode ler e escrever os parâmetros a eb definido pela função externa hipotenuse ().Essas regras de escopopois as funções aninhadas são muito importantes, e nós as consideraremosnovamente em §8.6.8.2 Funções de invocaçãoO código JavaScript que compõe o corpo de uma função não éexecutado quando a função é definida, mas sim quando é invocada.As funções JavaScript podem ser invocadas de cinco maneiras:Como funçõesComo métodosComo construtoresIndiretamente através de seus métodos de chamada () e Aplicação ()Implicitamente, via recursos de linguagem JavaScript que não aparecemcomo invocações de função normal8.2.1 Invocação da funçãoAs funções são invocadas como funções ou métodos com uma invocaçãoexpressão (§4.5).Uma expressão de invocação consiste em uma função

expressão que avalia para um objeto de função seguido por um aberto parênteses, uma lista separada por vírgula de zero ou mais argumentos expressões e um parêntese estreito. Se a expressão da função for um expressão de acesso à propriedade - se a função é propriedade de um objeto ou um elemento de uma matriz - então é uma expressão de invocação de método. Esse caso será explicado no exemplo a seguir. A seguir o código inclui várias expressões regulares de invocação de funções: printProps ({x: 1}); Deixe total = distância (0,0,2,1) + distância (2,1,3,5); Deixe a probabilidade = fatorial (5)/fatorial (13); Em uma invocação, cada expressão de argumento (aqueles entre os parênteses) é avaliado e os valores resultantes se tornam argumentos para a função. Esses valores são atribuídos aos parâmetros nomeados na definição da função. No corpo da função, uma referência a um parâmetro avalia o argumento correspondente valor. Para invocação regular de funções, o valor de retorno da função torna-se o valor da expressão de invocação. Se a função retornar como o intérprete atinge o fim, o valor de retorno é indefinido. Se a função retornar porque o intérprete executa uma declaração de retorno, então o valor de retorno é o valor da expressão que segue o retorno ou é indefinido se a declaração de retorno não tem valor. Invocação condicional No ES2020, você pode inserir?.após a expressão da função e antes do parêntese aberto em um invocação da função para invocar a função apenas se não for nula ou indefinida. Isto é, o

A expressão `f?.(x)` é equivalente (assumindo nenhum efeito colateral) a:`(f! == null && f! == indefinido)?f (x): indefinido`. Detalhes completos sobre essa sintaxe de invocação condicional estão no §4.5.1. Para invocação de funções no modo não estrito, o contexto de invocação (oeste valor) é o objeto global. No modo rigoroso, no entanto, o contexto de invocação é indefinido. Observe que as funções definidas usando a sintaxe da flecha se comporta de maneira diferente: eles sempre herdam o valor que está em vigor onde eles são definidos. Funções escritas para serem invocadas como funções (e não como métodos) normalmente não usa essa palavra-chave. A palavra-chave pode ser usada. No entanto, para determinar se o modo rigoroso está em vigor:// Defina e invocar uma função para determinar se estamos dentro do modo rigoroso.`const strict = (function () {return! this;})()`; Funções recursivas e a pilha Uma função recursiva é uma, como a função factorial () no início deste capítulo, que se chama. Alguns algoritmos, como os que envolvem estruturas de dados baseados em árvores, podem ser implementados particularmente elegantemente com funções recursivas. Ao escrever uma função recursiva, no entanto, é importante pensar sobre restrições de memória. Quando uma função a chama a função b e depois a função b chama a função c, o intérprete JavaScript precisa acompanhar os contextos de execução para todas as três funções. Quando a função C completa, o intérprete precisa saber onde retomar a execução da função b e quando a função B completa, precisa saber onde retomar a execução da função A. Você pode imaginar isso em contextos de execução como uma pilha. Quando uma função chama outra função, um novo contexto de execução é empurrado para a pilha. Quando essa função retorna, seu objeto de contexto de execução é retirado da pilha. Se uma função se chamar recursivamente 100 vezes, a pilha terá 100 objetos empurrados nela e depois desses 100 objetos surgiram. Esta pilha de chamadas leva a memória. No hardware moderno, é normalmente tudo bem escrever funções recursivas que se chamam centenas de vezes. Mas se uma função se chama dez milhares de vezes, é provável que falhe com um erro como "o tamanho máximo da pilha de chamadas excedido".

8.2.2 Invocação do método

Um método nada mais é do que uma função de JavaScript que é armazenada em umpropriedade de um objeto.Se você tem uma função f e um objeto o, você podeDefina um método chamado m de O com a seguinte linha:`o.m = f;`Tendo definido o método m () do objeto O, invocará assim:`O.M ()`;Ou, se m () esperar dois argumentos, você pode invocar assim:`O.M (x, y)`;O código neste exemplo é uma expressão de invocação: inclui umExpressão da função O.M e duas expressões de argumento, x e y.OA expressão da função é em si uma expressão de acesso à propriedade, e estesignifica que a função é invocada como um método e não como um regularfunção.Os argumentos e o valor de retorno de uma invocação de método são tratadosExatamente como descrito para invocação regular de função.Métodoas invocações diferem das invocações de funções de uma maneira importante,No entanto: o contexto de invocação.Expressões de acesso à propriedade consistem emDuas partes: um objeto (neste caso O) e um nome de propriedade (M).Em umExpressão de vocação de métodos como esta, o objeto O se torna ocontexto de invocação, e o corpo da função pode se referir a esse objeto porusando a palavra -chave isso.Aqui está um exemplo concreto:

Deixe a calculadora = { // um objeto literal operand1: 1, operand2: 1, add () { // estamos usando o método abreviação da sintaxe para estabelecer a função // observe o uso da palavra-chave para se referir ao contendo objeto. this.result = this.operand1 + this.operand2; }; calculator.add () // Uma invocação de método para calcular 1+1. calculator.Result // => 2 A maioria das invocações de métodos usa a notação de pontos para acesso à propriedade, mas expressões de acesso à propriedade que usam colchetes também causam método de invocação. A seguir, estão as duas invocações de método, por exemplo: o ["m"] (x, y); // Outra maneira de escrever o.m (x, y). a [0] (z) // também uma invocação de método (assumindo que um [0] é uma função). Invocações de método também podem envolver acesso mais complexo à propriedade expressões: client.surname.toUpperCase (); // Invocar o método Customer.surname(); M (); // Invocar o método m () em valor de retorno de f () Métodos e essa palavra-chave são centrais para os objetos orientados a objetos paradigm de programação. Qualquer função usada como método é passou efetivamente um argumento implícito - o objeto através do qual é invocado. Normalmente, um método executa algum tipo de operação nesse objeto, e a sintaxe de invocação de métodos é uma maneira elegante de expressar o fato de uma função estar operando em um objeto. Compare o

A seguir, duas linhas:Rect.setSize (largura, altura);setRectSize (ret, largura, altura);As funções hipotéticas invocadas nessas duas linhas de código podem executar exatamente a mesma operação no objeto (hipotético) rect,Mas a sintaxe de invocação de métodos na primeira linha indica mais claramente a ideia de que é o objeto rety que é o foco principal da operação.Encadeamento de métodoQuando os métodos retornam objetos, você pode usar o valor de retorno de uma invocação de método como parte de uma invocação subsequente.Isso resulta em uma série (ou "cadeia") de invocações de método como um único expressão.Ao trabalhar com operações assíncronas baseadas em promessas (consulte o capítulo 13), por exemplo, é comum escrever código estruturado assim:// Execute três operações assíncronas em sequência, manipulando erros.DOSTEPONE () .Então (DostepTwo) .THEN (DOSTEPThree) .CACTH (HODEERRORES);Quando você escreve um método que não tem um valor de retorno próprio, considere ter o método devolver isso.Se você fizer isso de forma consistente em toda a sua API, você permitirá um estilo de programação conhecido como encadeamento de método em que um objeto pode ser nomeado uma vez e, em seguida, vários métodos podem ser invocados nele:novo quadrado () .X (100) .y (100) .Size (50) .Outline ("vermelho") .Fill ("azul") .Draw ();Observe que essa é uma palavra-chave, não uma variável ou nome da propriedade.A sintaxe do JavaScript não permite atribuir um valor a isso.A palavra-chave não é escopo da maneira como as variáveis ??são e, exceto funções de seta, funções aninhadas não herdam o valor deste contendo função.Se uma função aninhada é invocada como um método, seu1

Este valor é o objeto em que foi invocado. Se uma função aninhada (isto é não uma função de seta) é invocada como uma função, então é esse valor será o objeto global (modo não rito) ou indefinido (modo rigoroso). É um erro comum supor que uma função aninhada definida em um método e invocada como uma função pode usar isso para obter o contexto de invocação do método. O código a seguir demonstra o problema: Seja `o = {}`; um objeto `o.M: function () { // Método m do objeto. Deixe eu = isso; // salve o valor "Este" em um variável. this === o } // => true`; "this" é o objeto `o.f ()`; // Agora chame a função auxiliar `f ()`. função `f () { // uma função aninhada f. this === o } // => false`; "this" é global ou indefinido `self === o } // => true`; eu é o exterior "esse valor".}); o.M (); // invocar o método m no objeto `o`. Dentro da função aninhada `f ()`, a palavra -chave não é igual ao objeto `o`. Isso é amplamente considerado uma falha no JavaScript linguagem, e é importante estar ciente disso. O código acima demonstra uma solução alternativa comum. Dentro do método `m`, nós atribua o valor a um `eu` variável e dentro do aninhado função `f`, podemos usar `eu` em vez disso para nos referir ao contendo objeto.

No ES6 e mais tarde, outra solução alternativa a esta questão é converter a função aninhada `f` em uma função de seta, que herdará adequadamente o valor deste:`const f = () => {this === o // verdadeiro, já que as funções de seta herdam isso};` Funções definidas como expressões em vez de declarações não são içadas. Então, para fazer com que este código funcione, a definição de função para `f` irá precisar ser movida dentro do método `m` para que ele apareça antes que seja invocado. Outra solução alternativa é invocar o método `bind()` do aninhado da função para definir uma nova função que é implicitamente invocada em um objeto especificado:`const f = (function () {this === o // verdadeiro, já que ligamos essa função ao exterior isto}).bind(this);` Falaremos mais sobre `Bind()` em §8.7.5.8.2.3 Invocação do construtor.

Se uma função ou invocação de método for precedida pela palavra-chave `new`, então é uma invocação do construtor. (As invocações construtivas foram introduzidas em §4.6 e §6.2.2, e os construtores serão cobertos em mais detalhes no Capítulo 9.) As invocações do construtor diferem das regular invocações de função e método no manejo de argumentos, contexto de invocação e valor de retorno.

Se uma invocação de construtor incluir uma lista de argumentos entre parênteses, essas expressões de argumento são avaliadas e passadas para a função em da mesma maneira que seriam para invocações de função e método. Isso é prática não comum, mas você pode omitir um par de parênteses vazios em uma invocação do construtor. As duas linhas a seguir, por exemplo, são equivalentes:
o = new Object();
o = novo objeto;
Uma invocação construtora cria um novo objeto vazio que herda o objeto especificado pela propriedade do protótipo do construtor. As funções do construtor têm como objetivo inicializar objetos, e este recentemente o objeto criado é usado como contexto de invocação, então o construtorA função pode se referir a ela com a palavra -chave esta. Observe que o novoO objeto é usado como contexto de invocação, mesmo que o construtorA invocação parece uma invocação de método. Isto é, na expressão O novo O.M (), O não é usado como contexto de invocação. As funções do construtor normalmente não usam a palavra -chave de retorno. Eles normalmente initialize o novo objeto e depois retorne implicitamente quando ele alcançar o fim do corpo. Nesse caso, o novo objeto é o valor de A expressão de invocação do construtor. Se, no entanto, um construtor usa explicitamente a declaração de retorno para devolver um objeto, então que O objeto se torna o valor da expressão de invocação. Se o construtor usa o retorno sem valor, ou se retornar um primitivo valor, esse valor de retorno é ignorado e o novo objeto é usado como o valor da invocação.

8.2.4 Invocação indiretaAs funções JavaScript são objetos e, como todos os objetos JavaScript, elestêm métodos.Dois desses métodos, Call () e Apply (), Invokea função indiretamente.Ambos os métodos permitem especificar explicitamente o valor deste valor para a invocação, o que significa que você pode invocar qualquer função como um método de qualquer objeto, mesmo que não seja realmente um método desse objeto.Ambos os métodos também permitem especificar os argumentos para a invocação.O método Call () usa sua própria lista de argumentos como argumentos para a função, e o método APLIC () espera uma matriz de valores a serem usados ??como argumentos.A chamada () e Apply ()Os métodos são descritos em detalhes em §8.7.4.8.2.5 Invocação de função implícitaExistem vários recursos de linguagem JavaScript que não parecem Invocações da função, mas que fazem com que as funções sejam invocadas.Ser extracuidadoso ao escrever funções que podem ser invocadas implicitamente, porque Bugs, efeitos colaterais e problemas de desempenho nessas funções são mais difíceis para diagnosticar e consertar do que em funções regulares pela simples razão de que pode não ser óbvio de uma simples inspeção do seu código quando ele está sendo chamados.Os recursos do idioma que podem causar invocação de função implícita incluir:Se um objeto tiver getters ou setters definidos, então consulta ou Definir o valor de suas propriedades pode invocar esses métodos.Consulte §6.10.6 para obter mais informações.Quando um objeto é usado em um contexto de string (como quando é

Concatenado com uma string), seu método `ToString()` é chamado. Da mesma forma, quando um objeto é usado em um contexto numérico, seu método `valueOf()` é chamado. Veja §3.9.3 para obter detalhes. Quando você percorre os elementos de um objeto iterável, há uma série de chamadas de método que ocorrem. O capítulo 12 explica como os iteradores funcionam no nível da chamada de função e demonstra como escrever esses métodos para que você possa definir o seu próprio tipo iterável. Um modelo etiquetado literal é uma invocação de função disfarçada. §14.5 demonstra como escrever funções que podem ser usadas em conjunção com cordas literais de modelo. Objetos de proxy (descritos no §14.7) têm seu comportamento completamente controlado por funções. Quase qualquer operação em um desses objetos, fará com que uma função seja invocada.

8.3 Argumentos e parâmetros de função

As definições de função JavaScript não especificam um tipo esperado para os parâmetros de função e invocações de função não fazem nenhum tipo de verificação dos valores de argumento que você passa. De fato, função JavaScript as invocações nem sequer verificam o número de argumentos que estão sendo aprovados. As subseções a seguir descrevem o que acontece quando uma função é invocada com menos argumentos do que os parâmetros declarados ou com mais argumentos do que os parâmetros declarados. Eles também demonstram como você pode testar explicitamente o tipo de argumentos de função, se você precisar garantir que uma função não é invocada com argumentos inadequados.

8.3.1 Parâmetros e padrões opcionais

Quando uma função é invocada com menos argumentos do que declarado

Parâmetros, os parâmetros adicionais são definidos como seu valor padrão, que normalmente é indefinido. Muitas vezes é útil escrever funções, então que alguns argumentos são opcionais. A seguir, um exemplo:// anexar os nomes das propriedades enumeráveis ??do objeto O para o// Matriz A, e retorne a. Se A for omitido, crie e retorne uma nova matriz.função getPropertyNames (o, a) {if (a === indefinido) a = [];// Se indefinido, use um novo variação de para (deixe a propriedade em O) a.push (propriedade); retornar a;}// getPropertyNames () pode ser invocado com um ou doisArgumentos: Seja o = {x: 1}, p = {y: 2, z: 3};// dois objetos para teste deixa a = getPropertyNames (o); // a == ["x"]; Obtenha O'spropriedades em uma nova matriz getPropertyNames (P, A); // a == ["x", "y", "z"]; Adicione P'spropriedades para issoEm vez de usar uma instrução IF na primeira linha desta função, você pode usar o || Operador dessa maneira idiomática:a = a ||[]; Lembre -se do §4.10.2 de que o || O operador retorna seu primeiro argumento seEsse argumento é verdade e retorna seu segundo argumento. EmEste caso, se algum objeto for passado como o segundo argumento, a função usará esse objeto. Mas se o segundo argumento for omitido (ou nulo ouOutro valor falsário é passado), uma matriz vazia recém -criada será usada em vez de.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Um parâmetro de descanso é precedido por três períodos, e deve ser o último parâmetro em uma declaração de função. Quando você invoca uma função com um Parâmetro REST, os argumentos que você passa são atribuídos pela primeira vez ao não-restaurante parâmetros, e então quaisquer argumentos restantes (isto é, o "descanso" do argumento) são armazenados em uma matriz que se torna o valor do resto parâmetro. Este último ponto é importante: dentro do corpo de uma função, o valor de um parâmetro REST sempre será uma matriz. A matriz pode ser vazia, mas um parâmetro de descanso nunca será indefinido. (Segue-se a partir disso, nunca é útil - e não é legal - definir um parâmetro padrão para um parâmetro de descanso.) Funções como o exemplo anterior que podem aceitar qualquer número de argumentos são chamados funções variádicas, funções variáveis ??de arity ou funções vararg. Este livro usa o termo mais coloquial, varargs, que data dos primeiros dias da linguagem de programação C. Não confunda o ... que define um parâmetro de descanso em uma função Definição com o ... Spread Operator, descrito em §8.3.4, que pode ser usado em invocações de função.

8.3.3 O objeto de argumentos

Os parâmetros de reposo foram introduzidos no JavaScript no ES6. Antes disso, na versão do idioma, as funções varargs foram escritas usando o Objeto de argumentos: dentro do corpo de qualquer função, o identificador Arguments refere-se ao objeto de argumentos para essa invocação. O Objeto de argumentos é um objeto semelhante a uma matriz (ver §7.9) que permite os valores de argumento passados ??para a função a serem recuperados por número, ao invés de nome. Aqui está a função max () de anteriores,

reescrito para usar o objeto Argumentos em vez de um parâmetro REST:função max (x) {Seja maxvalue = -infinity;// percorre os argumentos, procurando eLembrando, o maior.para (vamos i = 0; i <argumentos.Length; i ++){if (argumentos [i]> maxvalue) maxvalue = argumentos [i];}// retorna o maiorretornar maxvalue;}max (1, 10, 100, 2, 3, 1000, 4, 5, 6) // => 1000O objeto de argumentos remonta aos primeiros dias do JavaScript ecarrega consigo alguma bagagem histórica estranha que a torna ineficiente difícil de otimizar, especialmente fora do modo rigoroso.Você ainda podeEncontre código que usa o objeto de argumentos, mas você deve evitarUsando -o em qualquer novo código que você escreve.Ao refatorar o código antigo, se vocêencontrar uma função que use argumentos, você pode substituí -locom um ... parâmetro de repouso args.Parte do infeliz legado doO objeto de argumentos é que, no modo rigoroso, os argumentos são tratados como umpalavra reservada, e você não pode declarar um parâmetro de função ou um localvariável com esse nome.8.3.4 O operador de propagação para chamadas de funçãoO operador de spread ... é usado para descompactar ou "espalhar", oelementos de uma matriz (ou qualquer outro objeto iterável, como seqüências) em umcontexto em que os valores individuais são esperados.Nós vimos a propagaçãoOperador usado com literais de matriz em §7.1.2.O operador pode ser usado, emDa mesma maneira, nas invocações de função:

Erro ao traduzir esta página.

Erro ao traduzir esta página.

```
}Vectoradd ([1,2], [3,4]) // => [4,6]O código seria mais fácil de entender se destruímos os doisArgumentos vetoriais  
em parâmetros mais claramente nomeados:função vetoradd ([x1, y1], [x2, y2]) { // descompactar 2 argumentosem 4  
parâmetrosretornar [x1 + x2, y1 + y2];}Vectoradd ([1,2], [3,4]) // => [4,6]Da mesma forma, se você está definindo  
uma função que espera um objetoargumento, você pode destruir parâmetros desse objeto.Vamos usar umExemplo  
de vetor novamente, exceto desta vez, vamos supor que representamosvetores como objetos com parâmetros x e  
y:// multiplique o vetor {x, y} por um valor escalarfunção vectormultiply ({x, y}, escalar) {retornar {x: x*escalar, y:  
y*escalar};}vectormultiply ({x: 1, y: 2}, 2) // => {x: 2, y: 4}Este exemplo de destruir um único argumento de objeto em  
doisParâmetros é bastante claro porque os nomes de parâmetros que usamosCombine os nomes de propriedades  
do objeto de entrada.A sintaxe é maisVerboso e mais confuso quando você precisa destruir as propriedadescom  
um nome em parâmetros com nomes diferentes.Aqui está o vetorExemplo de adição, implementado para vetores  
baseados em objetos:função vetoradd ({x: x1, y: y1}, // descompacte o 1º objeto em x1 e y1params{x: x2, y: y2} //  
Desmarque o 2º objeto em x2 e y2
```

params){retornar {x: x1 + x2, y: y1 + y2};}VectorAdd ({x: 1, y: 2}, {x: 3, y: 4}) // => {x: 4, y: 6}A coisa complicada de destruir a sintaxe como {x: x1, y: y1} é lembrando quais são os nomes de propriedades e quais são os nomes de parâmetros.A regra a ter em mente para destruir atribuição e destruição de chamadas de função é que as variáveis ??ou Parâmetros que estão sendo declarados vão nos pontos onde você esperava em um objeto literal.Portanto, os nomes de propriedades estão sempre à mão esquerda do colón, e os nomes de parâmetros (ou variáveis) estão certo.Você pode definir padrões de parâmetro com parâmetros destrutivos.Aqui está a multiplicação de vetores que funciona com vetores 2D ou 3D:// multiplique o vetor {x, y} ou {x, y, z} por um valor escalarfunção vectormultiply ({x, y, z = 0}, escalar) {retornar {x: x*escalar, y: y*escalar, z: z*escalar};}vectormultiply ({x: 1, y: 2}, 2) // => {x: 2, y: 4, z: 0}Alguns idiomas (como o python) permitem que o chamador de uma função invocar uma função com argumentos especificados em nome = formulário de valor, que é conveniente quando há muitos argumentos opcionais ou quando a lista de parâmetros é longa o suficiente para que seja difícil lembrar o correto dem.Javascript não permite isso diretamente, mas você pode se aproximarDestruir um argumento de objeto em seus parâmetros de função.Considere uma função que copia um número especificado de elementos de

uma matriz em outra matriz com desativação de partida opcionalmente especificada para cada matriz. Como existem cinco parâmetros possíveis, alguns dos quais são padrões e seria difícil para um chamador lembrar qual para passar os argumentos, podemos definir e invocar o `Arraycopy()` função como esta:

```
Função Arraycopy
({de, para = de, n = de.length, deIndex = 0, tolIndex = 0}) {Deixe valuesto cópia = de.slice (FromIndex, FromIndex + N); to.splice (tolIndex, 0, ... valuesto cópia); retornar a;} Seja a = [1,2,3,4,5], b = [9,8,7,6,5]; Arraycopy ({de: a, n: 3, para: b, tolIndex: 4}) // =>[9,8,7,6,1,2,3,5]
```

Quando você destruir uma matriz, você pode definir um parâmetro de descanso para valores extras dentro da matriz que está sendo descompactada. Aquele descanso O parâmetro dentro dos colchetes é completamente diferente do Parâmetro Rest True para a função:// Esta função espera um argumento de matriz. Os dois primeiros elementos disso// a matriz são descompactados nos parâmetros X e Y. Qualquer elemento restante// são armazenados na matriz Coords. E quaisquer argumentos depois da primeira matriz// são embalados na matriz REST.

```
função f ([x, y, ... coords], ... descanso) {retornar [x+y, ... descanso, ... coordenar];} // Nota: espalhe operador aqui} f ([1, 2, 3, 4], 5, 6) // => [3, 5, 6, 3, 4]
```

No ES2018, você também pode usar um parâmetro de descanso quando destruir um objeto. O valor desse parâmetro de repouso será um objeto que tem algum

Erro ao traduzir esta página.

Os parâmetros do método JavaScript não têm tipos declarados, e nenhum tipoA verificação é executada nos valores que você passa para uma função.Você podeAjude a tornar seu código auto-documentação escolhendo nomes descritivospara argumentos de função e documentá -los cuidadosamente noComentários para cada função.(Alternativamente, consulte §17.8 para um idiomaExtensão que permite que você verifique o tipo de verificação em cima deJavascript.)Conforme descrito no §3.9, o JavaScript realiza a conversão do tipo liberal comonecessário.Então, se você escrever uma função que espera um argumento de string eEntão chame essa função com um valor de algum outro tipo, o valor que vocêPassado será simplesmente convertido em uma string quando a função tentar parause -o como uma string.Todos os tipos primitivos podem ser convertidos em cordas, e tudoObjetos possuem métodos `toString ()` (se não necessariamente úteis),Portanto, um erro nunca ocorre neste caso.Issso nem sempre é verdade, no entanto.Considere novamente o `Arraycopy ()`Método mostrado anteriormente.Espera um ou dois argumentos de matriz e irãoFalha se esses argumentos forem do tipo errado.A menos que você esteja escrevendo umfunção privada que será chamada apenas de partes próximas do seu código,Pode valer a pena adicionar código para verificar os tipos de argumentos como este.É melhor para uma função falhar imediatamente e previsivelmente quandopassou valores ruins do que começar a executar e falhar mais tarde com um erromensagem que provavelmente não é clara.Aqui está um exemplo de função queExecuta a verificação do tipo:// retorna a soma dos elementos Um objeto iterável a// Os elementos de um devem ser números.Função Sum (a) {Deixe total = 0;

para (deixe o elemento de a) { // lança TypeError se A não for iterável
if (typeof elemento! == "número") {jogue novo
TypeError ("Sum (): os elementos devem ser números");}
total += elemento;
}retorno total;
}soma ([1,2,3]) // => 6
soma (1, 2, 3); //! TypeError: 1 não é iterável
soma ([1,2, "3"]); //! TypeError: elemento 2 não é um número

8.4 Funções como valores
As características mais importantes das funções são que elas podem ser definidas e invocadas. Definição e invocação da função são recursos sintáticos de JavaScript e da maioria das outras linguagens de programação. Em JavaScript, no entanto, as funções não são apenas sintaxe, mas também valores, o que significa que elas podem ser atribuídas a variáveis, armazenadas nas propriedades dos objetos ou usados como argumentos para outras funções e assim por diante.

Para entender como as funções podem ser dados de JavaScript, bem como sua sintaxe, considere esta definição de função:

```
função quadrado (x)  
{return x*x;}
```

Esta definição cria um novo objeto de função e o atribui ao quadrado variável. O nome de uma função é realmente imaterial; isso é simplesmente o nome de uma variável que se refere ao objeto de função. A função pode ser atribuída a outra variável e ainda funciona da mesma forma:

```
caminho:3
```

Erro ao traduzir esta página.

§7.8.6.Exemplo 8-1 demonstra os tipos de coisas que podem ser feitas quandoAs funções são usadas como valores.Este exemplo pode ser um pouco complicado, masOs comentários explicam o que está acontecendo.Exemplo 8-1.Usando funções como dados// Definimos algumas funções simples aquifunção add (x, y) {return x + y;}função subtrair (x, y) {return x - y;}função multiplique (x, y) {return x * y;}função divide (x, y) {return x / y;}// Aqui está uma função que leva uma das funções anteriores// como um argumento e invoca em dois operandosfunção operar (operador, operand1, operand2) {operador de retorno (operand1, operand2);}// Podemos invocar esta função como essa para calcular o valor(2 + 3) + (4*5):Seja i = operar (add, opere (add, 2, 3), opere (multiplique, 4,5));// Para o bem do exemplo, implementamos o simplesfunciona novamente,// desta vez dentro de um objeto literal;operadores const = {Adicione: (x, y) => x+y,subtrair: (x, y) => x-y,Multiplique: (x, y) => x*y,Divida: (x, y) => x/y,Pow: Math.pow // isso funciona para funções predefinidastambém};// Esta função leva o nome de um operador, olha para cima queoperador// no objeto e, em seguida, chama -o nos operandos fornecidos.Observação

// A sintaxe usada para chamar a função do operador.função operate2 (operação, operand1, operand2) {if (typeof operadores [operação] === "function") {operadores de retorno [operação] (operand1, operand2);}caso contrário, jogue "Operador desconhecido";}Operate2 ("Add", "Hello", Operate2 ("Add", "", "World")) // =>"Hello World"Operate2 ("Pow", 10, 2) // => 1008.4.1 Definindo suas próprias propriedades de funçãoFunções não são valores primitivos em JavaScript, mas um tipo especializado de objeto, o que significa que as funções podem ter propriedades.Quando uma função precisa de uma variável "estática" cujo valor persiste entre invocações, muitas vezes é conveniente usar uma propriedade da função em si.Por exemplo, suponha que você quiser escrever uma função que retorne um inteiro único sempre que é invocado.A função nunca deve retornar o mesmo valor duas vezes.Para gerenciar isso, a função precisa acompanhar os valores que ele já retornou e esta informação deve persistir entre invocações de função.Você poderia armazenar essa informação em uma variável global, mas isso é desnecessário, porque as informações são usadas apenas pela própria função.É melhor armazenar informações em uma propriedade do objeto de função.Aqui está um exemplo: quando é chamado:// Inicialize a propriedade Contador do objeto de função.// declarações de função são iniciadas para que realmente possamos// Faça esta tarefa antes da declaração da função.ÚnicoInteger.counter = 0;// Esta função retorna um número inteiro diferente cada vez que for chamado.

```
// ele usa uma propriedade de si mesma para lembrar o próximo valor paraser devolvido.function ÚnicoInteger ()  
{retornar exclusivoInteger.counter ++;// retornar e incrementocontra -propriedade}Únicanteger () // =>  
0ÚnicoInteger () // => 1Como outro exemplo, considere a seguinte função factorial ()que usa propriedades de si  
mesma (tratando -se como uma matriz) para armazenar em cacheResultados previamente calculados:// calcular  
fatoriais e resultados de cache como propriedades dofunção em si.função factorial (n) {if (number.isInteger (n) &&  
n> 0) {// positivoApenas números inteirosif (! (n em factorial)) {// se nãoresultado em cachefactorial [n] = n * factorial  
(n-1);// calcular cache}retornar factorial [n];// Retornaro resultado em cache} outro {Nan de retorno;// se entradafoi  
ruim}}factorial [1] = 1;// initialize o cache para manter esta basecaso.Fatorial (6) // => 720factorial [5] // => 120;a  
chamada acima do armazenamento em cache este valor8.5 Funções como namespacesVariáveis ??declaradas  
dentro de uma função não são visíveis fora do
```

função. Por esse motivo, às vezes é útil definir uma função simplesmente para agir como um espaço de nome temporário no qual você pode definir variáveis ?? sem atrapalhar o espaço de nome global. Suponha, por exemplo, você tem um pedaço de código JavaScript que você deseja usar em vários programas JavaScript diferentes (ou, para o cliente-JavaScript lateral, em várias páginas da web diferentes). Suponha que isso é código, como a maioria do código, define variáveis ?? para armazenar os resultados intermediários de sua computação. O problema é que, uma vez que esse pedaço de código será usado em muitos programas diferentes, você não sabe se as variáveis criadas pelo programa que usam entrarão em conflito com variáveis ?? criadas pelos outros programas.

A solução é colocar o pedaço de código em uma função e depois invocá-la. Dessa forma, variáveis ?? que teriam sido globais tornam-se locais para a função:

```
function chunkNamespace () { // pedaço de código vai aqui // Quaisquer variáveis ?? definidas no pedaço são locais para essa função // em vez de atrapalhar o espaço de nome global. } // mas não se esqueça de invocar a função! Este código define apenas uma única variável global: o nome da função.
```

Se definir até uma única propriedade é demais, você pode definir e invocar uma função anônima em um único expressão:

```
(function () { // chunknamespace () função reescrita como uma expressão sem nome. // pedaço de código vai aqui } ()) // termina a função literal e invocar agora.
```

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

retornou.Uma falha dessa abordagem é que buggy ou malicioso código pode redefinir o contador ou definir -lo para um não -inteiro, causando oFunção exclusiva () para violar o "único" ou o "número inteiro"parte de seu contrato.Os fechamentos capturam as variáveis ??locais de um únicoInvocação da função e pode usar essas variáveis ??como estado privado.Aqui estáComo poderíamos reescrever o exclusivo () usando um imediatamenteExpressão da função invocada para definir um espaço para nome e um fechamento queUsos esse espaço para nome para manter seu estado privado:Deixe exclusivodeixe o contador = 0;// estado privado defunção abaixoReturn function () {retorna contador ++;};} ());Únicanteger () // => 0ÚnicoInteger () // => 1Para entender esse código, você deve lê -lo com cuidado.InicialmenteOlhar, a primeira linha de código parece estar atribuindo uma função ao variável exclusiva.De fato, o código está definindo e invocando(como sugerido pelos parênteses abertos na primeira linha) uma função, então é o valor de retorno da função que está sendo atribuída aÚnico.Agora, se estudarmos o corpo da função, vemosque seu valor de retorno é outra função.É esse objeto de função aninhadoIsso é atribuído ao ÚnicoInteger.A função aninhada tem acesso às variáveis ??em seu escopo e pode usar a variável contador definido na função externa.Uma vez que essa função externa retorne, nenhum outroO código pode ver a variável contador: a função interna tem exclusivoacesso a ele.

Variáveis ??privadas como o contador não precisam ser exclusivas de um únicoFechamento: é perfeitamente possível para duas ou mais funções aninhadas seremdefinido na mesma função externa e compartilhe o mesmo escopo.Considere o seguinte código:função contador () {Seja n = 0;retornar {contagem: function () {return n ++;},Redefinir: function () {n = 0;}};}Seja c = contador (), d = contador ()// Crie dois contadoresc.count () // => 0D.Count () // => 0: eles contam independentementec.reset ()// reset () e count ()Métodos compartilham estadoc.count () // => 0: porque redefinimoscD.Count () // => 1: D não foi redefinidoA função do contador () retorna um objeto "contador".Este objeto temDois métodos: count () retorna o próximo número inteiro e redefinir () redefinir o estado interno.A primeira coisa a entender é que os dois métodosCompartilhe o acesso à variável privada n.A segunda coisa a entenderé que cada invocação de contador () cria um novo escopo -independente dos escopos usados ??por invocações anteriores - e um novoVariável privada dentro desse escopo.Então, se você ligar para o contador () duas vezes,Você recebe dois contadores com diferentes variáveis ??privadas.Chamando count () ou reset () em um contador objeto não tem efeito nooutro.

Vale a pena notar aqui que você pode combinar esta técnica de fechamento com getters e setters de propriedades. A seguinte versão do Counter () Função é uma variação no código que apareceu em §6.10.6. Mas ele usa fechamentos para estado privado, em vez de confiar regularmente Propriedade do objeto: contador de função (n) { // Argumento de função n é o privado variável retornar { // Método Getter de propriedade retorna e incrementos Contador privado var. obtenha count () { return n ++; } // O Setter Property não permite o valor de n para diminuir Set Count (M) { if (m > n) n = m; caso contrário, o erro de lançamento ("a contagem só pode ser definida como um valor maior "); } } } Seja c = contador (1000); c.count // => 1000 c.count // => 1001 C.Count = 2000; C.Count // => 2000 C.Count = 2000; //! Erro: a contagem só pode ser definida como um valor maior Observe que esta versão da função do contador () não declara um variável local, mas apenas usa seu parâmetro n para manter o estado privado compartilhado pelos métodos de acessórios de propriedade. Isso permite o chamador de contador () especificar o valor inicial da variável privada. Exemplo 8-2 é uma generalização do estado privado compartilhado através do

Erro ao traduzir esta página.

Jogue novo TypeError ('set \$ {Name}: Valor inválido\$ {v} `);} outro {valor = v;});}// O código a seguir demonstra o addPrivateProperty ()método.Seja o = {};// aqui está um objeto vazio// Adicionar métodos de acessórios de propriedade GetNome e SetNome ()// Verifique se apenas os valores da string são permitidosaddPrivateProperty (O, "Nome", X => TIPOF x === "String");o.setName ("Frank");// Defina o valor da propriedadeo.getName () // => "Frank"o.setName (0);//! TypeError: tente definir um valor deo tipo erradoAgora vimos vários exemplos em que dois fechamentos sãodefinido no mesmo escopo e compartilhe acesso à mesma variável privadaou variáveis.Esta é uma técnica importante, mas é igualmente importantepara reconhecer quando os fechamentos compartilham inadvertidamente o acesso a uma variável queEles não devem compartilhar.Considere o seguinte código:// Esta função retorna uma função que sempre retorna Vfunção constfunc (v) {return () => v;}// Crie uma variedade de funções constantes:deixe funcs = []for (var i = 0; i <10; i ++) funcs [i] = constfunc (i); // A função no elemento da matriz 5 retorna o valor 5.funcs [5] () // => 5Ao trabalhar com código como este que cria vários fechamentos usando umLoop, é um erro comum tentar mover o loop dentro da função

Erro ao traduzir esta página.

Para todas as outras iterações, e cada um desses escopos tem seu próprio ligação independente de i. Outra coisa a lembrar ao escrever o fechamento é que este é um Palavra-chave JavaScript, não uma variável. Como discutido anteriormente, Arrowfunções herdam o valor deste da função que as contém, mas Funções definidas com a palavra-chave FUNCIONAL Não. Então, se você é escrevendo um fechamento que precisa usar o valor deste valorfunção, você deve usar uma função de seta ou chamada bind (), no fechamento antes de devolvê-lo ou atribuir esse valor externo a uma variável que seu fechamento herdará: const self = this; // Torne este valor disponível para funções aninhadas.

8.7 Propriedades, métodos e métodos da função Construtor Vimos que as funções são valores nos programas JavaScript. O operador do tipo de retorna a string "função" quando aplicado a um função, mas as funções são realmente um tipo especializado de javascriptobjeto. Como as funções são objetos, eles podem ter propriedades em métodos, como qualquer outro objeto. Existe até uma função () construtor para criar novos objetos de função. As subseções a seguir documentar as propriedades de comprimento, nome e protótipo; o Call (), Métodos Aplicar (), Bind () e ToString (); e o Function () construtor.

8.7.1 A propriedade de comprimento

A propriedade de comprimento somente leitura de uma função especifica a aridade da função - o número de parâmetros que declara em sua lista de parâmetros, que geralmente é o número de argumentos que a função espera. Se a função tem um parâmetro de descanso, esse parâmetro não é contado para fins desta propriedade de comprimento.

8.7.2 A propriedade NomeA propriedade Nome somente leitura de uma função especifica o nome que foi usado quando a função foi definida, se foi definida com um nome, ou o nome da variável ou propriedade que uma função sem nomeA expressão foi atribuída quando foi criada pela primeira vez. Esta propriedade é principalmente útil ao escrever mensagens de depuração ou erro.

8.7.3 A propriedade do protótipo Todas as funções, exceto as funções de seta, têm uma propriedade de protótipo que se refere a um objeto conhecido como objeto de protótipo. Toda função tem um objeto de protótipo diferente. Quando uma função é usada como um construtor, o objeto recém-criado herda as propriedades do Objeto de protótipo. Protótipos e a propriedade do protótipo foram discutidos no §6.2.3 e será coberto novamente no capítulo 9.

8.7.4 Os métodos Call () e Apply () Call () e Apply () permitem que você invocar indiretamente (§8.2.4) A Funcionar como se fosse um método de algum outro objeto. O primeiro argumento para Call () e Apply () é o objeto em que a função será invocada; Este argumento é o contexto de invocação e se torna o valor dessa palavra-chave dentro do corpo da função. Para invocar

a função f () como um método do objeto O (não passando argumentos),Você pode usar Calling () ou Apply ():F.Call (O);F.Apply (O);Qualquer uma dessas linhas de código é semelhante ao seguinte (que assumem que O ainda não possui uma propriedade chamada m):o.m = f;// Faça f um método temporário de o.O.M ()// Invoca, não passando sem argumentos.excluir o.m;// Remova o método temporário.Lembre -se de que as funções de seta herdam o valor deste contexto onde eles são definidos.Isso não pode ser substituído com a chamada () e aplicar () métodos.Se você ligar para qualquer um desses métodos em umFunção de seta, o primeiro argumento é efetivamente ignorado.Quaisquer argumentos a serem chamados () após o primeiro argumento de contexto de invocações são os valores que são passados ??para a função que é invocada (e estesArgumentos não são ignorados para funções de seta).Por exemplo, para passar dois números para a função f () e invocam como se fosse um método do objeto O, você pode usar o código assim:F.Call (O, 1, 2);O método Aplicar () é como o método Call (), exceto que os argumentos a serem passados ??para a função são especificados como uma matriz:f.Apply (O, [1,2]);

Se uma função é definida para aceitar um número arbitrário de argumentos, o Método Aplicar () permite invocar essa função no conteúdo de uma variedade de comprimento arbitrário. Em ES6 e mais tarde, podemos apenas usar o Operador espalhado, mas você pode ver o código ES5 que usa aplicar () em vez de. Por exemplo, para encontrar o maior número em uma variedade de números Sem usar o operador de spread, você pode usar o método Apply () Para passar os elementos da matriz para a função Math.max (): deixe maior = math.max.apply (matemática, Arrayofnumbers); A função Trace () definida a seguir é semelhante ao método timed () definida em §8.3.4, mas funciona para métodos em vez disso de funções. Ele usa o método Apply () em vez de um operador de spread, E ao fazer isso, é capaz de invocar o método embrulhado com os mesmos argumentos e o mesmo valor que o método Wrapper:// Substitua o método chamado m do objeto O com uma versão isso registra// mensagens antes e depois de invocar o método original. Trace da função (O, M) {Seja original = o [m];// Lembre -se do método original no fechamento.o [m] = função (... args) {// agora define o novo método.console.log (new Date (), "Entrando:", M);// Registre mensagem.deixe o resultado = original.Apply (isto, args); // Invoca o original.console.log (new Date (), "saindo:", m); // Registre mensagem.resultado de retorno; // Resultado de retorno.};}

8.7.5 O método bind ()O objetivo principal de Bind () é vincular uma função a um objeto.Quando você invoca o método bind () em uma função f e passa umObjeto O, o método retorna uma nova função.Invocando a nova função(em função) Invoca a função original f como um método de O.QualquerArgumentos que você passa para a nova função são passados ??para o originalfunção.Por exemplo:função f (y) {return this.x + y;} // Esta função precisapara ser amarradoSeja o = {x: 1};// um objeto que vamos vincularparaSeja g = f.bind (o);// chamando g (x) chama f () em og (2) // => 3Seja p = {x: 10, g};// invocar G () como umMétodo deste objetcop.g (2) // => 3: g ainda éligado a O, não p.Funções de seta herdam seu valor a partir do ambiente emque eles são definidos, e esse valor não pode ser substituído combind (), portanto, se a função f () no código anterior foi definida comoUma função de seta, a ligação não funcionaria.O uso mais comumCaso para chamadas bind () é fazer com que as funções não se comportem comoFunções de seta, no entanto, então essa limitação nas funções de seta de ligação não é um problema na prática.O método bind () faz mais do que apenas vincular uma função a um objeto,no entanto.Também pode executar a aplicação parcial: quaisquer argumentos que vocêPasse para Bind () Após o primeiro, estão vinculados junto com este valor.

Erro ao traduzir esta página.

que pode ser usado para criar novas funções:`const f = nova função ("x", "y", "return x*y;");`Esta linha de código cria uma nova função que é mais ou menos equivalentePara uma função definida com a sintaxe familiar:`const f = função (x, y) {return x*y;};`O construtor `function ()` espera qualquer número de stringarguments.O último argumento é o texto do corpo da função;podeconter declarações arbitrárias de javascript, separadas uma da outraSemicolons.Todos os outros argumentos para o construtor são strings queEspecifique os nomes dos parâmetros para a função.Se você está definindo umfunção que não leva argumentos, você simplesmente passaria uma única string- O corpo da função - para o construtor.Observe que o construtor `function ()` não foi aprovado em nenhum argumentolssso especifica um nome para a função que ele cria.Como literais da função,O construtor `function ()` cria funções anônimas.Existem alguns pontos que são importantes para entender sobre o`Function ()` construtor:O construtor `function ()` permite que as funções JavaScriptser criado dinamicamente e compilado em tempo de execução.O construtor da função () analisa o corpo da função ecria um novo objeto de função cada vez que é chamado.Se a chamadapara o construtor aparece dentro de um loop ou dentro de um frequentementeFunção chamada, esse processo pode ser ineficiente.Por outro lado,

Erro ao traduzir esta página.

eles mesmos particularmente bem para um estilo de programação funcional. OSeções a seguir demonstram técnicas para funcionalProgramação em JavaScript. Eles são destinados a uma expansão da menteExploração do poder das funções do JavaScript, não como uma receitaPara um bom estilo de programação.8.8.1 Matrizes de processamento com funçõesSuponha que tenhamos uma variedade de números e queremos calcular oDesvio médio e padrão desses valores.Podemos fazer isso emEstilo não funcional como este:deixe dados = [1,1,3,5,5];// Esta é a nossa variedade de números// a média é a soma dos elementos divididos pelo número de elementosDeixe total = 0;para (vamos i = 0; i <data.length; i ++) total+= dados [i];deixe mean = total/data.length;// média == 3;A média do nossoOs dados são 3// Para calcular o desvio padrão, primeiro somamos o quadrados de// O desvio de cada elemento da média.total = 0;para (vamos i = 0; i <data.length; i ++) {Deixe o desvio = dados [i] - Mean;total += desvio * desvio;}Seja stddev = math.sqrt (total/(data.length-1));// stddev ==2Podemos executar esses mesmos cálculos em estilo funcional concisousando os métodos de matriz map () e reduz () como este (consulte §7.8.1 paraRevise esses métodos):

// Primeiro, defina duas funções simplesconst sum = (x, y) => x+y;const square = x => x*x;// então use essas funções com métodos de matriz para calcularMédia e Stddevdeixe dados = [1,1,3,5,5];deixe mean = data.reduce (sum) /data.length;// média == 3Deixe os desvios = data.map (x => x-mean);Seja stddev =Math.sqrt (desvioss.map (quadrado) .Reduce (soma)/(data.length-1));stddev // => 2Esta nova versão do código parece bem diferente do primeiro,mas ainda está chamando métodos em objetos, por isso tem algum objetoConvenções orientadas restantes.Vamos escrever versões funcionais doMAP () e Reduce () Métodos:const map = function (a, ... args) {return a.map (... args);};const reduz = função (a, ... args) {returnA.Reduce (... args);};Com estes MAP () e Reduce () funções definidas, nosso código paraCalcule a média e o desvio padrão agora se parece com o seguinte:const sum = (x, y) => x+y;const square = x => x*x;deixe dados = [1,1,3,5,5];Deixe mean = redução (dados, soma) /data.length;Deixe os desvios = mapa (dados, x => x-mean);Seja stddev = math.sqrt (redução (mapa (desvios, quadrado),soma)/(data.length-1));stddev // => 28.8.2 Funções de ordem superior

Uma função de ordem superior é uma função que opera em funções, tomando uma ou mais funções como argumentos e retornando uma nova função. Aqui é um exemplo:// Esta função de ordem superior retorna uma nova função que passa// argumentos para f e retorna a negação lógica de f's valor de retorno; função não (f) {Retornar função (... args) {// retorna um novo funçãoLet Result = F.Apply (isto, args); // que chama f retornar! Resultado; // e nega seu resultado.};}const mesmo = x => x % 2 === 0;// uma função para determinar se um número é mesmoconst ímpar = não (par); // uma nova função que faz o oposto[1,1,3,5,5] .a todos (ímpares) // => true: todo elemento de A matriz é estranhaEsta função não () é uma função de ordem superior porque é necessário um Função Argumento e retorna uma nova função.Como outro exemplo, Considere a função Mapper () que se segue. É preciso uma função argumento e retorna uma nova função que mapeia uma matriz para outra usando essa função. Esta função usa a função map () definida antes, e é importante que você entenda como as duas funções são diferentes:// retorna uma função que espera um argumento de matriz e aplica -se a// Cada elemento, retornando a matriz de valores de retorno.// Contraste isso com a função map () de anteriores.função mapeador (f) {

```
retornar a => mapa (a, f);}const increment = x => x+1;const incrementall = mapeador (incremento);incremental ([1,2,3]) // => [2,3,4]Aqui está outro exemplo mais geral que leva duas funções, f eG, e retorna uma nova função que calcula F (G ()):// retorna uma nova função que calcula F (G (...)).// A função retornada H passa todos os seus argumentos para G,então passa// O valor de retorno de G para F, depois retorna o valor de retornodesligado.// ambos F e G são invocados com o mesmo valor que H foiinvocado com.função compõe (f, g) {Função de retorno (... args) {// usamos Call para F porque estamos passando um únicovalor e// solicita -se para G porque estamos passando uma variedade devalores.retornar f.call (this, g.apply (this, args));};}const sum = (x, y) => x+y;const square = x => x*x;compor (quadrado, soma) (2,3) // => 25;o quadrado da somaAs funções parciais () e memorize () definidas nas seçõesA seguir, são duas funções mais importantes de ordem superior.8.8.3 Aplicação parcial de funçõesO método bind () de uma função f (ver §8.7.5) retorna um novofunção que chama F em um contexto especificado e com um conjunto especificado
```

de argumentos.Dizemos que isso liga a função a um objeto eaplica parcialmente os argumentos.O método bind () aplica parcialmenteArgumentos à esquerda - isto é, os argumentos que você passa para vincular () sãocolocado no início da lista de argumentos que é passada para o originalfunção.Mas também é possível aplicar parcialmente argumentos nocerto:// Os argumentos para esta função são passados ??à esquerdafunction parcialleft (f, ... Outerargs) {Retornar função (... innerargs) {// retorna esta funçãoSeja args = [... Outerargs, ... innerargs];// construa oLista de argumentosretornar F.Apply (isto, args);// EntãoInvoca F com isso};}// Os argumentos para esta função são passados ??à direitafunction parcialright (f, ... Outerargs) {Retornar função (... innerargs) {// retorna esta funçãoSeja args = [... innerargs, ... Outerargs];// construa oLista de argumentosretornar F.Apply (isto, args);// EntãoInvoca F com isso};}// Os argumentos a essa função servem como um modelo.Valores indefinidos// Na lista de argumentos, são preenchidos com valores doConjunto interno.function parcial (f, ... Outerargs) {Função de retorno (... Innerargs) {Seja args = [... Outerargs];// cópia local da externaModelo de argsDeixe InnerIndex = 0;// que arg interno é o próximo// percorre os args, preenchendo valores indefinidosdo args internopara (vamos i = 0; i <args.length; i ++) {

Erro ao traduzir esta página.

Também podemos usar composição e aplicação parcial para refazer nossas médias e cálculos de desvio padrão em estilo funcional extremo:// As funções Sum () e Square () são definidas acima.Aqui estou um pouco mais:const
produto = (x, y) => x*y;const neg = parcial (produto, -1);const sqrt = parcial (math.pow, indefinido, .5);const
reciprocal = parcial (Math.pow, indefinido, neg (1));// Agora calcule a média e o desvio padrão.deixe dados =
[1,1,3,5,5];// nossos dadosDeixe mean = produto (redução (dados, soma),recíproco (data.length));Seja stddev =
sqrt (Produto (Reduce (mapa (dados,compor (quadrado,parcial (soma,neg (média))))),soma),recíproco (soma
(data.length, neg (1))));[média, stddev] // => [3, 2]Observe que este código para calcular a média e o desvio
padrão é invocações inteiramente funcionam;não há operadores envolvidos e o Número de parênteses ficou tão
grande que este JavaScript é Começando a se parecer com o código LISP.Novamente, este não é um estilo que
eu Advogado pela programação JavaScript, mas é um exercício interessantePara ver como o código JavaScript
profundamente funcional pode ser.8.8.4 MEMOIZAÇÃOEm §8.4.1, definimos uma função factorial que em cache
seu anteriormente resultados calculados.Na programação funcional, esse tipo de cache é chamado de memórias.O
código a seguir mostra uma ordem superior

função, memoize (), que aceita uma função como seu argumento eRetorna uma versão memorada da função:// retorna uma versão memorada de f.// só funciona se os argumentos para f todos tiverem string distintarepresentações.Função Memoize (f) {const cache = new map ():// cache de valor armazenado noencerramento.Função de retorno (... args) {// Crie uma versão de string dos argumentos para usar comouma chave de cache.deixe a chave = args.length + args.join (" +");if (cache.has (key)) {retornar cache.get (chave);} outro {Let Result = F.Apply (isto, args);cache.set (chave, resultado);resultado de retorno;}};}A função Memoize () cria um novo objeto a ser usado como cache eatribui esse objeto a uma variável local para que seja privada (nofechamento de) a função retornada.A função retornada converte seuArgumentos Array em uma string e usa essa string como um nome de propriedade parao objeto de cache.Se existir um valor no cache, ele o retornará diretamente.Caso contrário, ele chama a função especificada para calcular o valor para estesArgumentos, armazena em cache esse valor e o devolve.Aqui está como podemos usarMemoize ():// retorna o maior divisor comum de dois números inteiros usandooo euclidiano// algoritmo:

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Capítulo 9. Classes Os objetos JavaScript foram abordados no capítulo 6. Esse capítulo tratou cada objeto como um conjunto único de propriedades, diferente de todos os outros objetos. Isto geralmente é útil, no entanto, definir uma classe de objetos que compartilham certas propriedades. Membros, ou instâncias, da classe têm seus próprios propriedades para manter ou definir seu estado, mas também têm métodos que definir seu comportamento. Esses métodos são definidos pela classe e compartilhado por todas as instâncias. Imagine uma classe chamada complexo que representa e executa aritmética em números complexos, por exemplo. Um complexo a instância teria propriedades para manter as partes reais e imaginárias (o estado) do número complexo. E a classe complexa definiria métodos para realizar adição e multiplicação (o comportamento) daqueles números. No JavaScript, as classes usam herança baseada em protótipo: se dois objetos herdam propriedades (geralmente propriedades com valor de função ou métodos) do mesmo protótipo, então dizemos que esses objetos são casos de a mesma classe. Em poucas palavras, é assim que as classes JavaScript funcionam. Protótipos e herança JavaScript foram cobertos em §6.2.3 e §6.3.2, e você precisará estar familiarizado com o material naqueles seções para entender este capítulo. Este capítulo abrange protótipos em §9.1. Se dois objetos herdam do mesmo protótipo, este tipicamente (mas não necessariamente) significa que eles foram criados e inicializados pelo mesmo

função construtora ou função de fábrica. Construtores foram cobertos em §4.6, §6.2.2 e §8.2.3, e este capítulo tem mais no §9.2. O JavaScript sempre permitiu a definição de classes. ES6 introduziu uma sintaxe totalmente nova (incluindo uma palavra-chave de classe) que a torna uniforme e mais fácil de criar classes. Essas novas classes JavaScript funcionam da mesma forma como as aulas de estilo antigo, e este capítulo começa explicando o maneira antiga de criar classes porque isso demonstra mais claramente o que está acontecendo nos bastidores para fazer as aulas funcionarem. Uma vez que fizemos explicações desses fundamentos, mudaremos e começaremos a usar o novo. Sintaxe da definição de classe simplificada. Se você está familiarizado com a programação de objetos fortemente digitados em idiomas como Java ou C++, você notará que as classes JavaScript são muito diferentes das classes nessas línguas. Existem algumas semelhanças sintáticas, e você pode imitar muitos recursos de "clássico" aulas em JavaScript, mas é melhor entender antecipadamente que classes de JavaScript têm um mecanismo de herança baseado em protótipo, que é substancialmente diferente das classes e herança baseada em classes de Java e idiomas semelhantes.

9.1 Classes e protótipos

Em JavaScript, uma classe é um conjunto de objetos que herdam propriedades e métodos de um mesmo objeto de protótipo. O objeto de protótipo, portanto, é o central característica de uma classe. Capítulo 6 Cobriu o objeto Create () função que retorna um objeto recém-criado que herda de um objeto de protótipo especificado. Se definirmos um objeto de protótipo e depois usarmos Object.create () para criar objetos que herdem dele, temos

definiu uma classe JavaScript. Geralmente, as instâncias de uma classe exigem mais inicialização, e é comum definir uma função que crie e inicializa o novo objeto. Exemplo 9-1 demonstra o seguinte:

```
define um objeto de protótipo para uma classe que representa uma variedade de valores e também define uma função de fábrica que cria e inicializa um novo objeto da classe.
```

Exemplo 9-1. Uma classe JavaScript simples// Esta é uma função de fábrica que retorna um novo objeto de intervalo.

```
intervalo de funções (de, a) { // use object.create () para criar um objeto que herda o Objeto de protótipo definido abaixo. O objeto de protótipo é armazenado como uma propriedade desta função e define o compartilhadoMétodos (comportamento) para todos os objetos de intervalo. Seja r = object.create (range.methods); // Armazene os pontos de partida e final (estado) desta nova linha de código. // Estas são propriedades não herdadas que são exclusivas para este objeto. r.from = de; r.to = para; // finalmente retorna o novo objeto para retornar r; } // Este objeto de protótipo define métodos herdados por toda a faixa de objetos. range.methods = { // retorna true se x estiver no intervalo, false caso contrário // Este método funciona para intervalos textuais e de data, bem como numéricos. inclui (x) { return this.from <= x && x <= this.to; }, // uma função de gerador que torna as instâncias da classe iterável. // Observe que ele funciona apenas para intervalos numéricos.
```

*[Symbol.iterator] () {para (vamos x = math.ceil (this.from); x <= this.to; x ++)rendimento x;}// retorna uma representação de string do intervalo.toString () {return "(" + this.from + "..." + this.to +")";};}// Aqui estão os usos de exemplo de um objeto de intervalo.Seja r = intervalo (1,3);// Crie um objeto de intervalo.includes (2) // => true: 2 está no intervalo.toString () // => "(1 ... 3"[... r] // => [1, 2, 3];converter em uma matrizvia iteradorHá algumas coisas que valem a pena notar no Código do Exemplo 9-1:Este código define um intervalo de função de fábrica () para criarnovos objetos de intervalo.Ele usa a propriedade Métodos desta Função ()local conveniente para armazenar o objeto de protótipo que define oaula.Não há nada de especial ou idiomático em colocar oObjeto de protótipo aqui.A função range () define de e para propriedades emcada objeto de intervalo.Estes são os não -compartilhados, não herdadospropriedades que definem o estado único de cada indivíduoObjeto de alcance.O objeto Range.Methods usa a sintaxe de abreviação ES6para definir métodos, e é por isso que você não vê oFunção de palavra -chave em qualquer lugar.(Ver §6.10.5 para revisar o objetoSintaxe do método de abreviação literal.)Um dos métodos no protótipo tem o nome calculado(§6.10.2) Symbol.iterator, o que significa que é

Definindo um iterador para objetos de intervalo. O nome disso é prefixado com *, o que indica que é um método do gerador em vez de uma função regular. Iteradores e os geradores são cobertos em detalhes no capítulo 12. Por enquanto, o Upshot é que as instâncias dessa classe de intervalo podem ser usadas como loop for/of e com o ... Spread Operator. Os métodos compartilhados e herdados definidos em range.methodsTodos usam as propriedades de e para as propriedades que foram inicializadas na função de fábrica Range (). Para se referir a eles, eles usam a palavra -chave para se referir ao objeto através do qual eles foram invocados. Este uso disso é fundamental característica dos métodos de qualquer classe.

9.2 Classes e Construtores

O exemplo 9-1 demonstra uma maneira simples de definir uma classe JavaScript. Isto não é a maneira idiomática de fazê-lo, no entanto, porque não definiu um construtor. Um construtor é uma função projetada para a inicialização de objetos recém-criados. Os construtores são invocados usando a nova palavra-chave conforme descrito em

§8.2.3. Invocações construtoras

usando novaCrie automaticamente o novo objeto, para que o próprio construtor só precisa para inicializar o estado desse novo objeto. A característica crítica das invocações de construtor é que a propriedade do protótipo do construtor é usado como o protótipo do novo objeto.

§6.2.3. Protótipos

apresentou protótipos e enfatizou que, embora quase todos os objetos tenham um protótipo, apenas alguns objetos têm uma propriedade de protótipo. Finalmente, podemos esclarecer isso: são objetos de função que têm uma propriedade do protótipo. Isso significa que todos os objetos criados com a mesma função do construtor herda do mesmo objeto e são

Portanto, membros da mesma classe.Exemplo 9-2 mostra como não poderia alterar a classe de gama de exemplo 9-1 para usar um construtor função em vez de uma função de fábrica.Exemplo 9-2 demonstra maneira idiomática de criar uma classe em versões de javascript que não apoia a palavra -chave da classe ES6.Mesmo que a classe esteja bem suportada agora, ainda há muitos código JavaScript mais antigos em torno dissó define classes como essa, e você deve estar familiarizado com o idioma que você pode ler código antigo e para entender o que está acontecendo "Under the Hood" quando você usa a palavra -chave da classe.Exemplo 9-2.Uma classe de alcance usando um construtor// Esta é uma função construtora que inicializa o novo intervalo de objetos.// Observe que ele não cria ou retorna o objeto.Apenas inicializa isso.intervalo de funções (de, a) { // Armazene os pontos de partida e final (estado) desta nova linha de objeto.// Estas são propriedades não herdadas que são exclusivas para este objeto.this.from = de;this.to = para;}// Todos os objetos do intervalo herdam deste objeto.// Observe que o nome da propriedade deve ser "protótipo" para isso trabalhar.Range.prototype = { // retorna true se x estiver no intervalo, false caso contrário// Este método funciona para intervalos textuais e de data, bem como numérico.Inclui: function (x) {return this.from <= x && x <= this.to;},// uma função de gerador que torna as instâncias da classe iterável.// Observe que ele funciona apenas para intervalos numéricos.[Symbol.iterator]: function*() {

para (vamos x = math.ceil (this.from); x <= this.to; x ++)rendimento x;}// retorna uma representação de string do intervaloToString: function () {return "(" + this.from + "..." +this.to + ")";};// Aqui estão os usos de exemplo desta nova classeSeja r = novo intervalo (1,3);// Crie um objeto de intervalo;Observe uso de novor.includes (2) // => true: 2 está no intervalo.toString () // => "(1 ... 3"[... r] // => [1, 2, 3];converter em uma matrizvia iteradorVale a pena comparar exemplos 9-1 e 9-2 com bastante cuidado e observarAs diferenças entre essas duas técnicas para definir classes.Primeiro,Observe que renomeamos a função de fábrica Range () para Range ()Quando o convertemos em um construtor.Esta é uma codificação muito comumConvenção: as funções do construtor definem, em certo sentido, classes eAs aulas têm nomes que (por convenção) começam com letras maiúsculas.Funções e métodos regulares têm nomes que começam com minúsculascartas.Em seguida, observe que o construtor range () é invocado (no final de exemplo) com a nova palavra -chave enquanto a fábrica range ()A função foi invocada sem ela.Exemplo 9-1 usa função regularinvocação (§8.2.1) para criar o novo objeto e o exemplo 9-2 usaInvocação do construtor (§8.2.3).Porque o construtor range () éinvocado com novo, ele não precisa chamar object.create () ouTome qualquer ação para criar um novo objeto.O novo objeto é automaticamentecriado antes que o construtor seja chamado, e é acessível como o

valor. O construtor range () apenas precisa inicializar isso. Os construtores nem precisam devolver o objeto recém-criado. A invocação do construtor cria automaticamente um novo objeto, chama o construtor como um método desse objeto e retorna o novo objeto. Ofato de que a invocação do construtor é tão diferente da função regular A invocação é outra razão pela qual damos nomes de construtores que começam com letras maiúsculas. Os construtores são escritos para serem invocados como construtores, com a nova palavra -chave, e eles geralmente não funcionam adequadamente se forem chamados como funções regulares. Uma convenção de nomenclatura lasso mantém as funções do construtor distintas das funções regulares ajudamOs programadores sabem quando usar novos. Construtores e New. Target Dentro de um corpo de função, você pode dizer se a função foi invocada como um construtor com o expressão especial new. Target. Se o valor dessa expressão for definido, você sabe que a função foi invocada como construtor, com a nova palavra -chave. Quando discutirmos subclasses em §9.5, Veremos que o new. Target nem sempre é uma referência ao construtor em que é usado: ele também pode se referir à função construtora de uma subclasse. Se new. Target for indefinido, a função contendo foi invocada como uma função, sem nova palavra -chave. Os vários construtores de erro do JavaScript podem ser invocados sem novos, e se você quiser Emular esse recurso em seus próprios construtores, você pode escrevê-los assim: função c () {if (! New. Target) retorna novo C (); // O código de inicialização vai aqui} Essa técnica funciona apenas para construtores definidos dessa maneira antiquada. Classes criadas com oA palavra -chave de classe não permite que seus construtores sejam invocados sem novos. Outra diferença crítica entre os exemplos 9-1 e 9-2 é o caminho O objeto de protótipo é nomeado. No primeiro exemplo, o protótipo foi range. methods. Este foi um nome conveniente e descriptivo, mas

arbitrário. No segundo exemplo, o protótipo é Range.prototype, e esse nome é obrigatório. Uma invocação do construtor range () usa automaticamente range.prototype como protótipo do novo objeto de intervalo. Finalmente, observe também as coisas que não mudam entre os exemplos 9-1 e 9-2: os métodos de alcance são definidos e invocados da mesma maneira para ambas as classes. Porque o exemplo 9-2 demonstra a maneira idiomática para criar classes nas versões do JavaScript antes do ES6, ele não usa a sintaxe do método de abreviação ES6 no objeto de protótipo e explicitamente coloca os métodos com a palavra -chave da função. Mas você pode ver que a implementação dos métodos é a mesma nos dois exemplos. É importante ressaltar que nenhum dos dois exemplos de intervalo usa seta para definir construtores ou métodos. Lembre -se de §8.1.3: essas funções definidas dessa maneira não têm uma propriedade de protótipo assim não pode ser usado como construtores. Além disso, as funções de seta herdam essa palavra -chave do contexto em que são definidos em vez de definir -lo com base no objeto através do qual eles são invocados, e este torna inúteis para métodos porque a característica definidora de métodos é que eles usam isso para se referir à instância em que eles foram invocados. Felizmente, a nova sintaxe da classe ES6 não permite a opção de definir métodos com funções de seta, então isso não é um erro que você pode fazer acidentalmente ao usar essa sintaxe. Vamos cobrir o ES6 Palavra -chave da classe em breve, mas primeiro, há mais detalhes para cobrir sobre construtores.

9.2.1 Construtores, identidade de classe e instância

Como vimos, o protótipo objeto é fundamental para a identidade de umClasse: Dois objetos são casos da mesma classe se e somente se elesherdar do mesmo objeto de protótipo. A função do construtor quelnicializa o estado de um novo objeto não é fundamental: dois construtoresfunções podem ter propriedades de protótipo que apontam para o mesmoObjeto de protótipo. Então, ambos os construtores podem ser usados ??para criarInstâncias da mesma classe. Embora os construtores não sejam tão fundamentais quanto protótipos, oO construtor serve como face pública de uma classe. Mais obviamente, o nome da função do construtor é geralmente adotado como o nome daula. Dizemos, por exemplo, que o construtor range () criaObjetos de alcance. Mais fundamentalmente, no entanto, os construtores são usados ??como operando areia do operador da instância ao testarobjetos para associação a uma classe. Se tivermos um objeto r e queremosSaiba se é um objeto de intervalo, podemos escrever:r Instância de intervalo // => true: r herda deRange.prototypeA instância do operador foi descrita em §4.9.4. A esquerdaoperando deve ser o objeto que está sendo testado, e o caminhoOperando deve ser uma função construtora que nomeia uma classe. Oexpressão o instância de c avalia como verdadeiro se o herdarC.Prototipo. A herança não precisa ser direta: se o herdar deum objeto que herda de um objeto que herdaC.Prototipo, a expressão ainda será avaliada como verdadeira.

Erro ao traduzir esta página.

No Exemplo 9-2, definimos range.prototype para um novo objeto que continha os métodos para a nossa classe. Embora fosse conveniente expressar esses métodos como propriedades de um único objeto literal, não é realmente necessário para criar um novo objeto. Qualquer função regular (excluindo funções de seta, funções geradoras e assíncronas) podem ser usadas como construtor e invocações de construtor precisam de uma propriedade de protótipo. Portanto, todo JavaScript regular função automaticamente possui uma propriedade de protótipo. O valor disso é que é um objeto que possui um único construtor e não é enumerável. A propriedade do construtor é a função: Seja `f = function () {};` // Este é um objeto de função. Seja `p = f.prototype;` // Este é o objeto de protótipo associado a F. Seja `c = p.Constructor;` // Esta é a função associada com o protótipo. `c === f` // => true: `f.prototype.constructor === f` para qualquer f. A existência deste objeto de protótipo predefinido com sua propriedade construtora significa que os objetos normalmente herdam uma propriedade do construtor que se refere ao seu construtor. Desde que os construtores servem como identidade pública de uma classe, este construtorA propriedade fornece a classe de um objeto: Seja `o = novo f();` // Crie um objeto o da classe F. `o.Constructor === f` // => true: a propriedade do construtor especifica a classe. A Figura 9-1 ilustra essa relação entre a função do construtor, seu objeto de protótipo, a referência traseira do protótipo para o1

Erro ao traduzir esta página.

Constructor

Range()

prototype

Prototype

constructor
includes: ...
toString: ...

Instances

new Range(1,2)

new Range(3,4)



assim:// estende o range predefinido.prototype objeto, então não substituir// O Range.prototype automaticamente criado propriedade.Range.prototype.includes = function (x) {Retorne this.from <= x && x <= this.to;};Range.prototype.toString = function () {retornar "(" + this.from + "..." + this.to + ")";};9.3 Classes com a palavra -chave da classeAs aulas fazem parte do JavaScript desde a primeira versão da linguagem, mas no ES6, eles finalmente obtiveram sua própria sintaxe com a introdução da palavra -chave da classe.Exemplo 9-3 mostra o que nossoA classe Range parece quando escrita com esta nova sintaxe.Exemplo 9-3.A classe Range reescrita usando a classe intervalo de classe {construtor (de, para) {// armazenar os pontos de partida e final (estado) deste novo objeto de alcance.// estas são propriedades não herdadas que são exclusivas para este objeto.this.from = de;this.to = para;}// retorna true se x estiver no intervalo, false caso contrário// Este método funciona para intervalos textuais e de data, bem como numérico.inclui (x) {return this.from <= x && x <= this.to;}// uma função de gerador que torna as instâncias da classe iterável.

Erro ao traduzir esta página.

um do outro.(Embora os corpos de classe sejam superficialmente semelhantes aos literais de objetos, eles não são a mesma coisa. Em particular, eles não suportam a definição de propriedades comparáveis de nome/valor.) O construtor de palavras -chave é usado para definir o construtor da função para a classe. A função definida não é realmente nomeada "construtor", no entanto. A declaração de classeA instrução define um novo intervalo de variáveis ??e atribui o valor desta função especial ao construtor nessa variável. Se sua classe não precisar fazer nenhuma inicialização, você pode omitir a palavra -chave do construtor e seu corpo, e um vazioA função construtora será criada implicitamente para você. Se você deseja definir uma classe que subclasse - ou herda -Outra aula, você pode usar a palavra -chave Extends com a classePalavra -chave:// uma extensão é como um intervalo, mas em vez de inicializá-lo com um começo e um fim, inicializamos com um início e um comprimentoA classe Span se estende pelo intervalo {construtor (start, comprimento) {if (comprimento > = 0) {super (start, start + comprimento);} outro {super (start + comprimento, start);}}}Criar subclasses é um tópico completo. Voltaremos a isso eExplique as extensões e as palavras -chave mostradas aqui, em §9.5.

Como declarações de função, as declarações de classe têm afirmação e formas de expressão. Assim como podemos escrever: Deixe Square = function (x) {return x * x;}; quadrado (3) // => 9 Também podemos escrever: Let Square = classe {construtor (x) {this.area = x * x;}}; Novo quadrado (3) .Area // => 9 Como nas expressões de definição de função, expressões de definição de classe podem incluir um nome de classe opcional. Se você fornecer esse nome, isso o nome é definido apenas dentro do próprio corpo da classe. Embora as expressões de função sejam bastante comuns (particularmente com a função de seta abreviação), em programação JavaScript, definição de classe e expressões não são algo que você provavelmente usará muito, a menos que você se vê escrevendo uma função que faz uma aula como argumento e retorna uma subclasse. Concluiremos esta introdução à palavra-chave de classe mencionando algumas coisas importantes que você deve saber que não são aparentes da sintaxe da classe: Todo o código dentro do corpo de uma declaração de classe é implicitamente no modo rigoroso (§5.6.3), mesmo que não haja diretiva "use" rigorosa "aparece". Isso significa, por exemplo, que você não pode usar octal literais inteiros ou a declaração com os órgãos de classe e é mais provável que você obtenha erros de sintaxe se esquecer de declarar uma variável antes de usá-la.

Ao contrário das declarações de função, as declarações de classe não são "lçado." Lembre -se de §8.1.1 que as definições de função se comportam como se tivessem sido movidos para o topo do arquivo fechado ou envolver a função, o que significa que você pode invocar uma função em código que vem antes da definição real da função. Embora as declarações de classe sejam como declarações de função em algumas maneiras, eles não compartilham esse comportamento de içanão pode instanciar uma aula antes de declará -la.

9.3.1 Métodos estáticos

Você pode definir um método estático dentro de um corpo de classe prefixando o Declaração do método com a palavra -chave estática. Métodos estáticos são definidos como propriedades da função do construtor em vez de propriedades do objeto de protótipo. Por exemplo, suponha que adicionamos o seguinte código ao Exemplo 9-3: análise estática (s) {Seja Matches = S.Match (/^ (\d+) \. \. (\d+) \\$/);if (! Matches) {Jogue o novo TypeError ('não pode analisar o alcance de "\$ {s}".'});}Retorne o novo alcance (parseint (fósforos [1]),parseint (correspondências [2]));}O método definido por este código é range.parse (), nãoRange.prototype.parse (), e você deve invocá -lo através do construtor, não através de uma instância: Seja r = range.parse ('(1 ... 10)');// retorna um novo intervalo objeto

R.Parse('((1 ... 10)');// TypeError: R.Parse não é uma função

Às vezes você vê métodos estáticos chamados métodos de classe porque eles são chamados usando o nome da classe/construtor. Quando este termo é usado, é para contrastar métodos de classe com os métodos de instância regulares que são invocados nas instâncias da classe. Porque métodos estáticos são invocados no construtor e não em qualquer instância em particular, ele quase nunca faz sentido usar essa palavra-chave em um método estático. Veremos exemplos de métodos estáticos no Exemplo 9-4.

9.3.2 Getters, Setters e outros formulários de método

Dentro de um corpo de classe, você pode definir métodos getter e setter (§6.10.6). Assim como você pode nos literais de objetos. A única diferença é que em corpos de classe, você não coloca vírgula após o getter ou setter. O exemplo 9-4 inclui um exemplo prático de um método getter em uma classe. Em geral, todas as sintaxes de definição de método abreviado permitidas em os literais de objetos também são permitidos em corpos de classe. Isso inclui geradores de métodos (marcados com *) e métodos cujos nomes são o valor de uma expressão entre colchetes. Na verdade, você já viu (em Exemplo 9-3) um método gerador com um nome calculado que faz a classe Range iterável: *[Symbol.iterator] () { para (vamos x = math.ceil (this.from); x <= this.to; x++) rendimento x; }

9.3.3 Campos públicos, privados e estáticos Na discussão aqui das aulas definidas com a palavra -chave da classe, nós descreveram apenas a definição de métodos dentro do corpo da classe. O padrão ES6 apenas permite a criação de métodos (incluindo getters, setters e geradores) e métodos estáticos; não inclui sintaxe para definir campos. Se você deseja definir um campo (que é apenas um sinônimo orientado a objetos para "propriedade") em uma instância de classe, você deve fazer isso na função do construtor ou em um dos métodos. E se você quer definir um campo estático para uma aula, você deve fazer isso fora do corpo de classe, depois que a aula foi definida. Exemplo 9-4 inclui exemplos de ambos os tipos de campos. A padronização está em andamento, no entanto, para uma sintaxe de classe estendida que permite a definição de instância e campos estáticos, tanto em público quanto em formulários privados. O código mostrado no restante desta seção ainda não está JavaScript padrão no início de 2020, mas já é apoiado em Chrome e parcialmente suportado (apenas campos de instância pública) no Firefox. A sintaxe para campos de instância pública é de uso comum por JavaScript programadores usando a estrutura do React e o transpiler Babel. Suponha que você esteja escrevendo uma aula como esta, com um construtor que inicializa três campos:

```
classe Buffer {constructor () {this.size = 0;this.capacity = 4096;this.Buffer = new Uint8Array (this.Capacity);}}
```

Com a nova sintaxe do campo de instância que provavelmente será padronizada, você em vez disso, poderia escrever:
classe Buffer {tamanho = 0;capacidade = 4096;buffer = novo uint8array (this.capacity);}O código de inicialização do campo saiu do construtor e agora aparece diretamente no corpo da classe.(Esse código ainda é executado como parte do Construtor, é claro.Se você não definir um construtor, os campos são inicializados como parte do construtor implicitamente criado.) O this.Prefixos que apareceram no lado esquerdo das tarefas desapareceram,Mas observe que você ainda deve usar isso.para se referir a esses campos, mesmo em o lado direito das atribuições de inicializador.A vantagem de inicializar seus campos de instância dessa maneira é que essa sintaxe permite(mas não exige) você para colocar os inicializadores no topo da Definição de classe, deixando claro para os leitores exatamente o que os campos manterão o estado de cada instância.Você pode declarar campos sem um inicializador apenas escrevendo o nome do campo seguido de um ponto e vírgula.Se você fizer isso, o valor inicial do campo será indefinido.É um estilo melhorPara sempre tornar o valor inicial explícito para todos os seus campos de classe.Antes da adição dessa sintaxe de campo, os corpos de classe pareciam muito parecidos Literais de objeto usando a sintaxe do método de atalho, exceto que as vírgulas foram removidas.Esta sintaxe de campo - com é igual a sinais de semicolons em vez de telas e vírgulas - deixa claro que as classesOs corpos não são iguais aos literais do objeto.

Erro ao traduzir esta página.

esse:estático integerRangePattern = /\^(d+)\.\.(d+)\)\$;/análise estática (s) {Seja Matches = S.Match (Range.IntegerRangePattern);if (! Matches) {Jogue o novo TypeError (' não pode analisar o alcance de "\$ {s}"). '}Retornar nova faixa (parseint (correspondências [1]), correspondências [2]);}Se quiséssemos que esse campo estático fosse acessível apenas dentro da classe, nós poderíamos torná-lo privado usando um nome como #pattern.9.3.4 Exemplo: uma classe de números complexosExemplo 9-4 Define uma classe para representar números complexos.A classe é relativamente simples, mas inclui métodos de instância (incluindo getters), métodos estáticos, campos de instância e campos estáticos.IncluiAlguns comentaram o código demonstrando como podemos usar o não-Syntaxe ainda padrão para definir campos de instância e campos estáticos dentro do corpo da classe.Exemplo 9-4.Complex.js: uma classe de números complexos/** As instâncias dessa classe complexa representam números complexos.* Lembre-se de que um número complexo é a soma de um número real e um número imaginário e que o número imaginário I é a raiz quadrada de -1.*/Complexo de classe {// Uma vez que as declarações de campo de classe são padronizadas, poderíamos declarar// campos privados para manter as partes reais e imaginárias de um

Erro ao traduzir esta página.

Erro ao traduzir esta página.

9.4 Adicionando métodos às classes existentesJavaScript's prototype-based inheritance mechanism is dynamic: anObjeto herda as propriedades de seu protótipo, mesmo que as propriedades deA mudança de protótipo após a criação do objeto. Isso significa que podemosAumentar as classes JavaScript simplesmente adicionando novos métodos a seusObjetos de protótipo. Aqui, por exemplo, é o código que adiciona um método para calcular oConjugado complexo com a complexa classe do Exemplo 9-4:// retorna um número complexo que é o conjugado complexo deEste.Complex.prototype.conj = function () {return novoComplexo (this.r, -This.i);};O protótipo objeto de classes JavaScript embutido também é aberto comolsso, o que significa que podemos adicionar métodos a números, cordas, matrizes,funções, e assim por diante. Isso é útil para implementar um novo idiomaRecursos nas versões mais antigas do idioma:// se o novo método da string startSwith () ainda não estiverdefinido ...if (! string.prototype.startswith) {// ... em seguida, defina assim usando o Índicef () mais antigo ()método.String.prototype.startswith = função (s) {return this.IndexOf (s) === 0;};}Aqui está outro exemplo:

// invoca a função f isso muitas vezes, passando oNúmero da iteração// por exemplo, para imprimir "Hello" 3 vezes:// deixe n = 3;// n.times (i => {console.log ('hello \$ {i}');});Número.prototype.Times = function (f, contexto) {Seja n = this.valueof ();para (vamos i = 0; i <n; i++) f.call (contexto, i);};Adicionando métodos aos protótipos de tipos internos como esse geralmente é considerado uma má ideia porque causará confusão e problemas de compatibilidade no futuro se uma nova versão do JavaScript define um método com o mesmo nome. É até possível adicionar métodos para objeto.Prototype, disponibilizando -os para todos os objetos. Mas isso nunca é uma coisa boa a fazer porque as propriedades adicionadas para o objeto.Prototype são visíveis para/in loops (embora você possa evitar isso usando object.DefineProperty () [§14.1] para retornar a nova propriedade não-anexável). 9.5 subclasses Na programação orientada a objetos, uma classe B pode estender ou subclasse Outra classe A. Dizemos que A é a superclasse e B é a subclasse. Instâncias de B herdando os métodos de A. A classe B pode definir seus próprios métodos, alguns dos quais podem substituir os métodos de mesmo nome definido pela classe A. Se um método de B substituir um método de A, o método de substituição em B geralmente precisa invocar o método substituído em A. Da mesma forma, o construtor da subclasse B () deve normalmente invocar o construtor da superclasse A () para garantir que as instâncias sejam completamente inicializadas.

Erro ao traduzir esta página.

```
// Verifique se o protótipo de extensão herda do intervalo
protoSpan.prototype = Object.create
(range.prototype); // não queremos herdar range.prototype.constructor, então nós
// Defina nossa própria
propriedade construtora.
Span.prototype.constructor = span; // Ao definir seu próprio método toString (), o span
substitui o// ToString () Método que ele herdaria de Faixa.
Span.prototype.toString = function () {retornar `($
{this.from} ... +$ {this.to - this.from})`}; Para criar uma subclasse de alcance, precisamos
providenciar Span.prototype para herdar do range.prototype. A chavelinha de código no exemplo anterior é este e se
faz sentido para você, você entende como as subclasses funcionam em JavaScript:
Span.prototype = Object.create
(range.prototype); Objetos criados com o construtor span () herdarão do Objeto span.prototype. Mas criamos esse
objeto para herdar de Range.prototype, então os objetos de span herdarão de ambos Span.prototype e
range.prototype. Você pode notar que nosso construtor de span () define o mesmo de epara as propriedades que o
construtor range () faz e, portanto, não precisaPara invocar o construtor range () para inicializar o novo objeto. Da
mesma forma, o método ToString () de Span reimplementam completamente o conversão de string sem precisar
chamar a versão do range de ToString (). Isso faz com que seja um caso especial, e só podemos realmente escape
com esse tipo de subclassificação porque sabemos o
```

Detalhes da implementação da superclasse.Uma subclasse robustaO mecanismo precisa permitir que as classes invocem os métodos econstrutor de sua superclasse, mas antes do ES6, JavaScript nãoTenha uma maneira simples de fazer essas coisas.Felizmente, o ES6 resolve esses problemas com a super palavra -chave como parte da sintaxe da classe.A próxima seção demonstra como funciona.9.5.2 subclasses com extensões e superNo ES6 e mais tarde, você pode criar uma superclasse simplesmente adicionando umestende a cláusula a uma declaração de classe, e você pode fazer isso mesmo paraAulas internas:// Uma subclasse de matriz trivial que adiciona getters para o primeiroe último elementos.classe EZARRAY estende a matriz {obtenha primeiro () {retornar este [0];}obtenha last () {return this [this.length-1];}}Seja a = new EZARRAY ();uma instância de ezarray // => true: a é uma instância da subclasseuma instância de matriz // => true: a também é uma superclasseexemplo.a.push (1,2,3,4); // A.Length == 4;Podemos usar herdadoMétodosa.pop () // => 4: outro método herdadoa. primeiro // => 1: primeiro getter definido porsubclasseA.Last // => 3: Último getter definido porsubclassea [1] // => 2: Sintaxe de acesso regular à matrizainda funciona.Array.isArray (a) // => true: a instância da subclasse realmente éuma matrizEzarray.isArray (a) // => true: subclasse herda estática

Métodos também! Esta subclasse EZARRAY define dois métodos simples de getter. Instâncias de Ezarray se comportarão como matrizes comuns, e podemos usar métodos e propriedades como push(), pop() e comprimento. Mas nós também pode usar o primeiro e o último getters definidos na subclasse. Não somente métodos de instância como pop() são herdados, mas métodos estáticos. Como o Array.isArray também são herdados. Este é um novo recurso ativado pela sintaxe da classe ES6: ezArray() é uma função, mas herda de Array():// ezarray herda métodos de instância porque EzArray.prototype // herda do Array.prototype. Array.isArray.isPrototypeOf(ezarray.prototype) // => true // e ezarray herda métodos e propriedades estáticas porque// EZARRAY herda da Array. Esta é uma característica especial deo// estende a palavra -chave e não é possível antes do ES6.Array.isArray.isPrototypeOf(ezarray) // => true. Nossa subclasse Ezarray é simples demais para ser muito instrutiva. Exemplo 9-6 é um exemplo mais completamente desenvolvido. Ele define uma subclasse de mapa digitoso de A classe de mapa embutida que adiciona verificação de tipo para garantir que as chaves os valores do mapa são dos tipos especificados (de acordo com typeof). É importante ressaltar que este exemplo demonstra o uso da palavra-chave super para invocar o construtor e os métodos do Superclass. Exemplo 9-6. Typedmap.js: uma subclasse do mapa que verifica a chave e tipos de valor. classe typedmap estende mapa {

Erro ao traduzir esta página.

// retorna qualquer que o método da superclasse retorne.retornar super.set (chave, valor);}}Os dois primeiros argumentos para o construtor typeDMap () são oTipos de chave e valor desejados.Estas devem ser strings, como "número"e "booleano", que o operador do tipo de retorna.Você também pode especificarUm terceiro argumento: uma matriz (ou qualquer objeto iterável) de [chave, valor]Matrizes que especificam as entradas iniciais no mapa.Se você especificar algumentadas iniciais, então a primeira coisa que o construtor faz é verificar queseus tipos estão corretos.Em seguida, o construtor chama a superclasseConstrutor, usando a super palavra -chave como se fosse um nome de função.O construtor map () leva um argumento opcional: um objeto iterávelde matrizes [chave, valor].Então o terceiro argumento opcional doO construtor typeDMap () é o primeiro argumento opcional para o mapa ()construtor, e passamos a esse construtor de superclasse comsuper (entradas).Depois de invocar o construtor da superclasse para inicializar o estado da superclasse,O construtor typeDMap () a seguir inicializa seu próprio estado de subclasse pordefinindo this.KeyType e this.valuetype para o especificadotipos.Ele precisa definir essas propriedades para que possa usá -las novamente emo método set ().Existem algumas regras importantes que você precisará saber sobre o usosuper () nos construtores:Se você definir uma classe com a palavra -chave estende, então oconstrutor para sua classe deve usar super () para invocar o

construtor de superclasse. Se você não definir um construtor em sua subclasse, um será definido automaticamente para você. Isso definido implicitamente construtor simplesmente leva os valores passados ?? para ele e passa esses valores para super (). Você não pode usar essa palavra -chave em seu construtor até depois de invocar o construtor de superclasse com super(). Isso aplica uma regra que as superclasses chegam a inicializar -se antes que as subclasses façam. A expressão especial new.Target é indefinida em funções que são chamadas sem a nova palavra -chave. Em funções de construtor, no entanto, o New.Target é uma referência ao construtor que foi invocado. Quando uma subclasse o construtor é invocado e usa super () para invocar o construtor de superclasse, esse construtor de superclasse verá o construtor da subclasse como o valor de New.Target. Bem Superclass projetada não precisa saber se tem sido subclassificado, mas pode ser útil poder usar new.target.name nas mensagens de log, por exemplo. Após o construtor, a próxima parte do Exemplo 9-6 é um método chamado definir(). O mapa Superclass define um método chamado set () para adicionar uma nova entrada para o mapa. Dizemos que este método set () no tipo Dmap substitui o método set () de sua superclasse. Este mapa é digitoso simples. A subclasse não sabe nada sobre adicionar novas entradas para mapear, mas sabe como verificar os tipos, e é isso que faz primeiro, verificando que a chave e o valor a serem adicionados ao mapa têm os tipos corretos e lançando um erro se não o fizerem. Este método set () não tem qualquer maneira de acrescentar a chave e valor ao próprio mapa, mas é isso que o método de superclasse set () é para. Então, usamos a super palavra -chave novamente.

Erro ao traduzir esta página.

Em vez disso, envolvendo ou "compondo" outras classes. Esta delegação é frequentemente chamada de "composição" e é uma máxima frequentemente citada de programação orientada a objetos que se deve favorecer a composição sobre herança. Suponha, por exemplo, queríamos uma classe de histograma que se comportasse algo como a classe `Set` de JavaScript, exceto que, em vez de apenas acompanhar se um valor foi agregado a definir ou não, em vez disso mantém uma contagem do número de vezes que o valor foi adicionado. Porque a API para esta classe de histograma é semelhante ao conjunto, podemos considerar o conjunto de subclassificação e adicionar um método `count()`. Por outro lado, uma vez que começamos a pensar em como podemos implementar isso, podemos perceber que a classe de histograma é mais como um mapa do que um conjunto, porque ele precisa manter um mapeamento entre valores e o número de vezes que foram adicionados. Então, em vez de um conjunto de subclasse, podemos criar uma classe que define uma API do tipo conjunto, mas implementa esses métodos delegando a um objeto de mapa interno. O exemplo 9-7 mostra como poderíamos fazer isso.

Exemplo 9-7. Histogram.js: uma classe semelhante a um conjunto implementado com delegação

```
/** Uma classe parecida com um conjunto que acompanha quantas vezes um valor tem* foi adicionado. Ligue para add() e remove() como você faria para um conjunto, e* Call count() para descobrir quantas vezes um determinado valor tem sido adicionado.* O iterador padrão produz os valores que foram adicionados pelo menos* uma vez. Use entradas () se você deseja iterar [valor, contagem] pares.*/
```

classe Histogram {2}

```
// Para inicializar, apenas criamos um objeto de mapa para delegar para construtor () {this.map = new Map ();}// Para
qualquer chave, a contagem é o valor no mapa, ou zero// Se a chave não aparecer no mapa.count (key) {return
this.map.get (key) || 0;}// O método de conjunto de set () retorna true se a contagem for diferente de zero em (key)
{return this.count (key) > 0;}// O tamanho do histograma é apenas o número de entradas no mapa.get size () {return
this.map.size;}// Para adicionar uma chave, basta incrementar sua contagem no mapa.add (key) {this.map.set
(chave, this.count (key) + 1);}// Excluir uma chave é um pouco mais complicado, porque temos que excluir// A chave
do mapa se a contagem voltar para zero.delete (key) {Let count = this.count (chave);if (count === 1)
{this.map.Delete (chave);} else if (contagem > 1) {this.map.set (chave, contagem - 1);}}// Iterando um histograma
apenas retorna as chaves armazenadas nele[Symbol.iterator] () {return this.map.keys ();}// Esses outros métodos
de iterador apenas delegam ao mapa objeto keys () {return this.map.keys ();} valores () {return this.map.values
??();} entradas () {return this.map.entries ();}}
```

Erro ao traduzir esta página.

subclasses relacionadas.Uma superclasse abstrata pode definir um parcialimplementação que todas as subclasses herdam e compartilham.As subclasses,Então, só precisa definir seu próprio comportamento único, implementandoos métodos abstratos definidos - mas não implementados - porSuperclass.Observe que o JavaScript não tem nenhuma definição formal deMétodos abstratos ou classes abstratas;Estou simplesmente usando esse nome aquipara métodos não implementados e classes implementadas incompletamente.O exemplo 9-8 é bem comentado e permanece por conta própria.Eu encorajoVocê lê -lo como um exemplo capstone para este capítulo sobre as classes.OA classe final no Exemplo 9-8 faz muita manipulação de bits com o &, |,e ~ operadores, que você pode revisar no §4.8.3.Exemplo 9-8.Sets.js: Uma hierarquia de classes de conjuntos abstratos e concretos/**/ A classe AbstractSet define um único método abstrato,tem().*/classe AbstractSet { // joga um erro aqui para que as subclasses sejam forçadas// Para definir sua própria versão de trabalho deste método.tem (x) {lança um novo erro ("Método abstrato");}}/**/ O Notset é uma subclasse concreta do abstractset.* Os membros deste conjunto são todos valores que não são membrosde alguns* outro conjunto.Porque é definido em termos de outro conjuntonão é* gravável, e porque tem membros infinitos, não éenumerável.* Tudo o que podemos fazer com isso é teste para associação e convertê -lo para um* String usando notação matemática.*/classe Notset estende o abstractSet {

```
construtor (set) {super();this.set = set;}// nossa implementação do método abstrato que herdamostem (x) {return  
this.set.has (x);}// e também substituímos este método de objetotostring () {return `x |x ? $ {this.set.toString ()}`}  
`;}/*** O conjunto de alcance é uma subclasse concreta do AbstractSet.Seus membrossão* Todos os valores que  
estão entre os e os limites,inclusive.* Como seus membros podem ser números de ponto flutuante, não é*  
enumerável e não tem um tamanho significativo.*/Classe Rangeset estende o AbstractSet {construtor (de, para)  
{super();this.from = de;this.to = para;}tem (x) {return x >= this.from && x <= this.to;`}tostring () {return `x |$  
{this.from} ? x ? $ {this.to}`;}/** AbstractEnumerableset é uma subclasse abstrata deAbstractSet.Define* um  
getter abstrato que retorna o tamanho do conjunto e tambémdefine um* Iterador abstrato.E então implementa  
concretoisEmpty (), tostring () ,* e iguais () métodos em cima deles.Subclasse issoimplementar o* iterador, o  
tamanho do getter e o método has () obtêm estesconcreto* Métodos gratuitamente.*/
```

Erro ao traduzir esta página.

*[Symbol.iterator] () {rende this.Member;}/** AbstractWritablesset é uma subclasse abstrata deAbstractEnumerableset.* Define os métodos abstratos insert () e remove () queinsira e* Remova os elementos individuais do conjunto e depois implementaconcreto* Métodos Add (), Subtract () e Intersect () em cima deles.Observe que* Nossa API diverge aqui do conjunto de javascript padrãoaula.*/classe AbstractWritablesset estende abstractenumerablesetet {inserir (x) {lançar novo erro ("Método abstrato");}remover (x) {lançar novo erro ("Método abstrato");}add (set) {para (deixe o elemento do set) {this.insert (elemento);}}subtrair (set) {para (deixe o elemento do set) {this.remove (elemento);}}intersect (set) {para (deixe o elemento disso) {if (! set.has (elemento)) {this.remove (elemento);}}}}/** Um bitset é uma subclasse concreta de abstrumwritablesset com

Erro ao traduzir esta página.

```
e bitdeixe bit = x % 8;if (! this._has (byte, bit)) {// se esse bit forainda não está definidothis.data [byte] |= bitset.bits  
[bit];// então definathis.n ++;// etamanho do conjunto de incrementos} outro {jogue novo TypeError ("Elemento de  
conjunto inválido:" + x);}Remova (x) {if (this._valid (x)) {// se o valor forválidoodeixe byte = math.floor (x / 8);//  
Calcule o bytee bitdeixe bit = x % 8;if (this._has (byte, bit)) {// se esse bit forjá definidothis.data [byte] & =  
bitset.masks [bit];// então sem seremthis.n--;// etamanho decrescente} outro {jogue novo TypeError ("Elemento de  
conjunto inválido:" + x);} } // um getter para retornar o tamanhofunction get size () {return this.n;} // iterar o  
conjunto apenas verificando cada bit por sua vez.// (poderíamos ser muito mais inteligentes e otimizar  
isso substancialmente)*[Symbol.iterator] () {para (vamos i = 0; i <= this.max; i++) {if (this.has (i)) {rendimento i;
```

}}}}// Alguns valores pré-computados usados ??pelos has (), insert () eRemover () métodosBitset.bits = novo uint8array ([1, 2, 4, 8, 16, 32, 64, 128]);Bitset.masks = novo uint8array ([~ 1, ~ 2, ~ 4, ~ 8, ~ 16, ~ 32, ~ 64, ~ 128]);9.6 ResumoEste capítulo explicou os principais recursos das classes JavaScript:Objetos que são membros da mesma classe herdam propriedadesdo mesmo objeto de protótipo.O objeto de protótipo é oPrincipais características das classes JavaScript, e é possível definirAulas com nada mais do que o objeto.create ()método.Antes do ES6, as aulas eram mais tipicamente definidas pelo primeirodefinindo uma função construtora.Funções criadas com oA palavra -chave da função tem uma propriedade de protótipo e aO valor desta propriedade é um objeto usado como protótipode todos os objetos criados quando a função é invocada com novocomo um construtor.Ao inicializar este objeto de protótipo, você podeDefina os métodos compartilhados da sua classe.Embora oObjeto de protótipo é o principal recurso da classe, o construtorA função é a identidade pública da classe.ES6 apresenta uma palavra -chave de classe que facilita aDefinir classes, mas sob o capô, construtor e protótipoO mecanismo permanece o mesmo.As subclasses são definidas usando a palavra -chave Extends em uma classe

Erro ao traduzir esta página.

Capítulo 10. MódulosO objetivo da programação modular é permitir que grandes programas sejam montados usando módulos de código de autores e fontes diferentes para que todo esse código seja executado corretamente, mesmo na presença de código que os vários autores do módulo não anteciparam.Como prática comum, modularidade é principalmente sobre encapsular ou ocultar particularmente a implementação detalhada e mantendo o espaço de nome global arrumado para que os módulos não podem modificar acidentalmente as variáveis, funções e classes definidas por outros módulos.Até recentemente, o JavaScript não tinha suporte embutido para módulos e os programadores que trabalham em grandes bases de código fizeram o possível para usar o modularidade fraca disponível através de classes, objetos e fechamentos.Modularidade baseada em fechamento, com suporte de ferramentas de andamento de código, LED para uma forma prática de modularidade baseada em uma função requer (), que foi adotado pelo nó.require ()-módulos baseados em parte fundamental do ambiente de programação do nó, mas nunca foram adotados como parte oficial da linguagem JavaScript.Em vez disso,O ES6 define módulos usando palavras-chave de importação e exportação.Embora a importação e exportação fazem parte do idioma há anos, eles foram implementados apenas por navegadores da Web e só relativamente recentemente.E, como uma questão prática, a modularidade JavaScript ainda depende do código-ferramentas de agrupamento.As seções que seguem a capa:

Módulos do faça você mesmo com classes, objetos e fechamentosMódulos de nó usando requer ()Módulos ES6 usando exportação, importação e importação ()10.1 módulos com classes, objetos eFechamentosEmbora possa ser óbvio, vale ressaltar que um doscaracterísticas importantes das classes é que elas atuam como módulos para seusMétodos.Pense no exemplo 9-8.Esse exemplo definiu um númerode diferentes classes, todas que tinham um método chamado ().Mas vocênão teria nenhum problema em escrever um programa que usasse vários conjuntosClasses desse exemplo: não há perigo que a implementaçãode has () de singletonset substituirá o método has () deBitset, por exemplo.A razão pela qual os métodos de uma classe são independentes doMétodos de outras classes não relacionadas é que os métodos de cada classe sãodefinito como propriedades de objetos de protótipo independentes.A razão dissoAs classes são modulares é que os objetos são modulares: definir uma propriedade em umO objeto JavaScript é como declarar uma variável, mas adicionando propriedadespara objetos não afeta o espaço de nome global de um programa, nemafeta as propriedades de outros objetos.JavaScript define algunsfunções e constantes matemáticas, mas em vez de definir todos elasGlobalmente, eles são agrupados como propriedades de um único objeto de matemática global.Essa mesma técnica poderia ter sido usada no Exemplo 9-8.Em vez deDefinindo classes globais com nomes como singletonset e bitset, queExemplo poderia ter sido escrito para definir apenas um único conjunto global

objeto, com propriedades referenciando as várias classes. Usuários disso Sets Library poderia então se referir às classes com nomes como Sets.singleton e sets.bit. Usar classes e objetos para modularidade é comum e útil Técnica na programação JavaScript, mas não vai longe o suficiente. Em particular, ele não nos oferece nenhuma maneira de ocultar a implementação interna detalhes dentro do módulo. Considere o exemplo 9-8 novamente. Se fôssemos escrevendo esse exemplo como um módulo, talvez tenhamos gostado. Mantenha as várias classes abstratas internas ao módulo, apenas fazendo as subclasses de concreto disponíveis para os usuários do módulo. Da mesma forma, em A classe Bitset, os métodos _valid () e _has () são internos Utilitários que não devem realmente ser expostos aos usuários da classe. E Bitset.bits e bitset.Masks são detalhes de implementação que estaria melhor escondido. Como vimos em §8.6, variáveis ?? locais e funções aninhadas declaradas dentro de uma função é privada para essa função. Isso significa que podemos usar imediatamente invocou expressões de função para alcançar um tipo de modularidade deixando os detalhes da implementação e funções de utilidade escondido na função anexante, mas tornando a API pública domódulo o valor de retorno da função. No caso da classe Bitset, podemos estruturar o módulo como este:

```
const bitset = (function () { // defina bitset para o retorno valor desta função // Detalhes de implementação privada aqui ifunction isvalid (set, n) {...} função tem (set, byte, bit) {...} const bits = novo uint8array ([1, 2, 4, 8, 16, 32, 64, 128]);
```

```
const máscara = novo uint8array ([~ 1, ~ 2, ~ 4, ~ 8, ~ 16, ~ 32, ~ 64, ~ 128]);// A API pública do módulo é apenas a classe Bitset, que definimos// e volte aqui. A classe pode usar o privado funções e constantes// definido acima, mas eles serão ocultos de usuários de a classe A classe de retorno bitset estende abstractableSetet {// ... Implementação omitida ...};} () ); Essa abordagem da modularidade se torna um pouco mais interessante quando o O módulo tem mais de um item. O código a seguir, por exemplo, Define um mini módulo de estatística que exporta Mean () e StdDev () Funções ao deixar os detalhes da implementação ocultos:// é assim que poderíamos definir um módulo de estatísticas const stats = (function () {// funções de utilitário privadas para o módulo const sum = (x, y) => x + y; const square = x => x * x; // uma função pública que será exportada função média (dados) {return data.reduce (sum) / data.length;} // uma função pública que iremos exportar função stddev (dados) {Seja M = média (dados); retornar math.sqrt (data.map (x => x - m) .Map (quadrado) .Reduce (Sum) / (Data.Length - 1));} // Nós exportamos a função pública como propriedades de um
```

Erro ao traduzir esta página.

```
const sum = (x, y) => x + y;const square = x => x * x;exports.mean = function (dados) {...};exports.stddev = function (dados) {...};exportações de retorno;} ()Com módulos agrupados em um único arquivo como o mostrado noExemplo anterior, você pode imaginar escrever código como o seguinte paraFaça uso desses módulos:// Obtenha referências aos módulos (ou ao conteúdo do módulo) queNós precisamosconst stats = requer ("stats.js");const bitset = requer ("sets.js").bitset;// agora escreva código usando esses módulosSeja s = novo  
bitset (100);S.Insert (10);S.Insert (20);S.Insert (30);deixe a média = estatísticas.mean ([... s]);// A média é de 20Este código é um esboço aproximado de como as ferramentas de Bundling de código (comowebpack e pacote) para os navegadores da web funcionam, e também é um simplesIntrodução à função requer () como a usada no nóprogramas.10.2 Módulos no nóNa programação de nós, é normal dividir programas em tantos arquivoscomo parece natural.Esses arquivos de código JavaScript são assumidos para todos os vivosem um sistema de arquivos rápido.Ao contrário dos navegadores da web, que precisam ler arquivos de
```

Erro ao traduzir esta página.

Module.Exports:Module.Exports = classe Bitset estende abstrumwritableserset {/ implementação omitida};O valor padrão do Module.Exports é o mesmo objeto que Exportações refere -se a.No módulo de estatísticas anteriores, poderíamos teratribuído a função média ao module.exports.mean em vez deexports.Mean.Outra abordagem com módulos como as estatísticasmódulo é exportar um único objeto no final do módulo, em vez deExportando funções uma a uma enquanto você avança:// define todas as funções, público e privadoconst sum = (x, y) => x + y;const square = x => x * x;const média = dados => data.reduce (sum) /data.length;const stddev = d => {Seja m = média (d);retornar math.sqrt (d.map (x => x -m) .Map (quadrado) .Reduce (soma)/(d.Length-1));};// agora exporte apenas os públicosmodule.exports = {média, stddev};10.2.2 Importações de nósUm módulo de nó importa outro módulo chamando o requer ()função.O argumento desta função é o nome do módulo a serimportado, e o valor de retorno é qualquer valor (normalmente uma função,classe, ou objeto) que o módulo exporta.Se você deseja importar um módulo de sistema incorporado para o nó ou um módulo

Erro ao traduzir esta página.

Objeto que você planeja usar. Compare estas duas abordagens:// importe o objeto de estatísticas inteiras, com todas as suas funçõesconst stats = requer ('./ stats.js');// Temos mais funções do que precisamos, mas eles são ordenadamente// Organizado em um espaço de nome de "estatísticas" conveniente.deixe a média = estats.mean(dados);// Como alternativa, podemos usar a destruição idiomáticaatribuição para importar// exatamente as funções que queremos diretamente no localnamespace:const {stdDev} = requer ('./ stats.js');// Isso é bom e sucinto, embora tenhamos um pouco de contexto// sem o prefixo 'estatísticas' como um namespace para o stdDev ()função.Seja sd = stdDev (dados);10.2.3 módulos no estilo de nó na webMódulos com um objeto de exportação e uma função requer() são construídos no nó.Mas se você estiver disposto a processar seu código com um agrupamentoferramenta como webpack, também é possível usar esse estilo de módulosPara o código que se destina a ser executado em navegadores da web.Até recentemente, isso era uma coisa muito comum a fazer, e você pode ver muitos código que ainda faz isso.Agora que o JavaScript tem sua própria sintaxe de módulo padrão, no entanto, Os desenvolvedores que usam pacotes têm maior probabilidade de usar o oficialMódulos JavaScript com declarações de importação e exportação.10.3 Módulos em ES6

ES6 adiciona palavras-chave de importação e exportação ao JavaScript e finalmente suporta modularidade real como um recurso de linguagem principal. Modularidade ES6 é conceitualmente o mesmo que a modularidade do nó: cada arquivo é seu próprio módulo, e constantes, variáveis, funções e classes definidas em um arquivo são privado a esse módulo, a menos que sejam exportados explicitamente. Valoriza isso: os exportados de um módulo estão disponíveis para uso em módulos que importam explicitamente. Os módulos ES6 diferem dos módulos de nível global nos seguintes aspectos:

- Os módulos são definidos em navegadores da Web. As seções a seguir explicam essas coisas em detalhes.
- Primeiro, observe que os módulos ES6 também são diferentes dos regulares JavaScript "scripts" de algumas maneiras importantes. O mais óbvio é a própria modularidade: em scripts regulares, nível superior declarações de variáveis, funções e classes entram em um único contexto global compartilhado por todos os scripts. Com módulos, cada arquivo tem seu próprio contexto privado e pode usar as declarações de importação e exportação.
- Qual é o ponto principal, afinal. Mas existem outras diferenças entre módulos e scripts também. Código dentro de um módulo ES6 (como código dentro de qualquer definição de classe ES6) está automaticamente no modo rigoroso (Ver §5.6.3). Isso significa que, quando você começa a usar módulos ES6, você nunca precisará escrever "Use Strict" novamente. Isso significa que o código nos módulos não pode usar a instrução com os argumentos ou os argumentos de objeto ou variáveis declaradas. Os módulos ES6 são até um pouco mais rigorosos do que o modo rigoroso: no modo rigoroso, em funções invocadas como funções, isso é indefinido. Nos módulos, isso é indefinido mesmo no topo do código de nível. (Por outro lado, scripts em navegadores da web e no nível global.)

Módulos ES6 na web e no nóOs módulos ES6 estão em uso na web há anos com a ajuda de pacotes de código como o webpack, que combinam módulos independentes de código JavaScript em grande,Pacotes não modulares adequados para inclusão em páginas da web.No momento dissoEscrevendo, no entanto, os módulos ES6 são finalmente suportados nativamente por todos os navegadores dawebAlém do Internet Explorer.Quando usados ??nativamente, os módulos ES6 são adicionados emPáginas html com uma tag <script type = "módulo">, descrito posteriormenteNeste capítulo.Enquanto isso, tendo pioneiro na modularidade JavaScript, o nó se encontra nopoluição desajetada de ter que apoiar dois módulos não totalmente compatíveissistemas.O nó 13 suporta módulos ES6, mas por enquanto, a grande maioria do nóOs programas ainda usam módulos de nó.10.3.1 ES6 ExportaçõesExportar uma constante, variável, função ou classe de um módulo ES6,Basta adicionar a exportação de palavras -chave antes da declaração:exportar const pi = math.pi;Função de exportação degreestoradians (d) {return d * pi / 180;}Círculo de classe de exportação {construtor (r) {this.r = r; }área () {return pi * this.r * this.r; }}Como uma alternativa para espalhar palavras -chave de exportação ao longo de seu módulo, você pode definir suas constantes, variáveis, funções eaulas como você normalmente faria, sem declaração de exportação, e então(normalmente no final do seu módulo) Escreva uma única declaração de exportaçãolsso declara exatamente o que é exportado em um único local.Então, em vez de

Escrevendo três exportações individuais no código anterior, poderíamos terescrito equivalentemente uma única linha no final:`exportação {círculo, degreestoradians, pi};` Esta sintaxe parece a palavra -chave de exportação seguida de um objetaliteral (usando notação abreviada).Mas neste caso, os aparelhos encaracolados fazemna verdade não define um objeto literal.Esta sintaxe de exportação simplesmente requerUma lista separada por vírgula de identificadores dentro de aparelhos encaracolados.É comum escrever módulos que exportam apenas um valor (normalmente umfunção ou classe) e, neste caso, geralmente usamos o padrão de exportaçõoem vez de exportar:Exportar a classe padrão Bitset {`// implementação omitida`}As exportações padrão são um pouco mais fáceis de importar do que as exportações não padrão,Então, quando houver apenas um valor exportado, usando o padrão de exportaçãofacilita as coisas para os módulos que usam seu valor exportado.Exportações regulares com exportação só podem ser feitas em declarações que tem um nome.As exportações padrão com o padrão de exportação podem exportar qualquerexpressão, incluindo expressões de função anônima e anônimas e expressões de classe.Isso significa que, se você usar o padrão de exportação, vocêpode exportar literais de objetos.Então, diferente da sintaxe de exportação, se você viraparelho encaracolado após o padrão de exportação, é realmente um objeto literal queestá sendo exportado.

É legal, mas um tanto incomum, para que os módulos tenham um conjunto de Exportações regulares e também uma exportação padrão. Se um módulo tiver um padrãoExportar, ele só pode ter um. Por fim, observe que a palavra -chave de exportação só pode aparecer no nível superior do seu código JavaScript. Você não pode exportar um valor de dentro de uma classe, função, loop ou condicional. (Esta é uma característica importante do Sistema de módulos ES6 e permite a análise estática: uma exportação de módulos será sempre a mesma em cada execução, e os símbolos exportados podem ser determinados antes que o módulo seja realmente executado.)

10.3.2 ES6 importações

Você importa valores que foram exportados por outros módulos com o importar palavra-chave. A forma mais simples de importação é usada para módulos que define uma exportação padrão: importar bitset de './bitset.js'; Esta é a palavra-chave de importação, seguida por um identificador, seguido por uma palavra-chave da chave, seguida por uma string literal que nomeia o módulo cuja exportação padrão estamos importando. O valor de exportação padrão do módulo especificado se torna o valor do identificador especificado no módulo atual. O identificador ao qual o valor importado é atribuído é uma constante, como se tivesse sido declarado com a palavra -chave const. Como exportações, importações só podem aparecer no nível superior de um módulo e não é permitido dentro de aulas, funções, loops ou condicionais. Por que a convenção universal, as importações necessárias para um módulo são colocadas no início de

o módulo.Curiosamente, no entanto, isso não é necessário: como função de declarações, importações são "içadas" para o topo e todos os valores importados estão disponíveis para qualquer código do código do módulo.O módulo do qual um valor é importado é especificado como uma constanteString literal em citações únicas ou citações duplas.(Você não pode usar um variável ou outra expressão cujo valor é uma string, e você não pode usar uma string dentro de backticks porque os literais de modelo podem interpolar variáveis ??e nem sempre têm valores constantes.) Nos navegadores da web, Esta string é interpretada como um URL em relação à localização do módulo. Isso está fazendo a importação.(No nó, ou ao usar uma ferramenta de pacote,A string é interpretada como um nome de arquivo em relação ao módulo atual, mas isso faz pouca diferença na prática.) Uma sequência de especificadores de módulo deveSeja um caminho absoluto começando com "/", ou um caminho relativo começando com "./" ou "../", ou um URL completo com protocolo e nome de host.O ES6A especificação não permite strings especificadores de módulo não qualificado como "Util.js" porque é ambíguo se isso pretende citar um módulo no mesmo diretório que o atual ou algum tipo de sistema módulo instalado em algum local especial.(Esta restrição contra "especificadores de módulo nu" não é homenageado por ferramentas de agrupamento de código como o webpack, que pode ser facilmente configurado para encontrar módulos nus em um diretório da biblioteca que você especifica.) Uma versão futura do idioma Pode permitir "especificadores de módulo nu", mas, por enquanto, eles não são permitidos.Se você deseja importar um módulo do mesmo diretório que o atual, basta colocar ./? antes do nome do módulo e importar de"./Util.js" em vez de "util.js".Até agora, consideramos apenas o caso de importar um único valor de um módulo que usa o padrão de exportação.Para importar valores de um

Módulo que exporta vários valores, usamos uma sintaxe ligeiramente diferente:`importar {média, stddev} de "./stats.js";`Lembre -se de que as exportações padrão não precisam ter um nome no móduloIsso os define.Em vez disso, fornecemos um nome local quando importamosEsses valores.Mas as exportações não-defensivas de um módulo têm nomes noexportando módulo, e quando importamos esses valores, nos referimos a elespor esses nomes.O módulo de exportação pode exportar qualquer número devalor nomeado.Uma declaração de importação que faz referência a que o módulo podeimportar qualquer subconjunto desses valores simplesmente listando seus nomes dentroaparelho encaracolado.Os aparelhos encaracolados fazem desse tipo de declaração de importaçãoParece algo como uma tarefa de destruição e destruiçãoA tarefa é realmente uma boa analogia para o que esse estilo de importação éfazendo.Os identificadores dentro de aparelhos encaracolados são todos içados no topo deo módulo de importação e se comporta como constantes.Os guias de estilo às vezes recomendam que você importe explicitamente a cadaSímbolo que seu módulo usará.Ao importar de um módulo quedefine muitas exportações, no entanto, você pode importar facilmente tudo comUma declaração de importação como esta:`importação * como estatísticas de "./stats.js";`Uma declaração de importação como essa cria um objeto e o atribui a umconstante estatísticas nomeadas.Cada uma das exportações não padrão do móduloSer importado se torna uma propriedade desse objeto de estatísticas.Não-defasaAs exportações sempre têm nomes, e esses são usados ??como nomes de propriedadesdentro do objeto.Essas propriedades são efetivamente constantes: elas

não pode ser substituído ou excluído. Com a importação curinga mostrada em O exemplo anterior, o módulo de importação usaria o importadomean () e stddev () funcionam através do objeto STATS, invocando -os como estatísticas.mean () e stats.stddev (). Os módulos geralmente definem uma exportação padrão ou múltiplo nomeadoexportações. É legal, mas um tanto incomum, para um módulo usar os doisexportar e exportar inadimplência. Mas quando um módulo faz isso, vocêpode importar o valor padrão e os valores nomeados com umDeclaração de importação como esta:importar histograma, {média, stddev} de "./histogram-estats.js"; Até agora, vimos como importar de módulos com uma exportação padrão e de módulos com exportações não padrão ou nomeadas. Mas há umoutra forma da declaração de importação que é usada com módulos quenão têm exportações. Para incluir um módulo de não exportações em seuPrograma, basta usar a palavra -chave de importação com o especificador do módulo:importação "./Analytics.js"; Um módulo como esse é executado na primeira vez que é importado. (E subsequenteas importações não fazem nada.) Um módulo que apenas define funções é só útilse exportar pelo menos uma dessas funções. Mas se um módulo executar algunsCódigo, pode ser útil importar mesmo sem símbolos. UmMódulo de análise para um aplicativo da web pode executar o código para se registrarvários manipuladores de eventos e depois usam esses manipuladores de eventos para enviarDados de telemetria de volta ao servidor em horários apropriados. O módulo éindependente e não precisa exportar nada, mas ainda precisamos

Erro ao traduzir esta página.

fornecer outra maneira de importar módulos que definem ambas exportação padrão e exportações nomeadas. Lembre-se do `./HISTOMSTATS.JS`? módulo da seção anterior. Aqui está outra maneira de importar ambos as exportações padrão e nomeadas desse módulo: importar {padrão como histograma, média, stddev} de `./histogram stats.js`; Nesse caso, o padrão de palavra-chave JavaScript serve como espaço reservado e nos permite indicar que queremos importar e fornecer um nome para a exportação padrão do módulo. Também é possível renomear valores à medida que você os exporta, mas apenas quando usando a variante de cinta encaracolada da declaração de exportação. Não é comum precisar fazer isso, mas se você escolher nomes curtos e sucintos para usar dentro do seu módulo, você pode preferir exportar seus valores com nomes mais descritivos que têm menos probabilidade de entrar em conflito com outros módulos. Como nas importações, você usa a palavra-chave AS para fazer isso: exportação {layout como calculateLayout, renderizar como renderLayout}; Lembre-se de que, embora os aparelhos encaracolados pareçam algo como objetos literais, eles não são, e a palavra-chave de exportação espera um único identificador antes do AS, não uma expressão. Isso significa, infelizmente, que você não pode usar a renomeação de exportação assim: exportação {Math.sin como pecado, math.cos como cos}; // SyntaxError

Erro ao traduzir esta página.

Observe que os nomes significam e o stddev não são realmente usados ??nestecódigo.Se não estamos sendo seletivos com um reexport e simplesmente queremos exportar todos os valores nomeados de outro módulo, podemos usar umcuringa:exportação * de "./stats/mean.js";exportação * de "./stats/stddev.js";Reexportar sintaxe permite renomear com a mesma importação regular eDeclarações de exportação fazem.Suponha que queríamos reexportar a média ()função, mas também defina a média () como outro nome para a função.Poderíamos fazer isso assim:exportar {média, média como média} de "./stats/mean.js";exportar {stddev} de "./stats/stddev.js";Todos os reexports deste exemplo assumem que o ?./stats/mean.js?e os módulos ?./stats/stddev.js? exportam suas funções usando a exportaçãoem vez de padrão de exportação.De fato, no entanto, uma vez que estes sãomódulos com apenas uma única exportação, teria feito sentido definireles com o padrão de exportação.Se tivéssemos feito isso, então o reexportA sintaxe é um pouco mais complicada porque precisa definir um nomepara as exportações padrão sem nome.Podemos fazer isso assim:exportar {padrão como média} de "./stats/mean.js";exportar {padrão como stddev} de "./stats/stddev.js";Se você deseja reexportar um símbolo nomeado de outro módulo como oExportação padrão do seu módulo, você pode fazer uma importação seguida porum padrão de exportação, ou você pode combinar as duas declarações como

esse:// importar a função média () de ./stats.js e torná -lo o// Exportação padrão deste móduloexportar {significa como padrão} de "./stats.js"E finalmente, para reexportar a exportação padrão de outro módulo como oexportação padrão do seu módulo (embora não esteja claro por que você fariaquero fazer isso, já que os usuários podem simplesmente importar o outro módulodiretamente), você pode escrever:// O módulo médio.js simplesmente reexporta as estatísticas/mean.jsexportação padrãoexportar {default} de "./stats/mean.js"10.3.5 Módulos JavaScript na WebAs seções anteriores descreveram os módulos ES6 e sua importaçãoe exportar declarações de maneira um tanto abstrata.Nestaseção e na próxima, discutiremos como eles realmente funcionam emnavegadores da web e se você ainda não é uma web experienteDesenvolvedor, você pode encontrar o resto deste capítulo mais fácil de entenderDepois de ler o capítulo 15.No início de 2020, o código de produção usando módulos ES6 ainda é geralmentePacotado com uma ferramenta como Webpack.Existem trocas para fazer isso,Mas, no geral, o Bundling de código tende a dar melhor desempenho.Quepode muito bem mudar no futuro à medida que a velocidade da rede cresce e o navegadorOs fornecedores continuam otimizando suas implementações de módulos ES6.Mesmo que as ferramentas de agrupamento ainda possam ser desejáveis ??na produção, elasnão são mais necessários no desenvolvimento, pois todos os navegadores atuais1

Forneça suporte nativo para módulos JavaScript. Lembre -se de que os módulos usam Modo rigoroso por padrão, isso não se refere a um objeto global e superior. As declarações de nível não são compartilhadas globalmente por padrão. Desde módulos deve ser executado de maneira diferente do código não módulo legado, seu Introdução requer alterações no HTML, bem como JavaScript. Se você quiser usar de maneira nativamente as diretivas de importação em um navegador da web, você deve digitar ao navegador da web que seu código é um módulo usando um `<script type = "Module">` tag. Uma das características agradáveis ?? dos módulos ES6 é que cada módulo tem um Conjunto estático de importações. Então, dado um único módulo inicial, um navegador da web pode carregar todos os seus módulos importados e depois carregar todos os módulos importados por esse primeiro lote de módulos, e assim por diante, até que um completo o programa foi carregado. Vimos que o especificador do módulo em uma declaração de importação pode ser tratada como um URL relativo. Um `<script type = "Module">` tag marca o ponto de partida de um modular programa. Nenhum dos módulos que ele importa deve estar em `<Script>` tags, no entanto: em vez disso, elas são carregadas sob demanda como arquivos javascript regulares e são executados no modo rigoroso como ES6 regular módulos. Usando uma tag `<script type = "módulo">` para definir o ponto de entrada principal para um programa modular JavaScript pode ser tão simples quanto esse:

```
<script type = "módulo">
importar "./main.js"; </script>
```

Código dentro de uma tag em uma tag `<script type = "módulo">` é uma ES6 módulo e, como tal, podem usar a declaração de exportação. Não há nenhum a pontar para fazer isso, no entanto, porque a sintaxe de tag html `<cript>`

Erro ao traduzir esta página.

Erro ao traduzir esta página.

O sistema é usado por cada arquivo que carrega. Então, se você está escrevendo módulos ES6 e quero que eles sejam utilizáveis ?? com o nó, então pode ser útil adotar A Convenção de Nomeação . MJS. 10.3.6 Importações dinâmicas com importação () Vimos que as diretivas de importação e exportação do ES6 são completamente estático e habilite intérpretes de JavaScript e outros Ferramentas de javascript para determinar as relações entre módulos com análise de texto simples enquanto os módulos estão sendo carregados sem ter para realmente executar qualquer código nos módulos. Com estaticamente módulos importados, você tem garantia de que os valores que você importam para um módulo estarão pronto para uso antes de qualquer código do seu módulo começar a correr. Na web, o código deve ser transferido por uma rede em vez de ser lido no sistema de arquivos. E uma vez transferido, esse código é frequentemente executado em dispositivos móveis com CPUs relativamente lentas. Este não é o tipo de ambiente em que as importações de módulos estáticos - que exigem um programa inteiro a ser carregado antes de qualquer um deles - fazem muito sentido. É comum que os aplicativos da web carreguem inicialmente apenas o suficiente de seu código para renderizar a primeira página exibida ao usuário. Então, uma vez que o usuário tem algum conteúdo preliminar para interagir, eles podem começar a carregar muitas vezes muito maior de código necessária para o resto da web app. Os navegadores da web facilitam o carregamento dinamicamente usando o DOM API para injetar uma nova tag <script> no HTML atual documento e aplicativos da Web fazem isso há muitos anos.

Embora o carregamento dinâmico tenha sido possível há muito tempo, não tem faz parte do próprio idioma. Isso muda com a introdução da importação () no ES2020 (no início de 2020, a importação dinâmica é suportada por todos os navegadores que suportam módulos ES6). Você passa um módulo especificador para importar () e retorna um objeto de promessa que representa o processo assíncrono de carregar e executar o módulo especificado. Quando a importação dinâmica é concluída, a promessa é "cumprida" (ver Capítulo 13 para obter detalhes completos sobre programação assíncrona e Promessas) e produz um objeto como o que você obteria com o importação * como forma da declaração de importação estática. Então, em vez de importar o módulo "./stats.js" estaticamente, assim:importação * como estatísticas de "./stats.js"; Podemos importá-lo e usá-lo dinamicamente, assim:importar("./ stats.js"). Então (stats => {deixe a média = estats.mean (dados);}) Ou, em uma função assíncrona (novamente, pode ser necessário ler o capítulo 13 Antes de entender esse código), podemos simplificar o código com a guarda: Async analiseData (dados) {Deixe estatísticas = aguarda importar ("./ stats.js"); retornar {Média: Stats.Mean (Data), stdDev: stats.stddev (dados)};}

O argumento a ser import () deve ser um especificador de módulo, exatamente comoUm que você usaria com uma diretiva de importação estática.Mas com importação (),Você não está limitado a usar uma string constante literal: qualquer expressãoque avalia uma string na forma correta servirá.Dinâmico importar () parece uma invocação de funções, mas na verdade énão.Em vez disso, importar () é um operador e os parênteses são umparte necessária da sintaxe do operador.A razão para esse pouco incomum deA sintaxe é que o import () precisa ser capaz de resolver especificadores de módulocomo URLs em relação ao módulo atualmente em execução, e isso requer umum pouco de magia de implementação que não seria legal para colocar em umFunção de JavaScript.A função versus a distinção do operador raramentefaz a diferença na prática, mas você notará se tentar escrevercódigo como console.log (importação);ou que requer =importar;.Por fim, observe que o dinâmico importação () não é apenas para navegadores da web.Ferramentas de embalagem de código como o WebPack também podem fazer bom uso.OA maneira mais direta de usar um patrimônio de código é dizer o principalponto de entrada para o seu programa e deixe -o encontrar toda a importação estáticadiretivas e monte tudo em um arquivo grande.Por estrategicamenteUsando chamadas dinâmicas de importação (), no entanto, você pode quebrar esseMonolítico Bachar em um conjunto de feixes menores que podem ser carregadosSob demanda.10.3.7 Import.Meta.urlHá um recurso final do sistema de módulos ES6 para discutir.Dentro deum módulo ES6 (mas não dentro de um <script> regular ou um módulo de nó

carregado com requer ()), a sintaxe especial import.meta refere -se a um objeto que contém metadados sobre o módulo atualmente executando. A propriedade URL deste objeto é o URL do qual o módulo foi carregado. (No nó, este será um arquivo: // url.) O caso de uso primário de importação.meta.url deve ser capaz de se referir a imagens, arquivos de dados ou outros recursos que são armazenados no mesmo diretório como (ou relativo a) o módulo. O construtor url () fazé fácil resolver um URL relativo contra um URL absoluto como import.Meta.url. Suponha, por exemplo, que você tenha escrito um módulo que inclui seqüências que precisam ser localizadas e que os arquivos de localização são armazenados em um diretório L10N/mesmo diretório do próprio módulo. Seu módulo pode carregar suas cordas usando um URL criado com uma função, como esta:

```
Função LocalStrings(url) {returnar novo URL(`L10N/${Locale}.json`, import.meta.url);}
```

10.4 Resumo

O objetivo da modularidade é permitir que os programadores ocultem os detalhes da implementação de seu código para que pedaços de código de várias fontes podem ser montadas em grandes programas sem preocupar -se que um pedaço substitua as funções ou variáveis ?? de outro. Este capítulo explicou três módulos JavaScript diferentes: sistemas: Nos primeiros dias do JavaScript, a modularidade só poderia ser alcançada através do uso inteligente de invocar imediatamente

expressões de função. Node adicionou seu próprio sistema de módulos em cima do JavaScript linguagem. Os módulos de nó são importados com requer () eDefina suas exportações definindo propriedades do objeto de exportação, ou configurando a propriedade Module.Exports. No ES6, o JavaScript finalmente conseguiu seu próprio sistema de módulos com importar e exportar palavras -chave e o ES2020 está adicionando Suporte para importações dinâmicas com importação ().¹ Por exemplo: aplicativos da web que têm atualizações incrementais frequentes e usuários que fazem visitas de retorno frequentes podem achar que o uso de pequenos módulos em vez de maços grandes poderia resultar em melhores tempos de carga média devido à melhor utilização do navegador do usuário cache.

Capítulo 11. O JavaScript

Biblioteca padrão

Alguns tipos de dados, como números e strings (capítulo 3), objetos(Capítulo 6) e Matrizes (capítulo 7) são tão fundamentais para JavaScriptque podemos considerá -los parte da própria linguagem.Este capítuloabrange outras APIs importantes, mas menos fundamentais, que podem ser pensadasde definir a ?biblioteca padrão? para JavaScript: estes são úteisclasses e funções que são incorporadas ao JavaScript e disponíveis para todosProgramas JavaScript nos navegadores da Web e no nó.As seções deste capítulo são independentes uma da outra, e vocêpode lê -los em qualquer ordem.Eles cobrem:

As classes de conjunto e mapa para representar conjuntos de valores e Mapeamentos de um conjunto de valores para outro conjunto de valores.

Objetos semelhantes a matrizes conhecidos como typedarrays que representam matrizesde dados binários, juntamente com uma classe relacionada para extrair valoresde dados binários que não são de teatro.

Expressões regulares e a classe Regexp, que definepadrões textuais e são úteis para processamento de texto.

Esta seçãoTambém abrange a sintaxe de expressão regular em detalhes.

A classe de data para representar e manipular datas evezes.

A classe de erro e suas várias subclasses, casos de quesão lançados quando os erros ocorrem em programas JavaScript.

O objeto JSON, cujos métodos apóiam a serialização eDeserialização de estruturas de dados JavaScript compostas deobjetos, matrizes, cordas, números e booleanos.O objeto INTL e as classes que ele define que podem ajudá -loLocalize seus programas JavaScript.O objeto de console, cujos métodos produzem seqüências de seqüências de maneiras de maneirasque são particularmente úteis para programas de depuração e registrar o comportamento desses programas.A classe URL, que simplifica a tarefa de analisar eManipulando URLs.Esta seção também abrange funções globaispara codificar e decodificar URLs e suas partes componentes.setTimeout () e funções relacionadas para especificar o código paraser executado após o intervalo de tempo especificado.Algumas das seções deste capítulo - principalmente, as seções sobre digitadasmatrizes e expressões regulares - são bastante longas porque háinformações de fundo significativas que você precisa entender antes de vocêpode usar esses tipos de maneira eficaz.Muitas das outras seções, no entanto,são curtos: eles simplesmente apresentam uma nova API e mostram alguns exemplosde seu uso.11.1 conjuntos e mapasO tipo de objeto de JavaScript é uma estrutura de dados versátil que pode ser usada paraStrings de mapa (nomes de propriedades do objeto) para valores arbitrários.EQuando o valor que está sendo mapeado é algo fixo como verdadeiro, então oO objeto é efetivamente um conjunto de cordas.Os objetos são realmente usados ??como mapas e conjuntos de maneira bastante rotineiramente em JavaScript

Programação, mas isso é limitado pela restrição às cordas e complicado pelo fato de que objetos normalmente herdam propriedades comuns como "ToString", que normalmente não se destinam a fazer parte de um mapa ou conjunto. Por esse motivo, o ES6 apresenta as classes verdadeiras de conjunto e mapa, que vamos explorar nas subseções a seguir.

11.1.1 A classe definida

Um conjunto é uma coleção de valores, como uma matriz. Ao contrário de matrizes, no entanto, conjuntos não são ordenados ou indexados, e eles não permitem duplicatas: um valor é um membro de um conjunto ou não é um membro; não é possível perguntar quantas vezes um valor aparece em um conjunto. Crie um objeto definido com o construtor set():

```
Seja s = new Set(); // um novo conjunto vazio
```

Seja t = novo conjunto ([1, s]); // Um novo conjunto com dois membros

O argumento para o construtor set() não precisa ser uma matriz: nenhum objeto iterável (incluindo outros objetos set) é permitido:

```
Seja t = novo conjunto (s); // um novo conjunto que copia os elementos de s.
```

deixe exclusivo = new Set ("Mississippi"); // 4 elementos: "m", "i", "S" e "P"

A propriedade de tamanho de um conjunto é como a propriedade de comprimento de uma matriz: diz quantos valores o conjunto contém:

```
exclusivo.size // => 4
```

Os conjuntos não precisam ser inicializados quando você os cria. Você pode adicionar e remover os elementos a qualquer momento com add (), delete () e clear(). Lembre-se de que os conjuntos não podem conter duplicatas, então adicionar um valor para um conjunto quando já contém esse valor não tem efeito: Seja s = new Set(); // Comece vazios.size // => 0 s.add(1); // Adicione um número. size // => 1; agora o conjunto tem um membro s.add(1); // Adicione o mesmo número novamente. size // => 1; O tamanho não muda. add (verdadeiro); // Adicione outro valor; Observe que é tipos finos para misturá-los. size // => 2 s.add([1,2,3]); // Adicione um valor de matriz. size // => 3; A matriz foi adicionada, não a sua elemento s.delete(1) // => true: elemento excluído com sucesso! s.size // => 2; O tamanho está de volta para 2 s.delete("teste") // => false: "teste" não era um membro, a exclusão falhou! s.delete(true) // => true: delete foi bem-sucedido s.delete([1,2,3]) // => false: a matriz no conjunto é diferente. size // => 1: Ainda existe uma matriz em conjuntos.clear(); // Remova tudo do conjunto. size // => 0 Existem alguns pontos importantes a serem observados sobre este código: O método add () leva um único argumento; Se você passar um array, adiciona a própria matriz ao conjunto, não à matriz individualmente. add () sempre retorna o conjunto em que é chamado. No entanto, então se você deseja adicionar vários valores a um conjunto, você pode usar chamadas de método encadeado como

s.add ('a'). add ('b'). add ('c'); O método Delete () também exclui apenas um único elemento de conjunto de cada vez. Ao contrário de add (), no entanto, delete () retorna um valor booleano. Se o valor que você especificar foi realmente um membro do conjunto, o delete () o remove e retorna true. Caso contrário, não faz nada e retorna falsa. Finalmente, é muito importante entender que o conjunto de membros é baseado em verificações estritas de igualdade, como o operador === executa. Um conjunto pode conter o número 1 e a string "1", porque os considera valores distintos. Quando os valores são objetos (ou matrizes ou funções), eles também são comparados como se com ===. É por isso que não conseguimos excluir o elemento da matriz do conjunto neste código. Adicionamos uma matriz para o conjunto e depois tentou remover essa matriz passando uma matriz diferente (embora com os mesmos elementos) para o método delete (). Para que isso funcione, teríamos de passar uma referência exatamente à mesma matriz.

OBSERVAÇÃO

Os programadores Python tomam nota: essa é uma diferença significativa entre JavaScript e conjuntos de Python. Conjuntos de Python comparam os membros para a igualdade, não a identidade, mas a troca é que os conjuntos de Python apenas permitem membros imutáveis, como tuplas, e não permitem que listas e dicionários sejam adicionados aos conjuntos. Na prática, a coisa mais importante que fazemos com os conjuntos não é adicionar e remover os elementos deles, mas para verificar se um especificado valor é um membro do conjunto. Fazemos isso com o método Has ():

```
Let OneDigitPrimes = novo conjunto ([2,3,5,7]); oneDigitPrimes.has (2) // => true: 2 é um primo de um dígito
```

número onedigitPries.has (3) // => true: assim é 3 onedigitPries.has (4) // => false: 4 não é um primo onedigitPries.has ("5") // => false: "5" não é nem mesmo um número. A coisa mais importante a entender sobre os conjuntos é que eles são otimizados para testes de associação, e não importa quantos membros o conjunto tem, o método Has () será muito rápido. O inclui () Método de uma matriz também realiza testes de associação, mas os tempos tomados são proporcionais ao tamanho da matriz e usando uma matriz como um conjunto pode ser muito, muito mais lento do que usar um objeto definido real. A classe Set é iterável, o que significa que você pode usar um loop para/de loop para enumerar todos os elementos de um conjunto: deixe soma = 0; para (deixe p de onedigitPries) { // percorrer o único dígito primo soma += p; // e adicione -os } soma // => 17: 2 + 3 + 5 + 7 Porque os objetos definidos são iteráveis, você pode convertê-los em matrizes e argumento listas com o ... Spread Operator: [... onedigitPries] // => [2,3,5,7]: o conjunto convertido em uma matriz Math.max (... onedigitPries) // => 7: Definir elementos passados ?? como argumentos de função Os conjuntos são frequentemente descritos como "coleções não ordenadas". Isso não é exatamente verdadeiro para a classe JavaScript Set, no entanto. Um conjunto de JavaScript é Não indexado: você não pode pedir o primeiro ou o terceiro elemento de um conjunto

Você pode com uma matriz. Mas a classe JavaScript Set sempre se lembra a ordem em que os elementos foram inseridos e sempre usa esta ordem. Quando você itera um conjunto: o primeiro elemento inserido será o primeiro iterado (assumindo que você não o excluiu primeiro) e mais recentemente o elemento inserido será o último iterado. Além de ser iterável, a classe set também implementa um método `foreach` () que é semelhante ao método da matriz do mesmo nome: deixe o produto = 1; oneDigitPrimes.ForEach (n => {Product *= n;}); Produto // => 210: 2 * 3 * 5 * 7 O `foreach` () de uma matriz passa os índices de matriz como o segundo argumento para a função que você especificar. Conjuntos não têm índices, então a versão definida da classe deste método simplesmente passa o valor do elemento como o primeiro e o segundo argumento.

11.1.2 A classe do mapa

Um objeto de mapa representa um conjunto de valores conhecidos como chaves, onde cada chave tem outro valor associado a ela ("mapeado para"). Em certo sentido, um mapa é como uma matriz, mas em vez de usar um conjunto de números inteiros sequenciais como as chaves, mapas nos permitem usar valores arbitrários como "índices". Como matrizes, os mapas são rápidos: procurar o valor associado a uma chave será rápido (embora não seja tão rápido quanto a indexação de uma matriz). Não importa o tamanho do mapa. Crie um novo mapa com o construtor `map()`:

Seja m = novo map ()// Crie um novo mapa vazioSeja n = novo mapa ([]// um novo mapa inicializado com teclas de stringmapeado para números["One", 1],["dois", 2]]);O argumento opcional para o construtor map () deve ser um iterávelObjeto que produz dois elementos [chave, valor] Matrizes.Na prática,Isso significa que, se você deseja inicializar um mapa quando o criar,Você normalmente escreve as teclas desejadas e os valores associados como umMatriz de matrizes.Mas você também pode usar o construtor map () para copiaroutros mapas ou para copiar os nomes e valores de propriedades de um existenteobjeto:Deixe copiar = novo mapa (n)// um novo mapa com as mesmas chaves evalores como mapa nSeja o = {x: 1, y: 2}// um objeto com duas propriedadesSeja p = novo mapa (object.entries (o));// O mesmo que o novo mapa ([]["x",1], ["y", 2]])Depois de criar um objeto de mapa, você pode consultar o valorassociado a uma determinada chave com get () e pode adicionar uma nova chave/valorEmparelhar com set ().Lembre -se, porém, que um mapa é um conjunto de chaves, cada umdos quais tem um valor associado.Isso não é exatamente o mesmo que um conjunto depares de chave/valor.Se você ligar para o set () com uma chave que já existe nomapa, você mudará o valor associado a essa chave, não adicionar um novoMapeamento de chave/valor.Além de obter () e set (), a classe de mapaDefine também métodos que são como métodos definidos: o uso tem () para verificarce um mapa inclui a chave especificada;use delete () para remover umchave (e seu valor associado) do mapa;use clear () para remover

todos os pares de chave/valor do mapa; e use a propriedade de tamanho para descobrirQuantas chaves um mapa contém.Seja m = novo map ()// Comece com um mapa vazioM.Size // => 0: Mapas vazios não têm chavesM.set ("One", 1); // mapeie a chave "um" para o valor 1M.set ("dois", 2); // e a chave "dois" para o valor 2.M.Size // => 2: O mapa agora tem duas chavesm.get ("dois") // => 2: retorna o valor associado com chave "dois"m.get ("três") // => indefinido: esta chave não está nodefinirM.set ("One", verdadeiro); // altere o valor associado a umchave existenteM.Size // => 2: O tamanho não mudam.has ("um") // => true: o mapa tem uma chave "m.has (verdadeiro) // => false: o mapa não tem uma chaveverdadeiroM.Delete ("One") // => True: a chave existia e a exclusão conseguiuM.Size // => 1M.Delete ("três") // => false: Falha ao excluir umchave inexistenteM.clear ()// Remova todas as chaves e valores domapaComo o método add () de set, o método set () de mapa pode ser acorrentado, o que permite que os mapas sejam inicializados sem usar matrizes deMatrizes:Seja m = novo map ().set ("um", 1).set ("dois", 2).set ("três", 3);M.Size // => 3m.get ("dois") // => 2Como no conjunto, qualquer valor de JavaScript pode ser usado como uma chave ou um valor em umMapa. Isso inclui nulo, indefinido e nan, além de referência

Tipos como objetos e matrizes. E, como na classe Set, o mapa comparachaves por identidade, não por igualdade, portanto, se você usar um objeto ou matriz como umchave, será considerada diferente de todos os outros objetos e matrizes, Mesmo aqueles com exatamente as mesmas propriedades ou elementos: Seja m = novo map ():// Comece com um mapa vazio.m.set ({}, 1); // Mapeie um objeto vazio para o número 1.m.set ({}, 2); // mapear um objeto vazio diferente para oNúmero 2.M.Size // => 2: Existem duas chaves neste mapam.get ({}) // => indefinido: mas este objeto vazionão é uma chaveM.set (M, indefinido); // mapeia o mapa em si para o valorindefinido.m.has (m) // => true: m é uma chave em si mesmam.get (m) // => indefinido: o mesmo valor que obteríamos seM não foi uma chaveOs objetos de mapa são iteráveis ??e cada valor iterado é uma matriz de dois elementosonde o primeiro elemento é uma chave e o segundo elemento é o valorassociado a essa chave. Se você usar o operador de spread com um mapaobjeto, você receberá uma variedade de matrizes como as que passamos para oMap () construtor. E quando itera um mapa com um loop para/deé idiomático para usar a atribuição de destruição para atribuir a chave e o valorpara separar variáveis: Seja m = novo mapa ([["x", 1], ["y", 2]]); [... m] // => [["x", 1], ["y", 2]]para (let [chave, valor] de m) { // Na primeira iteração, a chave será "x" e o valor iráser 1// Na segunda iteração, a chave será "y" e valorserá 2}

Como a classe Set, a classe do mapa itera em ordem de inserção. O primeiro par de chaves/valores iterado será o menos recentemente adicionado ao mapa, e o último par iterado será o mais recentemente adicionado. Se você deseja iterar apenas as chaves ou apenas os valores associados a um Mapa, use os métodos Keys () e Values (): estes retornam iteráveis que iteram as chaves e valores, em ordem de inserção. (O Método de entradas () retorna um objeto iterável que itera a chave/valor pares, mas isso é exatamente o mesmo que iterando o mapa diretamente.) [... M.Keys ()] // => ["x", "y"] : apenas as chaves [... M.Values ?? ()] // => [1, 2] : apenas os valores [... M.Entries ()] // => [{"x": 1}, {"y": 2}]: o mesmo que [... m] Os objetos de mapa também podem ser iterados usando o método foreach () que foi implementado pela primeira vez pela classe da matriz. M.ForEach ((valor, chave) => {} // Valor da nota, chave não -chave, valor // Na primeira invocação, o valor será 1 e a chave será "x" // Na segunda invocação, o valor será 2 e a chave será "y"}); Pode parecer estranho que o parâmetro de valor venha antes da chave parâmetro no código acima, pois com/de iteração, a chave vem primeiro. Como observado no início desta seção, você pode pensar em um mapa como uma matriz generalizada na qual os índices da matriz inteira são substituídos por valores -chave arbitrários. O método de matrizes foreach () passa o elemento da matriz primeiro e o índice da matriz em segundo, então, por analogia, o foreach () método de um mapa passa o valor do mapa primeiro e o mapa

chave em segundo lugar.

11.1.3 Frawmap e fraco

A classe de mapa fraco é uma variante (mas não uma subclasse real) do mapa classe que não impede que seus valores -chave sejam coletados no lixo. Coleção de lixo é o processo pelo qual o intérprete JavaScript recuperar a memória de objetos que não são mais "alcançáveis" enão pode ser usado pelo programa. Um mapa regular é "forte" referências aos seus principais valores, e eles permanecem alcançáveis ??através do Mapa, mesmo que todas as outras referências a eles tenham desaparecido. O mapa fraco, porContraste, mantém referências "fracas" aos seus principais valores para que não sejam alcançável através do mapa fraco, e sua presença no mapa faznão impedir que sua memória seja recuperada. O construtor frAchapMap () é exatamente como o construtor map (), masExistem algumas diferenças significativas entre o Frafmap e o mapa: As teclas de mapa fraco devem ser objetos ou matrizes; Os valores primitivos sãonão está sujeito a coleta de lixo e não pode ser usado como chaves. FrawMap implementa apenas o get (), set (), tem () eMétodos delete (). Em particular, o mapa fraco não é iterávele não define as chaves (), valores () ou foreach (). SeFrafmap era iterável, então suas chaves seriam acessíveis eNão seria fraco. Da mesma forma, o Frafmap não implementa a propriedade de tamanhoPorque o tamanho de um mapa fraco pode mudar a qualquer momento comoObjetos são coletados de lixo. O uso pretendido de fracos é permitir que você associe valores a

objetos sem causar vazamentos de memória. Suponha, por exemplo, que você est o escrevendo uma fun o que pega um argumento de objeto e precisa executar algum c lculo demorado nesse objeto. Para efici ncia, voc  deseja cache o valor calculado para a reutiliza o posterior. Se voc  usa um objeto de mapa para implementar o cache, voc  impedir  qualquer um dos objetos de sempre sendo recuperados, mas usando um mapa fraco, voc  evite esse problema. (Muitas vezes voc  pode obter um resultado semelhante usando uma propriedade de s mbolo privado para cache o valor calculado diretamente no objeto. Veja §6.10.3.)

Fraqueza implementa um conjunto de objetos que n o impedem aqueles objetos de serem coletados. O construtor fracos () funciona como o construtor set (), mas os objetos fracos diferem do conjunto de objetos da mesma maneira que os objetos fracos diferem do mapa de objetos: o BRACHST n o permite valores primitivos como membros. Fraqueza implementa apenas o add (), tem () e m todos delete () e n o  e iter vel. O BRACHST n o possui uma propriedade de tamanho. O fraco n o  e usado com frequ ncia: seus casos de uso s o como os paraMap fraco. Se voc  deseja marcar (ou "marca") um objeto como tendo alguma propriedade ou tipo especial, por exemplo, voc  pode adicion -lo a um conjunto fraco. Ent o, em outros lugares, quando voc  quiser verificar essa propriedade ou tipo, voc  pode testar a associa o a esse conjunto fraco. Fazendo isso com um conjunto regular impediria que todos os objetos marcados fossem coletados de lixo, mas isso n o  e uma preocup o ao usar o fraco.

11.2 Matrizes digitadas e dados binários

Matrizes javascript regulares podem ter elementos de qualquer tipo e crescer ou encolher dinamicamente. As implementações de JavaScript realizam muitas otimizações para que os usos típicos das matrizes JavaScript sejam muito rápidos. No entanto, eles ainda são bem diferentes dos tipos de matriz de idiomas de nível inferior como C e Java. Matrizes digitadas, que são novas em ES6, estão muito mais próximas das matrizes de baixo nível desses idiomas. Matrizes digitadas não são tecnicamente matrizes (`Array.isArray()` retorna falso para elas), mas elas implementam todos os métodos de matriz descrito em §7.8, além de mais alguns deles. Eles diferem de matrizes regulares de algumas maneiras muito importantes, no entanto: Os elementos de uma matriz digitada são todos números. Ao contrário do regular, os números de JavaScript, no entanto, as matrizes digitadas permitem que você especifique o tipo (números inteiros assinados e não assinados e IEEE-754 ponto flutuante) e tamanho (8 bits a 64 bits) dos números a serem armazenados na matriz. Você deve especificar o comprimento de uma matriz digitada quando criá-la; esse comprimento nunca pode mudar. Os elementos de uma matriz digitada são sempre inicializados para 0 quando a matriz é criada.

11.2.1 Tipos de matriz digitadas

O JavaScript não define uma classe `typedArray`. Em vez disso, existem 11 tipos de matrizes digitadas, cada uma com um tipo de elemento diferente: construtor tipo numérico 3

`Int8array ()bytes assinados``Uint8Array ()bytes não assinados``UINT8ClampedArray ()bytes não assinados sem rolagem``Int16Array ()Inteiros curtos de 16 bits assinados``Uint16Array ()Não assinado 16 bits e números inteiros``Int32Array ()Inteiros assinados de 32 bits``Uint32Array ()Não assinado inteiros de 32 bits``Bigint64Array ()Valores Bigint de 64 bits assinados (ES2020)``Biguint64Array ()Valores Bigint de 64 bits não assinados (ES2020)``Float32Array ()Valor de ponto flutuante de 32 bits``Float64Array ()Valor de ponto flutuante de 64 bits: um número JavaScript regular``Os tipos cujos nomes começam com int Hold Hold Signated Inteiros, de 1, 2, ou 4 bytes (8, 16 ou 32 bits).``Os tipos cujos nomes começam com UintSegure números inteiros não assinados desses mesmos comprimentos.``O "bigint" e Os tipos "biguint" contêm números inteiros de 64 bits, representados em JavaScript como Valores bigint (ver §3.2.5).``Os tipos que começam com float seguram Números de ponto flutuante.``Os elementos de um float64Array são do mesmo tipo que números regulares de JavaScript.``Os elementos de um float32Array tem menor precisão e uma faixa menor, mas requerem apenas metade da memória.`(Este tipo é chamado de flutuação em C e Java.)`UINT8ClampedArray é uma variante de caso especial no Uint8Array.`Ambos os tipos mantêm bytes não assinados e podem representar números

entre 0 e 255. Com Uint8Array, se você armazenar um valor maior de 255 ou menor de zero em um elemento de matriz, ele ?envolve? e Você tem algum outro valor.É assim que a memória do computador funciona em um Baixo nível, então isso é muito rápido.UINT8ClampedArray faz um pouco Verificação de tipo extra para que, se você armazenar um valor maior que 255 ou menor do que 0, ele ?prende? a 255 ou 0 e não envolve.(Esse O comportamento de fixação é exigido pelo elemento html <chevas>API de baixo nível para manipular cores de pixels.)Cada um dos construtores de matriz tipados tem um BYTES_PER_ELEMENTpropriedade com o valor 1, 2, 4 ou 8, dependendo do tipo.11.2.2 Criando matrizes digitadasA maneira mais simples de criar uma matriz digitada é chamar o apropriado construtor com um argumento numérico que especifica o número deElementos que você deseja na matriz:Let Bytes = novo Uint8Array (1024); // 1024 bytesLet Matrix = new Float64Array (9); // uma matriz 3x3Let Point = new Int16Array (3); // Um ?? ponto no espaço 3DSeja RGBA = novo Uint8ClampedArray (4); // Um ??pixel de 4 bytes RGBAVvalorSeja sudoku = novo int8array (81); // um quadro de 9x9 SudokuQuando você cria matrizes digitadas dessa maneira, os elementos da matriz são todos garantidos a ser inicializado para 0, On ou 0,0.Mas se você conhece oValores que você deseja em sua matriz digitada, você também pode especificar esses valoresQuando você cria a matriz.Cada um dos construtores de matriz tipados tem estáticos de () e () métodos de fábrica que funcionam como Array.From () e Array.Of ():

Seja branco = uint8clampedArray.of (255, 255, 255, 0); // rgbabranco opacoLembre-se de que o método da fábrica do Array.From () espera um matrizou objeto iterável como seu primeiro argumento.O mesmo vale para os digitadosvariantes de matriz, exceto que o objeto iterável ou parecido com uma matriz também devetem elementos numéricos.Strings são iteráveis, por exemplo, mas seriaNão faça sentido em passá -los para o método de fábrica de () de um digitadovarietade.Se você está apenas usando a versão de um argumento de (), você pode solte o. From e passe seu objeto iterável ou de matriz diretamente paraA função construtora, que se comporta exatamente a mesma.Observe queTanto o construtor quanto o método de fábrica de () permitem que vocêCopie as matrizes digitadas existentes, ao mesmo tempo em que altera o tipo:deixe ints = uint32Array.From (branco); // os mesmos 4 números,mas como intsQuando você cria uma nova matriz digitada de uma matriz existente, iterável, ouobjeto semelhante a uma matriz, os valores podem ser truncados para ajustar o tiporestrições da sua matriz.Não há avisos ou erros quando issoacontece:// flutua truncado para ints, mais longos truncados para 8 bitsUint8array.of (1.23, 2,99, 45000) // => novo uint8Array ([1, 2,200])Finalmente, há mais uma maneira de criar matrizes digitadas que envolvem oTipo de matriz.Um ArrayBuffer é uma referência opaca a um pedaço dememória.Você pode criar um com o construtor;Apenas passe no

Número de bytes de memória que você deseja alocar:Let buffer = new ArrayBuffer (1024*1024);buffer.byteLength
// => 1024*1024;um megabyte de memóriaA classe ArrayBuffer não permite que você leia ou escreva nenhum dosbytes que você alocou.Mas você pode criar matrizes digitadas que usam a memória do buffer e isso permite que você leia e escreva issomemória.Para fazer isso, chame o construtor de matriz digitado com umArrayBuffer como o primeiro argumento, um deslocamento de byte dentro do buffer de matriz como o segundo argumento e o comprimento da matriz (em elementos, não em bytes) como o terceiro argumento.O segundo e o terceiro argumentos são opcionais.Se Você omita os dois, então a matriz usará toda a memória na matrizbuffer.Se você omitir apenas o argumento do comprimento, sua matriz usará toda a memória disponível entre a posição inicial e o fim de a matriz.Mais uma coisa a ter em mente sobre esta forma de digitadoConstrutor de matriz: as matrizes devem estar alinhadas com a memória, portanto, se você especificar umDeslocamento de byte, o valor deve ser um múltiplo do tamanho do seu tipo.OO construtor int32array () requer um múltiplo de quatro, por exemplo,e o float64array () requer um múltiplo de oito.Dado o ArrayBuffer criado anteriormente, você pode criar matrizes digitadas como estes:deixe asbytes = novo uint8array (buffer);// Visto como bytesLet Asints = new Int32Array (buffer);// Visto como INTS assinado de 32 bitsdeixe o lastk = novo uint8Array (buffer, 1023*1024);// Durar Kilobyte como bytesSeja ints2 = new Int32Array (buffer, 1024, 256);// 2º Kilobyte como 256 números inteiros

Erro ao traduzir esta página.

}A função aqui calcula o maior número primo menor que oNúmero que você especifica.O código é exatamente o mesmo que seria com umArray JavaScript regular, mas usando uint8array () em vez deArray () faz o código funcionar mais de quatro vezes mais rápido e usarOito vezes menos memória nos meus testes.Matrices digitadas não são matrizes verdadeiras, mas reimplementam a maioria das matrizesMétodos, para que você possa usá -los praticamente como você usaria regularmenteMatrizes:deixe ints = new int16Array (10);// 10 números inteiros curtosints.fill (3) .map (x => x*x) .Join ("") // => "999999999999"Lembre -se de que as matrizes digitadas têm comprimentos fixos, então o comprimentoA propriedade é somente leitura e métodos que alteram o comprimento da matriz(como push (), pop (), não (), shift () e splice ())não são implementados para matrizes digitadas.Métodos que alteram o conteúdode uma matriz sem alterar o comprimento (como Sort (),reverse () e preenchimento ()) são implementados.Métodos como mapa ()e slice () que retornam novas matrizes retornam uma matriz digitada do mesmodigite o que eles são chamados.11.2.4 Métodos e propriedades de matriz digitadaAlém dos métodos de matriz padrão, as matrizes digitadas também implementam umpoucos métodos próprios.O método set () define vários elementosde uma matriz digitada de uma só vez, copiando os elementos de um regular ou digitadoMatriz em uma matriz digitada:

Let Bytes = novo Uint8Array (1024); // um buffer de 1k
deixe padrão = novo uint8array ([0,1,2,3]); // Uma matriz de 4bytes
bytes.set (padrão); // copie -os para o início de outroArray de bytes
bytes.set (padrão, 4); // copie -os novamente em um diferente de Bytes
Bytes.set ([0,1,2,3], 8); // ou apenas copiar valores diretos de umArray
regularBytes.slice (0, 12) // => novoUint8array ([0,1,2,3,0,1,2,3,0,1,2,3])
O método set () pega uma matriz ou matriz digitada como seu primeiro argumento e um elemento deslocado como seu segundo argumento opcional, que padrão a 0 se não especificado. Se você está copiando valores de uma matriz digitada para outro, a operação provavelmente será extremamente rápida. Matrizes digitadas também têm um método de subarray que retorna uma parte de uma matriz em que é chamado:
Seja ints = new Int16Array ([0,1,2,3,4,5,6,7,8,9]); // 10 Inteiros curtos
Seja last3 = ints.subarray (ints.length-3, ints.length); // Últimos 3 deles
last3 [0] // => 7: É o mesmo que o INTS [7]
Subarray () leva os mesmos argumentos que o método slice () e parece funcionar da mesma maneira. Mas há uma diferença importante. Slice () retorna os elementos especificados em um novo e independente digitadoArray que não compartilha memória com a matriz original. Subarray () não copia nenhuma memória; apenas retorna uma nova visão dos mesmos valores subjacentes:
ints [9] = -1; // Alterar um valor na matriz original e ...
last3 [2] // => -1: também muda no subarray

Erro ao traduzir esta página.

o bufferbytes.buffer [1] // => 255: Isso apenas define um regularPropriedade JSbytes [1] // => 0: a linha acima não definiuo byteVimos anteriormente que você pode criar um ArrayBuffer com oArrayBuffer () construtor e, em seguida, crie matrizes digitadas que usamEsse buffer.Outra abordagem é criar uma matriz digitada inicial, entãoUse o buffer dessa matriz para criar outras visualizações:Let Bytes = novo Uint8Array (1024); // 1024 bytesdeixe ints = new Uint32Array (bytes.buffer); // ou 256InteirosDeixe Floats = new Float64Array (Bytes.Buffer); // ou 128duplos11.2.5 DataView e EndianessMatrizes digitadas permitem que você visualize a mesma sequência de bytes em pedaçosde 8, 16, 32 ou 64 bits.Isso expõe o "endianess": a ordem emQuais bytes são organizados em palavras mais longas.Para eficiênciac, digitadaAs matrizes usam a endianidade nativa do hardware subjacente.Em Little-Endian Systems, os bytes de um número são organizados em um ArrayBufferDo menos significativo ao mais significativo.Em plataformas grandes endianas, oOs bytes são organizados de mais significativos e menos significativos.Você podedeterminar a endianidade da plataforma subjacente com código comoesse:// Se o número inteiro 0x00000001 estiver organizado na memória como 01 0000 00, então// Estamos em uma plataforma pouco endiana.Em um grande endianoPlataforma, nós conseguiríamos// bytes 00 00 00 01 em vez disso.Deixe LittleEndian = new Int8Array (New Int32Array ([1]). Buffer)

[0] === 1;Hoje, as arquiteturas da CPU mais comuns são pouco endianas.MuitosProtocolos de rede e alguns formatos de arquivo binário exigem grande e-e-e-e-unspedidos de byte, no entanto.Se você estiver usando matrizes digitadas com dados queveio da rede ou de um arquivo, você não pode apenas assumir que oA plataforma Endianness corresponde à ordem de byte dos dados.Em geral,Ao trabalhar com dados externos, você pode usar o Int8Array eUint8array para ver os dados como uma variedade de bytes individuais, mas vocêNão deve usar as outras matrizes digitadas com tamanhos de palavras multibyte.Em vez disso, você pode usar a classe DataView, que define métodos paraler e escrever valores de um ArrayBuffer explicitamentePedido de byte especificado:// Suponha que tenhamos uma variedade digitada de bytes de dados binários paraprocesso.Primeiro,// criamos um objeto DataView para que possamos ler de maneira flexível eescrever// valores desses bytesLet View = new DataView (bytes.buffer,bytes.byteoffset,bytes.byTeLength);deixe int = view.getInt32 (0);// leia Big-Endian assinado intde byte 0int = view.getInt32 (4, false);// a seguir também é grandeEndianint = view.getuint32 (8, true);// o próximo int é pouco endianoe não assinadoview.setUint32 (8, int, false);// Escreva de volta em Big-Formato EndianoDataView define 10 métodos GET para cada uma das 10 matrizes digitadasAulas (excluindo o UINT8ClampedArray).Eles têm nomes comogetInt16 (), getInt32 (), getBigint64 () e

getfloat64 () . O primeiro argumento é o deslocamento do byte dentro do ArrayBuffer no qual o valor começa. Todos esses métodos getter, Além de getInt8 () e getUint8 (), aceite um opcional valor booleano como seu segundo argumento. Se o segundo argumento for omitido ou é falsa, a ordem de bytes big-endian é usada. Se o segundo argumento é verdadeiro, a ordem pouco endiana é usada. DataView também define 10 métodos de conjunto correspondentes que escrevem valores no ArrayBuffer subjacente. O primeiro argumento é o deslocamento em que o valor começa. O segundo argumento é o valor para escrever. Cada um dos métodos, exceto setInt8 () e setUint8 (), aceita um terceiro argumento opcional. Se o argumento for omitido ou é falso, o valor é escrito em formato big-endiano com o byte mais significativo primeiro. Se o argumento for verdadeiro, o valor é escrito em Little-Endian formato com o byte menos significativo primeiro. Matrizes digitadas e a classe DataView fornecem todas as ferramentas necessárias para processar dados binários e permitir que você escreva programas JavaScript que fazem coisas como descomprimir arquivos ZIP ou extrair metadados de arquivos jpeg.

11.3 Combinação de padrões com regular Expressões

Uma expressão regular é um objeto que descreve um padrão textual. O JavaScript Regexp Class representa expressões regulares e ambos String e regexp definem métodos que usam expressões regulares para realizar funções poderosas de correspondência de padrões e pesquisa e pesquisa

no texto. Para usar a API `regexp` de maneira eficaz, no entanto, você deve aprender também a descrever padrões de texto usando a expressão regular gramática, que é essencialmente uma mini linguagem de programação de seuter. Felizmente, a gramática de expressão regular JavaScript é bastante semelhante à gramática usada por muitas outras linguagens de programação, então você já pode estar familiarizado com isso. (E se você não é, o esforço que você investiu no aprendizado de expressões regulares JavaScript provavelmente será útil para você em outros contextos de programação também.) As subseções a seguir descrevem a gramática de expressão regular Primeiro, e depois, depois de explicar como escrever expressões regulares, eles explicam como você pode usá-los com métodos da `string` e `regexp` classes.

11.3.1 Definindo expressões regulares

No JavaScript, expressões regulares são representadas pelos objetos `regexp`. Objetos `regexp` podem ser criados com o construtor `regexp()`. Claro, mas eles são criados com mais frequência usando uma sintaxe literal especial. Assim como os literais de cordas são especificados como caracteres dentro de aspas, expressões regulares literais são especificados como caracteres dentro de um par de slash (/) caracteres. Assim, seu código JavaScript pode conter linhas como esse:

```
deixe padrão = /s$/; Esta linha cria um novo objeto regexp e o atribui à variável padrão. Este objeto regexp em particular corresponde a qualquer string que termine com a letra "s". Esta expressão regular pode ter equivalente
```

foi definido com o construtor `regexp()`, como este: Deixe padrão = novo `regexp("s $")`; Especificações de padrão de expressão regular consistem em uma série de caracteres. A maioria dos personagens, incluindo todos os personagens alfanuméricos, Simplesmente descreva os personagens a serem correspondidos literalmente. Assim, o regular expressão / java / corresponde a qualquer string que contenha a substring "Java". Outros personagens em expressões regulares não são correspondidos literalmente e têm significado especial. Por exemplo, a expressão regular / s \$ / contém dois caracteres. O primeiro, "S", se combina literalmente. O segundo, "\$", é um meta-caractere especial que corresponde ao fim de uma corda. Assim, essa expressão regular corresponde a qualquer string que contenha a letra "S" como seu último personagem. Como veremos, expressões regulares também podem ter uma ou mais bandeiras que afetam como eles funcionam. As bandeiras são especificadas após o segundo argumento para o construtor `regexp()`. Se quiséssemos combinar strings isso terminou com "s" ou "s", por exemplo, poderíamos usar a bandeira de `i` com nosso `Expressão regular` para indicar que queremos correspondência insensível ao caso: Deixe padrão = /s \$ /i; As seções a seguir descrevem os vários caracteres e meta-caracteres usados ?? em expressões regulares JavaScript. Personagens literais Todos os personagens e dígitos alfabéticos se combinam literalmente em

expressões regulares. A sintaxe de expressão regular JavaScript também suporta certos caracteres não alfabéticos através de sequências de fuga que começam com uma barra de barriga (\). Por exemplo, a sequência \n corresponde a um literal NEWLINE CARACTER em uma string. A Tabela 11-1 lista esses caracteres. Tabela 11-1. Personagens literais de expressão regular

Character	Partidas	Alfabeto	Umérico	Character	Em si	O caractere	nu (\u0000)	t	Guia (\u0009)	n	Newline (\u000a)	v	Guia vertical (\u000b)	f	Formulário de formulário (\u000c)	r	Retorno de carroagem (\u000d)	xnn	O caractere latino especificado pelo número hexadecimal nn; por exemplo, \x0a é o mesmo que \n.	\uxxxx	O caractere unicode especificado pelo número hexadecimal xxxx; para exemplo, \u0009 é o mesmo que \t.	\un	O caractere unicode especificado pelo codepoint n, onde n é um a seis dígitos hexadecimais entre 0 e 10ffff. Observe que esta sintaxe é apenas suportado em expressões regulares que usam o sinalizador U.	\cx	O caractere de controle ^x; Por exemplo, \cj é equivalente à nova linha caractere \n.
-----------	----------	----------	---------	-----------	-------	-------------	-------------	---	---------------	---	------------------	---	------------------------	---	-----------------------------------	---	-------------------------------	-----	---	--------	---	-----	--	-----	---

Vários caracteres de pontuação têm significados especiais em regulares expressões. Eles são: ^ \$. * + ? = ! : | \ / () [] {} Os significados desses personagens são discutidos nas seções que seguir. Alguns desses personagens têm significado especial apenas dentro certos contextos de uma expressão regular e são tratados literalmente em outros contextos. Como regra geral, no entanto, se você quiser incluir qualquer um dos desses personagens de pontuação literalmente em uma expressão regular, você deve precedê-los com um \. Outros personagens de pontuação, como citação Marcas e @, não têm significado especial e simplesmente combinam os mesmos literalmente em uma expressão regular. Se você não consegue se lembrar exatamente de quais personagens de pontuação precisam ser escapados com uma barra de barracaracter de pontuação. Por outro lado, observe que muitas letras e os números têm significado especial quando precedidos por uma barra de barriga, então qualquer letra ou número que você deseja combinar literalmente não deve ser escapado com uma barra de barriga. Para incluir um personagem de barragem literalmente em um Expressão regular, você deve escapar dela com uma barra de barriga, é claro. Para exemplo, a expressão regular seguinte corresponde a qualquer string que inclui uma barra de barra: \ \ /. (E se você usar o regexp () Construtor, lembre -se de que qualquer barra de barriga regular A expressão precisa ser dobrada, pois as strings também usam barras de barriga como um fuga de caráter.) Classes de personagens Caracteres literais individuais podem ser combinados em classes de personagens por

colocando -os dentro de colchetes.Uma classe de personagem corresponde a qualquer um personagem que está contido nele.Assim, a expressão regular/ [ABC]/ corresponde a qualquer uma das letras a, b ou c.Caráter negadoAs classes também podem ser definidas;estes correspondem a qualquer personagem, exceto aqueles contidos dentro dos colchetes.Uma classe de caracteres negada é especificada por Colocando um cuidador (^) como o primeiro caractere dentro do suporte esquerdo.O Regexp / [^abc] / corresponde a qualquer caractere que não seja a, b ou c.As classes de caracteres podem usar um hífen para indicar uma variedade de caracteres.Para Combine qualquer caractere minúsculo do alfabeto latino, use / [a-z] /, e para combinar com qualquer carta ou dígito do alfabeto latino, use / [A-Za-Z0-9] /. (E se você quiser incluir um hífen real em sua aula de personagem, basta torná-lo o último personagem antes do direito suporte.) Como certas classes de caracteres são comumente usadas, o javascript A sintaxe de expressão regular inclui caracteres especiais e fuga Sequências para representar essas classes comuns.Por exemplo, \ s corresponde ao personagem espacial, ao caractere da guia e qualquer outro unicod e caráter de espaço em branco;\ S corresponde a qualquer personagem que não seja unicod e espaço em branco.A Tabela 11-2 lista esses personagens e resume Sintaxe da classe de caracteres.(Observe que vários desses caracters As sequências de fuga combinam apenas com caracteres ASCII e não foram estendido para trabalhar com caracteres unicode.Você pode, no entanto, definir explicitamente suas próprias classes de personagens Unicode;por exemplo,/ [\ u0400- \ u04ff] / corresponde a qualquer caractere cirílico.)Tabela 11-2.Classes de personagens de expressão regular Cap

aracterPartidas[...]Qualquer um personagem entre os colchetes.[^...]Qualquer personagem não entre os colchetes..Qualquer caractere, exceto a nova linha ou outro terminador de linha Unicode.Ou, se oRegexp usa a bandeira S, então um período corresponde a qualquer caractere, incluindo linhaTerminadores.\cQualquer personagem de palavra ascii.Equivalente a [A-ZÀ-ZÒ-9_].\CQualquer personagem que não seja um personagem da palavra ascii.Equivalente a [^a-zA-Z0-9_].\sQualquer caractere de espaço em branco Unicode.\SQualquer personagem que não seja Unicode WhiteSpace.\dQualquer dígito ASCII.Equivalente a [0-9].\DQualquer personagem que não seja um dígito ASCII.Equivalente a [^0-9].\[B]Um backspace literal (caso especial).Observe que as fugas especiais da classe de caracteres podem ser usadas no quadradoSuportes.\s corresponde a qualquer caractere de espaço em branco e \d corresponde a qualqueDigit, SO / [\s\d] / corresponde a qualquer caractere ou dígito em branco.Observe que há um caso especial.Como você verá mais tarde, o \b escapetem um significado especial.Quando usado em uma classe de personagem, no entanto, érepresenta o caractere de backspace.Assim, para representar um backspacepersonagem literalmente em uma expressão regular, use a classe de personagem com

Um elemento: /[\b] /.Classes de caracteres UnicodeNo ES2018, se uma expressão regular usa a bandeira u, então as classes de caracteres \p {...} e sua negação\ P {...} são suportados.(No início de 2020, isso é implementado por nós, Chrome, Edge e Safari,mas não o Firefox.) Essas classes de caráter são baseadas em propriedades definidas pelo padrão Unicode,e o conjunto de caracteres que eles representam pode mudar à medida que o Unicode evolui.A classe de caracteres \D corresponde apenas aos dígitos ASCII.Se você quiser combinar um dígito decimal de qualquer um dosOs sistemas de escrita do mundo, você pode usar \p {decimal_number} /u.E se você quiser corresponder a algumUm personagem que não é um dígito decimal em nenhum idioma, você pode capitalizar o P e escrever\ P {decimal_number}.Se você deseja combinar com qualquer caráter semelhante ao número, incluindo frações eNumerais romanos, você pode usar \p {número}.Observe que "decimal_number" e "número" não são específicos para JavaScript ou para a Gramática de Expressão regular: é o nome de uma categoria de caracteresdefinido pelo padrão Unicode.A classe de caracteres \w funciona apenas para texto ascii, mas com \p, podemos aproximar umVersão internacionalizada como esta:/[\p {alfabético}\p {decimal_number}\p {mark}]/u(Embora seja totalmente compatível com a complexidade dos idiomas do mundo, precisamos realmente adicionarAs categorias também ?Connector_Punctuation? e ?junção_control?).)Como exemplo final, a sintaxe \p também nos permite definir expressões regulares que correspondem aoscaracteresDe um determinado alfabeto ou script:Seja greekLetter = \p {script = grego} /u;Seja CyrillicLetter = \p {script = Cirillic} /u;REPETIÇÃOCom a sintaxe de expressão regular que você aprendeu até agora, você pode descrever um número de dois dígitos como /\d\d / e um número de quatro dígitos como\ d\d\d\d/.Mas você não tem nenhuma maneira de descrever, por exemplo, umNúmero que pode ter qualquer número de dígitos ou uma sequência de três letrasseguido de um dígito opcional.Esses padrões mais complexos usam regularesExpressão sintaxe que especifica quantas vezes um elemento de um

A expressão regular pode ser repetida. Os personagens que especificam a repetição sempre seguem o padrão para que eles estão sendo aplicados. Porque certos tipos de repetição são bastante comumente usado, existem caracteres especiais para representar esses casos. Por exemplo, + corresponde a uma ou mais ocorrências do anterior padrão. A Tabela 11-3 resume a sintaxe da repetição.

Tabela 11-3. Personagens regulares de repetição de expressãoCharACTERSignificado{*n*, *m*}Combine o item anterior pelo menos *N* vezes, mas não mais que *M* vezes.{*n*,}Combine o item anterior *n* ou mais vezes.{*n*}Combine exatamente *n* ocorrências do item anterior.?Combine zero ou uma ocorrência do item anterior. Isto é, o anterior O item é opcional. Equivalente a {0,1}.+Combine uma ou mais ocorrências do item anterior. Equivalente a {1,}.*Combine zero ou mais ocorrências do item anterior. Equivalente a {0,}. As linhas a seguir mostram alguns exemplos:

Seja $r = \wedge d \{2,4\}$;// combina entre dois e quatro dígitos
 $r = \wedge w \{3\} \backslash d?$;// corresponde exatamente a três caracteres de palavras e um dígito opcional
 $r = \wedge s+java \backslash s+$;// combina "Java" com um ou mais espaçosantes e depois

r = /[^\n]* /; // corresponde a zero ou mais caracteres que não estão abertos entre parênteses. Observe que em todos esses exemplos, os especificadores de repetição se aplicam ao Classe de personagem ou personagem único que os precede. Se você quiser combinar repetições de expressões mais complicadas, você precisará Definir um grupo com parênteses, que são explicados no seguinte exemplo. Tenha cuidado ao usar o * e ? Personagens de repetição. Desde estes Os personagens podem corresponder a zero instâncias de qualquer que seja o que precede, eles podem não combinar nada. Por exemplo, a expressão regular / a */ Na verdade, corresponde à string "bbbb" porque a string contém zero ocorrências da letra A! Repetição sem graça Os caracteres de repetição listados na Tabela 11-3 correspondem quantas vezes possível enquanto ainda permite qualquer parte seguinte do regularexpressão para corresponder. Dizemos que essa repetição é "gananciosa". É também possível especificar que a repetição deve ser feita de maneira não-greedosa. Basta seguir o personagem ou personagens de repetição com uma perguntaMark: ??, +?, *?, ou mesmo {1,5}?. Por exemplo, o regularexpressão / a+ / corresponde a uma ou mais ocorrências da letra a. Quando aplicado à string "AAA", ela corresponde a todas as três letras. Mas / a+? / corresponde a uma ou mais ocorrências da letra A, combinando com poucos caracteres conforme necessário. Quando aplicado à mesma string, este padrão corresponde apenas à primeira letra a. Usar a repetição sem graça pode nem sempre produzir os resultados que você

Erro ao traduzir esta página.

/ java (script)?/ Matches ?java? seguido pelo opcional "Script".E / (ab | cd)+| ef / corresponde à string "ef" ou um ou mais repetições de qualquer uma das cordas "ab" ou "cd". Outro objetivo dos parênteses em expressões regulares é definir subpadrões dentro do padrão completo. Quando uma expressão regular é combacto com sucesso contra uma sequência de destino, é possível extrair opartes da sequência de destino que correspondiam a qualquer parênteses em particularSubpaterno. (Você verá como essas substringas correspondentes são obtidas mais tarde nesta seção.) Por exemplo, suponha que você esteja procurando um ouMais letras minúsculas seguidas por um ou mais dígitos. Você pode usar o padrão /[a-z]+\\d+. Mas suponha que você realmente se importe com os dígitos no final de cada partida. Se você colocar essa parte do padrão em parênteses ([a-z]+(\\d+)), você pode extrair os dígitos de qualquerMatches que você encontra, como explicado mais tarde. Um uso relacionado de subexpressões entre parênteses é permitir que você consulte de volta a uma subexpressão mais tarde na mesma expressão regular. Isso é feito seguindo um caractere \\ por um dígito ou dígitos. Os dígitos se referem à posição da subexpressão entre parênteses dentro do regularexpressão. Por exemplo, \\1 refere -se à primeira subexpressão e \\3 refere -se ao terceiro. Observe que, porque as subexpressões podem ser aninhadasDentro de outros, é a posição dos parênteses esquerdos que são contados. Em uma seguinte expressão regular, por exemplo, a aninhadaA subexpressão (crívio [SS]) é referida como \\2:/(Jj) ava ([ss] crip?) \\S \\s (diversão \\w*)/Uma referência a uma subexpressão anterior de uma expressão regular faz

não se refere ao padrão para essa subexpressão, mas sim ao texto que combinou com o padrão. Assim, referências podem ser usadas para aplicar umRestrições que partes separadas de uma corda contêm exatamente o mesmocaracteres.Por exemplo, a seguinte expressão regular corresponde a zeroou mais caracteres dentro de citações únicas ou duplas.No entanto, não requer as cotações de abertura e fechamento para combinar (ou seja, ambos solteiroscitações ou ambas as citações duplas):/[""] [^"]*[""]/Para exigir as cotações para corresponder, use uma referência:/([""])[^"]*\1/O \1 corresponde a qualquer que a primeira subexpressão entre parênteses combinado.Neste exemplo, ele aplica a restrição de que o fechamentoCitação corresponde à cotação de abertura.Esta expressão regular não permiteCitações únicas em seqüências de cordas duplas ou vice-versa.(Não é legalPara usar uma referência em uma classe de caracteres, para que você não possa escrever:/([""])[^"]*\1/.)Quando cobrimos a API regexp mais tarde, você verá que esse tipo deReferência a uma subexpressão entre parênteses é uma característica poderosa deOperações de pesquisa e substituição de expressão regular.Também é possível agrupar itens em uma expressão regular sem criando uma referência numerada a esses itens.Em vez de simplesmente agrupando os itens dentro (e), inicie o grupo com (? : e termine com).Considere o seguinte padrão:

/([Jj] ava (? : [Ss] crip)?) \ Sis \ s (diversão \ w*)/Neste exemplo, a subexpressão (? : [ss] crívio) é usada simplesmente para agrupamento, então o? O caractere de repetição pode ser aplicado ao grupo. Esses parênteses modificados não produzem uma referência, portanto, neste expressão regular, \ 2 refere-se ao texto correspondente por (Fun \ W*). A Tabela 11-4 resume a alternância regular de expressão, agrupamento, e operadores de referência. Tabela 11-4. Alternância regular de expressão, agrupamento e caracteres de referência Chumrumcter Significado|Alternância: corresponda à subexpressão à esquerda ou à subexpressão aocerto. (...) Agrupamento: Agrupe itens em uma única unidade que pode ser usada com *, +, ?, |, E assim sobre. Lembre-se também dos personagens que correspondem a esse grupo para uso com mais tarde Referências. (? : ...) Apenas agrupamento: agrupar itens em uma única unidade, mas não se lembro dos personagens que correspondem a este grupo.

\nCorresponder aos mesmos personagens que foram correspondidos quando o número n foi o primeiro combinado. Os grupos são subexpressões dentro de parênteses (possivelmente aninhados). Grupo Os números são atribuídos contando parênteses esquerdos da esquerda para a direita. Grupos formados com (? : não estão numerados. Nomeados grupos de captura O ES2018 padroniza um novo recurso que pode tornar as expressões regulares mais autodocumentadoras e mais fáceis de entender. Este novo recurso é conhecido como "Grupos de captura nomeado" e nos permite associar um nome a cada parêntese esquerdo em uma expressão regular para que possamos nos referir ao Texto correspondente por nome e não por número. Igualmente importante: o uso de nomes permite que o código para entender mais facilmente o objetivo dessa parte da expressão regular. Como No início de 2020, esse recurso é implementado em Node.js, Chrome, Edge e Safari, mas ainda não pelo Firefox. Para citar um grupo, use (? <...>) em vez de (e coloque o nome entre os suportes de ângulo). Para exemplo, aqui está uma expressão regular que pode ser usada para verificar a formatação da linha final de um Endereço de correspondência dos EUA: /(? <City> \w+) (? <state> [a-z]{2}) (? Observe quanto contexto os nomes de grupos fornecem para facilitar a expressão regular entender. No §11.3.2, quando discutirmos os métodos String repl.replace () e correspondência () e os Método regexp EXEC (), você verá como a API REGEXP permite que você consulte o texto que corresponde Cada um desses grupos por nome e não por posição. Se você quiser se referir a um grupo de captura nomeado dentro de uma expressão regular, você pode fazer isso por nome também. No exemplo anterior, fomos capazes de usar uma expressão regular de "referência" para Escreva um regexp que corresponesse a uma string única ou dupla, onde as citações abertas e fechadas estejam combinadas. Poderíamos reescrever este regexp usando um grupo de captura nomeado e um nome nomeado Referência de fundo como esta: /(?) O \k <-Quote> é uma referência de volta nomeada para o grupo nomeado que captura a citação aberta marca. Especificando a posição da correspondência Como descrito anteriormente, muitos elementos de uma expressão regular correspondem a um caractere único em uma string. Por exemplo, \s corresponde a um único personagem de espaço em branco. Outros elementos regulares de expressão correspondem às posições

entre caracteres em vez de caracteres reais.\ b, por exemplo, corresponde a um limite da palavra ascii - o limite entre um \ w (Caractere da palavra ascii) e a \ w (caráter não -palavras), ou o limite entre um personagem de palavra ascii e o início ou o fim de uma corda. Elementos como \ b não especificam nenhum caractere a ser usado em uma string correspondente; O que eles especificam, no entanto, são posições legais no qual pode ocorrer uma partida. Às vezes, esses elementos são chamados âncoras de expressão regular porque ancoram o padrão a uma posição específica na sequência de pesquisa. A âncora mais usada é ^, que liga o padrão ao início da corda, e \$, que ancora o padrão até o final da string. Por exemplo, para combinar a palavra "javascript" em uma linha por si só, você pode usar a expressão regular /^javascript \$/. Se você quiser procurar por "java" como uma palavra por si só (não como um prefixo, como está em "JavaScript"), você pode experimentar o padrão ^ s java \ s /, que requer um espaço antes e depois da palavra. Mas há dois problemas com essa solução. Primeiro, ele não corresponde a "java" no início ou no final de uma string, mas apenas se aparecer com espaço de ambos os lados. Segundo, quando esse padrão encontra uma correspondência, a string correspondente que ele retorna tem liderança e espaços à direita, o que não é exatamente o que é necessário. Então, em vez de combinando caracteres espaciais reais com \ s, corresponda (ou âncora) a palavras limites com \ b. A expressão resultante é ^ b java \ b /. O elemento \ b ancora a partida em um local que não é uma palavra limite. Assim, o padrão / \ b [ss] crivo / corresponde a "JavaScript" e "PostScript", mas não "script" ou "script". Você também pode usar expressões regulares arbitrárias como condições de âncora. Se

você inclui uma expressão dentro (? = e) caracteres, é uma afirmação lookahead, e especifica que os personagens fechados devemCombine, sem realmente combiná -los. Por exemplo, para corresponder a nome de uma linguagem de programação comum, mas apenas se for seguida por um colón, você poderia usar /jj] ava ([ss] crívio)? (? = \ :) /. Esse padrão corresponde à palavra "javascript" em "javascript: o definitivoGuia ", mas não corresponde a " java " em poucas palavras "porque não é seguido por um colón. Se você apresentar uma afirmação com (?!, É um negativoAfirmation de Lookahead, que especifica que os seguintes caracteres devem não corresponde. Por exemplo, /java (?! Script) ([a-z] \ w*) /Matches ?Java? seguido de uma letra maiúscula e qualquer número de Personagens adicionais da palavra ascii, desde que "java" não seja seguido por "Script". Combina "Javabeans", mas não "Javanese", e corresponde "JavaScript", mas não "JavaScript" ou "JavaScript". Tabela 11-5 resume as âncoras de expressão regular. Tabela 11-5. Caracteres de âncora de expressão regularCharumCterSignificado^Combine o início da corda ou, com a bandeira M, o início de uma linha.\$Combine o final da corda e, com a bandeira M, o final de uma linha.\bCombine um limite da palavra. Isto é, combine a posição entre um personagem \ w e um caractere ou entre um caractere \ w e o início ou o fim de uma string. (Observe, no entanto, que [\ b] corresponde ao backspace.)

\BCombine uma posição que não é um limite de palavra.(?= p)Uma afirmação positiva.Exigir que os seguintes caracteres correspondam ao Padrão P, mas não inclua esses personagens na partida.(?!p)Uma afirmação negativa de loteahead.Exigir que os seguintes caracteres não corresponder ao padrão p.ASSERÇÕES LOLHEBEHINDO ES2018 estende a sintaxe de expressão regular para permitir as afirmações "Lookbehind".Estes são comoLookaHeadAsserções, mas consulte o texto antes da posição atual da correspondência.No início de 2020, estes são implementados no Chrome e Edge, mas não Firefox ou Safari.Especifique uma afirmação positiva de aparência com (? <= ...) e uma afirmação de aparência negativa com (?<! ...).Por exemplo, se você estivesse trabalhando conosco endereços de correspondência, poderá combinar umzip de 5 dígitosCódigo, mas somente quando segue uma abreviação estadual de duas letras, como esta:/(? <= [A-z] {2}) \ d {5}/E você pode combinar uma série de dígitos que não são precedidos por um símbolo de moeda unicode com umAfirmação negativa do LookBehind como esta:/(?<!BandeirasToda expressão regular pode ter uma ou mais bandeiras associadas a elepara alterar seu comportamento correspondente.JavaScript define seis bandeiras possíveis,cada um dos quais é representado por uma única letra.As bandeiras são especificadas depois segundo / caráter de uma expressão regular literal ou como uma cordapassou como o segundo argumento para o construtor regexp ().O

Bandeiras suportadas e seus significados são:
gA bandeira G indica que a expressão regular é "global" - ou seja, que pretendemos usá-lo para encontrar todas as correspondências em uma string, em vez de apenas encontrar a primeira partida. Esta bandeira não altera o caminho. Essa correspondência de padrões é feita, mas, como veremos mais tarde, ele altera o comportamento do método String correponde () e o regexp método EXEC () de maneiras importantes.
euA bandeira I especifica que a correspondência de padrões deve ser casoinsensível.
mA bandeira M especifica que a correspondência deve ser feita em "Multiline" modo. Diz que o regexp será usado com cordas multilineas que as âncoras ^ e \$ devem corresponder ao começo e final da corda e também o começo e o fim das linhas individuais dentro da string.
sComo a bandeira M, a bandeira S também é útil ao trabalhar com textos que incluem novas linhas. Normalmente, um "" em uma expressão regular corresponde a qualquer caractere, exceto um terminador de linha. Quando a bandeira S é usada, no entanto, "" vai corresponder a qualquer personagem, incluindo terminadores de linha.
A bandeira S foi adicionada ao JavaScript no ES2018 e, com o início de 2020, é apoiado em navegadores como Google Chrome, Mozilla Firefox, Safari, mas não em Internet Explorer.
uA bandeira U significa Unicode e faz a expressão regular

Erro ao traduzir esta página.

11.3.2 Métodos de string para correspondência de padrõesAté agora, descrevemos a gramática usada para definir regularmente expressões, mas não explicando como essas expressões regulares podem realmente ser usado no código JavaScript.Agora estamos mudando para cobrir o API para usar objetos regexp.Esta seção começa explicando os métodos de string que usam expressões regulares para executar o padrãoOperações de correspondência e pesquisa e substituição.As seções que se seguemEste continua a discussão sobre o padrão de correspondência com JavaScriptExpressões regulares discutindo o objeto Regexp e seus métodos e propriedades.PROCURAR()As cadeias suportam quatro métodos que usam expressões regulares.O mais simples é pesquisa ().Este método leva um argumento de expressão regular e retorna a posição do personagem do início da primeira correspondência substring ou -1 se não houver correspondência:"Javascript" .search (/script/ui) // => 4"Python" .search (/script/ui) // => -1Se o argumento a searar () não é uma expressão regular, é o primeiro convertido a um passando -o para o construtor regexp.procurar()não suporta pesquisas globais;Ignora a bandeira G de seu regularargumento de expressão.SUBSTITUIR()O método substitui () executa uma operação de pesquisa e substituição.Isto toma uma expressão regular como seu primeiro argumento e uma corda de substituição

como seu segundo argumento. Ele pesquisa a string na qual é necessário corresponde ao padrão especificado. Se a expressão regular tiver o gConjunto de sinalizadores, o método substituir () substitui todas as correspondências na string com a sequência de substituição; Caso contrário, ele substitui apenas a primeira partida encontra. Se o primeiro argumento a substituir () é uma string em vez de um Expressão regular, o método procura por essa corda literalmente do que convertê-lo em uma expressão regular com o regexp () Construtor, como search () faz. Como exemplo, você pode usar substituir () como segue para fornecer capitalização uniforme da palavra "JavaScript" em uma série de texto:// Não importa como seja capitalizado, substitua -o pelo capitalização correta text.Replace (/javascript/gi, "javascript"); Substituir () é mais poderoso que isso, no entanto. Lembre -se disso As subexpressões entre parênteses de uma expressão regular são numeradas da esquerda para a direita e que a expressão regular se lembra do texto que cada subexpressão corresponde. Se um \$ seguido de um dígito aparecer em A corda de substituição, substitua () substitui esses dois caracteres com o texto que corresponde à subexpressão especificada. Isso é muitorecurso útil. Você pode usá-lo, por exemplo, para substituir as aspas em uma string com outros personagens:// Uma cotação é uma cotação, seguida por qualquer número de// caracteres de marca não-detentora (que capturamos), seguidos// por outra cotação. Seja quote = /"([""])*") " /g; // Substitua as aspas retas por GuilleMets// deixando o texto citado (armazenado em US \$ 1) inalterado.'Ele disse "pare". Substitua (citação, '«\$1»') // => 'ele disse"parar'"

Se o seu regexp usar grupos de captura nomeados, você poderá se referir ao Texto correspondente por nome e não por número: Seja quote = /"(?'Ele disse "pare".disse «pare» 'Em vez de passar uma corda de substituição como o segundo argumento para substituir (), você também pode passar uma função que será invocada para calcular o valor de reposição. A função de reposição é invocada com vários argumentos. Primeiro é todo o texto correspondente. Em seguida, se o regexp tem grupos de captura, depois as substâncias que eram capturadas por esses grupos são passadas como argumentos. O próximo argumento é a posição dentro da sequência na qual a partida foi encontrada. Depois disso, a sequência inteira que substitui () foi chamada e passada. E finalmente, se o regexp continha algum grupo de captura nomeado, o último argumento para a função de reposição é um objeto cuja propriedade Os nomes correspondem aos nomes dos grupos de captura e cujos valores são o texto correspondente. Como exemplo, aqui está o código que usa uma substituição de função para converter números inteiros decimais em uma string em hexadecimal: Seja s = "15 vezes 15 é 225"; s.Replace (\d+/gu, n => ParseInt (n) .ToString (16)) // => "fvezes f é e1 "CORRESPONDER() O método Match () é o mais geral da string regular. Métodos de expressão. É preciso uma expressão regular como seu único argumento (ou converte seu argumento em uma expressão regular, passando para o REGEXP () construtor) e retorna uma matriz que contém os resultados

da partida, ou nulo se nenhuma correspondência for encontrada.Se a expressão regular tem o conjunto de bandeira G, o método retorna uma matriz de todas as correspondências que aparecer na string.Por exemplo:"7 mais 8 é igual a 15" .Match (\wedge d+/g) // => ["7", "8", "15"]Se a expressão regular não tiver o conjunto de bandeira G, corresponde ()não fazer uma pesquisa global;Simplesmente procura a primeira partida.Nesta Caso Nonglobal, Match () ainda retorna uma matriz, mas os elementos da matriz são completamente diferentes.Sem a bandeira G, o primeiro elemento doArray retornado é a string correspondente e quaisquer elementos restantes são as substrings que correspondem aos grupos de captura entre parênteses da expressão regular.Assim, se a correspondência () retornar uma matriz A, a [0] contém a correspondência completa, a [1] contém a substring que corresponde à primeira expressão entre parênteses e assim por diante.Para desenhar um paralelo com o método substituir (), a [1] é a mesma string que \$ 1, a [2] é o mesmo que US \$ 2, e assim por diante.Por exemplo, considere analisar um URL com o seguinte código:// Um ??URL muito simples parsing regexpvamos url = /(\w+):V\W([\w.]+)V(\s**)/;Let Text = "Visite meu blog em http://www.example.com/~david";Seja Match = Text.match (URL);Deixe Fullurl, protocolo, host, caminho;if (correspondência == null) {FullUrl = Match [0];// Fullurl == "http://www.example.com/~david"protocolo = correspondência [1];// protocolo == "http"host = correspondência [2];// host == "www.example.com"caminho = correspondência [3];// caminho == "~ David"}5

Neste caso não global, a matriz retornada por match () também tem algunsPropriedades do objeto Além dos elementos de matriz numerados.OA propriedade de entrada refere -se à string na qual correspondência () foi chamada.A propriedade do índice é a posição dentro daquela string na qual oa partida começa.E se a expressão regular contiver captura nomeadagrupos, então a matriz retornada também tem uma propriedade de grupos cujavalor é um objeto.As propriedades deste objeto correspondem aos nomes doGrupos nomeados e os valores são o texto correspondente.Poderíamos reescreverO exemplo anterior de análise de URL, por exemplo, como este:Vamos url =/(?<TACH> \ S*)/;Let Text = "Visite meu blog em http://www.example.com/~david";Seja Match = Text.match (URL);Combine [0] // => "http://www.example.com/~david"match.input // => textomatch.index // => 17match.groups.protocol // => "http"match.groups.host // => "www.example.com"match.groups.path // => "~ David"Vimos que a correspondência () se comporta de maneira bastante diferente, dependendo dese o regexp tem o conjunto de sinalizadores G ou não.Também existem importantesMas diferenças menos dramáticas no comportamento quando o sinalizador Y é definido.Lembrarque a bandeira y faz uma expressão regular "pegajosa" restringindo onde nas correspondências da string pode começar.Se um regexp tem o G eY Sinalizadores definidos, depois corresponde () retorna uma variedade de cordas correspondentes, exatamentecomoFaz quando G é definido sem y.Mas a primeira partida deve começar noinício da string, e cada partida subsequente deve começar noPersonagem imediatamente após a partida anterior.

Se a bandeira `y` estiver definida sem `g`, então corresponde () tentar encontrar um único corresponder e, por padrão, esta partida é restrita ao início docorda.Você pode alterar esta posição de início de correspondência padrão, no entanto, porDefinindo a propriedade `LastIndex` do objeto `regexp` no índice emem que você deseja combinar.Se uma partida for encontrada, então este `LastIndex` será atualizado automaticamente para o primeiro caractere após a partida, então sevocê liga para `match ()` novamente, neste caso, ele procurará um subsequentecorresponder.(`Lastindex` pode parecer um nome estranho para uma propriedade queEspecifica a posição na qual iniciar a próxima partida.Vamos ver issoNovamente quando cobrimos o método `regexp exec ()`, e seu nome podefaça mais sentido nesse contexto.)Seja vogal = `/[aeiou] /y;`// partida de vogal pegajosa"teste" .`match (vogal) // => null:` "teste" não começacom uma vogalvogal.`LastIndex = 1;`// Especifique uma correspondência diferenteposição"teste" .`Match (vogal) [0] // => "e":` encontramos uma vogal emPosição 1vogal.`lastindex // => 2:` o `LastIndex` foi automaticamenteatualizado"Teste" .`Match (vogal) // => nulo:` nenhuma vogal na posição 2vogal.`lastindex // => 0:` `LastIndex` é redefinido apósfalhou correspondênciaVale a pena notar que passar uma expressão regular não global para o`Match ()` Método de uma string é o mesmo que passar a string para o`method exec ()` da expressão regular: a matriz retornada e seuAs propriedades são as mesmas nos dois casos.`Matchall ()`O método `matchall ()` é definido no ES2020 e no início de 2020

é implementado pelos modernos navegadores da web e nó. Matchall () espera um regexp com o conjunto de sinalizadores G. Em vez de devolver uma variedade de Subtramentos correspondentes como Match (), no entanto, retorna um iterador que produz o tipo de correspondência que corresponde () retorna quando usado com um regexp não global. Isso torna o Matchall () o mais fácil e maneira mais geral de percorrer todas as partidas dentro de uma string. Você pode usar o matchall () para percorrer as palavras em uma sequência de texto como este:// um ou mais caracteres alfabéticos unicode entre a palavra limites const words = /\b \p{alfabetico}+\b /gu; // \p não é suportado no Firefox ainda const text = "Este é um teste ingênuo do Matchall () método."; para (deixe a palavra de texto.matchall (palavras)) {console.log (` encontrado '\$ {word [0]}' no índice \$ {word.index} . `); } Você pode definir a propriedade LastIndex de um objeto regexp para dizer matchall () em qual índice na string para começar a corresponder. Diferente os outros métodos de correspondência de padrões, no entanto, matchall () nunca modifica a propriedade LastIndex do regexp que você chama isso torna muito menos a probabilidade de causar bugs em seu código. DIVDIR() O último dos métodos de expressão regular do objeto String é dividir(). Este método quebra a string na qual é chamada em um matriz de substrings, usando o argumento como um separador. Pode ser usado

com um argumento de string como este:"123.456.789" .split (",") // => ["123", "456", "789"]O método split () também pode assumir uma expressão regular como seu argumento, e isso permite especificar separadores mais gerais.AquiNós o chamamos com um separador que inclui uma quantidade arbitrária deEspaço em branco de ambos os lados:"1, 2, 3, \ n4, 5" .split (\ s*, \ s*) // => ["1", "2", "3", "4", "5"]Surpreendentemente, se você ligar para Split () com um delimitador regexp e oA expressão regular inclui capturar grupos, depois o texto que correspondeOs grupos de captura serão incluídos na matriz devolvida.Para exemplo:const htmlTag = /<([>]+)> /; // <seguido por um ou mais não>, seguido por>"Teste
 1,2,3" .Split (htmlTag) // => ["teste", "br/", "1,2,3"]11.3.3 A classe RegexpEsta seção documenta o construtor regexp (), as propriedades deInstâncias de regexp e dois métodos importantes de correspondência de padrões definido pela classe regexp.O construtor regexp () leva um ou dois argumentos de string e cria um novo objeto regexp.O primeiro argumento a este construtor é umstring que contém o corpo da expressão regular - o texto que

apareceria em barras em uma literal regular de expressão. Observe que tanto os literais de cordas quanto as expressões regulares usam o caractere \ para sequências de fuga, então quando você passa uma expressão regular para `RegExp()` como uma string literal, você deve substituir cada caractere \ por \\. O segundo argumento para `regexp()` é opcional. Se for fornecido, indica os sinalizadores de expressão regular. Deve ser g, i, m, s, u, y ou qualquer combinação dessas cartas. Por exemplo:// Encontre todos os números de cinco dígitos em uma string. Observe o dobro \\ nesse caso. Seja `zipcode = new regexp ("\\d {5}", "g")`; O construtor `regexp()` é útil quando uma expressão regular é sendo criado dinamicamente e, portanto, não pode ser representado com a expressão regular sintaxe literal. Por exemplo, para procurar uma string digitada pelo usuário, uma expressão regular deve ser criada em tempo de execução com `regexp()`. Em vez de passar uma string como o primeiro argumento para `regexp()`, você pode passar também um objeto `regexp`. Isso permite que você copie um regular expression e altere suas bandeiras: Let `ExactMatch = /JavaScript/`; Deixe `CaseInsensitive = novo regexp (exactmatch, "i")`; Propriedades `regexp` Os objetos `regexp` têm as seguintes propriedades:

fonteEsta propriedade somente leitura é o texto de origem da expressão regular:Os personagens que aparecem entre as barras em um literal regexp.bandeirasEsta propriedade somente leitura é uma string que especifica o conjunto de letras que representam os sinalizadores para o regexp.globalUma propriedade booleana somente leitura que é verdadeira se o sinalizador G estiver definido.ignorecaseUma propriedade booleana somente leitura que é verdadeira se o sinalizador I estiver definido.MultilineUma propriedade booleana somente leitura que é verdadeira se o sinalizador M estiver definido.DotallUma propriedade booleana somente leitura que é verdadeira se o sinalizador S estiver definido.unicodeUma propriedade booleana somente leitura que é verdadeira se o sinalizador U estiver definido.pegajosoUma propriedade booleana somente leitura que é verdadeira se o sinalizador Y estiver definido.LastIndexEsta propriedade é um número inteiro de leitura/gravação.Para padrões com o g ou ybandeiras, ele especifica a posição do personagem na qual a próxima pesquisa é para começar.É usado pelos métodos EXEC () e Test (),descrito nas próximas duas subseções.

TESTE()O método test () da classe regexp é a maneira mais simples de usar umexpressão regular.É preciso um único argumento de string e retorna verdadeiroSe a string corresponder ao padrão ou falsa se não corresponder.teste () funciona simplesmente chamando o (muito mais complicado)método exec () descrito na próxima seção e retornando true seEXEC () retorna um valor não nulo.Por causa disso, se você usar o teste ()Com um regexp que usa as bandeiras G ou Y, seu comportamento depende o valor da propriedade LastIndex do objeto regexp, que pode mudar inesperadamente.Veja ?A propriedade LastIndex e RegexpReutilize ?para mais detalhes.Exec ()O método regexp EXEC () é a maneira mais geral e poderosa deUse expressões regulares.É preciso um único argumento de string e procura uma partida nessa string.Se nenhuma correspondência for encontrada, ele retornará nulo.Se uma partidaé encontrada, no entanto, ele retorna uma matriz exatamente como a matriz devolvida peloMatch () Método para pesquisas não globais.Elemento 0 da matrizcontém a string que correspondia à expressão regular e qualquerOs elementos de matriz subsequentes contêm as substringas que correspondiam a qualquercaptura de grupos.A matriz devolvida também possui propriedades: oA propriedade de índice contém a posição do caractere na qual a partida ocorreu, e a propriedade de entrada especifica a string que foi pesquisado, e a propriedade dos grupos, se definida, refere -se a um objeto quemantém as substringas que correspondem aos grupos de captura qualquer nome.

Ao contrário do método String Match (), EXEC () retorna o mesmo tipo deArray se a expressão regular tem ou não a bandeira G global. Lembre -se de que o Match () retorna uma variedade de partidas quando passou um globalexpressão regular. EXEC (), por outro lado, sempre retorna uma única partida e fornece informações completas sobre essa correspondência. Quando o EXEC () é chamado a uma expressão regular que tem a bandeira G global ou oConjunto de bandeira Y Sticky, ele consulta a propriedade LastIndex do regexpObjeto para determinar onde começar a procurar uma correspondência. (E se o yFlag está definido, também restringe a partida para começar nessa posição.) Para um objeto regexp recém -criado, LastIndex é 0 e a pesquisa começaNo início da string. Mas cada vez executiva () encontra com sucesso umcorresponder, ele atualiza a propriedade LastIndex para o índice doPersonagem imediatamente após o texto correspondente. Se Exec () não encontrar umcorresponde, ele redefine o LastIndex para 0. Este comportamento especial permite que você ligue para o Exec () repetidamente para percorrer todo o regularA expressão corresponde a uma string. (Embora, como descrevemos, emES2020 e, posterior, o método matchall () da string é uma maneira mais fácilPara fazer uma pancada em todas as partidas.) Por exemplo, o loop no seguinteO código será executado duas vezes:deixe padrão = /java /g;deixe texto = "javascript> java";deixe a partida;while ((match = Pattern.exec (text))! == null) {console.log ('correspondente \$ {corresponde [0]} em \$ {match.index}`);console.log (`Próxima pesquisa começa em\$ {Pattern.LastIndex} `);}A propriedade LastIndex e a reutilização regexp

Como você já viu, a API de expressão regular de JavaScript é complicada. O uso da propriedade LastIndex com as bandeiras G e Y é uma parte particularmente estranha desta API. Quando você usa essas bandeiras, você precisa ser particularmente cuidadoso ao ligar para a correspondência (), Exec () ou teste () métodos porque o comportamento desses métodos depende do LastIndex e do valor de O LastIndex depende do que você fez anteriormente com o objeto regexp. Isso facilita para escrever código de buggy. Suponha, por exemplo, que queríamos encontrar o índice de todas as tags <p> em uma string de texto HTML. Podemos escrever código como este:

```
deixe corresponder, posições = [];
while ((match = /<p>/g.exec(html)) != null) { // possível loop
    infinitoposições.push (match.index);
}
```

Este código não faz o que queremos. Se a sequência html contiver pelo menos uma tag <p>, loop para sempre. O problema é que usamos um literal regexp na condição de loop while. Para cada iteração do loop, estamos criando um novo objeto regexp com o LastIndex definido como 0, então exec () sempre Começa no início da corda e, se houver uma correspondência, ela continuará correspondendo repetidamente. O Solução, é claro, é definir o regexp uma vez e salvá-lo em uma variável para que estejamos usando o mesmo objeto regexp para cada iteração do loop. Por outro lado, às vezes reutilizar um objeto regexp é a coisa errada a se fazer. Suponha, para Exemplo, que queremos percorrer todas as palavras em um dicionário para encontrar palavras que contêm pares de cartas duplas:

```
Deixe dicionário = ["Apple", "Book", "Coffee"];
Seja DoubleletterWords = [];
Seja Doubleletter = /( \W ) \1 /g;
para (Let Word of Dictionary) {
    if (Doubleletter.test (word))
        DoubleletterWords.push (Word);
}
DoubleletterWords // => ["Apple", "Coffee"];
```

"Book" está faltando! Como definimos a bandeira G no regexp, a propriedade LastIndex é alterada após o sucesso Matches, e o método test () (que é baseado no EXEC ()) começa a procurar uma correspondência na posição especificada por LastIndex. Depois de combinar o "PP" em "Apple", o Lastindex é 3 e, portanto, começamos Pesquisando a palavra "livro" na posição 3 e não veja o "OO" que ele contém. Poderíamos resolver esse problema removendo a bandeira G (que não é realmente necessária nesse exemplo), ou movendo o regexp literal para o corpo do loop, para que seja recriado em cada iteração ou redefinindo explicitamente o LastIndex para zero antes de cada chamada para testar (). A moral aqui é que o LastIndex faz com que o erro da API Regexp seja propenso. Portanto, tenha muito cuidado quando usando as bandeiras G ou Y e o loop. E no ES2020 e posterior, use o método String Matchall ()

Em vez de Exec () para evitar esse problema, pois o Matchall () não modifica o LastIndex.11,4 datas e horáriosA classe de data é a API da JavaScript para trabalhar com datas e horários.Crie um objeto de data com o construtor date ().Sem argumentos,Ele retorna um objeto de data que representa a data e a hora atuais:deixe agora = new Date ():// A hora atualSe você passar um argumento numérico, a data () interpreta o construtorEsse argumento como o número de milissegundos desde a época de 1970:Seja epoch = nova data (0);// meia -noite, 1º de janeiro de 1970, GMTSe você especificar dois ou mais argumentos inteiros, eles são interpretados comoO ano, mês, dia de mês, hora, minuto, segundo e milissegundosno seu fuso horário local, como no seguinte:Let Century = New Date (2100, // Ano 21000, // janeiro1, // 1º2, 3, 4, 5); // 02: 03: 04.005, localtempoUma peculiaridade da API da data é que o primeiro mês de um ano é o número 0,Mas o primeiro dia de um mês é o número 1. Se você omitir os campos de tempo, oDATE () Constructor Padrates todos para 0, definindo o tempo parameia-noite.Observe que, quando invocado com vários números, a data ()

construtor os interpreta usando qualquer fuso horário o localO computador está definido como.Se você deseja especificar uma data e hora no UTC(Tempo coordenado universal, também conhecido como GMT), então você pode usar oDate.utc ().Este método estático leva os mesmos argumentos que oDate () construtor, os interpreta no UTC e retorna umMilissegund Timestamp que você pode passar para o construtor DATE ():// meia -noite na Inglaterra, 1 de janeiro de 2100Let Century = New Date (Date.utc (2100, 0, 1));Se você imprimir uma data (com console.log (século), por exemplo), elePor padrão, será impresso no seu fuso horário local.Se você quiserexibir uma data no UTC, você deve convertê -lo explicitamente em uma string comtoutcString () ou ToisSotring ().Finalmente, se você passar uma string para o construtor date (), ele tentaráPara analisar essa string como especificação de data e hora.O construtor podedatas de parse especificadas nos formatos produzidos pelo tostring (),Métodos ToutcString () e ToisSoString ():Let Century = New Date ("2100-01-01T00: 00: 00Z");// um ISOData de formatoDepois de ter um objeto de data, vários métodos de obtenção e definição permitem que vocêconsultar e modificar o ano, mês, dia de mês, hora, minuto,segundo, e milissegundos campos da data.Cada um desses métodos temduas formas: uma que recebe ou conjunta usando a hora local e uma que recebe ouconjuntos usando o tempo UTC.Para obter ou definir o ano de um objeto de data, paraPor exemplo, você usaria o getutclyear (), getutclyear (),setutclyear () ou setutclyear ():

Seja d = new Date ()// Comece com odata atuald. sinerlyear (d.getlyear () + 1);// incremento o anoPara obter ou definir os outros campos de uma data, substitua "totalmente" noNome do método com "mês", "data", "horas", "minutos", "segundos",ou "milissegundos".Alguns dos métodos conjuntos de data permitem que você defina maisdo que um campo de cada vez.setlyear () eSetutclyear () também permite opcionalmente que você defina o mês eDia de mês também.E sethours () e setutchours ()Permita que você especifique os campos de atas, segundos e milissegundos emadição ao campo de horas.Observe que os métodos para consultar o dia do mês são GetDate ()e getutcdate ().As funções mais naturaisgetDay () e getutcday () retornam o dia da semana (0 para domingoaté 6 para sábado).O dia da semana é somente leitura, então não há umMétodo SetDay () correspondente.

11.4.1 TimestampsJavaScript representa datas internamente como números inteiros que especificam oNúmero de milissegundos desde (ou antes) meia -noite de 1º de janeiro de 1970,Hora da UTC.Inteiros tão grande quanto 8.640.000.000.000.000 são suportados,Portanto, o JavaScript não ficará sem milissegundos por mais de270.000 anos.Para qualquer objeto de data, o método gettime () retorna este internoValue e o método setTime () o define.Então você pode adicionar 30 segundosPara uma data com código como este, por exemplo:

D.setTime (D.GetTime () + 30000);Esses valores de milissegundos às vezes são chamados de timestamps, e às vezes útil para trabalhar com eles diretamente, e não com a dataobjetos.O método estático date.now () retorna a hora atual como umTimestamp e é útil quando você deseja medir quanto tempoO código leva para executar:Seja starttime = date.now ();reticulatesplines ()// Faça alguma operação demoradadeixe endtime = date.now ();Console.log (' reticulação spline levou \${EndTime - startTime} ms.');// Ttimestamps de alta resoluçãoOs registros de data e hora retornados por date.now () são medidos em milissegundos.Um milissegundo é na verdade relativamente longo para um computador e às vezes você pode medir o tempo decorrido com maiorprecisão.A função performance.now () permite isso: também retorna um milissegundo baseadoTimestamp, mas o valor de retorno não é um número inteiro, por isso inclui frações de um milissegundo.O valorRetornado por performance.now () não é um registro de data e hora absoluto como o valor DAT.now ().Em vezO processo do nó foi iniciado.O objeto de desempenho faz parte de uma API de desempenho maior que não é definida pelo ecmascriptpadrão, mas é implementado por navegadores da web e por nó.Para usar o objeto de desempenhoNo nó, você deve importá-lo com:const {performance} = requer ("perf_hooks");Permitir que o tempo de alta precisão na web possa permitir que sites sem escrúpulos visitem a impressão digital,entãoOs navegadores (principalmente o Firefox) podem reduzir a precisão do desempenho.now () por padrão.Como um webDesenvolvedor, você deve ser capaz de reativar o tempo de alta precisão de alguma forma (como definirPrivacy.ReduceTimerPrecision to False in Firefox).11.4.2 Data aritmética

Os objetos de data podem ser comparados com o padrão de JavaScript <, <=,> e> = operadores de comparação. E você pode subtrair um objeto de data de outro para determinar o número de milissegundos entre os dois dados. (Isso funciona porque a classe de data define um valueof ()Método que retorna um registro de data e hora.) Se você deseja adicionar ou subtrair um número especificado de segundos, minutos, ou horas a partir de uma data, geralmente é mais fácil simplesmente modificar o registro de data e horaComo demonstrado no exemplo anterior, quando adicionamos 30 segundos para uma data. Esta técnica se torna mais pesada se você quiser Adicione dias e não funciona por meses e anos desde que ele estêem um número variável de dias. Fazer datar a aritmética envolvendo dias, meses e anos, você pode usar o setDate (), SetMonth () e SetYear (). Aqui, por exemplo, é o código que adiciona três meses duas semanas até a data atual: Seja d = new Date (); D.setMonth (d.getMonth () + 3, d.getDate () + 14); Os métodos de configuração de data funcionam corretamente, mesmo quando transbordam. Quando adicionamos três meses ao mês atual, podemos acabar com um valor maior que 11 (que representa dezembro). O setMonth ()lida com isso incrementando o ano, conforme necessário. Da mesma forma, quando nósDefina o dia do mês como um valor maior que o número de dias emO mês, o mês é incrementado adequadamente.

11.4.3 Strings de data de formatação e análise

Se você estiver usando a classe de data para realmente acompanhar as datas e horários(em vez de apenas medir intervalos de tempo), é provável que você

Precisa exibir datas e horários para os usuários do seu código. A dataclasse define vários métodos diferentes para converter objetos de datapara cordas. Aqui estão alguns exemplos: Seja d = nova data (2020, 0, 1, 17, 10, 30); // 17:10:30 no novoDia do ano 2020d.ToString () // => "Qua Jan 01 2020 17:10:30 GMT-0800(Time padrão do Pacífico)" d.toutcString () // => "qui, 02 de janeiro de 2020 01:10:30 GMT" d.tolocaledatestring () // => "1/1/2020": 'en-us' Localed.tolocaletimestring () // => "17:10:30": 'en-us' LocaleD.ToisSoString () // => "2020-01-02T01: 10: 30.000Z" Esta é uma lista completa dos métodos de formatação da string da classe de data: ToString () Este método usa o fuso horário local, mas não formate a data e tempo de maneira consciente do local. toutcString () Este método usa o fuso horário da UTC, mas não formate a data de uma maneira consciente do local. ToisSoString () Este método imprime a data e a hora no último mês-Horário do dia: Minutos: segundos.ms Formato do padrão ISO-8601. A letra "t" separa a parte da data da saída da época parte da saída. O tempo é expresso na UTC, e isso é indicado com a letra "Z" como a última letra da saída. Tolocalestring () Este método usa o fuso horário local e um formato que éapropriado para o local do usuário.

`TodATestring ()`Este método forma apenas a parte da data da data e omite o tempo.Ele usa o fuso horário local e não faz localidade-formatação apropriada.`tolocaledatestring ()`Este método formata apenas a data.Ele usa o fuso horário local e um formato de data apropriado para localidade.`Totimestring ()`Este método formata apenas a hora e omite a data.Ele usa o fuso horário local, mas não formate o tempo de maneira consciente do local.`tolocaletimestring ()`Este método formata o tempo de maneira consciente do local e usa o fuso horário local.Nenhum desses métodos de data para cordas é ideal quando as datas de formatação e tempos a serem exibidos para usuários finais.Ver §11.7.2 para um mais geralTécnica de formatação para propósito e data de conhecimento e local.Finalmente, além desses métodos que convertem um objeto de data em umString, há também um método estático de data.`parse ()` que leva uma stringComo argumento, tentativas de analisá-lo como uma data e hora, e retorna umTimestamp representando essa data.`Date.parse ()` é capaz de analisar o mesmoas cordas que o construtor DAT () pode e é garantidoCapaz de analisar a saída de `ToisSotring ()`, `ToutcString ()`,e `tostring ()`.

Erro ao traduzir esta página.

O erro é capturado. Além da classe de erro, JavaScript define uma série de subclasses que ele usa para sinalizar tipos específicos de erros definidos por ECMAScript. Essas subclasses são avaliadas, RangeError, ReferenceError, SyntaxError, TypeError e UriError. Você pode usar essas classes de erro em seu próprio código, se parecerem apropriadas. Como classe de erro base, cada uma dessas subclasses tem um construtor que leva um argumento de mensagem única. As instâncias de cada uma dessas subclasses têm uma propriedade de nome cujo valor é o mesmo que o construtor nome. Você deve se sentir livre para definir suas próprias subclasses de erro que melhorem encapsular as condições de erro do seu próprio programa. Observe que você não está limitado ao nome e propriedades da mensagem. Se você criar uma classe, você pode definir novas propriedades para fornecer detalhes de erro. Se você estiver escrevendo um analisador, por exemplo, você pode achar útil definir uma classe de parseerror com propriedades de linha e coluna que especificam a localização exata da falha da análise. Ou se você estiver trabalhando com httpSolicitações, você pode querer definir uma classe httperror que tenha uma propriedade de status que contém o código de status HTTP (como 404 ou 500) da solicitação fracassada. Por exemplo:

```
class httperror extends Error {  
    constructor(status, statustext, url) {super(`$ {status} $ {statustext}: $ {url}`);this.status = status;this.statustext = statustext;}}  
const error = new httperror(404, 'Not Found', 'https://www.google.com');  
console.log(error.message); // "404 Not Found: https://www.google.com"
```

this.url = url; }
get name () { return "httperror"; }
Deixe erro = novo httperror (404, "não
encontrado", "http://example.com/");
erro.status // => 404
erro.message // => "404 não
encontrado:
http://example.com/"
erro.name // => "httperror"
11.6 Serialização e análise de JSON JSON Quando um
programa precisa salvar dados ou transmitir dados em uma conexão de rede com outro programa, deve converter
seu in-Estruturas de dados de memória em uma sequência de bytes ou caracteres do que pode ser salvo ou
transmitido e depois será analisado para restaurar o originais estruturas de dados de memória. Este processo de
conversão de estruturas de dados em fluxos de bytes ou caracteres são conhecidos como serialização (ou
marshaling) ou até decapagem). A maneira mais fácil de serializar dados em JavaScript usa uma
serialização formato conhecido como json. Este acrônimo significa ?Objeto JavaScript Notação? e, como o nome
indica, o formato usa o objeto JavaScript a sintaxe literal da matriz para converter estruturas de dados que
consistem em objetos e matrizes em cordas. JSON suporta números e cordas primitivas e também os valores
verdadeiros, falsos e nulos, bem como matrizes e objetos construídos a partir desses valores primitivos. JSON não
suporta outros tipos de JavaScript, como mapa, set, regexp, data ou matrizes digitadas. No entanto, provou ser um
formato de dados notavelmente versátil

e é de uso comum, mesmo com programas não baseados em Javascript. JavaScript suporta a serialização e a deserialização de JSON com os dois funções `json.stringify()` e `json.parse()`, que eram cobertos brevemente em §6.8. Dado um objeto ou matriz (aninhado arbitrariamente profundamente) que não contém valores não-serializáveis ?? como regexes, objetos ou matrizes digitadas, você pode serializar o objeto simplesmente passando para `json.stringify()`. Como o nome indica, o valor de retorno de Esta função é uma string. E dada uma string devolvida por `json.stringify()`, você pode recriar a estrutura de dados original Passando a string para `json.parse()`: Seja `o = {s: "", n: 0, a: [true, false, null]}`; Seja `s = json.Stringify(O)`; // `s == '{"s": "", "n": 0, "a": [Verdadeiro, falso, nulo]}` Se deixarmos de fora a parte em que os dados serializados são salvos em um arquivo ou enviado pela rede, podemos usar este par de funções como um pouco maneira ineficiente de criar uma cópia profunda de um objeto:// Faça uma cópia profunda de qualquer objeto ou matriz serializável função Deepcopy (`O`) {retornar `json.parse(json.stringify(o))`}; JSON é um subconjunto de JavaScript Quando os dados são serializados para o formato JSON, o resultado é o código -fonte JavaScript válido Para uma expressão que avalia uma cópia da estrutura de dados original. Se você prefixar uma string json com `var dados = e` passe o resultado para `eval()`, você vai obter uma cópia da estrutura de dados original atribuída aos dados da variável. Você

nunca deveria fazer isso, no entanto, porque é um enorme buraco de segurança - se um atacante poderia injetar código JavaScript arbitrário em um arquivo json, eles poderiam fazer o seu programa executar seu código. É mais rápido e seguro usar apenas `json.parse()` para decodificar dados formatados por JSON. Às vezes, o JSON é usado como um formato de arquivo de configuração legível pelo homem. Se você encontrar-se editando um arquivo JSON, observe que o formato JSON é um muito rigoroso subconjunto de JavaScript. Os comentários não são permitidos e os nomes de propriedades devem ser incluídos em cotações duplas, mesmo quando o JavaScript não exigiria isso. Normalmente, você passa apenas um único argumento para `json.stringify()` e `json.parse()`. Ambas as funções aceitam um segundo opcional argumento que nos permite estender o formato JSON, e estes são descritos a seguir. `Json.stringify()` também leva um terceiro opcional argumento que discutiremos primeiro. Se você gostaria de seu json-string formatada para ser legível por humanos (se estiver sendo usada como um arquivo de configuração, por exemplo), então você deve passar nulo como o segundo argumento e passe um número ou string como o terceiro argumento. Este terceiro argumento diz a `json.stringify()` que deve formatar os dados em várias linhas recuadas. Se o terceiro argumento for um número, em seguida, ele usará esse número de espaços para cada nível de indentação. Se o terceiro argumento é uma série de espaço em branco (como '\t'), ele usará isso para cada nível de indent. Seja `o = {s: "test", n: 0}; Json.Stringify(o, null, 2) // => '{\n "s": "teste",\n "n": 0\n}'`. `Json.parse()` ignora o espaço em branco, então passando um terceiro argumento para `Json.stringify()` não tem impacto em nossa capacidade de converter o

String de volta em uma estrutura de dados. 11.6.1 Customizações JSON Se `json.Stringify()` for solicitado a serializar um valor que não é nativamente apoiado pelo formato JSON, parece ver se esse valor tem um método `toJSON()` e, se assim for, chama esse método e depois registra o valor de retorno no lugar do valor original. Objetos de data implementam `toJSON()`: ele retorna a mesma string que o método `ToISOString()` faz. Isso significa que se você serializar um objeto que inclui uma data, a data será automaticamente convertida para uma string para você. Quando você analisa a corda serializada, o recriado a estrutura de dados não será exatamente a mesma que a que você começou porque ele terá uma string em vez do objeto original que teve uma data. Se você precisar recriar objetos de data (ou modificar o objeto analisado em qualquer outra maneira), você pode passar uma função de "reviver" como a segunda argumento para `json.parse()`. Se especificado, esta função "Reviver" é invocada uma vez para cada valor primitivo (mas não os objetos ou matrizes que contêm esses valores primitivos) analisado na sequência de entrada. A função é invocada com dois argumentos. O primeiro é um nome de propriedade - um nome de propriedade do objeto ou um índice de matriz convertido em uma string. O segundo argumento é o valor primitivo dessa propriedade de objeto ou elemento da matriz. Além disso, a função é invocada como um método do objeto ou matriz que contém o valor primitivo, para que você possa se referir a isso contendo o objeto com essa palavra-chave. O valor de retorno da função Reviver se torna o novo valor da propriedade nomeada. Se retornar seu segundo argumento, a propriedade irá

permanecer inalterado. Se retornar indefinido, então a propriedade nomeada será excluída do objeto ou matriz antes de json.parse () retornar ao usuário. Como exemplo, aqui está uma chamada para json.parse () que usa um reviver função para filtrar algumas propriedades e recriar objetos de data: Deixe dados = json.parse (texto, função (chave, valor) { // Remova todos os valores cujo nome da propriedade começa com um sublinhado if (key [0] === "_") retorna indefinido; // Se o valor for uma string no formato ISO 8601 Data converte -o em uma data. if (typeof value === "string" && /\d\ \d\ \d\ \d- \d\ \d\ \dt\ \d\ \d: \d\ \d: \d\ \d. \d\ \d\ \dz/. test (value)) { retornar nova data (valor); } // caso contrário, retorne o valor inalterado valor de retorno; }); Além do uso de Tojson () descrito anteriormente, Json.stringify () também permite que sua saída seja personalizada por passando uma matriz ou uma função como o segundo argumento opcional. Se uma variedade de cordas (ou números - eles são convertidos em cordas) é passado como o segundo argumento, eles são usados ?? como nomes de Propriedades do objeto (ou elementos da matriz). Qualquer propriedade cujo nome não seja na matriz, será omitida da Stringification. Além disso, a string retornada incluirá propriedades na mesma ordem em que elas aparecem na matriz (que pode ser muito útil ao escrever testes).

Se você aprovar uma função, é uma função de substituição - efetivamente o inverso da função `reviver` opcional, você pode passar para `json.parse()`. Se `EsSpecificado`, a função `Replacer` é invocada para que cada valor seja tratado. O primeiro argumento para a função `Replacer` é o nome do objeto ou propriedade ou índice de matriz do valor dentro desse objeto, e o segundo argumento é o próprio valor. A função substituta é invocada com um método do objeto ou matriz que contém o valor a ser rigoroso. O valor de retorno da função `Replacer` é rigoroso no lugar do valor original. Se o substituto retornar indefinido ou não retornará nada em `Todos`, então esse valor (e seu elemento de matriz ou propriedade de objeto) é omitido da sequência. // Especifique quais campos serializarem e que ordem serializará -los em `Deixe o texto = json.stringify(endereço, ["cidade", "estado", "país"]);` // Especifique uma função de substituição que omite o valor `regExp` `propriiedades`. Seja `json = json.stringify(o, (k, v) => v instanceof RegExp ? undefined : v);` As duas chamadas `JSON.Stringify()` aqui usam o segundo argumento em uma maneira benigna, produzindo saída serializada que pode ser deserializada sem exigir uma função de `reviver` especial. Em geral, porém, se você definir um método `toJSON()` para um tipo, ou se você usar um substituto de função que na verdade substitui valores não-serializáveis ?? por serializável outro, então você normalmente precisará usar uma função de `reviver` personalizada com `Json.parse()` para recuperar sua estrutura de dados original. Se você fizer isso, você deve entender que está definindo um formato de dados personalizado e sacrificando a portabilidade e a compatibilidade com um grande ecossistema de

Erro ao traduzir esta página.

chamado de "números árabes" 0 a 9 são usados ??em muitos idiomas, estenão é universal, e os usuários em alguns países esperam ver númerosescrito usando os dígitos de seus próprios scripts.A classe `intl.numberFormat` define um método `format ()` que levaTodas essas possibilidades de formatação em consideração.O construtor tomadois argumentos.O primeiro argumento especifica o local que o númerodeve ser formatado e o segundo é um objeto que especifica maisDetalhes sobre como o número deve ser formatado.Se o primeiro argumentoé omitido ou indefinido, então o local do sistema (que assumimospara ser o local preferido do usuário) será usado.Se o primeiro argumento for umstring, especifica um local desejado, como "en-us" (inglês como usadoNos Estados Unidos), "FR" (francês) ou "Zh-Hans-Cn" (chinês,usando o sistema de escrita Han simplificado, na China).O primeiro argumentotambém pode ser uma variedade de cordas de localidade e, neste caso,`Intl.NumberFormat` escolherá o mais específico que está bemsuportado.O segundo argumento para o construtor `intl.numberFormat` ()especificado, deve ser um objeto que defina um ou mais dos seguintespropriedades:estiloEspecifica o tipo de formatação numérica necessária.OO padrão é "decimal".Especifique "porcentagem" para formatar um númerocomo uma porcentagem ou especificar "moeda" para especificar um número como umquantidade de dinheiro.moeda

Se o estilo for "moeda", esta propriedade é necessária para especificarO código de moeda ISO de três letras (como "USD" para dólares americanosou "GBP" para libras britânicas) da moeda desejada.MoedaDisplaySe o estilo é "moeda", esta propriedade especifica como oA moeda é exibida.O valor padrão "símbolo" usa umSímbolo da moeda se a moeda tiver uma.O valor "código" usaO código ISO de três letras e o valor "nome" soletram onome da moeda em forma longa.useGroupingDefina esta propriedade como false se você não quiser que os números tenhammilhares de separadores (ou seus equivalentes apropriados para localidade).MinimintegerDigitsO número mínimo de dígitos a serem usados ??para exibir a parte inteira deo número.Se o número tiver menos dígitos do que isso, seráacolchoado à esquerda com zeros.O valor padrão é 1, mas você podeUse valores até 21.MINIMUMFRACTIONDDIGITS, MAXIMUMFRACTIONDDIGITSEssas duas propriedades controlam a formatação da parte fracionária deo número.Se um número tiver menos dígitos fracionários do que oMínimo, ele será acolchoado com zeros à direita.Se tiver maisdo que o máximo, então a parte fracionária será arredondada.JurídicoOs valores para ambas as propriedades estão entre 0 e 20. O padrãoo mínimo é 0 e o máximo padrão é 3, exceto quandoformatando quantidades monetárias, quando o comprimento do fracionárioA parte varia dependendo da moeda especificada.MinimumsignificantDigits,MaximumsInSignificAntDDigits

Essas propriedades controlam o número de dígitos significativos usados ??quandoformatando um número, tornando -os adequados ao formatarDados científicos, por exemplo.Se especificado, essas propriedades substituem as propriedades inteiras e de dígitos fracionários listadas anteriormente.JurídicoOs valores estão entre 1 e 21.Depois de criar um objeto intl.NumberFormat com o desejadoLocal e opções, você o usa, passando um número para seu formato ()Método, que retorna uma string adequadamente formatada.Por exemplo:Let Euros = intl.NumberFormat ("es", {style: "moeda",moeda: "Eur"});EUROS.FORMAT (10) // => "10,00 ?": dez euros, espanholformataçãoDeixe Pounds = intl.NumberFormat ("EN", {style: "Moutrency",moeda: "GBP"});libras.format (1000) // => "£ 1.000,00": mil libras,Formatação em inglêsUma característica útil do intl.NumberFormat (e as outras classes INTL)Além disso) é que seu método formato () está ligado ao número de formaçãooobjeto ao qual pertence.Então, em vez de definir uma variável que se referepara o objeto de formatação e depois invocar o método format ()isso, você pode apenas atribuir o método format () a uma variável e usá -loComo se fosse uma função independente, como neste exemplo:deixe dados = [0,05, 0,75, 1];Seja formatData = intl.numberFormat (indefinido, {Estilo: "porcentagem",MinimumFractionDigits: 1,MaximumFractionDigits: 1}).formatar;data.map (formatData) // => ["5,0%", "75,0%", "100,0%"]: em IN

Localidade de EN-USAlguns idiomas, como o árabe, usam seu próprio script para decimaldígitos:Seja árabe = `intl.NumberFormat ("AR", {useGrouping:false}).formato;`árabe (1234567890) // => "?????????"Outros idiomas, como o hindi, usam um script que tem seu próprio conjunto dedígitos, mas tendem a usar os dígitos ASCII 0?9 por padrão.Se você quisersubstituir o script padrão usado para dígitos, adicione -u-nu- ao locale siga -o com um nome de script abreviado.Você pode formatar númeroscom agrupamento de estilo indiano e dígitos de devanagari como este, paraexemplo:Seja hindi = `intl.NumberFormat ("Hi-in-u-nu-deva").Formato;`Hindi (1234567890) // => "?,??,??,??,??"-u- em um local especifica que o que vem a seguir é uma extensão unicode.Nu é o nome de extensão do sistema de numeração, e Deva é curtoPara Devanagari.O padrão da API INTL define nomes para váriosoutros sistemas de numeração, principalmente para os idiomas indicados do sul eSudeste Asiático.11.7.2 Datas e horários de formataçãoA classe Intl.DateTimeFormat é muito parecida com o `intl.numberFormat`aula.O construtor `intl.dateTeMormat ()` leva o mesmoDois argumentos que o `intl.numberFormat ()` faz: um local ou matrizde locais e um objeto de opções de formatação.E a maneira como você usa um

Erro ao traduzir esta página.

mêsUse "numérico" para um número possivelmente curto como "1", ou "2-dígito "para uma representação numérica que sempre tem dois dígitos,como "01".Use "Long" para um nome completo como "janeiro", "curto"para um nome abreviado como "Jan" e "estreito" para um altamenteNome abreviado como "J" que não é garantido para ser único.diaUse "numérico" para um número de um ou dois dígitos ou "2 dígitos"Para um número de dois dígitos para o dia do mês.Dia da semanaUse "Long" para um nome completo como "segunda -feira", "curto" para um nome abreviado como "seg" e "estreito" para um altamenteNome abreviado como "M" que não é garantido para ser único.eraEsta propriedade especifica se uma data deve ser formatada com umEra, como CE ou BCE.Isso pode ser útil se você estiver formatandodatas de muito tempo atrás ou se você estiver usando um calendário japonês.Os valores legais são "longos", "curtos" e "estreitos".hora, minuto, segundoEssas propriedades especificam como você gostaria de ser exibido.Uitar"numérico" para um campo de um ou dois dígitos ou "2 dígitos" para forçarnúmeros de um dígito a serem acolchoados à esquerda com 0.fuso horárioEsta propriedade especifica o fuso horário desejado para o qual a datadeve ser formatado.Se omitido, o fuso horário local é usado.As implementações sempre reconhecem "UTC" e também podem reconhecerNomes de fuso horário da Autoridade de Números da Internet atribuídos (IANA),

como "America/Los_Angeles".`TimeZoneName` Esta propriedade especifica como o fuso horário deve ser exibido em um data ou hora formatada. Use "Long" para um tempo totalmente soletrado. `Nome` da zona e "curto" para um fuso horário abreviado ou numérico. `hora12` Esta propriedade booleana especifica se deve ou não usar 12 horas. O padrão é dependente do local, mas você pode substituí-lo com `issopropriedade.Hourcycle`. Esta propriedade permite especificar se a meia-noite está escrita como 0 horas, 12 horas ou 24 horas. O padrão depende do local, mas você pode substituir o padrão por esta propriedade. Observe que `Hour12` tem precedência sobre esta propriedade. Use o valor "H11" para especificar que a meia-noite é 0 e a hora antes da meia-noite é 23:00. Use "H12" para especificar que a meia-noite é 12. Use "H23" para especificar que a meia-noite é 0 e a hora antes da meia-noite é 23. e use "H24" para especificar que a meia-noite é 24. Aqui estão alguns exemplos: Seja `d` = nova data ("2020-01-02T13:14:15Z"); // 2 de janeiro, 2020, 13:14:15 UTC // Sem opções, obtemos um formato de data numérica básica: `Intl.dateFormat("en-us")`. Formato (`d`) // => "1/2/2020" `Intl.dateFormat("fr-fr")`. Formato (`d`) // => "02/01/2020" // soletrou o dia da semana e o mês. Let `Opts = {Weekday: "Long", Mês: "Long", Ano: "Numeric", Dia: "numérico"};` `Intl.dateFormat("en-us", opts)` .Format (`d`) // => "Quinta-feira, quinta-feira,

2 de janeiro de 2020 "Intl.DateTimeFormat ("es-es", opts) .Format (d) // => "Jueves, 2de Enero de 2020 "// O tempo em Nova York, para um canadense de língua francesaopts = {Hour: "numeric", minuto: "2 dígitos", fuso horário:"America/new_york"};Intl.DateTimeFormat ("FR-CA", OPTS) .FORMAT (D) // => "8 H 14"Intl.dateTeMformat pode exibir datas usando calendários além doCalendário Juliano Padrão com base na era cristã.Embora algunsOs locais podem usar um calendário não-cristão por padrão, você sempre podeEspecifique explicitamente o calendário a ser usado adicionando -u-ca- ao locale seguindo isso com o nome do calendário.Calendário possívelOs nomes incluem "budista", "chinês", "copta", "etiópico", "gregory", "Hebraico", "Indiana", "Islâmico", "Iso8601", "Japonês" e "Persa".Continuando o exemplo anterior, podemos determinar o ano emVários calendários não-cristãos:Seja opts = {ano: "numérico", ERA: "Short"};Intl.dateFormat ("en", opts) .Format (d) //=> "2020 AD"Intl.dateFormat ("en-u-ca-isto8601", opts) .Format (d) //=> "2020 AD"Intl.dateFormat ("en-u-ca-hebrew", opts) .Format (d) //=> "5780 AM"Intl.dateFormat ("en-u-ca-buddhist", opts) .Format (d) //=> "2563 ser"Intl.DateTimeFormat ("en-u-ca-islâmico", opts) .Format (d) //=> "1441 ah"Intl.DateTimeFormat ("En-u-Ca-Persian", OPTS) .Format (d) //=> "1398 AP"Intl.DateTimeFormat ("en-u-ca-indiano", opts) .Format (d) //=> "1941 Saka"Intl.DateTimeFormat ("en-u-ca-chinese", opts) .Format (d) //=> "36 78"Intl.dateFormat ("en-u-ca-japonesa", opts) .Format (d) //=> "2 reiwa"

11.7.3 Comparando strings O problema de classificar seqüências em ordem alfabética (ou mais um pouco? Ordem geral de agrupamento? para scripts não alfábéticos) é mais desafiador do que os falantes de inglês costumam perceber. O inglês usa um alfabeto relativamente pequeno sem letras acentuadas, e temos o benefício de uma codificação de caracteres (ASCII, uma vez incorporada em Unicode) cujos valores numéricos correspondem perfeitamente à nossa string padrão. Ordem de classificação. As coisas não são tão simples em outros idiomas. Espanhol, por exemplo trata ñ como uma letra distinta que vem depois de n e antes de o. Lituano em alfabetos y antes de J, e galês trata dígrafos como che DD como letras únicas com CH chegando após C e DD classificando depois D. Se você quiser exibir strings para um usuário em uma ordem que eles encontrarião natural, não é suficiente usar o método `sort()` em uma variedade de cordas. Mas se você criar um objeto `Intl.Collator`, poderá passar no `compare()` método desse objeto ao método `sort()` para executar a local-classificação apropriada das cordas. Os objetos `Intl.Collator` podem ser configurados para que o método `compare()` execute o caso sensível comparações ou mesmo comparações que apenas consideram a letra base e ignoram detalhes e outros diacríticos. Como `intl.numberFormat()` e `intl.dateFormat()`. O construtor `Intl.Collator()` leva dois argumentos. O primeiro especifica um local ou uma variedade de locais, e o segundo é um opcional Objeto cujas propriedades especificam exatamente que tipo de comparação de string deve ser feito. As propriedades suportadas são estes:

usoEsta propriedade especifica como o objeto Collator deve ser usado.OO valor padrão é "classificar", mas você também pode especificar "pesquisa".A idéia é que, ao classificar seqüências, você normalmente quer um colisor isso diferencia o maior número possível de cordas para produzir um confiável pedindo.Mas ao comparar duas cordas, alguns locais podem querer uma comparação menos rigorosa que ignora sotaques, por exemplo.sensibilidadeEsta propriedade especifica se o colatador é sensível à letraCase e sotaques ao comparar seqüências.O valor "base" causa comparações que ignoram casos e sotaques, considerando apenas a letra base para cada personagem.(Observe, no entanto, que alguns idiomas consideram certos personagens acentuados como base distintas.) "Accent" considera sotaques em comparações, mas ignora caso."Caso" considera o caso e ignora sotaques.E"Variante" realiza comparações estritas que consideram ambos os casos e sotaques.O valor padrão para esta propriedade é "variante"Quando o uso é "classificar".Se o uso for "pesquisa", então a sensibilidade padrão depende do local.ignorepuncçãoDefina esta propriedade como True para ignorar espaços e pontuação quandoComparando strings.Com esta propriedade definida como true, as cordas ?qualquerUm "e" qualquer pessoa ", por exemplo, serão considerados iguais.numéricoDefina esta propriedade como true se as cordas que você estiver comparando são números inteiros ou contêm números inteiros e você deseja que eles sejam classificados em ordem numérica em vez de ordem alfabética.Com este conjunto de opções,A string ?versão 9? será classificada antes da ?versão 10?, para exemplo.

Casefirst Esta propriedade especifica qual caso de carta deve vir primeiro. Se você especificar "Upper", então "A" classificará antes de "A". E se você especificar "Lower", então "A" será classificado antes de "A". Em ambos os casos, observe que as variantes superiores e minúsculas da mesma letra irão estar próximo um do outro em ordem de classificação, que é diferente da ordenação lexicográfica (o comportamento padrão da matriz ()método) em que todas as letras maiúsculas ASCII vêm antes de todas as LETRAS PORTUGUESAS. O padrão para esta propriedade é localidade-dependente e implementações podem ignorar esta propriedade e não permitir que você substitua o pedido de classificação do caso. Depois de criar um objeto `Intl.Collator` para a localidade desejada e opções, você pode usar o método `compare()` para comparar duas strings. Este método retorna um número. Se o valor retornado for menor que zero, então a primeira string vem antes da segunda sequência. Se for maior que zero, então a primeira string vem depois da segunda sequência. Se se `Compare()` retorna zero, então as duas cordas são iguais até esta `Collator` está preocupado. Este método `compare()` que leva duas cordas e retorna um número. Menos que, igual a ou maior que zero é exatamente o que a matriz `Método Sort()` espera por seu argumento opcional. Além disso, `Intl.Collator` automaticamente o método `compare()` à sua instância, então você pode passar diretamente para `classificar()` sem ter que escrever um invólucro função e invocá-la através do objeto `Collator`. Aqui estão alguns exemplos:// Um exemplo básico para classificar na localidade do usuário.// nunca classifique se as cordas legais sem passar algo assim:

```
const collator = new Intl.Collator().Compare(["A", "Z", "A", "Z"]).Sort(Collator) // => ["A", "A", "Z", "Z"]// Os nomes de arquivos geralmente incluem números, então devemos classificá -los especialmente const filenameOrder = new Intl.Collator (indefinido, {numeric:true}).Compare(["Page10", "Page9"]).Sort (FileNameOrder) // => ["Page9", "Page10"]// Encontre todas const Fuzzymatcher = new Intl.Collator (indefinido, {Sensibilidade: "base", Ignorepunctuation: Verdadeiro}).comparar; Deixe strings = ["comida", "tolo", "barra føø` "]; strings.findIndex (s => fuzzymatcher (s, "foobar") === 0) //=> 2 Alguns locais têm mais de uma ordem de agrupamento possível. Na Alemanha, Por exemplo, os livros telefônicos usam uma ordem de classificação um pouco mais fonética do que Dicionários fazem. Na Espanha, antes de 1994, ?CH? e ?LL? foram tratados como cartas separadas, para que o país agora tenha uma ordem de classificação moderna e um Ordem de classificação tradicional. E na China, a ordem de agrupamento pode ser baseada em codificações de caracteres, o radical base e os traços de cada personagem, ou na romanização de Pinyin de personagens. Essas variantes de agrupamento não podem ser selecionado através do argumento das opções do Intl.Collator, mas elas podem ser selecionado adicionando -u-co- à string de localidade e adicionando o nome da variante desejada. Use "de-de-u-co-phonebk" para Livro telefônico pedidos na Alemanha, por exemplo, e "zh-tw-u-co-Pinyin" para pedidos de Pinyin em Taiwan. // Antes de 1994, CH e LL foram tratados como letras separadas em Espanha const
```

```
const tradicionalpanish = intl.collator ("es-es-u-co-troc "). Compare;Deixe Palabras = ["Luz", "Llama", "Como",  
"Chico"];Palabras.sort (ModernSpanish) // => ["Chico", "Como", "Llama", "Luz"]palabras.sort (tradicionalpanish) // =>  
["como", "chico", "Luz", "Llama"]11.8 A API do consoleVocê viu a função console.log () usada ao longo dissoLivro:  
Em navegadores da web, ele imprime uma string na guia "Console" doPainel de ferramentas de desenvolvedor do  
navegador, que pode ser muito útil quandodepuração.No nó, console.log () é uma saída de uso geralfunção e  
imprime seus argumentos para o fluxo de stdout do processo, ondeNormalmente, aparece para o usuário em uma  
janela de terminal como saída do programa.A API do console define uma série de funções úteis, além  
deconsole.log ().A API não faz parte de nenhum padrão ECMAScript,mas é apoiado pelos navegadores e por nó e  
tem sido formalmenteEscrito e padronizado em https://console.spec.whatwg.org.A API do console define as  
seguintes funções:console.log ()Esta é a mais conhecida das funções do console.Ele converte seuArgumentos  
para as cordas e produzem para o console.Incluiespacos entre os argumentos e inicia uma nova linha após a  
saídatodos os argumentos.console.debug (), console.info (), console.warn (),console.error ()
```

Essas funções são quase idênticas ao console.log (). Em Node, Console.error () envia sua saída para o Stderr Stream em vez do fluxo de stdout, mas as outras funções são aliases de console.log (). Nos navegadores, as mensagens de saída geradas por cada uma dessas funções pode ser prefixada por um ícone que indica seu nível ou gravidade, e o console do desenvolvedor também pode permitir desenvolvedores para filtrar mensagens de console por nível.console.assert () Se o primeiro argumento é verdade (isto é, se a afirmação passar), então isso não faz nada. Mas se o primeiro argumento for falso ou Outro valor falsamente, então os argumentos restantes são impressos como se eles foram passados ?? para console.error () com uma ?afirmaçãofalhou ?prefixo. Observe que, diferentemente das funções típicas assert (), console.assert () não joga uma exceção quando uma afirmação falha.console.clear () Esta função limpa o console quando isso é possível. Isso funciona nos navegadores e no nó quando o nó está exibindo sua saída para um terminal. Se a saída do nó tiver sido redirecionada para um arquivo ou um tubo, No entanto, chamar essa função não tem efeito.console.table () Esta função é um recurso notavelmente poderoso, mas pouco conhecido para produzindo saída tabular e é particularmente útil no código que precisam produzir saída que resume os dados.console.table () tenta exibir seu argumento no formulário tabular (embora, se não puder fazer isso, exiba -o usando regular console.log () formatação). Isso funciona melhor quando o argumento é uma variedade relativamente curta de objetos, e todos os objetos na matriz, têm o mesmo conjunto de propriedades (relativamente pequenas). Em Este caso, cada objeto na matriz é formatado como uma fileira da tabela,

e cada propriedade é uma coluna da tabela.Você também pode passar umMatriz de nomes de propriedades como um segundo argumento opcional para especificar o conjunto desejado de colunas.Se você passar por um objeto em vez de uma matriz de objetos, então a saída será uma tabela com uma coluna para Nomes de propriedades e uma coluna para valores de propriedade.Ou, se aqueles Os valores de propriedade são os próprios objetos, seus nomes de propriedades vão ser colunas na tabela.console.Trace () Esta função registra seus argumentos como console.log () faz e,Além disso, segue sua saída com um rastreamento de pilha.No nó, o A saída vai para Stderr em vez de stdout.console.count () Esta função leva um argumento de string e registra essa string, seguida Pelo número de vezes que foi chamado com essa string.Isso podeSeja útil ao depurar um manipulador de eventos, por exemplo, se você precisa acompanhar quantas vezes o manipulador de eventos tem sidoprovocado.console.countreset () Esta função leva um argumento de string e redefine o contador para issocorda.console.Group () Esta função imprime seus argumentos para o console como se tivessem sido passados para console.log (), então define o estado interno do console para que todas as mensagens subsequentes do console (até o próximo console.groupEnd () chamada) serão recuadas em relação a mensagem que acabou de imprimir.Isso permite um grupo de relacionados mensagens a serem agrupadas visualmente com o recuo.Em navegadores da web,O console do desenvolvedor normalmente permite que mensagens agrupadas sejam fechadas em colapso e expandido como um grupo.Os argumentos para

`console.group()` são normalmente usados ??para fornecer um explicativo nome para o grupo.
`console.GroupCollapSed()` Esta função funciona como `console.group()`, exceto que na webNavegadores, o grupo será "colapo" por padrão e oAs mensagens que ele contém serão ocultas, a menos que o usuário clique para expandir o grupo.No nó, esta função é um sinônimo de `console.Group()`.
`console.Grupend()` Esta função não leva argumentos.Não produz saída própria mas termina o recuo e o agrupamento causados ??pelo mais recente ligue para `console.group()` ou `console.GroupCollapsed()`.
`console.time()` Esta função requer um único argumento de string, faz uma nota de tempo foi chamado com essa string e não produz saída.
`console.timelog()` Esta função leva uma string como seu primeiro argumento.Se essa string tivesse sido passado anteriormente para `console.time()`, então ele imprime a string seguida pelo tempo decorrido desde o `console.time()` chamar.Se houver algum argumento adicional para `console.timelog()`, eles são impressos como se tivessem sido passados para `console.log()`.
`console.TimeEnd()` Esta função leva um único argumento de string.Se esse argumento tivesse sido passado anteriormente para `console.time()`, então ele imprime o argumento e o tempo decorrido.Depois de ligar `console.TimeEnd()`, não é mais legal ligar

console.timelog () sem primeiro chamar console.time ()de novo.11.8.1 Saída formatada com consoleFunções de console que imprimem seus argumentos como console.log ()tenha um recurso pouco conhecido: se o primeiro argumento é uma string que inclui%s, %i, %d, %f, %O ou %C, então este primeiro argumento é tratado comoString de formato e os valores dos argumentos subsequentes são substituídosna corda no lugar das seqüências de dois caracteres.Os significados das seqüências são os seguintes:%sO argumento é convertido em uma string.%i e %dO argumento é convertido em um número e depois truncado para umInteiro.%fO argumento é convertido em um número%O e %O.O argumento é tratado como um objeto, e nomes de propriedades eOs valores são exibidos.(Nos navegadores da web, essa tela é normalmenteinterativo, e os usuários podem expandir e colapsar propriedades para exploraruma estrutura de dados aninhada.) %O e %o Ambos os detalhes do objeto exibem.OA variante da maçaneta usa um formato de saída dependente da implementaçãoIsso é considerado mais útil para desenvolvedores de software.6

%cNos navegadores da web, o argumento é interpretado como uma série de CSSestilos e usado para estilizar qualquer texto a seguir (até o próximo %cSequência ou o final da string).No nó, a sequência %C e seuO argumento correspondente é simplesmente ignorado.Observe que geralmente não é necessário usar uma string de formato com oFunções de console: geralmente é fácil obter uma produção adequada por simplesmentepassando um ou mais valores (incluindo objetos) para a função epermitindo que a implementação os exiba de uma maneira útil.Como um exemplo, observe que, se você passar um objeto de erro para console.log (), eleé impresso automaticamente junto com seu rastreamento de pilha.11.9 URL APIsComo JavaScript é tão comumente usado em navegadores da web e webServidores, é comum o código JavaScript precisar manipular URLs.A classe URL analisa URLs e também permite modificação (adicionandoParâmetros de pesquisa ou caminhos de alteração, por exemplo) dos URLs existentes.Isto também lida adequadamente o tópico complicado de escapar e descontagemos vários componentes de um URL.A classe URL não faz parte de nenhum padrão ECMAScript, mas funciona emNó e todos os navegadores da Internet que não sejam o Internet Explorer.Isso é padronizado em <https://url.spec.whatwg.org>.Crie um objeto URL com o construtor url (), passando um absolutoString de URL como o argumento.Ou passar um URL relativo como o primeiroargumento e o URL absoluto de que é relativo como o segundo

argumento.Depois de criar o objeto URL, seus váriosPropriedades permitem que você consulte versões unespadas das várias partesdo URL:Vamos url = new url ("https://example.com:8000/path/name?q = termo#fragmento ");url.href // => "https://example.com:8000/path/name?q = termo#fragmento "url.origin // => "https://example.com:8000"url.protocol // => "https:"url.host // => "exemplo.com:8000"url.hostname // => "exemplo.com"url.port // => "8000"url.pathname // => "/path/name"url.search // => "? q = termo"url.hash // => "#fragment"Embora não seja comumente usado, os URLs podem incluir um nome de usuário ou umnome de usuário e senha, e a classe URL pode analisar esses URLcomponentes também:Vamos url = new url ("ftp: // admin: 1337!@ftp.example.com/");url.href // => "ftp: // admin: 1337!@ftp.example.com/"url.origin // => "ftp://ftp.example.com"Url.username // => "Admin"url.password // => "1337!"A propriedade de origem aqui é uma combinação simples do URLProtocolo e host (incluindo a porta, se for especificado).Como tal, é umpropriedade somente leitura.Mas cada uma das outras propriedades demonstradas emO exemplo anterior é lido/gravação: você pode definir qualquer uma dessas propriedadesPara definir a parte correspondente do URL:vamos url = new url ("https://example.com");// Comece com nosso

servidorurl.pathname = "API/pesquisa"; // Adicione um caminho para um endpoint da API url.search = "q = teste"; // Adicione uma consulta para um parâmetro url.toString () // => "https://example.com/api/search?q=test" Uma das características importantes da classe URL é que ela adiciona corretamente pontuação e escapa de caracteres especiais em URLs quando isso é necessário: vamos url = new URL ("https://example.com"); url.pathname = "Path with Spaces"; url.search = "q = foo#bar"; url.pathname // => "/caminho%20with%20spaces" url.search // => "? q = foo%23bar" url.href // => "https://example.com/path%20with%20spaces?q=foo%23bar" A propriedade HREF nesses exemplos é especial: ler Href é equivalente a chamar toString (): ele remonta a todas as partes do URL na forma de corda canônica do URL. E definir href para um novo String reencontra o analisador de URL na nova string como se você tivesse ligado o construtor URL () novamente. Nos exemplos anteriores, estamos usando a propriedade de pesquisa para consultar toda a parte de consulta de um URL, que consiste no Personagens de um ponto de interrogação até o final do URL ou para o primeiro caractere hash. Às vezes, é suficiente apenas tratar isso como um único Propriedade da URL. Freqüentemente, no entanto, as solicitações HTTP codificam os valores devários campos de forma ou vários parâmetros da API na parte de consulta de um URL usando o aplicativo/x-www-form-urlencoded formatar. Neste formato, a parte de consulta do URL é um ponto de interrogação

seguido por um ou mais pares de nome/valor, que são separados de uns aos outros por amperands.O mesmo nome pode parecer mais do queUma vez, resultando em um parâmetro de pesquisa nomeado com mais de um valor.Se você deseja codificar esses tipos de nomes/valores em pares na consultaparte de um URL, então a propriedade SearchParams será maisútil que a propriedade de pesquisa.A propriedade de pesquisa é umaLeia/escreva string que permite obter e defina toda a parte de consulta doUrl.A propriedade SearchParams é uma referência somente leitura a umObjeto URLSearchParams, que tem uma API para obter, configurar,Adicionando, excluindo e classificando os parâmetros codificados na consultaparte do URL:vamos url = new url ("https://example.com/search");url.search // => "": sem consulta aindaurl.searchparams.append ("q", "termo");// Adicione uma pesquisaparâmetrourl.search // => "? q = termo"url.searchparams.set ("q", "x");// Alterar o valor deeste parâmetrourl.search // => "? q = x"url.searchparams.get ("q") // => "x": consulte ovalor do parâmetrourl.searchparams.has ("q") // => true: existe umq parâmetrourl.searchparams.has ("p") // => false: existesem parâmetro purl.searchparams.append ("opts", "1");// Adicione outra pesquisaparâmetrourl.search // => "? q = x & opts = 1"url.searchparams.append ("opts", "&");// Adicione outro valorPara o mesmo nomeurl.search // => "?Q = X & OPTS =%26 ": Nota Escapeurl.searchparams.get ("opts") // => "1": o primeirovalor

Erro ao traduzir esta página.

funções de escape () e unescape (), que agora estão preteridasmas ainda amplamente implementado.Eles não devem ser usados.Quando Escape () e UNESCAPE () foram obsoletos, ecmascriptIntroduziu dois pares de funções globais alternativas:codeuri () e decodeuri ()codeuri () pega uma string como seu argumento e retorna um novostring em que caracteres não-ASCII, além de certos caracteres ASCII(como o espaço) são escapados.Decodeuri () reverte o processo.Os personagens que precisam ser escapados são convertidos pela primeira vez em seu UTF-8 codificação, então cada byte dessa codificação é substituído por um %xxSequência de fuga, onde xx são dois dígitos hexadecimais.Porquecodeuri () destina -se a codificar URLs inteiros, ele nãoEscape URL Separator caracteres como /,? e #.Mas issosignifica que o codeuri () não pode funcionar corretamente para os URLs queTenha esses personagens em seus vários componentes.codeuricomponent () e decodeuricomponent ()Este par de funções funciona como o codeuri () edecodeuri (), exceto que eles pretendem escapar do indivíduocomponentes de um URI, então eles também escapam de personagens como /,? e# que são usados ??para separar esses componentes.Estes são os maisÚtil das funções do URL legado, mas esteja ciente de quecodeuricomponent () escapará / caracteres em um caminhoNome que você provavelmente não deseja escapar.E vai converterespaços em um parâmetro de consulta para %20, embora os espaços sejamdeveria ser escapar com A + naquela parte de um URL.O problema fundamental de todas essas funções legadas é que elasprouce aplicar um único esquema de codificação a todas as partes de um URL quando o

O fato é que diferentes partes de um URL usam codificações diferentes. Se você quer um URL adequadamente formatado e codificado, a solução é simplesmente usar a classe URL para toda a manipulação de URL que você faz.

11.10 Timers

Desde os primeiros dias de JavaScript, os navegadores da Web definiram duas funções - setTimeout () e setInterval () - que permitem programar para pedir ao navegador que invocasse uma função após um especificado quantidade de tempo decorrida ou para invocar a função repetidamente em um intervalo especificado. Essas funções nunca foram padronizadas como parte do idioma central, mas eles funcionam em todos os navegadores e em nós são uma parte de fato da biblioteca padrão JavaScript. O primeiro argumento para setTimeout () é uma função, e o segundo argumento é um número que especifica quantos milissegundos devem decorrer antes que a função seja invocada. Após a quantidade especificada de tempo (e talvez um pouco mais se o sistema estiver ocupado), a função irá ser chamada sem argumentos. Aqui, por exemplo, são três setTimeout () chamando mensagens de console de impressão após um segundo, dois segundos e três segundos:

```
setTimeout (() => {console.log ("pronto ...");}, 1000);setTimeout (() => {console.log ("set ...");}, 2000);setTimeout (() => {console.log ("go!");}, 3000);
```

Observe que o setTimeout () não espera o tempo para decorrer antes de retornar. Todas as três linhas de código neste exemplo são executadas quase instantaneamente. Mas então nada acontece até que 1.000 milissegundos se destacam.

Erro ao traduzir esta página.

Relógio digital com a API do console:// Uma vez por segundo: limpe o console e imprima a corrente de tempo Deixe relógio = setInterval (() => {console.clear (); console.log (new Date ().toLocaleTimeString ())}, 1000); // Após 10 segundos: Pare o código repetido acima.setTimeout (() => {ClearInterval (relógio);}, 10000); Veremos setTimeout () e setInterval () novamente quando nós cubra a programação assíncrona no capítulo 13.11.11 Resumo Aprender uma linguagem de programação não é apenas dominar a gramática. É igualmente importante estudar a biblioteca padrão para que você está familiarizado com todas as ferramentas enviadas com o idioma. Este capítulo documentou a biblioteca padrão do JavaScript, que inclui: Estruturas de dados importantes, como matrizes de conjunto, mapa e digitado. As classes de data e URL para trabalhar com datas e URLs. Gramática de expressão regular do JavaScript e sua classe Regexp Para correspondência de padrões textuais. Biblioteca de internacionalização da JavaScript para datas de formatação, tempo e números e para classificar seqüências. O objeto JSON para serializar e desserializar estruturas simples e o objeto de console para registrar mensagens.

1Nem tudo documentado aqui é definido pela especificação do idioma JavaScript:Algumas das classes e funções documentadas aqui foram implementadas pela primeira vez na webnavegadores e depois adotados por nós, tornando -os membros de fato do JavaScriptBiblioteca padrão.2Esta ordem de iteração previsível é outra coisa sobre os conjuntos de JavaScript que PythonOs programadores podem achar surpreendente.3Matrizes digitadas foram introduzidas pela primeira vez no JavaScript do lado do cliente quando os navegadoresda Web adicionaramSuporte para gráficos WebGL.O que há de novo no ES6 é que eles foram elevados a umRecurso do idioma central.4Exceto dentro de uma classe de caracteres (colchetes quadrados), onde \ b corresponde ao backspacepersonagem.5Analisar URLs com expressões regulares não é uma boa ideia.Veja §11.9 para um mais robustoAnalizador de URL.6C Programadores reconhecerão muitas dessas seqüências de personagens do Printf ()função.

Capítulo 12. Iteradores e Geradores
Objetos iteráveis ??e seus iteradores associados são uma característica do ES6 que vimos várias vezes ao longo deste livro. Matrizes (incluindo TypeDarrays) são iteráveis, assim como as cordas e os objetos de ajuste e mapa. Esse significa que o conteúdo dessas estruturas de dados pode ser iterado - loopado acaba - com o loop for/de, como vimos no §5.4.4: deixe soma = 0; para (vamos i de [1,2,3]) { // loop uma vez para cada um desses valores soma += i;} soma // => 6 Os iteradores também podem ser usados ??com o ... operador para expandir ou "espalhar" um objeto iterável em um inicializador de matriz ou invocação de funções, como nós SAW em §7.1.2: deixe chars = [... "ABCD"]; // chars == ["A", "B", "C", "D"] deixe dados = [1, 2, 3, 4, 5]; Math.max (... dados) // => 5 Os iteradores podem ser usados ??com a atribuição de destruição: Deixe PurpleHaze = uint8Array.of (255, 0, 255, 128); Seja [r, g, b, a] = purplehaze; // a == 128 Quando você itera um objeto de mapa, os valores retornados são [chave,

valor] pares, que funcionam bem com a atribuição de destruição em um para/de loop: Seja M = novo mapa ([[{"One", 1}, {"dois", 2}]); para (vamos [k, v] de m) console.log (k, v); // Logs 'One 1' e 'dois 2'. Se você deseja iterar apenas as chaves ou apenas os valores, em vez de Pares, você pode usar os métodos Keys () e valores (): [... m] // => [[{"One", 1}, {"dois", 2}]]: Padrão de iteração [... M.Entries ()] // => [[{"One", 1}, {"dois", 2}]]: entradas () Método é o mesmo [... M.Keys ()] // => ["One", "Two"]]: Keys () Método itera apenas as chaves de mapa [... M.Values ?? ()] // => [1, 2]: Values ?? () Método itera apenas Valores do mapa Finalmente, várias funções e construtores internos que são comumente usados com objetos de matriz são realmente escritos (em ES6 mais tarde) aceitar iteradores arbitrários. O construtor Set () é Uma dessas API: // As cordas são iteráveis, então os dois conjuntos são iguais: novo conjunto ("abc") // => new Set ("A", "B", "C") Este capítulo explica como os iteradores funcionam e demonstra como Crie suas próprias estruturas de dados que sejam iteráveis. Depois de explicar os básicos iteradores, este capítulo abrange geradores, um novo recurso poderoso do ES6. Isso é usado principalmente como uma maneira particularmente fácil de criar iteradores. 12.1 como os iteradores funcionam

O operador for/de loop e espalhamento trabalha perfeitamente com iterávelobjetos, mas vale a pena entender o que está realmente acontecendo comfazer a iteração funcionar.Existem três tipos separados que você precisaEntenda para entender a iteração em JavaScript.Primeiro, há oObjetos iteráveis: esses são tipos como matriz, conjunto e mapa que podem seriterado.Segundo, existe o próprio objeto de iterador, que executa oiteração.E terceiro, existe o objeto de resultado da iteração que mantém oresultado de cada etapa da iteração.Um objeto iterável é qualquer objeto com um método de iterador especial queRetorna um objeto iterador.Um iterador é qualquer objeto com um próximo ()Método que retorna um objeto de resultado da iteração.E um resultado de iteraçãoO objeto é um objeto com propriedades nomeadas e feitas.Para iterarUm objeto iterável, você primeiro chama seu método de iterador para obter um iteradorobjeto.Em seguida, você chama o método próximo () do objeto iteradorrepetidamente até que o valor retornado tenha sua propriedade concluída definida como true.O mais complicado disso é que o método do iterador de um iterávelO objeto não tem um nome convencional, mas usa o símboloSymbol.iterator como seu nome.Então, um simples para/de loop sobre umobjeto iterável iterável também pode ser escrito da maneira mais difícil, comoesse:deixe iterável = [99];deixe iterator = iterable [symbol.iterator] ():for (let resultado = iterator.next ();! result.done; resultado = iterator.Next ()) {console.log (resultado.value) // resultado.value == 99}O objeto iterador dos tipos de dados iteráveis ??embutidos é iterável.

Erro ao traduzir esta página.

* Range define um método tem () para testar se um determinado Número é um membro* da faixa. O alcance é iterável e itera todos os números inteiros dentro do intervalo.*/intervalo de classe {construtor (de, para) {this.from = de; this.to = para;}// Faça um intervalo agir como um conjunto de números tem (x) {return typeof x === "número" && this.from <= x && x <= this.to;}// retorna a representação da string do intervalo usando o conjunto notaçāo to string () {return `x | \${this.from} ? x ? \$ {this.to}`;}// Torne uma faixa iterável retornando um objeto Iterador// Observe que o nome deste método é um símbolo especial, não é uma string.[Symbol.iterator] () {// Cada instância do iterador deve iterar o intervalo independentemente de// outros. Então, precisamos de uma variável de estado para rastrear nossoLocalização no// iteração. Começamos no primeiro número inteiro> = de.deixe o próximo = math.ceil (this.from); // este é o próximo valor que retornamos deixa last = this.to; // Não vamos voltar qualquer coisa> isso retornar {// este é o objeto iterador// este método próximo () é o que faz disso um objeto iterador.// ele deve retornar um objeto de resultado do iterador.próximo() {retornar (a seguir <= último) // se não tivermos retornou o último valor ainda?{Valor: Next ++} // Retorne o próximo valor e incrementá -lo: {done: true};// de outra forma indica

que terminamos.}// Como conveniência, fazemos o próprio iteradoriterável.[Symbol.iterator] () {return this;});}} para (deixe x de novo intervalo (1,10)) console.log (x);// Logs números 1 a 10[... novo intervalo (-2,2)] // => [-2, -1, 0, 1, 2]Além de tornar suas aulas iteráveis, pode ser bastante útilDefina funções que retornam valores iteráveis.Considere estes iteráveis-alternativas baseadas nos métodos map () e filter () deJavaScript Matrizes:// retorna um objeto iterável que itera o resultado deAplicando f ()// para cada valor da fonte iterávelmapa de função (iterable, f) {deixe iterator = iterable [symbol.iterator] ();retornar {// este objeto é iterador e iterável[Symbol.iterator] () {return this;},próximo() {Seja v = iterator.Next ();if (v.done) {retornar v;} outro {return {value: f (v.value)};}}};};// mapeie uma variedade de números inteiros para seus quadrados e converter para umvariedade

[... mapa (novo intervalo (1,4), x => x*x)] // => [1, 4, 9, 16]// retorna um objeto iterável que filtra o especificadoiterável,// iterando apenas os elementos para os quais o predicadoreturna verdadeirofiltro de função (iterável, predicado) {deixe iterator = iterable [symbol.iterator] () ;retornar {// este objeto é iterador e iterável[Symbol.iterator] () {return this;} ;próximo() {para(;;) {Seja v = iterator.Next ();if (v.done || predicado (v.value)) {retornar v;}}};};// filtrar um intervalo para que fiquemos com apenas números pares[... filtro (novo intervalo (1,10), x => x % 2 === 0)] // =>[2,4,6,8,10]Uma característica essencial de objetos iteráveis ??e iteradores é que eles são inherentemente preguiçoso: quando a computação é necessária para calcular o próximo valor, esse cálculo pode ser adiado até que o valor seja realmente necessário.Suponha, por exemplo, que você tenha uma série muito longa de texto que você deseja tokenizar em palavras separadas pelo espaço.Você poderiaBasta usar o método split () da sua string, mas se você fizer isso,então toda a string deve ser processada antes que você possa usar até oPrimeira palavra.E você acaba alocando muita memória para os devolvidosArray e todas as cordas dentro dela.Aqui está uma função que permite vocêpara atingir preguiçosamente as palavras de uma corda sem mantê-lasmemória de uma só vez (no ES2020, essa função seria muito mais fácil de

Implementar usando o método de mattall () de retorno do iterador ()descrito em §11.3.2):Palavras de função {var r = \s+| \$ /g;// corresponde a um ou mais espaços ou fimr.LastIndex = s.Match (/[^]/).// Comece a combinarno primeiro não espaço retornar {// retorna um objeto iterador iterável[Symbol.iterator] () {// Isso nos faz iterável devolver isso;},próximo () {// isso nos torna um iterador deixa iniciar = r.LastIndex;// retomar onde oA última partida terminouse (start <s.length) {// se não formos feitoSeja Match = R.Exec (S);// corresponde ao próximo limite da palavrarse (corresponder) {// se encontrarmos um, devolver a palavra retornar {valor: s.substring (start,match.index)};}}return {done: true};// Caso contrário, digamos que terminamos}};}{... palavras ("abc def ghi!")} // => ["abc", "def", "ghi!"]]}12.2.1 ?Fechando? um iterador: o método de retorno Imagine uma variante JavaScript (do lado do servidor) das palavras () iterador

Erro ao traduzir esta página.

JavaScript, então, quando você está criando APIs, é uma boa ideia usá-las quando possível. Mas tendo que trabalhar com um objeto iterável, seu iterador-objeto, e os objetos de resultado do iterador tornam o processo um pouco complicado. Felizmente, os geradores podem simplificar drasticamente o Criação de iteradores personalizados, como veremos no restante deste capítulo.

12.3 Geradores

Um gerador é um tipo de iterador definido com um novo ES6 poderoso sintaxe; é particularmente útil quando os valores a serem iterados não são elementos de uma estrutura de dados, mas o resultado de uma computação. Para criar um gerador, você deve primeiro definir uma função de gerador. Uma função do gerador é sintaticamente como uma função JavaScript regular, mas é definida com a função de palavra-chave * em vez de função. (Tecnicamente, essa não é uma palavra-chave nova, apenas uma * depois da palavra-chave função e antes do nome da função.) Quando você invoca um função do gerador, na verdade não executa o corpo da função, mas em vez disso, retorna um objeto gerador. Este objeto gerador é um iterador. Chamar seu método próximo () causa o corpo da função do gerador para funcionar desde o início (ou qualquer que seja sua posição atual) até chegar a uma declaração de rendimento. O rendimento é novo no ES6 e é algo como um宣言 de retorno. O valor da declaração de rendimento se torna o valor retornado pela próxima () chamada no iterador. Um exemplo faz este mais claro:

```
// uma função de gerador que gera o conjunto de um dígito (Base-10)
Prima.função* onedigitPries () { // invocando esta função não execute o código
    ...
}
```

rendimento 2:// mas apenas retorna um geradorobjeto.Chamandorendimento 3;// O Método Next ()Gerador é executadorendimento 5;// o código até um rendimentoA declaração fornecerendimento 7;// o valor de retorno para oMétodo seguinte ().};// Quando invocamos a função do gerador, temos um geradordeixe os primos = onedigitPries ():// um gerador é um objeto de iterador que itera ovalores produzidosPrimes.Next (). Valor // => 2Primes.Next (). Valor // => 3Primes.Next (). Valor // => 5Primes.Next (). Valor // => 7Primes.next (). feito // => true// Os geradores têm um método de símbolo.iterator para torná -lositerávelprimos [symbol.iterator] () // => primos// Podemos usar geradores como outros tipos iteráveis[... onedigitPries ()] // => [2,3,5,7]deixe soma = 0;para (deixe o primo do onedigitPriMes ()) soma += prime;soma // => 17Neste exemplo, usamos uma declaração de função* para definir umgerador.Como funções regulares, no entanto, também podemos definirgeradores em forma de expressão.Mais uma vez, apenas colocamos um asterisco depois a palavra -chave da função:const seq = função*(de, para) {para (vamos i = de; i <= para; i ++) rendimento i;};[... SEQ (3,5)] // => [3, 4, 5]

Nas classes e literais de objetos, podemos usar a notação de talha para omitir oFunção de palavra -chave inteiramente quando definimos métodos.Para definir umgerador neste contexto, simplesmente usamos um asterisco antes do métodoNome onde a palavra -chave da função teria sido, se a usássemos:Seja o = {x: 1, y: 2, z: 3, // um gerador que gera cada uma das chaves desteobjeto*g () {para (deixe a chave do object.keys (this)) {chave de rendimento;}}}; [... o.g ()] // => ["x", "y", "z", "g"]Observe que não há como escrever uma função de gerador usando setaFunção Sintaxe.Os geradores geralmente facilitam a definição de classes iteráveis.Podemos substituir o método [symbol.iterator] () mostrar emExemplo 12-1 com um muito mais curto *[Symbol.iterator & rbrack; () Função do gerador que pareceassim:[Symbol.iterator] () {para (vamos x = math.ceil (this.from); x <= this.to; x ++)rendimento x;}Veja o exemplo 9-3 no capítulo 9 para ver este iterador baseado em geradorfunção no contexto.

12.3.1 Exemplos de geradoresOs geradores são mais interessantes se realmente gerarem os valoresEles produzem fazendo algum tipo de computação.Aqui, por exemplo, é umFunção do gerador que gera números de Fibonacci:função* fibonaccisequence () {Seja x = 0, y = 1;para(;;) {rendimento y;[x, y] = [y, x+y];// Nota: atribuição de destruição}}Observe que a função do gerador FibonAccisequence () aqui temUm loop infinito e produz valores para sempre sem retornar.Se issoO gerador é usado com o ... Spread Operator, ele fará um loop atéA memória está esgotada e o programa trava.Com cuidado, é possívelPara usá-lo em um loop for/de, no entanto:// retorna o número de fibonacci do Nth Fibonacci:função fibonacci (n) {para (deixe f de fibonaccisequence ()) {if (n-- <= 0) retornar f;}}Fibonacci (20) // => 10946Esse tipo de gerador infinito se torna mais útil com uma tomada ()gerador como este:// produz os primeiros n elementos do iterável especificadoobjeto:função* Take (n, iterable) {deixe = iterable [symbol.iterator] ():// Obtenha iterador para

```
objeto iterávelwhile (n--> 0) { // loop n vezes:deixe o próximo = it.Next () ;// Obtenha o próximo item do iterador.if  
(next.done) retornar; // se não houver mais valores, retorne mais cedo else rendimento a seguir. Value; // caso  
contrário, produza o valor} // Uma matriz dos 5 primeiros números de Fibonacci[... Take (5, FibonAccisequence ())]  
// => [1, 1, 2, 3, 5] Aqui está outra função de gerador útil que intercala os elementos de múltiplos objetos iteráveis: //  
recebeu uma variedade de iteráveis, produz seus elementos em ordem intercalada. função * zip (... iterables) { //  
Obtenha um iterador para cada iterável let iterators = iterables.map (i => i [símbolo.iterator] ()) ;deixe index =  
0; enquanto (iterators.length > 0) { // enquanto houver ainda alguns iteradores if (index > = iterators.length) { // se  
chegarmos ao último iterador índice = 0; // Volte para o primeiro. } Deixe item = iteradores [index] .Next () ;// Obtenha o  
próximo item do próximo iterador. if (item.done) { // se isso iterador está feito iterators.splice (índice, 1); // então  
remova da matriz. } else { // caso contrário, ceder item.value; // produz o valor iterado índice ++; // e siga em frente o  
próximo iterador.
```

}}}// interlame três objetos iteráveis[... ZIP (ONEDIGITPRIMES (), "AB", [0])] // =>[2, "A", 0,3, "B", 5,7]12.3.2 Rendimento* e geradores recursivosAlém do gerador zip () definido no exemplo anterior,Pode ser útil ter uma função de gerador semelhante que produz elementos de múltiplos objetos iteráveis ??sequencialmente, em vez de intercalando -os.Poderíamos escrever aquele gerador assim:função* sequência (... iterables) {para (deixe iterável de iterables) {para (deixe o item de iterável) {item de rendimento;}}}{... sequência ("ABC", OneDigitPrimes ())]} // =>["A", "B", "C", 2,3,5,7]Este processo de produzir os elementos de algum outro objeto iterável éComum o suficiente em funções geradoras que o ES6 possui sintaxe especial paraisto.A palavra -chave de rendimento* é como rendimento, exceto que, em vez de produzir um único valor, ele itera um objeto iterável e produz cada um dosos valores resultantes.A função do gerador de sequências () que temos usado pode ser simplificado com rendimento* assim:função* sequência (... iterables) {para (deixe iterável de iterables) {

rendimento* iterável;}}][... sequência ("ABC", OneDigitPries ())] // =>["A", "B", "C", 2,3,5,7]O método da matriz foreach () é frequentemente uma maneira elegante de fazer um loop os elementos de uma matriz, então você pode ser tentado a escrever o sequence () função assim: função* sequência (... iterables) {iterables.foreach (iterable => rende* iterable); //Erro}Isso não funciona, no entanto. rendimento e rendimento* só podem ser usados nas funções do gerador, mas a função de seta aninhada neste código é uma função regular, não uma função do gerador de função*, então o rendimento é não é permitido. rendimento* pode ser usado com qualquer tipo de objeto iterável, incluindo iteráveis ??implementados com geradores. Isso significa que o rendimento* nos permite definir geradores recursivos, e você pode usar esse recurso para permitir iteração simples não recursiva sobre uma árvore recursivamente definida estrutura, por exemplo.

12.4 Recursos avançados do gerador

O uso mais comum das funções geradoras é criar iteradores, mas a característica fundamental dos geradores é que eles nos permitem pausar uma computação, produzir resultados intermediários e depois retomar o

Computação mais tarde. Isso significa que os geradores têm recursos além dos iteradores, e exploramos esses recursos no seguinte seção.

12.4.1 O valor de retorno de uma função de gerador

As funções do gerador que vimos até agora não tiveram retornos declarados, ou se tiverem, foram usadas para causar um início de retorno, para não retornar um valor. Como qualquer função, porém, um gerador A função pode retornar um valor. Para entender o que acontece neste caso, lembre-se de como funciona a iteração. O valor de retorno do próximo () A função é um objeto que possui uma propriedade de valor e/ou um feito de propriedade. Com iteradores e geradores típicos, se a propriedade do valor é definida, então a propriedade feita é indefinida ou é falsa. E se feito é verdadeiro, então o valor é indefinido. Mas no caso de um gerador que retorna um valor, a chamada final para a próxima retorna um objeto, isso tem valor e feito definido. A propriedade Value segura o valor de retorno da função do gerador e a propriedade feita é verdade, indicando que não há mais valores para iterar. Esta final O valor é ignorado pelo loop for/de e está disponível para codificar que itera manualmente com chamadas explícitas para proximo(): função *oneanddone () { rendimento 1; retornar "feito"; } // O valor de retorno não aparece na iteração normal. [...]

```
Oneanddone ()] // => [1]
```

```
// mas está disponível se você ligar explicitamente a seguir ()deixe o gerador = oneanddone ();generator.next () // => {value: 1, feito: false}generator.next () // => {value: "done", feito: true}// Se o gerador já estiver feito, o valor de retorno não será voltou novamentegenerator.next () // => {value: indefinido, feito:verdadeiro }12.4.2 O valor de uma expressão de rendimentoNa discussão anterior, tratamos o rendimento como uma declaração queleva um valor, mas não tem valor próprio.De fato, no entanto, o rendimento é uma expressão e pode ter um valor.Quando o próximo método () de um gerador é invocado, o geradorA função é executada até atingir uma expressão de rendimento.A expressão que segue a palavra -chave de rendimento é avaliada e esse valor se torna oValor de retorno do próximo () invocação.Neste ponto, o geradorA função para de executar bem no meio da avaliação do rendimentoexpressão.Na próxima vez que o método próximo () do gerador for chamado, o argumento passou para o próximo () se torna o valor do rendimento de expressão que foi interrompida.Então o gerador retorna valores para o chamador com rendimento e o chamador passa valores para o gerador com o próximo ().O gerador e o chamador são dois fluxos separados de Execução Passando valores (e controle) para frente e para trás.A seguirO código ilustra:função* smallNumbers () {console.log ("Next () invocou a primeira vez; argumento descartado ");Seja y1 = rendimento 1;// y1 == "b"
```

Console.log ("Next () invocou uma segunda vez com argumento",y1);Seja y2 = rendimento 2;// y2 == "c"console.log ("Next () invocou uma terceira vez com argumento",y2);Seja y3 = rendimento 3;// y3 == "d"console.log ("Next () invocou a quarta vez com argumento",y3);retornar 4;}Seja g = smallnumbers ();console.log ("gerador criado; nenhum código é executado ainda");Seja n1 = g.next ("a");// n1.value == 1console.log ("gerador rendido", n1.value);Seja n2 = g.next ("b");// n2.value == 2console.log ("gerador rendido", n2.value);Seja n3 = g.next ("c");// n3.Value == 3console.log ("gerador rendido", n3.Value);Seja n4 = g.next ("d");// n4 == {value: 4, feito: true}console.log ("gerador retornado", n4.Value);Quando esse código é executado, ele produz a seguinte saída que demonstra o visto e vindos entre os dois blocos de código: gerador criado; Nenhum código é executado ainda. Em seguida () invocou a primeira vez; argumento descartado. O gerador produziu 1. Em seguida () invocou uma segunda vez com o argumento BO gerador produziu 2. Em seguida () invocou uma terceira vez com o argumento CO gerador produziu 3. Em seguida () invocou uma quarta vez com o argumento DO gerador retornou 4. Observe a assimetria neste código. A primeira invocação de Next () inicia o gerador, mas o valor passado para essa invocação não está acessível para o gerador.

12.4.3 Os métodos de retorno () e tiro () de umGeradorVimos que você pode receber valores produzidos ou devolvidos por umFunção do gerador.E você pode passar valores para um gerador em execução porpassando esses valores quando você chama o próximo () método do gerador.Além de fornecer entrada para um gerador com o próximo (), você pode também alterar o fluxo de controle dentro do gerador chamando seuMétodos de return () e Throw ().Como os nomes sugerem, chamandoEsses métodos em um gerador fazem com que retorne um valor ou jogue umexceção como se a próxima declaração no gerador fosse um retorno oulançar.Lembre -se de anterior ao capítulo de que, se um iterador define umReturn () Método e iteração para mais cedo, depois o intérpretechama automaticamente o método Return () para dar uma chance ao iteradorpara fechar arquivos ou fazer outra limpeza.No caso de geradores, você não podeDefina um método de retorno () personalizado para lidar com a limpeza, mas você podeestruturar o código do gerador para usar uma declaração de tentativa/finalmente quegarante que a limpeza necessária seja feita (no bloco finalmente) quando o gerador retorna.Forçando o gerador a retornar, oO método de retorno interno () do gerador garante que o código de limpezaé executado quando o gerador não será mais usado.Assim como o método próximo () de um gerador nos permite passar arbitráriovalores em um gerador em execução, o método throw () de um geradornos dá uma maneira de enviar sinais arbitrários (na forma de exceções) para

um gerador.Chamar o método throw () sempre causa uma exceção dentro do gerador.Mas se a função do gerador for escrita com código de manipulação de exceção apropriado, a exceção não precisa ser fatal mas pode ser um meio de alterar o comportamento do gerador.Imagine, por exemplo, um contra-gerador que produz um sempresequência crescente de números inteiros.Isso poderia ser escrito para que uma exceção enviada com arremesso () redefiniria o contador para zero.Quando um gerador usa rendimento* para produzir valores de outros objeto iterável, então uma chamada para o próximo () método do gerador causa uma chamada para o método próximo () do objeto iterável.O mesmo é verdadeiro dos métodos de retorno () e arremesso ().Se um gerador usar render* em um objeto iterável que tem esses métodos definidos, então Chamando devolver () ou arremesso () no gerador causa o iterador Método retornar () ou Throw () a ser chamado por sua vez.Todos os iteradores Deve ter um método próximo ().Iteradores que precisam limpar depois A iteração incompleta deve definir um método de retorno ().E qualquer iterator pode definir um método de arremesso (), embora eu não conheça nenhuma razão prática para fazê-lo.12.4.4 Uma nota final sobre geradoresOs geradores são uma estrutura de controle generalizada muito poderosa.Eles dão à capacidade de pausar um cálculo com rendimento e reiniciá-lo novamente em alguns arbitrários posteriormente com um valor de entrada arbitrária.É possível usar geradores para criar um tipo de sistema de roteamento cooperativo dentro do Código JavaScript de thread único.E é possível usar geradores para Máscara as partes assíncronas do seu programa para que seu código apareça

sequencial e síncrono, embora algumas de suas chamadas de funçõesão realmente assíncronos e dependem dos eventos da rede.Tentar fazer essas coisas com geradores leva ao código que é a menteDiffícilmente difícil de entender ou explicar.Foi feito, no entanto,e o único caso de uso realmente prático tem sido para gerenciar código assíncrono.JavaScript agora tem palavras -chave assíncronas e aguardas(veja o capítulo 13) Para esse mesmo objetivo, no entanto, e não há maisQualquer motivo para abusar dos geradores dessa maneira.12.5 ResumoNeste capítulo, você aprendeu:O loop for/of e o ... o operador espalhado trabalham comobjetos iteráveis.Um objeto é iterável se tiver um método com o nome simbólico[Symbol.iterator] que retorna um objeto iterador.Um objeto iterador tem um método próximo () que retorna umobjeto de resultado da iteração.Um objeto de resultado da iteração possui uma propriedade de valor que detém oPróximo valor iterado, se houver um.Se a iteração tiverConcluído, então o objeto de resultado deve ter uma propriedade feitadefinido como true.Você pode implementar seus próprios objetos iteráveis, definindo um[Symbol.iterator] () Método que retorna um objetocom um método próximo () que retorna objetos de resultado da iteração.Você também pode implementar funções que aceitam o iteradorargumentos e valores de retorno do iter.

Funções do gerador (funções definidas com função*em vez de função) são outra maneira de definir iteradores. Quando você invoca uma função de gerador, o corpo da função não é executada imediatamente; Em vez disso, o valor de retorno é um objeto iterador iterável. Cada vez que o próximo () método do iterador é chamado, outro pedaço da função do gerador corre. As funções do gerador podem usar o operador de rendimento para especificar valores que são retornados pelo iterador. Cada chamada para a próxima () faz com que a função do gerador suba para o próximo rendimento expressão. O valor dessa expressão de rendimento se torna o valor retornado pelo iterador. Quando não há mais expressões de rendimento, então a função do gerador retorna e a iteração está completa.

Capítulo 13. AssíncronoJavaScriptAlguns programas de computador, como simulações científicas e máquinasOs
modelos de aprendizado, são ligados a computação: eles correm continuamente, sem pause, até que tenham
calculado seu resultado.O computador mais real do mundo realOs programas, no entanto, são significativamente
assíncronos.Isso significa issoEles geralmente precisam parar de calcular enquanto aguardam a chegada dos
dados ouPara que algum evento ocorra.Os programas JavaScript em um navegador da web sãonormalmente
orientado a eventos, o que significa que eles esperam o usuário clicar ouToque antes que eles realmente façam
qualquer coisa.E servidores baseados em JavaScriptnormalmente aguarda os pedidos do cliente chegarem pela
rede antes de elesfaça qualquer coisa.Esse tipo de programação assíncrona é comum emJavaScript, e este
capítulo documenta três idiomas importantesRecursos que ajudam a facilitar o trabalho com código
assíncrono.Promessas, novas no ES6, são objetos que representam o que ainda não está disponívelresultado de
uma operação assíncrona.As palavras -chave async e aguardamforam introduzidos no ES2017 e fornecem uma
nova sintaxe que simplificaProgramação assíncrona, permitindo que você estruture sua promessacódigo baseado
como se fosse síncrono.Finalmente, iteradores assíncronos o loop for/wait foi introduzido no ES2018 e permitir
que vocêTrabalhe com fluxos de eventos assíncronos usando loops simples queParece síncrono.

Ironicamente, embora o JavaScript forneça esses recursos poderosos para trabalhar com código assíncrono, não há recursos do núcleo idioma que são eles mesmos assíncronos. Para demonstrar promessas, assíncronas, aguardam, e para/aguardar, portanto, iremos primeiro. Faça um desvio para o Javascript do lado do cliente e do servidor para explicar alguns dos recursos assíncronos dos navegadores da Web e do nó. (Você pode aprender mais sobre o JavaScript do lado do cliente e do servidor nos capítulos 15 e 16.)

13.1 Programação assíncrona com retornos de chamada

Em seu nível mais fundamental, programação assíncrona em JavaScript é feito com retornos de chamada. Um retorno de chamada é uma função que você escreva e depois passe para outra função. Essa outra função então invoca ("chama de volta") sua função quando alguma condição é atendida ou alguns eventos (assíncronos) ocorrem. A invocação do retorno de chamada função você fornece notifica-o sobre a condição ou evento e. Às vezes, a invocação incluirá argumentos de função que fornecem detalhes adicionais. Isso é mais fácil de entender com algum exemplo concreto e as subseções a seguir demonstram várias formas de programação assíncrona baseada em retorno de chamada usando ambos JavaScript e nó.

13.1.1 Timers

Um dos tipos mais simples de assincronia é quando você deseja executar alguns o código após um certo período de tempo foi decorrido. Como vimos em §11.10, Você pode fazer isso com a função setTimeout ():

setTimeout (checkForupDates, 60000);O primeiro argumento para setTimeout () é uma função e a segunda éUm intervalo de tempo medido em milissegundos.No código anterior, umA função hipotética checkForupDates () será chamada de 60.000milissegundos (1 minuto) após a chamada setTimeout ().checkForupDates () é uma função de retorno de chamada que seu programapode definir, e setTimeout () é a função para a qual você invocaRegistre sua função de retorno de chamada e especifique sob o que assíncronoCondições deve ser invocada.setTimeout () chama a função de retorno de chamada especificada uma vez,não transmitindo argumentos e depois esquece.Se você está escrevendo umfunção que realmente verifica se há atualizações, você provavelmente deseja que ela seja executadarepetidamente.Você pode fazer isso usando setInterval () em vez desetTimeout ():// Ligue para o checkForupDates em um minuto e depois novamente a cada minuto depois dissoDeixe updateInterValid = setInterval (checkForupDates, 60000);// setInterval () retorna um valor que podemos usar para parar o repetido// Invocações chamando clearInterval ().(De forma similar,setTimeout ()// Retorna um valor que você pode passar para cleartimeout ())função stopcheckingForupDates () {ClearInterval (UpdateIntervalId);}13.1.2 EventosOs programas JavaScript do lado do cliente são quase universalmente orientados a eventos:

Em vez de executar algum tipo de computação predeterminada, ele normalmente espera o usuário fazer algo e depois responder aoAções do usuário.O navegador da web gera um evento quando o usuário pressiona uma tecla no teclado, move o mouse, clica em um mousebotão ou toca em um dispositivo de tela sensível ao toque.JavaScript orientado a eventosProgramas Registrar funções de retorno de chamada para tipos especificados de eventos emContextos especificados e o navegador da web chama essas funções sempre que os eventos especificados ocorrem.Essas funções de retorno de chamada são chamados manipuladores de eventos ou ouvintes de eventos, e eles são registrados comaddEventListener ():// Peça ao navegador da web para devolver um objeto que representa oHtml// elemento <butter> que corresponde a este seletor CSSLet Okk = Document.querySelector ('#confirmUpDateDialogButton.OKay');// Agora registre uma função de retorno de chamada a ser invocada quando ousuário// clica nesse botão.ok.Neste exemplo, ApplUpDate () é uma função hipotética de retorno de chamadaque assumimos ser implementado em outro lugar.A chamada para document.querySelector () retorna um objeto que representa umElemento especificado único na página da web.Nós chamamosaddEventListener () nesse elemento para registrar nosso retorno de chamada.Então o primeiro argumento para addEventListener () é uma string queEspecifica o tipo de evento em que estamos interessados ??- um clique do mouse ouToque em tela sensível ao toque, neste caso.Se o usuário clicar ou torneiras nesse específicoElemento da página da web, então o navegador invocará nossoFunção de retorno de chamada APLAPUPDATE (), passando um objeto que inclui

Detalhes (como o tempo e o ponteiro do mouse coordenam) sobre o evento.

13.1.3 Eventos de rede

Outra fonte comum de assincronia na programação JavaScript é solicitações de rede. JavaScript em execução no navegador pode buscar dados de um servidor da web com código como este:

```
function getCurrentVersionNumber(versãoCallback) {  
    // Nota: argumento de retorno de chamada  
    // Faça uma solicitação HTTP com script para uma API de versão de back-end.  
    // Deixe solicitação = novo XMLHttpRequest(); request.open("get", "http://www.example.com/api/version"); request.send();  
    // Registre um retorno de chamada que será invocado quando a resposta chegar.  
    request.onload = function () {  
        if (request.status === 200) {  
            // Se o status http for bom, obtenha o número da versão e ligue para o retorno de chamada.  
            // Deixe o currentVersion = parseFloat(request.responseText);  
            // Caso contrário, relate um erro ao retorno de chamada.  
            // VersionCallback(null, CurrentVersion);  
        } else {  
            // Registre outro retorno de chamada que será invocado para erros de rede.  
            request.onerror = request.ontimeout = function (e) {  
                VersionCallback(e.type, null);  
            };  
        }  
    };  
}
```

O código JavaScript do lado do cliente pode usar a classe XMLHttpRequest PLUSFunções de retorno de chamada para fazer solicitações HTTP e manipular assíncrona resposta do servidor quando chega.OGetCurrentVersionNumber () Função definida aqui (podemosImagine que é usado pelos hipotéticos checkForUpdates ()Função discutimos no §13.1.1) faz uma solicitação HTTP e definem manipuladores de eventos que serão invocados quando a resposta do servidor for recebido ou quando um tempo limite ou outro erro faz com que a solicitação falhe.Observe que o exemplo de código acima não chama addEventListener () como nosso exemplo anterior fez.Para a maioria da webAPIs (incluindo este), os manipuladores de eventos podem ser definidos invocando addEventListener () no objeto que gera o evento ePassando o nome do evento de interesse junto com o retorno de chamada função.Normalmente, porém, você também pode registrar um único ouvinte de evento atribuindo -o diretamente a uma propriedade do objeto.Isso é o que fazemosNeste código de exemplo, atribuindo funções ao Onload, OnError,e propriedades on -timeout.Por convenção, propriedades do ouvinte de eventosComo estes, sempre tem nomes que começam.addEventListener () é a técnica mais flexível porque permite vários manipuladores de eventos.Mas nos casos em que você tem certeza queNenhum outro código precisará registrar um ouvinte para o mesmo objeto eTipo de evento, pode ser mais simples simplesmente definir a propriedade apropriada comoSeu retorno de chamada.Outra coisa a observar sobre o getCurrentVersionNumber ()função neste exemplo, o código é que, porque faz umsolicitação assíncrona, ele não pode retornar síncrono o valor (o1

número de versão atual) em que o chamador está interessado. Em vez disso, oO chamador passa uma função de retorno de chamada, que é invocada quando o resultado é pronto ou quando ocorrer um erro.Nesse caso, o chamador fornece umFunção de retorno de chamada que espera dois argumentos.Se o XMLHttpRequestFunciona corretamente, então getCurrentVersionNumber () invoca oretorno de chamada com um primeiro argumento nulo e o número da versão como osegundo argumento.Ou, se ocorrer um erro, entãogetCurrentVersionNumber () chama o retorno de chamada com errodetalhes no primeiro argumento e nulo como o segundo argumento.13.1.4 retornos de chamada e eventos no nóO ambiente JavaScript do lado do Node.js é profundamenteassíncrono e define muitas APIs que usam retornos de chamada e eventos.A API padrão para ler o conteúdo de um arquivo, por exemplo, éassíncrono e chama uma função de retorno de chamada quando o conteúdo doO arquivo foi lido:const fs = requer ("fs");// O módulo "FS" tem sistema de arquivos-APIs relacionadasdeixe opções = {// um objeto para manter opções parano nosso programaa// Opções padrão iriam aqui}// Leia um arquivo de configuração e ligue para a função de retorno de chamadafs.readFile ("config.json", "utf-8", (err, texto) => {if (err) {// Se houve um erro, exiba um aviso, mascontinuarconsole.warn ("Não foi possível ler o arquivo de configuração:", err);} outro {// caso contrário, analise o conteúdo do arquivo e atribua ao objeto de opçõesObject.assign (options, json.parse (text));

// Em ambos os casos, agora podemos começar a executar o programastartProgram (opções);});A função fs.readFile () do Node recebe um retorno de chamada de dois parâmetros comoseu último argumento.Ele lê o arquivo especificado de forma assíncrona e depoisInvoca o retorno de chamada.Se o arquivo foi lido com sucesso, ele passa o arquivoconteúdo como o segundo argumento de retorno de chamada.Se houve um erro, épassa o erro como o primeiro argumento de retorno de chamada.Neste exemplo, nóexpressar o retorno de chamada como uma função de seta, que é um sucinto eSintaxe natural para esse tipo de operação simples.O Node também define uma série de APIs baseadas em eventos.A seguirA função mostra como fazer uma solicitação http para o conteúdo de umURL no nó.Tem duas camadas de código assíncrono tratado comovintes de eventos.Observe que o nó usa um método on () para registrarouvintes de eventos em vez de addEventListener ():const https = requer ("https");// Leia o conteúdo de texto do URL e passa de forma assíncronapara o retorno de chamada.função getText (url, retorno de chamada) {// Inicie um http get solicitação para o URLsolicitação = https.get (url);// Registre uma função para lidar com o evento "resposta".request.on ("resposta", resposta => {// O evento de resposta significa que os cabeçalhos de respostaforam recebidosSeja httpstatus = resposta.statusCode;// O corpo da resposta HTTP não foirecebido ainda.

```
// Então, registramos mais manipuladores de eventos para serem chamados Quando
chegar.Response.setEncoding ("UTF-8");// estamos esperando Texto unicode Deixe Body = "";// que vamos acumular
aqui.// Este manipulador de eventos é chamado quando um pedaço do corpo está pronto respostas.on ("dados",
chunk => {body += chunk;});// Este manipulador de eventos é chamado quando a resposta é completa response.on
("end", () => {if (httpstatus === 200) {// se o httpA resposta foi boa retorno de chamada (nulo, corpo); // Passe o
corpo de resposta para o retorno de chamada} else {//, caso contrário, passe um erro retorno de chamada
(httpstatus, nulo);}});});// Também registramos um manipulador de eventos para o nível inferior erros de
rederequest.on ("erro", (err) => {retorno de chamada (err, nulo);});}13.2 promessas Agora que vimos exemplos de
retorno de chamada e eventos baseados em eventos Programação assíncrona em JavaScript do lado do cliente e
do servidor Ambientes, podemos apresentar promessas, um recurso de idioma central Projetado para simplificar a
programação assíncrona.
```

Uma promessa é um objeto que representa o resultado de um assíncronocomputação. Esse resultado pode ou não estar pronto ainda, e a promessaAPI é intencionalmente vaga sobre isso: não há como obter o valor de uma promessa; você só pode pedir a promessa de ligar para umFunção de retorno de chamada quando o valor estiver pronto. Se você está definindo umAPI assíncrona como a função getText ()seção, mas quero torná-la baseada em promessa, omitir o retorno de chamadaargumento e, em vez disso, retorne um objeto de promessa. O chamador pode então registre um ou mais retornos de chamada neste objeto de promessa e eles serão invocados quando a computação assíncrona é feita. Então, no nível mais simples, as promessas são apenas uma maneira diferente de trabalhar com retornos de chamada. No entanto, existem benefícios práticos em usá-los. Um problema real com a programação assíncrona baseada em retorno de chamada é que é comum acabar com retornos de chamada dentro de retornos de chamada dentro de retornos de chamada, com linhas de código tão altamente recuadas que é difícil. As promessas permitem que esse tipo de retorno de chamada aninhado seja reexpressado como uma cadeia de promessa mais linear que tende a ser mais fácil de ler e mais fácil razão sobre. Outro problema com retornos de chamada é que eles podem cometer erros de maneira difícil. Se uma função assíncrona (ou um invocado assíncrono retorno de chamada) lança uma exceção, não há como essa exceção propagar de volta ao iniciador da operação assíncrona. Este é um fato fundamental sobre programação assíncrona: quebra manuseio de exceção. A alternativa é para a pistameticulouslyamente propagar erros com argumentos de retorno de chamada e valores de retorno, mas isso é tedioso e difícil de acertar. Promessas ajuda aqui padronizando uma maneira de lidar com erros e fornecer uma maneira de erros se propagar

corretamente através de uma cadeia de promessas. Observe que as promessas representam os resultados futuros de único assíncronocálculos. Eles não podem ser usados ?? para representar repetidos assíncronoscálculos, no entanto. Mais tarde neste capítulo, escreveremos uma promessa-Alternativa baseada na função `setTimeout ()`, por exemplo. Mas Não podemos usar promessas para substituir o `setInterval ()` porque isso A função chama uma função de retorno de chamada repetidamente, o que é algo Que promessas simplesmente não foram projetadas para fazer. Da mesma forma, poderíamos usar umPromessa em vez do manipulador de eventos de "carga" de um `xmlhttprequestObjeto`, já que esse retorno de chamada é chamado apenas uma vez. Mas nós normalmente nõo usaria uma promessa no lugar de um manipulador de eventos de "clique" de umObjeto de botão HTML, já que normalmente queremos permitir que o usuário cliqueum botão várias vezes. As subseções a seguir: Explique a terminologia da promessa e mostre o uso básico de promessaMostre como as promessas podem ser acorrentadasDemonstrar como criar suas próprias APIs baseadas em promessas! IMPORTANTE As promessas parecem simples no começo, e o caso de uso básico para promessas é, de fato, direto e simples. Mas eles podem se tornar surpreendentemente confusos para Qualquer coisa além dos casos de uso mais simples. As promessas são um idioma poderoso paraprogramação assíncrona, mas você precisa entendê -los profundamente para usá -los corretamente e com confiança. Vale a pena dedicar um tempo para desenvolver tão profundoEntendendo, no entanto, e peço que você estude este longo capítulo com cuidado.

Erro ao traduzir esta página.

Método de um objeto de promessa várias vezes, cada uma das funções que você especificar será chamado quando o cálculo prometido estiver concluído. Ao contrário de muitos ouvintes de eventos, porém, uma promessa representa um único cálculo e cada função registrada com então () será invocado apenas uma vez. Vale a pena notar que a função que você passa em então () é invocada de forma assíncrona, mesmo que o assíncrono A computação já está completa quando você liga para então (). Em um nível sintático simples, o método então () é o distintivo característico das promessas, e é idiomático anexar. então () diretamente à invocação de funções que retorna a promessa, sem a etapa intermediária de atribuir o objeto Promise a uma variável. Também é idiomático nomear funções que retornam promessas e funções que usam os resultados das promessas com verbos e essas idiomas levar ao código que é particularmente fácil de ler:// Suponha que você tenha uma função como essa para exibir um usuário no perfilFunção DisplayUserProfile (perfil) { /* Implementação omitida */}// Veja como você pode usar essa função com uma promessa.// Observe como essa linha de código é quase como um inglês frase:getJSON ("api/user/perfil"). Então (DisplayUserProfile); Lidar com erros com promessasOperações assíncronas, particularmente aquelas que envolvem redes, normalmente pode falhar de várias maneiras, e o código robusto deve ser escrito para lidar com os erros que inevitavelmente ocorrerão.

Para promessas, podemos fazer isso passando uma segunda função para o então () método: getjson (" /api/user/perfil"). Então (DisplayUserProfile, handleprofileError); Uma promessa representa o resultado futuro de uma computação assíncrona, o que ocorre depois que o objeto Promise for criado. Porque a computação é realizada após a devolução do objeto de promessa, não há como o cálculo tradicionalmente retornar um valor ou jogar uma exceção que podemos pegar. As funções para as quais passamos então () fornecem alternativas. Quando um cálculo síncrono conclui normalmente, ele simplesmente retorna seu resultado ao seu chamador. Quando a computação assíncrona baseada em promessa é concluída normalmente, ela passa seu resultado para a função que é o primeiro argumento para então (). Quando algo dá errado em um cálculo síncrono, ela joga uma exceção que se propaga na pilha de chamadas até que haja um problema. Cláusula para lidar com isso. Quando uma computação assíncrona é executada, seu chamador não está mais na pilha, então se algo der errado, simplesmente não é possível reivindicar uma exceção de volta ao chamador. Em vez disso, os cálculos assíncronos baseados em promessa passam a exceção (normalmente como um objeto de erro de algum tipo, embora isso não seja necessário) para a segunda função passada para então (). Então, no código acima, se getjson () é executado normalmente, ele passa seu resultado para DisplayUserProfile (). Se houver um erro (o usuário não estiver conectado, o servidor está desativado, a conexão com a Internet do usuário caiu, a solicitação cronometrada, etc.), então getjson () passa um objeto de erro para

handleprofileError ().Na prática, é raro ver duas funções passadas para então ().Há um maneira melhor e mais idiomática de lidar com erros ao trabalhar comPromessas.Para entender, primeiro considere o que acontece se Getjson ()completa normalmente, mas ocorre um erro emDisplayUserProfile ().Essa função de retorno de chamada é invocadaassíncrono quando getjson () retorna, então também é assíncrono e não pode jogar significativamente uma exceção (porque não há código na pilha de chamadas para lidar com isso).A maneira mais idiomática de lidar com erros nesse código é assim:getjson ("/api/user/perfil"). Então (DisplayUserProfile) .Catch (handleprofileError);Com este código, um resultado normal de getjson () ainda é passado para displayUserProfile (), mas qualquer erro em getjson () ou emDisplayUserProfile () (incluindo todas as exceções jogadas por displayUserProfile) seja passado para handleprofileError ().O método Catch () é apenas uma abreviação de ligar então () com um primeiro argumento nulo e o manipulador de erros especificado funciona como o segundo argumento.Teremos mais a dizer sobre Catch () e este idioma de manipulação de errosQuando discutirmos as cadeias prometidas na próxima seção.Terminologia da promessaAntes de discutirmos mais promessas, vale a pena fazer uma pausa para definir alguns termos.Quando não somos

Programação e falamos sobre promessas humanas, dizemos que uma promessa é "mantida" ou "quebrada". Quando discutindo promessas de JavaScript, os termos equivalentes são "cumpridos" e "rejeitados". Imagine que você chamei o método então () de uma promessa e aprovou duas funções de retorno de chamada. Nós dizemos que a promessa foi cumprida se e quando o primeiro retorno de chamada for chamado. E dizemos que a promessa foi rejeitada se e quando o segundo retorno de chamada for chamado. Se uma promessa não for cumprida nem rejeitada, então está pendente. E uma vez que uma promessa é cumprida ou rejeitada, dizemos que está resolvida. Observação: uma promessa nunca pode ser cumprida e rejeitada. Uma vez que uma promessa se acalma, nunca mudará de cumprido a rejeitado ou vice-versa. Lembre-se de como definimos promessas no início desta seção: ? Uma promessa é um objeto que representa o resultado de uma operação assíncrona. É importante lembrar que as promessas não são apenas abstrair maneiras de registrar retornos de chamada para executar quando algum código assíncrono terminar - eles representam o resultado desse código assíncrono. Se o código assíncrono executar normalmente (e a promessa é cumprida), então isso é o resultado essencialmente o valor de retorno do código. E se o código assíncrono não concluir normalmente (e a promessa é rejeitada), então o resultado é um objeto de erro ou algum outro valor que o código pode ter jogado se não fosse assíncrono. Qualquer promessa que se resolveu tem um valor associado com ele, e esse valor não mudará. Se a promessa for cumprida, o valor é um valor de retorno que é passado para quaisquer funções de retorno de chamada registradas como o primeiro argumento de então (). Se a promessa for rejeitada, então o valor é um erro de algum tipo que é passado para qualquer função de retorno de chamada registrada com catch () ou como o segundo argumento de então (). A razão pela qual eu quero ser preciso sobre a terminologia da promessa é que as promessas também podem ser resolvidas. É fácil confundir este estado resolvido com o estado cumprido ou com o estado liquidado, mas não é precisamente o mesmo que. Compreender o estado resolvido é uma das chaves para um profundo entendimento das promessas, e voltarei a isso depois de discutirmos as cadeias de promessas abaixo.

13.2.2 Promessas de encadeamento

Um dos benefícios mais importantes das promessas é que eles fornecem uma maneira natural de expressar uma sequência de operações assíncronas como uma cadeia linear de invocações de método então (), sem ter que nenhuma operação dentro do retorno de chamada do anterior. Aqui, para exemplo, é uma cadeia de promessas hipotéticas:

```
busca('document.url') // faça uma solicitação HTTPS.  
then(Response => Response.json()) // Peça o JSON do corpo da resposta.  
then(document => {  
  // quando obtemos o documento  
  // renderize o documento para o usuário  
  // exiba o documento para o usuário  
})
```

}).THEN (renderizado => {// quando obtemos o documento renderizadoCacheIndatabase (renderizado); // cache no Banco de dados local.}).catch (error => handle (erro)); // lide com qualquer erro isso ocorre. Este código ilustra como uma cadeia de promessas pode facilitar expressar uma sequência de operações assíncronas. Nós não vamos discutir essa cadeia de promessas em particular, no entanto. Vamos continuar para explorar a idéia de usar cadeias de promessas para fazer solicitações HTTP, no entanto. No início deste capítulo, vimos o objeto XMLHttpRequest usado para fazer uma solicitação HTTP no JavaScript. Aquela é o objeto estranhamente nomeado tem uma API antiga e desajeitada, e foi amplamente substituída pela API de busca mais recente baseada em promessa (§15.11.1). Em sua forma mais simples, esta é a nova API HTTP é apenas a função busca (). Você passa por um URL, e ela retorna uma promessa. Essa promessa é cumprida quando a resposta HTTP começa a chegar e o status e os cabeçalhos HTTP estão disponíveis: busca ("/api/user/perfil"). Então (resposta => {// Quando a promessa resolver, temos status e cabeçalhos se (Response.ok && Response.Headers.get ("Content-Type") === "Application/json") {// O que podemos fazer aqui? Na verdade, não temos o corpo da resposta ainda.}}); Quando a promessa retornada por busca () é cumprida, ela passa

Objeto de resposta à função que você passou para o seu método então (). Este objeto de resposta fornece acesso ao status e aos cabeçalhos de solicitação, e também define métodos como text () e json (), que lhe dão acesso ao corpo da resposta no texto e em formas agitadas de JSON, respectivamente. Mas embora a promessa inicial seja cumprida, o corpo da resposta ainda pode não ter chegado. Então esses text () e json () métodos para acessar o corpo da resposta retorna Promessas. Aqui está uma maneira ingênua de usar Fetch () e o Método Response.json () para obter o corpo de uma resposta HTTP: busca (" /api / user / perfil "). Então (resposta => { Response.json () }. Então (perfil => { // Peça o json - corpo analisado // Quando o corpo da resposta chegar, será automaticamente // analisou como JSON e passou para esta função . displayUserProfile (perfil); }); }); Esta é uma maneira ingênua de usar Promessas porque as aninhamos, com retornos de chamada, que derrotam o objetivo. O idioma preferido é usar Promessas em uma cadeia seqüencial com código como este: busca (" /api / user / perfil "). then (resposta => { REPORTE DE REPORTE . JSON () }; }). then (perfil => { displayUserProfile (perfil); }); Vejamos as invocações do método neste código, ignorando o

Argumentos que são passados ??para os métodos:busca (). Então (). Então ()Quando mais de um método é invocado em uma única expressão como esta, Chamamos isso de cadeia de métodos. Sabemos que a função Fetch () retorna um objeto de promessa, e podemos ver que o primeiro. então () nesta cadeia Invoca um método sobre esse objeto de promessa retornado. Mas há um segundo. THEN () na cadeia, o que significa que a primeira invocação do Então () o método deve retornar uma promessa. Às vezes, quando uma API é projetada para usar esse tipo de método encadeamento, há apenas um único objeto, e cada método desse objeto Retorna o próprio objeto para facilitar o encadeamento. Não é assim Promete o trabalho, no entanto. Quando escrevemos uma cadeia de .then () invocações, não estamos registrando vários retornos de chamada em um único Objeto de promessa. Em vez disso, cada invocação do método então () Retorna um novo objeto de promessa. Esse novo objeto de promessa não é cumprido Até que a função passada para então () esteja concluída. Vamos retornar a uma forma simplificada da cadeia Fetch () original acima. Se definirmos as funções passadas para as invocações então () Em outros lugares, podemos refatorar o código para ficar assim: buscar (theurl) // tarefa 1; Retorna a promessa 1. THEN (chamada de chamada1) // Tarefa 2; Retorna Promise 2. THEN (chamado de retorno2); // Tarefa 3; Retorna Promessa 3 Vamos percorrer esse código em detalhes:

1. Na primeira linha, Fetch () é invocado com um URL. IniciaUm HTTP recebe solicitação para esse URL e retorna uma promessa. Vamos chamar isso de solicitação http de "Tarefa 1" e chamaremos oPromise "Promise 1". 2. Na segunda linha, invocamos o método então () dePromise 1, passando a função de retorno de chamada1 que queremos ser chamado quando a promessa 1 for cumprida. O método então () Armazena nosso retorno de chamada em algum lugar e depois retorna uma nova promessa. Chamaremos a nova promessa retornada nesta etapa "Promise 2", E diremos que a "Tarefa 2" começa quando o retorno de chamada1 éinvocado. 3. Na terceira linha, invocamos o método de promessa então ()2, passando a função de retorno2 que queremos invocar quandoA promessa 2 é cumprida. Este método então () se lembra do nossoretorno de chamada e retorna mais uma promessa. Vamos dizer essa "tarefa3 ?começa quando o retorno de chamada2 é invocado. Nós podemos chamar issoÚltima promessa "Promise 3", mas realmente não precisamos de um nomePor isso, porque não o usaremos. 4. As três etapas anteriores acontecem de forma síncrona quando oA expressão é executada pela primeira vez. Agora temos um assíncronopausa enquanto a solicitação HTTP iniciada na etapa 1 é enviadana Internet. 5. Eventualmente, a resposta HTTP começa a chegar. Oparte assíncrona da chamada fetch () envolve o httpStatus e cabeçalhos em um objeto de resposta e cumpre a promessa 1com esse objeto de resposta como o valor. 6. Quando a promessa 1 é cumprida, seu valor (o objeto de resposta) éPassado para a nossa função de retorno de chamada1 () e a tarefa 2 começa. O trabalho desta tarefa, dado um objeto de resposta como entrada, é obterO corpo de resposta como um objeto JSON. 7. Vamos supor que a Tarefa 2 complete normalmente e é capaz de

analisar o corpo da resposta HTTP para produzir um JSONObjeto. Este objeto JSON é usado para cumprir a promessa 2.8. O valor que cumpre a promessa 2 se torna a entrada da tarefa 3. Quando é passado para a função de retorno2(). Este terceiro Tarefa agora exibe os dados para o usuário em alguns não especificados caminho. Quando a Tarefa 3 está concluída (assumindo que ela completa normalmente), então a promessa 3 será cumprida. Mas porque nós nunca fez nada com a promessa 3, nada acontece quando issoPromessa se estabelece e a cadeia de computação assíncrona termina neste ponto.

13.2.3 Resolvendo promessas

Enquanto explica a cadeia de promessas que buscam o URL com a lista no Última seção, conversamos sobre as promessas 1, 2 e 3. Mas existe na verdade um objeto de quarta promessa também envolvido, e isso nos leva ao nosso Discussão importante sobre o que isso significa para uma promessa ser "resolvida". Lembre -se de que Fetch() retorna um objeto de promessa que, quando cumprido, passa um objeto de resposta para a função de retorno de chamada que registrarmos. Este objeto de resposta possui .text(), .json() e outros métodos para solicite o corpo da resposta HTTP de várias formas. Mas desde o corpo pode ainda não ter chegado, esses métodos devem devolver a promessa objetos. No exemplo, estudamos, "Tarefa 2" chama o.json() Método e retorna seu valor. Esta é a quarta promessa objeto, e é o valor de retorno da função de chamada 1(). Vamos reescrever o código que busca o URL mais uma vez em um verboso e maneira não -idiomática que torna os retornos de chamada e promete explícitos:

```
função c1 (resposta) { // retorno de chamada 1
```

Seja P4 = Response.json ();retornar P4;// Retorna Promise 4}função c2 (perfil) {// retorno de chamada
2displayUserProfile (perfil);}Seja p1 = buscar ("/api/usuário/perfil");// Promise 1, Tarefa 1Seja P2 = P1.THEN (C1);//
Promise 2, Tarefa 2Seja p3 = p2.then (c2);// Promise 3, Tarefa 3Para que as cadeias de promessas funcionem de
maneira útil, a saída da Tarefa 2 deveTorne -se a entrada para a Tarefa 3. E no exemplo que estamos
considerando aqui,A entrada para a Tarefa 3 é o corpo do URL que foi buscado, analisado como umObjeto
json.Mas, como acabamos de discutir, o valor de retorno do retorno de chamadaC1 não é um objeto JSON, mas
prometa P4 para esse objeto JSON.EsseParece uma contradição, mas não é: quando P1 é cumprido, C1
éInvocado, e a Tarefa 2 começa.E quando P2 é cumprido, C2 é invocado,e a tarefa 3 começa.Mas só porque a
Tarefa 2 começa quando C1 é invocado,Isso não significa que a Tarefa 2 deve terminar quando C1
retornar.Promessas sãosobre o gerenciamento de tarefas assíncronas, afinal, e se a Tarefa 2 forassíncrono (o que
é, neste caso), então essa tarefa não seráCompleto no momento em que o retorno de chamada retorna.Agora
estamos prontos para discutir os detalhes finais que você precisa paraEntenda para realmente dominar
promessas.Quando você passa um retorno de chamada C paraO método então (), então () retorna uma promessa
P e organizaInvocar de maneira assíncrona C em algum momento posterior.O retorno de chamada é
executadoAlguns computação e retorna um valor v. Quando o retorno de chamada retorna, Pé resolvido com o
valor v. Quando uma promessa é resolvida com um valor

Erro ao traduzir esta página.

Erro ao traduzir esta página.

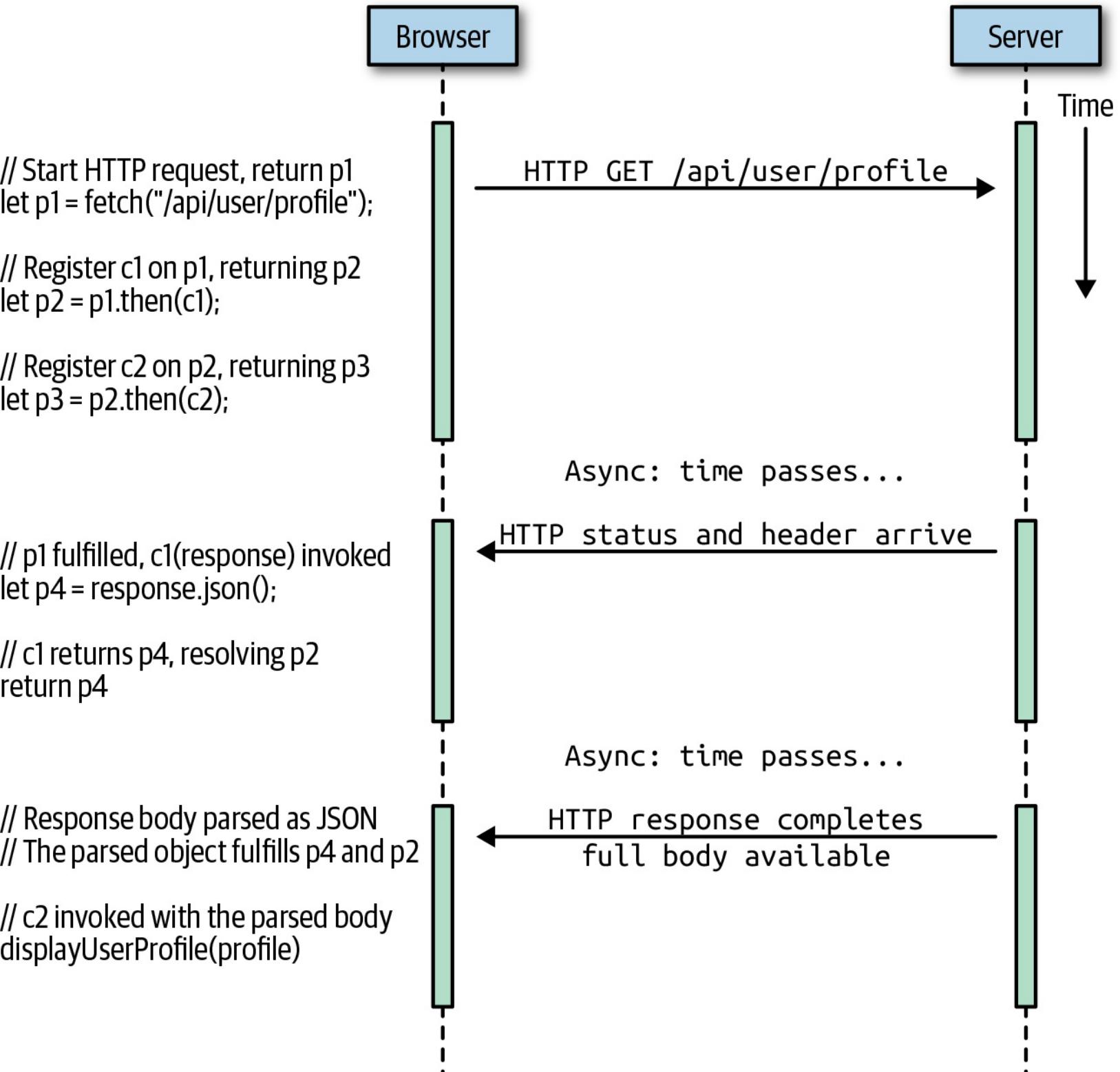


Figura 13-1.Buscando um URL com promessas

13.2.4 Mais sobre promessas e erros

No início do capítulo, vimos que você pode passar um segundo retorno de chamada função no método `.then()` e que esta segunda função será chamado se a promessa for rejeitada. Quando isso acontece, o argumento para Esta segunda função de retorno de chamada é um valor - normalmente um objeto de erro - Isso representa o motivo da rejeição. Também aprendemos que é incomum (e até unidiomático) para passar dois retornos de chamada para um `.then()` método. Em vez disso, erros relacionados à promessa são normalmente tratado adicionando um método `.catch()` invocação a uma promessa corrente. Agora que examinamos as correntes de promessa, podemos voltar para manipulação de erros e discutir isso com mais detalhes.

Para facilitar a discussão, Eu gostaria de enfatizar que o tratamento cuidadoso de erros é realmente importante quando fazendo programação assíncrona. Com código síncrono, se você Deixe de fora o código de manipulação de erros, você terá pelo menos uma exceção e um stack rastreio que você pode usar para descobrir o que está dando errado. Com Código assíncrono, exceções não tratadas geralmente não são relatadas, E os erros podem ocorrer silenciosamente, tornando -os muito mais difíceis de depurar. A boa notícia é que o método `.catch()` facilita o manuseio de erros ao trabalhar com promessas. A captura e finalmente métodos O método `.catch()` de uma promessa é simplesmente uma maneira de ligar `.then()` com nulo como o primeiro argumento e um manipulação de erros retorno de chamada como o segundo argumento. Dada qualquer promessa P e um retorno de chamada C, as duas linhas a seguir são equivalentes:

p. then (null, c);p.catch (c);A abreviação .Catch () é preferida porque é mais simples ePorque o nome corresponde à cláusula de captura em uma tentativa/capturadedeclaração de manipulação de exceção.Como discutimos, exceções normaisNão trabalhe com código assíncrono.O método .catch () deAs promessas são uma alternativa que funciona para o código assíncrono.Quandoalgo dá errado em código síncrono, podemos falar de umExceção ?borbulhando a pilha de chamadas? até encontrar um bloco de captura.Com uma cadeia de promessas assíncronas, a metáfora comparávelpode ser um erro "escorrendo pela corrente" até encontrar um.Catch () Invocation.No ES2018, os objetos de promessa também definem um método .Finalmente ()cujo objetivo é semelhante à cláusula finalmenteExperimente/Catch/finalmente declaração.Se você adicionar um .Finalmente ()invocação para sua cadeia de promessas, então o retorno de chamada que você passa para.Finalmente () será invocado quando a promessa que você o chame estabelece.Seu retorno de chamada será chamado se a promessa cumprir ou rejeitar,e não será aprovado nenhum argumento, então você não pode descobrir secumpriu ou rejeitou.Mas se você precisar executar algum tipo de limpezacódigo (como fechar arquivos abertos ou conexões de rede) em ambos os casos,Um retorno de chamada. Finalmente () é a maneira ideal de fazer isso.Como .hen ()e .Catch (), .Finalmente () retorna um novo objeto de promessa.Oo valor de retorno de um retorno de chamada .Finalmente () é geralmente ignorado, e oPromessa devolvida por .Finalmente () normalmente resolverá ou rejeitará como mesmo valor que a promessa que .Finalmente () foi invocada

resolve ou rejeita com. Se um retorno de chamada. Finalmente () jogar um exceção, no entanto, então a promessa retornada por .Finalmente () irá rejeitar com esse valor.O código de busca de URL que estudamos nas seções anteriores não Faça qualquer manuseio de erros.Vamos corrigir isso agora com um mais realista Versão do código:busca ("/api/user/perfil") // Inicie a solicitação HTTP.Then (resposta => {// Chame isso quando o status e Cabeçalhos estão prontosif (! Response.ok) {// se obtivemos um 404 não encontrado ou erro semelhante retornar nulo;// talvez o usuário esteja logado;Retorne perfil nulo}// agora verifique os cabeçalhos para garantir que o servidor nos enviou json// se não, nosso servidor está quebrado, e este é um Erro sério!deixe tipo = resposta.Headers.get ("conteúdo-tipo");if (type! == "Application/json") {jogue novo TypeError (` esperado JSON, conseguiu\$ {type} `);}// Se chegarmos aqui, então temos um status 2xx e um JSON Content-Type// para que possamos retornar com confiança uma promessa para o resposta// corpo como um objeto JSON.REPORTE DE REPORTE.JSON ();}.then (perfil => {// chamado com o analisado corpo de resposta ou nuloif (perfil){displayUserProfile (perfil);}}

else {// se obtivemos um erro 404 acima e devolvemosnulo nós acabamos aquidisplayLoggedoutProfilePage ()};}.catch (e => {if (e instanceof NetworkError) {// fetch () pode falhar dessa maneira se a internetA conexão está baixaDisplayErRormessage ("Verifique sua internetconexão.");}else if (e instanceof typeError) {// isso acontece se jogarmos TypeError acimaDisplayErRormessage ("Algo está errado com nossoservidor! ");}outro {// Isso deve ser algum tipo de erro imprevistoconsole.error (e);}});Vamos analisar esse código olhando o que acontece quando as coisas vãoerrado.Usaremos o esquema de nomenclatura que usamos antes: p1 é oPromessa devolvida pela chamada fetch ().P2 é a promessa devolvida porA primeira chamada .hen () e C1 é o retorno de chamada que passamos para isso.then () ligue.p3 é a promessa devolvida pelo segundo.Ligue e C2 é o retorno de chamada que passamos para essa chamada.Finalmente, C3 é oRetorno que passamos para a chamada .catch ().(Essa chamada retorna umPrometa, mas não precisamos nos referir a ele pelo nome.)A primeira coisa que poderia falhar é a solicitação de busca ().Se oA conexão de rede está baixa (ou por algum outro motivo um https solicitação não pode ser feita), então a promessa de P1 será rejeitada com um

Erro ao traduzir esta página.

O cabeçalho está errado, trata isso como um problema não recuperável e joga um TypeError. Quando um retorno de chamada passou para .then () (ou .catch ()) joga um valor, a promessa que foi o valor de retorno do. então () A chamada é rejeitada com esse valor jogado. Nesse caso, o código em C1 que aumenta um TypeError faz com que P2 seja rejeitado com esse objeto TypeError. Como não especificamos um manipulador de erros para P2, P3 será rejeitado com o bem. C2 não será chamado, e o TypeError será passado para C3, que tem código para verificar e lidar explicitamente esse tipo de erro. Vale a pena notar algumas coisas sobre esse código. Primeiro, observe que o objeto de erro jogado com um arremesso síncrono regular. A declaração acaba sendo tratada de forma assíncrona com um .Catch () Invocação de método em uma cadeia de promessas. Isso deve deixar claro o porquê. Este método abreviado é preferido em passar um segundo argumento para .Then (), e também por que é tão idiomático terminar as correntes de promessa com um .catch () ligue. Antes de deixarmos o tópico de manuseio de erros, quero ressaltar que, Embora seja idiomático encerrar todas as cadeias de promessas com um .Catch () Para limpar (ou pelo menos registrar) quaisquer erros que ocorreram na cadeia, é Também perfeitamente válido para usar .catch () em outros lugares em uma cadeia de promessas. Se um dos estágios da sua cadeia de promessa pode falhar com um erro e se o erro é algum tipo de erro recuperável que não deve parar o resto de A corrente de corrida, então você pode inserir uma chamada .CATCH () no cadeia, resultando em código que pode ser assim: startAsyncOperation (). então (Dostagethwo).catch (recuperação de estagiário)

. então (Dostagethree). então (DostageFour).catch (LogStagethreeAndFourErrors);Lembre -se de que o retorno de chamada que você passa para .catch () será apenasInvocado se o retorno de chamada em um estágio anterior lançar um erro.Se oRetorno de chamada retorna normalmente, então o retorno de chamada .catch () será pulados, e o valor de retorno do retorno de chamada anterior se tornará oEntrada para o próximo retorno de chamada .Then ().Lembre -se também disso .Catch ()Os retornos de chamada não são apenas para relatar erros, mas para manuseio e recuperando de erros.Uma vez que um erro foi passado para um .catch ()Retorno de chamada, ele para de propagar a cadeia de promessas.A .Catch ()Retorno de chamada pode causar um novo erro, mas se ele retornar normalmente, do que issoO valor de retorno é usado para resolver e/ou cumprir a promessa associada, eO erro para de se propagar.Vamos ser concretos sobre isso: no exemplo do código anterior, se qualquer umstartasyncoperation () ou DostageTwo () lança um erro,então a função recuperada de fromstageTwoError () seráinvocado.Se recuperar de questagethreeError () retornar normalmente,então seu valor de retorno será passado para Dostagethree () e oA operação assíncrona continua normalmente.Por outro lado, sere recuperado de estagiário () não conseguiu se recuperar.Por si só, eleva um erro (ou ele repete o erro que ele foi aprovado).EmEste caso, nem Dostagethree () nem DostageFour () serãoinvocado, e o erro lançado porrecuperado de estagiário () seria passado paraLOGSTAGETHREEANDFOURERRORS ().

Às vezes, em ambientes de rede complexos, os erros podem ocorrer mais ou menos aleatoriamente, e pode ser apropriado lidar com esses erros por simplesmente repetindo a solicitação assíncrona. Imagine que você escreveu um operação baseada em promessa para consultar um banco de dados: queryDatabase().
THEN (DisplayTable).catch(displayDatabaseError); Agora, suponha que problemas transitórios de carga de rede estejam causando falhas cerca de 1% do tempo. Uma solução simples pode ser tentar novamente a consulta com uma chamada .catch():queryDatabase().catch(e => wait(500).then(queryDatabase)) // em fracasso, espere e tente novamente.
THEN (DisplayTable).catch(displayDatabaseError); Se as falhas hipotéticas forem verdadeiramente aleatórias, adicionando esta linha ao código deve reduzir sua taxa de erro de 1% para 0,01%. Retornando de um retorno de chamada de promessa Vamos voltar uma última vez ao exemplo anterior do URL e considerar o retorno de chamada C1 que nós passou para a primeira invocação .then(). Observe que existem três maneiras pelas quais C1 pode rescindir. Pode retornar normalmente com a promessa devolvida pela chamada .json(). Isso faz com que P2 seja resolvido, mas se essa promessa é cumprida ou rejeitada depende do que acontece com os recém-devolvidos Promessa. C1 também pode retornar normalmente com o valor nulo, o que faz com que P2 seja cumprido imediatamente. Finalmente, o C1 pode terminar lançando um erro, o que faz com que o P2 seja rejeitado. Esses são os três resultados possíveis para uma promessa, e o código em C1 demonstra como o retorno de chamada pode causar cada resultado. Em uma cadeia de promessas, o valor retornou (ou jogado) em um estágio da cadeia se torna a entrada para o próximo estágio da cadeia, por isso é fundamental acertar isso. Na prática, esquecendo de retornar um valor de uma função de retorno de chamada é na verdade uma fonte comum de bugs relacionados à promessa, e isso é exacerbado por sintaxe de atalho da função de seta do JavaScript. Considere esta linha de código que vimos anteriormente:

.catch (e => espera (500) .THEN (QueryDatabase))Lembre -se do capítulo 8 de que as funções de seta permitem muitos atalhos.Já que existe exatamente um argumento (o valor do erro), podemos omitir os parênteses.Já que o corpo da função é um único expressão, podemos omitir os aparelhos encaracolados ao redor do corpo da função e o valor da expressão torna -se o valor de retorno da função.Devido a esses atalhos, o código anterior está correto.Mas considere essa mudança inócuă:.catch (e => {wait (500) .then (querydatabase)})Ao adicionar os aparelhos encaracolados, não obtemos mais o retorno automático.Esta função agora retorna indefinido em vez de devolver uma promessa, o que significa que a próxima etapa nesta cadeia de promessas irá ser invocado com indefinido como entrada e não como resultado da consulta em novo.É um erro sutil isso pode não ser fácil de depurar.13.2.5 promessas em paraleloPassamos muito tempo conversando sobre cadeias de promessas por sequencialmente executando as etapas assíncronas de uma operação assíncrona maior.Às vezes, porém, queremos executar uma série de assíncronas operações em paralelo.A função promessa.all () pode fazer isso.Promete.all () leva uma variedade de objetos de promessa como sua entrada e retorna uma promessa.A promessa devolvida será rejeitada se algum dos as promessas de entrada são rejeitadas.Caso contrário, será cumprido com uma matriz dos valores de atendimento de cada uma das promessas de entrada.Então, por exemplo,Se você deseja buscar o conteúdo de texto de vários URLs, você pode usar o código como este:// começamos com uma variedade de URLsconst urls = [/* zero ou mais URLs aqui */]; // e converta -o em uma variedade de objetos de promessas promessas = urls.map (url => busca (url) .then (r => r.text ())); // agora tem uma promessa de executar todas essas promessas em paraleloPromise.All (promessas).Then (corpos => { /* Faça algo com a matriz de strings */ })

.catch (e => console.error (e));Promise.All () é um pouco mais flexível do que o descrito anterior.OA matriz de entrada pode conter os objetos prometidos e os valores de não promessa.Se um elemento da matriz não é uma promessa, é tratado como se fosse o valor de uma promessa já cumprida e é simplesmente copiado na matriz de saída.A promessa devolvida por promise.all () rejeita quando qualquer um dosAs promessas de entrada são rejeitadas.Isso acontece imediatamente no primeiro rejeição e pode acontecer enquanto outras promessas de entrada ainda estão pendentes.No ES2020, Promise.AllSettled () recebe uma variedade de entradaPromete e retorna uma promessa, assim como a promise.All () faz.MasPromete.AllSettled () nunca rejeita a promessa devolvida, e ela não cumpre essa promessa até que todas as promessas de entrada se estabeleçam.A promessa resolve uma variedade de objetos, com um objeto para cada promessa de entrada.Cada um desses objetos retornados tem uma propriedade de status definido como "cumprido" ou "rejeitado".Se o status for "cumprido", então o objeto também terá uma propriedade de valor que fornece o valor de atendimento.ESe o status for "rejeitado", o objeto também terá um motivo propriedade que fornece o erro ou o valor de rejeição do correspondentePromise:Promise.AllSetTled ([Promise.Resolve (1), Promise.Reject (2,3)]. Então (resultados => {Resultados [0] // => {status: "cumprido", valor: 1}Resultados [1] // => {status: "rejeitado", razão: 2}Resultados [2] // => {status: "cumprido", valor: 3}});

Ocasionalmente, você pode querer executar várias promessas de uma só vez, mas pode se preocupar apenas com o valor do primeiro a cumprir. Nesse caso, você pode usar o `Promise.race()` em vez de `Promise.all()`. Isso retorna uma promessa que é cumprida ou rejeitada quando o primeiro dosAs promessas na matriz de entrada são cumpridas ou rejeitadas. (Ou, se houver algum valor não promessa na matriz de entrada, ele simplesmente retorna o primeiro de aqueles.)

13.2.6 Fazendo promessas

Usamos a função de retorno da promessa () em muitos dos exemplos anteriores porque é uma das funções mais simples incorporadas para Navegadores da Web que retornam uma promessa. Nossa discussão sobre promessas temTambém se baseou em funções hipotéticas de remuneração de promessas `getJSON()` e `espera()`. Funções escritas para devolver promessas realmente são bastante úteis, e esta seção mostra como você pode criar sua própria promessa-APIs baseadas. Em particular, mostraremos implementações de `getJSON()` e `espera()`. Promessas baseadas em outras promessas É fácil escrever uma função que retorne uma promessa se você tiver alguma outra função de retorno de promessa para começar. Dada uma promessa, você sempre pode criar (e retornar) um novo chamando `.then()`. Então, se usamos a função `Fetch()` existente como um ponto de partida, podemos escrever `getJSON(url)` assim:

```
function getJSON(url) {  
  return fetch(url).then(response => response.json());  
}
```

O código é trivial porque o objeto de resposta da API busca () possui um método JSON () predefinido. O método json () retorna umPromessa, que retornamos do nosso retorno de chamada (o retorno de chamada é uma flecha função com um corpo de expressão única, portanto o retorno está implícito), de modo que oPromessa devolvida por getjson () resolve a promessa devolvida porResponse.json (). Quando essa promessa cumpre, a promessareturnado por getjson () atende ao mesmo valor. Observe que háNão há tratamento de erro nesta implementação getjson (). Em vez deverificando a resposta.OK e o cabeçalho do tipo de conteúdo, em vez dissoBasta permitir que o método json () rejeite a promessa que retornou com umSyntaxError se o corpo de resposta não puder ser analisado como JSON. Vamos escrever outra função de retorno de promessa, desta vez usandogetjson () como fonte da promessa inicial:função gethighscore () {retornar getjson ("/api/user/perfil"). Então (perfil => perfil.highscore);} Estamos assumindo que essa função faz parte de algum tipo de Web baseada na Webjogo e que o URL ?/API/User/Perfil? retorna um JSON formatadoEstrutura de dados que inclui uma propriedade Highscore. Promessas baseadas em valores síncronosÀs vezes, pode ser necessário implementar um existente baseado em promessaAPI e devolver uma promessa de uma função, mesmo que oA computação a ser realizada não exige nenhumaoperações assíncronas. Nesse caso, os métodos estáticos

Promey.Resolve () e Promise.Reject () farão o que você quer.Promete.resolve () assume um valor como seu único argumento e retorna uma promessa que será imediatamente (mas assíncrono) ser cumprido a esse valor.De forma similar,Promete.reject () pega um único argumento e retorna uma promessa isso será rejeitado com esse valor como o motivo.(Para deixar claro: as promessas devolvidas por esses métodos estáticas ainda não são cumpridas ou rejeitadas quando forem devolvidos, mas eles cumprirão ou rejeitarão imediatamente após a atual parte síncrona de código terminar ou correr.)Lembre -se do §13.2.3 de que uma promessa resolvida não é a mesma coisa que uma promessa cumprida.Quando chamamos de promete.resolve (), normalmente passamos o valor do cumprimento para criar um objeto de promessa que em breve cumprir com esse valor.O método não é nomeado Promete.fulfill (), no entanto.Se você passar uma promessa P1 para Promete.Resolve (), ele retornará uma nova promessa P2, que é imediatamente resolvida, mas que não será cumprido ou rejeitado até P1 é cumprido ou rejeitado.É possível, mas incomum, escrever uma função baseada em promessa onde o valor é calculado de forma síncrona e devolvida de forma assíncrona com Promete.Resolve ().É bastante comum, no entanto, ter casos especiais síncronos dentro de uma função assíncrona, e você pode lidar com esses casos especiais com promete.resolve () e Promete.reject ().Em particular, se você detectar condições de erro (como valores de argumento ruim) antes de começar um assíncrono

Operação, você pode relatar esse erro retornando uma promessa criada comPromise.reject ().(Você também pode simplesmente jogar um erro síncrono nesse caso, mas isso é considerado uma forma ruim porqueEntão o chamador da sua função precisa escrever um síncronoCláusula de captura e use um método .Catch () assíncrono para lidar erros.) Finalmente, Promise.Resolve () às vezes é útil para criarA promessa inicial em uma cadeia de promessas.Vamos ver algunsExemplos que o usam dessa maneira.Promessas do zeroPara getjson () e gethighscore (), começamos chamando uma função existente para obter uma promessa inicial e criada e deve lhe trazer uma nova promessa chamando o método .then () dessa inicialPromessa.Mas que tal escrever uma função de retorno de promessa quando Você não pode usar outra função de retorno de promessa como o ponto de partida?Nesse caso, você usa o construtor Promise () para criar um novoPromise objeto que você tem controle completo.Aqui está como é Trabalhos: você invoca o construtor Promise () e passa uma função como seu único argumento.A função que você passa deve ser escrita para esperarDois parâmetros, que, por convenção, devem ser nomeados resolvendo ou rejeitar.O construtor chama de síncrona sua função comArgumentos de função para os parâmetros de resolução e rejeição.Depois Chamando sua função, o construtor Promise () retorna o recémPromessa criada.Essa promessa retornada está sob o controle do função que você passou para o construtor.Essa função deve desempenhar alguma operação assíncrona e depois chama a função de resolução para resolver ou cumprir a promessa devolvida ou chamar a função de rejeição para

rejeitar a promessa devolvida. Sua função não precisa ser assíncrono: pode chamar de resolver ou rejeitar de forma síncrona, mas o A promessa ainda será resolvida, cumprida ou rejeitada de forma assíncrona se você faz isso. Pode ser difícil entender as funções passadas para uma função passada para um construtor apenas lendo sobre isso, mas espero que alguns exemplos vai deixar isso claro. Veja como escrever a espera baseada em promessa () função que usamos em vários exemplos anteriormente no capítulo: função espera (duração) { // Crie e retorne uma nova promessa retornar nova promessa ((resolver, rejeitar) => { // estesControle a promessa// Se o argumento for inválido, rejeite a promessa if (duração <0) { rejeitar (novo erro ("Viagem no tempo ainda não implementado ")); } // Caso contrário, espere de forma assíncrona e resolva a promessa. // setTimeout invocará resolve () sem argumentos, o que significa// que a promessa cumprirá com os indefinidos valor.setTimeout (resolve, duração); }) } Observe que o par de funções que você usa para controlar o destino de umPromessa criada com o construtor Promise () é nomeado resolve () e rejeit (), não cumpre () e rejeitar (). Se Você passa uma promessa de resolver (), a promessa devolvida será resolvida para essa nova promessa. Muitas vezes, no entanto, você passará a uma não promessa

valor, que cumpre a promessa devolvida com esse valor. Exemplo 13-1 é outro exemplo de uso da promessa ()construtor. Este implementa nossa função getjson () para uso emNó, onde a API Fetch () não é incorporada. Lembre -se de que nósIniciou este capítulo com uma discussão sobre retornos de chamada assíncronos eeventos. Este exemplo usa retornos de chamada e manipuladores de eventos e é umboa demonstração, portanto, de como podemos implementar a promessaAPIs baseadas no topo de outros estilos de programação assíncrona. Exemplo 13-1. Uma função assíncrona getjson ()const http = requer ("http");função getjson (url) { // Crie e retorne uma nova promessareturnar nova promessa ((resolver, rejeitar) => { // Inicie um http get solicitação para o URL especificadosolicitação = http.get (url, resposta => { // chamado quandoA resposta começa// rejeitar a promessa se o status HTTP estiver erradoif (Response.statuscode! == 200) {rejeite (novo erro ('status http\$ {Response.statuscode} `));Response.Resume ();// então não vazamos memória}// e rejeite se os cabeçalhos de resposta estiverem erradoscaso contrário, if (Response.Headers ["Content-Type"]! == "Application/json") {rejeitar (novo erro ("tipo de conteúdo inválido"));Response.Resume ();// Não vaze memória}outro {// Caso contrário, registre eventos para ler o corpoda respostaDeixe Body = "";Response.setEncoding ("UTF-8");

resposta.on ("dados", chunk => {body += chunk;});Response.on ("end", () => {// Quando o corpo de resposta estiver completo, tentar analisá-lo. Seja analisado = json.parse (corpo); // se for divulgado com sucesso, cumprir a promise resolver (analisado);} catch (e) {// Se a análise falhou, rejeite a promise rejeitar (e);});});});// também rejeitamos a promise se a solicitação falhar de nós // até obtenha uma resposta (como quando a rede é abaixo)request.on ("erro", erro => {rejeitar (erro);});}13.2.7 Promessas em sequênciaPromey.all () facilita a execução de um número arbitrário de promessas em paralelo. As cadeias de promessas facilitam o expressar um Sequência de um número fixo de promessas. Executando um número arbitrário de promessas em sequência é mais complicado, no entanto. Suponha, por exemplo, que você tem uma variedade de URLs para buscar, mas isso para evitar sobrecarga sua rede, você deseja buscá-los um de cada vez. Se a matriz for decomprimento arbitrário e conteúdo desconhecido, você não pode escrever uma promise

cadeia com antecedência, então você precisa construir um dinamicamente, com código como esse:

```
Função busca
(URLs) {//
  Vamos armazenar os corpos de URL aqui enquanto os buscamos
  const corpos = [] // Aqui está uma
  função de denúncia de promessa que busca um corpo
  const fetchOne = url => {
    Retornar Fetch (URL).Then
    (Response => Response.Text ())
    .Then (Body => {
      // salvamos o corpo na matriz, e somos propostamente
      // omitindo
      // um valor de retorno aqui (retornando indefinido)
      corpos.push (corpo);
    });
  }
  // Comece com uma promessa que cumprirá
  // imediatamente (com valor indefinido)
  Seja p = Promise.Resolve (indefinido);
  // agora percorre os URLs desejados,
  // construindo uma promessa corrente
  // de comprimento arbitrário, buscando um URL em cada estágio de
  // execução para (URL de URLs)
  p = p.then (() => fetchOne (url));
  // Quando a última promessa nessa cadeia é
  // cumprida, então o array de corpos está pronto.
  Então, vamos devolver uma promessa para que
  // matriz de
  // corpos. Observe que não incluímos nenhum erro Manipuladores:
  // queremos permitir que erros se propagem para o
  // chamador.
  retornar P.Then (() => corpos);}
```

Com esta função fetchSequencialmente () definida, poderíamos buscaros URLs um de cada vez com código como o código de busca em paraleloUsamos anteriormente para demonstrar Promise.all ():FetchSequencialmente (URLs).Then (corpos => { /* Faça algo com a matriz deStrings */}).catch (e => console.error (e));A função FetchSequencialmente () começa criando uma promessalssso cumprirá imediatamente após o retorno.Em seguida, constrói um longo e linearPromessa se encaixa nessa promessa inicial e retorna a última promessa ema corrente.É como montar uma fileira de dominó e depois bater oprimeiro um.Há outra abordagem (possivelmente mais elegante) que podemos adotar.Em vez de criar as promessas com antecedência, podemos ter o retorno de chamadaPara cada promessa, crie e retorne a próxima promessa.Isto é, em vez deCriando e encadeando várias promessas, criamos promessasIsso resolve outras promessas.Em vez de criar um dominóCadeia de promessas, estamos criando uma sequênciade promessasaninhado um dentro do outro como um conjunto de bonecas Matryoshka.Com issoabordagem, nosso código pode retornar a primeira promessa (mais externa), sabendoque acabará cumprindo (ou rejeitará!) Para o mesmo valor que o último(Interior) promessa na sequênci.OPromisessequence () A função a seguir é escrita para ser genérica e não é específico para buscar o URL.Esta aquí no final do nossoDiscussão das promessas porque é complicada.Se você leu issoCapítulo com cuidado, no entanto, espero que você consiga entender como éfunciona.Em particular, observe que a função aninhada dentro

Promisessequence () parece se chamar recursivamente, mas porqueA chamada "recursiva" é através de um método então (), na verdade não háQualquer recursão tradicional acontecendo:// Esta função leva uma matriz de valores de entrada e umFunção "Promisemaker".// Para qualquer valor de entrada x na matriz, o promissor (x) deve devolver uma promessa// que atenderá a um valor de saída.Esta função retorna uma promessa// que atende a uma matriz dos valores de saída computados./// em vez de criar as promessas de uma só vez e deixá-los correm// Paralelo, no entanto, o promission () executa apenas uma promessa de cada vez// e não chama o promissor () para obter um valor até a promessa anterior// cumpriu.Função Promisessequence (Entradas, Promisemaker) { // Faça uma cópia privada da matriz que podemos modificarEntrada = [... entradas]; // Aqui está a função que usaremos como uma promessa ligar de volta// Esta é a magia pseudorecursiva que faz com que tudo trabalhar.function handleNextInput (saídas) {if (inputs.length === 0) { // Se não houver mais entradas, retorne a matriz// de saídas, finalmente cumprindo esta promessa e tudo// Promessas resolvidas anteriores resolvidas, mas não cumpridas.retornar saídas;} outro { // Se ainda houver valores de entrada para processar,Então nós vamos// devolver um objeto de promessa, resolvendo a correntePromessa// com o valor futuro de uma nova promessa.deixe nextInput = inputs.shift ()// Obtenha o próximo valor de entrada,

Return Promesemaker (NextInput) // Calcule oPróximo valor de saída, // então crie uma nova matriz de saídas com
onovo valor de saída.THEN (output => outputs.CONCAT (saída))// então "Recurse", passando o novo, mais
tempo, matriz de saídas.Then (handlenextInput);})// Comece com uma promessa que cumpre uma matriz vazia e
uso// A função acima como seu retorno de chamada.return promey.resolve ([]). Então (handlenextInput);} Esta
função promisesequence () é intencionalmente genérica. Nós podemos usá-lo para buscar URLs com código como
este:// Dado um URL, retorne uma promessa que cumpre o URL texto corporal função fetchbody (url) {return fetch
(url) .hen (r =>r.Text ());}// Use -o para buscar sequencialmente um monte de corpos de URLPromisesequence
(URLs, Fetchbody).Then (corpos => { /* Faça algo com a matriz deStrings */}).catch (console.error); 13.3 assíncrono
e aguardar O ES2017 apresenta duas novas palavras -chave - Async and AguardRepresentar uma mudança de
paradigma na programação JavaScript assíncrona. Essas novas palavras -chave simplificam drasticamente o uso
de promessas e Permita-nos escrever um código assíncrono baseado em promessa que se parece com código
síncrono que bloqueia enquanto aguarda respostas de rede ou

Outros eventos assíncronos. Embora ainda seja importante entender como as promessas funcionam, grande parte de sua complexidade (e às vezes até sua própria presença!) desaparece quando você os usa com assíncrona e aguarde. Como discutimos anteriormente no capítulo, o código assíncrono não pode retornar um valor ou jogar uma exceção da maneira como o código síncrono regular pode. É por isso que as promessas são projetadas da maneira que são. O valor de uma promessa cumprida é como o valor de retorno de uma função síncrona. E o valor de uma promessa rejeitada é como um valor jogado por uma função síncrona. Esta última similaridade é explicitada pelo nomeação do método .Catch().assíncrono e aguardado Tomar eficiente. Código baseado em promessa e oculte as promessas para que você seja assíncrono. O código pode ser tão fácil de ler e tão fácil de raciocinar quanto ineficiente. Código síncrono de bloqueio.

13.3.1 Aguardar expressões

A palavra -chave aguardar pega uma promessa e a transforma de volta em um retorno de valor ou uma exceção jogada. Dado um objeto de promessa P, a expressão Aguarda P espera até P se estabelecer. Se p cumpre, então o valor de aguardar P é o valor de atendimento de p. Por outro lado, se p for rejeitado, então a expressão aguarda p joga o valor de rejeição da p. Nós não Geralmente o uso aguarda com uma variável que mantém uma promessa; Em vez disso, nós Use -o antes da invocação de uma função que retorna uma promessa: deixe a resposta = aguarda buscar (" /api / usuário / perfil "); Deixe perfil = aguarde Response.json();

É fundamental entender imediatamente que a palavra -chave aguardar nãofazer com que seu programa bloquee e literalmente não faça nada até que o especificadoPromessa se acalma.O código permanece assíncrono, e o aguardarSimplesmente disfarça esse fato.Isso significa que qualquer código que use aguarda éele próprio assíncrono.13.3.2 Funções assíncronasPorque qualquer código que usa aguarda é assíncrono, há umRegra crítica: você só pode usar a palavra -chave aguardar em funções queforam declarados com a palavra -chave assíncrona.Aqui está uma versão doGethighScore () função do início do capítulo, reescrito paraUse assíncrono e aguarde:função assíncrona gethighscore () {deixe a resposta = aguarda buscar ("/api/usuário/perfil");Deixe perfil = aguarde Response.json ();Return perfil.highscore;}Declarar uma função assíncrona significa que o valor de retorno da funçãoserá uma promessa, mesmo que nenhum código relacionado à promessa apareça no corpoda função.Se uma função assíncrona parece retornar normalmente, entãoO objeto de promessa que é o valor de retorno real da função seráResolva para esse aparente valor de retorno.E se uma função assíncronaparece fazer uma exceção, então o objeto de promessa que ele retornaserá rejeitado com essa exceção.A função GethighScore () é declarada assíncrona, por isso retorna umPromessa.E porque retorna uma promessa, podemos usar o aguardar

Erro ao traduzir esta página.

Seja valor2 = aguarda getjson (url2);O problema com este código é que ele é desnecessariamente seqüencial: oA busca do segundo URL não começará até que a primeira busca seja concluída.Se o segundo URL não depender do valor obtido do primeiroURL, provavelmente devemos tentar buscar os dois valores no mesmotempo.Este é um caso em que a natureza baseada em promessa de AsyncFunções mostra.Para aguardar um conjunto de execução simultaneamentefunções assíncronas, usamos o prometido.all () exatamente como faríamos seTrabalhando com promessas diretamente:Seja [Value1, Value2] = Aguarda Promise.all ([Getjson (URL1),getjson (url2)]);13.3.4 Detalhes da implementaçãoFinalmente, para entender como as funções assíncronas funcionam, pode ajudarPensar no que está acontecendo sob o capô.Suponha que você escreva uma função assíncrona como esta:função assíncrona f (x) { / * body * /}Você pode pensar nisso como uma função de retorno de promessa embrulhadaAo redor do corpo da sua função original:função f (x) {Retornar nova promessa (função (resolver, rejeitar) {tentar {resolve ((function (x) { / * body * /}) (x));}Catch (e) {rejeitar (e);

});}É mais difícil expressar a palavra -chave aguardar em termos de sintaxe transformação como esta. Mas pense na palavra -chave aguardar como um marcador que quebra um corpo de função em separado, sincronopedações.Um intérprete de ES2017 pode dividir o corpo da função em um Sequência de subfunções separadas, cada uma das quais é passada para o Então () Método da promessa aguardada que a precede.13.4 iteração assíncronaComeçamos este capítulo com uma discussão sobre retorno de chamada e eventos baseados em eventos assíncronos, e quando introduzimos promessas, observamos que elas foram úteis para cálculos assíncronos de tiro únicoAdequado para uso com fontes de eventos assíncronos repetitivos, como setInterval (), o evento "clique" em um navegador da web ou os "dados" evento em um fluxo de nós.Porque promessas únicas não funcionam para Sequências de eventos assíncronos, também não podemos usar assíncronos regulares funções e as declarações aguardadas para essas coisas.O ES2018 fornece uma solução, no entanto.Iteradores assíncronos são como iteradores descritos no capítulo 12, mas são baseados em promessa e devem ser usados ??com uma nova forma do loop for/of:para/aguardar.13.4.1 O loop for/waitO nº 12 torna seus fluxos legíveis iteráveis ??de forma assíncrona.Esse

significa que você pode ler pedaços sucessivos de dados de um fluxo com um para/aguarde loop como este:
const fs = require ("fs");
Função assíncrona parsefile (nome do arquivo) {
Deixe Stream = fs.createReadStream (nome do arquivo, {encoding: "UTF-8"});
para aguardar (Let Chunk of Stream) {parsechunk (pedaço); // Suponha que parsechunk () seja definido em outros lugares}}
Como uma expressão regular, o loop for/aguardar é promessa baseado. Grosso falando, o iterador assíncrono produz umPromessa e o loop for/aguardar aguardam essa promessa de cumprir, atribui o valor de atendimento à variável de loop e executa o corpo do loop. E então começa de novo, recebendo outra promessa do iterador e aguardando essa nova promessa de cumprir. Suponha que você tenha uma variedade de URLs:
const urls = [url1, url2, url3];
Você pode chamar Fetch () em cada URL para obter uma variedade de promessas:
promete const = urls.map (url => busca (url));
Vimos anteriormente no capítulo que agora poderíamos usarPromise.all () para esperar que todas as promessas da matriz sejam cumpridas. Mas suponha que queremos os resultados da primeira busca assim que eles ficam disponíveis e não queremos esperar que todos os URLs sejam

Erro ao traduzir esta página.

um que pode ser usado com um loop for/de. Define um método com o nome simbólico `Symbol.iterator`. Este método retorna um iterador objeto. O objeto iterador tem um método próximo () , que pode ser chamado repetidamente para obter os valores do objeto iterável. O próximo () O método do objeto iterador retorna objetos de resultado da iteração. O objeto de resultado da iteração possui uma propriedade de valor e/ou uma propriedade feita. Os iteradores assíncronos são bastante semelhantes aos iteradores regulares, mas há duas diferenças importantes. Primeiro, um objeto de maneira assíncrona implementa um método com o nome simbólico `Symbol.asyncIterator` em vez de símbolo `iterator`. (Como vimos anteriormente, pois/aguarda é compatível com iterável regular objetos, mas prefere objetos de maneira assíncrona e tenta o Método `Symbol.asyncIterator` antes de tentar o Método `Symbol.iterator`.) Segundo, o próximo () método de um iterador assíncrono retorna uma promessa que se resolve a um iterador objeto de resultado em vez de retornar diretamente um objeto de resultado do iterador. OBSERVAÇÃO Na seção anterior, quando usamos/aguardamos regularmente, de forma síncrona Matriz de promessas iteráveis, estávamos trabalhando com o resultado do iterador síncrono Objetos nos quais a propriedade Value era um objeto de promessa, mas a propriedade feita era síncrono. Os verdadeiros iteradores assíncronos retornam promessas para o resultado da iteração Objetos, e tanto o valor quanto as propriedades feitas são assíncronas. A diferença é sutil: com iteradores assíncronos, a escolha sobre quando Os terminais de iteração podem ser feitos de forma assíncrona.

13.4.3 geradores assíncronos

Como vimos no capítulo 12, a maneira mais fácil de implementar um iterador éfrequentemente para usar um gerador.O mesmo vale para iteradores assíncronos,que podemos implementar com as funções do gerador que declaramosassíncrono.Um gerador assíncrono tem as características das funções assíncronas e oRecursos de geradores: você pode usar aguardar como faria em um regularfunção assíncrona, e você pode usar o rendimento como faria em um regulargerador.Mas os valores que você produz são automaticamente envolvidosPromessas.Até a sintaxe para geradores assíncronos é uma combinação:Função e função assíncronas * combinam -se em assíncronasfunção *.Aqui está um exemplo que mostra como você pode usar umgerador assíncrono e um loop para/aguardar para executar o código repetidamenteintervalos corrigidos usando a sintaxe do loop em vez de um setInterval ()Função de retorno de chamada:// um invólucro baseado em promessa em torno do setTimeout () que podemosuse aguarda com:// retorna uma promessa que cumpre o número especificado demilissegundosFunção DastaDtime (ms) {return nova promessa (resolve => setTimeout (resolve, ms));}// uma função do gerador assíncrono que incrementa um contador eproduz isso// um número especificado (ou infinito)intervalo.função assíncrona* relógio (intervalo, max = infinito) {para (let count = 1; count <= max; count++) {// regularpara loopaguardar tempo de pasteld (intervalo);// Esperehora de passarcontagem de rendimentos;// produz ocontador}}

```
// uma função de teste que usa o gerador assíncrono compara/aguardarteste de função assíncrona () {}  
assíncropode usar para/aguardarpara aguardar (deixe o tick de relógio (300, 100)) {}// loop 100vezes a cada  
300msconsole.log (tick);}}13.4.4 Implementando iteradores assíncronosEm vez de usar geradores assíncronos  
para implementar iteradores assíncronos,Também é possível implementá -los diretamente, definindo um  
objetocom um método symbol.asynciterator () que retorna um objetocom um método próximo () que retorna uma  
promessa que resolve para umobjeto de resultado do iterador.No código a seguir, reimplementamos oclock ()  
função do exemplo anterior para que não seja umgerador e, em vez disso, apenas retorna um objeto de maneira  
assíncrona.Observe que o método próximo () neste exemplo não explicitamente devolver uma promessa;Em vez  
disso, apenas declaramos o próximo () ser assíncrono:relógio de função (intervalo, max = infinito) {}// Uma versão  
prometida do setTimeout que podemos usaraguarde com// Observe que isso leva um tempo absoluto em vez de  
umintervalo.função até (horário) {Retorne nova promessa (resolve => setTimeout (resolve,tempo - data.now ()));}//  
devolver um objeto de maneira assíncronareturnar {StartTime: date.now (), // lembre -se de quando  
começamosCONTA: 1, // Lembre -se de qual iteraçãoEstamos ligados
```

assíncrono next () {// o método next () fazEste é um iteradorif (this.count > max) // terminamos?return {done: true};// resultado de iteraçãoindicando feito}// descobrir quando a próxima iteração devecomeçar,Deixe TargetTime = this.startTime + this.count *interval;// Espere até aquele momento,aguarde até (TargetTime); // e retorne o valor da contagem em uma iteraçãooobjeto de resultado.return {value: this.count ++};};// Este método significa que este objeto de iterador étambém um iterável.[Symbol.asynciterator] () {return this;}};}Esta versão baseada em iterador da função clock () corrige uma falha ema versão baseada em gerador.Observe que, neste código mais recente, temos como alvoo tempo absoluto em que cada iteração deve começar e subtrair otempo atual disso para calcular o intervalo para o qual passamossetTimeout ().Se usarmos o relógio () com um loop for/aguardar, esteA versão será executada de iterações de loop mais precisamente no intervalo especificadoPorque é responsável pelo tempo necessário para realmente executar o corpo dolaço.Mas essa correção não é apenas uma precisão de tempo.O para/aguardarLoop sempre espera pela promessa devolvida por uma iteração para sercumprido antes de começar a próxima iteração.Mas se você usar umiterador assíncrono sem um loop para/aguardar, não há nada paraimpedir você de chamar o método próximo () sempre que quiser.Com a versão baseada em gerador de relógio (), se você ligar para o

Próximo () Método três vezes sequencialmente, você receberá três promessastudo isso vai cumprir quase exatamente ao mesmo tempo, o que provavelmente énão o que você quer.A versão baseada em iterador que implementamos aquinão tem esse problema.O benefício dos iteradores assíncronos é que eles nos permitem representarfluxos de eventos ou dados assíncronos.A função do relógio ()discutido anteriormente era bastante simples de escrever porque a fonte deA assincronia foi as chamadas setTimeout () que estávamos fazendo nós mesmos.Mas quando estamos tentando trabalhar com outros assíncronosfontes, como o desencadeamento dos manipuladores de eventos, torna -sesubstancialmente mais difícil de implementar iteradores assíncronos - normalmenteter uma única função de manipulador de eventos que responde aos eventos, mas cadaChamada para o método Next () do iterador deve retornar uma promessa distintaobjeto, e várias chamadas para a próxima () podem ocorrer antes do primeiroPromessa resolve.Isso significa que qualquer método de iterador assíncronodeve ser capaz de manter uma fila interna de promessas que ela resolveem ordem como eventos assíncronos estão ocorrendo.Se encapsularmos issoComportamento de prometo em uma aula de assíncrona, então se tornaMuito mais fácil escrever iteradores assíncronos com base no assíncrono.A aula de assíncrona a seguirMétodos dequeue () como você esperava para uma aula de filas.Oo método dequeue () retorna uma promessa em vez de um valor real,No entanto, o que significa que não há problema em chamar Dequeue () antesENQUEUE () já foi chamado.A classe Asyncqueue também é umiterador assíncrono e deve ser usado com um para/aguardarLoop cujo corpo corre uma vez cada vez que um novo valor é assíncrono3

preso.(Asyncqueue tem um método Close ()). Uma vez chamado, nãoMais valores podem ser envolvidos.Quando uma fila fechada está vazia, opara o loop/aguardar vai parar de fazer o loop.)Observe que a implementação do AsyncQueue não usa assíncrono ouAguardar e, em vez disso, trabalhe diretamente com promessas.O código éum pouco complicado, e você pode usá -lo para testar sua compreensão deO material que abordamos neste longo capítulo.Mesmo que você nãoentender a implementação do assíncico, dê uma olhada noExemplo mais curto que o segue: implementa um simples, mas muitolterador assíncrono interessante no topo do Asyncqueue.*** Uma classe de fila de maneira assíncrona.Adicione valores comenquistar ()* e remova -os com dequeue ().Dequeue () retorna aPromessa, que* significa que os valores podem ser desquedados antes de serempreso.O* a classe implementapode* ser usado com o loop for/aguardar (que não terá terminadoaté* O método Close () é chamado.)*/classe Asyncqueue {construtor () {// valores que foram na fila, mas ainda nãosão armazenados aquithis.values ??= [];// quando as promessas são desquedadas antes de seusOs valores correspondentes são// na fila, os métodos de resolução para essas promessas sãocomoarmazenado aqui.this.resolvers = [];// Uma vez fechado, não há mais valores pode ser inserido eNão é mais não realizado// As promessas retornadas.this.closed = false;

```
 }enquistar (valor) {if (this.closed) {lançar um novo erro ("AsyncQueue fechado");}if (this.resolvers.length> 0) {// Se esse valor já foi prometido, Resolva essa promessaconst resolve = this.resolvers.shift ();resolver (valor);}outro {// caso contrário, filathis.values.push (valor);}dequeue () {if (this.values.length> 0) {// Se houver um valor na fila, retorne um resolvidoPromessa para issoconst valor = this.values.shift ();return promey.resolve (valor);}else if (this.closed) {// se não houver valores na fila e estamos fechados, retorne umresolvido// Promessa para o marcador "fim da stream"return promey.resolve (asyncQueue.eos);}outro {// caso contrário, retorne uma promessa não resolvida,// fila a função do resolvedor para uso posteriorretornar nova promessa ((resolve) => {this.resolvers.push (resolve);});}}fechar() {// Uma vez que a fila estiver fechada, não serão mais valorespreso.
```

```
// resolva as promessas pendentes com o final demarcador de fluxo while (this.resolvers.length > 0)
{this.resolvers.shift () (asyncqueue.eos);}this.closed = true;}// define o método que torna esta classe de forma
assíncrona iterável[Symbol.asynciterator] () {return this;}// define o método que torna este um assíncrono iterador.O//
Dequeue () Promise resolve um valor ou o EOSSentinel se formos// fechado.Aqui, precisamos devolver uma
promessa de quer resolve para um// objeto de resultado do iterador.próximo() {Retorne this.Dequeue (). Então (valor
=> (valor === Assíncqueue.eos)?{valor: indefinido, feito: verdadeiro}: {valor: valor, feito:falso});} // Um ?? valor
sentinela retornado por Dequeue () para marcar "fim defluxo "quando fechadoAsyncQueue.eos = símbolo ("fim da
corrente");Porque esta classe de assíncrona define a iteração assíncrona Noções básicas, podemos criar nossos
próprios iteradores assíncronos mais interessantes Simplesmente por valores de fila assíncronos.Aqui está um
exemplo que usa asyncqueue para produzir um fluxo de eventos do navegador da web que podem ser tratados com
um loop para/aguardar:// Push eventos do tipo especificado no especificado
```

elemento do documento// em um objeto assíncrono e devolva a fila para uso comoum fluxo de eventosFunção EventsTream (ELT, Type) {const q = new AsyncQueue ()// Crie afilaELT.AddeventListener (tipo, e => q.enqueue (e));// Enqueueeventosretornar q;}Função assíncreada HandleKeys () {// Obtenha um fluxo de eventos de KeyPress e loop uma vez para cadaumpara aguardar (const Event of EventStream (documento,"KeyPress")) {console.log (event.key);}13.5 ResumoNeste capítulo, você aprendeu:A maioria da programação JavaScript no mundo real é assíncrona.Tradicionalmente, a assincronia foi tratada com eventos eFunções de retorno de chamada.Isso pode ficar complicado, no entanto,Porque você pode acabar com vários níveis de retorno de chamadaaninhado dentro de outros retornos de chamada, e porque é difícil de fazerManuseio de erro robusto.As promessas fornecem uma nova maneira de estruturar funções de retorno de chamada.Se usado corretamente (e infelizmente, as promessas são fáceis de usarincorrectamente), eles podem converter código assíncrono queforam aninhados em cadeias lineares de então () chamadas ondeUm passo assíncrono de um cálculo segue outro.Além disso, as promessas permitem que você centralize seu manuseio de erros

codificar uma chamada única () no final de uma cadeia de então () chamadas. As palavras -chave assíncronas e aguardas nos permitem escrever código assíncrono que é baseado em promessa sob o capô, mas parece código síncrono. Isso facilita o código para entender e raciocinar. Se uma função for declarada ASYNC, ele retornará implicitamente uma promessa. Dentro de um assíncrono de função, você pode aguardar uma promessa (ou uma função que retorna uma promessa) como se o valor da promessa fosse de forma síncrona calculado. Objetos que são de maneira assíncrona podem ser usados ??com um para/aguarda loop. Você pode criar iterável de forma assíncrona objetos implementando um [Symbol.asyncIterator](). Método ou invocando uma função assíncrona *Função do gerador. Iteradores assíncronos fornecem uma alternativa aos eventos de "dados" em fluxos no nó e pode ser usado para representar um fluxo de eventos de entrada do usuário no lado do cliente JavaScript. 1 A classe XMLHttpRequest não tem nada a ver com XML. No cliente moderno-JavaScript lateral, foi amplamente substituído pela API Fetch (), que é coberta em §15.11.1. O exemplo de código mostrado aqui é o último exemplo baseado em XMLHttpRequest permanecendo neste livro. 2 Normalmente, você pode usar aguarda no nível superior no console do desenvolvedor de um navegador. 3 Há uma proposta pendente para permitir o nível superior aguardar em uma versão futura do JavaScript. 3 Aprendi sobre essa abordagem da iteração assíncrona do blog do Dr. Axel Rauschmayer, <https://2ality.com>.

Erro ao traduzir esta página.

§14.4 Ajustando o comportamento de seus tipos com bem conhecido Símbolos
§14.5 Criando DSLs (idiomas específicos de domínio) com Funções de tag de modelo
§14.6 Sondando objetos com métodos refletidos
§14.7 Controlando o comportamento do objeto com proxy

14.1 Atributos da propriedade
As propriedades de um objeto JavaScript têm nomes e valores, é claro, mas cada propriedade também possui três atributos associados que especificam como essa propriedade se comporta e o que você pode fazer com ela:

- O atributo `gravável` especifica se o valor de uma propriedade pode mudar.
- O atributo `enumerável` especifica se a propriedade é enumerada pelo loop `for/in` e o método `Object.keys()`.
- O atributo `configurável` especifica se uma propriedade pode ser excluída e também se os atributos da propriedade podem ser mudados.

Propriedades definidas em literais de objeto ou por atribuição comum a um objeto são graváveis, enumeráveis e configuráveis. Mas muitas das propriedades definidas pela biblioteca padrão JavaScript não são. Esta seção explica a API para consultar e definir a propriedade de atributos. Esta API é particularmente importante para os autores da biblioteca porque: Ele permite que eles adicionem métodos a protótipos de objetos e façam

eles não são adequados, como métodos internos. Permite que eles "travem" seus objetos, definindo propriedades que não podem ser alteradas ou excluídas. Lembre -se de §6.10.6 que, enquanto as propriedades de dados? têm um valor, Os Propriedades do Acessor? têm um método Getter e/ou Setter. Para os propósitos desta seção, vamos considerar o getter e Métodos Setter de uma propriedade acessadora para serem atributos da propriedade. Após essa lógica, até diremos que o valor de uma propriedade de dados é um atributo também. Assim, podemos dizer que uma propriedade tem um nome e quatro atributos. Os quatro atributos de uma propriedade de dados são valor, gravável, enumerável e configurável. Propriedades do acessador não ter um atributo de valor ou um atributo gravável: sua escritura é determinada pela presença ou ausência de um levantador. Então os quatro atributos de uma propriedade acessadora são Get, Set, Enumerable e configurável. Os métodos JavaScript para consultar e definir os atributos de uma propriedade usa um objeto chamado Descritor de propriedade para representar o conjunto de quatro atributos. Um objeto de descritor de propriedades possui propriedades com os mesmos nomes que os atributos da propriedade descrevem. Assim, o objeto do descritor de propriedades de uma propriedade de dados possui propriedades nomeadas valor, gravável, enumerável e configurável. E o descritor de uma propriedade de acessórios recebe e definir propriedades de valor e gravidade. O gravável, enumerável e Propriedades configuráveis ?? são valores booleanos, e o Get and SetPropriedades são valores de função. Para obter o descritor da propriedade para uma propriedade nomeada de um especificado Objeto, Call Object.getOwnPropertyDescriptor ():

```
// retorna {value: 1, gravável: verdadeiro, enumerável: verdadeiro,Configurável:  
True}Object.getownProperTyDescriptor ({x: 1}, "x");// Aqui está um objeto com uma propriedade de acessador  
somente leituraconst aleatoriamente = {obtenha Octet () {return Math.floor (Math.random ()*256);},};// retorna  
{get:/*func*/ , set: indefinido, enumerável: true,Configurável: True}Object.GetownPropertyDescriptor (Random,  
"Octet");// retorna indefinidos para propriedades e propriedades herdadasisso não  
existe.Object.getownPropertyDescriptor ({},"x") // =>indefinido;Não tal PropObject.getownPropertyDescriptor ({},"  
ToString") // =>indefinido;herdadoComo o próprio nome indica, object.GetownPropertyDescriptor ()Funciona  
apenas para propriedades próprias.Para consultar os atributos de herdadoPropriedades, você deve atravessar  
explicitamente a cadeia de protótipo.(VerObject.getProTypeOf () em §14.3);Veja também o  
semelhanteReflete.getownPropertyDescriptor () Função em §14.6.)Para definir os atributos de uma propriedade ou  
criar uma nova propriedade com osatributos especificados, chamado object.DefineProperty (), passando oobjeto a  
ser modificado, o nome da propriedade a ser criado ou alterado,e o objeto do descritor da propriedade:Seja o = {};//  
Comece sem propriedades// Adicione uma propriedade de dados não enumerável x com o valor  
1.Object.DefineProperty (O, "X", {Valor: 1,gravável: verdadeiro,enumerável: falso,
```

Configurável: Verdadeiro});// verifique se a propriedade está lá, mas não é entusiasmadaO.x // => 1Object.Keys (O) // => []// Agora modifique a propriedade x para que seja somente leituraObject.DefineProperty (O, "X", {Writable: false});// Tente alterar o valor da propriedadeO.x = 2;// falha silenciosamente ou joga TypeError em rigorosomodoO.x // => 1// A propriedade ainda está configurável, para que possamos mudar seuvalor como este:Object.DefineProperty (O, "X", {value: 2});O.x // => 2// agora mude x de uma propriedade de dados para uma propriedade de acessóriosObject.DefineProperty (O, "X", {get: function () {return 0;}});O.x // => 0O descritor da propriedade que você passa para objeto.DefineProperty ()não precisa incluir todos os quatro atributos.Se você está criando um novoPropriedade, os atributos omitidos são considerados falsos ouindefinido.Se você está modificando uma propriedade existente, então oOs atributos que você omitem simplesmente permanecem inalterados.Observe que este métodoaltera uma propriedade própria existente ou cria uma nova propriedade própria, masnão alterará uma propriedade herdada.Veja também a função muito semelhanteReflete.DefineProperty () em §14.6.Se você deseja criar ou modificar mais de uma propriedade por vez, useObject.DefineProperties ().O primeiro argumento é o objeto

Erro ao traduzir esta página.

Object.DefineProperties () atirar TypeError se a tentativa deCriar ou modificar uma propriedade não é permitida. Isso acontece se você tentarPara adicionar uma nova propriedade a um objeto não extensível (consulte §14.2). O outrorazões que esses métodos podem lançar o TypeError têm a ver com oatributos. O atributo gravável governa as tentativas deAltere o atributo de valor. E o atributo configurável governatenta mudar os outros atributos (e também especifica se umA propriedade pode ser excluída). As regras não são completamente diretas, no entanto. É possível alterar o valor de uma propriedade não escritor seEssa propriedade é configurável, por exemplo. Além disso, é possível mudaruma propriedade de gravidade a não escritor, mesmo que essa propriedade sejaNão Configurável. Aqui estão as regras completas. Chamadas paraObject.DefineProperty () ouObject.DefineProperties () que tentam violá -losJogue um TypeError: Se um objeto não for extensível, você pode editar seu próprioPropriedades, mas você não pode adicionar novas propriedades. Se uma propriedade não estiver configurável, você não pode alterar seuatributos configuráveis ??ou enumeráveis. Se uma propriedade acessadora não estiver configurável, você não poderá mudarSeu método getter ou setter, e você não pode alterá -lo para um dadospropriedade. Se uma propriedade de dados não estiver configurável, você não pode alterá -lo parauma propriedade acessadora. Se uma propriedade de dados não estiver configurável, você não pode alterar seuatributo gravável de false para verdadeiro, mas você pode mudarde verdadeiro a falso.

Se uma propriedade de dados não estiver configurável e não gravável, você não pode alterar seu valor. Você pode alterar o valor de umpropriedade que é configurável, mas não escritor, no entanto(porque isso seria o mesmo que torná -lo gravável, entãoAlterar o valor e convertê -lo novamente em não escritura).§6.7 descreveu a função object.assign () que copia a propriedadevalores de um ou mais objetos de origem em um objeto de destino.Object.assign () apenas copia propriedadesenumeráveis ??e propriedadesvalores, não atributos de propriedade.Isso é normalmente o que queremos, massignifica, por exemplo, que se um dos objetos de origem tiver umPropriedade do acessador, é o valor retornado pela função getter que éCopiado para o objeto de destino, não a própria função getter.Exemplo 14-1demonstra como podemos usarObject.GetownProperTyDescriptor () eObject.DefineProperty () para criar uma variante deObject.assign () que copia descritores inteiros de propriedadesdo que apenas copiar valores de propriedades.Exemplo 14-1.Copiar propriedades e seus atributos de um objetopara outro/** Definir um novo object.assignDescriptors () Função que funcionacomo* Object.assign (), exceto que ele copia os descritores de propriedadesde* fonte de objetos no objeto de destino em vez de apenascópia* valores de propriedade.Esta função copia todas as próprias propriedades,ambos* enumerável e não enumerável.E porque copiadescritores,* Copia as funções getter de objetos de origem esubstitui o setter* funciona no objeto de destino em vez de invocar aquelesgetters e

Erro ao traduzir esta página.

Seja o = {c: 1, obtenha count () {return this.c ++;}};// define objeto com getter
Seja p = object.assign ({} , o); // copie os valores da propriedade
Seja q = object.assignDescriptors ({} , o); // copie os descriptores de propriedade
desp.count // => 1: agora é apenas uma propriedade de dados para p.count // => 1: ... o contador não aumenta.
q.count // => 2: incrementado uma vez quando o copiamos o primeiro tempo, q.count // => 3: ... mas copiamos o método getter,
então increments.
14.2 Extensibilidade do objeto
O atributo extensível de um objeto especifica se novas propriedades podem ser adicionadas ao objeto ou não. Objetos JavaScript comuns são extensíveis por padrão, mas você pode mudar isso com as funções descritas nesta seção. Para determinar se um objeto é extensível, passe para Object.isExtensible (). Para tornar um objeto não extensível, passe para Object.PreventExtensions (). Depois de fazer isso, qualquer tentativa de adicionar uma nova propriedade ao objeto lançará um TypeError no modo rigoroso e simplesmente falha silenciosamente sem um erro em modo não rigoroso. Além disso, tentando alterar o protótipo (ver §14.3) de um objeto não extensível sempre lançará um TypeError. Observe que não há como tornar um objeto extensível novamente quando você tornaram não extensível. Observe também que chama Object.PreventExtensions () afeta apenas a extensibilidade do próprio objeto. Se novas propriedades forem adicionadas ao protótipo de um não

objeto extensível, o objeto não extensível herdará aqueles novos propriedades. Duas funções semelhantes, `Object.isExtensible()` e `Object.preventExtensions()`, são descritas em §14.6. O objetivo do atributo extensível é poder "travar" objetos em um estado conhecido e impedir adulteração externa. O atributo extensível dos objetos é frequentemente usado em conjunto com os atributos configuráveis ??e graváveis ??de propriedades e `Object.defineProperties` que facilitam a definição desses atributos: `Object.Seal()` funciona como `Object.PreventExtensions()`, mas além de tornando o objeto não extensível, também faz com que todas as propriedades desse objeto não sejam configuráveis. Isso significa que novas propriedades não podem ser adicionadas ao objeto e existentes as propriedades não podem ser excluídas ou configuradas. Propriedades existentes que são graváveis ??ainda podem ser definidas, no entanto. Não há como despertar um objeto selado. Você pode usar `Object.isSealed()` para determinar se um objeto está selado. `Object.freeze()` trava objetos ainda mais firmemente. Além de tornar o objeto não extensível e suas propriedades não confundíveis, também faz com que todas as propriedades de dados próprios somente leitura. (Se o objeto tiver métodos de setter, estes não são afetados e podem ainda ser chamados por atribuição à propriedade.) Use `Object.isFrozen()` para determinar se um objeto está congelado. É importante entender que `Object.seal()` e `Object.freeze()` afeta apenas o objeto que eles passam: eles têm

nenhum efeito no protótipo desse objeto. Se você quiser travar completamente para baixo de um objeto, você provavelmente precisa selar ou congelar os objetos na cadeia de protótipo também. `Object.PreventExtensions()`, `object.Seal()` e `Object.freeze()` todos retornam o objeto que eles passam, o que significa que você pode usá-los em invocações de funções aninhadas:// Crie um objeto selado com um protótipo congelado e um não-objetos enumeráveis. Seja o = `object.seal(object.create(object.freeze({x: 1}), {y: {value: 2, gravidade: verdadeiro}}))`; Se você está escrevendo uma biblioteca JavaScript que passa os objetos para o retorno de chamada, funções escritas pelos usuários da sua biblioteca, você pode usar `Object.freeze()` nesses objetos para impedir que o código do usuário modifique-os. Isso é fácil e conveniente, mas há comercio OFFs: Objetos congelados podem interferir nos testes JavaScript comuns e estratégias, por exemplo.

14.3 O atributo do protótipo O atributo de protótipo de um objeto especifica o objeto de que ele herda as propriedades. (Revisão §6.2.3 e §6.3.2 Para obter mais informações sobre protótipos e herança de propriedades.) Este é um atributo tão importante que nós geralmente simplesmente dizemos "o protótipo de O" em vez de "o protótipo do atributo de o." Lembre-se também de que quando o protótipo aparece em fonte de código, refere-se a uma propriedade de objeto comum, não ao atributo do protótipo: Capítulo 9 explicou que o protótipo

propriedade de uma função construtora especifica o atributo de protótipo dos objetos criados com esse construtor. O atributo do protótipo é definido quando um objeto é criado. Objetos criados a partir de literais de objeto, use objeto.prototype como seu protótipo. Objetos criados com novo use o valor do protótipo da propriedade de sua função construtora como protótipo. E objetos criados com object.create () use o primeiro argumento para isso (que pode ser nula) como seu protótipo. Você pode consultar o protótipo de qualquer objeto, passando esse objeto para Object.getPrototypeOf (): Object.getPrototypeOf ({}) // => object.prototype Object.getPrototypeOf ([]) // => Array.prototype Object.getPrototypeOf ((() => {})) // => function.prototype Uma função muito semelhante, reflect.getPrototypeOf (), é descrito em §14.6. Para determinar se um objeto é o protótipo de (ou faz parte da cadeia de protótipo de) Outro objeto, use o ISPrototypeOf () método: Seja p = {x: 1}; // Defina um protótipo de objeto. Seja o = object.create (p); // Crie um objeto com esse protótipo. p.isPrototypeOf (o) // => true: o herda de p Object.prototype.isPrototypeOf (p) // => true: p herda de Object.prototype

`Object.prototype.isPrototypeOf (o) // => true: o também faz observe que o ISPrototypeOf () desempenha uma função semelhante ao Instância do operador (consulte §4.9.4). O atributo do protótipo de um objeto é definido quando o objeto é criado e normalmente permanece fixo. Você pode, no entanto, mudar o protótipo de um objeto com object.SetPrototypeOf (): Seja o = {x: 1}; Seja p = {y: 2}; Object.setPrototypeOf (O, P); // Defina o protótipo de O para PO. y // => 2; o agora herda a propriedade y. Seja a = [1, 2, 3]; Object.setPrototypeOf (a, p); // defina o protótipo da matriz aprincipalA. Join // => indefinido: A não tem mais um método de junção () Geralmente não há necessidade de usar Object.setPrototypeOf (). As implementações de JavaScript podem fazer otimizações agressivas com base na suposição de que o protótipo de um objeto é fixo e imutável. Isso significa que se você sempre ligue para object.setPrototypeOf (), qualquer código que use os objetos alterados podem ser executados muito mais lentos do que normalmente. Uma função semelhante, reflet.setPrototypeOf (), é descrito em §14.6. Algumas implementações iniciais do navegador de JavaScript expuseram o atributo do protótipo de um objeto através da propriedade __proto__ (Escrito com dois sublinhados no início e no final). Isso há muito tempo`

foi obsoleto, mas o código existente suficiente na web depende de __proto__ que o padrão EcmaScript exige para todos implementações JavaScript que são executadas nos navegadores da Web.(Supor tes do nó também, embora o padrão não exija para o nó.) No moderno JavaScript, __proto__ é legível e escrito, e você pode(embora você não deva) usá-lo como uma alternativa a Object.getPrototypeOf () e Object.setPrototypeOf (). Um uso interessante de __Proto__, no entanto, é definir o protótipo de um objeto literal: Seja p = {z: 3}; Seja o = {x: 1, y: 2, __proto__: p}; O.z // => 3: o herda de P14.4 Símbolos bem conhecidos O tipo de símbolo foi adicionado ao JavaScript em ES6, e um dos principais razões para isso foi adicionar com segurança extensões ao idioma sem quebrar a compatibilidade com o código já implantado na web. Vimos um exemplo disso no capítulo 12, onde nós aprendemos que você pode tornar uma classe iterável implementando um método cujo "nome" é o símbolo símbolo.iterator.Symbol.iterator é o exemplo mais conhecido de ?conhecidoSímbolos. ?Estes são um conjunto de valores de símbolo armazenados como propriedades do Função de fábrica de símbolo () que são usados ??para permitir que o código JavaScript

Controle certos comportamentos de baixo nível de objetos e classes. As subseções a seguir descrevem cada um desses símbolos conhecidos e explique como eles podem ser usados.

14.4.1 Symbol.iterator e Symbol.asyncIterator

Os símbolos iterator e símbolo.asyncIterator permitem que objetos ou classes se tornem iteráveis ?? ou assíncronas iteráveis. Eles foram cobertos em detalhes nos Capítulo 12 e §13.4.2, respectivamente, e são mencionados novamente aqui apenas para completar.

14.4.2 Symbol.HasInstance

Quando a instância do operador foi descrito em §4.9.4, dissemos que o lado da direita deve ser uma função construtora e que o Expressão o Instância de F foi avaliada procurando o valor F. Protótipo dentro da cadeia de protótipos de O. Isso ainda é verdade, mas no ES6 e além, o símbolo. O Hasinstance fornece uma alternativa. No ES6, se o lado direito de mão de Instanceof é um objeto com um [Symbol.hasinstance] Método, então esse método é invocado com o valor colateral à esquerda como argumento e o valor de retorno do método, convertido em um booleano, torna -se o valor da instância do operador. É, é claro, se o valor no caminhado não tem um método [símbolo.hasinstance], mas é uma função, então o operador da instância se comporta de maneira comum. Symbol.hasinstance significa que podemos usar a instância operador para fazer uma verificação do tipo genérico com pseudótipo adequadamente definido objetos. Por exemplo:

```
// define um objeto como um "tipo" que podemos usar com a instânciaSeja uint8 = {[Symbol.hasinstance] (x)
{retornar número.isinteger (x) && x >= 0 && x <= 255;}};128 instância de uint8 // => true256 instância de uint8 // =>
false: muito grandeMath.pi instância de uint8 // => false: não é um número inteiroObserve que este exemplo é
inteligente, mas confuso porque usa umObjeto não clássico onde uma classe normalmente seria
esperada.Seria tão fácil - e mais claro para os leitores do seu código - para escrever umIsuint8 () função em vez de
confiar nissoSymbol.Hasinsance Behavior.14.4.3 Symbol.ToStringTagSe você invocar o método tostring () de um
objeto JavaScript básico,você recebe a sequência "[objeto objeto]":{} .tostring () // => "[objeto objeto]"Se você
invocar esse mesmo objeto.prototype.toString ()função como um método de instâncias de tipos internos, você
obtém alguns resultados interessantes:Object.prototype.toString.Call ([]) // => "[Array do
objeto]"Object.prototype.tostring.call (/./) // => "[ObjetoRegexp]"Object.prototype.toString.Call (() => {}) // =>
"[ObjetoFunção]"Object.prototype.toString.Call ("") // => "[ObjetoCorda]"Object.prototype.toString.Call (0) // =>
"[Objeto
```

Número]"Object.prototype.toString.Call (false) // => "[ObjetoBooleano] "Acontece que você pode usar issoObject.prototype.toString (). Call () Técnica com qualquerValor Javascript para obter o "atributo de classe" de um objeto queContém informações de tipo que não estão disponíveis de outra forma.A seguir a função de classe () é indiscutivelmente mais útil do que o tipo deOperador, que não faz distinção entre tipos de objetos:função classf (o) {Return Object.Prototype.ToString.Call (O) .Slice (8, -1);}classe de (nulo) // => "nulo"classe de (indefinido) // => "indefinido"classe de (1) // => "Número"classe de (10n ** 100n) // => "bigint"Classof ("") // => "String"classe de (false) // => "booleano"Classof (símbolo ()) // => "Símbolo"classef ({})) // => "objeto"Classof ([])) // => "Array"Classof (./.) // => "regexp"classef ((() => {})) // => "função"Classof (novo mapa ()) // => "mapa"Classof (new Set ()) // => "Set"classe de (new Date ()) // => "Data"Antes do ES6, esse comportamento especial doObject.prototype.toString () Método estava disponível apenas paraInstâncias de tipos internos e se você chamou essa função de classe ()Em uma instância de uma aula que você havia definido, simplesmenteretornar "objeto".Em ES6, no entanto,

Object.prototype.toString () procura uma propriedade com oNome simbólico Symbol.ToStringTag em seu argumento, e se talExiste uma propriedade, ele usa o valor da propriedade em sua saída. Isso significa que se você definir uma classe própria, poderá fazer com que funcione facilmente com Funções como Classof ():intervalo de classe {get [symbol.toStringTag] () {return "range";}// O resto desta classe é omitido aqui}Seja r = novo intervalo (1, 10);Object.prototype.toString.call (r) // => "[intervalo de objeto]"classe de (r) // => "Range"14.4.4

Symbol.SpeciesAntes do ES6, o JavaScript não forneceu nenhuma maneira real de criar robustos subclasses de classes internas como a matriz. Em ES6, no entanto, você pode estender qualquer classe interna simplesmente usando a classe e estendepalavras -chave. §9.5.2 demonstrou que, com esta subclasse simples da matriz:// Uma subclasse de matriz trivial que adiciona getters para o primeiro e último elementos.classe EZARRAY estende a matriz {obtenha primeiro () {retornar este [0];}obtenha last () {return this [this.length-1];}}Seja E = novo EZARRAY (1,2,3);Seja f = e.map (x => x * x);E.Last // => 3: O último elemento de Ezarray eF.Last // => 9: F também é um ezarray com uma última propriedadeArray define métodos concat (), filtro (), map (), slice (),

e Splice (), que retornam matrizes. Quando criamos uma matrizsubclasse como ezarray que herda esses métodos, deve ser herdadoInstâncias de retorno do método de matriz ou instâncias de ezarray? BomArgumentos podem ser feitos para qualquer opção, mas a especificação ES6 diz que (por padrão) os cinco métodos de retorno de matriz retornarãoInstâncias da subclasse. Aqui está como funciona: No ES6 e mais tarde, o construtor Array () tem uma propriedade com o símbolo do nome simbólico. Spécies. (Observe que issoO símbolo é usado como o nome de uma propriedade do construtorfunção. A maioria dos outros símbolos bem conhecidos descritosAqui são usados ??como o nome dos métodos de um objeto de protótipo.) Quando criamos uma subclasse com estendências, o resultanteO construtor de subclasse herda as propriedades da superclasseconstrutor. (Isso é um acréscimo ao tipo normal de herança, onde casos dos métodos de herdamento da subclasse dea superclasse.) Isso significa que o construtor para cadaA subclasse da Array também possui uma propriedade herdada com nomeSymbol. Spécies. (Ou uma subclasse pode definir seu própriopropriedade com este nome, se quiser.) Métodos como map () e slice () que criam e retornamNovas matrizes são ligeiramente aprimoradas no ES6 e mais tarde. Em vez deApenas criando uma matriz regular, eles (com efeito) invocam novosthis.Constructor [Symbol.Species] () para criara nova matriz. Agora aqui está a parte interessante. Suponha que issoArray [Symbol.Species] era apenas uma propriedade de dados regular, definido assim:

Erro ao traduzir esta página.

obtenha last () {return this [this.length-1];}Seja E = novo EZARRAY (1,2,3);Seja f = e.map (x => x - 1);E.Last // => 3F.Last // => indefinido: f é uma matriz regular sem últimogetterCriar subclasses úteis de matriz foi o principal caso de uso quem motivou a introdução de símbolo.único local que este símbolo bem conhecido é usado.Aulas de matriz digitadasUse o símbolo da mesma maneira que a classe da matriz.De forma similar,O método Slice () de ArrayBuffer olha para oSymbol.pécies bens this.Constructor em vez deSimplesmente criando um novo ArrayBuffer.E promessa métodos comoentão () que retornam novos objetos de promessa criam esses objetos por meio dissoProtocolo de espécies também.Finalmente, se você se encontrar no mapa subclassificador(por exemplo) e métodos de definição que retornam novos objetos de mapa, você pode querer usar o símbolo.subclasses da sua subclasse.14.4.5 Symbol.iscoNcatsPreadableO método da matriz concat () é um dos métodos descritos noseção anterior que usa símbolo.pécies para determinar o queconstrutor para usar para a matriz retornada.Mas concat () também usaSymbol.iscoNcatsPreadable.Lembre -se do §7.8.3 de que oO método concat () de uma matriz trata seu valor e sua matrizArgumentos de maneira diferente dos seus argumentos não provocados: argumentos não marcantessão simplesmente anexados à nova matriz, mas a matriz e qualquer

Os argumentos da matriz são achados ou "espalhados" para que os elementos da matriz sejam concatenados em vez do próprio argumento da matriz. Antes do ES6, concat () acabou de usar o Array.isArray () para determinar se deve tratar um valor como uma matriz ou não. No ES6, o algoritmo é alterado ligeiramente: se o argumento (ou este valor) para concat () for um objeto e tiver uma propriedade com o nome simbólicoSymbol.isConcatSpreadable, então o valor booleano daquela propriedade é usada para determinar se o argumento deve ser "espalhado". Se não existir tal propriedade, então Array.isArray () é usado como nas versões anteriores do idioma. Existem dois casos em que você pode querer usar este símbolo: Se você criar um objeto de matriz (consulte §7.9) e deseja que ele se comporte como uma matriz real quando passada para concat (), você pode simplesmente adicionar a propriedade simbólica ao seu objeto: Deixe a matriz = {Comprimento: 1, 0: 1, [Symbol.isConcatSpreadable]: Verdadeiro}; [] .CONCAT (Matriz) // => [1]: (Seja [[1]] se não for espalhado) As subclasses de matriz são espalhadas por padrão, então se você estiver definindo uma subclasse de matriz que você não deseja agir como um Array quando usado com concat (), então você pode adicionar um getter assim para a sua subclasse:

classe não readableArray estende a matriz {get [symbol.isConcatSpreadable] () {retornar falso;}}Seja a = novo não adolescente (1,2,3);[] .CONCAT (a) .Length // => 1;(seria 3elementos longos se A foi espalhado)14.4.6 Símbolos de correspondência de padrões§11.3.2 documentou os métodos de string que executam a correspondência de padrõesoperações usando um argumento regexp.No ES6 e mais tarde, esses métodos foram generalizados para trabalhar com objetos regexp ou qualquer objeto que define o comportamento de correspondência de padrões por meio de propriedades com nomes simbólicos.Para cada um dos métodos de string correspondem (), matchAll (), pesquisa (),substituir () e split (), existe um conhecido bem conhecidoSímbolo: Symbol.Match, Symbol.Search, e assim por diante.Os regexps são uma maneira geral e muito poderosa de descrever textualpadrões, mas eles podem ser complicados e não adequados para confusoscorrespondência.Com os métodos generalizados de string, você pode definir suasclasses de padrões próprios usando os métodos de símbolo bem conhecidos para fornecer correspondência personalizada.Por exemplo, você pode executar comparações de stringUsando o Intl.Collator (consulte o §11.7.3) para ignorar sotaques ao corresponder.Ouvocê pode definir uma classe de padrão baseada no algoritmo SoundEx paraCombine as palavras com base em seus sons aproximados ou para combinar vagamenteStrings até uma dada distância de Levenshtein.Em geral, quando você invoca um desses cinco métodos de string em um

Objeto de padrão como este: String.method (padrão, arg) Essa invocação se transforma em uma invocação de um nome simbolicamente nomeado Método em seu objeto de padrão: padrão [símbolo] (string, arg) Como exemplo, considere a classe de correspondência de padrões na próxima Exemplo, que implementa a correspondência de padrões usando o simples * e ? Wildcards com os quais você provavelmente é familiar de sistemas de arquivos. Esse Estilo de correspondência de padrões remonta aos primeiros dias do Unix sistema operacional, e os padrões são frequentemente chamados de globos: classe Glob { construtor (glob) {this.glob = glob; // Implementamos a correspondência global usando regexp internamente. // ? corresponde a qualquer personagem exceto /, e * corresponde zero ou mais // desses personagens. Usamos grupos de captura ao redor de cada um. Seja regexptext = glob.replace ("?", "[[^/]]"). Substitua ("*", "[^/]*"); // Usamos o sinalizador de U para obter correspondência com reconhecimento de unicode. // Globos destinam -se a combinar com seqüências inteiras, então nós Use o ^ e \$ // ancora e não implementa pesquisa () ou matchall () desde que eles // não são úteis com padrões como este. this.Regexp = novo regexp (^\$ {regexptext} \$` , "u"); } toString () { return this.glob; }}

[Symbol.search] (s) {return s.search (this.regexp);} [Symbol.match] (s) {return s.match (this.Regexp);} [Symbol.replace] (S, substituição) {retornar S.Replace (this.Regexp, substituição);} Deixe o padrão = novo glob ("docs/*.txt"); "Docs/js.txt" .search (padrão) // => 0: corresponde ao caractere0"docs/js.htm" .search (padrão) // => -1: não corresponde Seja match = "docs/js.txt" .match (padrão); corresponder [0] // => "Docs/js.txt" corresponder [1] // => "JS" match.index // => 0 "Docs/js.txt" .replace (padrão, "web/\$1.htm") // => "Web/js.htm" 14.4.7 Símbolo.Toprimitivo§3.9.3 explicou que o JavaScript tem três algoritmos ligeiramente diferentes para converter objetos em valores primitivos. Vagamente falando, para conversões em que um valor de string é esperado ou preferido, JavaScript Invoca o método ToString () de um objeto primeiro e volta ao Método Valueof () se ToString () não for definido ou não retornar um valor primitivo. Para conversões onde um valor numérico é preferido, JavaScript tenta o método ValueOf () primeiro e cai para trás no tostring () se valueof () não for definido ou se não retornar um valor primitivo. E finalmente, nos casos em que não há preferência, é Vamos a classe decidir como fazer a conversão. Objetos de data convertem usando o tostring () primeiro, e todos os outros tipos tentam valueof () primeiro. No ES6, o conhecido símbolo símbolo.Toprimitive permite que você

para substituir esse comportamento de objeto-para-primitivo padrão e lhe dar controle completo sobre como as instâncias de suas próprias aulas serão convertidas em valores primitivos. Para fazer isso, defina um método com o mesmo nome simbólico. O método deve retornar um valor primitivo que de alguma forma representa o objeto. O método que você definir será invocado com um único argumento de string que informa que tipo de conversão JavaScript está tentando fazer em seu objeto: Se o argumento for "string", significa que JavaScript está fazendo a conversão em um contexto em que esperaria ou preferia (mas não exigir) uma string. Isso acontece quando você interpolar o objeto em um modelo literal, por exemplo. Se o argumento for "número", significa que o JavaScript está fazendo a conversão em um contexto em que esperaria ou preferia (mas não exigir) um valor numérico. Isso acontece quando você usa o objeto com um operador <ou> ou com operadores aritméticos como - e *. Se o argumento for "padrão", significa que o JavaScript está convertendo seu objeto em um contexto em que um numérico ou o valor da string pode funcionar. Isso acontece com os operadores +, == e !=. Muitas classes podem ignorar o argumento e simplesmente retornar o mesmo valor primitivo em todos os casos. Se você deseja que as instâncias da sua classe sejam comparáveis e classificáveis com <e>, então esse é um bom motivo para definir um método [symbol.toPrimitive]. 14.4.8 Symbol.unscopables O símbolo final bem conhecido que abordaremos aqui é obscuro, que foi introduzido como uma solução alternativa para questões de compatibilidade causadas por

o depreciado com a declaração. Lembre -se de que a declaração comum objeto e executa seu corpo de declaração como se estivesse em um escopo ondeAs propriedades desse objeto eram variáveis. Isso causou compatibilidadeproblemas quando novos métodos foram adicionados à aula de matriz, e issoquebrou algum código existente. Symbol.Unscopables é o resultado. Em ES6 e, posterior, a declaração com foi ligeiramente modificada. Quando Usado com um objeto O, a com declaração calculaObject.Keys (o [symbol.unscopables] || {}) e ignorarpropriedades cujos nomes estão na matriz resultante ao criar o escopo simulado para executar seu corpo. ES6 usa isso para adicionar novoMétodos para Array.prototype sem quebrar o código existente ema web. Isso significa que você pode encontrar uma lista da matriz mais recenteMétodos avaliando: Deixe newArrayMethods = Object.Keys (Array.prototype [symbol.unscopables]); 14.5 Tags de modeloStrings dentro de backsticks são conhecidas como "literais de modelo" e foram coberto em §3.3.4. Quando uma expressão cujo valor é uma função é seguido por um modelo literal, ele se transforma em uma invocação de funções e Chamamos isso de "Literal de modelo marcado". Definindo uma nova função de tag para Use com os literais de modelo marcado pode ser pensado com metaprogramação, porque os modelos marcados são frequentemente usados ?? para definirDSLs-idiomas específicos de domínio-e definir uma nova função de tag é Como adicionar nova sintaxe ao JavaScript. Modelos marcados que os literais têm sido adotado por vários pacotes JavaScript de front -end. O A linguagem de consulta GraphQL usa uma função de tag gql` para permitir consultas

a ser incorporado no código JavaScript. E a biblioteca de emoção usa um função de tag css`` para permitir que os estilos CSS sejam incorporados JavaScript. Esta seção demonstra como escrever sua própria tagfunções como essas. Não há nada de especial nas funções de tags: elas são comunsAs funções JavaScript e nenhuma sintaxe especial são necessárias para defini -las. Quando uma expressão de função é seguida por um modelo literal, oA função é invocada. O primeiro argumento é uma variedade de cordas, e isso éseguido por zero ou mais argumentos adicionais, que podem ter valoresde qualquer tipo. O número de argumentos depende do número de valores que sãointerpolado no modelo literal. Se o modelo literal é simplesmente umstring constante sem interpolações, então a função de tag será chamado com uma matriz dessa corda e nenhum argumento adicional. SeO modelo literal inclui um valor interpolado, depois a tagA função é chamada com dois argumentos. O primeiro é uma variedade de doisStrings, e a segunda é o valor interpolado. As cordas nissomatriz inicial são a string à esquerda do valor interpolado e oString à sua direita, e qualquer um deles pode ser a corda vazia. Se oModelo literal inclui dois valores interpolados, depois a função da tagé invocado com três argumentos: uma variedade de três cordas e os doisvalores interpolados. As três cordas (uma ou todas as quais podem servazios) são o texto à esquerda do primeiro valor, o texto entre os doisvalores e o texto à direita do segundo valor. No caso geral,Se o modelo literal tiver n valores interpolados, a função da tagserá invocado com n+1 argumentos. O primeiro argumento será um

Matriz de N+1 Strings, e os argumentos restantes são os nOs valores interpolados, na ordem em que aparecem no modelo literal.O valor de um modelo literal é sempre uma string.Mas o valor de umO modelo tag literal é o valor que a função de tag retorna.Esse pode ser uma string, mas quando a função de tag é usada para implementar um DSL,O valor de retorno é tipicamente uma estrutura de dados que não corta representação da string.Como exemplo de uma função de tag de modelo que retorna uma string, considere o seguinte modelo html` , que é útil quando você deseja interpolar os valores com segurança em uma sequência de HTML.A tag executaHTML escape em cada um dos valores antes de usá -lo para construir o finalcorda:função html (strings, ... valores) { // converte cada valor em uma string e escape de HTML especialcaracteresDeixe escapar = valores.map (v => string (v).place ("&", "&").place ("<", "<").place (">", ">").place ("\"", "\"").place ("'", "'"); // retorna as cordas concatenadas e valores escapadosdeixe resultado = strings [0];para (vamos i = 0; i < escapado.length; i ++){resultado += escape [i] + strings [i +1];}resultado de retorno;}deixe operator = "<";html` x \$ {operator} y ` // =>" x & lt;

y "Let Kind = "Game", Name = "D&D";html` <div class = "\$ {Kind}"> \$ {name} </div> `// => '<divclass = "Game"> d & d </div> 'Para um exemplo de uma função de tag que não retorna uma string, masEm vez disso, uma representação analisada de uma corda, pense no globoClasse de padrão definida no §14.4.6.Como o construtor glob () leva umargumento de string única, podemos definir uma função de tag para criar novosObjetos Glob:função glob (strings, ... valores) {// monta as cordas e valores em uma única stringSeja s = strings [0];para (vamos i = 0; i <valores.length; i ++) {s += valores [i] +strings [i +1];}// retorna uma representação analisada dessa stringRetornar New Glob (s);}deixe root = "/tmp";deixe filepattern = glob` \$ {root}/*. html`;// um regexpalternativa"/tmp/test.html".match(FilePattern):1] // =>" teste "Um dos recursos mencionados de passagem no §3.3.4 é oFunção de tag String.raw`` que retorna uma string em sua forma "RAW"sem interpretar nenhuma das sequências de fuga de barragem.Isso éimplementado usando um recurso de invocação de funções de tag que temosainda não discutido.Quando uma função de tag é invocada, vimos que éO primeiro argumento é uma variedade de cordas.Mas esta matriz também tem uma propriedadechamado Raw, e o valor dessa propriedade é outra gama de cordas,com o mesmo número de elementos.A matriz de argumentos inclui strings

que tiveram sequências de fuga interpretadas como de costume. E a matriz brutal inclui strings nas quais as sequências de fuga não são interpretadas. Esse recurso obscuro é importante se você deseja definir um DSL com um Gramática que usa barras de barriga. Por exemplo, se quiséssemos nosso Função de tag glob `` para suportar a correspondência de padrões no estilo Windows caminhos (que usam barras de barragem em vez de barras para a frente) e nós fizemos não quero que os usuários da tag tenham que dobrar cada barra de barriga, poderíamos reescrever essa função para usar strings.raw [] em vez de strings []. A desvantagem, é claro, seria que não poderíamos usar mais longo escapes como \ u em nossos literais globais.

14.6 A API Refletir

O objeto refletir não é uma classe; Como o objeto de matemática, suas propriedades Basta definir uma coleção de funções relacionadas. Essas funções, adicionadas No ES6, defina uma API para "refletir sobre" objetos e seus propriedades. Há pouca funcionalidade nova aqui: o objeto refletir define um conjunto conveniente de funções, tudo em um único namespace, que imita o comportamento da sintaxe da linguagem central e duplicar os recursos de várias funções de objeto pré-existentes. Embora as funções refletidas não forneçam novos recursos, eles agrupam os recursos em uma API conveniente. É importante ressaltar que o conjunto de funções refletidas mapeia um a um com o Conjunto de métodos de manipulador de proxy sobre os quais aprenderemos no §14.7. A API reflete consiste nas seguintes funções:

Reflete.Apply (f, o, args) Esta função chama a função f como um método de O (ou chama -acomo uma função sem nenhum valor se o for nulo) e passa ovalores na matriz args como argumentos.É equivalente aF.Apply (O, Args).Reflet.Construct (C, Args, NewTarget) Esta função chama o construtor C como se a nova palavra -chave tivessefoi usado e passa os elementos da matriz args como argumentos.Se o argumento opcional newTarget for especificado, ele é usado como oValor de New.Target dentro da invocação do construtor.Se nãoEspecificado, o valor do novo.Target será c.Reflete.DefineProperty (O, nome, desritor) Esta função define uma propriedade no objeto O, usando o nome (astring ou símbolo) como o nome da propriedade.O desritorObjeto deve definir o valor (ou getter e/ou setter) e atributosda propriedade.Reflete.DefineProperty () é muito semelhantepara object.DefineProperty (), mas retorna true no sucessoe false em falhas.(Object.DefineProperty () retornaao Sobre o sucesso e lança o TypeError na falha.)Reflete.DeleteProperty (O, nome) Esta função exclui a propriedade com a string especificada ou nome simbólico do objeto O, retornando true se for bem -sucedido (ouse não existisse essa propriedade) e falsa se a propriedade não pudesse serexcluído.Chamar esta função é semelhante a escrever deleteo [nome].Reflete.get (o, nome, receptor) Esta função retorna o valor da propriedade de O com oNome especificado (uma string ou símbolo).Se a propriedade for um acessador

método com um getter, e se o argumento do receptor opcional forespecificado, então a função getter é chamada de método dереceptor em vez de como método de o.Chamar esta função éSemelhante a avaliar o [nome].Reflet.GetownPropertyDescriptor (O, nome)Esta função retorna um objeto de descriptor de propriedade que descreve oatributos da propriedade nomeada nome do objeto O, ou retorna indefinido se não existir tal propriedade.Esta função é quaseidêntico a object.GetownPropertyDescriptor (),exceto que o reflete a versão da API da função exige que oO primeiro argumento é um objeto e lança o TypeError, se não for.Reflete.getPrototypeOf (O)Esta função retorna o protótipo do objeto O ou nulo se oObjeto não possui protótipo.Joga um TypeError se o é um primitivovalor em vez de um objeto.Esta função é quase idêntica aObject.getProTypeOf (), exceto issoObject.getPrototypeOf () apenas joga um TypeError paraArgumentos nulos e indefinidos e coages outros primitivosvalores para seus objetos de wrapper.Reflete.Has (o, nome)Esta função retorna true se o objeto O tiver uma propriedade com oNome especificado (que deve ser uma string ou um símbolo).Chamando issoA função é semelhante a avaliar o nome em o.Reflete.isextensible (O)Esta função retorna true se o objeto O for extensível (§14.2) eFalso se não for.Ele lança um TypeError se O não for um objeto.Object.isExtensible () é semelhante, mas simplesmente retornaFalse quando passou um argumento que não é um objeto.

Refletir.wowys (O) Esta função retorna uma matriz dos nomes das propriedades do Objeto O ou lança um TypeError se O não for um objeto. Os nomes em A matriz devolvida será strings e/ou símbolos. Chamando isso A função é semelhante à chamada Object.getOwnPropertyNames () e Object.getOwnPropertySymbols () e combinando seus resultados. Reflete.PreventExtensions (O) Esta função define o atributo extensível (§14.2) do objeto O para Falso e retorna verdadeiro para indicar sucesso. Joga um TypeError se O não for um objeto. Object.preventExtensions () tem o mesmo efeito, mas retorna o em vez de verdadeiro e não joga TypeError para argumentos não -objeto. Reflete.set (o, nome, valor, receptor) Esta função define a propriedade com o nome especificado do Objeto O para o valor especificado. Ele retorna verdadeiro sobre o sucesso e Falso na falha (o que pode acontecer se a propriedade for somente leitura). Joga TypeError se o não for um objeto. Se a propriedade especificada for uma propriedade acessadora com uma função de setter e se o opcional O argumento do receptor é passado, então o setter será invocado como um método de receptor em vez de ser invocado como um método de O. Chamar essa função geralmente é a mesma que avaliar o [nome] = valor. Reflete.SetPrototypeOf (O, P) Esta função define o protótipo do objeto O para P, retornando verdadeiro no sucesso e falso no fracasso (o que pode ocorrer se o fornão extensível ou se a operação causaria um protótipo circular).

corrente).Joga um TypeError se o não for um objeto ou se p não forum objeto nem nulo.Object.SetProTypeOf () é semelhante,mas retorna o Sucesso e lança o TypeError na falha.Lembre -se de que chamar uma dessas funções provavelmente faráSeu código mais lento, interrompendo o intérprete JavaScriptotimizações.14.7 Objetos de proxyA aula de proxy, disponível no ES6 e mais tarde, é maispoderoso recurso de metaprogramação.Nos permite escrever código quealtera o comportamento fundamental dos objetos JavaScript.A API refletidadescrito no §14.6 é um conjunto de funções que nos dá acesso direto a umConjunto de operações fundamentais em objetos JavaScript.Que proxyA classe faz é nos permite uma maneira de implementar esses fundamentaisoperações de nós mesmos e criam objetos que se comportam de maneiras que não sãopossível para objetos comuns.Quando criamos um objeto de proxy, especificamos dois outros objetos, o alvoobjeto e os manipuladores objeto:deixe proxy = novo proxy (destino, manipuladores);O objeto de procuração resultante não tem estado ou comportamento próprio.Sempre que você executa uma operação nela (leia uma propriedade, escreva umpropriedade, defina uma nova propriedade, procure o protótipo, invocar como umfunção), ele despacha essas operações para o objeto de manipuladores ou para oobjeto alvo.As operações suportadas por objetos de procuração são iguais a

definido pela API refletida. Suponha que P seja um objeto de proxy e você Escreva Delete P.X. A função reflecte.DeleteProperty () tem o mesmo comportamento que o operador de exclusão. E quando você usa o Excluir operador para excluir uma propriedade de um objeto proxy, ele procura um Método deleteProperty () no objeto Handlers. Se tal O método existe, ele chama isso. E se não existe esse método, então o Objeto proxy executa a exclusão da propriedade no objeto de destino em vez de. Os proxies funcionam desta maneira para todas as operações fundamentais: se um O método apropriado existe no objeto de manipuladores, ele chama que Método para executar a operação. (Os nomes e assinaturas de métodos são iguais aos das funções refletidas abordadas no §14.6.) E se Esse método não existe no objeto de manipuladores, então o proxy executa a operação fundamental no objeto de destino. Isso significa que um proxy pode obter seu comportamento do objeto alvo ou manipuladores objeto. Se o objeto de manipuladores estiver vazio, o proxy será essencialmente um invólucro transparente em torno do objeto de destino: Seja t = {x: 1, y: 2}; Seja p = novo proxy (t, {}); p.x // => 1 Exclua P.Y // => True: Exclua a propriedade y do proxy. t.y // => indefinido: isso o exclui no alvo, também p.z = 3; // Definindo uma nova propriedade no proxy. t.z // => 3: define a propriedade no alvo. Esse tipo de proxy de invólucro transparente é essencialmente equivalente ao objeto de destino subjacente, o que significa que realmente não há um motivo para Use -o em vez do objeto embrulhado. Invólucros transparentes podem ser

Útil, no entanto, quando criado como "proxies revogáveis". Em vez de Criando um proxy com o construtor proxy (), você pode usar o Função de fábrica proxy.revocable (). Esta função retorna um objeto que inclui um objeto proxy e também uma função revoke (). Depois de chamar a função revoke (), o proxy para imediatamente trabalhando: Função AccessThEdAtabase () { /* Implementação omitida */ retornar 42; } Seja {proxy, revoke} = proxy.revocable (acesssthedatabase, {}); proxy () // => 42: o proxy dá acesso ao subjacente função alvoregar(); // mas esse acesso pode ser desligado sempre que nós querer proxy (); //! TypeError: não podemos mais chamar essa função Observe que, além de demonstrar proxies revogáveis, o anterior O código também demonstra que os proxies podem funcionar com funções de destino com obrem como objetos de destino. Mas o ponto principal aqui é que proxies revogáveis são um bloco de construção para um tipo de isolamento de código e você pode usar Ao lidar com bibliotecas de terceiros não confiáveis, por exemplo. Se Você tem que passar uma função para uma biblioteca que você não controla, você pode Passe um proxy revogável e depois revogue o proxy quando você está terminou com a biblioteca. Isso impede a biblioteca de manter um referência à sua função e chamando -a em momentos inesperados. Esse tipo de programação defensiva não é típica em programas JavaScript, mas A classe de proxy pelo menos torna isso possível. Se passarmos a um manipulador não vazio se opõe ao construtor proxy (), Então não estamos mais definindo um objeto de invólucro transparente e somos

Em vez disso, implementar o comportamento personalizado para o nosso proxy. Com o conjunto certodos manipuladores, o objeto alvo subjacente se torna essencialmente irrelevante. No código a seguir, por exemplo, é como poderíamos implementar umobjeto que parece ter um número infinito de propriedades somente leitura, onde o valor de cada propriedade é o mesmo que o nome dopropriedade:// usamos um proxy para criar um objeto que parece tertodo// Propriedade possível, com o valor de cada propriedade igualpara seu nomeSeja identidade = novo proxy ({}), {// Cada propriedade tem seu próprio nome como seu valorobtenha (o, nome, destino) {return name;},// Cada nome de propriedade é definidotem (o, nome) {return true;},// Existem muitas propriedades para enumerar, então apenas nóslançarOwnKeys (O) {jogue novo RangeError ("Número Infinito depropriedades ");},// todas as propriedades existem e não são graváveis,configurável ou enumerável.getOwnPropertyDescriptor (o, nome) {retornar {Valor: Nome,enumerável: falso,gravável: falso,Configurável: falso};},// Todas as propriedades são apenas leituras para que não possam ser definidasset (o, nome, valor, destino) {return false;},// Todas as propriedades não são confundíveis, então não podem serexcluído deleteProperty (o, nome) {return false;},// todas as propriedades existem e não são confundíveis, então nósNão posso definir maisdefineProperty (o, nome, desc) {return false;},

// Com efeito, isso significa que o objeto não é extensível (o) {return false;},// Todas as propriedades já estão definidas neste objeto, então não poderia// herdar qualquer coisa, mesmo que tenha um protótipo
objeto.getPrototypeOf (o) {return null;},// O objeto não é extensível, então não podemos mudar o protótipo
setPrototypeOf (o, proto) {return false;},});identity.x // => "x"identity.toString // => "ToString"identidade [0] // => "0"identity.x = 1;// A definição de propriedades não tem efeito
identity.x // => "x"Excluir identity.x // => false: Não é possível excluir propriedades também
identity.x // => "x"Object.Keys (identidade);//! RangeError: não consigo listar todos os chaves para (deixe p de identidade);//! RangeError Objetos de proxy podem derivar seu comportamento do objeto de destino e de os manipuladores se opõem e os exemplos que vimos até agora usaram um objeto ou outro. Mas geralmente é mais útil definir proxies que usam os dois objetos. O código a seguir, por exemplo, usa proxy para criar um somente leitura invólucro para um objeto de destino. Quando o código tenta ler valores do objeto, essas leituras são encaminhadas para o objeto de destino normalmente. Mas se qualquer código tenta modificar o objeto ou suas propriedades, métodos do objeto manipulador joguem um TypeError. Um proxy como esse pode ser útil para escrever testes: suponha que você tenha escrito uma função que leva um objeto argumento e deseja garantir que sua função não faça nenhum

Erro ao traduzir esta página.

* Retornar um objeto de proxy que envolve o, delegando tudo operações para* Esse objeto após registrar cada operação.objName é aFaça isso* aparecerá nas mensagens de log para identificar o objeto.Seo tem próprio* propriedades cujos valores são objetos ou funções, então se você perguntar* O valor dessas propriedades, você receberá um LoggingProxyde volta, para que isso* O comportamento de registro desse proxy é "contagioso".*/função loggingproxy (o, objName) {// Defina manipuladores para o nosso objeto proxy de registro.// Cada manipulador registra uma mensagem e depois delega para o objeto alvo.manipuladores const = {// este manipulador é um caso especial porque para o próprio propriedade// cujo valor é um objeto ou função, ele retorna umProxy, em vez disso// do que retornar o próprio valor.Get (Target, Property, Receiver) {// Logre a operação GetConsole.log ('Handlerget (\$ {objName}, \$ {Property.toString ()}) `');// Use a API refletir para obter o valor da propriedadeDeixe o valor = refletir.get (destino, propriedade,receptor); // se a propriedade é uma propriedade própria do alvo e// O valor é um objeto ou função e retorne um proxy para isso.if (reflete.ownskeys (Target) .includes (Propriedade) &&(tipo de valor === "objeto" || TIPO DE VALOR === "function") {Retornar LoggingProxy (valor,\$ {objName}. \$ {Property.toString ()}`);}// de outra forma, retorne o valor não modificado.valor de retorno;

```
},// não há nada de especial nos três seguintesMétodos:// eles registram a operação e delegados ao alvoobjeto.//  
eles são um caso especial simplesmente para que possamos evitarregistrando o// objeto receptor que pode causar  
infinitoRecursão.SET (Target, Prop, Valor, Receptor) {Console.log (` Handlerset ($ {objName}, $ {prop.toString ()}, $  
{value}) `);retorno reflete.Set (Target, Prop, Valor,receptor);},aplicar (alvo, receptor, args) {console.log  
(` manipulador $ {objName} ($ {args}) `);retorno reflete.Apply (Target, Receiver, Args);},construto (alvo, args,  
receptor) {console.log (` manipulador $ {objName} ($ {args}) `);retorno reflete.Construct (Target, args, receptor);}//  
Podemos gerar automaticamente o resto domanipuladores// metaprograma ftw!Reflete.ownskeys (refletir)  
.ForEach (HandlerName => {if (! (nome do manipulador em manipuladores)) {Manipuladores [nome do  
manutenção] = função (Target, ... args){// registrar a operaçãoconsole.log (` manipulador $ {handlername}($  
{objName}, $ {args}) `);// Delegar a operaçãoretorno reflete [handlername] (Target, ... args);}}};});
```

// retorna um proxy para o objeto usando esses registrosmanipuladoresretornar novo proxy (O, manipuladores);}A função LoggingProxy () definida anteriormente cria proxies queregistre todas as maneiras pelas quais eles são usados.Se você está tentando entender comoUma função não documentada usa os objetos que você passa, usando um registroProxy pode ajudar.Considere os seguintes exemplos, que resultam em alguns genuínosInsights sobre a iteração da matriz:// define uma matriz de dados e um objeto com uma funçãoopropriedadeixe dados = [10,20];deixe métodos = {quadrado: x => x*x};// Crie proxies de registro para a matriz e o objetodeixe proxydata = loggingproxy (dados, "dados");deixe proxymethods = loggingproxy (métodos, "métodos");// Suponha que queremos entender como o método da matriz.map ()funcionadata.map (métodos.square) // => [100, 400]// Primeiro, vamos tentar com uma matriz proxy de madeiraproxydata.map (métodos.square) // => [100, 400]// produz esta saída:// manipulador get (dados, mapa)// manipulador obtém (dados, comprimento)// manipulador get (dados, construtor)// Handler tem (dados, 0)// manipulador get (dados, 0)// Handler tem (dados, 1)// manipulador get (dados, 1)// Agora vamos tentar com um objeto de métodos de proxydata.map (proxymethods.square) // => [100, 400]// Saída de log:

```
// Manipulador Get (Métodos, quadrado)// Handler Methods.Square (10,0,10,20)// Handler Methods.Square  
(20,1,10,20)// Finalmente, vamos usar um proxy de registro para aprender sobre oProtocolo de iteraçãoopara (deixe  
x de proxydata) console.log ("datum", x);// Saída de log:// manipulador get (dados, símbolo (símbolo.iterator))//  
manipulador obtém (dados, comprimento)// manipulador get (dados, 0)// Datum 10// manipulador obtém (dados,  
comprimento)// manipulador get (dados, 1)// Datum 20// manipulador obtém (dados, comprimento)Desde o primeiro  
pedaço de saída de madeira, aprendemos que oMétodo Array.map () verifica explicitamente a existência de  
cadaelemento da matriz (fazendo com que o manipulador tenha () seja invocado) antesNa verdade, lendo o valor  
do elemento (que aciona o manipulador get ()).Presumivelmente, isso pode distinguir elementos de matriz  
inexistentesde elementos que existem, mas indefinidos.O segundo pedaço de saída de madeira pode nos lembrar  
que a funçãoPassamos para Array.map () é invocado com três argumentos: ovalor do elemento, o índice do  
elemento e a própria matriz.(Há umProblema em nossa saída de registro: o método Array.toString ()não inclui  
suportes quadrados em sua saída e as mensagens de logseria mais claro se eles fossem incluídos na lista de  
argumentos (10,0,[10,20]).)A terceira parte da saída de log nos mostra que o loop for/offunciona procurando um  
método com nome simbólico
```

[Symbol.iterator].Também demonstra que a aula de matrizA implementação deste método de iterador é cuidadosa para verificar a matrizcomprimento em cada iteração e não assume que o comprimento da matrizpermanece constante durante a iteração.14.7.1 Invariantes de procuraçãoA função readOnlyProxy () definida anteriormente cria proxyObjetos que são efetivamente congelados: qualquer tentativa de alterar um valor de propriedadeou atributo de propriedade ou para adicionar ou remover propriedadesexceção.Mas enquanto o objeto de destino não estiver congelado, descobriremos queSe pudermos consultar o proxy com reflete.isextensible () eReflet.GetownPropertyDescriptor (), e ele nos diráque devemos ser capazes de definir, adicionar e excluir propriedades.EntãoReadOnlyProxy () cria objetos em um estado inconsistente.Nóspoderia consertar isso adicionando isextensible () egetOwnPropertyDescriptor () manipuladores, ou podemos simplesmente vivercom esse tipo de inconsistência menor.A API de manipulador de proxy nos permite definir objetos com majorinconsistências, no entanto, e neste caso, a própria classe de procuração iráimpedir -nos de criar objetos de proxy que são inconsistentes em um maucaminho.No início desta seção, descrevemos proxies como objetos comnenhum comportamento próprio porque eles simplesmente encaminham todas as operações paraos manipuladores objeto e o objeto de destino.Mas isso não é totalmente verdadeiro:Depois de encaminhar uma operação, a classe proxy realiza alguma sanidadeverificações no resultado para garantir que invariantes importantes de JavaScript não sejam sendo violado.Se detectar uma violação, o proxy jogará umTypeError em vez de deixar a operação prosseguir.

Como exemplo, se você criar um proxy para um objeto não extensível, oProxy jogará um TypeError se o manipulador isoxtensible () retorna true:Deixe o destino = object.PreventExtensions ({});deixe proxy = new proxy (destino, {isextensible () {return true;}});Reflete.isextensible (proxy);//! TypeError: InvariantviolaçãoDe acordo, objetos de proxy para alvos não extensíveis podem não ter umgetPrototypeOf () manipulador que retorna qualquer coisa que não seja oObjeto de protótipo real do alvo.Além disso, se o objeto de destino tiverPropriedades não escritas e não confundíveis, então a classe de proxy iráJogue um TypeError se o manipulador get () retornar qualquer coisa que não sejaO valor real:deixe o destino = object.freeze ({x: 1});Seja proxy = novo proxy (destino, {get () {return 99;}});proxy.x;//! TypeError: Valor retornado por get ()não corresponde ao destinoProxy aplica vários invariantes adicionais, quase todos elestendo a ver com objetos-alvo não estetensivos e não-configurávelpropriedades no objeto de destino.14.8 ResumoNeste capítulo, você aprendeu:Os objetos JavaScript têm um atributo e objeto extensíveisAs propriedades têm gravidade, enumerável e configurável

atributos, bem como um valor e um atributo getter e/ou setter. Você pode usar esses atributos para "bloquear" seus objetos em várias maneiras, incluindo a criação de "selado" e "congelado" objetos. JavaScript define funções que permitem atravessar a cadeia de protótipo de um objeto e até alterar o protótipo de um objeto (embora fazer isso possa tornar seu código mais lento). As propriedades do objeto de símbolo têm valores que são "Símbolos conhecidos", que você pode usar como propriedade ou nomes de métodos para os objetos e classes que você define. Fazer isso permite controlar como seu objeto interage com recursos de linguagem JavaScript e com a biblioteca principal. Para exemplo, símbolos conhecidos permitem que você faça suas aulas iterável e controle a string que é exibida quando uma instância é passada para `object.prototype.toString()`. Antes do ES6, esse tipo de personalização estava disponível apenas para as classes nativas que foram incorporadas a uma implementação. Os literais de modelo marcados são uma sintaxe de invocação de funções e definir uma nova função de tag é como adicionar um novo literal Sintaxe ao idioma. Definir uma função de tag que analisa seu argumento da string de modelo permite incorporar DSLs dentro do Código JavaScript. As funções de tags também fornecem acesso a um cru, forma não descontada de literais de cordas onde as barras não têm significado especial. A classe de proxy e a API refletida relacionada permitem baixo nível controle sobre os comportamentos fundamentais dos objetos JavaScript. Objetos de proxy podem ser usados ?? como embalagens opcionalmente revogáveis ?? para melhorar o encapsulamento de código e também pode ser usado para implementar comportamentos de objetos não padronizados (como alguns dos APIs de casos especiais definidas pelos primeiros navegadores da web).

¹Um bug no mecanismo V8 JavaScript significa que este código não funciona corretamente no nó13.

Capítulo 15. JavaScript na WebNavegadores A linguagem JavaScript foi criada em 1994 com o propósito expressode ativar o comportamento dinâmico nos documentos exibidos pela Webnavegadores.O idioma evoluiu significativamente desde então, e noao mesmo tempo, o escopo e as capacidades da plataforma da web cresceramexplosivamente.Hoje, os programadores JavaScript podem pensar na web como umplataforma completa para desenvolvimento de aplicativos.Navegadores da webespecialize -se na exibição de texto e imagens formatados, mas como nativoSistemas operacionais, os navegadores também fornecem outros serviços, incluindoGráficos, vídeo, áudio, networking, armazenamento e encadeamento.JavaScripté o idioma que permite que os aplicativos da web usem os serviçosfornecido pela plataforma da web, e este capítulo demonstra como vocêpode usar o mais importante desses serviços.O capítulo começa com o modelo de programação da plataforma da web,Explicando como os scripts são incorporados nas páginas HTML (§15.1) eComo o código JavaScript é acionado de forma assíncrona por eventos (§15.2).As seções que seguem este material introdutório documentam o núcleoJavaScript APIs que permitem que seus aplicativos da Web:Conteúdo do documento de controle (§15.3) e estilo (§15.4)Determine a posição na tela dos elementos do documento(§15.5)

Crie componentes de interface do usuário reutilizáveis ??(§15.6)Desenhar gráficos (§15.7 e §15.8)Jogue e gerar sons (§15.9)Gerenciar navegação e história do navegador (§15.10)Trocá dados sobre a rede (§15.11)Armazene dados no computador do usuário (§15.12)Executar computação simultânea com threads (§15.13)JavaScript do lado do clienteNeste livro, e na web, você verá o termo "JavaScript do lado do cliente". O termo é simplesmente umSinônimo de JavaScript escrito para executar em um navegador da web e contrasta com o código "do lado doservidor", que é executado em servidores da Web.Os dois "lados" se referem às duas extremidades da conexão de rede que separam o servidor da web e onavegador da web e desenvolvimento de software para a web normalmente exige que o código seja escrito em ambos "Lados." O lado do cliente e o lado do servidor também são chamados de "front-end" e "back-end". Edições anteriores deste livro tentaram cobrir de maneira abrangente a todosJavascript APIs definidas pelos navegadores da web e, como resultado, este livroFoi há muito tempo há uma década.O número e a complexidade das APIs da Webcontinuou a crescer, e eu não acho mais que faz sentido tentarPara cobrir todos eles em um livro.Na sétima edição, meu objetivo é cubra a linguagem JavaScript definitivamente e para fornecer uma profundidadadeIntrodução ao uso do idioma com o nó e com os navegadores da Web.Este capítulo não pode cobrir todas as APIs da Web, mas apresenta maisimportantes com detalhes suficientes para que você possa começar a usá -los certosausente.E, tendo aprendido sobre as APIs principais cobertas aqui, vocêdeve ser capaz de pegar novas APIs (como as resumidas no §15.15)Quando e se você precisar deles.

Node tem uma única implementação e uma única fonte de autoridade para documentação APIs da web, por outro lado, são definidas por consenso. Entre os principais fornecedores de navegador da web e o autoritário A documentação assume a forma de uma especificação destinada ao C ++ programadores que implementam a API, não para o JavaScript programadores que o usarão. Felizmente, o "MDN Web Docs" de Mozilla O Project é uma fonte confiável e abrangente para a API da Web documentação APIs legadas. Nos 25 anos desde que o JavaScript foi lançado pela primeira vez, os fornecedores de navegador estão adicionando recursos e APIs para os programadores usarem. Muitas dessas APIs agora são obsoletas. Eles incluem: APIs proprietárias que nunca foram padronizadas e/ou nunca implementadas por outro navegador fornecedores. O Internet Explorer da Microsoft definiu muitas dessas APIs. Alguns (como o Propriedade Innerhtml) se mostrou útil e acabou sendo padronizada. Outros (como o Método ATPLEVENT ()) tem sido obsoleto há anos. APIs ineficientes (como o método document.write ()) que têm um impacto desempenho que seu uso não é mais considerado aceitável. APIs desatualizadas que há muito foram substituídas por novas APIs por alcançar o mesmo coisa. Um exemplo é document.bgcolor, que foi definido para permitir que JavaScript defuisse ocorre de fundo de um documento. Com o advento do CSS, document.bgcolor se tornou um caso especial pitoresco sem propósito real. APIs mal projetadas que foram substituídas por melhores. Nos primeiros dias da web, Comitês de padrões definiram a principal API do modelo de objeto de documento em um idioma agnóstico caminho para que a mesma API possa ser usada nos programas Java para trabalhar com documentos XML e em programas JavaScript para trabalhar com documentos HTML. Isso resultou em uma API que foi Não é bem adequado para a linguagem JavaScript e que possuía recursos que os programadores da web Não se importava particularmente. Levou décadas para se recuperar desses erros de design antecipados. Mas os navegadores da Web de hoje suportam um modelo de objeto de documento muito melhorado. Os fornecedores do navegador podem precisar apoiar essas APIs herdadas no futuro próximo, a fim de garantir compatibilidade com versões anteriores, mas não há mais necessidade de este livro para documentá-los ou para você aprender sobre eles. A plataforma da web amadureceu e se estabilizou, e se você é uma web experiente Desenvolvedor que se lembra da quarta ou quinta edição deste livro, então você pode ter tanto Conhecimento desatualizado para esquecer, pois você tem um novo material para aprender. 1

15.1 básicos de programação da web
Esta seção explica como os programas JavaScript para a Web são estruturado, como eles são carregados em um navegador da web, como eles obtêm entrada, como eles produzem saída e como eles correm de forma assíncrona respondendo a eventos.

15.1.1 JavaScript em tags HTML

Os navegadores da Web exibem documentos HTML. Se você quer um navegador da web para executar o código JavaScript, você deve incluir (ou referenciar) esse código de um documento HTML, e é isso que a tag HTML <Script> faz. O código JavaScript pode aparecer em linha dentro de um arquivo HTML entre <Script> e </Script> tags. Aqui, por exemplo, é um arquivo HTML que inclui uma tag de script com código JavaScript que dinamicamente atualiza um elemento do documento para fazê-lo se comportar como um relógio:

```
<!DOCTYPE html>
<!- Este é um arquivo HTML5->
<html>
  <!- o elemento raiz ->
  <head>
    <!- título, scripts e estilos ->
  </head>
  <body>
    <h1 id="clock">Relógio digital</h1>
    <script>
      // Script para atualizar o relógio
      function updateClock() {
        var now = new Date();
        var hours = now.getHours();
        var minutes = now.getMinutes();
        var seconds = now.getSeconds();

        var clockElement = document.getElementById("clock");
        clockElement.textContent = `${hours} : ${minutes} : ${seconds}`;
      }

      setInterval(updateClock, 1000);
    </script>
  </body>
</html>
```

Fronteira: Black Solid Black 2px; /* e uma borda preta sólida */ Radio de fronteira: 10px; /* Com cantos arredondados. */</style></head><Body> <!- O corpo mantém o conteúdo do documento.-><h1> relógio digital </h1> <!- exibir um título.-> <!- Vamos inserir o tempo em este elemento.-><Cript>// Defina uma função para exibir o horário atualFunção DisplayTime () {deixe clock = document.querySelector ("#relógio"); // Pegar elemento com id = "relógio" deixe agora = new Date (); // Pegar horário atualclock.textContent = agora.toLocaleTimeString (); // MostrarTempo no relógio}displaytime () // exibe o tempo certoausentesetInterval (DisplayTime, 1000); // e depois atualize -o a cada segundo.</script></body></html> Embora o código JavaScript possa ser incorporado diretamente dentro de um Tag <script>, é mais comum usar o atributo src de A tag <Script> para especificar o URL (um URL absoluto ou um URL em relação ao URL do arquivo html sendo exibido) de um arquivo contendo código JavaScript. Se tirássemos o código JavaScript desse arquivo html e o armazenou em seus próprios scripts/digital_clock.js, então o arquivo<Script> tag pode fazer referência a esse arquivo de código como este:<script src = "scripts/digital_clock.js"> </script>

Um arquivo javascript contém javascript puro, sem tags <script> ouQualquer outro html.Por convenção, os arquivos do código JavaScript têm nomesEsse fim com .js.Uma tag <script> com o atributo A SRC se comporta exatamente como se oO conteúdo do arquivo JavaScript especificado apareceu diretamente entre o<Script> e </sCript> tags.Observe que o fechamento </sCript>A tag é necessária nos documentos HTML, mesmo quando o atributo SRC éEspecificado: HTML não suporta uma tag <script/>.Há várias vantagens em usar o atributo SRC:Ele simplifica seus arquivos HTML, permitindo que você removaGrandes blocos de código JavaScript deles - ou seja, ajudaMantenha o conteúdo e o comportamento separados.Quando várias páginas da web compartilham o mesmo código JavaScript,O uso do atributo SRC permite que você mantenha apenas um únicocópia desse código, em vez de ter que editar cada arquivo htmlQuando o código muda.Se um arquivo de código JavaScript for compartilhado por mais de uma página, elesó precisa ser baixado uma vez, na primeira página que usa- Páginas subsequentes podem recuperá -lo do cache do navegador.Porque o atributo SRC toma um URL arbitrário como seu valor,Um programa JavaScript ou página da web de um servidor da web podeEmpregue código exportado por outros servidores da Web.Muita internetA publicidade depende desse fato.Módulos§10.3 documenta os módulos JavaScript e abrange sua importação e

Diretivas de exportação. Se você escreveu seu programa JavaScript usandomódulos (e não usaram uma ferramenta de construção de código para combinar todos os seusmódulos em um único arquivo não modular de javascript), então você deveCarregue o módulo de nível superior do seu programa com uma tag <cript> que tem um atributo type = "módulo". Se você fizer isso, então o módulo vocêEspecificar será carregado e todos os módulos que ele importa serão carregados,e (recursivamente) todos os módulos que eles importam serão carregados.Ver§10.3.5 Para detalhes completos.Especificando o tipo de scriptNos primeiros dias da web, pensava -se que os navegadores poderiam algunsDia implementa idiomas que não sejam JavaScript e programadoresAtributos adicionados como linguagem = "javascript" etype = "Application/JavaScript" para suas tags <Script>.Isso é completamente desnecessário.JavaScript é o padrão (e somente)linguagem da web.O atributo do idioma está depreciado e aliSão apenas dois motivos para usar um atributo de tipo em uma tag <cript>:Para especificar que o script é um móduloPara incorporar dados em uma página da web sem exibi -los (veja§15.3.4)Quando os scripts são executados: assíncronos e adiadosQuando o JavaScript foi adicionado aos navegadores da web, não havia APIpara atravessar e manipular a estrutura e o conteúdo de um jádocumento renderizado.A única maneira de o código JavaScript afetar oO conteúdo de um documento era gerar esse conteúdo em tempo real, enquanto oO documento estava em processo de carregamento.Fez isso usando o

Erro ao traduzir esta página.

O atributo assíncrono tem precedência. Observe que os scripts diferidos são executados na ordem em que aparecem no documento. Scripts assíncronos funcionam enquanto carregam, o que significa que eles podem ser executados fora de ordem. Scripts com o atributo type = "módulo" são, por padrão, executados depois que o documento foi carregado, como se eles tivessem um atributo de adiamento. Você pode substituir esse padrão pelo atributo assíncrono, que causará o código a ser executado assim que o módulo e todos os seus dependências foram carregadas. Uma alternativa simples aos atributos assíncronos e adiados - especialmente para o código que está incluído diretamente no HTML - é simplesmente colocar os scripts no final do arquivo HTML. Dessa forma, o script pode correr sabendo que o conteúdo do documento antes de ter sido analisado e pronto para ser manipulado. Carregando scripts sob demanda às vezes, você pode ter código JavaScript que não é usado quando o documento é carregado. Às vezes, é necessário se o usuário tomar alguma ação, como clicar em um botão ou abrir um menu. Se você está desenvolvendo seu código usando módulos, você pode carregar um módulo sob demanda com importação (), conforme descrito em §10.3.6. Se você não estiver usando módulos, pode carregar um arquivo de JavaScript em demanda simplesmente adicionando uma tag <script> ao seu documento quando você quer que o script carregue:

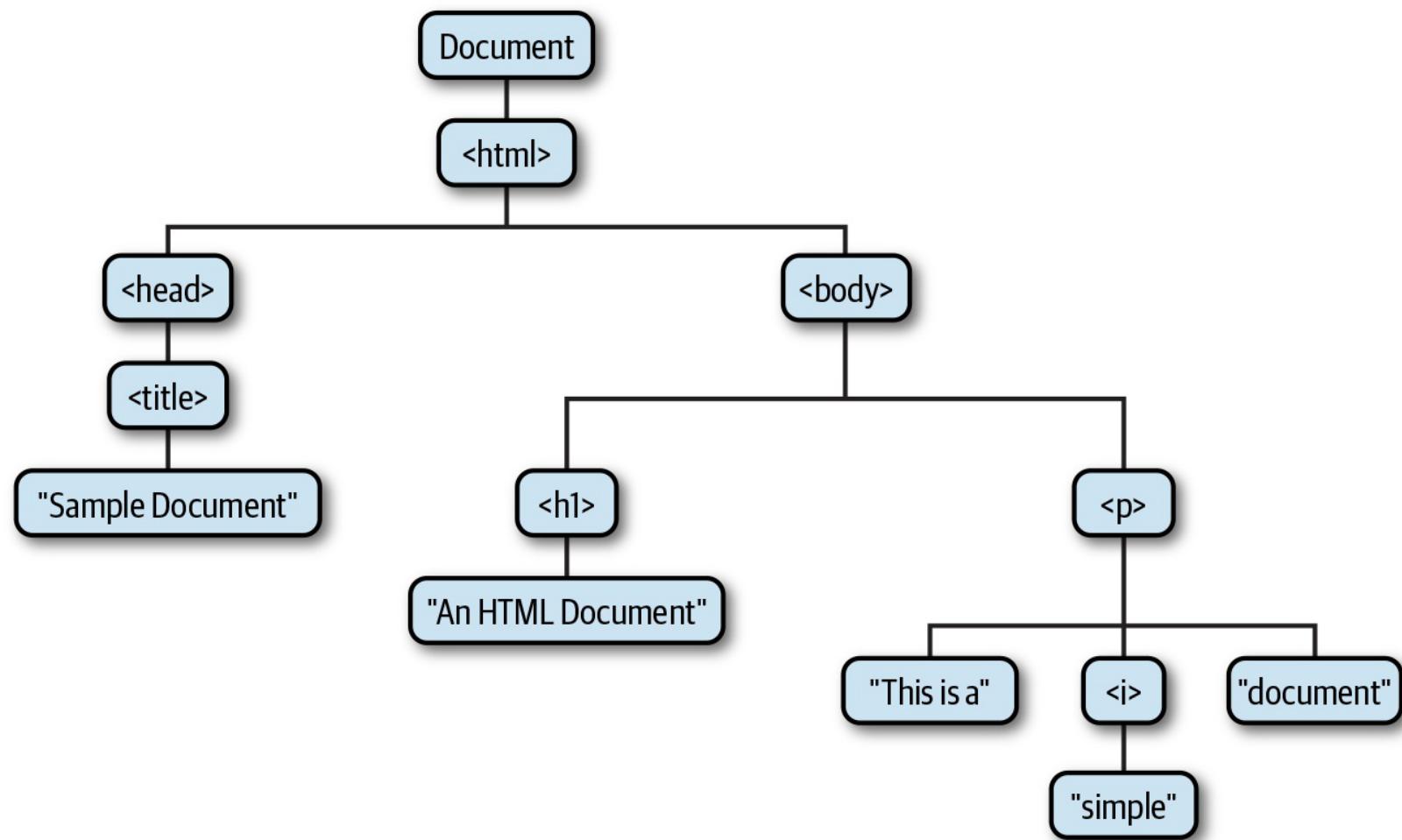
Erro ao traduzir esta página.

formando uma árvore. Considere o seguinte documento HTML simples:

```
<html><head><title> Documento de amostra </title></head><Body><H1> Um documento html </h1><p> Este é um documento <i> simples </i>.</p></body></html>
```

A tag `<html>` de nível superior contém tags `<head>` e `<body>`. O `<Head>` tag contém uma tag `<title>`. E a tag `<body>` contém `<H1>` e `<p>` tags. As tags `<title>` e `<h1>` contêm seqüências de cordas de texto e a tag `<p>` contém duas seqüências de texto com uma tag `<i>` entre eles. A API DOM reflete a estrutura da árvore de um documento HTML. Para cada tag html no documento, há um JavaScript correspondente objeto de elemento, e para cada execução de texto no documento, há um objeto de texto correspondente. As classes de elemento e texto, bem como a classe de documentos em si são todas subclasses do nó mais geral. Os objetos de classe e nó são organizados em uma estrutura de árvore que o JavaScript pode consultar e atravessar usando a API DOM. O domA representação deste documento é a árvore retratada na Figura 15-1.

Figura 15-1.A representação da árvore de um documento HTML Se você ainda não está familiarizado com as estruturas de árvores no computador



Programação, é útil saber que eles emprestam a terminologia de árvores familiares. O nó diretamente acima de um nó é o pai desse nó. Os nós um nível diretamente abaixo de outro nó são filhos de aquele nó. Nós no mesmo nível, e com o mesmo pai, são irmãos. O conjunto de nós, qualquer número de níveis abaixo de outro nó são os descendentes desse nó. E o pai, avô e todos os outros nós acima de um nó são os ancestrais desse nó. A API DOM inclui métodos para criar um novo elemento e textos, e para inseri-los no documento como filhos de outros objetos de elemento. Também existem métodos para mover elementos dentro do documento e para removê-los completamente. Enquanto o servidor aplicativo pode produzir saída de texto simples escrevendo sequências de strings com `Console.log()`, um aplicativo JavaScript do lado do cliente pode produzir saída HTML formatada construindo ou manipulando a árvore de documentos usando a API DOM. Existe uma classe JavaScript correspondente a cada tipo de tag HTML e cada ocorrência da tag em um documento é representada por uma instância da classe. A tag `<body>`, por exemplo, é representada por uma instância de `htmlBodyElement` e uma tag `<table>` é representada por uma instância de `htmlTableElement`. Os objetos do elemento JavaScript possuem propriedades que correspondem aos atributos HTML das tags. Para exemplo, instâncias de `htmlImageElement`, que representam `` tags, têm uma propriedade `SRC` que corresponde ao atributo `src` da marcação. O valor inicial da propriedade `SRC` é o valor do atributo que aparece na tag HTML e definindo esta propriedade com JavaScript altera o valor do atributo `src` (e faz com que o navegador

carregar e exibir uma nova imagem). A maioria das classes de elemento JavaScriptBasta refletir os atributos de uma tag html, mas alguns definemMétodos. As classes HtmlaudioElement e HtmlVideoElement, por exemplo, definamétodos como play () e pausa () paraControlando a reprodução de arquivos de áudio e vídeo. 15.1.3 O objeto global nos navegadores da WebExiste um objeto global por janela ou guia do navegador (§3.7). Todos os Código JavaScript (exceto código em execução em threads de trabalhadores; consulte §15.13) Em execução nessa janela compartilha esse único objeto global. Isto é verdade independentemente de quantos scripts ou módulos estão no documento: todos os scripts e módulos de um documento compartilham um único objeto global; se um script define uma propriedade nesse objeto, essa propriedade é visível a todos os outros scripts também. O objeto global é onde a biblioteca padrão de JavaScript é definida - a função parseint (), o objeto de matemática, a classe definida e assim por diante. Em Navegadores da web, o objeto global também contém os principais pontos de entrada de várias APIs da Web. Por exemplo, a propriedade do documento representa o documento atualmente exibido, o método Fetch () fabrica httpsolicitações de rede e o construtor Audio () permite JavaScript programas para jogar sons. Nos navegadores da web, o objeto global é duplo de dever: além de definindo tipos e funções internos, ele também representa a web atual Janela do navegador e define propriedades como a história (§15.10.2), que representam a história de navegação da janela, e a integridade interna, que mantém a largura da janela em pixels. Uma das propriedades deste

Objeto global é nomeado janela e seu valor é o objeto globalem si.Isso significa que você pode simplesmente digitar janela para se referir aoObjeto global no seu código do lado do cliente.Ao usar a janela específicaRecursos, geralmente é uma boa ideia incluir uma janela.prefixo:Window.innerWidth é mais claro que a integridade interna, por exemplo.15.1.4 Os scripts compartilham um espaço para nomeCom módulos, as constantes, variáveis, funções e classes definidasno nível superior (ou seja, fora de qualquer função ou definição de classe) doO módulo é privado para o módulo, a menos que sejam exportados explicitamente, emQue caso, eles podem ser importados seletivamente por outros módulos.(Observaçãoque essa propriedade dos módulos é homenageada por ferramentas de aglomeração de código comobem.)Com scripts não módulos, no entanto, a situação é completamentediferente.Se o código de nível superior em um script definir uma constante, variável,função, ou classe, essa declaração será visível para todos os outros scripts em o mesmo documento.Se um script define uma função f () e outraO script define uma classe C, então um terceiro script pode invocar a função elnstaniciar a classe sem precisar tomar nenhuma ação para importá -los.Então, se você não estiver usando módulos, os scripts independentes em seudocumento compartilhe um único espaço para nome e se comportar como se fossem todos parte deum único script maior.Isso pode ser conveniente para pequenos programas, masA necessidade de evitar a nomeação de conflitos pode se tornar problemática para maioresProgramas, especialmente quando alguns dos scripts são bibliotecas de terceiros.Existem algumas peculiaridades históricas com a forma como este espaço de nome compartilhadofunciona.Var e declarações de função no nível superior criam

Erro ao traduzir esta página.

Diferente objeto global e objeto de documento do que o código no incorporar documento e pode ser considerado um JavaScript separado programa. Lembre -se, porém, que não há uma definição formal do que os limites de um programa JavaScript são. Se o documento do contêiner ou o documento contido são carregados do mesmo servidor, o código em um documento pode interagir com o código no outro, e você pode tratá-los como duas partes que interagem de um único programa, se desejar. §15.13.6 explica como um programa JavaScript pode enviar e receber mensagens para e para o código JavaScript em execução em um <frame>. Você pode pensar na execução do programa JavaScript como ocorrendo em duas fases. Na primeira fase, o conteúdo do documento é carregado e o código de <script> elementos (scripts embutidos e scripts externos) é executado. Os scripts geralmente são executados na ordem em que aparecem no documento, embora este pedido padrão possa ser modificado pelo assíncrono e adiar atributos que descrevemos. O código JavaScript dentro de qualquer script único é executado de cima para baixo, sujeito, é claro, para condições, loops e outras declarações de controle de JavaScript. Alguns scripts realmente não fazem nada durante esta primeira fase e, em vez disso, apenas definem funções e classes para uso na segunda fase. Outros scripts podem fazer um trabalho significativo durante a primeira fase e depois não fazer nada em segundo. Imagine um script no final de um documento que encontra todos <h1> e <h2> tags no documento e modifica o documento por gerando e inserindo um índice no início do documento. Isso pode ser feito inteiramente na primeira fase. (Ver §15.3.6 Para um exemplo que faz exatamente isso.) Depois que o documento é carregado e todos os scripts executados, JavaScript

A execução entra em sua segunda fase. Esta fase é assíncrona e orientada a eventos. Se um script vai participar desta segunda fase, então uma das coisas que deve ter feito durante a primeira fase é registrar pelo menos um manipulador de eventos ou outra função de retorno de chamada que será invocado assíncrono. Durante esta segunda fase orientada a eventos, o navegador da web chama as funções do manipulador de eventos e outros retornos de chamada em resposta a eventos que ocorrem de forma assíncrona. Os manipuladores de eventos são mais comumente chamados em resposta à entrada do usuário (cliques de mouse, teclas de teclas, etc.) mas também pode ser desencadeado pela atividade da rede, documento e carregamento de recursos, tempo decorrido ou erros no código JavaScript. Eventos e os manipuladores de eventos são descritos em detalhes no §15.2. Alguns dos primeiros eventos a ocorrer durante a fase orientada a eventos são o Evento "DomContentLoaded" e "Load". "DomContentLoaded" é acionado quando o documento HTML foi completamente carregado e analisado. O evento de "carga" é acionado quando todos os documentos do documentoRecursos externos - como imagens - também estão totalmente carregados. JavaScript Os programas geralmente usam um desses eventos como um sinal de gatilho ou inicial. Isto é comum ver programas cujos scripts definem funções, mas não levam ação diferente de registrar uma função de manipulador de eventos a ser acionada pelo evento de "carga" no início da fase orientada a eventos de execução. É esse manipulador de eventos de "carga" que manipula o Documento e faça o que for que o programa deve fazer. Observe que é comum na programação JavaScript para um manipulador de eventosFunção como o manipulador de eventos "Load" descrito aqui para se registrarOutros manipuladores de eventos. A fase de carregamento de um programa JavaScript é relativamente curta: idealmente menos de um segundo. Depois que o documento é carregado, o evento orientado pelo evento

Erro ao traduzir esta página.

documentar o conteúdo, não compartilha nenhum estado com o tópico principal ou com outros trabalhadores e só pode se comunicar com o tópico principal e outros trabalhadores através de eventos de mensagens assíncronos, de modo que a concorrência não é detectável para o tópico principal, e os trabalhadores da web fazem não alterar o modelo básico de execução de thread único de javascript programas. Consulte §15.13 para obter detalhes completos sobre o encadeamento seguro da Webmecanismo. Linha do tempo do JavaScript do lado do cliente. Já vimos que os programas JavaScript começam em um script-fase de execução e, em seguida, faça a transição para uma fase de manipulação de eventos. Esses duas fases podem ser divididas nas etapas seguintes:

1. O navegador da web cria um objeto de documento e começa analisando a página da web, adicionando objetos de elemento e nós de texto para o documento enquanto ele analisa elementos html e seus textuais conteúdo. A propriedade Document.ReadyState tem o valor ?carregamento? nesta fase.
2. Quando o analisador html encontra uma tag <script> que não possui nenhum dos assíncronos, adiados, ou TIPE = atributos "módulo", ele acrescenta essa tag de script ao Documento e, em seguida, execute o script. O script é executado síncrono, e o analisador html faz uma pausa enquanto o script Downloads (se necessário) e execuções. Um script como esse pode usar document.write () para inserir texto no fluxo de entrada, e esse texto se tornará parte do documento quando o analisador é retomado. Um script como esse muitas vezes simplesmente define funções e registros de manipuladores de eventos para uso posterior, mas pode atravessar e manipular a árvore de documentos como ela existe naquele tempo. Isto é, scripts que não são do módulo que não têm um assíncrono ou adiar o atributo pode ver sua própria tag <script> e

Erro ao traduzir esta página.

objeto.8. A partir deste momento, os manipuladores de eventos são invocados de forma assíncronaEm resposta a eventos de entrada do usuário, eventos de rede, temporizadorExpira, e assim por diante.15.1.6 Entrada e saída do programaComo qualquer programa, os programas JavaScript do lado do cliente processam dados de entradaPara produzir dados de saída.Há uma variedade de insumos disponíveis:O conteúdo do próprio documento, que o código JavaScript podeAcesso com a API DOM (§15.3).Entrada do usuário, na forma de eventos, como cliques de mouse (outoques de tela de toque) em elementos html <button> ou textoEntrou em elementos HTML <Textarea>, por exemplo. §15.2 demonstra como os programas JavaScript podem responder aEventos de usuário como esses.O URL do documento que está sendo exibido está disponível paraJavaScript do lado do cliente como document.url.Se você passar issostring para o construtor url () (§11.9), você pode acessar facilmenteAs seções do caminho, consulta e fragmento do URL.O conteúdo do cabeçalho da solicitação de "cookie" http está disponívelpara o código do lado do cliente como document.cookie.Cookies sãoGeralmente usado pelo código do lado do servidor para manter as sessões de usuário,Mas o código do lado do cliente também pode ler (e escrevê-los) senecessário.Consulte §15.12.2 para obter mais detalhes.A propriedade Global Navigator fornece acesso alnformações sobre o navegador da web, o sistema operacional está em execuçãode e as capacidades de cada um.Por exemplo,Navigator.UserAgent é uma string que identifica a webNavegador, Navigator.Language é preferido pelo usuário

Erro ao traduzir esta página.

A função Window.onerror será invocada com três stringargumentos.O primeiro argumento para Window.onerror é uma mensagemdescrevendo o erro.O segundo argumento é uma string que contém oURL do código JavaScript que causou o erro.O terceiro argumento é o número da linha dentro do documento em que o erro ocorreu.Se o OnError Handler retorna true, diz ao navegador que o manipuladorlidou com o erro e que nenhuma ação adicional é necessária - em outras palavras, o navegador não deve exibir sua própria mensagem de erro.Quando uma promessa é rejeitada e não há função .Catch () paralidar com isso, essa é uma situação como uma exceção não tratada: umErro imprevisto ou um erro lógico em seu programa.Você pode detectar isso definindo uma janela.onunhandledReject Função ouusando window.addeventListener () para registrar um manipulador paraEventos de ?rejeição não entrega?.O objeto de evento passou para este manipuladorterá uma propriedade de promessa cujo valor é o objeto de promessa querejeitado e uma propriedade pela qual a propriedade cujo valor é o que teria sidoPassado para uma função .catch ().Como nos manipuladores de erros descritosAntes, se você ligar para o PreventDefault () na rejeição não enfrentadaobjeto de evento, ele será considerado tratado e não causará um erromensagem no console do desenvolvedor.Não é frequentemente necessário definir o OnError ouOs manipuladores de rejeição de OnUnHandled, mas podem ser bastante úteis como ummechanismo de telemetria se você deseja relatar erros do lado do cliente aoservidor (usando a função fetch () para fazer uma solicitação de postagem http,por exemplo) para que você possa obter informações sobre erros inesperadosIsso acontece nos navegadores de seus usuários.

15.1.8 O modelo de segurança da webO fato de as páginas da web podem executar o código JavaScript arbitrário em seu dispositivo pessoal tem implicações de segurança claras e fornecedores de navegador trabalham duro para equilibrar dois objetivos concorrentes:Definindo APIs poderosas do lado do cliente para ativar a Web útilAplicaçõesImpedindo que o código malicioso leia ou altere seus dados, comprometendo sua privacidade, enganando você ou desperdiçando seu tempoAs subseções a seguir fornecem uma rápida visão geral da segurançaRestrições e questões que você, como um programador JavaScript, deveriam estar ciente de.O que JavaScript não pode fazerA primeira linha de defesa dos navegadores da web contra código malicioso é que elesSimplesmente não suporta certos recursos.Por exemplo, lado do clienteO JavaScript não fornece nenhuma maneira de escrever ou excluir arquivos arbitrários ouListe os diretórios arbitrários no computador cliente.Isso significa aO programa JavaScript não pode excluir dados ou plantar vírus.Da mesma forma, o JavaScript do lado do cliente não tem uso geralRecursos de rede.Um programa JavaScript do lado do cliente pode fazerSolicitações HTTP (§15.11.1).E outro padrão, conhecido comoWebSockets (§15.11.3), define uma API do tipo soquete para comunicação com servidores especializados.Mas nenhuma dessas APIs permite acesso à rede mais ampla.Clientes de internet de uso geral eOs servidores não podem ser gravados no JavaScript do lado do cliente.

A política da mesma origemA política da mesma origem é uma restrição de segurança abrangente sobre qual código JavaScript de conteúdo pode interagir.Normalmente entra em jogoQuando uma página da web inclui elementos <frame>.Nesse caso, oA política da mesma origem governa as interações do código JavaScript em umenquadrar com o conteúdo de outros quadros.Especificamente, um script pode lerSomente as propriedades de janelas e documentos que têm o mesmoorigem como o documento que contém o script.A origem de um documento é definida como o protocolo, o host e o porto deo URL a partir do qual o documento foi carregado.Documentos carregadosDe diferentes servidores da Web têm origens diferentes.Documentos carregadosAtravés de diferentes portas do mesmo host, têm origens diferentes.E aDocumento carregado com o http: o protocolo tem uma origem diferente deum carregado com o https: protocolo, mesmo que eles vierem domesmo servidor da web.Os navegadores normalmente tratam todos os arquivos: URL como umorigem separada, o que significa que se você estiver trabalhando em um programa queExibe mais de um documento do mesmo servidor, você não podeser capaz de testá -lo localmente usando o arquivo: URLs e terá que executar umServidor da Web estático durante o desenvolvimento.É importante entender que a origem do próprio script não érelevante para a política da mesma origem: o que importa é a origem dodoocumento no qual o script está incorporado.Suponha, por exemplo, queUm script hospedado pelo host A está incluído (usando a propriedade SRC de um<SCRIPT> elemento) em uma página da web servida pelo host B. A origem deEsse script é o host B e o script tem acesso total ao conteúdo dodoocumento que o contém.Se o documento contiver um <frame> que

contém um segundo documento do Host B, e o script também tem completoacesso ao conteúdo desse segundo documento.Mas se o nível superiorO documento contém outro <frame> que exibe um documento deHost C (ou mesmo um do Host A), então a política da mesma origem vemEm efeito e impede que o script acesse este documento aninhado.A política da mesma origem também se aplica às solicitações HTTP com script (ver§15.11.1).O código JavaScript pode fazer solicitações http arbitrárias para o servidor da web a partir do qual o documento contendo foi carregado, masnão permite que os scripts se comuniquem com outros servidores da web (a menosEsses servidores da Web optam por CORS, como descrevemos a seguir).A política da mesma origem apresenta problemas para grandes sites que usamVários subdomínios.Por exemplo, scripts com origemorders.example.com pode precisar ler propriedades de documentos emexemplo.com.Para suportar sites multidomânticos desse tipo, os scripts podemAlterar sua origem definindo document.Domain para um sufixo de domínio.Então, um script com origem https://orders.example.com pode alterar sua origempara https://example.com, configurando document.domain para "Explet.com."Mas esse script não pode definir document.Domain para "Ordens.example", "ample.com" ou "com".A segunda técnica para relaxar a política da mesma origem é cruzadaCompartilhamento de Recursos de Origin, ou CORS, que permite aos servidores decidirquais origens eles estão dispostos a servir.CORS estende http com umNova origem: Cabeçalho de solicitação e um novo controle de acessoCabeçalho de resposta de origem da autorização.Ele permite que os servidores usem um cabeçalho paraListe explicitamente as origens que podem solicitar um arquivo ou usar um curinga e

Deixe um arquivo ser solicitado por qualquer site. Os navegadores honram esses corscabeçalhos e não relaxam restrições da mesma origem, a menos que seja presente. Script de câmara cruzadaScripts cross sites, ou XSS, é um termo para uma categoria de problemas de segurança em que um atacante injeta tags ou scripts HTML em um site de destino. Os programadores JavaScript do lado do cliente devem estar cientes e defenderContra, scripts de sites cruzados. Uma página da web é vulnerável a scripts cruzados se ela dinamicamente gerar conteúdo de documentos e bases desse conteúdo em substituição de usuários dados sem primeiro "higienizar" esses dados removendo qualquerTags html a partir dele. Como exemplo trivial, considere a seguinte webpágina que usa JavaScript para cumprimentar o usuário pelo nome:<Cript>Deixe o nome = novo URL (document.url).searchparams.get ("nome"); Document.QuerySelector ('H1'). Innerhtml = "Hello" + Name;</script>Este script de duas linhas extrai a entrada do parâmetro de consulta "nome" O URL do documento. Em seguida, ele usa a API DOM para injetar um htmlString na primeira tag <H1> no documento. Esta página pretendesse invocado com um URL como este:<http://www.example.com/greet.html?name=david> Quando usado assim, ele exibe o texto "Olá David". Mas considere o que acontece quando é invocado com este parâmetro de consulta:

Nome =%3cimg%20src =%22x.png%22%20onload =%22Alert (%27Hacked%27)%22/%3eQuando os parâmetros escapados pelo URL são decodificados, este URL causa o Após o HTML a ser injetado no documento:Olá Após a carga da imagem, a sequência de JavaScript no atributo OnLoadé executado.A função Global Alert () exibe um diálogo modalcaixa.Uma única caixa de diálogo é relativamente benigna, mas demonstra queA execução do código arbitrária é possível neste site porque exibeHTML não senitado.Os ataques de script de sites são assim chamados porque mais de um site éenvolvido.O site B inclui um link especialmente criado (como o do exemplo anterior) para o site A. Se o Site B puder convencer os usuários a clicar noLink, eles serão levados para o site A, mas esse site agora estará executando o códigoNo site B. Esse código pode definir a página ou causardefeituoso.Mais perigosamente, o código malicioso poderia ler cookiesarmazenado pelo site A (talvez números de conta ou outro pessoalmenteidentificação de informações) e enviar esses dados de volta ao site B. o injetadoO código pode até rastrear as teclas do usuário e enviar esses dados de volta paraLocal B.Em geral, a maneira de impedir ataques XSS é remover tags HTMLDe qualquer dados não confiáveis ??antes de usá -los para criar um documento dinâmicoconteúdo.Você pode corrigir o arquivo greet.html mostrado anteriormente substituindocaracteres HTML especiais na sequência de entrada não confiável com seusEntidades HTML equivalentes:

nome = nome.place (///g, "& amp;").Place (/</g, "& lt;").place (/>/g, "& gt;").Place (/"/g, "'").place (/& // g, "/")Outra abordagem do problema do XSS é estruturar sua webaplicativos para que o conteúdo não confiável seja sempre exibido em um<frame> com o atributo Sandbox definido para desativar scripts eOutros recursos.O script entre sites é uma vulnerabilidade perniciosa cujas raízes vão profundamente arquitetura da web.Vale a pena entender issovulnerabilidade em profundidade, mas uma discussão mais aprofundada está além do escopo deeste livro.Existem muitos recursos on -line para ajudá -lo a se defenderScript de câmara cruzada.15.2 EventosOs programas JavaScript do lado do cliente usam um evento assíncrono orientadomodelo de programação.Nesse estilo de programação, o navegador da webgera um evento sempre que algo interessante acontece com odocumento ou navegador ou para algum elemento ou objeto associado a ele.Por exemplo, o navegador da web gera um evento quando terminarCarregando um documento, quando o usuário move o mouse sobre um hiperlink,ou quando o usuário atinge uma chave no teclado.Se um JavaScripto aplicativo se importa com um tipo específico de evento, pode registrar um ouMais funções a serem invocadas quando ocorrem eventos desse tipo.Observe quelsso não é exclusivo da programação da web: todos os aplicativos com gráfico

As interfaces de usuário são projetadas dessa maneira - elas ficam esperando para serem interagidas com (ou seja, elas esperam os eventos ocorrerem) e depois respondem. No JavaScript do lado do cliente, os eventos podem ocorrer em qualquer elemento dentro de um documento HTML, e esse fato torna o modelo de evento da Web Navegadores significativamente mais complexos do que o modelo de evento do Node.js. Nós Começamos esta seção com algumas definições importantes que ajudam a explicar esse modelo de evento:

Tipo de evento: Esta string especifica qual tipo de evento ocorreu. O tipo "mousemove", por exemplo, significa que o usuário moveu o mouse. O tipo "keydown" significa que o usuário pressionou uma tecla no teclado para baixo. E o tipo "carga" significa que um documento (ou algum outro recurso) terminou o carregamento da rede.

Como o tipo de evento é apenas uma string, às vezes é chamado de nome de evento e, de fato, usamos esse nome para identificar o tipo de evento sobre o qual estamos falando.

Alvo de eventos: Este é o objeto em que o evento ocorreu ou com o qual o evento está associado. Quando falamos de um evento, devemos especificar tanto o tipo quanto o alvo. Um evento de carga em uma janela, por exemplo, ou um evento de clique em um elemento <button>. Janelas, objetos de documentos e elementos são as metas de eventos mais comuns em aplicativos JavaScript do lado do cliente, mas alguns eventos são acionados em outros tipos de objetos. Por exemplo, um objeto de trabalhador (um tipo de Tópico, coberto §15.13) é um alvo para eventos de "mensagem" que ocorrem quando o tópico do trabalhador envia uma mensagem para o thread principal.

Manipulador de eventos, ou ouvinte de eventos:

Esta função lida ou responde a um evento. Aplicações Registre seu manipulador de eventos funções no navegador da web, especificando um tipo de evento e um alvo de eventos. Quando um evento do tipo especificado ocorre no alvo especificado, o navegador chama a função manipuladora. Quando os manipuladores de eventos são invocados para um objeto, dizemos que o navegador "demitiu", "acionado" ou "Despacho" o evento. Existem várias maneiras de se registrar manipuladores de eventos, e os detalhes do registro de manipuladores e a invocação é explicada em §15.2.2 e §15.2.3. O objeto de evento Este objeto está associado a um evento específico e contém detalhes sobre esse evento. Os objetos de evento são passados ?? como argumento para a função do manipulador de eventos. Todos os objetos de evento têm uma propriedade de tipo que especifica o tipo de evento e uma propriedade de destino que especifica o alvo de eventos. Cada tipo de evento define um conjunto de propriedades para o seu objeto de evento associado. O objeto associado a um evento do mouse inclui as coordenadas do ponteiro do mouse, por exemplo, e o objeto associado a um evento de teclado contém detalhes sobre a tecla que foi pressionada e as teclas modificadoras que foram retidas. Muitos tipos de eventos definem apenas algumas propriedades padrão - como digite e alvo - e não carrega muito outro informação. Para esses eventos, é a simples ocorrência do evento, não os detalhes do evento, esse assunto. Propagação de eventos Este é o processo pelo qual o navegador decide qual se opõe gatilho manipuladores de eventos ligados. Para eventos específicos para um único objeto - como o evento de "carga" no objeto da janela ou um evento de ?mensagem? em um objeto de trabalhador - nenhuma propagação é necessária. Mas quando certos tipos de eventos ocorrem em elementos dentro do documento html, no entanto, eles se propagam ou "bubble". Árvore de documentos. Se o usuário mover o mouse sobre um hiperlink, o evento de mousemove é o primeiro disparado no elemento <a> que define que?

link. Então é disparado sobre os elementos contendo: talvez a <p>elemento, um elemento <Section> e o próprio objeto de documento. Isto às vezes é mais conveniente para registrar um único manipulador de eventos em um documento ou outro elemento de contêiner do que registrar manipuladores em cada elemento individual em que você está interessado. Um manipulador de eventos pode parar a propagação de um evento para que não continue a bubble e não aciona manipuladores no contendo elementos. Os manipuladores fazem isso invocando um método do objeto de evento. Em outra forma de propagação de eventos, conhecida como captura de eventos, os manipuladores registrados especialmente em elementos de contêiner têm oportunidade de interceptar (ou "capturar") eventos antes de serem entregues ao seu alvo real. Eventos borbulhando e captura são cobertos em detalhes em §15.2.4. Alguns eventos têm ações padrão associadas a eles. Quando um cliqueEvento ocorre em um hiperlink, por exemplo, a ação padrão é para o Navegador para seguir o link e carregar uma nova página. Os manipuladores de eventos podem evitar essa ação padrão, invocando um método do objeto de evento. Isso às vezes é chamado de "cancelar" o evento e é coberto em §15.2.5.15.2.1 Categorias de eventos O JavaScript do lado do cliente suporta um número tão grande de tipos de eventos que não é possível neste capítulo cobrir todos eles. Pode ser útil, embora, agrupar eventos em algumas categorias gerais, para ilustrar o escopo e ampla variedade de eventos suportados: Eventos de entrada dependentes do dispositivo Esses eventos estão diretamente ligados a um dispositivo de entrada específico, como o mouse ou teclado. Eles incluem tipos de eventos como

"Mousedown", "Mousemove", "MouseUp", "Touchstart", "Touchmove", "Touchend", "Keydown" e "KeyUp".Eventos de entrada independentes do dispositivoEsses eventos de entrada não estão diretamente ligados a um dispositivo de entrada específico.O evento "clique", por exemplo, indica que um link ou botão (ououtro elemento de documento) foi ativado.Isto geralmente é feito viaum clique do mouse, mas também pode ser feito pelo teclado ou (no toque-dispositivos sensíveis) com uma torneira.O evento de "entrada" é um dispositivoAlternativa independente ao evento "KeyDown" e suportaentrada do teclado, bem como alternativas como corte e colarMétodos de entrada usados ??para scripts ideográficos.O "ponteiro down".Os tipos de eventos "Pointermove" e "Pointerup" são independentes de dispositivosalternativas para o mouse e tocar eventos.Eles trabalham para o tipo de mousePonteiros, para telas de toque, e para entrada no estilo de caneta ou canetabem.Eventos de interface do usuárioEventos de interface do usuário são eventos de nível superior, geralmente em elementos de forma HTMLIsso define uma interface do usuário para um aplicativo da Web.Eles incluem oEvento "Focus" (quando um campo de entrada de texto ganha foco no teclado), oEvento de "mudança" (quando o usuário altera o valor exibido por umelemento do formulário) e o evento "enviar" (quando o usuário clica em umEnviar botão em um formulário).Eventos de mudança de estadoAlguns eventos não são acionados diretamente pela atividade do usuário, mas poratividade de rede ou navegador e indica algum tipo de ciclo de vida oumudança relacionada ao estado.Os eventos "Carregar" e "DomContentLoaded"?Filado na janela e no documento objetos, respectivamente, noFim do carregamento de documentos - provavelmente é o mais comumente usadoDesses eventos (consulte ?Linha do tempo do JavaScript do lado do cliente?).NavegadoresFire eventos ?online? e ?offline? no objeto da janela quando

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

O argumento do evento significa que seu código de manipulador pode se referir ao objeto de evento atual como evento. O com declarações significa que o código do seu manipulador pode se referir às propriedades do objeto de destino, o contendo <form> (se houver) e o objeto de documento que contém diretamente, como se fossem variáveis ?? no escopo. A declaração é proibida no modo rigoroso (§5.6.3), mas o código JavaScript em HTML atributos nunca são rigorosos. Os manipuladores de eventos definidos dessa maneira são executados em um ambiente em que variáveis ?? inesperadas são definidas. Isso pode ser uma fonte de bugs confusos e é um bom motivo para evitar escrevendo manipuladores de eventos em html.AddEventListener () Qualquer objeto que possa ser um alvo de eventos - isso inclui a janela e Documentar objetos e todos os elementos do documento - define um método nomeado addEventListener () que você pode usar para registrar um evento manipulador para esse alvo.addEventListener () leva três argumentos. O primeiro é o tipo de evento para o qual o manipulador está sendo registrado. O tipo de evento (ou nome) é uma string que não inclui o prefixo ?on? usado ao definir propriedades de manipulador de eventos. O segundo argumento para addEventListener () é a função que deve ser invocada quando o tipo de evento especificado ocorre. O terceiro argumento é opcional e é explicado abaixo. O código a seguir registra dois manipuladores para o evento "clique" em um <button> elemento. Observe as diferenças entre as duas técnicas usadas:

<botão id = "myButton"> clique em mim </button><Cript>Seja b = document.querySelector("#myButton");B.OnClick = function () {Console.log ("Obrigado por clicarmeu!"); };B.AddeventListener ("Click", () => {Console.log ("Obrigadode novo!");});</script>Chamando addEventListener () com "clique" como seu primeiro argumentonão afeta o valor da propriedade OnClick.Neste código, umO clique do botão registrará duas mensagens no console do desenvolvedor.E se nóschamado addEventListener () primeiro e depois definido oClick, nósainda registraria duas mensagens, exatamente na ordem oposta.MaisÉ importante ressaltar que você pode ligar para addEventListener () várias vezes paraRegistre mais de uma função de manipulador para o mesmo tipo de evento nomesmo objeto.Quando um evento ocorre em um objeto, todos os manipuladoresRegistrado para esse tipo de evento é invocado na ordem em que elesforam registrados.Invocando addEventListener () mais de uma vezno mesmo objeto com os mesmos argumentos não tem efeito - o manipuladorA função permanece registrada apenas uma vez e a invocação repetidaõ altera a ordem em que os manipuladores são invocados.addEventListener () é emparelhado com umRemoneeventListener () Método que espera os mesmos doisArgumentos (além de um terceiro opcional), mas remove um manipulador de eventosfunção de um objeto em vez de adicioná -lo.Muitas vezes é útilregistre temporariamente um manipulador de eventos e remova -o em brevedepois.Por exemplo, quando você recebe um evento "Mousedown", vocêpode registrar manipuladores de eventos temporários para ?mousemove? eEventos "MouseUp" para que você possa ver se o usuário arrasta o mouse.

Erro ao traduzir esta página.

O ouvinte será removido automaticamente após o acionado uma vez. Se issoA propriedade é falsa ou é omitida, então o manipulador nunca é removido automaticamente. Se o objeto de opções tiver uma propriedade passiva definida como true, indica que o manipulador de eventos nunca ligará para preventDefault () para cancelarA ação padrão (consulte §15.2.5). Isso é particularmente importante para o toqueEventos em dispositivos móveis - se os manipuladores de eventos para eventos "touchmove" pode impedir a ação de rolagem padrão do navegador, depois o navegador não pode implementar rolagem suave. Esta propriedade passiva fornece uma maneira de registrar um manipulador de eventos potencialmente disruptivo desse tipo, mas informe o navegador da web sabe que ele pode começar com segurança seu comportamento padrão- Como rolar - enquanto o manipulador de eventos está em execução. SuaveA rolagem é tão importante para uma boa experiência do usuário que Firefox e Chrome Make "Touchmove" e "Mousewheel" eventos passivos por padrão. Então, se você realmente deseja registrar um manipulador que liga preventDefault () para um desses eventos, você deve explicitamenteDefina a propriedade passiva como falsa. Você também pode passar em um objeto de opções para removeventListener (), Mas a propriedade de captura é a única que é relevante. Não há precisar especificar uma vez ou passivo ao remover um ouvinte e Essas propriedades são ignoradas. 15.2.3 Invocação do manipulador de eventosDepois de registrar um manipulador de eventos, o navegador da web invocará ele automaticamente quando um evento do tipo especificado ocorre no objeto especificado. Esta seção descreve a invocação do manipulador de eventos em

Erro ao traduzir esta página.

Eventos, por exemplo, têm propriedades ClientX e ClientY que especificam as coordenadas da janela nas quais o evento ocorreu. Contexto de manipulador de eventos Quando você registra um manipulador de eventos definindo uma propriedade, parece que você está definindo um novo método no objeto de destino: Target.OnClick = function () { / * Código do manipulador */ }; Não é surpreendente, portanto, que os manipuladores de eventos sejam invocados como métodos do objeto em que eles são definidos. Isto é, dentro do corpo de um manipulador de eventos, a palavra -chave refere-se ao objeto em que o manipulador de eventos foi registrado. Os manipuladores são invocados com o alvo como esse valor, mesmo quando registrado usando addEventListener(). Isso não funciona para manipuladores definidos como funções de seta, no entanto: Funções de seta sempre tenham o mesmo valor que o escopo em que são definidos. Valor de retorno do manipulador No JavaScript moderno, os manipuladores de eventos não devem devolver nada. Você pode ver os manipuladores de eventos que retornam valores no código mais antigo e o retorno é tipicamente um sinal para o navegador de que não deve executar ação padrão associada ao evento. Se o manipulador de um clique de um enviar o botão em um formulário retorna false, por exemplo, depois a web navegador não enviará o formulário (geralmente porque o manipulador de eventos determinou que a entrada do usuário falha na validação do lado do cliente). A maneira padrão e preferida de impedir o navegador de

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

A API de evento do JavaScript do lado do cliente é relativamente poderosa, e você pode usá-lo para definir e despachar seus próprios eventos. Suponha, para exemplo, que seu programa precisa periodicamente realizar um longo cálculo ou faça uma solicitação de rede e que, embora esta operação seja pendente, outras operações não são possíveis. Você quer deixar o usuário saber sobre isso exibindo "spinners" para indicar que a aplicação está ocupada. Mas o módulo que está ocupado não deve precisar saiba onde os spinners devem ser exibidos. Em vez disso, esse módulo pode simplesmente despachar um evento para anunciar que está ocupado e depois despacha outro evento quando não está mais ocupado. Então, o módulo da interface do usuário pode registrar manipuladores de eventos para esses eventos e levar qualquer UIAs ações são apropriadas para notificar o usuário. Se um objeto JavaScript tiver um método addEventListener () é um "alvo de eventos", e isso significa que também possui um expediente () método. Você pode criar seu próprio objeto de evento com o CustomEvent () construtor e passa para despachoEvent (). O primeiro argumento para CustomEvent () é uma string que especifica o tipo do seu evento, e o segundo argumento é um objeto que especifica as propriedades do objeto de evento. Defina a propriedade detalhada deste objeto a uma string, objeto ou outro valor que represente o conteúdo de seu evento. Se você planeja despachar seu evento em um elemento de documento e quero borbulhar a árvore de documentos, adicione bolhas: fiel ao segundo argumento:// Despacha um evento personalizado para que a interface do usuário saiba que estamos ocupadosdocument.dispatchEvent (novo customevent ("ocupado", {detalhe: true}));// Execute uma operação de rede

buscar (url).THEN (HandleNetworkResponse).Catch (HandleNetWorkError).Finalmente () => {// após a solicitação de rede ter conseguido ou falhar, expedição// Outro evento para deixar a interface do usuário saber que não somos mais ocupado.document.dispatchEvent (novo customevent ("ocupado", {Detalhe: false}));};// em outros lugares, em seu programa, você pode registrar um manipulador para eventos "ocupados"// e use -o para mostrar ou ocultar o spinner para deixar o usuário saber.document.addEventListener ("ocupado", (e) => {if (e.detail) {showspinner ();} outro {Hidespinner ();}});15.3 Documentos de script JavaScript do lado do cliente existe para transformar documentos HTML estáticos em Aplicativos da Web interativos. Portanto, roteirizar o conteúdo das páginas da web é realmente o objetivo central do JavaScript. Cada objeto de janela tem uma propriedade de documento que se refere a um objeto de documento. O objeto de documento representa o conteúdo da janela e é o assunto desta seção. O objeto de documento faz não está sozinho, no entanto. É o objeto central no DOM para representando e manipulando o conteúdo do documento.

O DOM foi introduzido em §15.1.2. Esta seção explica a API em detalhe. Cobre:

- Como consultar ou selecionar elementos individuais de um documento.
- Como atravessar um documento e como encontrar os ancestrais, irmãos e descendentes de qualquer elemento de documento.
- Como consultar e definir os atributos dos elementos do documento.
- Como consultar, definir e modificar o conteúdo de um documento.
- Como modificar a estrutura de um documento criando, inserindo e excluindo nós.

15.3.1 Selecionando elementos do documento

Os programas JavaScript do lado do cliente geralmente precisam manipular um ou mais elementos dentro do documento. A propriedade `document` global refere-se ao objeto do documento, e o objeto de documento tem cabeça e corpo propriidades que se referem aos objetos do elemento para o `<head>` e `<body>` tags, respectivamente. Mas um programa que deseja manipular um elemento incorporado mais profundamente no documento deve de alguma forma obter ou selecionar os objetos do elemento que se referem a esses elementos do documento.

Selecionando elementos com seletores CSS As folhas de estilo CSS têm uma sintaxe muito poderosa, conhecida como seletores, para descrever elementos ou conjuntos de elementos em um documento. O domMétodos `querySelector()` e `querySelectorAll()` permitem-nós para encontrar o elemento ou elementos em um documento que corresponda ao Seletor CSS especificado. Antes de cobrirmos os métodos, começaremos com um tutorial rápido sobre sintaxe de seletor CSS.

Os seletores CSS podem descrever elementos por nome da tag, o valor de seu idatributo, ou as palavras em seu atributo de classe:div // qualquer elemento <div>#nav // O elemento com id = "Nav".warning // qualquer elemento com "aviso" em seuatributo de classeO caractere # é usado para corresponder com base no atributo ID e no.O caractere é usado para corresponder com base no atributo de classe.Elementos podem também ser selecionado com base em valores de atributo mais gerais:p [lang = "fr"] // um parágrafo escrito em francês: <plang = "fr">*[name = "x"] // qualquer elemento com um nome = "x"atributoObserve que esses exemplos combinam um seletor de nome de tag (ou a * tagNome Wildcard) com um seletor de atributos.Combinações mais complexastambém são possíveis:span.fatal.error // qualquer com "fatal" e"Erro" em sua classespan [lang = "fr"]. Aviso // qualquer em francês com aula"aviso"Os seletores também podem especificar a estrutura do documento:#log span // qualquer descendente doelemento com id = "log"#log> span // qualquer filho do elementocom id = "log"corpo> h1: primeiro filho // o primeiro <h1> filho do <body>img + p.caption // a <p> com a classe "Legenda"Imediatamente depois de um <mg>h2 ~ p // qualquer <p> que segue um <H2> eé um irmão disso

Se dois seletores forem separados por vírgula, significa que selecionamosElementos que correspondem a um dos seletores:botão, entrada [type = "button"] // all <butto> e <entrdatatype = "Button"> elementosComo você pode ver, os seletores de CSS nos permitem referir a elementos dentro de umdocumento por tipo, id, classe, atributos e posição dentro dodoocumento.O método querySelector () leva um seletor CSSstring como seu argumento e retorna o primeiro elemento correspondente noDocumento que ele encontra ou retorna nulo se nenhum corresponde:// Encontre o elemento de documento para a tag html com atributoid = "Spinner"deixe spinner = document.querySelector ("#spinner");querySelectorAll () é semelhante, mas retorna todas as correspondênciasElementos no documento, em vez de apenas retornar o primeiro:// Encontre todos os objetos de elemento para <H1>, <H2> e <H3> tagsLet Titles = Document.QuerySelectorAll ("H1, H2, H3");O valor de retorno de querySelectorAll () não é uma variedade deObjetos de elemento.Em vez disso, é um objeto semelhante a uma matriz conhecido como umNodelist.Os objetos nodelistas têm uma propriedade de comprimento e pode serindexados como matrizes, para que você possa percorrer -las com um tradicional paralaço.Os nodelistas também são iteráveis, para que você possa usá -los com/deloops também.Se você deseja converter uma lista de nodelas em uma verdadeira matriz,Basta passar para Array.From ().A lista de nodelas retornada por querySelectorAll () terá um

Propriedade de comprimento definida como 0 se não houver nenhum elemento no documento que correspondem ao seletor especificado. querySelector () e querySelectorAll () são implementados pela classe de elementos, bem como pela classe de documentos. Quando invocado em um elemento, esses métodos retornarão apenas elementos que são descendentes desse elemento. Observe que o CSS define :: Primeira linha e :: Primeira letra pseudoelementos. No CSS, essas partes correspondem aos nós de texto em vez de elementos reais. Eles não corresponderão se usados ?? com querySelectorAll () ou QuerySelector (). Além disso, muitos navegadores se recusarão a retornar partidas para o: link e: visitado pseudoclasses, pois isso pode expor informações sobre o usuário. História de navegação. Outro método de seleção de elementos baseado em CSS é mais próximo (). Esse método é definido pela classe de elemento e toma um seletor como seu único argumento. Se o seletor corresponde ao elemento em que é invocado, ele retorna esse elemento. Caso contrário, ele retorna o elemento ancestral mais próximo que o seletor corresponde ou retorna nulo se nenhum corresponde. Em certo sentido, mais próximo () é o oposto de queryselector (): mais próximo () começa em um elemento e procura uma partida acima dele na árvore, enquanto querySelector () começa com um elemento e procura uma partida abaixo dele na árvore. Mais próximo () pode ser útil quando você tiver registrado um manipulador de eventos em um nível alto na árvore de documentos. Se você estiver lidando com um evento de "clique", por exemplo, você pode querer saber Seja um clique em um hiperlink. O objeto de evento lhe dirá o que

alvo era, mas esse alvo pode ser o texto dentro de um link em vez doA própria tag do Hyperlink.Seu manipulador de eventos pode procurar omais próximo contendo hiperlink assim:// Encontre a etiqueta de gabinete mais próxima que tem um hrefatributo.deixe hyperlink = event.target.closest ("a [href]");Aqui está outra maneira de usar mais próximo ():// retorna true se o elemento e estiver dentro de uma lista HTMLelementofunção insidelist (e) {retornar e.closest ("ul, ol, dl")! == null;}O método relacionado corresponde () não retorna ancestrais ouDescendentes: simplesmente testa se um elemento é correspondido por um CSSseletor e retorna true se for e false, caso contrário:// retorna true se E for um elemento de cabeçalho HTMLfunção isheading (e) {retornar e.matches ("H1, H2, H3, H4, H5, H6");}Outros métodos de seleção de elementosAlém de queryselector () e querySelectorAll (),O DOM também define vários métodos de seleção de elementos mais antigosque são mais ou menos obsoletos agora.Você ainda pode ver alguns delesMétodos (especialmente getElementById ()) em uso, no entanto:// Procure um elemento por id.O argumento é apenas o id,sem// O prefixo seletor CSS #.Semelhante a

Document.querySelector ("#Sect1")Seja sect1 = document.getElementById ("Sect1");// Procure todos os elementos (como caixas de seleção de formulário) que têm um nome = "cor"// atributo.Semelhante ao document.querySelectorAll ("*[name = "color"] ");Let Colors = Document.getElementsByName ("Color");// Procure todos os elementos <H1> no documento// semelhante ao document.querySelectorall ("H1")Let Headings = Document.getElementsByTagName ("H1");// getElementsByTagName () também é definido em elementos// Obtenha todos os elementos <H2> dentro do elemento SECT1.Seja subtítulos = sect1.getElementsByTagName ("h2");// Procure todos os elementos que têm a classe "ToolTip"// semelhante ao document.querySelectorAll (" . ToolTip")Let ToolTips = Document.getElementsByClassName ("Tooltip");// Procure todos os descendentes da seção que têm a classe "barra lateral"// semelhante ao sect1.QuerySelectorAll (" . Barra lateral")Let Barras laterais = sect1.getElementsByClassName ("barra lateral");Como querySelectorAll (), os métodos neste código retornam um NodeList (exceto getElementById (), que retorna um único objeto de elemento).Ao contrário de querySelectorAll (), no entanto, os nodelistas retornados por esses métodos de seleção mais antigos são "vivos", o que significa que o comprimento e o conteúdo da lista podem mudar se o documento alterar suas alterações de conteúdo ou estrutura.Elementos pré-selecionadosPor razões históricas, a classe de documentos define propriedades de atalho para acessar certos tipos de nós.As imagens, formas e linksPropriedades, por exemplo, fornecem fácil acesso ao , <input>, e <a> elementos (mas apenas <a> tags que têm um atributo href) de um

documento. Essas propriedades se referem a objetos `htmlcollection`, que são muito parecidos com objetos `nodelistas`, mas também podem ser indexados por ID do elemento ou nome. Com a propriedade `Document.Forms`, para exemplo, você pode acessar a tag `<form id = "endereço">` como `document.forms.address`. Uma API ainda mais desatualizada para selecionar elementos é o `Document.All` Property, que é como uma `HtmlCollection` para todos os elementos no documento. `Document.All` está depreciado e você não deve mais usá-lo.

15.3.2 Estrutura de documentos e travessia

Depois de selecionar um elemento de um documento, às vezes você precisa encontrar porções estruturalmente relacionadas (pai, irmãos, filhos) do documento. Quando estamos interessados principalmente nos elementos de um documento em vez do texto dentro deles (e o espaço em branco entre eles, que também é texto), há uma API de travessia que nos permite tratar um documento como uma árvore de objetos de elemento, ignorando nós de texto que são também parte do documento. Esta API de travessia não envolve nenhum método; é simplesmente um conjunto de propriedades em objetos de elemento que permitem:

- Nós: nos referimos aos pais, filhos e irmãos de um determinado elemento;
- `parentnode`: Esta propriedade de um elemento refere-se aos pais do elemento, que será outro elemento ou um objeto de documento.
- `crianças`: Esta lista de nodos contém os filhos de um elemento, mas

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Uma lista de nodel de somente leitura que contém todas as crianças (não apenasCrianças de elemento) do nó.FirstChild, LastChildO primeiro e o último filho dos nós de um nó, ou nulo se o nó não tivercrianças.Nextsibling, anteriorsiblingOs próximos e anteriores nós de irmãos de um nó.Essas propriedadesConecete nós em uma lista duplamente vinculada.NodeTypeUm número que especifica que tipo de nó é esse.Nós do documentoter valor 9. Os nós do elemento têm valor 1. Os nós de texto têm valor3. Os nós de comentários têm valor 8.NodValueO conteúdo textual de um nó de texto ou comentário.NodenameO nome da tag html de um elemento, convertido em maiúsculas.Usando essas propriedades de nó, o segundo nó filho do primeiro filho deO documento pode ser referido com expressões como estas:Document.ChildNodes [0] .ChildNodes
[1]document.firstChild.firstChild.nextSiblingSuponha que o documento em questão seja o seguinte:<html> <head> <title> teste </title> </head> <body> olá mundo!</body> </html>

Então o segundo filho do primeiro filho é o elemento <body>. Tem umNodeType de 1 e um nome noden do "corpo". Observe, no entanto, que esta API é extremamente sensível a variações no Texto do documento. Se o documento for modificado inserindo um único nova linha entre a tag <html> e a <head>, por exemplo, o Nó de texto que representa que a Newline se torna o primeiro filho do Primeiro filho, e o segundo filho é o elemento <head> em vez do <body> elemento. Para demonstrar essa API Traversal baseada em nó, aqui está uma função que Retorna todo o texto dentro de um elemento ou documento:// retorna o conteúdo de texto simples do elemento e, recolocando-se em elementos filhos.// Este método funciona como a propriedade TextContentFunção TextContent (e) {Seja s = ""; // Acumula o texto aqui para (deixe criança = e.firstChild; filho! == null; criança = filho = Child.NextSibling) {Seja tipo = Child.nodeType; if (type === 3) { // se for um texto nós += Child.NodeValue; // Adicione o texto conteúdo para nossa string. } else if (type === 1) { // se for um Nó do elementos += textContent (criança); // então recorrente. } } retorno s;} Esta função é apenas uma demonstração - na prática, você simplesmente

Erro ao traduzir esta página.

Let Image = Document.querySelector ("#main_image");deixe url = image.src;// O atributo SRC é o URL dea imagemimage.id === "main_image" // => true;Nós procuramos a imagempor idDa mesma forma, você pode definir os atributos de submissionamento de formulário de <form>Elemento com código como este:Seja f = document.querySelector ("formulário");// primeiro <morm>no documentof.action = "https://www.example.com/submit";// defina o URLpara enviá -lo para.f.method = "post";// defina o httpTipo de solicitação.Para alguns elementos, como o elemento <input>, alguns htmlNomes de atributo Mapa para propriedades de nome diferente.O htmo atributo de valor de um <input>, por exemplo, é refletido peloPropriedade JavaScript DefaultValue.A propriedade JavaScript Valuedo elemento <input> contém a entrada atual do usuário, mas mudapara a propriedade Value não afeta a propriedade DefaultValue nemo atributo de valor.Os atributos HTML não são sensíveis ao maiúsculas, mas nomes de propriedades JavaScriptsão.Para converter um nome de atributo para a propriedade JavaScript, escreva -o emminúsculo.Se o atributo for mais de uma palavra de comprimento, no entanto, coloque oPrimeira letra de cada palavra após a primeira em maiúsculas:DefaultChecked e Tabindex, por exemplo.Manipulador de eventosPropriedades como OnClick são uma exceção, no entanto, e são escritas emminúsculo.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

No DOM, os objetos de elemento têm uma propriedade de conjunto de dados que se refere a um objeto que possui propriedades que correspondem aos atributos de dados com o prefixo removido. Assim, `DataSet.x` manteria o valor do atributo `Data-X`. Mapas de atributos hifenizados para camelcase nomes de propriedades: o número de seção de dados do atributo se torna `oPropriedade DataSet.SectionNumber`. Suponha que um documento HTML contenha este texto: `<h2 id = "title" seção de dados-number = "16.1"> atributos </h2>` Então você pode escrever JavaScript como este para acessar esse número de seção: Deixe o número `=document.querySelector("#title").`

`DataSet.SectionNumber;` 15.3.4 Conteúdo do elemento
Veja novamente a árvore de documentos na Figura 15-1 e pergunte: você mesmo qual é o "conteúdo" do elemento `<p>`. Existem duas maneiras. Podemos responder a esta pergunta: O conteúdo é uma string HTML? Isso é um `<i>` simples `</i>` documento". O conteúdo é a sequência de texto simples? Este é um simples documento". Ambas são respostas válidas, e cada resposta é útil em seu próprio caminho. As seções a seguir explicam como trabalhar com a HTML Representação e a representação de texto simples do conteúdo de um elemento.

Conteúdo do elemento como htmlLer a propriedade Innerhtml de um elemento retorna o conteúdo desse elemento como uma sequência de marcação. Definindo esta propriedade em um O elemento chama o analisador do navegador da web e substitui o elemento Conteúdo atual com uma representação analisada da nova string. Você pode Teste isso ao abrir o console do desenvolvedor e digitar: document.body.innerHTML = "<h1> oops </h1>"; Você verá que toda a página da web desaparece e é substituída por O título único, "oops". Os navegadores da web são muito bons em analisar HTML e a configuração do InnerHTML geralmente são bastante eficientes. Observação, No entanto, esse texto anexo à propriedade Innerhtml com o += O operador não é eficiente porque requer uma etapa de serialização para converter o conteúdo do elemento em uma string e depois uma etapa de análise para converter a nova string de volta ao conteúdo do elemento. AVISO Ao usar essas APIs HTML, é muito importante que você nunca insira o usuário entrada no documento. Se você fizer isso, você permite que usuários maliciosos injetem seu próprio scripts em seu aplicativo. Consulte "Scripts de crosso-sites" para obter detalhes. A propriedade OuterHtml de um elemento é como Innerhtml, exceto que seu valor inclui o próprio elemento. Quando você consulta OuterHtml, o valor inclui as tags de abertura e fechamento do elemento. E quando você define o OuterHtml em um elemento, o novo O conteúdo substitui o próprio elemento.

Um método de elemento relacionado é `insertAdjacentHTML()`, que permite que você insira uma série de marcação html arbitrária "adjacente" ao elemento especificado. A marcação é passada como o segundo argumento para este método, e o significado preciso de "adjacente" depende do valor do primeiro argumento. Este primeiro argumento deve ser uma string com um dos valores "Antes Begin", "Afterbegin", "Antes end", ou "Depois." Esses valores correspondem a pontos de inserção que são ilustrados na Figura 15-2.

Figura 15-2. Pontos de inserção para `insertAdjacentHTML()`

Conteúdo do elemento como texto simples

Às vezes você deseja consultar o conteúdo de um elemento como texto simples ou para inserir texto simples em um documento (sem ter que escapar do ângulo). Suportes e ampeiros usados ?? na marcação HTML.

A maneira padrão de fazer isso é com a propriedade `textContent`:

```
Seja para = document.querySelector("p");// primeiro <p> node
document.createTextNode = "Hello World!";// alterar o texto de o parágrafo
A propriedade textContent é definida pela classe Node, para que funcione para nós de texto e nós de elementos. Para nós de elementos, ele encontra e retorna todo o texto em todos os descendentes do elemento.
```

```
<div id="target">This is the element content</div>
  ↑          ↑          ↑          ↑
beforebegin afterbegin beforeend afterend
```

A classe de elemento define uma propriedade InnerText que é semelhante a TextContent.InnerText tem alguns incomuns e complexos comportamentos, como tentar preservar a formatação da tabela. Não está bem especificado nem implementado de forma compatível entre os navegadores, no entanto, e não deve mais ser usado. Texto em elementos <Script> Elementos em linha <Cript> (ou seja, aqueles que não têm um atributo SRC) têm uma propriedade de texto que você pode usar para recuperar seu texto. O conteúdo de um elemento <Script> nunca é exibido pelo navegador, e o analisador HTML ignora suportes de ângulo e amperantes dentro de um script. Isso faz um <Script> Elemento Um local ideal para incorporar dados textuais arbitrários para uso pelo seu aplicativo. Simplesmente defina o atributo de tipo do elemento para algum valor (como "Texto/X-Custom-Data") que o deixa claro que o script não é executável JavaScript Code. Se você fizer isso, o intérprete JavaScript irá ignorar o script, mas o elemento existirá na árvore de documentos e sua propriedade de texto retornará os dados para você.

15.3.5 Criação, inserção e exclusão de nós

Vimos como consultar e alterar o conteúdo do documento usando seqüências de strings de HTML e de texto simples. E também vimos que podemos atravessar um Documento para examinar o elemento individual e os nós de texto que é feito de. Também é possível alterar um documento no nível do indivíduo nos. A classe de documento define métodos para criar elementos e objetos de elemento e texto têm métodos para inserção, excluindo e substituindo nós na árvore. Crie um novo elemento com o método createElement () do Documento a classe e anexar seqüências de texto ou outros elementos a ele com seus métodos Append () e Prepend ():

```
Seja parágrafo = document.createElement ("p");// Crie um elemento vazio <p>
```

deixe ênfase = document.createElement ("em");// Crie umElemento vazio ênfase.append ("mundo");// Adicionar texto ao elemento parágrafo.Append ("Hello", ênfase "!");// Adicionar texto e para <p>parágrafo.Prepend ("i");// Adicione mais textono início de <p>paragraph.innerHTML // => "iHello mundo ! "append () e precend () tome qualquer número de argumentos quepode ser objetos ou strings do nó.Arguments de string são automaticamenteconvertido em nós de texto.(Você pode criar nós de texto explicitamente comdocument.createTextNode (), mas raramente há motivo parafaça isso.) Append () adiciona os argumentos ao elemento no final doLista de crianças.Agenda () adiciona os argumentos no início da lista de crianças.Se você deseja inserir um elemento ou nó de texto no meio docontendo a lista infantil do elemento, depois nem appeê () ouApresença () funcionará para você.Nesse caso, você deve obter umreferência a um nó de irmão e ligue antes () para inserir o novoconteúdo antes desse irmão ou depois () para inseri -lo após esse irmão.Por exemplo:// Encontre o elemento de cabeçalho com Class = "Saudações"Let cumprimentos = document.querySelector ("H2.Greetings");// agora insira o novo parágrafo e uma regra horizontal depoisquele cabeçalhoSaudações.Como append () e precend (), depois () e antes () pegarqualquer número de argumentos de string e elemento e insira todos eles em

Erro ao traduzir esta página.

parágrafo.remove ();A API DOM também define uma geração mais antiga de métodos para inserção e remoção de conteúdo.appendChild (),insertbefore (), replacechild () e removechild () são mais difíceis de usar do que os métodos mostrados aqui e nunca devem ser necessários.15.3.6 Exemplo: gerando um índiceExemplo 15-1 mostra como criar dinamicamente um índice para um documento.Demonstra muitos dos scripts de documentosTécnicas descritas nas seções anteriores.O exemplo está bem comentado e você não deve ter problemas para seguir o código.Exemplo 15-1.Gerando um índice com a API DOM/** Toc.js: Crie um índice para um documento.** Este script é executado quando o evento DomContentLoaded é disparado* gera automaticamente um índice para o documento.* Não define nenhum símbolo global, então não deve conflitar* com outros scripts.** Quando esse script é executado, ele primeiro procura um elemento de documento com* Um ID de "Toc".Se não existe esse elemento, cria um nó* Início do documento.Em seguida, a função encontra tudo <H2>através* <H6> tags, os trata como títulos de seção e cria uma tabela de* Conteúdo dentro do elemento TOC.A função adiciona seções números* para cada seção cabeçalho e envolver os títulos em nomeadosâncoras

Erro ao traduzir esta página.

Com este estilo definido, você pode ocultar (e depois mostrar) um elemento com código como este:// Suponha que esse elemento "ToolTip" tenha class = "oculto" em arquivo html:// Podemos torná-lo visível assim:Document.querySelector ("#ToolTip").classList.remove ("Hidden");// e podemos esconder novamente assim:Document.querySelector ("#ToolTip").classList.add ("Hidden");15.4.2 Estilos embutidosPara continuar com o exemplo da dica de ferramenta anterior, suponha que o documento está estruturado com apenas um único elemento da dica de ferramenta, e queremosPara posicioná-lo dinamicamente antes de exibi-lo.Em geral, não podemoscriar uma classe de folha de estilo diferente para cada posição possível do Tooltip, para que a propriedade da lista de classe não nos ajude no posicionamento.Nesse caso, precisamos escrever o atributo de estilo da dica de ferramentaElemento para definir estilos embutidos específicos para esse elemento.ODom define uma propriedade de estilo em todos os objetos de elementos que correspondem ao atributo de estilo.Ao contrário da maioria dessas propriedades, no entanto, oPropriedade de estilo não é uma string.Em vezObjeto: uma representação analisada dos estilos CSS que aparecem em textualforma no atributo de estilo.Para exibir e definir a posição de nossoDistima de ferramentas hipotética com JavaScript, podemos usar código assim:Função Displayat (ToolTip, X, Y) {Tooltip.style.display = "bloco";Tooltip.style.Position = "Absolute";Tooltip.style.left = `\\$ {x} px`;

Tooltip.style.top = `\\$ {y} px`;)Convenções de nomenclatura: Propriedades do CSS em JavaScriptMuitas propriedades do estilo CSS, como o tamanho da fonte, contêm hífens em seus nomes. Em JavaScript, a O hífen é interpretado como um sinal de menos e não é permitido em nomes de propriedades ou outros identificadores. Portanto, os nomes das propriedades do objeto CSSSTYLEDECLARATION são ligeiramente diferentes dos nomes das propriedades reais do CSS. Se um nome de propriedade CSS contiver um ou mais hífens, o O nome da propriedade cssstyleDeclaration é formado removendo os hífens e capitalizando a letra imediatamente após cada hífen. A largura da propriedade da propriedade CSS é acessada por meio de A propriedade JavaScript BordleftWidth, por exemplo, e a propriedade CSS-Font-Family é Escrito como Fontfamily em JavaScript. Ao trabalhar com as propriedades de estilo da cssstyledeclarationObjeto, lembre -se de que todos os valores devem ser especificados como strings. Em um SHILLESHEET ou atributo de estilo, você pode escrever: exibição: bloco; Font-Family: Sans-Serif; cor de fundo: #ffffff; Para realizar a mesma coisa para um elemento e com JavaScript, você tem que citar todos os valores: e.style.display = "bloco"; e.style.fontfamily = "sans-serif"; e.style.backgroundColor = "#ffffff"; Observe que os semicolons saem das cordas. Estes são apenas normais JavaScript Semicolons; Os semicolons que você usa nas folhas de estilo CSS são Não é necessário como parte dos valores da string que você definiu com JavaScript. Além disso, lembre -se de que muitas propriedades do CSS requerem unidades como "PX" para pixels ou "pt" para pontos. Portanto, não é correto definir o

Erro ao traduzir esta página.

do objeto CSSSTYLEDECLARATION:// Copie os estilos embutidos do elemento E para o elemento F:F.SetAttribute ("Style", E.GetAttribute ("Style"));// ou faça assim:f.style.csStext = e.style.csStext;Ao consultar a propriedade de estilo de um elemento, lembre -se de que representa apenas os estilos embutidos de um elemento e que a maioria dos estilos paraA maioria dos elementos é especificada nas folhas de estilo e não em linha.Além disso, os valores que você obtém ao consultar a propriedade de estilos usarão quaisquer unidades e qualquer formato de propriedade de atalho realmente usado no atributo html, e seu código pode ter que fazerAlguns sofisticados analisando para interpretá -los.Em geral, se você quiserConsulte os estilos de um elemento, você provavelmente deseja o estilo calculado,que é discutido a seguir.15.4.3 Estilos computadosO estilo calculado para um elemento é o conjunto de valores de propriedade que oO navegador deriva (ou calcula) do estilo embutido do elemento, além de todosRegras de estilo aplicáveis ??em todas as folhas de estilo: é o conjunto de propriedadesrealmente usado para exibir o elemento.Como estilos embutidos, estilos computados são representados com um objeto CSSStyleDeclaration.Ao contrário de embutidoOs estilos, no entanto, os estilos computados são somente leitura.Você não pode definir issoEstilos, mas o objeto CSSStyleDECLARATION CULUTADO para um elementoPermite determinar quais propriedades de estilo valores o navegador usou quandorenderizando esse elemento.Obter o estilo calculado para um elemento com o

`getComputedStyle ()` Método do objeto de janela.O primeiroArgumento para este método é o elemento cujo estilo calculado édesejado.O segundo argumento opcional é usado para especificar um CSSpseudoelemento, como ">:: antes" ou ">:: depois":
`deixe title = document.querySelector ("#Section1Title");Let Styles = Window.getComputedStyle (título);Let beforestyles = window.getComputedStyle (título, "::antes");`O valor de retorno do `getComputedStyle ()` é umObjeto `cssstyleDeclaration` que representa todos os estilos que se aplicam ao elemento especificado (ou pseudoelemento).Existem váriosdiferenças importantes entre um objeto `CSSStyleDeclaration` querepresenta estilos embutidos e um que representa estilos computados:As propriedades do estilo computado são somente leitura.As propriedades de estilo computado são absolutas: unidades relativas comoPorcentagens e pontos são convertidos em valores absolutos.Qualquerpropriedade que especifica um tamanho (como um tamanho de margem ou uma fontetamanho) terá um valor medido em pixels.Este valor será umstring com um sufixo "px", então você ainda precisará analisá -lo, masvocê não precisará se preocupar em analisar ou converter outrosunidades.Propriedades cujos valores são cores serão devolvidos emFormato "rgb ()" ou "rgba ()".As propriedades de atalho não são calculadas - apenas o fundamentalPropriedades em que elas são baseadas são.Não consulte a `margempropriedade`, por exemplo, mas use `marginleft`, `margintop`,e assim por diante.Da mesma forma, não consulte a fronteira ou mesmolargura de fronteira.Em vez `BordertopWidth`, e assim por diante.

A propriedade CSSTEXT do estilo computado é indefinida.Um objeto CSSStyleDeclaration retornado pelo getComputedStyle ()geralmente contém muito mais informações sobre um elemento do que oCssstyleDeclaration obtido da propriedade em estilo em linhaelemento.Mas os estilos computados podem ser complicados, e consultá -losNem sempre fornece as informações que você pode esperar.Considere oAtributo da família de fontes: aceita uma lista separada por vírgulaFamílias de fontes para portabilidade entre plataformas.Quando você consulta oPropriedade da Fontfamília de um estilo computado, você está simplesmente recebendo ovalor do estilo mais específico da família de fontes que se aplica aoelemento.Isso pode retornar um valor como "Arial, Helvetica, Sans-Serif".O que não diz qual tipo de tipo de tipo está realmente em uso.Da mesma forma, seUm elemento não está absolutamente posicionado, tentando consultar sua posiçãoe tamanho através das propriedades superior e esquerda de seu estilo calculadofrequentemente retorna o valor automático.Este é um valor de CSS perfeitamente legal, masProvavelmente não é o que você estava procurando.Embora CSS possa ser usado para especificar com precisão a posição e o tamanho deelementos de documentos, consultar o estilo calculado de um elemento não éA maneira preferida de determinar o tamanho e a posição do elemento.Ver§15.5.2 Para uma alternativa mais simples e portátil.15.4.4 folhas de estilo de scriptAlém de atributos de classe de script e estilos embutidos, o JavaScript podetambém manipular as folhas de estilo.As folhas de estilo estão associadas aUm documento HTML com uma tag <yo> ou com um <linkrel = "STILELES SHEET"> TAG.Ambos são tags HTML regulares, então

você pode dar a eles os dois atributos de identificação e depois procurá -los comDocument.querySelector ().Os objetos do elemento para tags <yo> e <link> têm umPropriedade desativada que você pode usar para desativar toda a folha de estilo.Você pode usá -lo com código assim:// Esta função muda entre o "luz" e "escuro"temasfunção TogGleTheme () {Deixe LightTheme = Document.querySelector ("#Light-Them");Let DarkTheme = Document.querySelector ("#Dark-Them");if (DarkTheme.Disabled) {// atualmente leve,Mude para o escuroLightTheme.disabled = true;DarkTheme.Disabled = false;} else {// atualmente escuro,Mude para a luzLightTheme.disabled = false;DarkTheme.Disabled = true;}}Outra maneira simples de folhas de estilo de script é inserir novas noDocumento usando as técnicas de manipulação de DOM que já vimos.Por exemplo:Função setTheMe (nome) {// Crie um novo elemento <link rel = "STILESSHEET">A folha de estilo nomeadaLet Link = Document.createElement ("Link");link.id = "tema";link.rel = "STILELES SHEET";link.href = `temas/\${name}.css`;// Procure um link existente com o ID "tema"

deixe currentTheme = document.querySelector ("#tema");if (Currenttheme) {// Se houver um tema existente, substitua -o pelo novo.CurrentTheme.ReplaceWith (link);} outro {// caso contrário, basta inserir o link para o temafolha de estilo.document.head.append (link);} Menos sutilmente, você também pode apenas inserir uma sequência de html contendo um<estilo> tag no seu documento.Este é um truque divertido, por exemplo:document.head.insertAdjacentHTML ("Antes do rend", "<style> corpo {transform: rotate (180deg)} </style>");Os navegadores definem uma API que permite que o JavaScript olhe para dentrofolhas de estilo para consultar, modificar, inserir e excluir regras de estilo nissofolha de estilo.Esta API é tão especializada que não está documentada aqui.Você pode ler sobre isso no MDN pesquisando "CSSStyleSheet" e "Modelo de objeto CSS."15.4.5 Animações e eventos CSSSuponha que você tenha as duas classes CSS a seguir definidas em umFolha de estilo:.Transparent {Opacity: 0;}.faderable {transição: opacidade .5s facilidade}Se você aplicar o primeiro estilo a um elemento, ele será totalmente transparente e

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Inclua uma propriedade AnimationName que especifica a propriedade de nome de animação que define a animação e umPropriedade de tempo decorrido que especifica quantos segundos passaram desde que a animação começou. 15.5 Geometria de documentos e rolagem Neste capítulo até agora, pensamos em documentos como abstratos árvores de elementos e nós de texto. Mas quando um navegador renderiza um documento dentro de uma janela, ele cria uma representação visual do documento no qual cada elemento tem uma posição e um tamanho. Freqüentemente, webAs aplicações podem tratar documentos como árvores de elementos e nunca precisam pensar em como esses elementos são renderizados na tela. Às vezes, No entanto, é necessário determinar a geometria precisa de um elemento. Se, por exemplo, você deseja usar o CSS para posicionar dinamicamente um elemento (como uma dica de ferramenta) ao lado de algum navegador comum-elemento posicionado, você precisa ser capaz de determinar a localização desse elemento. As seguintes subseções explicam como você pode ir e voltar entre o modelo abstrato e baseado em árvores de um documento e a visão geométrica, baseada em coordenadas do documento, como é apresentada em um Janela do navegador. 15.5.1 Documentar coordenadas e viewportCoordenadas A posição de um elemento de documento é medida em pixels CSS, com a coordenada X aumentando para a direita e a coordenada Y aumentando

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Elementos de bloco, como imagens, parágrafos e elementos <div> são sempre retangulares quando estabelecido pelo navegador. Elementos embutidos, tal como , <code> e elementos, no entanto, podem abranger múltiplas linhas e, portanto, consistem em múltiplos retângulos. Imagine, para exemplo, algum texto dentro das tags e que acontece exibido para que ele envolva duas linhas. Seus retângulos consistem no final da primeira linha e início da segunda linha. Se você ligar getBoundingClientRect () neste elemento, o limite do retângulo incluiria toda a largura de ambas as linhas. Se você quiser consultar os retângulos individuais de elementos embutidos, chame o método getClientRects () para obter um objeto de matriz somente leitura cujos elementos são objetos retangulares como os devolvidos por getBoundingClientRect ().

15.5.3 Determinando o elemento em um ponto

O método getBoundingClientRect () nos permite determinar a posição atual de um elemento em uma viewport. Às vezes queremos vá na outra direção e determine qual elemento está em um dado localização na viewport. Você pode determinar isso com o método elementFromPoint () do objeto de documento. Chame isso com as coordenadas X e Y de um ponto (usando a ViewportCoordenadas, não coordenadas de documentos: o clientX e a clientY são coordenadas de um trabalho de evento do mouse, por exemplo). ElementFromPoint () retorna um objeto de elemento que está na posição especificada. O algoritmo de detecção de acerto para selecionar o elemento não é especificado com precisão, mas a intenção desse método é que ele retorna o mais interno (mais profundamente aninhado) e o ponto mais alto (mais alto CSS Z-)

elemento do atributo de índice) nesse ponto.15.5.4 RolagemO método scrollTo () do objeto da janela leva o X e Ycoordenadas de um ponto (em coordenadas de documentos) e as definem como oOffsets de barra de rolagem.Isto é, ele rola a janela para que o especificadoO ponto está no canto superior esquerdo da viewport.Se você especificar um pontoisso é muito próximo do fundo ou muito próximo da borda direita dodoocumento, o navegador o moverá o mais próximo possível do superiorEsquerca esquerda, mas não conseguirá chegar até lá.A seguirO código rola o navegador para que a página mais inferior do documentoé visível:// Obtenha as alturas do documento e da viewport.Let DocumentHeight = Document.documentElement.offsetHeight;Seja ViewPorthEight = Window.innerHeight;// e role para que a última "página" seja exibida na viewportWindow.ScrollTo (0, DocumentHeight - ViewPorthEight);O método scrollby () da janela é semelhante ascrollto (), mas seus argumentos são relativos e são adicionados aoPosição atual de rolagem:// role 50 pixels para baixo a cada 500 ms.Observe que não há comoPara desligar isso!setInterval (() => {scrollby (0,50)}, 500);Se você deseja rolar sem problemas com scrollto () ou scrollby (),passe um único argumento de objeto em vez de dois números, como este:window.scrollto ({Esquerda: 0,

Erro ao traduzir esta página.

Para janelas do navegador, o tamanho da viewport é dado pelo `window.innerWidth` e `window.innerHeight`. Propriedades.(Páginas da web otimizadas para dispositivos móveis geralmente usam um `<meta name = "viewport">` tag em seus `<head>` para definir o desejo de largura da viewport para a página.) O tamanho total do documento é o mesmo que o tamanho do elemento `<html>`, `document.documentElement`. Você pode ligar `getBoundingClientRect()` ON `document.documentElement` para obter a largura e a altura do documento, ou você pode usar a largura de deslocamento e o peso das propriedades do `document.documentElement`. Os compensações de rolagem do documento em sua viewport está disponível como `janela.scrollx` e `window.scrollTop`. Essas são propriedades somente de leitura, então você não pode definir -os para rolar o documento: use `window.scrollTo()`. As coisas são um pouco mais complicadas para elementos. Cada elementoObjeto define os três grupos a seguir de propriedades:
`OffsetWidth ClientWidth Scrollwidth`
`OffSetHeight ClientHeight ScrollHeight`
`OffsetLeft ClientLeft ScrollLeft`
`OFFSETTOP CLIENTTOP SCROLLTOP`
`OffsetParent As`
As propriedades de largura de offset e offSetset de um elementoRetorne seu tamanho na tela nos pixels CSS. Os tamanhos retornados incluem oBorder e preenchimento do elemento, mas não margens. O `offsetleft` e As propriedades `offsetTop` retornam as coordenadas X e Y do elemento. Para muitos elementos, esses valores são coordenadas de documentos. Mas para descendentes de elementos posicionados e para alguns outros elementos, como

Como células da tabela, essas propriedades retornam coordenadas que são relativas a um elemento ancestral em vez do próprio documento. O offsetParent Property especifica qual elemento as propriedades são em relação a. Essas propriedades deslocadas são todas somente leitura. ClientWidth e ClientHeight são como offsetwidth e offsetheight, exceto que eles não incluem o tamanho da fronteira - apenas a área de conteúdo e seu preenchimento. O clientleft e o clienttop As propriedades não são muito úteis: eles retornam a horizontal e a vertical distância entre o exterior do preenchimento de um elemento e o exterior de sua fronteira. Geralmente, esses valores são apenas a largura da esquerda e superior fronteiras. Essas propriedades do cliente são todas somente leitura. Para elementos embutidos Como <i>, <code> e , todos retornam 0. scrollwidth e scrollheight retornam o tamanho de um elemento Área de conteúdo mais seu preenchimento mais qualquer conteúdo transbordante. Quando o conteúdo se encaixa na área de conteúdo sem transbordamento, essas propriedades são os mesmos que a largura do cliente e o peso do cliente. Mas quando já está transbordando, eles incluem o conteúdo transbordante e os valores de retorno maior que a largura do cliente e o peso do cliente. rollleft e scrolltop Dê o deslocamento de rolagem do conteúdo do elemento dentro do viewport do elemento. Ao contrário de todas as outras propriedades descritas aqui, rollleft e scrolltop são propriedades graváveis, e você pode definir -os para rolar o conteúdo dentro de um elemento. (Na maioria dos navegadores, os objetos de elemento também possuem métodos scrollto () e scrollby () Como o objeto da janela, mas estes ainda não estão universalmente suportados.)

15.6 Componentes da WebHTML é um idioma para marcação de documentos e define um rico conjunto deTags para esse fim.Nas últimas três décadas, tornou -se umlinguagem usada para descrever as interfaces do usuário da webAplicativos, mas tags básicas HTML, como <input> e <butto>são inadequados para designs modernos da interface do usuário.Desenvolvedores da Web são capazes defazer funcionar, mas apenas usando CSS e JavaScript para aumentar oAparência e comportamento das tags HTML básicas.Considere um usuário típicoComponente de interface, como a caixa de pesquisa mostrada na Figura 15-3.Figura 15-3.Um componente de interface do usuário da caixa de pesquisaO elemento html <input> pode ser usado para aceitar uma única linha deentrada do usuário, mas não tem como exibir ícones comoA lupa à esquerda e o cancelamento X à direita.Em ordemPara implementar um elemento moderno de interface do usuário como este para a web, nósprecisa usar pelo menos quatro elementos html: um elemento <input> paraAceite e exiba a entrada do usuário, dois elementos <mg> (ou nestecaso, dois elementos exibindo glifos unicode) e umContainer <div> elemento para segurar esses três filhos.Além disso,Temos que usar o CSS para ocultar a borda padrão do <input>elemento e defina uma borda para o contêiner.E precisamos usarJavaScript para fazer com que todos os elementos HTML funcionem juntos.Quando oO usuário clica no ícone X, precisamos de um manipulador de eventos para limpar a entrada

A large, light gray rectangular input field with the placeholder text "Search..." centered inside it.

Do elemento <input>, por exemplo. Isso é muito trabalho a fazer toda vez que você deseja exibir uma caixa de pesquisa em um aplicativo da web, e a maioria dos aplicativos da web hoje não está escrita usando HTML "cru". Em vez disso, muitos desenvolvedores da web usam estruturas como React e Angular que apoiam a criação de usuário reutilizável. Componentes de interface como a caixa de pesquisa mostrada aqui. Componentes da Web é uma alternativa nativa ao navegador para essas estruturas baseadas em três adições relativamente recentes aos padrões da Web que permitem estender HTML com novas tags que funcionam como interface de usuário reutilizável e independente de componentes. As subseções a seguir explicam como usar componentes da Web definido por outros desenvolvedores em suas próprias páginas da web e explique cada uma das três tecnologias em que os componentes da Web se baseiam e finalmente amarre todos os três em um exemplo que implementa a caixa de pesquisa elemento retratado na Figura 15-3.

15.6.1 Usando componentes da Web

Os componentes da Web são definidos no JavaScript, então para usar uma componente da web no seu arquivo HTML, você precisa incluir o arquivo JavaScript que define o componente. Porque os componentes da web são relativamente nova tecnologia, eles são frequentemente escritos como módulos JavaScript, então você pode incluir um em seu HTML como este:

```
<script type="module" src="componentes/search-box.js">
```

Os componentes da web definem seus próprios nomes de tags HTML, com a restrição importante de que esses nomes de tags devem incluir um hífen. (Esse

significa que as versões futuras do HTML podem introduzir novas tags semhífens, e não há chance de que as tags entrem em conflitocomponente da web de qualquer pessoa.) Para usar um componente da web, basta usar sua tag emSeu arquivo html:<Search-Box Placeholder = "Search ..." > </search-box>Os componentes da Web podem ter atributos como as tags HTML regulares podem;A documentação para o componente que você está usando deve dizerquais atributos são suportados.Componentes da Web não podem ser definidoscom tags de fechamento automático.Você não pode escrever <Search-box/>, paraexemplo.Seu arquivo html deve incluir a tag de abertura e oTag de fechamento.Como elementos HTML regulares, alguns componentes da web são escritos paraEspere que crianças e outras sejam escritas de tal maneira que elas nãoEspere (e não exibirá) crianças.Alguns componentes da web sãoescrito para que eles possam aceitar opcionalmente crianças rotuladas especialmente queaparecerá em chamado "slots".O componente <search-box>Na foto na Figura 15-3 e implementada no Exemplo 15-3, usa "slots"Para os dois ícones que ele exibe.Se você quiser usar um <search-box>Com ícones diferentes, você pode usar o HTML como este:<search-box></search-box>O atributo slot é uma extensão para HTML que é usada para especificarquais crianças devem ir para onde.Os nomes de slot - "esquerda" e "direita" emEste exemplo - é definido pelo componente da web.Se o componente

Você está usando slots de suporte, esse fato deve ser incluído em seu documento. Eu observei anteriormente que os componentes da web são frequentemente implementados como módulos JavaScript e pode ser carregado em arquivos HTML com um `<script type = "módulo">` tag. Você pode se lembrar do início deste capítulo, os módulos são carregados após o documento. O conteúdo é analisado, como se eles tivessem uma tag diferente. Então isso significa que um navegador da web normalmente analisa e renderiza tags como `<search-box>`. Antes de executar o código que informará o que é um `<arch-box>`. Isso é normal ao usar componentes da Web. Analisadores HTML na webOS navegadores são flexíveis e perdoam muito sobre a contribuição que não entender. Quando eles encontram uma tag de componente da web antes disso, o componente foi definido, eles adicionam um htmlelement genérico ao Dom Tree, mesmo que eles não saibam o que fazer com isso. Mais tarde, quando o elemento personalizado é definido, o elemento genérico é "atualizado" para que pareça e se comporte como desejado. Se um componente da web tiver filhos, essas crianças provavelmente serão exibidos incorretamente antes que o componente seja definido. Você pode usar isso CSS para manter os componentes da web escondidos até serem definidos: /* Torne o componente `<search-box>` invisível antes de ser definido. */ E tente duplicar seu eventual layout e tamanho para que próximo. * O conteúdo não se move quando é definido. */ Caixa de pesquisa: não (: definido) { opacidade: 0; Exibição: bloco embutido; }

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Os elementos de modelo não precisam aparecer literalmente em um html documento para ser útil. Você pode criar um modelo em seu Código JavaScript, crie seus filhos com innerhtml e depois faça tantas clones quanto necessário sem a análise acimaInnerhtml. É assim que os modelos HTML são normalmente usados ??na web Componentes e Exemplo 15-3 demonstram essa técnica.

15.6.3 Elementos personalizados

O segundo recurso do navegador da web que permite componentes da web é? Elementos personalizados?: a capacidade de associar uma classe JavaScript a umNome da tag html para que essas tags no documento sejamTransformado automaticamente em instâncias da classe na árvore Dom.OO método CustomElements.Define () leva uma tag de componente da Web nome como seu primeiro argumento (lembre -se de que o nome da tag deve incluir umhífen) e uma subclasse de htmlelement como seu segundo argumento.QualquerOs elementos existentes no documento com esse nome de tag são "atualizados" paraInstâncias recém -criadas da classe.E se o navegador analisar qualquerHTML No futuro, ele criará automaticamente uma instância da classepara cada uma das tags encontra.A classe passada para alparafements.define () deve estenderHtmlelement e não um tipo mais específico comoHtmlbuttonElement.Lembre -se do capítulo 9 que quando um javascriptClasse estende outra classe, a função do construtor deve chamar super () antes de usar essa palavra -chave, por isso se o elemento personalizadoA classe tem um construtor, deve chamar super () (sem argumentos)antes de fazer qualquer outra coisa.4

Erro ao traduzir esta página.

O analisador HTML instancia mais duas bolinhas:<Diâmetro do círculo inline = "1.2em" color = "Blue"></inline-círculo><diâmetro do círculo inline = ". 6em" ??color = "Gold"> </inline-círculo>. Quantas bolas de gude o documento contém agora?</p>Figura 15-4.Um elemento personalizado em círculo embutidoPodemos implementar este elemento personalizado <inline-circle> com o Código mostrado no Exemplo 15-2:Exemplo 15-2.O elemento personalizado <circle>CustomElements.Define ("Círculo embutido", Classe InLineCircleestende Htmlelement {/ / O navegador chama esse método quando um <circle inline>elemento// é inserido no documento.Há também umdesconectedCallback ()// que não precisamos neste exemplo.conectadoCallback () {/ / Defina os estilos necessários para criar círculosthis.style.display = "inline-block";this.style.borderradius = "50%";this.style.border = "Solid Black 1px";this.style.Transform = "Tradlatey (10%)";

The document has one marble: . The HTML parser instantiates two more marbles:  . How many marbles does the document contain now?

```
// Se ainda não houver um tamanho definido, defina um tamanho padrão// que é baseado no tamanho da fonte  
atual.if (! this.style.width) {this.style.width = "0.8em";this.style.Height = "0.8em";} // A estática observadatributes  
Property especifica qualatributos// queremos ser notificados sobre as alterações para.(Usamos umgetter aqui  
desde então// Só podemos usar "estático" com métodos.)estático ser observadotributes () {return  
["diâmetro","cor"]}; // Este retorno de chamada é invocado quando um dos atributoslistado acima// muda, quando o  
elemento personalizado é analisado pela primeira vez,ou mais tarde.AttributeChangedCallback (Nome, OldValue,  
newValue) {Switch (nome) {caso "diâmetro":// Se o atributo de diâmetro mudar, atualize oEstilos de  
tamanhothis.style.width = newValue;this.style.Height = newValue;quebrar;caso "cor":// Se o atributo de cor mudar,  
atualize a corestilothis.style.backgroundColor = newValue;quebrar;} // define propriedades JavaScript que  
correspondem aoelemento// atributos.Esses getters e setters apenas recebem e definem o subjacente//  
atributos.Se uma propriedade JavaScript estiver definida, isso define o atributo// que desencadeia uma chamada  
para attributeChangedCallback ()
```

Erro ao traduzir esta página.

matriz do elemento hospedeiro e não são visitados por DOM normalMétodos de Traversal, como `querySelector()`. Por contraste, os filhos normais e regulares de um host das sombras são às vezes referidos para como o "Light Dom". Para entender o propósito da sombra dom, imagine os elementos HTML <audio> e <video>: eles exibem um usuário não trivial interface para controlar a reprodução da mídia, mas a peça e a pausa são outros elementos da interface do usuário não fazem parte da árvore dom e não podem ser manipulados por JavaScript. Dado que os navegadores da web são projetados para exibir HTML, é natural que os fornecedores do navegador gostariam de exibir UIs internas usando HTML. De fato, a maioria dos navegadores feito algo assim há muito tempo, e a sombra dom torna uma parte padrão da plataforma da web. Encapsulamento de sombra DOM A principal característica do Shadow DOM é o encapsulamento que ele fornece. Os descendentes de uma raiz das sombras estão escondidos - e independentes de - a árvore dominante regular, quase como se estivessem em um independente documento. Existem três tipos muito importantes de encapsulamento fornecido pela Shadow DOM:

Como já mencionado, elementos na sombra dom herdam de métodos DOM regulares como `querySelectorAll()`. Quando uma raiz das sombras é criada e anexada ao seu host das sombras, ela pode ser criada em "aberto" ou modo "fechado". Uma raiz de sombra fechada está completamente selada longe e inacessível. Mais comumente, porém, raízes de sombras são criadas no modo "aberto", o que significa que o host das sombras tem uma propriedade de raiz de sombra que JavaScript pode usar para ganhar

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

exibe um* Campo de entrada de texto <input> mais dois ícones ou emoji.Porpadrão, ele exibe um* Emoji de copo de lupa (indicando pesquisa) à esquerda do campo de texto* e um X emoji (indicando cancelamento) à direita do textocampo.Isto* esconde a borda no campo de entrada e exibe uma borda ao seu redor,* criando a aparência de que os dois emoji estão dentro do entradado campo.Da mesma forma, quando o campo de entrada interno está focado,o anel de foco* é exibido em torno da <arch-box>.* Você pode substituir os ícones padrão, incluindo ou crianças* de <search-box> com slot = "esquerda" e slot = "direita"atributos.* <search-box> suporta o HTML normal desativado e ocultoatributos e* também atributos de tamanho e espaço reservado, que têm o mesmos significado para isso* elemento como eles fazem para o elemento <input>.* Eventos de entrada do elemento interno <input> borbulham e aparecer com* O campo de destino está definido para o elemento <search-box>.* O elemento dispara um evento de "pesquisa" com a propriedade detalhadadefinido para o* String de entrada atual quando o usuário clica no emoji esquerdo(a ampliação* vidro).O evento "busca" também é despachado quando o campo de texto interno* gera um evento de "mudança" (quando o texto mudou eos tipos de usuário* Retornar ou guia).** O elemento dispara um evento "claro" quando o usuário clica no emoji certo* (o x).Se nenhuma chamada de manipulador prevenirdefault () no eventoentão o elemento* Limpa a entrada do usuário quando a expedição de eventos estiver concluída.* Observe que não há propriedades Onsearch e OnClear ouAtributos:

* Manipuladores para os eventos "Pesquisa" e "Clear" só podem ser registrado com* addEventListener ()).*/classe Searchbox estende htmlelement {construtor () {super();// Invoca o construtor de superclasse;deve serprimeiro.// Crie uma árvore de sombra Dom e anexe -a a issoelemento, configuração// o valor deste.shadowroot.this.attachshadow ({mode: "aberto"});// clonar o modelo que define os descendentes efolha de estilo para// este componente personalizado e anexar esse conteúdo aa raiz da sombra.this.shadowroot.append (searchbox.template.content.cloneNode (true));// recebe referências aos elementos importantes noShadow Domthis.input = this.shadowroot.querySelector ("#input");Deixe esquerdallot =this.shadowroot.QuerySelector ('slot [name = "left"]');Let RightsLot =this.shadowroot.QuerySelector ('slot [name = "right"]');// Quando o campo de entrada interno recebe ou perde o foco,defina ou remova// o atributo "focado" que causará nossosfolha de estilo interna// para exibir ou ocultar um anel de foco falso em todocomponente.Observação// que os eventos "Blur" e "Focus" borbulham e aparecempara originar// do <search-box>.this.input.onfocus = () => {this.setAtAttribute ("focado", "");};this.input.onblur = () => {this.Removeattribute ("focado")};// Se o usuário clicar na lupa, gatilho

uma "pesquisa">// evento.Também o desencadeia se o campo de entrada disparar um"mudar">// evento.(O evento "Mudança" não borbulhaa sombra dom.)leftSlot.OnClick = this.input.onchange = (event) => {event.stopPropagation ()// Prevendo eventos de clique de borbulhar if (this.disabled) retornar;// não faz nada quando desabilitado this.dispatchEvent (novo customevent ("pesquisa", {Detalhe: this.input.value}));}// Se o usuário clicar no X, acionar um "claro" evento // se preventDefault () não for chamado no evento, limpe a entrada.DireitosLot.OnClick = (Evento) => {event.stopPropagation ()// Não deixe o clique borbulhar if (this.disabled) retornar;// Não faça nada se desabilitado Seja e = novo customevent ("claro", {cancelável: true});this.dispatchEvent (e);if (! E.DefaultPrevented) {// se o evento não foi cancelado "this.input.value = "";// então limpe a entrada da campo}};// Quando alguns de nossos atributos são definidos ou alterados, precisamos para definir o// valor correspondente no elemento interno <input>.Este ciclo de vida// Método, juntamente com a estática observada attributes propriedade abaixo,// cuida disso.AttributeChangedCallback (Nome, OldValue, NewValue) {

```
if (nome === "desativado") {this.input.disabled = newValue! == null;} else if (nome === "espaço reservado") {this.input.placeholder = newValue;} else if (name === "size") {this.input.size = newValue;} else if (nome === "value") {this.input.value = newValue;}// Finalmente, definimos Getters e Setters de propriedades parapropriedades isso// corresponde aos atributos HTML que apoiamos.Ogetters simplesmente retornam// o valor (ou a presença) do atributo.E oSetters acabou de definir// o valor (ou a presença) do atributo.Quando aMétodo Setter// Altera um atributo, o navegador irá automaticamenteinvocar o// attributeChangedCallback acima.Obtenha espaço reservado () {retornarthis.getAttribute ("espaço reservado");}get size () {return this.getAttribute ("size");}get value () {return this.getAttribute ("value");}get desativado () {return this.hasattribute ("desativado");}enlouquecer () {return this.hasattribute ("hidden");}Definir espaço reservado (value) {this.setattribute ("espaço reservado",valor);}Set Tamanho (valor) {this.SetAttribute ("tamanho", valor);}Definir valor (text) {this.setAttribute ("value", texto);}set desativado (value) {if (value) this.setAttribute ("desativado", "");caso contrário, este.Removeattribute ("desativado");}Definir Hidden (Value) {if (value) this.setAttribute ("Hidden", "");caso contrário, este.Removeattribute ("Hidden");}
```

```
}// este campo estático é necessário para oMétodo AttributeChangedCallback// apenas atributos nomeados nesta matriz desencadearão chamadas paraEsse método.SearchBox.ObservedAttributes = ["desativado", "espaço reservado", "tamanho", "valor"];// Crie um elemento <Sodemplate> para segurar a folha de estilo e oárvore de// elementos que usaremos para cada instância da caixa de pesquisaelemento.SearchBox.Template = document.createElement ("modelo");// Inicializamos o modelo analisando essa sequência de HTML.Nota, no entanto,// Quando quando instanciamos uma caixa de pesquisa, somos capazes de apenasclonar os nós// no modelo e tem que analisar o HTML novamente.Searchbox.Template.innerhtml = `<estilo>/** O: seletor de host refere-se ao elemento <search-box> noluz* Dom.Esses estilos são padrões e podem ser substituídos pelousuário do* <search-box> com estilos no DOM da luz.*/:hospedar {Exibição: bloco embutido;/ * O padrão é exibição embutida */Fronteira: Black Solid 1px; /* Uma borda arredondada ao redor do<input> e <slots> */Radio de fronteira: 5px;preenchimento: 4px 6px; /* E algum espaço dentro da fronteira*/}: host ([Hidden]) { /* Observe os parênteses: quando hostescondeu ... */Exibir: Nenhum; /* ... Conjunto de atributos não exibi -lo*/}: host ([desativado]) { /* Quando o host tem o desativadoatributo ... */Opacidade: 0,5; /* ... Gray It Out */`
```

Erro ao traduzir esta página.

formatos de imagem, como GIF, JPEG e PNG, que especificam uma matriz de valores de pixel. Em vez disso, uma ?imagem? SVG é uma resolução precisaDescrição independente (daí "escalável") das etapas necessárias paraDesenhe o gráfico desejado. As imagens SVG são descritas por arquivos de texto usandoA linguagem de marcação XML, que é bastante semelhante à HTML. Existem três maneiras de usar o SVG nos navegadores da web:1. Você pode usar arquivos de imagem .svg com tags html regulares , Assim como você usaria uma imagem .png ou .jpeg.2. Porque o formato SVG baseado em XML é muito semelhante ao HTML, Você pode realmente incorporar tags SVG diretamente em seu htmldocumentos. Se você fizer isso, o analisador HTML do navegador permitevocê omita namespaces xml e tratar tags SVG como se elaseram tags html.3. Você pode usar a API DOM para criar dinamicamente SVGelementos para gerar imagens sob demanda. As subseções a seguir demonstram o segundo e o terceiro usos deSvg. Observe, no entanto, que o SVG tem um grande e moderadamente complexogramática. Além das primitivas simples de desenho de formas, incluiSuporte a curvas arbitrárias, texto e animação. Os gráficos SVG podematé incorporar scripts JavaScript e folhas de estilo CSS para adicionarInformações de comportamento e apresentação. Uma descrição completa do SVG éMuito além do escopo deste livro. O objetivo desta seção é apenas paraMostre como você pode usar o SVG em seus documentos e script HTMLcom JavaScript. 15.7.1 SVG em HTML

As imagens SVG podem, é claro, ser exibidas usando tags HTML <MG>.Mas você também pode incorporar SVG diretamente no HTML.E se você fizer isso,Você pode até usar folhas de estilo CSS para especificar coisas como fontes, cores,e larguras de linha.Aqui, por exemplo, é um arquivo html que usa SVG paraExiba uma face do relógio analógico:<html><head><title> relógio analógico </title><estilo>/* Esses estilos CSS se aplicam aos elementos SVG definidosabaixo */#clock { /* estilos para tudoNo relógio:*/AVC: preto;/ * linhas pretas */AVC-LINECAP: redondo;/ * Com extremidades arredondadas */preenchimento: #ffe; /* em um esbranquiçadofundo */}#clock .face {width: 3;} / * Contorno do relógio * /#clock .Ticks {Width: 2;} /* Linhas que marcam cada hora *//#clock .hands {stroke-width: 3;} /* Como desenhar o relógiomãos */#clock .numbers { /* como desenhar onúmeros */Font-Family: Sans-Serif;tamanho de fonte: 10;peso-fonte:audacioso;A âncora de texto: meio;AVC: nenhum;preenchimento: preto;}</style></head><Body><svg id = "relógio" viewBox = "0 0 100 100" width = "250"altura = "250"><!- ??Os atributos de largura e altura são o tamanho da telado gráfico -><!- ??O atributo ViewBox fornece a coordenada internasistema -><círculo class = "face" cx = "50" cy = "50" r = "45"/> <!- oFace do relógio ->

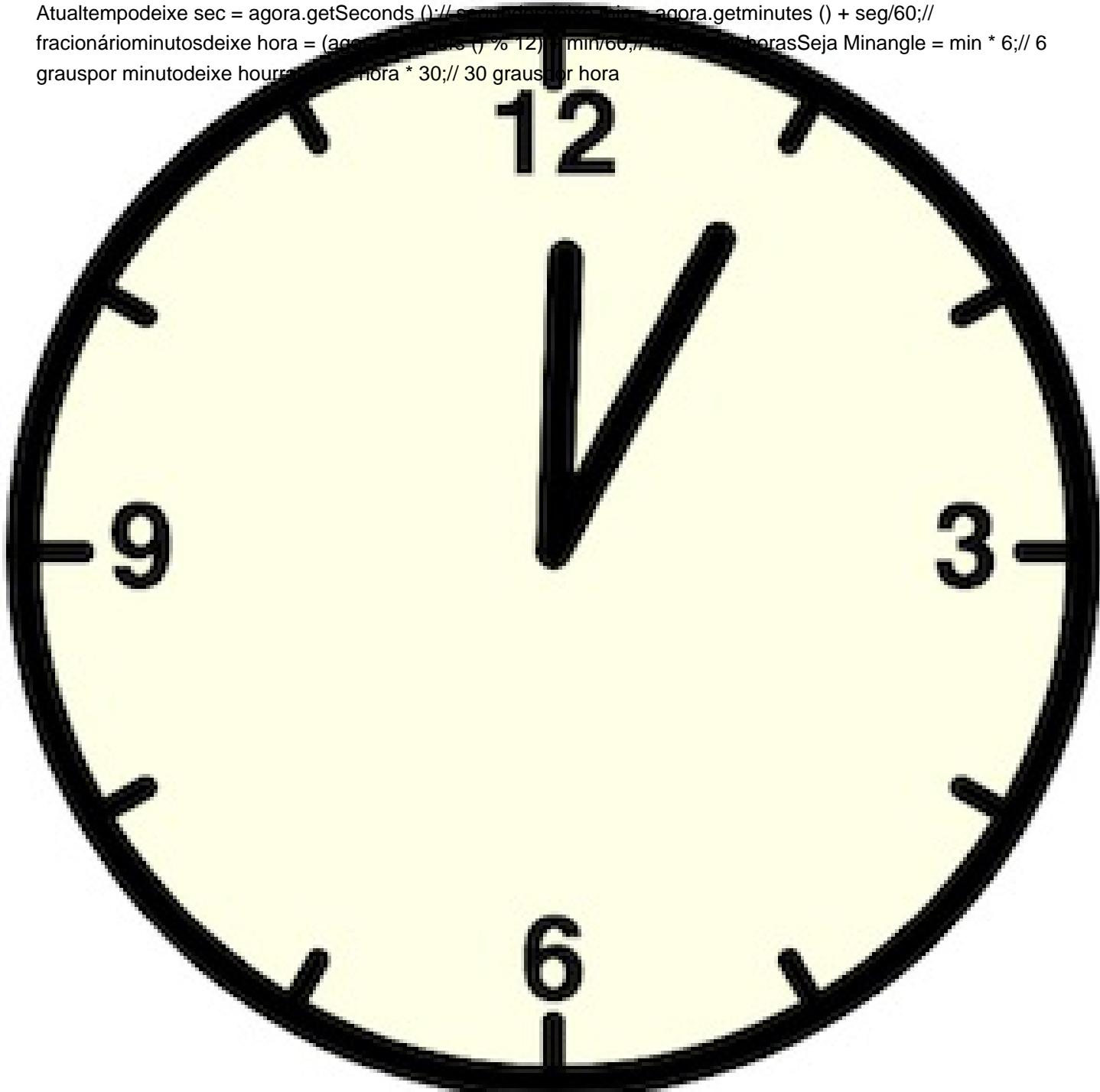
```
<g class = "ticks"> <!- Marcas de ticks para cada um dos 12horas -><linha x1 = '50 'y1 =' 5.000 'x2 = '50 .00' y2 = '10 .00 '/><linha x1 = '72 .50 'y1 = '11 .03' x2 = '70 .00 'y2 = '15 .36'/><linha x1 = '88 .97 'y1 = '27 .50' x2 = '84 .64 'y2 = '30 .00'/><linha x1 = '95 .00 'y1 = '50 .00' x2 = '90 .00 'y2 = '50 .00'/><linha x1 = '88 .97 'y1 = '72 .50' x2 = '84 .64 'y2 = '70 .00'/><linha x1 = '72 .50 'y1 = '88 .97' x2 = '70 .00 'y2 = '84 .64'/><linha x1 = '50 .00 'y1 = '95 .00' x2 = '50 .00 'y2 = '90 .00'/><linha x1 = '27 .50 'y1 = '88 .97' x2 = '30 .00 'y2 = '84 .64'/><linha x1 = '11 .03 'y1 = '72 .50' x2 = '15 .36 'y2 = '70 .00'/><linha x1 = '5.000' y1 = '50 .00 'x2 = '10 .00' y2 = '50 .00 '/><linha x1 = '11 .03 'y1 = '27 .50' x2 = '15 .36 'y2 = '30 .00'/><linha x1 = '27 .50 'y1 = '11 .03' x2 = '30 .00 'y2 = '15 .36'/></g><g class = "números"> <!-Número das direções cardinal--><texto x = "50" y = "18"> 12 </sext> <texto x = "85"y = "53"> 3 </sext><texto x = "50" y = "88"> 6 </sext> <texto x = "15"y = "53"> 9 </sext></g><g class = "Hands"> <!- Desenhe as mãos apontando para cima.-><line class = "hourhand" x1 = "50" y1 = "50" x2 = "50"y2 = "25"/><line class = "minutehand" x1 = "50" y1 = "50" x2 = "50"y2 = "20"/></g></svg><script src = "clock.js"> </sCript></body></html>Você notará que os descendentes da tag <Svg> não são normaisTags html.<circ>, <line> e <sext> tags têm óbvios propósitos, porém, e deve ficar claro como esse gráfico SVG funciona.Existem muitas outras tags SVG, no entanto, e você precisará consultarUma referência SVG para saber mais.Você também pode notar que oA folha de estilo é estranha.Estilos como preenchimento, largura de derrame e textoA âncora não são propriedades normais do estilo CSS.Nesse caso, CSS é
```

sendo essencialmente usado para definir atributos de tags SVG que aparecem no documento. Observe também que a propriedade de abreviação da fonte CSS não trabalha para tags SVG, e você deve definir explicitamente a família de fontes, tamanho de fonte e peso de fonte como propriedades de estilo separadas.

15.7.2 Scripts SVG

Um motivo para incorporar SVG diretamente nos seus arquivos HTML (em vez de apenas usando tags estático) é que, se você fizer isso, poderá usar o DOM API para manipular a imagem SVG. Suponha que você use SVG para exibir ícones em seu aplicativo da web. Você poderia incorporar SVG dentro de um <Sodemplate> tag (§15.6.2) e, em seguida, clone o conteúdo do modelo sempre que você precisar inserir uma cópia desse ícone em sua interface do usuário. E se você quer que o ícone responda à atividade do usuário - mudando de cor quando o usuário põe o mouse sobre ele, por exemplo - você pode alcançá-lo com CSS. Também é possível manipular dinamicamente os gráficos SVG que são diretamente incorporados em HTML. O exemplo do relógio de rosto no anteriorA seção exibe um relógio estático com mãos de hora e minuto voltadas para exibindo o meio-dia ou meia-noite. Mas você pode perceber que o arquivo HTML inclui uma tag <script>. Esse script é executado uma função periodicamente para verificar o tempo e transformar a hora em mãos minuciosas girando-lhes o número apropriado de graus para que o relógio realmente exiba o horário atual, como mostrado na Figura 15-5.

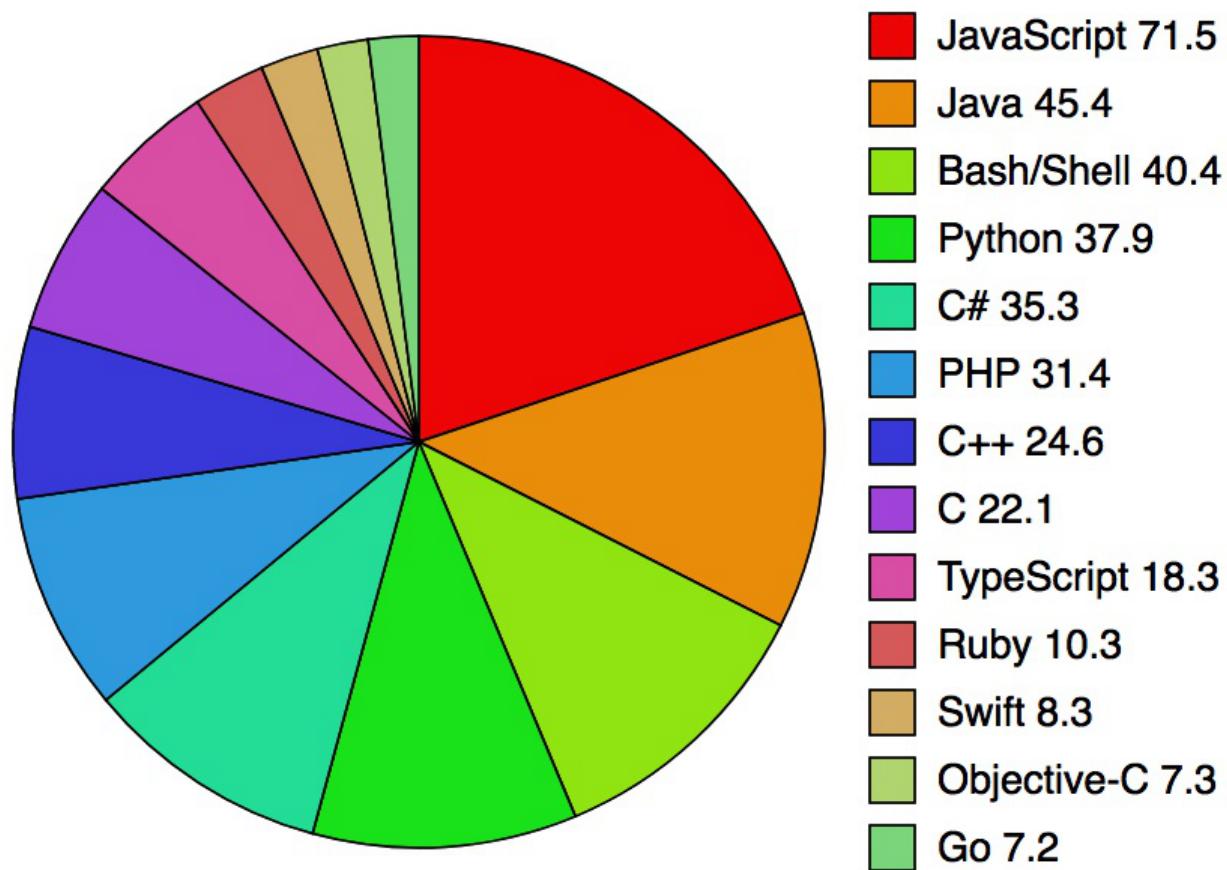
Figura 15-5.Um relógio analógico SVG com scriptO código para manipular o relógio é direto.Determina o ângulo adequado das mãos de hora e minuto com base na hora atual,Em seguida, usa querySelector () para procurar os elementos SVG que exibir essas mãos, então define um atributo de transformação para elesGire -os ao redor do centro da face do relógio.A função usasetTimeout () para garantir que ele funcione uma vez por minuto:(função updateClock () {// Atualize o gráfico do relógio SVG para Mostre a hora atualdeixe agora = new Date ();// Atualtempodeixe sec = agora.getSeconds ():// e agora.getMinutes () + seg/60;// fracionáriominutosdeixe hora = (agora.getHours () % 12) * min/60, // horaSeja Minangle = min * 6;// 6 grauspor minutodeixe hourangle = hora * 30;// 30 grauspor hora



Erro ao traduzir esta página.

Figura 15-6.Um gráfico de pizza SVG construído com JavaScript (dados do Stack Overflow's 2018 Pesquisa de desenvolvedor das tecnologias mais populares) Além do uso de createElementns (), o gráfico de pizza O código no Exemplo 15-4 é relativamente direto.Há um poucoMatemática para converter os dados que estão sendo traçados em ângulos de picada.O volume

Programming languages by percentage of professional developers who report their use



do exemplo, no entanto, é o código DOM que cria elementos SVG e define atributos nesses elementos. A parte mais opaca deste exemplo é o código que desenha o realfatias de torta. O elemento usado para exibir cada fatia é <TACH>. Esse elemento SVG descreve formas arbitrárias compostas por linhas e curvas. A descrição da forma é especificada pelo atributo D do <Path> elemento. O valor deste atributo usa uma gramática compacta de letraCódigos e números que especificam coordenadas, ângulos e outros valores. A letra m, por exemplo, significa "mudar para" e é seguida por x e y coordenadas. A letra L significa "linha para" e desenha uma linha do ponto atual para as coordenadas que o seguem. Este exemplo também usa a letra A para desenhar um arco. Esta carta é seguida por sete números descrevendo o arco e você pode procurar a sintaxe online se quiser. Saiba mais. Exemplo 15-4. Desenhando um gráfico de pizza com javascript e svg/*** Crie um elemento <SVG> e desenhe um gráfico de pizza para ele.** Esta função espera um argumento de objeto com as seguintes propriedades:** Largura, altura: o tamanho do gráfico SVG, em pixels* cx, cy, r: o centro e raio da torta* lx, ly: o canto superior esquerdo da lenda do gráfico* Dados: um objeto cujos nomes de propriedades são rótulos de dados e cujos* Os valores das propriedades são os valores associados a cada rótulo** A função retorna um elemento <Svg>. O chamador deve inserir -o* O documento para torná-lo visível.*/função piechart (opções) {

```
Seja {largura, altura, cx, cy, r, lx, ly, dados} = opções;// Este é o espaço de nome XML para elementos SVGSeja
svg = "http://www.w3.org/2000/svg";// Crie o elemento <Svg> e especifique o tamanho do pixel ecoordenadas do
usuárioLet Chart = Document.CreateElements (SVG, "SVG");Chart.SetAttribute ("Largura",
largura);Chart.SetAttribute ("altura", altura);Chart.SetAttribute ("ViewBox", `0 0 ${width} ${height}`);// Defina os
estilos de texto que usaremos para o gráfico.Se nósDeixe isso// Valores não definidos aqui, eles podem ser
definidos com CSS.Chart.SetAttribute ("Font-Family", "Sans-Serif");Chart.SetAttribute ("Font-Size", "18");// obtém
rótulos e valores como matrizes e adicione os valores paranós sabemos como// grande a torta é.deixe os rótulos =
object.keys (dados);deixe valores = object.Values ??(dados);Seja total = valores.Reduce ((x, y) => x+y);// Descubra
os ângulos para todas as fatias.Fatia que eu começoem ângulos [i]// e termina em ângulos [i+1].Os ângulos são
medidos emradianos.deixe ângulos = [0];valores.foreach ((x, i) => ângulos.push (ângulos [i] + x/total *2 *
math.pi));// agora percorre as fatias da tortavalores.foreach ((valor, i) => {// Calcule os dois pontos em que nossa
fatia cruzao círculo// essas fórmulas são escolhidas para que um ângulo de 0 sejaÀs 12 horas// e ângulos positivos
aumentam no sentido horário.Seja x1 = cx + r * math.sin (ângulos [i]);Seja y1 = cy - r * math.cos (ângulos [i]);Seja
x2 = cx + r * math.sin (ângulos [i + 1]);
```

Seja $y2 = cy - r * \text{math.cos}(\text{ângulos}[i+1])$; // esta é uma bandeira para ângulos maiores que um meio círculo// é necessário pelo componente de desenho de arco SVGDeixe grande = ($\text{ângulos}[i+1] - \text{ângulos}[i]$)> math.pi ?1: 0;// Esta string descreve como desenhar uma fatia da tortagráfico:Deixe o caminho = `m \$ {cx}, \$ {cy} + // move -se para o círculo centro.`L \$ {x1}, \$ {y1} + // Desenhe linha para(x1, y1).`A \$ {r}, \$ {r} 0 \$ {big} 1` + // desenhar um arco deraio r ...`\$ {x2}, \$ {y2}` + // ... terminando em(x2, y2)."Z"; // Fechar o caminho de volta para(CX, CY).// Calcule a cor CSS para esta fatia.Esta fórmula funciona apenas para// cerca de 15 cores.Portanto, não inclua mais de 15 fatias em um gráfico.deixe color = `hsl (\$ {(i*40)%360}, \$ {90-3*i}%, \$ {50+2*i}%)`;// Descrevemos uma fatia com um elemento <TATH>.Observação createElementns ().Deixe Slice = document.createElementns (SVG, "Path");// Agora defina atributos no elemento <ty Path>slice.setAttribute ("d", caminho); // Defina o caminho para esta fatia slice.setAttribute ("preenchimento", cor); // Defina a fatia slice.setAttribute ("Stroke", "Black"); // ContornoCorte em preto slice.setAttribute ("largura de derrame", "1"); // 1 pixel CSSpessoChart.Append (Slice); // Adicione a fatia para o gráfico// agora desenha um pequeno quadrado correspondente para a chaveDeixe icon = document.createElementns (svg, "rect"); icon.setAttribute ("x", lx); // posição quadrado

Erro ao traduzir esta página.

Erro ao traduzir esta página.

operações gráficas. WebGL não está documentado neste livro, no entanto: os desenvolvedores da web são mais prováveis para usar bibliotecas de utilitários construídos sobre o WebGL do que usar a API WebGL diretamente. A maior parte da API de desenho de tela é definida não na <Canvas> próprio elemento, mas em um objeto de "contexto de desenho" obtido como método getContext () da tela. Ligue para getContext () com o argumento "2d" para obter um objeto de renderingContext2d que você pode usar para desenhar gráficos bidimensionais na tela. Como um exemplo simples da API de tela, o seguinte HTML usa os elementos e alguma JavaScript para exibir duas formas simples:

```
<p> Este é um quadrado vermelho: <canvas id = "quadrado" largura = 10 Altura = 10> </ canvas >.<p> Este é um círculo azul: <canvas id = "círculo" largura = 10 Altura = 10> </ canvas >.<script>Let Canvas = Document.querySelector ("#Square");// Obtenha primeiro elemento de tela deixe context = canvas.getContext ("2D");// Obtenha 2d contexto de desenho context.fillStyle = "#f00";// defina preenchimento cor para vermelho context.fillRect (0,0,10,10);// preencha a quadrado canvas = document.querySelector ("#círculo");// Segundo elemento de tela context = canvas.getContext ("2d");// pega seu contexto context.beginPath ()// comece anovo "caminho" context.arc (5, 5, 5, 0, 2 * math.pi, verdadeiro);// Adicione acírculo para o caminho context.fillStyle = "#00f";// Defina azul Preencha a cor
```

Erro ao traduzir esta página.

Erro ao traduzir esta página.

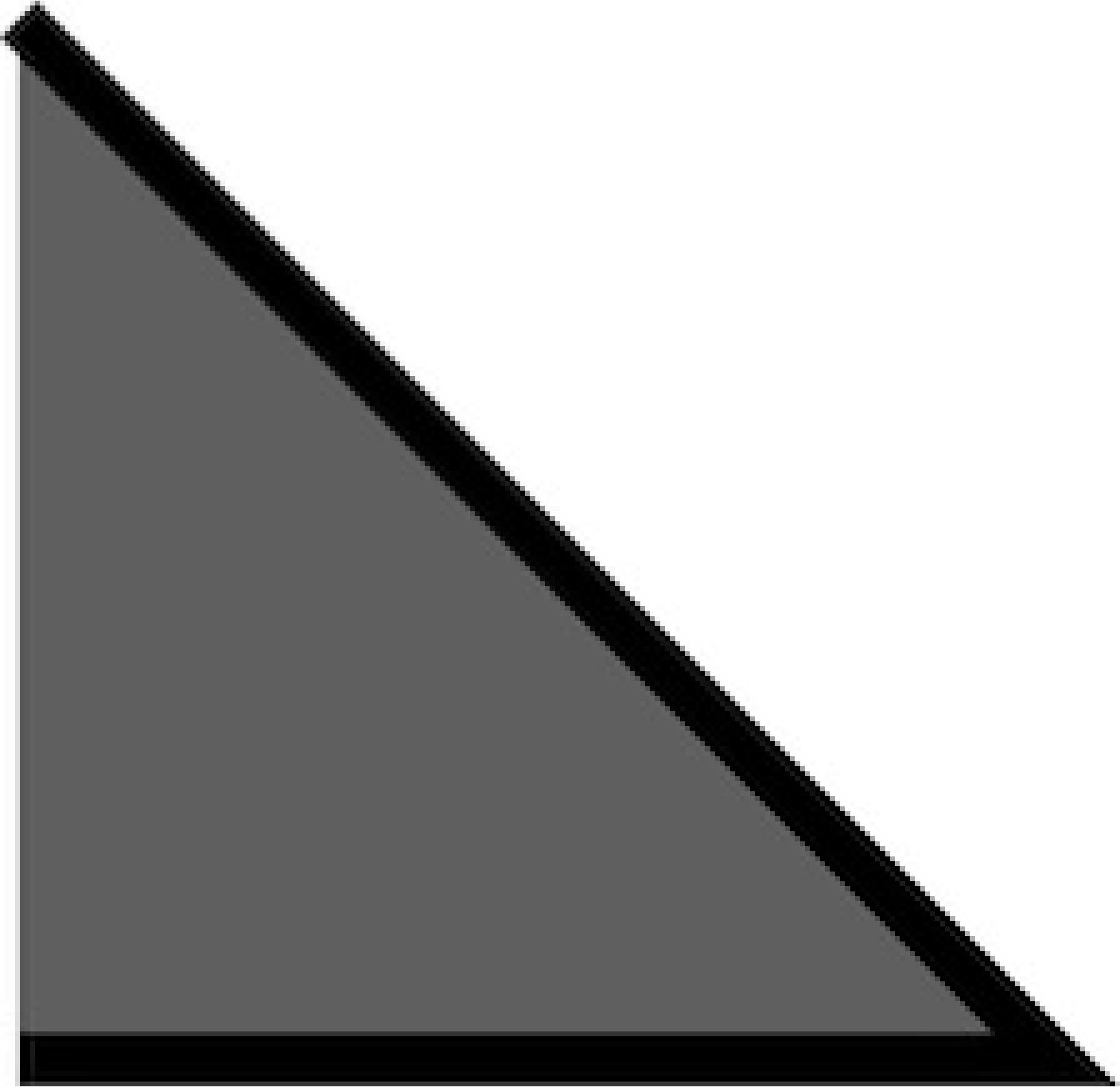
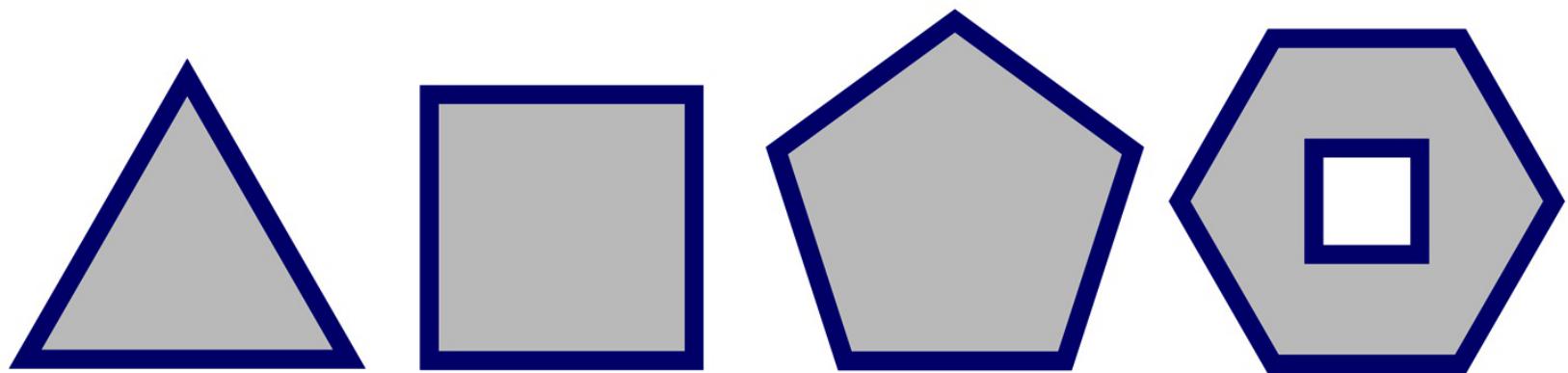


Figura 15-7.Um caminho simples, cheio e acariciadoObserve que o subpatina definido na Figura 15-7 está "aberto".Consiste em apenas dois segmentos de linha, e o ponto final não está conectado de volta ao ponto de partida.Isso significa que ele não inclui uma região.Oo método preenchimento () preenche subpaths abertos agindo como se fosse uma linha retaconnectou o último ponto no subspato ao primeiro ponto da subspata.É por isso que esse código preenche um triângulo, mas acaricia apenas dois lados do triângulo.Se você quisesse acariciar todos os três lados do triângulo acabados de mostrar, você chamaria o método closePath () para conectar o ponto final do Subpata ao ponto de partida.(Você também pode ligar para Lineto (100.100),Mas então você acaba com três segmentos de linha que compartilham um início e fim ponto, mas não estão realmente fechados.Ao desenhar com linhas largas, o visualOs resultados são melhores se você usar o closepath ().)Existem outros dois pontos importantes a serem notados sobre o Stroke () e preencher().Primeiro, ambos os métodos operam em todos os subspates na corrente caminho.Suponha que tivéssemos adicionado outro subpatil no código anterior:C.MoveTo (300.100);// Comece um novo sub -caminho em (300.100);c.lineto (300.200);// Desenhe uma linha vertical para(300.200);Se então chamássemos o Stroke (), desenharíamos duas bordas conectadas de um triângulo e uma linha vertical desconectada.O segundo ponto a ser observado sobre Stroke () e Fill () é que nem um altera o caminho atual: você pode chamar de preenchimento () e o caminho ainda vai

Esteja lá quando você liga para o Stroke (). Quando você termina com um caminho E quero começar outro, você deve se lembrar de chamar BEGNPATH (). Caso contrário, você acabará adicionando novos subspates ao caminho existente, E você pode acabar desenhando esses subspates antigos repetidamente. Exemplo 15-5 define uma função para desenhar polígonos regulares e demonstra o uso de moveto (), lineto () e closepath () para definir subspates e de preench () e stroke () para desenhar Esses caminhos. Produz o desenho mostrado na Figura 15-8. Figura 15-8. Polígonos regulares Exemplo 15-5. Polígonos regulares com moveto (), lineto () e ClosePath () // define um polígono regular com n lados, centralizado em (x, y) com raio r. // Os vértices estão igualmente espaçados ao longo da circunferência de um círculo. // Coloque o primeiro vértice direto ou no ângulo especificado. // gira no sentido horário, a menos que o último argumento seja verdadeiro. função polígono (c, n, x, y, r, ângulo = 0,



Erro ao traduzir esta página.

Erro ao traduzir esta página.

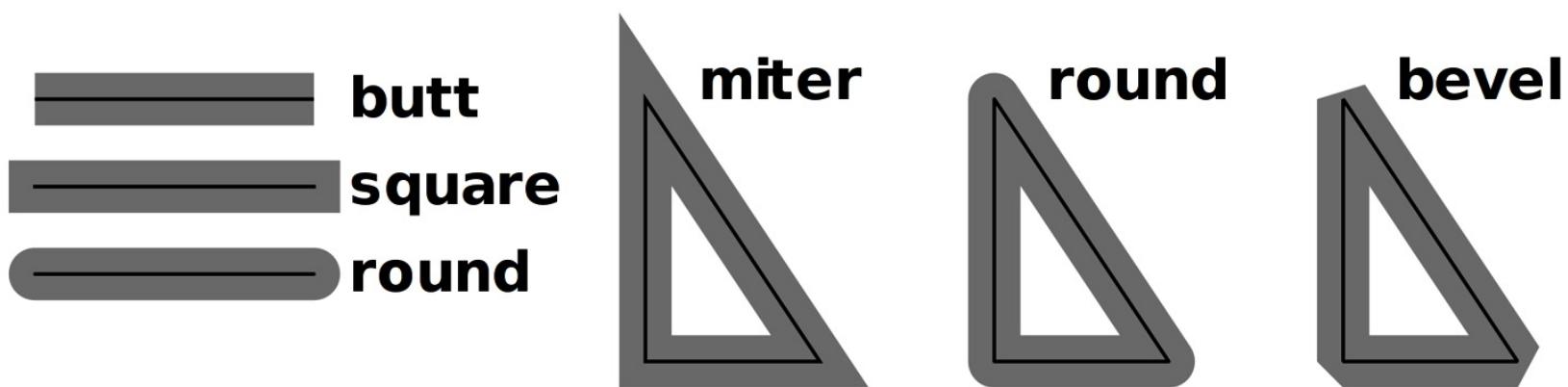
Número de pixels em que a tela pode atrair. Quatro bytes de memórias são alocados para cada pixel, portanto, se a largura e a altura estiverem definidas como 100, a tela aloca 40.000 bytes para representar 10.000 pixels. Os atributos de largura e altura também especificam o tamanho padrão (emPixels css) no qual a tela será exibida na tela. Se window.devicePixelRatio é 2, então 100×100 pixels CSS na verdade, 40.000 pixels de hardware. Quando o conteúdo da tela é desenhado na tela, os 10.000 pixels na memória precisarão ser ampliados para cobrir 40.000 pixels físicos na tela, e isso significa que seus gráficos não serão tão nítidos quanto poderiam ser. Para uma qualidade de imagem ideal, você não deve usar a largura e os atributos de altura para definir o tamanho da tela da tela. Em vez disso, defina o tamanho desejado no tamanho da tela CSS da tela com CSSAtributos do estilo de largura e altura. Então, antes de começar a desenhar em seu código JavaScript, defina as propriedades de largura e altura da tela se opõe ao número de CSS Pixels TimesWindow.DevicePixelRatio. Continuando com o anterior exemplo, essa técnica resultaria na exibição da tela em 100×100 pixels CSS, mas alocando memória por 200×200 pixels. (Mesmo com esta técnica, o usuário pode aumentar o zoom na tela e pode ver gráficos difusos ou pixelados, se o fizerem. Isso contrasta com o SVGGráficos, que permanecem nítidos, independentemente do tamanho ou zoom na tela.)

15.8.3 Atributos gráficos

Exemplo 15-5 Defina o FillStyle, Strokestyle e

Linha de linha no objeto de contexto da tela. Essas propriedades são atributos gráficos que especificam a cor a ser usada por preencher () e porStroke () e a largura das linhas a serem desenhadas por Stroke (). Observe que esses parâmetros não são passados ?? para o preenchimento () e Métodos Stroke (), mas fazem parte do estado gráfico geral da tela. Se você definir um método que desenha uma forma e não defina essas propriedades você mesmo, o chamador do seu método pode definir a cor da forma, definindo o StrokeStyle e o FillStyle propriedades antes de chamar seu método. Esta separação do estado gráfico dos comandos de desenho é fundamental para a API de tela e é semelhante à separação da apresentação do conteúdo alcançado pela aplicação Folhas de estilo CSS para documentos HTML. Existem várias propriedades (e também alguns métodos) no Objeto de contexto que afeta o estado gráfico da tela. Eles são detalhados abaixo. Estilos de linha A propriedade de largura de linha especifica o quanto amplo (em pixels CSS) as linhas desenhadas por Stroke () serão. O valor padrão é 1. É importante entender que a largura da linha é determinada pelo A propriedade de largura de linha no momento do momento () é chamada, não no momento que lineto () e outros métodos de construção de caminho são chamados. Para totalmente Entenda a propriedade de largura de linha, é importante visualizar caminhos como linhas unidimensionais infinitamente finas. As linhas e curvas desenhadas por O método Stroke () está centrado no caminho, com metade da Linha de linha de ambos os lados. Se você está acariciando um caminho fechado e apenas

quero que a linha apareça fora do caminho, acaricie o caminho primeiro e depois preencha com uma cor opaca para esconder a parte do golpe que aparece dentro do caminho. Ou se você deseja que a linha só apareça dentro de um fechadoCaminho, ligue para os métodos salvadores () e clip () primeiro e depois ligueStroke () e Restore ().(O salvamento (), restaure () eOs métodos clip () são descritos posteriormente.)Ao desenhar linhas que têm mais de cerca de dois pixels de largura, oAs propriedades LineCap e Linejoin podem ter um impacto significativo em a aparência visual das extremidades de um caminho e os vértices nos quaisDois segmentos de caminho se encontram.A Figura 15-9 ilustra os valores e resultantesAparência gráfica de LineCap e Linejoin.Figura 15-9.Os atributos LineCap e LinejoinO valor padrão para LineCap é "Butt".O valor padrão para



Linejoin é "Mitre". Observe, no entanto, que se duas linhas se encontrarem em um muitoângulo estreito, então a mitra resultante pode se tornar bastante longa e distrair visualmente. Se a mitra em um determinado vértice seria maior do que Metade da largura da linha vezes a propriedade Miterlimit, que o vértice será desenhado com uma junção chanfrada em vez de uma junção de mitered. O padrão O valor do Miterlimit é 10. O método Stroke () pode desenhar linhas tracejadas e pontilhadas, bem como Linhas sólidas e o estado gráfico de uma tela inclui uma variedade de números. Isso serve como um "padrão de traço" especificando quantos pixels desenham. Então, quantos para omitir. Ao contrário de outras propriedades de desenho de linha, o traçopadrão é definido e consultado com os métodos setLinedash () e getLinedash () em vez de com uma propriedade. Para especificar uma corrida pontilhadapadrão, você pode usar setLinedash () como este: C.setLinedash ([18, 3, 3, 3]); // 18px Dash, 3px Espaço, 3px DOT, 3px Espaço Finalmente, a propriedade LinedashOffset especifica até que ponto no O desenho do padrão de traço deve começar. O padrão é 0. Caminhos acariciados com O padrão de traço mostrado aqui começa com um traço de 18 pixels, mas se LinedashOffset está definido como 21, então esse mesmo caminho começaria com Um ponto seguido de um espaço e uma corrida. Cores, padrões e gradientes As propriedades de FillStyle e StrokeStyle especificam como caminhos são preenchidos e acariciados. A palavra "estilo" geralmente significa cor, mas estes propriedades também podem ser usadas para especificar um gradiente de cores ou uma imagem para ser usado para encher e acariciar. (Observe que desenhar uma linha é basicamente o

o mesmo que preencher uma região estreita em ambos os lados da linha, e preencher e acariciar são fundamentalmente a mesma operação.) Se você deseja encher ou acariciar com uma cor sólida (ou uma cor translúcida), Basta definir essas propriedades como uma corda de cor CSS válida. Nada mais é obrigatório. Para encher (ou derrame) com um gradiente de cores, defina um estilo de abastecimento (ou `strokeStyle`) a um objeto de graduação de gaiola devolvido pelo `createLinearGradient()` ou `createRadialGradient()`. Métodos do contexto. Os argumentos para `createLinearGradient()` são as coordenadas de dois pontos que definem uma linha (ela não precisa ser horizontal ou vertical) ao longo da qual as cores variam. Os argumentos para o `createRadialGradient()` Especifique os centros e os raios de dois círculos. (Eles não precisam ser concêntrico, mas o primeiro círculo normalmente está inteiramente dentro do segundo.) Áreas dentro do círculo menor ou fora do maior serão preenchidas com cores sólidas; Áreas entre os dois serão preenchidas com um gradiente de cores. Depois de criar o objeto `CanvasGradient` que define as regiões dotela que será preenchida, você deve definir as cores do gradiente ligando o método `addColorStop()` do `CanvasGradient`. O primeiro argumento deste método é um número entre 0,0 e 1,0. O segundo argumento é uma especificação de cores CSS. Você deve chamar esse método pelo menos duas vezes para definir um gradiente de cores simples, mas você pode chamá-lo mais do que isso. A cor em 0,0 aparecerá no início do gradiente e a cor em 1,0 aparecerá no final. Se você especificar cores adicionais, elas aparecerão na posição fracionária especificada dentro do gradiente.

Entre os pontos que você especificar, as cores serão interpoladas suavemente. Aqui estão alguns exemplos:// um gradiente linear, na diagonal através da tela (assumindo sem transformações) Deixe bgfade =c.createlineargradiente (0,0, tela.width, canvas.Height);bgfade.addcolorstop (0,0, "#88f");// Comece com azul claro no canto superior esquerdo bgfade.addcolorstop (1.0, "#fff");// desaparece em branco em inferior certo// Um ?? gradiente entre dois círculos concêntricos. Transparente no meio// Desbotando a cinza translúcido e depois voltou a transparente. Seja DONUT = C.CreamRadialGradient (300.300.100, 300.300.300);donut.addcolorstop (0,0, "transparent");// Transparente donut.addcolorstop (0,7, "RGBA (100.100.100, .9)");// Cinza translúcido donut.addcolorstop (1.0, "rgba (0,0,0,0)");// Transparente novamente Um ponto importante a entender sobre os gradientes é que eles não são independentes da posição. Quando você cria um gradiente, você especifica limites para o gradiente. Se você tentar preencher uma área fora daqueles limites, você terá a cor sólida definida em uma extremidade ou outra do gradiente. Além de cores e gradientes de cores, você também pode preencher e derrotar usando imagens. Para fazer isso, defina um estilo de enchimento ou estrogo para um CanvAs pattern retornado pelo método CreatePattern () do objeto de contexto. O primeiro argumento para este método deve ser um <MG> ou elemento <Canvas> que contém a imagem que você deseja preencher ou golpe com. (Observe que a imagem de origem ou tela não precisa ser

Erro ao traduzir esta página.

sombrias. Se você definir essas propriedades adequadamente, qualquer linha, área, texto, ou imagem que você desenha terá uma sombra, o que fará com que pareça como se estivesse flutuando acima da superfície da tela. A propriedade `ShadowColor` especifica a cor da sombra. O padrão é totalmente transparente preto e as sombras nunca aparecerão, a menos que você defina esta propriedade para uma cor translúcida ou opaca. Esta propriedade pode apenas ser definida como uma corda de cor: padrões e gradientes não são permitidos nas sombras. Usar uma cor de sombra translúcida produz o mais realista efeitos de sombra porque permite que o plano de fundo seja exibido. As propriedades `ShadowOffsetX` e `ShadowOffsetY` especificam as compensações X e Y da sombra. O padrão para ambas as propriedades é 0, que coloca a sombra diretamente abaixo do seu desenho, onde não está visível. Se você definir as duas propriedades como um valor positivo, as sombras irão aparecer abaixo e à direita do que você desenha, como se houvesse uma luz fonte acima e à esquerda, brilhando na tela de fora da tela do computador. Compensações maiores produzem sombras maiores e fazem os objetos desenhados parecerem estar flutuando "mais alto" acima da tela. Esses valores não são afetados por transformações de coordenadas (§15.8.5): a direção da sombra e a "altura" permanecem consistentes mesmo quando as formas são giradas e escaladas. A propriedade `ShadowBlur` especifica o quanto embacado as bordas da sombra são. O valor padrão é 0, que produz nítido e não-ilícitos sombras. Valores maiores produzem mais borrão, até uma implementação-limite superior definida. Translucidez e composição

Se você deseja acariciar ou encher um caminho usando uma cor translúcida, você pode definir StrokeStyle ou FillStyle usando uma sintaxe de cor CSS como "RGBA (...) " que suporta a transparência da Alpha.O "a" em "rgba" significa "alfa" e é um valor entre 0 (totalmente transparente) e 1 (totalmente opaco).Mas a API de tela fornece outra maneira de trabalhar com cores translúcidas.Se você não deseja especificar explicitamente um alfacanal para cada cor, ou se você quiser adicionar translucidez ao opacolagens ou padrões, você pode definir a propriedade GlobalAlpha.TodoO pixel que você desenha terá seu valor alfa multiplicado pela GlobalAlpha.O padrão é 1, o que não adiciona transparência.Se você definirGlobalAlpha a 0, tudo o que você desenha será totalmente transparente,E nada aparecerá na tela.Mas se você definir esta propriedade como 0,5, então os pixels que de outra forma seriam opacos serão 50%opaco e pixels que teriam sido 50% opacos serão 25%em vez disso.Quando você acaricia as linhas, enche as regiões, desenhe texto ou copia imagens, você geralmente espera que os novos pixels sejam desenhados em cima dos pixels que já estão na tela.Se você está desenhando pixels opacos, elesBasta substituir os pixels que já estão lá.Se você está desenhando comPixels translúcidos, o novo pixel (" fonte ") é combinado com o antigo(?Destino?) Pixel para que o pixel antigo apareça através do novo pixelCom base em quanto transparente é esse pixel.Este processo de combinar novos pixels de origem (possivelmente translúcidos)Com os pixels de destino existentes (possivelmente translúcidos) são chamadoscomposição, e o processo de composição descrito anteriormente é o maneira padrão de que a API de tela combina pixels.Mas você pode definir o

propriedade globalComPoTePoperation para especificar outras maneiras de Combinando pixels. O valor padrão é "fonte-over", o que significa que os pixels de origem são desenhados "sobre" os pixels de destino e são combinados com eles se a fonte for translúcida. Mas se você definir GlobalComposePoperation to "Destination-Over", então a tela combinará pixels como se os novos pixels de origem fossem desenhados sobre os pixels de destino existentes. Se o destino for translúcido ou transparente, parte ou toda a cor da fonte de pixels é visível na cor resultante. Como outro exemplo, o modo de composição ?fonte-no topo ?combina os pixels de origem com a transparência do pixels de destino para que nada seja desenhado em partes da tela que já são totalmente transparentes. Existem vários valores legais para o GlobalComposePoperation, mas a maioria tem apenas especializações e não são cobertos aqui. Salvando e restaurando o estado gráfico Como a API de tela define atributos gráficos no objeto de contexto, você pode ficar tentado a chamar getContext () várias vezes para obter múltiplos objetos de contexto. Se você pudesse fazer isso, você poderia definir atributos diferentes em cada contexto: cada contexto seria como um pincel diferente e pintaria com uma cor diferente ou desenhar linhas de diferentes larguras. Infelizmente, você não pode usar a tela dessa maneira. Cada elemento <Canvas> tem apenas um único objeto de contexto e todos ligam para getContext () retorna o mesmo canvas AsRenderingContext2D objeto. Embora a API de tela permita apenas você definir um único conjunto de atributos gráficos de cada vez, ele permite salvar a corrente

Os gráficos afirmam que você pode alterá-lo e depois restaurá-lo facilmente mais tarde. O método `save()` empurra o estado gráfico atual para uma pilha de estados salvos. O método `Restore()` aparece na pilha e restaura o mais recentemente salvo. Todas as propriedades que foram descritas nesta seção fazem parte do estado salvo, assim como o `actualRegion` de transformação e recorte (ambos explicados posteriormente). É importante ressaltar que o caminho atualmente definido e o ponto atual não são parte do estado gráfico e não pode ser salvo e restaurado.

15.8.4 Operações de desenho de tela

Já vimos alguns métodos básicos de lona - `BeginPath()`, `MoveTo()`, `LineTo()`, `ClosePath()`, `Fill()` e `Stroke()`. Para definir, encher e desenhar linhas e polígonos. Mas a API também inclui outros métodos de desenho.

`RetângulosCanvasRenderingContext2D` define quatro métodos para desenhar retângulos. Todos os quatro métodos de retângulo esperam dois argumentos que especificam um canto do retângulo seguido pela largura do retângulo e altura. Normalmente, você especifica o canto superior esquerdo e depois passa a largura positiva e altura positiva, mas você também pode especificar outros cantos e passar dimensões negativas.

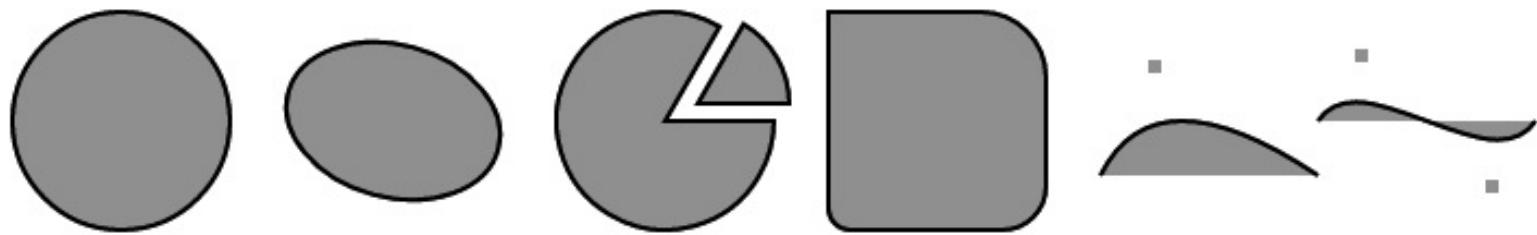
`FILLRECT()` preenche o retângulo especificado com a corrente `FillStyle`. `strokeRect()` desenha o contorno do retângulo usando os atributos atuais do `StrokeStyle` e outras linhas.

`ClearRect()` é como `FillRect()`, mas ignora o preenchimento atual.

estilo e preenche o retângulo com pixels pretos transparentes (o padrão cor de todas as telas em branco). O importante sobre esses três métodos é que eles não afetam o caminho atual ou o ponto atual dentro desse caminho. O método do retângulo final é chamado `Rect()` e afeta o Caminho atual: adiciona o retângulo especificado, em uma subspata própria, ao caminho. Como outros métodos de definição de caminho, ele não preenche ou golpea qualquer coisa em si. Curvas Um caminho é uma sequência de subspates, e um subspato é uma sequência de pontos conectados. Nos caminhos que definimos em §15.8.1, esses pontos foram conectados com segmentos de linha reta, mas que nem sempre precisam ser assim. O objeto `CanvasRenderingContext2D` define uma série de métodos que adicionam um novo ponto ao subspato e conectam a corrente a ponte para esse novo ponto com uma curva: `arco()`. Este método adiciona um círculo ou uma parte de um círculo (um arco), ao caminho. O arco a ser desenhado é especificado com seis parâmetros: o `x` e `y` coordenadas do centro de um círculo, o raio do círculo, os ângulos de início e final do arco e a direção (no sentido horário ou no sentido anti-horário) do arco entre esses dois ângulos. Se houver um ponto atual no caminho, então este método conecta o ponto atual à ponte para o início do arco com uma linha reta (que é útil ao desenhar cunhas ou fatias de torta), então conecta o início do arco até o final do arco com uma parte de um círculo, deixando o fim do arco como o novo ponto atual. Se não houver ponto atual Quando esse método é chamado, então adiciona apenas o arco circular ao

caminho.elipse()Este método é muito parecido com o arc (), exceto que adiciona uma elipse ou um parte de uma elipse para o caminho.Em vez de um raio, ele tem dois:um raio do eixo x e um raio do eixo y.Além disso, porque as elipses não são radialmente simétrico, esse método leva outro argumento que especifica o número de radianos pelos quais a elipse é girada no sentido horário sobre seu centro.arcto ()Este método desenha uma linha reta e um arco circular como o método arc () faz, mas especifica o arco a ser desenhado usando parâmetros diferentes.Os argumentos para Arcto () especificam pontos P1 e P2 e um raio.O arco que é adicionado ao caminho tem o raio especificado.Começa no ponto tangente com o (imaginário)linha do ponto atual para P1 e termina no ponto tangente com a linha (imaginária) entre P1 e P2.Isso é incomumO método de especificar arcos é realmente bastante útil para desenhar formas com cantos arredondados.Se você especificar um raio de 0, este método apenas desenha uma linha reta do ponto atual para P1.Com um raio diferente de zero, no entanto, ele desenha uma linha reta da corrente apontar na direção de P1, depois curva essa linha em um círculoAté que esteja indo na direção de P2.beziercurveto ()Este método adiciona um novo ponto P ao subspato e o conecta ao ponto atual com uma curva de bezier cúbica.A forma da curva é especificado por dois "pontos de controle", C1 e C2.No início da curva (no ponto atual), a curva cabeças na direção de C1.No final da curva (no ponto P), a curva chega na direção de C2.Entre esses pontos, a direção da curva varia suavemente.O ponto P se torna o novo ponto atual para o

`Subpath quadraticCurveTo ()` Este método é como `BezierCurveTo ()`, mas usa um quadráticoCurva bezier em vez de uma curva bezier cúbica e tem apenas um único ponto de controle. Você pode usar esses métodos para desenhar caminhos como os da Figura 15-10. Figura 15-10. Caminhos curvos em uma tela O exemplo 15-6 mostra o código usado para criar a Figura 15-10. Os métodos demonstrados neste código são alguns dos mais complicados da API de tela; consulte uma referência on-line para obter detalhes completos sobre os métodos e seus argumentos. Exemplo 15-6. Adicionando curvas a um caminho // Uma função de utilidade para converter ângulos de graus em radianos `function rads (x) {return math.pi*x/180;}` // Obtenha o objeto de contexto do elemento de tela do documento Seja `c = document.querySelector ("Canvas").getContext ("2D")`; // Defina alguns atributos gráficos e desenhe as curvas `c.fillStyle = "#aaa";` // preenchimentos cinza `c.lineWidth = 2;` // linhas pretas 2-pixels (por padrão)



// Desenhe um círculo.// não há ponto atual, então desenhe apenas o círculo semdireto// linha do ponto atual para o início do círculo.C.BeginPath ();c.arc (75,100,50, // centro em (75.100), raio 500, rads (360), falso);// vá no sentido horário de 0 a 360 graus.c.fill ()// preencha o círculo.c.stroke ()// ALTE seu esboço.// agora desenhe uma elipse da mesma maneiraC.BeginPath ()// iniciar um novo caminho não conectado ao círculoC.Ellipse (200, 100, 50, 35, Rads (15), // Center, Radii erotação0, rads (360), falso);// Iniciar ângulo, fimângulo, direção// Desenhe uma cunha.Ângulos são medidos no sentido horário a partir doeixo x positivo.// Observe que o arc () adiciona uma linha do ponto atual aoARC START.C.Moveto (325, 100);// Comece no centro do círculo.c.arc (325, 100, 50, // Circle Center e Radiusrads (-60), rads (0), // comece no ângulo -60 e vá para o ângulo0verdadeiro);// no sentido anti-horárioC.ClosePath ()// Adicione o raio de volta ao centro deo círculo// cunha semelhante, compensar um pouco e na direção opostaC.Moveto (340, 92);c.arc (340, 92, 42, rads (-60), rads (0), false);C.ClosePath ()// Use Arcto () para cantos arredondados.Aqui desenhamos um quadrado com// canto superior esquerdo em (400,50) e cantos de raios variados.C.Moveto (450, 50);// começa no meio do topoC.Arcto (500.50.500.150,30);// Adicione parte da borda superior e superiorCanto direito.C.Arcto (500.150.400.150,20);// Adicione a borda direita e inferior direitocanto.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Na versão de três argumentos de drawimage (), o segundo e o terceiroArgumentos especificam as coordenadas X e Y nas quais a parte superior esquerda canto da imagem deve ser desenhado.Nesta versão do método, oA imagem de origem inteira é copiada para a tela.As coordenadas X e Y sãointerpretado no sistema de coordenadas atual e a imagem é escaladae girado, se necessário, dependendo da transformação da tela atualmente com efeito.A versão de cinco argumentos de drawimage () adiciona largura eArgumentos de altura para os argumentos X e Y descritos anteriormente.EssesQuatro argumentos definem um retângulo de destino dentro da tela.Ocanto superior esquerdo da imagem de origem vai em (x, y), e o inferiorO canto direito vai em (x+largura, y+altura).Novamente, o todoA imagem de origem é copiada.Com esta versão do método, a fonteA imagem será dimensionada para caber no retângulo de destino.A versão de nove argumentos de drawimage () especifica uma fonteretângulo e um retângulo de destino e copia apenas os pixels dentroo retângulo de origem.Argumentos dois a cinco especificam a fonteretângulo.Eles são medidos em pixels CSS.Se a imagem de origem forOutra tela, o retângulo de origem usa o sistema de coordenadas padrão para aquela tela e ignora qualquer transformação que tenha sidoespecificado.Argumentos de seis a nove especificam o retângulo de destinono qual a imagem é desenhada e está no sistema de coordenadas atualda tela, não no sistema de coordenadas padrão.Além de desenhar imagens para uma tela, também podemos extrair oConteúdo de uma tela como uma imagem usando o método Todataurl ().

Erro ao traduzir esta página.

O método `setTransform()` permite definir uma matriz de transformação diretamente, mas coordenar transformações do sistema geralmente são mais fáceis de especificar como uma sequência de traduções, rotações e operações de escala. A Figura 15-11 ilustra essas operações e seu efeito no sistema de coordenadas de tela. O programa que produziu a figura desenhou o mesmo conjunto de eixos sete vezes seguidos. A única coisa que mudou cada vez foi a transformação atual. Observe que as transformações afetam o texto e as linhas desenhadas.

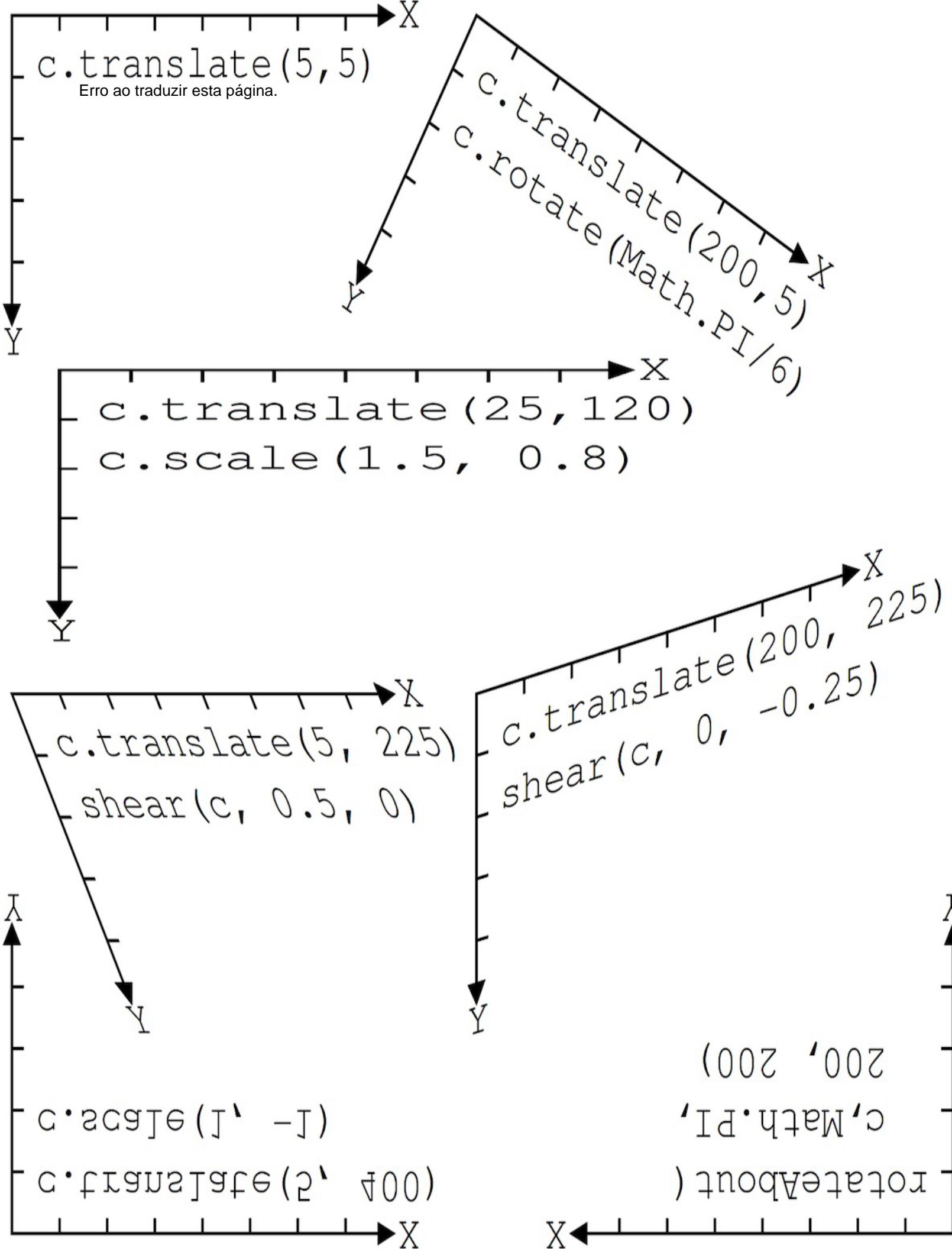


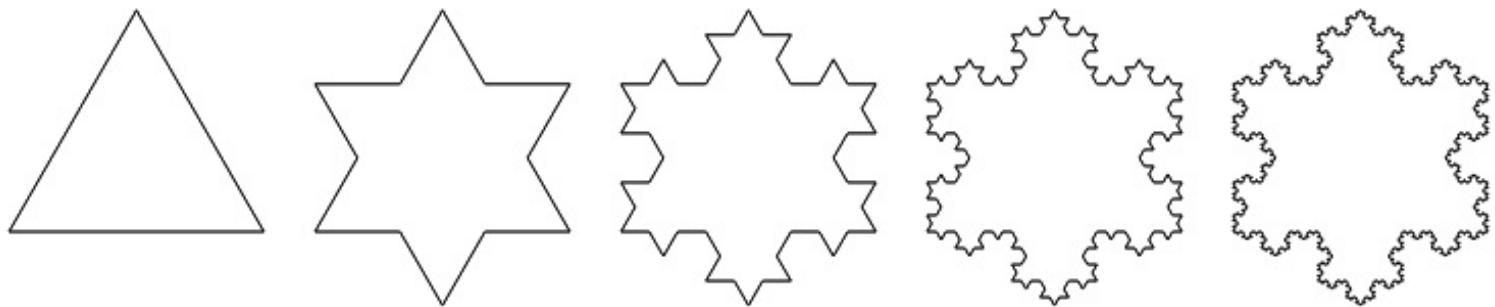
Figura 15-11. Coordenar transformações do sistema O método TRANSTE () simplesmente move a origem da coordenadas sistema esquerdo, direita, para cima ou para baixo. O método girate () gira os eixos no sentido horário pelo ângulo especificado. (A API de tela sempre especifica os ângulos em radianos. Para converter graus em radianos, dividir por 180 e multiplicar por math.pi.) o método escala () se estende ou contrata distâncias ao longo dos eixos x ou y. Passando um fator de escala negativo para o método da escala () vira esse eixo através da origem, como se fosse refletido em um espelho. Isso é o que era feito no canto inferior esquerdo da Figura 15-11: TRADLE () foi usado para Mover a origem para o canto inferior esquerdo da tela e, em seguida, escala () foi usado para virar o eixo y ao redor, para que as coordenadas Y aumentassem quando subíssemos a página. Um sistema de coordenadas invertidas como esse é familiar da classe de álgebra e pode ser útil para plotar pontos de dados nos gráficos. Observação: No entanto, isso dificulta a leitura do texto! Compreensão de transformações Matematicamente Acho mais fácil entender transformações geometricamente, pensando sobre tradução (), giro () e escala () como transformando os eixos do sistema de coordenadas, conforme ilustrado na Figura 15-11. É também possível entender transformações algebraicamente como equações que mapeiam as coordenadas de um ponto (x, y) no sistema de coordenadas transformadas de volta às coordenadas (x', y') do mesmo ponto no anterior sistema de coordenadas. O método chama C.Translate (DX, DY) pode ser descrito com

Essas equações: $x' = x + dx$; // Uma coordenada X de 0 no novo sistema é DX
no velho $y' = y + dy$; As operações de escala têm equações igualmente simples. Uma chamada C.Scale (SX, SY) pode ser descrito assim: $x' = sx * x$; $y' = sy * y$; As rotações são mais complicadas. A chamada c.rotate (a) é descrita por essas equações trigonométricas: $x' = x * \cos(a) - y * \sin(a)$; $y' = y * \cos(a) + x * \sin(a)$; Observe que a ordem das transformações é importante. Suponha que começamos como sistema de coordenadas padrão de uma tela, depois traduzi-lo e depois escalar. Para mapear o ponto (x, y) na coordenada atual para o ponto (x'', y'') no sistema de coordenadas padrão, devemos primeiro aplicar as equações de escala para mapear o ponto para um ponto intermediário (x', y') na coordenada traduzida, mas não escalada, e use as equações de tradução para mapear este ponto intermediário para (x'', y'') . O resultado é o seguinte: $x'' = sx * x + dx$; $y'' = sy * y + dy$; Se, por outro lado, chamamos de escala () antes de ligar TRADLATE (), as equações resultantes seriam diferentes:

$x'' = sx*(x + dx); y'' = sy*(y + dy);$ A coisa principal a lembrar ao pensar em algebraicamente sobre Sequências de transformações são que você deve trabalhar para trás doÚltima (mais recente) transformação para a primeira. Quando pensarGeometricamente sobre eixos transformados, no entanto, você trabalha adianteDa primeira transformação para o último. As transformações apoiadas pela tela são conhecidas como afinstransforma. As transformações afine podem modificar as distâncias entrePontos e os ângulos entre as linhas, mas as linhas paralelas sempre permanecemparalelo após uma transformação afim - não é possível, por exemplo, para especificar uma distorção da lente olho de peixe com uma transformação afim. UmTransformação afim arbitrária pode ser descrita pelos seis parâmetros aAtravés de F nessas equações: $x' = ax + cy + ey' = bx + dy + f$ Você pode aplicar uma transformação arbitrária à coordenada atuaisistema passando esses seis parâmetros para o método transform (). A Figura 15-11 ilustra dois tipos de transformações-tenhas erotações sobre um ponto especificado - que você pode implementar com oMétodo transform () como este:// Transformação de cisalhamento:// $x' = x + kx*y;$ // $y' = ky*x + y;$ função cisalhamento (c, kx, ky) {c.transform (1, ky, kx, 1, 0, 0);} // gira os radianos teta no sentido anti -horário

Erro ao traduzir esta página.

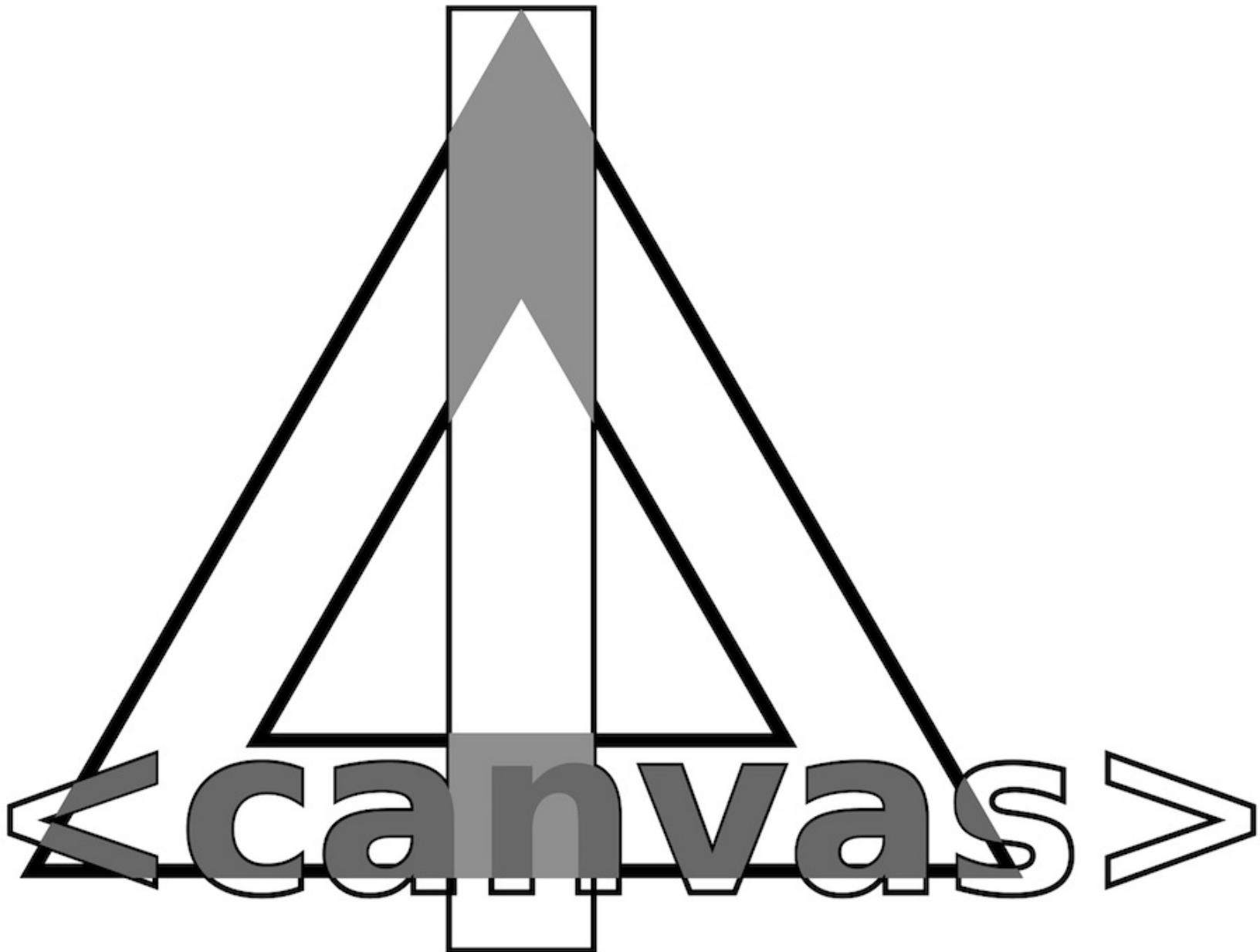
Figura 15-12.Koch SnowflakesO código que produz esses números é elegante, mas o uso de recursivoAs transformações do sistema de coordenadas tornam um pouco difficultender.Mesmo se você não seguir todas as nuances, observe que o código inclui apenas uma única invocação do método lineto ().TodoO segmento de linha única na Figura 15-12 é desenhada assim:c.lineto (Len, 0);O valor da variável len não muda durante a execução deo programa, assim a posição, orientação e duração de cada uma das linhassegmentos são determinados por traduções, rotações e escalaoperações.Exemplo 15-7.Um floco de neve Koch com transformaçõesDeixe deg = math.pi/180;// para converter graus em radianos// Desenhe um Fractal de Floco de Neve Koch Level-N no contexto da telac,// com canto inferior à esquerda em (x, y) e comprimento lateral len.função snowflake (c, n, x, y, len){C.Save ()// Salvar a transformação atualc.Translate (x, y)// traduz a origem para o ponto de partida.c.Moveto (0,0)// Comece um novo subspato no novoorigemperna (n)// Desenhe a primeira etapa do floco de nevec.rotate (-120*gra);// agora gira 120 graus



no sentido anti -horárioperna (n);// Desenhe a segunda perna.c.rotate (-120*gra);// gira novamenteperna (n);// Desenhe a perna finalC.ClosePath ()// Fechar o subspôC.Restore ()// e restaurar a transformação original// Desenhe uma única perna de um floco de neve de nível N Koch// Esta função deixa o ponto atual no final doperna tem// desenhado e traduz o sistema de coordenadas para que oO ponto atual é (0,0).// isso significa que você pode chamar facilmente girtate () depois de desenhar umperna.Função perna (n) {C.Save ()// salvar a correntetransformaçâoif (n === 0) {// caso não recursivo:c.lineto (Len, 0);// Apenas desenhe uma horizontallinha} // _else {// caso recursivo: desenhe 4 sub-pernas como: V/c.Scale (1/3,1/3);// Sub-pernas são 1/3 do tamanho deesta pernaperna (n-1);// recorrente para o primeiro sub-perna.c.rotate (60*graus);// gira 60 graus no sentido horárioperna (n-1);// Segunda sub-perna.c.rotate (-120*gra);// gira 120 graus de voltaperna (n-1);// Terceira sub-perna.c.rotate (60*graus);// Gire de volta para o nosso originalcabeçalhoperna (n-1);// Sub-perl}C.Restore ()// restaurar a transformaçâoc.Translate (Len, 0);// mas traduza para fazer o fim deperna (0,0)}{}}

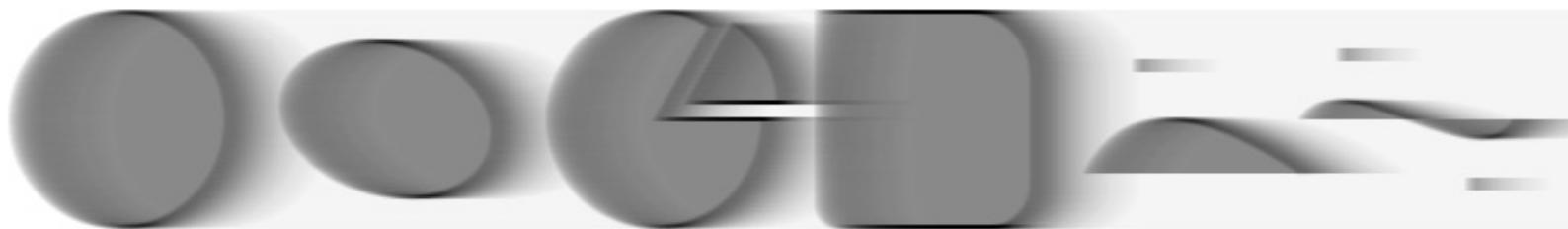
Seja c = document.querySelector ("Canvas"). GetContext ("2D");Snowflake (C, 0, 25, 125, 125); // um floco de neve de nível-0 é um triânguloSnowflake (c, 1, 175, 125, 125); // A Nível 1 Snowflake é um 6-estrela do ladoSnowflake (C, 2, 325, 125, 125); // etc.Snowflake (C, 3, 475, 125, 125);Snowflake (C, 4, 625, 125, 125); // ANELComo um floco de neve!c.stroke (); // golpe muito complicado caminho 15.8.6 recorteDepois de definir um caminho, você geralmente chama Stroke () ou Fill () (ou ambos). Você também pode chamar o método clip () para definir um recorteregião.Uma vez definido uma região de recorte, nada será desenhado fora disso.A Figura 15-13 mostra um desenho complexo produzido usando regiões de recorte.A faixa vertical correndo pelo meio e o O texto ao longo da parte inferior da figura foi acariciado sem recorteregião e depois preenchida após a definição da região de recorte triangular.

Figura 15-13.Movimentos não soltos e preenchimentos cortadosA Figura 15-13 foi gerada usando o método polygon () deExemplo 15-5 e o seguinte código:// Defina alguns atributos de desenhoC.Font = "Bold 60pt sem serif";// Big Font



```
C.LineWidth = 2;// linhas estreitasc.strokeStyle = "#000";// linhas pretas// descreva um retângulo e algum  
textoC.STRASTERECT (175, 25, 50, 325);// Uma faixa vertical para baixooc.strokeText ("<Canvas>", 15,  
330);// Nota StroKetext ()Em vez de FillText ()// Defina um caminho complexo com um interior que está  
fora.polígonos (c, 3.200.225.200); // Triângulo grande polígonos (c, 3.200.225.100,0, verdadeiro); // reverso  
menorTriângulo dentro// Faça esse caminho a região de recorte.c.clip (); // acaricia o caminho com uma linha de 5  
pixels, inteiramente dentro doregião de recorte.C.LineWidth = 10;// metade desta linha de 10 pixels  
será cortado c.stroke (); // preencha as partes do retângulo e texto que estão dentro a região de recorte c.fillstyle =  
"#aaa"; // cinza claro C.FillRect (175, 25, 50, 325); // preencha a faixa vertical c.fillstyle = "#888"; // cinza mais  
escuro C.FillText ("<Canvas>", 15, 330); // preencha o texto É importante observar que quando você chama clip (), o  
caminho atual é Ele próprio cortou para a região de recorte atual, depois esse caminho cortado torna -se a nova  
região de recorte. Isso significa que o método clip () pode encolher a região de recorte, mas nunca pode ampliá  
-la. Não há Método para redefinir a região de recorte, então antes de ligar para clip (), você deve normalmente ligar  
para salvar () para que você possa restaurar mais tarde () a região não cheia.
```

15.8.7 Manipulação de pixels O método `getImagedata()` retorna um objeto de imagem que representa os pixels brutos (como R, G, B e A componentes) de uma região retangular da sua tela. Você pode criar imagens vazias ou objetos com `createImagedata()`. Os pixels em uma imagem Objeto é gravável, para que você possa configurá-los da maneira que quiser e copiar Esses pixels de volta à tela com `PutImagedata()`. Esses métodos de manipulação de pixels fornecem acesso de nível muito baixo ao tela. O retângulo que você passa para `getImagedata()` está no padrão Sistema de coordenadas: suas dimensões são medidas em pixels CSS, e não afetado pela transformação atual. Quando você liga `putImagedata()`, a posição que você especifica também é medida no Sistema de coordenadas padrão. Além disso, `PutImagedata()` ignora Todos os atributos gráficos. Não realiza nenhuma composição, não Multiplique os pixels pela GlobalAlpha e não desenha sombras. Os métodos de manipulação de pixels são úteis para implementar a imagem processamento. Exemplo 15-8 mostra como criar um borrão de movimento simples ou Efeito ?Smamar? como o mostrado na Figura 15-14.



Erro ao traduzir esta página.

para (deixe a linha = 0; linha <altura; linha++) { // para cada linhadeixe i = linha*largura*4 + 4; // O deslocamento do segundopixel da linhafor (let col = 1; col <width; col ++, i+= 4) { // paracada colunadados [i] = (dados [i] + dados [i-4]*m)/n; // Vermelhocomponente pixeldados [i+1] = (dados [i+1]+dados [i-3]*m)/n; // Verdedados [i+2] = (dados [i+2]+dados [i-2]*m)/n; // Azuldados [i+3] = (dados [i+3]+dados [i-1]*m)/n; // alfacomponente}} // agora copie os dados de imagem manchados de volta para o mesmoposição na telac.putImagedata (pixels, x, y); } 15.9 APIs de áudio As tags html <audio> e <dide> permitem que você inclua facilmentesom e vídeos em suas páginas da web. Estes são elementos complexos com APIs significativas e interfaces de usuário não triviais. Você pode controlar a mídia Playback com os métodos play () e pause (). Você pode definir oPropriedades de volume e reprodução para controlar o volume de áudio e velocidade de reprodução. E você pode pular para um determinado momento dentro domídia configurando a propriedade CurrentTime. Não abordaremos tags <audio> e <dide> em nenhum detalhe adicional Aqui, no entanto. As subseções a seguir demonstram duas maneiras de adicionar Efeitos sonoros com roteiro em suas páginas da web. 15.9.1 O construtor Audio ()

Você não precisa incluir uma tag <audio> no seu documento HTMLPara incluir efeitos sonoros em suas páginas da web.Você podeCrie dinamicamente elementos <audio> com o DOM normalDocument.createElement () Método, ou, como um atalho, você podeBasta usar o construtor Audio ().Você não precisa adicionar oElemento criado para o seu documento para reproduzi -lo.Você pode simplesmenteChame seu método play ():// carrega o efeito sonoro com antecedência, para que esteja pronto para usoLet Soundeffft = new Audio ("Soundeffft.mp3");// Reproduza o efeito sonoro sempre que o usuário clica no mousebotãodocument.addeventListener ("clique", () => {Soundeffft.CloneNode (). Play ()// Carregar e reproduzir osom});Observe o uso de CLONENODE () aqui.Se o usuário clicar no mouseRapidamente, queremos poder ter várias cópias sobrepostas doEfeito sonoro tocando ao mesmo tempo.Para fazer isso, precisamos de múltiplosElementos de áudio.Porque os elementos de áudio não são adicionados aoDocumento, eles serão coletados de lixo quando terminarem.15.9.2 A API WebaudioAlém da reprodução de sons gravados com elementos de áudio, WebOs navegadores também permitem a geração e a reprodução dos sons sintetizadoscom a API Webaudio.Uso a API Webaudio é como conectarUm sintetizador eletrônico de estilo antigo com cordões de remendo.Com webudio,Você cria um conjunto de objetos de audionodo, que representa fontes,Transformações, ou destinos de formas de onda, e depois conectam estes

nós juntos em uma rede para produzir sons. A API não é particularmente complexo, mas uma explicação completa requer um entendimento de música eletrônica e conceitos de processamento de sinais que estão além do escopo deste livro. O código a seguir abaixo usa a API Webaudio para sintetizar umas cordas curtos que desaparecem por cerca de um segundo. Este exemplo demonstra o básico da API Webaudio. Se isso for interessante para você, você pode encontrar muito mais sobre esta API online:// Comece criando um objeto Audiocontext. Safari ainda quer// nós para usar o webkitaudiocontext em vez do Audiocontext. Deixe Audiocontext = novo(this.audiocontext || this.webkitaudiocontext)();// Defina o som base como uma combinação de três senos puros. Deixe as notas = [293.7, 370.0, 440.0];// D Major acorde: D, F# e A// Crie três osciladores para cada uma das notas que queremos jogar. Deixe osciladores = notas.map (Note => {Seja o = Audiocontext.createScilator (); o.frequency.value = Note; retornar o;});// molda o som controlando seu volume ao longo do tempo.// Começando no tempo 0 Aumentar rapidamente o volume total.// Em seguida, começando no tempo 0,1 subindo lentamente para 0. Deixe volumecontrol = audiocontext.creategain (); volumecontrol.gain.setTargetattime (1, 0, 0, 0, 02); volumecontrol.gain.setTargetattime (0, 0, 1, 0, 2); // Vamos enviar o som para o destino padrão:// os alto-falantes do usuário. Deixe os falantes = Audiocontext.Destination:

```
// Conecte cada uma das notas de origem ao controle de volumeoscillators.foreach (o => o.connect  
(volumecontrol));// e conecte a saída do controle de volume ao alto -falantes.volumecontrol.connect (palestrantes);/  
Agora comece a tocar os sons e deixe -os correr para 1,25segundos.Seja startTime =  
Audiocontext.CurrentTime;Deixe StopTime = StartTime + 1,25;oscillators.foreach (o => {o.start (starttime);O.Stop  
(StopTime);});// Se queremos criar uma sequência de sons, podemos usar o eventomanipuladoresosciladores [0]  
.AddeventListener ("Ended", () => {// Este manipulador de eventos é chamado quando a nota pararjogando});15.10  
Localização, navegação e históriaA propriedade de localização da janela e dos objetos do documentorefere -se ao  
objeto de localização, que representa o URL atual dodocumento exibido na janela e que também fornece uma API  
paraCarregando novos documentos na janela.O objeto de localização é muito parecido com um objeto URL (§11.9)  
e vocêpode usar propriedades como protocolo, nome de host, porta e caminho paraAcesse as várias partes do  
URL do documento atual.OA propriedade Href retorna todo o URL como uma string, assim como oMétodo ToString  
().
```

As propriedades de hash e pesquisa do objeto de localização são interessantes. A propriedade Hash retorna o "identificador de fragmento" parte do URL, se houver um: uma marca de hash (#) seguida por um ID do elemento. A propriedade de pesquisa é semelhante. Retorna a parte do URL que começa com um ponto de interrogação: muitas vezes algum tipo de consulta é feita. Em geral, esta parte de um URL é usada para parametrizar o URL e fornece uma maneira de incorporar argumentos nele. Enquanto estes os argumentos geralmente são destinados a scripts executados em um servidor, não há razão pela qual eles também não podem ser usados ??em páginas habilitadas para JavaScript. Os objetos de URL têm uma propriedadeSearchParams que é analisada representação da propriedade de pesquisa. O objeto de localização não tem uma propriedadeSearchParams, mas se você quiser analisarWindow.Location.Search, você pode simplesmente criar um objeto URL no objeto Localização e, em seguida, use os params de pesquisa do URL: vamos url = novo url (janela.location); Deixe consulta = url.searchparams.get ("q"); deixe numResults = parseInt (url.searchparams.get ("n") || "10"); Além do objeto de localização a que você pode se referir a Window.Location ou Document.Location, e o URL () Construtor que usamos anteriormente, os navegadores também definem um Document.url Propriedade. Surpreendentemente, o valor desta propriedade é Não é um objeto de URL, mas apenas uma string. A string contém o URL do documento atual.

15.10.1 Carregando novos documentos

Erro ao traduzir esta página.

Você também pode carregar uma nova página passando uma nova string para oAtribuir () Método do objeto de localização.Isso é o mesmo queatribuir a string à propriedade Location, no entanto, não éparticularmente interessante.O método substituir () do objeto de localização, por outro lado, ébastante útil.Quando você passa uma string para substituir (), ela éinterpretadocomo um URL e faz com que o navegador carregue uma nova página, assim comoatribuir () faz.A diferença é que substitui () substitui oDocumento atual na história do navegador.Se um script no documento ADefine a propriedade Location ou as chamadas atribuídas () para carregar o documento BE então o usuário clica no botão Voltar, o navegador voltará aDocumento A. Se você usa substituir (), então o documento A éapagado da história do navegador e quando o usuário clica na parte traseiraBotão, o navegador retorna a qualquer documento exibido antesDocumento A.Quando um script carrega incondicionalmente um novo documento, a substituição ()O método é uma escolha melhor do que atribuir ().Caso contrário, o botão traseirolevaria o navegador de volta ao documento original e o mesmoO script carregaria novamente o novo documento.Suponha que você tenha umVersão aprimorada de JavaScript da sua página e uma versão estática que faznão use JavaScript.Se você determinar que o navegador do usuário nãoSuporte as APIs da plataforma da web que você deseja usar, você pode usarlocation.Replace () para carregar a versão estática:// Se o navegador não suportar as APIs de JavaScript, nósprecisar,// redireciona para uma página estática que não usa JavaScript.if (! IsbrowsherSupported ())

location.Replace ("staticpage.html");Observe que o URL passou para substituir () é relativo.ParenteURLs são interpretados em relação à página em que aparecem, assim comoEles seriam se fossem usados ??em um hiperlink.Além dos métodos Atribuir () e substituir (), o localObjeto também define Reload (), que simplesmente faz o navegadorRecarregue o documento.15.10.2 História de navegaçãoA propriedade da história do objeto de janela refere -se à históriaobjeto para a janela.O objeto de história modela o histórico de navegaçãode uma janela como uma lista de documentos e documentos estados.O comprimentoPropriedade do objeto de história especifica o número de elementos noLista de histórico de navegação, mas por razões de segurança, os scripts não podemAcesse os URLs armazenados.(Se pudessem, qualquer script poderia bisbilhotarsua história de navegação.)O objeto de história tem métodos de back () e forward () quese comportar como os botões de trás e para frente do navegador: eles fazem oO navegador vai para trás ou para frente um passo em sua história de navegação.UMterceiro método, go (), pega um argumento inteiro e pode pular qualquernúmero de páginas adiante (para argumentos positivos) ou para trás (paraargumentos negativos) na lista de histórico:history.go (-2);// Volte 2, como clicar no botão Voltarduas vezesHISTÓRIA.GO (0);// Outra maneira de recarregar a página atual

Se uma janela contiver janelas infantis (como elementos <frame>), as histórias de navegação das janelas infantis são cronologicamente intercaladas com a história da janela principal. Isso significa issoChamando history.back () (por exemplo) na janela principal pode fazer com que uma das janelas da criança navegue de volta para um anteriormente documento exibido, mas deixa a janela principal em seu estado atual. O objeto de história descrito aqui remonta aos primeiros dias do Web quando os documentos eram passivos e todo o cálculo foi realizado no servidor. Hoje, os aplicativos da Web geralmente geram ou carregam conteúdos dinamicamente e exibir novos estados de aplicativos sem realmente carregando novos documentos. Aplicações como esses devem executar seu gerenciamento de histórico próprio se eles querem que o usuário possa usar os botões de volta e para frente (ou gestos equivalentes) para navegar de um estado de aplicação para outro de maneira intuitiva. Existem duas maneiras para conseguir isso, descrito nas próximas duas seções.

15.10.3 Gerenciamento de história com hashchange

Uma técnica de gerenciamento de história envolve localização.hash e o evento "Hashchange". Aqui estão os principais fatos que você precisa saber para entender esta técnica:

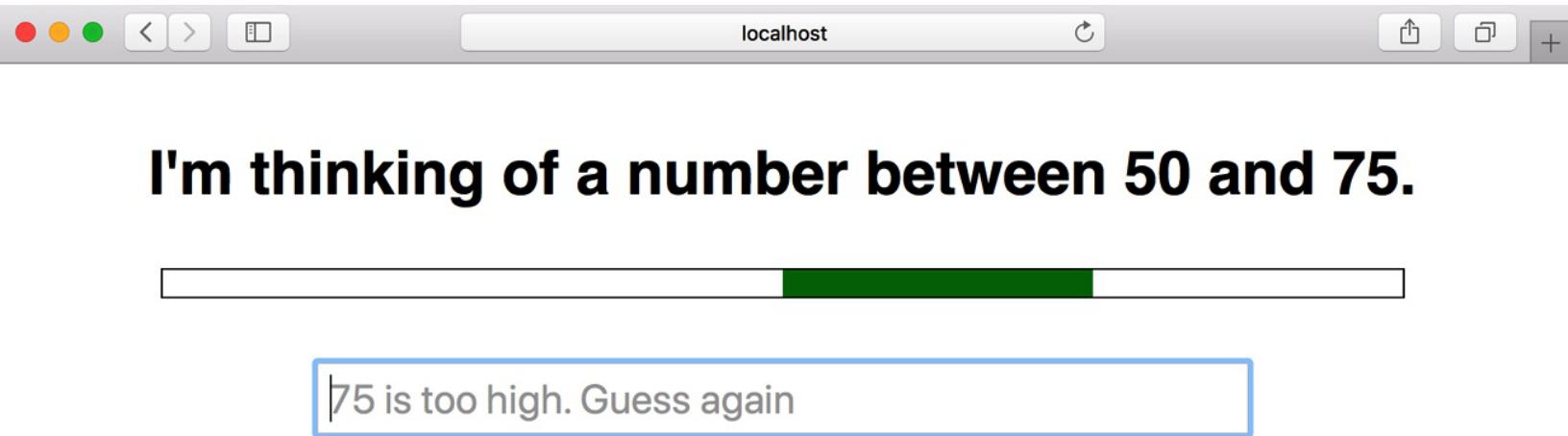
- A propriedade Location.Hash define o identificador de fragmento da URL e é tradicionalmente usado para especificar o id de uma seção de documentos para rolar. Mas location.hash não tem que ser um ID do elemento: você pode definir -lo como qualquer string. Desde que nenhum elemento tem essa string como seu id, o navegador não rola quando você define a propriedade Hash como

esse. Definindo o local.exibido na barra de localização e, muito importante, adiciona uma entrada para a história do navegador. Sempre que o identificador de fragmento do documento muda, o navegador dispara um evento "hashchange" no objeto da janela. Se você definir o local.hash, o navegador cria uma nova entrada no histórico de navegação do navegador. Então, se o usuário agora clicar no botão Voltar, o navegador irá voltar ao seu URL anterior antes de definir o local.hash. Mas isso significa que o identificador de fragmento mudou novamente. Então, outro evento "hashchange" é disparado neste caso. Isso significa que, desde que você possa criar um identificador de fragmento único para cada estado possível de seu aplicativo, eventos "hashchange" notificam o usuário se ele se mover para trás e para frente embora sua história de navegação. Para usar esse mecanismo de gerenciamento de histórico, você precisará ser capaz de decodificar as informações de estado necessárias para renderizar uma "página" da sua aplicação em uma série relativamente curta de texto adequado para uso como um identificador de fragmento. E você precisará escrever uma função para converter a página em uma string e outra função para analisar a string e recriar o estado da página que ela representa. Depois de escrever essas funções, o resto é fácil. Definir a `window.onhashchange` função (ou registre uma `hashchange` ouvir com `addEventListener()`) que lê `location.hash`, converte essa string em uma representação de seu estado de aplicação e depois toma quaisquer ações necessárias para exibir esse novo estado de aplicativo.

Erro ao traduzir esta página.

O segundo argumento pretendia ser uma string de título para o estado, masA maioria dos navegadores não o apóia, e você deve simplesmente passar um vaziocorda.O terceiro argumento é um URL opcional que será exibido ema barra de localização imediatamente e também se o usuário retornar a este estado viaBotões para trás e para frente.URLs relativos são resolvidos contra oLocalização atual do documento.Associando um URL a cada estadoPermite que o usuário marque os estados internos do seu aplicativo.Lembre -se, porém, que se o usuário salvar um marcador e depois visitarUm dia depois, você não receberá um evento "popstate" sobre essa visita: você vaitem que restaurar seu estado de aplicação analisando o URL.O algoritmo de clone estruturadoO método history.pushstate () não usa json.stringify () (§11.6) para serializar o estadodados.Em vezTécnica conhecida como algoritmo de clone estruturado, definido pelo padrão HTML.O algoritmo de clone estruturado pode serializar qualquer coisa que JSON.Stringify () pode, mas além disso, elePermite a serialização da maioria dos outros tipos de JavaScript, incluindo mapa, conjunto, data, regexp e digitadoMatrizes e pode lidar com estruturas de dados que incluem referências circulares.O clone estruturadoNo entanto, o algoritmo não pode serializar funções ou classes.Quando clonar objetos, ele não copia oObjeto de protótipo, getters e setters, ou propriedades não enumeráveis.Enquanto o clone estruturadoO algoritmo pode clonar a maioria dos tipos de javascript embutidos, não pode copiar os tipos definidos pelo hostambiente, como objetos de elemento do documento.Isso significa que o objeto de estado que você passa para a história.pushstate () não precisa se limitar aoObjetos, matrizes e valores primitivos que JSON.Stringify () suporta.Observe, no entanto, que se vocêPasse uma instância de uma classe que você definiu, essa instância será serializada como um comumJavascript Object e perderá seu protótipo.Além do método pushState (), o objeto de história tambémdefine replacestate (), que leva os mesmos argumentos, massubstitui o estado da história atual em vez de adicionar um novo estado aoHistória de navegação.Quando um aplicativo que usa pushState () éPrimeiro carregado, geralmente é uma boa ideia chamar substituição () para

Defina um objeto de estado para este estado inicial do aplicativo. Quando o usuário navega para os estados da história salva usando o traseiro ou Botões para a frente, o navegador dispara um evento "popstate" na janelaobjeto. O objeto de evento associado ao evento tem uma propriedadenomeado estado, que contém uma cópia (outro clone estruturado) doObjeto de estado que você passou para PushState (). Exemplo 15-9 é um aplicativo da web simples-o jogo de adulsão de númerosNa figura 15-15-que usa pushState () para salvar sua história, permitindo que o usuário "volte" para revisar ou refazer suas suposições. Figura 15-15.Um jogo de aviso de númeroExemplo 15-9.Gerenciamento de história com pushState ()<html> <head> <title> estou pensando em um número ... </title><estilo>



```
corpo {altura: 250px;exibição: flex;Direcção flexível: coluna;alinhado-itens: centro;Justify-Content: Space-Munly;}#heading {font: Bold 36px sans-serif;margem: 0;}#Container {Border: Solid Black 1px;Altura: 1em;largura: 80%;}#Range {Background-Color: Green;margem-esquerda: 0%;Altura: 1em;largura: 100%;}#Input {Display: Block;Size da fonte: 24px;largura: 60%;preenchimento:5px;}#playAgain {font-size: 24px;preenchimento: 10px;Radio de fronteira:5px;}</style></head><Body><h1 id = "cabeçalho"> estou pensando em um número ...</h1><!- ??uma representação visual dos números que não foram descartado -><div id = "contêiner"> <div id = "range"> </div> </div><!-onde o usuário entra em seu palpite-><input id = "input" type = "text"><!- ??Um botão que recarrega sem sequência de pesquisa.Escondido até o jogo termina.-><button id = "playagain" ocultoOnClick = "Location.Search = ";"> reproduza novamente </botão><Cript>/** Uma instância desta classe gamestate representa o interno estado de* Nossa jogo de adivinhação de número.A classe define fábrica estáticaMétodos para* Inicializando o estado do jogo de diferentes fontes, um método para* atualizar o estado com base em um novo palpite e em um método para modificar o* Documento com base no estado atual.*/classe gamestate {}// Esta é uma função de fábrica para criar um novo jogo estático newgame () {Seja s = new GameState ();S.Secret = S.Randomint (0, 100);}// Um ??número inteiro: 0 <n <100s.low = 0;// suposições devem ser maiores que isso
```

```
s.High = 100;// suposições devem serMenos do que issos.numguesses = 0;// Quantas suposições foram feitosS.Guess = NULL;// que último palpite é retornado s;}// Quando salvamos o estado do jogo com history.pushstate(), é apenas// um objeto JavaScript simples que é salvo, não uma instância do gamestate// então esta função de fábrica recria um objeto GameState com base no// Objeto simples que obtemos de um evento popstate.estático destateObject (StateObject) {Seja s = new GameState ();para (let Key of Object.Keys (StateObject)) {s [key] = stateObject [key];}retorno s;}// Para ativar a marca, precisamos ser capazes decodificar o// Estado de qualquer jogo como URL.Isso é fácil de fazer com URLSearchParams.Tourl () {vamos url = novo url (janela.location);url.searchparams.set ("l", this.low);url.searchparams.set ("h", this.high);url.searchparams.set ("n", this.numguesses);url.searchparams.set ("g", this.guess);// Observe que não podemos codificar o número secreto nourl ou isso// vai doar o segredo.Se o usuário marcar opágina com// esses parâmetros e depois retorna a ele, nós iremosBasta escolher um// novo número aleatório entre baixo e alto.retornar url.href;}
```

```
// Esta é uma função de fábrica que cria um novo GameStateobjeto e// inicializa -o a partir do URL especificado.Se  
o URL fizernão conter o// parâmetros esperados ou se forem malformados apenasretorna nulo.estático deurl (url)  
{Seja s = new GameState ();deixe parâmetros = novo URL (URL) .searchParams;s.low = parseint (params.get  
("l"));s.High = parseint (params.get ("h"));s.numguesses = parseint (params.get ("n"));s.guess = parseint  
(params.get ("g")); // Se o URL estiver faltando algum dos parâmetros de que precisamosou se// Eles não  
analisaram como números inteiros e depois retornaram nulos;if (isnan (s.low) || isnan (s.high) ||isnan  
(s.numguesses) ||isnan (S.Guess)) {retornar nulo;}// Escolha um novo número secreto no intervalo certo  
cadatempo nós// restaura um jogo de um URL.S.Secret = S.Randomint (S.Low, S.High);retorno s;// retorna um  
número inteiro n, min <n <maxrandomint (min, max) {Retornar min + math.ceil (math.random () * (máx - min -1));}//  
modifica o documento para exibir o estado atual dojogo.render () {Let Heading = Document.querySelector  
("#Heading");//O <H1> no topoLet Range = Document.querySelector ("#Range");//Exibir intervalo de adivinhação
```

```
Deixe input = document.querySelector ("#input");//Adivinhe o campo de entradaDeixe PlayAgain =
document.querySelector ("#PlayAgain");// Atualize o cabeçalho e o título do documentoheading.TextContent =
document.title = `Estou pensando em um número entre ${this.low} e ${this.high} . `;// Atualize a faixa visual de
númerosrange.style.marginleft = `${this.low}%` ;range.style.width = `${(this.high-this.low)}%`;// Verifique se o campo
de entrada está vazio e focado.input.value = "";input.focus ()// Exibe o feedback com base no último palpite do
usuário.Oentrada// espaço reservado será exibido porque fizemos a entradacampo vazio.if (this.guess === null)
{input.placeholder = "Digite seu palpite e pressioneDigitar";} else if (this.guess <this.secret) {input.placeholder = `$ 
{this.guess} é muito baixo.Adivinhe novamente`;} else if (this.guess> this.secret) {input.placeholder = `$ {this.guess} 
é muito alto.Adivinhe novamente`;} outro {input.placeholder = document.title = `${this.guess}está correto!
`;heading.TextContent = `você vence em${this.numguesses} suposições! `;playAgain.hidden = false;}// Atualize o
estado do jogo com base no que o usuárioadivinhou// retorna true se o estado foi atualizado e falso de outra forma.
```

```
updateForGuess (adivinhe) { // se for um número e está no intervalo certo if ((adivinhe > this.low) && (adivinhe < this.high)) { // Atualize o objeto de estado com base neste palpite if (adivinhe < this.secret) this.low = adivinhe; caso contrário, se (adivinhe > this.secret) this.high = adivinhe; this.guess = adivinhe; this.numguesses++; retornar true;} caso contrário { // um palpite inválido: notifique o usuário, mas não esteja de atualização alerta ('por favor digite um número maior que $ {this.low} e menos de $ {this.high}') ); retornar falso; } } // Com a classe GameState definida, fazer o jogo funcionar é apenas um assunto // de inicializar, atualizar, salvar e renderizar o estado do objeto em // Os tempos apropriados. // Quando estamos carregados pela primeira vez, tentamos obter o estado do jogo do URL // e se isso falhar, em vez disso, começamos um novo jogo. Então, se os favoritos do usuário a // jogo esse jogo pode ser restaurado do URL. Mas se carregarmos uma página com // Sem parâmetros de consulta, apenas conseguiremos um novo jogo. Let GameState = GameState.FromUrl (Window.Location) || Gamestate.NewGame (); // salve este estado inicial do jogo na história do navegador, mas use // substitua em vez de pushstate () para esta página inicial history.ReplaceState (gamestate, "", gamestate.Tourl ()) // exibe este estado inicial gamestate.render ();
```

```
// Quando o usuário adivinhar, atualize o estado do jogo com base em seu palpite// então salve o novo estado para  
a história do navegador e renderize o novo estadoDocument.querySelector ("#input"). OnChange = (Event) => {if  
(gamestate.UpDateForGuess (parseint (event.target.value)))){history.pushstate (gamestate, "", gamestate.Tourl  
())}gamestate.render ();// Se o usuário voltar ou avançar na história, obteremos umEvento PopState// no objeto  
da janela com uma cópia do objeto de estado nôssalvo com// pushState.Quando isso acontecer, renderize o novo  
estado.window.onpopstate = (event) => {gamestate = gamestate.FromStateObject (event.state); //Restaurar o  
estadogamestate.render ();// eexibi -lo};</script></body> </html>15.11 Rede de redeToda vez que você carrega  
uma página da web, o navegador faz solicitações de rede- Usando os protocolos HTTP e HTTPS - para um  
arquivo HTML, bem comoAs imagens, fontes, scripts e folhas de estilo das quais o arquivo depende.MasAlém de  
poder fazer solicitações de rede em resposta ao usuárioAções, os navegadores da web também expõem as APIs  
de JavaScript para redes comobem.Esta seção abrange três APIs de rede:
```

O método Fetch () define uma API baseada em promessa paraFazendo solicitações HTTP e HTTPS.A API Fetch ()simplifica os pedidos básicos, mas tem um abrangenteConjunto de recursos que também suporta praticamente qualquer possível uso HTTPcaso.A API de eventos enviados pelo servidor (ou SSE) é uma conveniente, eventosinterface baseada nas técnicas de ?pesquisa longa? httpO servidor da web mantém a conexão de rede aberta para que possaEnvie dados para o cliente sempre que quiser.WebSockets é um protocolo de rede que não é HTTP, mas éProjetado para interoperar com HTTP.Define umAPI assíncrona que passa por mensagens onde clientes e servidorespode enviar e receber mensagens um do outro de uma maneira queé semelhante aos soquetes de rede TCP.15.11.1 Fetch ()Para solicitações básicas de HTTP, o uso de fetch () é um processo de três etapas:1. Call Fetch (), passando pelo URL cujo conteúdo você desejar recuperar.2. Obtenha o objeto de resposta que é devolvido de forma assíncrona porEtapa 1 Quando a resposta HTTP começa a chegar e ligar para ummétodo deste objeto de resposta para pedir o corpo doresposta.3. Obtenha o objeto corporal que é devolvido de forma assíncrona pela etapa 2e processá -lo como quiser.A API Fetch () é completamente baseada em promessa e há doisPassos assíncronos aqui, então você normalmente espera duas chamadas () ouDuas aguardam expressões ao usar Fetch ().(E se você tem

Esquecido o que são, você pode querer reler o capítulo 13 antes continuando com esta seção.) Aqui está como é uma solicitação de busca () se você estiver usando então () E espere que a resposta do servidor à sua solicitação seja formatada por JSON:busca ("/api/usuários/corrente") // faça um http (ouHttps) Obtenha solicitação.Then (Response => Response.json ()) // Analisar seu corpo como umObjeto json.then (currentUser => {// então processe queobjeto analisadoDisplayUserinfo (CurrentUser);}); Aqui está uma solicitação semelhante feita usando as palavras -chave assíncronas e aguarda uma API que retorna uma corda simples em vez de um objeto JSON:função assíncrona isredserviceReady () {deixe a resposta = aguarda buscar ("/api/serviço/status"); Deixe o corpo = aguardar resposta.text (); retornar corpo === "pronto";} Se você entende esses dois exemplos de código, você sabe 80% deO que você precisa saber para usar a API Fetch (). As subseções queSiga demonstrará como fazer solicitações e receber respostas que são um pouco mais complicados do que os mostrados aqui. Adeus xmlHttPrequestA API Fetch () substitui a API barroca e enganosamente nomeada xmlhttprequest (que temnada a ver com XML). Você ainda pode ver XHR (como é frequentemente abreviado) no código existente, mas láHoje não é motivo para usá -lo em novo código e não está documentado neste capítulo. Há umExemplo de xmlhttprequest neste livro, no entanto, e você pode se referir ao §13.1.3 se quiser ver umExemplo de rede JavaScript de estilo antigo.

Códigos de status HTTP, cabeçalhos de resposta eErros de redeO processo Fetch () de três etapas mostrado em §15.11.1 elide todos os erros-Código de manuseio.Aqui está uma versão mais realista:busca ("/api/usuários/atual") // faz um http (ou https) obtersolicitar..Then (resposta => {// quando obtemos uma resposta,primeiro verifiqueif (Response.ok && // para um código de sucesso e oTipo esperado.Response.Headers.get ("Content-Type") ==="Application/json") {REPORTE DE REPORTE.JSON ();// retorna uma promessa para o corpo.} outro {lançar um novo erro (/ ou lançar um erro.`Status de resposta inesperada\$ {Response.status} ou tipo de conteúdo`);}}).THEN (CurrentUser => {// Quando o Response.json ()Promessa resolveDisplayUserinfo (CurrentUser);// Faça algo como corpo analisado.}).catch (error => {// ou se alguma coisa deu errado,Basta registrar o erro// Se o navegador do usuário estiver offline, buscar () próprio rejeitar// Se o servidor retornar uma resposta ruim, então jogaremos um erro acima.console.log ("Erro ao buscar o usuário atual:",erro);});

A promessa devolvida por fetch () resolve para um objeto de resposta.OA propriedade de status deste objeto é o código de status HTTP, como 200Para solicitações bem -sucedidas ou 404 para respostas "não encontradas".(Statustext fornece o texto em inglês padrão que acompanha oCódigo de status numérico.) Convenientemente, a propriedade OK de uma resposta é verdadeiro se o status for 200 ou qualquer código entre 200 e 299 e falso para qualquer outro código.fetch () resolve sua promessa quando a resposta do servidor começa aChegue, assim que o status HTTP e os cabeçalhos de resposta estiverem disponíveis,Mas normalmente antes que o corpo de resposta completo chegasse.Mesmo que oO corpo ainda não está disponível, você pode examinar os cabeçalhos neste segundoEtapa do processo de busca.A propriedade dos cabeçalhos de um objeto de respostaé um objeto de cabeçalhos.Use seu método Has () para testar a presença de umCabeçalho ou use seu método get () para obter o valor de um cabeçalho.HttpOs nomes dos cabeçalhos são insensíveis a maiúsculas, para que você possa passar em minúsculas ou mistasNomes de cabeçalho de casos para essas funções.O objeto de cabeçalhos também é iterável se você precisar fazer isso:busca (url) .then (resposta => {para (Let [Nome, Value] de Response.Headers) {console.log (`\$ {name}: \$ {value}`);}});Se um servidor da web responder à sua solicitação busca (), então a promessaque foi devolvida será cumprido com um objeto de resposta, mesmo que oA resposta do servidor foi um erro 404 não encontrado ou um servidor interno 500Erro.Fetch () apenas rejeita a promessa que retorna se não puder entrar em contato

o servidor da web. Isso pode acontecer se o computador do usuário estiver offline, O servidor não responde, ou o URL especifica um nome de host que faznão existe. Porque essas coisas podem acontecer em qualquer solicitação de rede, é Sempre uma boa ideia incluir uma cláusula .catch () sempre que você fizeruma chamada de busca (). Definindo parâmetros de solicitação Às vezes você quer passar parâmetros extras junto com o URL Quando você faz um pedido. Isso pode ser feito adicionando nome/valorPares no final de um URL depois de A? Os URL e URLSearchParams As aulas (que foram cobertas no §11.9) facilitam a construção de URLs Nesta forma, e a função fetch () aceita objetos de URL como seusprimeiro argumento, para que você possa incluir parâmetros de solicitação em um busca () Solicitação como este: Pesquisa de função assíncrona (termo) {vamos url = new url (" /api/search "); url.searchparams.set ("q", termo); deixe a resposta = aguarda buscar (url); if (! Response.OK) lançar um novo erro (Response.statustext); Deixe o ResultArray = Aguarda Response.json (); retornar os resultados do resultado;} Definindo cabeçalhos de solicitação Às vezes, você precisa definir cabeçalhos em suas solicitações de busca (). Se Você está fazendo solicitações de API da Web que exigem credenciais, por exemplo, então você pode precisar incluir um cabeçalho de autorização que contém Essas credenciais. Para fazer isso, você pode usar os dois argumentos versão de busca (). Como antes, o primeiro argumento é uma string ou url

objeto que especifica o URL a buscar. O segundo argumento é um objeto que pode fornecer opções adicionais, incluindo cabeçalhos de solicitação: deixe authheaders = new Headers () // Não use auth básico, a menos que esteja acima de um httpsconexão. AuthHeaders.set ("Autorização", "Basic " + btoa (" " + {nome de usuário}: {senha})); fetch (" /api/users /", {cabeçalhos: authheaders}). Then (Response => Response.json ()) // Errormanuseio omitido THEN (UsersList => DisplayAllers (UsersList)); Existem várias outras opções que podem ser especificadas no segundo argumento a buscar (), e veremos novamente mais tarde. Uma alternativa apassar dois argumentos para buscar () é passar os mesmos doisArgumentos para o construtor de solicitação () e depois passam o resultanteObjeto de solicitação para buscar (): deixe solicitação = nova solicitação (url, {cabeçalhos}); busca (solicitação) .then (resposta => ...); Analisar os corpos de respostaNo processo de busca de três etapas () que demonstramos, oO segundo passo termina chamando os métodos json () ou text () doObjeto de resposta e retornando o objeto de promessa de que esses métodosretornar. Então, o terceiro passo começa quando essa promessa resolve com o corpo da resposta analisou como um objeto JSON ou simplesmente como uma série de texto. Estes são provavelmente os dois cenários mais comuns, mas eles não sãoAs únicas maneiras de obter o corpo da resposta de um servidor da web. Em

adição a json () e text (), o objeto de resposta também tem essesMétodos:ArrayBuffer ()Este método retorna uma promessa que resolve a um ArrayBuffer.Isso é útil quando a resposta contém dados binários.Você pode usaro ArrayBuffer para criar uma matriz digitada (§11.2) ou um dataViewObjeto (§11.2.5) do qual você pode ler os dados binários.blob ()Este método retorna uma promessa que resolve a um objeto BLOB.Blobsnão estão cobertos com detalhes neste livro, mas o nome significa "Objeto grande binário" e eles são úteis quando você espera grandequantidades de dados binários.Se você pedir o corpo da resposta como umBLOB, a implementação do navegador pode transmitir os dados de resposta para um arquivo temporário e depois retorne um objeto BLOB que representa quearquivo temporário.Objetos de blob, portanto, não permitem acesso aleatório para o corpo de resposta, a maneira como um matriz faz.Uma vez que vocêter uma bolha, você pode criar um URL que se refere a eleUrl.createObjecturl (), ou você pode usar o evento baseado em eventoAPI FileReader para obter assíncrono o conteúdo do Blob como uma string ou uma matriz.No momento da redação deste artigo, algunsOs navegadores também definem o texto baseado em promessa () eMétodos ArrayBuffer () que fornecem uma rota mais direta paraobtendo o conteúdo de um blob.formData ()Este método retorna uma promessa que resolve um objeto FormData.Você deve usar este método se esperar o corpo da respostaa ser codificado no formato "Multipart/Form-Data".Este formato é comum em solicitações de postagem feitas a um servidor, mas incomum emRespostas do servidor, portanto, esse método não é usado com frequênciia.

Corpos de resposta de streaming

Além dos cinco métodos de resposta que retornam assíncronosAlguma forma do corpo de resposta completo para você, há também umopção para transmitir o corpo de resposta, o que é útil se houver algunstipo de processamento que você pode fazer nos pedaços do corpo de resposta comoEles chegam pela rede.Mas transmitir a resposta também é útilSe você quiser exibir uma barra de progresso para que o usuário possa ver oProgresso do download.A propriedade corporal de um objeto de resposta é um objeto ReadableStream.SeVocê já chamou um método de resposta como text () ou json ()Isso lê, analisa e devolve o corpo, então BodyUsed será verdadeiropara indicar que o fluxo corporal já foi lido.Se Bodysed usoué falso, no entanto, o fluxo ainda não foi lido.Nesse caso,você pode ligar para getReader () em resposta.body para obter um fluxoObjeto do leitor e use o método read () deste objeto de leitor paraLeia de forma assíncrona pedaços de texto do fluxo.O read ()o método retorna uma promessa que resolve a um objeto com feito ePropriedades do valor.feito será verdadeiro se todo o corpo for lidoou se o fluxo foi fechado.E o valor será o próximo pedaço,Como um Uint8Array, ou indefinido se não houver mais pedaços.Esta API de streaming é relativamente direta se você usar async eaguarde, mas é surpreendentemente complexo se você tentar usá -lo com cruPromessas.Exemplo 15-10 demonstra a API definindo umfunção streambody ().Suponha que você quisesse baixar um grandeJSON FILE E RELATÓRIO Download Progresso para o usuário.Você não pode fazer isso

com o método json () do objeto de resposta, mas você pode fazer isso com a função Streambody (), como esta (assumindo que uma função UpdateProgress () é definida para definir o atributo de valorem um elemento html <progress>):busca ('big.json').Then (Response => Streambody (resposta, atualização)).THEN (BodyText => json.parse (BodyText)).Then (handlebigjsonObject);A função Streambody () pode ser implementada como mostrado emExemplo 15-10.Exemplo 15-10.Tranmitindo o corpo de resposta a partir de uma solicitação de busca ()/** Uma função assíncrona para transmitir o corpo de umObjeto de resposta* obtido a partir de uma solicitação de busca ().Passar no objeto de resposta como o primeiro* Argumento seguido por dois retornos de chamada opcionais.** Se você especificar uma função como o segundo argumento, queRelatórioProgress* Retorno de chamada será chamado uma vez para cada pedaço que é recebido.O primeiro* O argumento aprovado é o número total de bytes recebidos distante.O segundo* O argumento é um número entre 0 e 1 especificando como completo o download* é.Se o objeto de resposta não tiver cabeçalho de "comprimento de conteúdo",No entanto, então* Este segundo argumento sempre será NAN.** Se você deseja processar os dados em pedaços quando eles chegam,Especifique a* funcionar como o terceiro argumento.Os pedaços serão passados,como uint8array* Objetos, para este retorno de chamada do ProcessChunk.** Streambody () retorna uma promessa que resolve uma string.Se

um ProcessChunk* O retorno de chamada foi fornecido, então esta string é a concatenação dos valores* devolvidos por esse retorno de chamada.Caso contrário, a string é a concatenação de* Os valores de bloco convertidos em strings UTF-8.*/Função assíncrona streambody (resposta, reportprogress,ProcessChunk) { // Quantos bytes esperamos, ou nan se não houver cabeçalhoDeixe o esperadoComprimento");deixe bytesRead = 0; // Quantos bytes recebido até agoraLet Reader = Response.body.getReader ()// leia bytes com esta funçãoDeixe decodificador = novo textDecoder ("UTF-8");// para converter bytes para textoDeixe Body = "";// Texto lido Sodistantewhile (true) { // loop até Saímos abaixoSeja {feito, value} = aguardar leitor.read ()// Leia apedaçado (valor) { // se conseguirmos Uma matriz de bytes:if (ProcessChunk) { // Processo os bytes se deixe processado = processChunk (valor); // o retorno foi aprovado.if (processado) {corpo += processado;} else { // caso contrário, converter bytes corpo += decodificador.decode (valor, {stream: true}); // para texto.}if (reportProgress) { // se um O retorno de chamada do progresso foi

bytesread += value.length;// passou,Então chameRelatórioProgress (BYTESRAD, BYTESRAD /esperado);}}se
(feito) {// se isso foro último pedaço,quebrar;// Saia dolaço}corpo de retorno;// retorna o texto do corpo que
acumulamos}Esta API de streaming é nova no momento da redação deste artigo e deveevoluir.Em particular, há
planos para fazer objetos readableStreamitemerável de forma assíncrona para que eles possam ser usados
??para/aguardamLoops (§13.4.1).Especificando o método de solicitação e solicitaçãoCORPOEm cada um dos
exemplos de busca () mostrados até agora, fizemos umHttp (ou https) obtenha solicitação.Se você quiser usar um
pedido diferenteMétodo (como post, put ou exclusão), basta usar os dois-Versão de argumento de fetch (),
passando um objeto de opções com umparâmetro do método:Fetch (url, {método: "post"}). Então (r =>r.json ()).
Então (HandleResponse);Publicar e colocar solicitações normalmente têm um corpo de solicitação contendo
dadosa ser enviado ao servidor.Desde que a propriedade do método não seja definida como

"Get" ou "Head" (que não suportam órgãos de solicitação), você pode especificar um corpo de solicitação definindo a propriedade do corpo das opções objeto: busca (url, {Método: "post", Corpo: "Hello World"}) Quando você especifica um corpo de solicitação, o navegador adiciona automaticamente um Cabeçalho apropriado de ?comprimento de conteúdo? para a solicitação. Quando o corpo é uma string, como no exemplo anterior, o navegador padrão Cabeçalho do tipo "Conteúdo" para "Text/Plain; Charset = UTF-8". Você pode precisar para substituir esse padrão se você especificar um corpo de sequência de mais um pouco Tipo específico, como "text/html" ou "aplicativo/json": busca (url, {Método: "post", Cabeçalhos: novos cabeçalhos {"conteúdo-tipo": "Application/json"}}, corpo: json.stringify (requestbody)}) A propriedade corporal do objeto de opções fetch () não precisa ser uma corda. Se você tiver dados binários em uma matriz digitada ou um objeto DataView ou um matriz, você pode definir a propriedade do corpo para esse valor e especificar um cabeçalho apropriado do ?tipo de conteúdo?. Se você tem dados binários na forma de blob, você pode simplesmente definir o corpo para a bolha. Blobs têm uma propriedade que especifica seu tipo de conteúdo e o valor deste A propriedade é usada como o valor padrão do cabeçalho "do tipo conteúdo". Com solicitações de postagem, é um pouco comum passar um conjunto de

Nome/Valor Parâmetros no corpo da solicitação (em vez de codificá-los na parte de consulta do URL). Existem duas maneiras de fazer isso: Você pode especificar seus nomes e valores de parâmetros com URLSearchParams (que vimos anteriormente nesta seção, que está documentado em §11.9) e depois passam pelo URLSearchParams objeto como valor da propriedade do corpo. Se você fizer isso, o corpo será definido para uma string que se pareça parte de consulta de um URL e o cabeçalho do "tipo de conteúdo" será definido automaticamente como "Aplicativo/X-Www-Form-Urlencoded; Charset = UTF-8". Se você especificar seus nomes e valores de parâmetros com um Objeto formData, o corpo usará um multipart mais detalhado. A codificação e o "tipo de conteúdo" serão definidos como "Multipart/Form-dados; limite = ?" com uma string de limite exclusiva que corresponde ao corpo. Usar um objeto FormData é particularmente útil quando os valores que você deseja carregar são longos ou são arquivos ou objetos BLOB que podem ter seu próprio "tipo de conteúdo". Objetos formData podem ser criados e inicializados com valores por passando um elemento <form> para o construtor formData(). Mas você também pode criar órgãos de solicitação "multipart/formulários" invocando o construtor formData() sem argumentos inicializando o nome do nome/valor que ele representa com os métodos set() e append(). Upload de arquivo com fetch() O upload de arquivos do computador de um usuário para um servidor da web é comumente realizada usando um objeto FormData como solicitação corpo. Uma maneira comum de obter um objeto de arquivo é exibir uma <input type = "arquivo"> elemento na sua página da web e ouvir "Change" eventos nesse elemento. Quando um evento de "mudança" ocorre, os arquivos

A matriz do elemento de entrada deve conter pelo menos um objeto de arquivo.ArquivoOs objetos também estão disponíveis na API de arrastar e soltar HTML.QueAPI não está abordada neste livro, mas você pode obter arquivos doDataTransfer.Files Array do objeto de evento passou para um evento ouvinte para eventos "drop".Lembre -se também de que os objetos de arquivo são um tipo de bolha, e às vezes pode ser útil para fazer upload de blobs.Suponha que você tenha escrito uma webAplicativo que permite ao usuário criar desenhos em um <Canvas>elemento.Você pode fazer upload dos desenhos do usuário como arquivos PNG com códigoComo o seguinte:// A função Canvas.toblob () é baseada em retorno de chamada// Este é um invólucro baseado em promessa para ele.função assíncrona getCanvasblob (canvas) {retornar nova promessa ((resolver, rejeitar) => {canvas.toblob (resolve);});}// Aqui está como enviamos um arquivo PNG de uma telafunção assíncrona uploadCanvasimage (Canvas) {Seja pngblob = aguarda getCanvasblob (tela);Seja formData = new FormData ();formData.set ("Canvasimage", pngblob);Deixe a resposta = aguarda buscar ("/upload", {método: "post", corpo: formData});Deixe o corpo = aguardar resposta.json ();}Solicitações de origem cruzadaNa maioria das vezes, o busca () é usado por aplicativos da web para solicitar dados de seu próprio servidor da web.Pedidos como esses são conhecidos como a mesma origem

solicitações porque o URL passou para buscar () tem a mesma origem(Protocol Plus HostName Plus Port) como o documento que contém oScript que está fazendo a solicitação.Por razões de segurança, os navegadores da web geralmente não perseguem (embora lá são exceções para imagens e scripts) solicitações de rede de origem cruzada.No entanto, o compartilhamento de recursos de origem cruzada, ou CORS, permite segurosolicitações de origem cruzada.Quando Fetch () é usado com uma origem cruzadaURL, o navegador adiciona um cabeçalho de "origem" à solicitação (e nãoPermitir que ele seja substituído pela propriedade dos cabeçalhos) para notificar a webservidor que a solicitação vem de um documento com um diferenteorigem.Se o servidor responder à solicitação com um apropriadoCabeçalho ?Access-Control-Allow-Origin? e a solicitação prossegue.Caso contrário, se o servidor não permitir explicitamente a solicitação, então a promessa devolvida por fetch () é rejeitada.Abortando um pedidoÀs vezes, você pode querer abortar um pedido de busca ()já emitido, talvez porque o usuário clique em um botão de cancelamento ou oO pedido está demorando muito.A API busca permite que os pedidos sejam abortadosusando as classes abortcontroller e abortSignal.(Essas classesdefinir um mecanismo de aborto genérico adequado para uso por outras APIs comobem.)Se você quiser ter a opção de abortar uma solicitação buscada (), entãoCrie um objeto abortController antes de iniciar a solicitação.OA propriedade de sinal do objeto do controlador é um objeto abortSignal.Passe este objeto de sinal como o valor da propriedade de sinal do

Opções objeto que você passa para buscar ().Tendo feito isso, você pode chamar o método abort () do objeto do controlador para abortar a solicitação, que causará quaisquer objetos de promessa relacionados à solicitação de busca parar e rejeitar com uma exceção.Aqui está um exemplo de usar o mecanismo abortcontroller para fazer cumprir um tempo limite para solicitações de busca:// Esta função é como buscar (), mas adiciona suporte a um tempo esgotado// propriedade no objeto de opções e aborta a busca se ela não está completa// dentro do número de milissegundos especificados por issopropriedade.função fetchwithtimeout (url, opções = {}) {if (options.timeout) {// se houver propriedade de tempo limite é diferente de zero let controller = novo abortController ();// Crie um controlador options.signal = controller.signal;// Defina a propriedade do sinal// Inicie um temporizador que enviará o sinal de aborto após o especificado// O número de milissegundos se passou.Observe que nós nunca cancelaremos// Este temporizador.Chamar abort () após a busca for completa has// sem efeito.setTimeout (() => {controller.abort ();}, options.timeout);}// agora apenas faça uma busca normal retornar buscar (URL, opções);}Opções de solicitação diversasVimos que um objeto de opções pode ser passado como o segundo argumento a buscar () (ou como o segundo argumento para a solicitação ())

construtor) para especificar o método de solicitação, os cabeçalhos de solicitação e a solicitação corpo. Ele também suporta várias outras opções, incluindo estas:`cacheUse` esta propriedade para substituir o cache padrão do navegador comportamento. O cache http é um tópico complexo que está além do escopo deste livro, mas se você souber algo sobre como funciona, Você pode usar os seguintes valores legais do cache: "padrão" Este valor especifica o comportamento de cache padrão. Novas respostas No cache são servidos diretamente do cache e respostas obsoletas são revalidados antes de serem servidos. "Sem lojas" Esse valor faz com que o navegador ignore seu cache. O cache não é verificado por correspondências quando a solicitação é feita e não é atualizado quando a resposta chegar. "recarregar" Este valor diz ao navegador sempre fazer uma rede normal solicitar, ignorando o cache. Quando a resposta chega, No entanto, é armazenado no cache. "Sem cache" Este valor (enganosamente nomeado) diz ao navegador para não servir Valores novos do cache. Valores em cache frescos ou obsoletos são revalidado antes de ser devolvido. "Force-cache" Este valor diz ao navegador para servir as respostas do cache Mesmo se eles forem obsoletos.

Erro ao traduzir esta página.

construído é que os clientes iniciam solicitações e servidores respondem a elas. Alguns aplicativos da web acham útil, no entanto, ter seu servidor enviar notificações quando ocorrerem eventos. Isso não é natural para HTTP, mas a técnica que foi criada é para o cliente fazer uma solicitação ao servidor, e então nem o cliente nem o servidor fechar a conexão. Quando o servidor tem algo para dizer ao cliente sobre, ele grava dados na conexão, mas a conexão permanece aberta. O efeito é como se o cliente fizesse uma solicitação de rede e o servidor responde de uma maneira lenta e estourada com pausas significativas entre explosões de atividade. Conexões de rede como essa geralmente não ficam abertas para sempre, mas se o cliente detectar que a conexão foi fechada, pode simplesmente fazer outra solicitação para reabrir a conexão. Esta técnica permite que os servidores enviem mensagens para os clientes eficientemente (embora possa ser caro no lado do servidor porque o servidor deve manter uma conexão ativa com todos os seus clientes). Porque é um padrão de programação útil, lado do cliente o JavaScript suporta o com a API do EventSource. Para criar esse tipo de conexão de longa duração para um servidor da web, basta passar uma URL para o construtor EventSource(). Quando o servidor grava (corretamente formatados) dados para a conexão, o objeto EventSource traduz aqueles em eventos que você pode ouvir:

```
let Ticker = new EventSource("Stockprices.php");
ticker.addEventListener("bid", (evento) => {displayNewbid(event.data);});
```

O objeto de evento associado a um evento de mensagem tem uma propriedade de dados que mantém qualquer string que o servidor enviou como carga útil para este evento.

O objeto de evento também possui uma propriedade de tipo, como todos os objetos de evento,Isso especifica o nome do evento.O servidor determina o tipo deos eventos que são gerados.Se o servidor omitar um nome de evento noDados que ele escreve e, em seguida, o tipo de evento é padronizado para "mensagem".O protocolo de evento enviado ao servidor é direto.O cliente inicia umconexão com o servidor (quando cria o objeto Eventsource),e o servidor mantém essa conexão aberta.Quando ocorre um evento, oO servidor grava linhas de texto na conexão.Um evento passando por cima doWire pode ficar assim, se os comentários foram omitidos:Evento: BID // define o tipo de objeto de eventoDados: Goog // Define a propriedade de dadosDados: 999 // anexa uma nova linha e mais dados// Uma linha em branco marca o final do eventoExistem alguns detalhes adicionais para o protocolo que permitem que os eventos sejamdados IDs e permitir que um cliente de reconexão para dizer ao servidor qual é o IDdo último evento que recebeu foi, para que um servidor possa reenviar quaisquer eventosperdeu.Esses detalhes são invisíveis no lado do cliente, no entanto, enão são discutidos aqui.Um aplicativo óbvio para eventos enviados ao servidor é para multiuserColaborações como bate -papo online.Um cliente de bate -papo pode usar o fetch () paraposte mensagens na sala de bate -papo e assine o fluxo de conversascom um objeto Eventsource.Exemplo 15-11 demonstra como é fácié escrever um cliente de bate -papo como esse com o EventSource.Exemplo 15-11.Um cliente de bate -papo simples usando o EventSource<html><Head> <title> SSE Chat </title> </head><Body>

```
<!-A interface do usuário de bate-papo é apenas um único campo de entrada de texto-><!- ??novas mensagens de bate-papo serão inseridas antes deste campo de entrada-><input id = "input" style = "largura: 100%; preenchimento: 10px; borda: sólidoBlack 2px "/><Cript>// Cuide de alguns detalhes da interface do usuárioDeixe Nick = Prompt ("Digite seu apelido");// Obtenha o usuárioapelidodeixe input = document.getElementById ("input");// Encontre a entradacampoinput.focus ()// Defina o tecladofoco// Registre -se para notificação de novas mensagens usando o EventSourceLet Chat = New Eventsource ("/Chat");Chat.addeventListener ("Chat", evento => {// Quando um bate -papoMensagem chegaDeixe div = document.createElement ("div");// Crie um <div>div.append (event.data);// Adicionar texto dea mensageminput.be antes (div);// e adicione divantes da entradainput.ScrollIntoView ()// Garanta a entradaELT é visível});// Publique as mensagens do usuário no servidor usando buscainput.addeventListener ("alteração", () => {// Quando o usuáriogreves retornambusca ("/chat", {// iniciar um httpsolicitação a este URL.Método: "Post", // Faça uma postagemssolicitação com o corpocorpo: nick + ":" + input.value // definido como o usuárioNick e entrada.}).catch (e => console.error); // ignora a resposta, masregistre quaisquer erros.input.value = "";// Limpe a entrada});</script></body></html>O código do lado do servidor para este programa de bate-papo não é muito mais
```

complicado que o código do lado do cliente.Exemplo 15-12 é um nó simplesServidor HTTP.Quando um cliente solicita o URL root "/", ele envia o chatCódigo do cliente mostrado no Exemplo 15-11.Quando um cliente faz um GetSolicitação para o URL "/bate -papo", ele salva o objeto de resposta e mantém issoconexão aberta.E quando um cliente faz uma solicitação de postagem para "/bate -papo",Ele usa o corpo da solicitação como uma mensagem de bate -papo e a escreve, usando oFormato "Text/Event-stream" para cada um dos objetos de resposta salvos.OO código do servidor escuta na porta 8080, então depois de executá -lo com nó, pontoSeu navegador para http://localhost: 8080 para se conectar e começarconversando consigo mesmo.Exemplo 15-12.Um servidor de bate-papo para eventos enviados ao servidor// Este é o JavaScript do lado do servidor, destinado a ser executado comNodejs// Ele implementa uma sala de bate -papo muito simples e completamente anônima// Publique novas mensagens para/bate-papo ou obtenha uma corrente de texto/evento demensagens// do mesmo URL.Fazer uma solicitação para / retorna umarquivo html simples// que contém a interface do usuário do chat do cliente.const http = requer ("http");const fs = requer ("fs");const url = requer ("url");// O arquivo html para o cliente de bate -papo.Usado abaixo.const clienthtml = fs.readFileSync ("chatclient.html");// Uma matriz de objetos ServerResponse que vamos enviarparaeventos para clientes = [];// Crie um novo servidor e ouça na porta 8080// Conecte -se a http:// localhost: 8080/para usá -lo.Deixe servidor = novo http.server ();Server.Listen (8080);// Quando o servidor receber uma nova solicitação, execute esta funçãoServer.on ("Solicitação", (solicitação, resposta) => {

```
// analisar o URL solicitado
let PathName = url.parse (request.url) .PathName; // Se a solicitação foi para "/", envie o
chat do lado do clienteUi.if (pathname === "/") { // uma solicitação para a interface do usuário de bate
-papoResponse.writeHead (200, {"content-type": "text/html"}). end (clienthtml); } // de outra forma, envie um erro 404
para qualquer caminho que não seja "/bate -papo" ou para // qualquer método que não seja "Get" e "Post" caso
contrário, if (pathname! == "/chat" ||(request.method! == "get" && request.method! == "PUBLICAR"))
{Response.writeHead (404) .nd (); } // Se a solicitação de /bate -papo foi uma obtenção, então um cliente
será conectando. Caso contrário, se (request.method === "get") {aceitnewclient (solicitação, resposta); } // Caso
contrário, a solicitação de /bate -papo é um post de uma nova mensagem outro {BroadcastNewMessage
(solicitação, resposta); }}; // Esta alça recebe solicitações para o endpoint /bate -papo que são gerados quando // o
cliente cria um novo objeto de ourcepção (ou quando oEventssource // se reconecta automaticamente). Função
AcceptNewClient (solicitação, resposta) { // Lembre -se do objeto de resposta para que possamos enviar
futuromensagens para issoclientes.push (resposta); // Se o cliente fechar a conexão, remova o correspondente //
objeto de resposta da matriz de clientes ativos
request.connection.on ("end", () => {clientes.splice (clients.indexOf
(resposta), 1);
```

```
resposta.END ();// defina cabeçalhos e envie um evento de bate -papo inicial para isso apenasum
clienteResponse.writehead (200, {"Type de conteúdo": "texto/fluxo de eventos","Conexão": "Keep-alive","Controle
de cache": "sem cache"});Response.Write ("Evento: Chat \ ndata: conectado \ n \ n");// Observe que
intencionalmente não chamamos de resposta.end ()aqui// manter a conexão aberta é o que torna o servidor
enviadoEventos funcionam.}// Esta função é chamada em resposta a postar solicitações para o/chat endpoint// que
os clientes enviam quando os usuários digitam uma nova mensagem.Função assíncrona BroadcastNewMessage
(solicitação, resposta) {// Primeiro, leia o corpo da solicitação para obter o usuário mensagem.request.setEncoding
("utf8");Deixe Body = "";// para aguardar (deixe o pedaço de solicitação) {corpo += pedaço;}// Depois de lermos o
corpo, envie uma resposta vazia feche a conexãoResponse.writehead (200) .nd ()// formate a mensagem no
formato Text/Event-stream,Prefixando cada um// linha com "dados:"deixe message = "Data:" + body.replace ("\ n",
"\ ndata:");// Dê aos dados da mensagem um prefixo que define como um evento "bate -papo"// e dê a ele um
sufixo de nova linha dupla que marca o fim do evento.Let Event = `Evento: Chat \ N $ {message} \ n \ n`;
```

// Agora envie este evento para todos os clientes de escutaclientes.ForEach (client => client.write (evento));}15.11.3 WebsocketsA API do WebSocket é uma interface simples para um complexo e poderosoProtocolo de rede.Os websockets permitem o código JavaScript no navegadorTroca facilmente mensagens de texto e binário com um servidor.Como emEventos enviados ao servidor, o cliente deve estabelecer a conexão, mas uma vezA conexão é estabelecida, o servidor pode enviar de forma assíncronamensagens para o cliente.Ao contrário da SSE, as mensagens binárias são suportadas eAs mensagens podem ser enviadas em ambas as direções, não apenas do servidor para o cliente.O protocolo de rede que habilita a WebSockets é um tipo de extensão para http.Embora a API da WebSocket seja uma reminiscência de baixo nívelsoquetes de rede, terminais de conexão não são identificados por endereço IPe porta.Em vez disso, quando você deseja se conectar a um serviço usando oProtocolo WebSocket, você especifica o serviço com um URL, assim como vocêfaria para um serviço da web.URLs WebSocket começam com WSS: //em vez de https: //, no entanto.(Os navegadores normalmente restringemWebSockets para trabalhar apenas em páginas carregadas em https seguros: //conexões).Para estabelecer uma conexão WebSocket, o navegador primeiro estabelece umConexão HTTP e envia ao servidor uma atualização: WebSocketCabeçalho solicitando que a conexão seja alterada do HTTPprotocolo para o protocolo Websocket.O que isso significa é que, a fim deUse websockets em seu JavaScript do lado do cliente, você precisará ser

Trabalhando com um servidor da web que também fala o protocolo WebSocket, você precisará ter o código do lado do servidor escrito para enviar e receber dados usando esse protocolo. Se o seu servidor estiver configurado dessa maneira, então issoA seção explicará tudo o que você precisa saber para lidar com o cliente-extremidade lateral da conexão. Se o seu servidor não suportar oProtocolo da WebSocket, considere o uso de eventos enviados pelo servidor (§15.11.2)em vez de.Criando, conectando e desconectandoWebSocketsSe você deseja se comunicar com um servidor habilitado para WebSocket, crie umObjeto WebSocket, especificando o wss:// url que identifica oServidor e serviço que você deseja usar:Deixe soquete = new websocket ("wss://example.com/stockticker"); Quando você cria um webSocket, o processo de conexão começaaautomaticamente. Mas um WebSocket recém -criado não será conectadoquando é devolvido pela primeira vez. A propriedade ReadyState do soquete especifica em que afirmaA conexão está dentro. Esta propriedade pode ter os seguintes valores:WebSocket.ConnectingEste WebSocket está conectando. WebSocket.openEste WebSocket está conectado e pronto para comunicação. WebSocket.closing

Esta conexão WebSocket está sendo fechada.`WebSocket.closed`Este webSocket foi fechado;Nenhuma comunicação adicional é possível.Este estado também pode ocorrer quando a conexão inicialA tentativa falha.Quando um WebSocket transita da conexão para a aberturaEstado, ele dispara um evento "aberto" e você pode ouvir este evento porDefinindo a propriedade `ONOPEN` do WebSocket ou ligando`addEventListenere()` nesse objeto.Se um protocolo ou outro erro ocorrer para uma conexão WebSocket, oO objeto WebSocket dispara um evento de "erro".Você pode definir o `OnError` paraDefina um manipulador ou, alternativamente, use `addEventListenere()`.Quando terminar com um webSocket, você pode fechar a conexão porChamando o método `Close ()` do objeto WebSocket.Quando aWebSocket muda no estado fechado, ele dispara um evento "próximo" eVocê pode definir a propriedade `OCLOSE` para ouvir este evento.Enviando mensagens sobre um webSocketPara enviar uma mensagem para o servidor do outro lado de um webSocketconexão, basta invocar o método `send ()` do `websocketobjeto.send ()` espera um único argumento de mensagem, que pode ser umString, Blob, ArrayBuffer, Array digitado ou objeto DataAView.O método `send ()` buffer a mensagem especificada a ser transmitidae retorna antes que a mensagem seja realmente enviada.O

propriedade bufferedamount do objeto websocket especifica oNúmero de bytes que são tamponados, mas ainda não enviados.(Surpreendentemente,Os websockets não disparam nenhum evento quando esse valor atingir 0.)Recebendo mensagens de uma webSocketPara receber mensagens de um servidor em um WebSocket, registre um eventoManipulador para eventos de "mensagem", definindo o OnMessagepropriedade do objeto WebSocket, ou ligandoaddEventListener ().O objeto associado a uma "mensagem"Evento é uma instância do MessageEvent com uma propriedade de dados que contéma mensagem do servidor.Se o servidor enviou um texto codificado UTF-8, entãoEvent.Data será uma string que contém esse texto.Se o servidor enviar uma mensagem que consiste em dados binários em vez detexto, então a propriedade de dados (por padrão) será um objeto BLOBrepresentando esses dados.Se você preferir receber mensagens binárias comoArraybuffers em vez de blobs, defina a propriedade Binarype doWebSocket objeto para a string "ArrayBuffer".Existem várias APIs da Web que usam objetos de MessageEvent paratrocando mensagens.Algumas dessas APIs usam o clone estruturadoAlgoritmo (consulte "O algoritmo de clone estruturado") para permitir o complexoEstruturas de dados como carga útil da mensagem.Websockets não é um dessesAPIs: as mensagens trocadas por um websocket são uma única stringde caracteres unicode ou uma única sequência de bytes (representada como uma bolhaou um matriz).Negociação de protocolo

O protocolo WebSocket permite a troca de texto e binário mensagens, mas não diz nada sobre a estrutura ou significado de essas mensagens. Aplicativos que usam websockets devem construir seu próprio Protocolo de comunicação em cima desta troca de mensagens simples mecanismo. O uso de WSS: // URLs ajuda com isso: cada URL irá normalmente, têm suas próprias regras sobre como as mensagens devem ser trocadas. Se você escreve código para se conectar ao wss: //example.com/stockticker, então você provavelmente sabe que você receberá mensagens sobre os preços das ações. Os protocolos tendem a evoluir, no entanto. Se uma citação hipotética de estoque O protocolo é atualizado, você pode definir um novo URL e conectar-se ao serviço atualizado como wss: //example.com/stockticker/v2. O versão baseado em URL nem sempre é suficiente, no entanto. Com protocolos complexos que evoluíram ao longo do tempo, você pode acabar com servidores implantados que suportam várias versões do protocolo e clientes implantados que suportam um conjunto diferente de versões de protocolo. Antecipando essa situação, o protocolo WebSocket e a API incluem um recurso de negociação de protocolo no nível do aplicativo. Quando você chama o WebSocket () construtor, o wss: // url é o primeiro argumento. Mas você também pode passar uma variedade de cordas como o segundo argumento. Se você faz isso, você está especificando uma lista de protocolos de aplicativos que você conhece. Como lidar e pedir ao servidor para escolher um. Durante o processo de conexão, o servidor escolherá um dos protocolos (ou falhará com erro se não suportar nenhuma das opções do cliente). Uma vez que a conexão foi estabelecida, a propriedade do protocolo do objeto WebSocket especifica qual versão do protocolo o servidor escolheu.

15.12 ArmazenamentoAplicativos da Web podem usar APIs do navegador para armazenar dados localmente no computador do usuário.Este armazenamento do lado do cliente serve para dar a web navegador uma memória.Os aplicativos da web podem armazenar preferências do usuário, por exemplo, ou até armazenar seu estado completo, para que eles possam retomar exatamente de onde você parou no final de sua última visita.O armazenamento do lado do cliente é segregado por origem, então as páginas de um site não podem ler os dados armazenados por páginas de outro site.Mas duas páginas do mesmo site podem compartilhar armazenamento e usá-lo como mecanismo de comunicação.Entrada de dados em um formulárioEm uma página, pode ser exibido em uma tabela em outra página, por exemplo.Os aplicativos da Web podem escolher a vida útil dos dados que eles armazenam: dados podem ser armazenados temporariamente para que seja retido apenas até a janela fecha ou o navegador sai, ou pode ser salvo no computador do usuário e armazenado permanentemente para que esteja disponível meses ou anos depois.Existem várias formas de armazenamento do lado do cliente:Armazenamento na webA API de armazenamento da Web consiste na localização de objetos de sessão, que são essencialmente persistentesObjetos que mapeiam as teclas String para valores de string.O armazenamento da web é muito fácil de usar e é adequado para armazenar grandes (mas não enormes) quantidades de dados.BiscoitosOs cookies são um mecanismo de armazenamento antigo do lado do cliente que foi projetado para uso por scripts do lado do servidor.Uma API JavaScript estranha faz Cookies Escritáveis no lado do cliente, mas eles são difíceis de usar e adequados apenas para armazenar pequenas quantidades de dados textuais.Além disso, qualquer

Os dados armazenados como cookies são sempre transmitidos ao servidor a cada solicitação HTTP, mesmo que os dados sejam interessantes apenas ao cliente. indexedDB é uma API assíncrona para um banco de dados de objetos que suporta indexação. Armazenamento, segurança e privacidade Os navegadores da web geralmente oferecem para lembrar senhas da web para você, e eles armazenam com segurança em formulário criptografado no dispositivo. Mas nenhuma das formas de armazenamento de dados do lado do cliente descrito neste capítulo envolve criptografia: Você deve assumir que qualquer coisa que seus aplicativos da Web salvam residem no dispositivo do usuário em forma não criptografada. Os dados armazenados são, portanto, acessíveis a usuários curiosos que compartilham acesso ao dispositivo e a software malicioso (como spyware) que existe no dispositivo. Por esta razão, nenhuma forma de armazenamento do lado do cliente deve ser usada para senhas, números de contas financeiras, ou outras informações igualmente sensíveis.

15.12.1 LocalStorage e sessionStorage

As propriedades LocalStorage e sessionStorage do objeto de janela Consulte os objetos de armazenamento. Um objeto de armazenamento se comporta assim como um objeto JavaScript comum, exceto que:

- Os valores da propriedade dos objetos de armazenamento devem ser strings.
- As propriedades armazenadas em um objeto de armazenamento persistem.

Se você definir uma propriedade do objeto LocalStorage e depois o usuário recarrega a página, o valor que você salvou nessa propriedade ainda está disponível para o seu programa. Você pode usar o objeto LocalStorage como este, por exemplo:

```
Deixe o nome = localStorage.UserName; // consulta um armazenado valor.
if (! nome) { nome = prompt ("Qual é o seu nome?"); } // pergunte ao usuário um
```

pergunta.LocalStorage.UserName = Name;// armazenar o usuárioresposta.}Você pode usar o operador de exclusão para remover as propriedades deLocalStorage e SessionStorage, e você pode usar umpara/em loop ou object.Keys () para enumerar as propriedades de umObjeto de armazenamento.Se você deseja remover todas as propriedades de um objeto de armazenamento,Chame o método clear ():LocalStorage.clear ();Objetos de armazenamento também definem getItem (), setItem () eMétodos DeleteItem (), que você pode usar em vez de diretoAcesso à propriedade e o operador Excluir, se desejar.Lembre -se de que as propriedades dos objetos de armazenamento só podem armazenarcordas.Se você deseja armazenar e recuperar outros tipos de dados, você vaitem que codificar e decodificá -lo sozinho.Por exemplo:// Se você armazenar um número, ele é automaticamente convertido em umcorda// Não se esqueça de analisá -lo ao recuperá -lo do armazenamento.LocalStorage.x = 10;Seja x = parseint (localStorage.x);// converte uma data em uma string ao definir e analisá -la quandorecebendoLocalStorage.Lastread = (new Date ()). ToutcString ();Seja Lastread = new Date (DATEPARSE (LocalStorage.Lastread));// json faz uma codificação conveniente para qualquer primitivo ou dados

estruturaLocalStorage.data = json.Stringify (dados); // codifica eloja deixa dados = json.parse (localStorage.data); // recuperar edecodificar. Vida útil e escopo de armazenamento A diferença entre LocalStorage e SessionStorage envolve a vida e o escopo do armazenamento. Dados armazenados no LocalStorage é permanente: não expira e permanece armazenados no dispositivo do usuário até que um aplicativo da web o exclua ou o usuário pergunta ao navegador (através de alguma interface do usuário específica do navegador) para excluí-lo. O LocalStorage é escopo para a origem do documento. Conforme explicado? A política da mesma origem?, a origem de um documento é definida por seuProtocolo, nome do host e porta. Todos os documentos com o mesmo compartilhamento de origens mesmos dados de armazenamento local (independentemente da origem dos scripts que realmente acessam o localStorage). Eles podem ler um do outro, e eles podem substituir os dados um do outro. Mas documentos de diferentes origens nunca podem ler ou substituir os dados um do outro (mesmo que ambos estão executando um script do mesmo servidor de terceiros). Observe que o LocalStorage também é escopo pela implementação do navegador. Se você visitar um site usando o Firefox e depois visite novamente usando o Chrome (para exemplo), quaisquer dados armazenados durante a primeira visita não serão acessíveis durante a segunda visita. Os dados armazenados através do sessionStorage têm uma vida diferente do que os dados armazenados através do localStorage: tem a mesma vida que o Guia de janela ou navegador de nível superior na qual o script que o armazenou é

correndo. Quando a janela ou a guia é fechada permanentemente, quaisquer dados armazenados através do SessionStorage são excluídos. (Observe, no entanto, que os navegadores modernos têm a capacidade de reabrir abas recentemente fechadas e restaurar a última sessão de navegação, então a vida inteira dessas guias e seus Associated SessionStorage pode ser maior do que parece.) Como LocalStorage, a SessionStorage é escopo para documentar a origem para que documentos com origens diferentes nunca Compartilhe sessionStorage. Mas sessionStorage também está escopo por uma base por janela. Se um usuário tiver duas guias do navegador exibindo documentos da mesma origem, essas duas guias têm separado Dados da SessionStorage: os scripts em execução em uma guia não podem ler ou substituir os dados escritos por scripts na outra guia, mesmo que ambas as guias estejam visitando exatamente a mesma página e estão executando exatamente o mesmo script. Eventos de armazenamento Sempre que os dados armazenados no LocalStorage mudam, o navegador desencadeia um evento de "armazenamento" em qualquer outro objeto de janela para os quais esses dados são visíveis (mas não na janela que fizeram a alteração). Se o navegador tem duas guias abertas para páginas com a mesma origem e uma delas armazena um valor no localStorage, a outra guia irá receber um evento de "armazenamento". Registre um manipulador para eventos de "armazenamento", configurando window.onstorage ou ligando window.addEventListener() com tipo de evento "armazenamento".

O objeto de evento associado a um evento de "armazenamento" tem alguns importantes propriedades:
chave O nome ou a chave do item que foi definido ou removido. Se o Clear () Método foi chamado, esta propriedade será nula. NewValue Mantém o novo valor do item, se houver um. Se RemoveItem () foi chamado, esta propriedade não estará presente. OldValue Mantém o valor antigo de um item existente que mudou ou foi excluído. Se uma nova propriedade (sem valor antigo) for adicionada, esta propriedade não estará presente no objeto de evento. Storage e o objeto de armazenamento que mudou. Este é geralmente o Objeto LocalStorage.url O URL (como uma string) do documento cujo script fez essa mudança de armazenamento. Observe que o LocalStorage e o evento "armazenamento" podem servir como um mecanismo de transmissão pelo qual um navegador envia uma mensagem para todos os Windows que atualmente estão visitando o mesmo site. Se um usuário solicitar que um site pare de executar animações, por exemplo, o site pode armazenar essa preferência no localStorage para que possa honrá-la nas visitas futuras. E armazenando a preferência, gera um evento que

permite que outras janelas exibam o mesmo site para honrar a solicitação quebem.Como outro exemplo, imagine um aplicativo de edição de imagem baseado na WebIsso permite ao usuário exibir paletas de ferramentas em janelas separadas.QuandoO usuário seleciona uma ferramenta, o aplicativo usa o LocalStorage para salvaro estado atual e para gerar uma notificação para outras janelas que umNova ferramenta foi selecionada.15.12.2 CookiesUm cookie é uma pequena quantidade de dados nomeados armazenados pelo navegador da webe associado a uma página ou site da Web específico.Os biscoitos eramProjetado para programação do lado do servidor e, no nível mais baixo, eles sãointernamente implementado como uma extensão ao protocolo HTTP.Dados de cookies sãotransmitido automaticamente entre o navegador da web e o servidor da web, entãoOs scripts do lado do servidor podem ler e escrever valores de cookies que são armazenados em o cliente.Esta seção demonstra como os scripts do lado do cliente também podemmanipular cookies usando a propriedade de cookies do documentoobjeto.Por que "cookie"?O nome "Cookie" não tem muito significado, mas não é usado sem precedentes.NoAnais da história da computação, o termo "cookie" ou "biscoito mágico" tem sido usado para se referir a umpequenopedaço de dados, particularmente um pedaço de dados privilegiados ou secretos, semelhante a uma senha, queprova a identidadeou permite acesso.No JavaScript, os cookies são usados ??para salvar o estado e podem estabelecer um tipo deidentidadePara um navegador da web.Os cookies em JavaScript não usam nenhum tipo de criptografia, no entanto, e nãosãoSegure de qualquer maneira (embora transmiti -los em um HTTPS: a conexão ajuda).A API para manipular biscoitos é antiga e enigmática.HáNão há métodos envolvidos: os cookies são consultados, definidos e excluídos pela leitura

e escrever a propriedade Cookie do objeto de documento usando cordas especialmente formatadas. A vida e o escopo de cada biscoito podem ser especificados individualmente com atributos de cookie. Esses atributos são também especificados com seqüências de cordas especialmente formatadas no mesmo cookiePropriedade. As subseções a seguir explicam como consultar e definir valores de cookies e atributos. Lendo cookies Quando você lê a propriedade Document.Cookie, ele retorna uma string que contém todos os cookies que se aplicam ao documento atual. OString é uma lista de pares de nome/valor separados um do outro por um Semicolon e um espaço. O valor do cookie é apenas o próprio valor e não inclui nenhum dos atributos que podem estar associados a issoCookie. (Vamos falar sobre atributos a seguir.) Para fazer uso da Propriedade Document.Cookie, normalmente você deve chamar a divisão ()Método para dividir-lo em pares de nome/valor individuais. Depois de extrair o valor de um biscoito da biscoitPropriedade, você deve interpretar esse valor com base em qualquer formato ou a codificação usada pelo criador do cookie. Você pode, por exemplo, passar o valor do cookie para decodeURIComponent () e depois para JSON.parse (). O código a seguir define uma função getCookie () que analisa a propriedade Document.Cookie e retorna um objeto cujas propriedades especificam os nomes e valores dos cookies do documento:

```
// retorna os cookies do documento como um objeto de mapa.// assume que os valores de cookies são codificados  
comcodeuricomponent ().function getcookies () {deixe cookies = novo map ()// o objeto que retornaremosdeixe  
tudo = document.cookie;// Obtenha todos os cookies em um grande cordadeixe list = all.split ("");// dividido em  
individualpares de nome/valorpara (vamos cookie da lista) {// para cada cookie naquele listaif (! Cookie.includes  
("=")) continuar;// Pule se lánão é = sinalSeja p = cookie.indexof ("="); // Encontre oprimeiro = sinaldeixe o nome =  
cookie.substring (0, p); // Obtenha biscoitonome deixa valor = cookie.substring (p+1); // Obtenha biscoito valor value =  
decodeuricomponent (valor); // decodifique o valor cookies.set (nome, valor); // Lembrar Nome e valor do  
biscoito}devolver cookies; }Atributos de biscoito: vida e escopoAlém de um nome e um valor, cada cookie tem  
atributos opcionais que controlam sua vida útil e escopo. Antes que possamos descrever como definirCookies com  
JavaScript, precisamos explicar os atributos de cookies. Os cookies são transitórios por padrão; Os valores que eles  
armazenam duram para a duração da sessão do navegador da web, mas são perdidos quando o usuário sai  
do navegador. Se você deseja que um biscoito dure além de uma única sessão de navegação, você deve dizer ao  
navegador quanto tempo (em segundos) você gostaria que mantenha o cookie especificando um atributo de idade  
máxima. Se você especificar um
```

Erro ao traduzir esta página.

Por padrão, os cookies são escopos por origem do documento. Grandes sites podem querer que os cookies sejam compartilhados entre os subdomínios, no entanto. Para Exemplo, o servidor em order.example.com pode precisar ler cookieValores definidos em catalog.example.com. É aqui que o domínio atributo entra. Se um cookie criado por uma página em catalog.example.com Define seu atributo de caminho para "/" e seu domínio atributo a ".example.com", esse cookie está disponível para todas as páginas da web em catalog.example.com, orders.example.com e qualquer outro servidor em O domínio Exempli.com. Observe que você não pode definir o domínio de umCookie para um domínio diferente de um domínio pai do seu servidor. O atributo final do cookie é um atributo booleano chamado seguro que especifica como os valores dos cookies são transmitidos pela rede. Por padrão, os cookies são inseguros, o que significa que eles são transmitidos sobre uma conexão HTTP normal e insegura. Se um biscoito estiver marcado Seguro, no entanto, é transmitido apenas quando o navegador e o servidor estão conectados via HTTPS ou outro protocolo seguro. Limitações de biscoitos Os cookies destinam-se ao armazenamento de pequenas quantidades de dados por scripts do lado do servidor, e esses dados são transferidos para o servidor sempre que um URL relevante é solicitado. O padrão que define cookies incentiva os fabricantes de navegadores a permitir um número ilimitado de cookies de tamanho irrestrito, mas não exige que os navegadores retenham mais de 300 cookies no total, 20 cookies por servidor da web ou 4 kb de dados por cookie (nome e valor da contagem de valores para esse limite de 4 kb). Na prática, os navegadores permitem muitos mais de 300 cookies no total, mas o limite de tamanho de 4 kb ainda pode ser aplicado por alguns. Armazenando biscoitos Para associar um valor de cookie transitório ao documento atual, simplesmente Defina a propriedade Cookie como um nome = string de valor. Por exemplo:

document.cookie = `versão = \${codeuricomponent (document.lastmodified)}`;Na próxima vez que você ler a propriedade do cookie, o nome do nome/valor vocêO armazenado está incluído na lista de cookies para o documento.Valores de biscoitoNão pode incluir semicolons, vírgulas ou espaço em branco.Por esse motivo,Você pode querer usar a função global JavaScript centralcodeuricomponent () para codificar o valor antes de armazená-lo em biscoito.Se você fizer isso, terá que usar o correspondenteDecodeuricomponent () função quando você lê o cookievalor.Um cookie escrito com um nome simples de nome/valor dura para a correnteSessão de navegação na Web, mas é perdida quando o usuário sai do navegador.ParaCrie um cookie que possa durar nas sessões do navegador, especifique seuLifetime (em segundos) com um atributo em idade máxima.Você pode fazer isso porDefinir a propriedade Cookie como uma string do formulário: name = value;Max-Aage = segundos.A função a seguir define um cookie com umAtributo opcional Max-Age:// armazenar o par de nome/valor como um cookie, codificando o valorcom// codeuricomponent () para escapar de semicolons,vírgulas e espaços// Se Daystolive for um número, defina o atributo da era máxima paraque o biscoito// expira após o número especificado de dias.Passe 0 paraExclua um biscoito.função setcookie (nome, valor, diastolive = null) {Seja cookie = `\${name} = \${codeuricomponent (value)}`;if (Daystolive! == null) {Cookie += `;max-Arane = \${DayStolive*60*60*24}`};}document.cookie = cookie;

Erro ao traduzir esta página.

Os bancos de dados indexedDB são escoposDocumentos: Duas páginas da web com a mesma origem podem acessá-los, mas páginas da web de diferentes origens não podem. Cada origem pode ter qualquer número de bancos de dados indexedDB. Cada banco tem um nome que deve ser único dentro da origem. No indexedDB API, um banco de dados é simplesmente uma coleção de lojas de objetos nomeadas. Como o nome implica, um loja de objetos armazena objetos. Objetos são serializados para o armazenamento de objetos usando o algoritmo de clone estruturado (consulte [?O Algoritmo de clone estruturado ?](#)), o que significa que os objetos que você armazena podem ter propriedades cujos valores são mapas, conjuntos ou matrizes digitadas. Cada objeto deve ter uma chave pela qual pode ser classificado e recuperado da loja. As chaves devem ser únicas - dois objetos na mesma loja podem não ter a mesma chave - e eles devem ter uma ordem natural para que eles possam ser classificados. Strings de JavaScript, números e objetos de data são válidos chaves. Um banco de dados indexedDB pode gerar automaticamente uma chave única para cada objeto que você insere no banco de dados. Muitas vezes, porém, os objetos que você insere em um armazenamento de objetos já terão uma propriedade que é adequada para uso como chave. Nesse caso, você especifica um "caminho-chave" para isso. Propriedade ao criar o armazenamento de objetos. Conceitualmente, um caminho-chave é um valor que informa ao banco de dados como extrair a chave de um objeto do objeto. Além de recuperar objetos de um armazenamento de objetos por seu primário valor da chave, você pode querer pesquisar com base no valor de outras propriedades no objeto. Para poder fazer isso, você pode definir qualquer número de índices no armazenamento de objetos. (A capacidade de indexar um armazenamento de objetos explica o nome "indexedDB".) Cada índice define uma chave secundária para os objetos armazenados. Esses índices não são geralmente

Objetos únicos e múltiplos podem corresponder a um único valor de chave. IndexedDB fornece garantias de atomicidade: consultas e atualizações para o banco de dados está agrupado em uma transação para que todos tenham sucesso juntos ou todos falham juntos e nunca saem do banco de dados em um estado indefinido e parcialmente atualizado. Transações no indexedDB são mais simples do que em muitas APIs de banco de dados; Vamos mencioná-las novamente mais tarde. Conceitualmente, a API indexedDB é bastante simples. Para consultar ou atualizar um banco de dados, você abre primeiro o banco de dados que deseja (especificando-o pelo nome). Em seguida, você cria um objeto de transação e usa esse objeto para procurar o armazenamento de objetos desejado dentro do banco de dados, também pelo nome. Finalmente, você procura um objeto chamando o método get () do armazenamento de objetos ou armazene um novo objeto chamando put () (ou chamando add (), se você quiser evitar a substituição de objetos existentes). Se você quiser procurar os objetos para uma variedade de chaves, você cria um objeto IdRange que especifica os limites superior e inferior e passa para os métodos getAll () ou OpenCursor () o armazenamento de objetos. Se você deseja fazer uma consulta usando uma chave secundária, você procure o nomeado índice do armazenamento de objetos e ligue para o get (), getAll () ou OpenCursor () métodos do objeto de índice, passando uma única chave ou um objeto IdRange. Essa simplicidade conceitual da API indexedDB é complicada, No entanto, pelo fato de a API ser assíncrona (para que os aplicativos da web possam usá-lo sem bloquear o thread principal da interface do navegador). Indexeddb

foi definido antes que as promessas fossem amplamente apoiadas, então a API é baseada em eventos, em vez de baseado em promessa, o que significa que não trabalhe com assíncrono e aguarde. Criar transações e procurar lojas de objetos e índices são operações síncronas. Mas abrir um banco de dados, atualizando um objeto, a consulta de uma loja ou índice são todas operações assíncronas. Todos esses métodos assíncronos retornam imediatamente um objeto de solicitação. O navegador aciona um evento de sucesso ou erro no objeto de solicitação. Quando a solicitação é bem-sucedida ou falha, você pode definir manipuladores para as propriedades OnSuccess e OnError. Dentro de um AccessManipulador, o resultado da operação está disponível como propriedade de resultado do objeto de solicitação. Outro evento útil é o evento "completo" despachado em objetos de transação quando uma transação foi concluída com sucesso. Uma característica conveniente desta API assíncrona é que ela simplifica gerenciamento de transações. A API indexedDB força você a criar um objeto de transação para obter o armazenamento de objetos no qual você pode executar consultas e atualizações. Em uma API síncrona, você esperaria para marcar explicitamente o fim da transação chamando um método commit(). Mas com indexedDB, as transações são automaticamente comprometidas (se você não as abortar explicitamente) quando todos os manipuladores de eventos OnSuccess forem executados e não há mais solicitações assíncronas que se referem a essa transação. Há mais um evento importante para a API IndexedDB. Quando você abre um banco de dados pela primeira vez, ou quando você incrementa o

Número da versão de um banco de dados existente, indexeddb dispara umEvento "Atualizada" no objeto de solicitação retornado peloIndexedb.open () CALL.O trabalho do manipulador de eventos paraEventos "Atualizados" é definir ou atualizar o esquema para o novobanco de dados (ou a nova versão do banco de dados existente).Para indexedDBbancos de dados, isso significa criar armazenamentos de objetos e definir índices emEsses objetos armazenam.E, de fato, a única vez que a API indexedDBpermite que você crie um armazenamento de objetos ou um índice é uma resposta a umEvento "Upgradeneeded".Com esta visão geral de alto nível do indexedDB em mente, você deve agoraser capaz de entender o exemplo 15-13.Esse exemplo usa indexedDBPara criar e consultar um banco de dados que nos mapeia códigos postais (códigos postais) paraCidades dos EUA.Demonstra muitos, mas não todos, das características básicas deIndexedb.O exemplo 15-13 é longo, mas bem comentado.Exemplo 15-13.Um banco de dados indexedDB dos EUA Códigos postais// Esta função de utilidade obtém assíncronoobjeto (criando// e inicializando o db, se necessário) e passa para oligar de volta.função withdb (retorno de chamada) {let request = indexeddb.open ("zipcodes", 1); // Solicitação v1do banco de dadosrequest.onerror = console.error; // registre quaisquer errosrequest.Onsuccess = () => {}// ou ligue para isso quando feitoSeja db = request.Result; // o resultado do pedidoé o banco de dadosretorno de chamada (dB); // Invoque o retorno de chamada como banco de dados}; // Se a versão 1 do banco de dados ainda não existir, entâoeste evento// O manipulador será acionado.Isso é usado para criar e

inicializar// Objetos armazenam e índices quando o banco de dados é criado pela primeira vez ou para modificar// eles quando mudamos de uma versão do esquema de banco de dados para outro.request.onUpGradEneeded = () => {initdb (request.result, ligar de volta);};// withdb () chama essa função se o banco de dados não tiver sido inicializado ainda// Configuramos o banco de dados e o preenchemos com dados, depois passamos o banco de dados para// a função de retorno de chamada./// Nossa base de dados de código ZIP inclui um armazenamento de objetos que mantém objetos como este:///{// ZIPCODE: "02134",// cidade: "Allston",// estado: "ma",//}/// Usamos a propriedade "ZipCode" como a chave do banco de dados e criamos um índice para// O nome da cidade.function initdb (db, retorno de chamada) { // Crie o armazenamento de objetos, especificando um nome para a loja// Um ??objeto de opções que inclui o "caminho -chave" especificando o// Nome da propriedade do campo Chave para esta loja.Let Store = DB.CreateObjectStore ("ZipCodes", // Nome da loja{Keypath: "ZIPCODE"}); // agora indexe o armazenamento de objetos pelo nome da cidade, bem como por CEP// Com este método, a chave do caminho da chave é passada diretamente como um// argumento exigido e não como parte de uma opção de objeto.store.createIndex ("cidades", "cidade"); // Agora obtenha os dados que vamos inicializar o banco de dados com.

```
// O arquivo de dados zipcodes.json foi gerado a partir de CC-dados licenciados de//  
www.geonames.org:https://download.geonames.org/export/zip/us.zipbusca ("zipcodes.json") // faça um http  
obtersolicitar.Then (Response => Response.json ()) // Analisar o corpocomo JSON.THEN (ZIPCODES => {}// Get  
40K CEPPregistros// para inserir dados de código postal noBanco de dados que precisamos de um// Objeto de  
transação.Para criar nossa transaçãoobjeto, precisamos// Para especificar quais armazenamentos de objetos  
usaremos(Nós só temos// um) e precisamos dizer que estaremos fazendoescreve para o// banco de dados, não  
apenas lê:Deixe transação = db.transaction (["ZipCodes"],"readwrite");transaction.onerror = console.error;//  
Obtenha nossa loja de objetos na transaçãoodeixe armazenar = transação.ObjectStore ("ZipCodes");// A melhor  
parte da API indexedDB é queArmazenamento de objetos// são * realmente * simples.Veja como adicionamos  
(ouatualização) nossos registros:para (deixe o registro de zipcodes) {store.put (registro);} // Quando a transação é  
concluída com êxito, obanco de dados// está inicializado e pronto para uso, para que possamos ligaro// função de  
retorno de chamada que foi originalmente passado parawithdb ()transaction.oncomplete = () => {retorno de  
chamada (db);};});} // Dado um código postal, use a API indexedDB para assíncronaProcure a cidade// com esse  
código postal e passa para o retorno de chamada especificado,
```

ou passar nulo se// Nenhuma cidade é encontrada.função lookupcity (zip, retorno de chamada) {withdb (db => {}// Crie um objeto de transação somente leitura para issoconsulta.O// O argumento é uma variedade de lojas de objetos que precisaremospara usar.Deixe transação = db.transaction (["ZIPCODES"]);// Obtenha a loja de objetos da transaçãoSeja zipcodes = transaction.ObjectStore ("ZipCodes");// agora solicita o objeto que corresponda ao especificadoChave ZIPCODE// As linhas acima eram síncronas, mas esta éassíncrono.deixe solicitação = zipcodes.get (zip);request.onerror = console.error;// erros de logrequest.Onsuccess = () => {}// ou chame issofunção no sucessodeixe registrar = request.Result;// Esta é a consultaresultadose (registro) {}// se encontrarmos uma correspondência, passe para o retorno de chamadareturno de chamada ('\$ {Record.city}, \$ {Record.state}`');} else {}// caso contrário, diga ao retorno de chamada que não falhamosreturno de chamada (nulo);});});};// Dado o nome de uma cidade, use a API indexedDB paraassíncrono// Procure todos os registros de código ZIP para todas as cidades (em qualquer estado)isso tem// esse nome (sensível ao caso).função lookupzipcodes (cidade, retorno de chamada) {withdb (db => {}// Como acima, criamos uma transação e obtemos o objetoloja

Deixe transação = db.transaction(["ZIPCODES"]);deixe armazenar = transação.ObjectStore ("ZipCodes");// Desta vez, também obtemos o índice da cidade do objeto e deixe index = store.index ("cidades");// Peça todos os registros correspondentes no índice com o especificado// nome da cidade, e quando os pegamos, passamos para oligar de volta// Se esperávamos mais resultados, poderíamos usarOpenCursor () em vez disso.Let request = index.getall (cidade);request.onerror = console.error;request.Onsuccess = () => {retorno de chamada (request.Result);});}15.13 fios de trabalhador e mensagensUma das características fundamentais do JavaScript é que ele é únicoTrebo: um navegador nunca executará dois manipuladores de eventos ao mesmo tempo,e nunca acionará um cronômetro enquanto um manipulador de eventos está em execução, para exemplo.Atualizações simultâneas para o estado do aplicativo ou para o documentosimplesmente não são possíveis, e os programadores do lado do cliente não precisamPense em programação simultânea ou mesmo entender.Um corolárioÉ que as funções JavaScript do lado do cliente não devem levar muito tempo;Caso contrário, eles amarrarão o loop do evento e o navegador da web iránão responder à entrada do usuário.Esta é a razão pela qual Fetch () é uma função assíncrona, por exemplo.Navegadores da web relaxam com muito cuidado o requisito de thread único comA classe do trabalhador: as instâncias desta classe representam threads que executam

Simultaneamente com o encadeamento principal e o loop do evento. Trabalhadores vivem em um ambiente de execução independente com um completamente independente. Objeto global e sem acesso à janela ou objetos de documento. Os trabalhadores podem se comunicar com o tópico principal apenas através de mensagens assíncronas passando. Isso significa que simultaneamente modificações do DOM permanecem impossíveis, mas também significa que você pode escrever funções de longa duração que não param o loop do evento e pendure o navegador. Criar um novo trabalhador não é um peso pesado operação como abrir uma nova janela do navegador, mas os trabalhadores não são? fibras? de peso mosca e não faz sentido criar novos trabalhadores para realizar operações triviais. Aplicativos da Web complexos podem ser úteis criando dezenas de trabalhadores, mas é improvável que uma aplicação com centenas ou milhares de trabalhadores seja prática. Os trabalhadores são úteis quando seu aplicativo precisa executar tarefas computacionalmente intensivas, como processamento de imagens. Usando o trabalhador move tarefas como essa para fora do fio principal para que o navegador não se torna sem resposta. E os trabalhadores também oferecem a possibilidade de dividir o trabalho entre vários tópicos. Mas os trabalhadores também são útil quando você precisa executar frequentes tarefas moderadamente intensivas de cálculos. Suponha, por exemplo, que você esteja implementando um editor de código simples no navegador e deseja incluir destaque da sintaxe. Para acertar o destaque, você precisa analisar o código em cada tecla. Mas se você fizer isso no tópico principal, é provável que o código de análise impedirá os manipuladores de eventos que respondem ao usuário. Os principais golpes de execução prontamente e a experiência de digitação do usuário será lento. Como em qualquer API de roteamento, existem duas partes na API do trabalhador. O

Primeiro é o objeto de trabalhador: é assim que um trabalhador se parece dolá fora, para o tópico que o cria.O segundo é o `WorkerGlobalScope`: este é o objeto global para um novo trabalhador, e eleé como é um tópico de trabalhador, por dentro, para si mesmo.As seções a seguir cobrem o trabalhador e o `WorkerGlobalScope` e tambémExplique a API que passa de mensagem que permite que os trabalhadores se comuniquemcom o fio principal e um ao outro.A mesma API de comunicação éusado para trocar mensagens entre um documento e `<frame>`elementos contidos no documento, e isso é coberto noSeções a seguir também.15.13.1 Objetos trabalhadoresPara criar um novo trabalhador, ligue para o construtor do trabalhador (), passando umURL que especifica o código JavaScript que o trabalhador deve executar:Deixe DatacRuncher = new Worker ("Utils/Cruncher.js");Se você especificar um URL relativo, ele será resolvido em relação ao URL dodoocumento que contém o script que chamou de trabalhador ()construtor.Se você especificar um URL absoluto, deve ter o mesmoOrigem (mesmo protocolo, host e porta) como o documento que contém.Depois de ter um objeto de trabalhador, você pode enviar dados para ele com`PostMessage ()`.O valor que você passa para `PostMessage ()` serácopiado usando o algoritmo de clone estruturado (consulte ?O clone estruturadoAlgoritmo ?), e a cópia resultante será entregue ao trabalhador viaUm evento de mensagem:

datacruncher.postMessage ("/api/data/to/crunch");Aqui estamos apenas passando uma única mensagem de string, mas você também pode usarObjetos, matrizes, matrizes digitadas, mapas, conjuntos e assim por diante.Você pode recebermensagens de um trabalhador ouvindo eventos de "mensagem" noObjeto de trabalhador:
datacruncher.onmessage = function (e) {deixe estatísticas = e.data;// A mensagem é a propriedade de dadosdo eventoconsole.log ('média: \$ {stats.mean}');}Como todas as metas de eventos, os objetos do trabalhador definem o padrãoaddEventListener () e RemovEventListener ()Métodos, e você pode usá -los no lugar da OnMessage.Além de PostMessage (), os objetos trabalhadores têm apenas um outrométodo, terminine (), que força um fio de trabalhador a pararcorrendo.
15.13.2 O objeto global em trabalhadoresQuando você cria um novo trabalhador com o construtor trabalhador (), vocêEspecifique o URL de um arquivo de código JavaScript.Esse código é executado em umNovo ambiente de execução de JavaScript, isolado doScript que criou o trabalhador.O objeto global para essa nova execuçãO ambiente é um objeto WorkerglobalsCope.Um Workerglobalscope éalgo mais do que o objeto global Javascript central, mas menos de umObjeto de janela do lado do cliente completo.

O objeto WorkerglobalsCope possui um método PostMessage () e uma propriedade de manipulador de eventos onmessage que é como os dosObjeto de trabalhador, mas trabalhe na direção oposta: chamandoPostMessage () dentro de um trabalhador gera um evento de mensagem para o trabalhador e as mensagens enviadas de fora do trabalhador são transformadas em eventos e entregues ao manipulador de OnMessage. Porque o Workerglobalscope é o objeto global para um trabalhador, PostMessage () e OnMessage parecem uma função global e Variável global para o código do trabalhador. Se você passar um objeto como o segundo argumento para o trabalhador () construtor, e se esse objeto tiver uma propriedade de nome, o valor de Essa propriedade se torna o valor da propriedade do nome no objeto global. Um trabalhador pode incluir esse nome em qualquer mensagemImprima com console.warn () ou console.error (). A função Close () permite que um trabalhador se encerre, e é semelhante em vigor ao método termine () de um objeto de trabalhador. Como o Workerglobalscope é o objeto global para os trabalhadores, ele tem todasAs propriedades do objeto global Javascript central, como o JSONobjeto, a função isNaN () e o construtor date (). Em Além disso, no entanto, o workerglobalscope também tem o seguintePropriedades do objeto de janela do lado do cliente: O eu é uma referência ao próprio objeto global. Workerglobalscope não é um objeto de janela e nãoDefina uma propriedade de janela.

Os métodos do timer setTimeout (), clearTimeout (),setInterval () e ClearInterval ().Uma propriedade de localização que descreva o URL que foiPassado para o construtor Worker ().Esta propriedade refere -se a umObjeto de localização, exatamente como a propriedade de localização de uma janelafaz.O objeto de localização possui propriedades href, protocolo,Host, HostName, Port, Pathname, Search e Hash.Em um trabalhador, essas propriedades são somente leitura, no entanto.Uma propriedade de navegador que se refere a um objeto compropriedades como as do objeto de navegador de uma janela.UMO objeto Navigator do trabalhador tem o nome de propriedades AppName,AppVersion, Platform, UserAgent e Online.Os métodos de destino de evento usuais addEventListener ()e removeEventListener ().Finalmente, o objeto WorkerglobalsCope inclui um importante lado ao clienteAPIs JavaScript, incluindo o objeto do console, a função fetch (),e a API indexedDB.Workerglobalscope também inclui oTrabalhador () construtor, o que significa que os threads de trabalhadores podem criarseus próprios trabalhadores.15.13.3 Importar código para um trabalhadorOs trabalhadores foram definidos em navegadores da web antes do JavaScript ter um móduloSistema, portanto, os trabalhadores têm um sistema exclusivo para incluir código adicional.Workerglobalscope define o importaScripts () como um globalfunção que todos os trabalhadores têm acesso a:// Antes de começarmos a trabalhar, carregue as aulas e utilitáriosVamos precisarImportScripts ("utils/histogram.js", "utils/bitset.js");

`ImportsScripts()` leva um ou mais argumentos de URL, cada um dosque deve se referir a um arquivo de código JavaScript.URLs relativos sãoresolvido em relação ao URL que foi passado para o trabalhador ()construtor (não em relação ao documento que contém).`ImportScripts()` Carrega e executa de maneira síncronaum após o outro, na ordem em que foram especificados.Se carregarUm script causa um erro de rede, ou se a execução lança um erro de qualquerClassifique, nenhum dos scripts subsequentes é carregado ou executado.Um scriptCarregado com importação () pode ser chamadol`ImportScripts()` para carregar os arquivos depende.Nota, no entanto,que o `importaScripts()` não tenta acompanhar quais scriptsjá carregou e não faz nada para evitar ciclos de dependência.`ImportScripts()` é uma função síncrona: não retornaaté que todos os scripts tenham carregado e executado.Você pode começar a usaros scripts que você carregou assim que o `importScripts()` retorna: láNão é necessário um retorno de chamada, manipulador de eventos, então () método ou aguarda.Depois de internalizar a natureza assíncrona do lado do clienteJavaScript, é estranho voltar a simples, síncronoprogramação novamente.Mas essa é a beleza dos tópicos: você pode usar umBloqueio de chamadas de função em um trabalhador sem bloquear o loop de evento em o fio principal, e sem bloquear os cálculos sendorealizada simultaneamente em outros trabalhadores.Módulos em trabalhadoresPara usar os módulos nos trabalhadores, você deve passar um segundo argumento para o construtor `Worker()`.Este segundo argumento deve ser um objeto com uma propriedade de tipo definida para a sequênciawww."Module".Passando aTIPO: Opção "Módulo" para o construtor `Worker()` é muito parecido com o uso do tipo = "módulo"atributo em uma tag html <script>: significa que o código deve ser interpretado como um módulo e

essas declarações de importação são permitidas. Quando um trabalhador carrega um módulo em vez de um script tradicional, o `workerGlobalScope` não define a função `importScripts()`. Observe que, no início de 2020, o Chrome é o único navegador que suporta módulos e importação verdadeiros de declarações em trabalhadores.

15.13.4 Modelo de execução do trabalhador

Os threads trabalhadores executam seu código (e todos os scripts ou módulos importados) sincronamente de cima para baixo e depois entram em uma fase assíncrona em que eles respondem a eventos e temporizadores. Se um trabalhador registrar um manipulador de eventos de "mensagem", ele nunca sairá enquanto houver uma possibilidade de que os eventos de mensagem ainda cheguem. Mas se um trabalhador não ouvir mensagens, ele será executado até que não haja mais tarefas pendentes (como buscar promessas e temporizadores) e todos os retornos de chamada relacionados à tarefa foram chamados. Depois que todos os retornos de chamada registrados foram chamados, não é de maneira alguma para um trabalhador iniciar uma nova tarefa, por isso é seguro para o tópico `saia`, o que fará automaticamente. Um trabalhador também pode fechar explicitamente ele mesmo chamando a função `GlobalClose()`. Observe que aí não são propriedades ou métodos no objeto de trabalhador que especifique se um fio de trabalhador ainda está funcionando ou não, então os trabalhadores não devem fechar de alguma forma, sem coordenar isso com o tópico dos pais. Erros em trabalhadores

Se ocorrer uma exceção em um trabalhador e não for pegado por nenhuma captura de cláusula, então um evento de "erro" é acionado no objeto global do trabalhador. Se este evento for manuseado e o manipulador chamar `o.preventDefault()`, o método do objeto de evento, o erro é propagado. Caso contrário, o evento "erro" é demitido no trabalhador.

objeto.Se preventDefault () for chamado lá, então a propagação termina.Caso contrário, uma mensagem de erro é impressa no console do desenvolvedor o manipulador OnError (§15.1.7) do objeto da janela é invocado.// lida com erros de trabalhador não capturados com um manipulador dentro do trabalhador.self.onerror = function (e) {console.log (`Erro no Trabalhador em\$ {e.filename}: \$ {e.lineno}: \$ {e.message}`);E.PreventDefault ();};// ou, lide com erros de trabalhador não capturados com um manipulador do lado de fora trabalhador.trabalhador.onerror = function (e) {console.log ('Erro no Trabalhador em\$ {e.filename}: \$ {e.lineno}: \$ {e.message}`);E.PreventDefault ();};Como o Windows, os trabalhadores podem registrar um manipulador para ser invocado quando uma promessa é rejeitada e não há função .catch () para lidar com isso.Dentro de um trabalhador, você pode detectar isso definindo uma função de Self.OnUnHandledRejection ou usando addEventListener () para registrar um manipulador global paraEventos de ?rejeição não entrega?.O objeto de evento passou para este manipulador terá uma propriedade de promessa cujo valor é o objeto de promessa rejeitado e uma propriedade pela qual a propriedade cujo valor é o que teria sidoPassado para uma função .catch ().15.13.5 PostMessage (), Messageports eMessagechannels

O método PostMessage () do objeto de trabalhador e o globalPostMesage () função definida dentro de um trabalhador que trabalha porInvocando os métodos de PostMessage () de um par de Messageportobjetos que são criados automaticamente junto com o trabalhador.Cliente-JavaScript lateral não pode acessar diretamente esses criados automaticamenteMessagePort Objects, mas pode criar novos pares de portas conectadascom o construtor messagechannel ():deixe canal = new Messagechannel;// Crie anovo canal.deixe myport = canal.port1;// temduas portasdeixe yourport = canal.port2;// conectadoum com o outro.myport.postMessage ("Você pode me ouvir?");// uma mensagemPostado em um Willyourport.onMessage = (e) => console.log (e.data);// serrecebido no outro.Um Messagechannel é um objeto com propriedades Port1 e Port2que se referem a um par de objetos de Messageport conectados.Um messageport éUm objeto com um método PostMessage () e um evento onMessagePropriedade do manipulador.Quando PostMessage () é chamado em um porto de umPar conectado, um evento de "mensagem" é disparado na outra porta no par.Você pode receber esses eventos de "mensagem" definindo o OnMessagepropriedade ou usando addEventListener () para registrar um ouvintepara eventos de "mensagem".As mensagens enviadas para uma porta são na fila até que a propriedade OnMessage seja definido ou até que o método start () seja chamado na porta.Esseimpede que as mensagens enviadas por uma extremidade do canal sejam perdidas

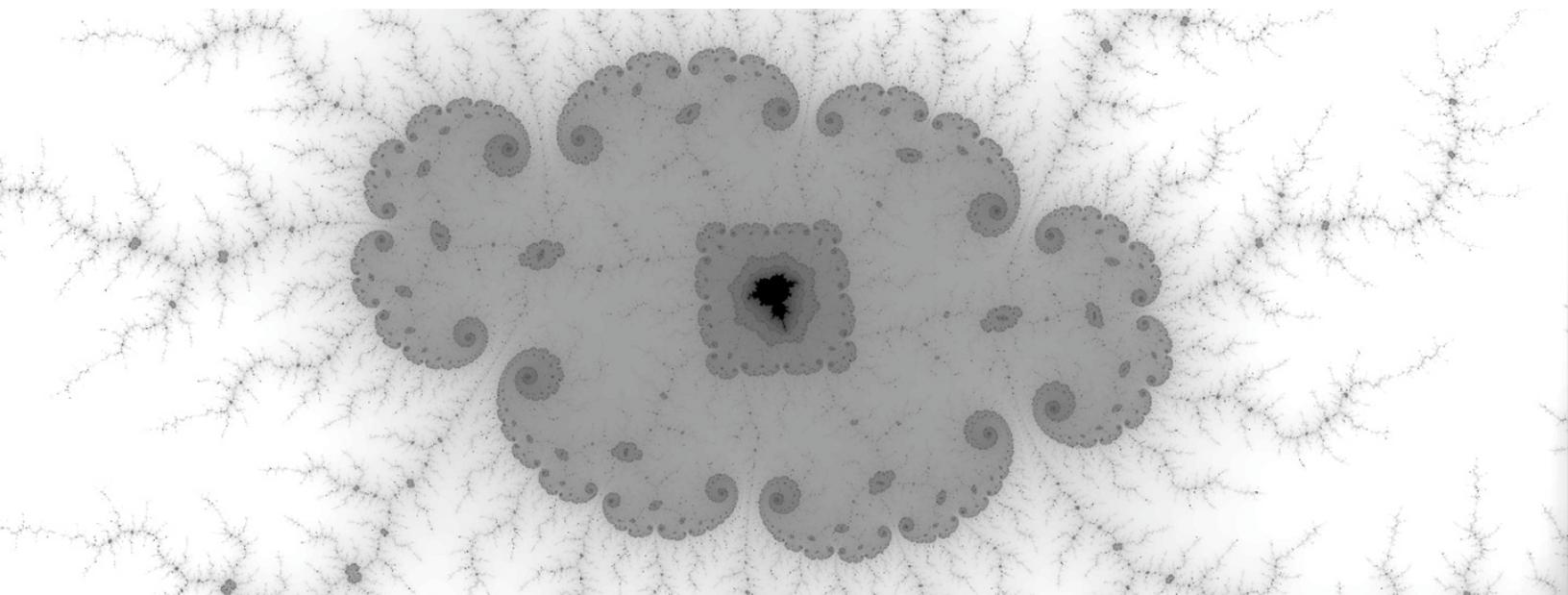
pela outra extremidade.Se você usar addEventListener () com umMessageport, não se esqueça de ligar para Start () ou você pode nunca ver ummensagem entregue.Todas as chamadas PostMessage () que vimos até agora fizeram um únicoargumento da mensagem.Mas o método também aceita um segundo opcionalargumento.Este segundo argumento é uma variedade de itens que devem sertransferido para o outro extremo do canal em vez de enviar uma cópiaatravés do canal.Os valores que podem ser transferidos em vez de copiados sãoMessageports e ArrayBuffers.(Alguns navegadores também implementam outrosTipos transferíveis, como ImageBitmap e OffScreencanvas.Essesnão são universalmente apoiados, no entanto, e não são cobertos nesteLivro.) Se o primeiro argumento a PostMessage () incluir umMessageport (aninhado em qualquer lugar dentro do objeto de mensagem), então queMessageport também deve aparecer no segundo argumento.Se você fizer isso,então o Messageport estará disponível para o outro extremo docanal e se tornará imediatamente não funcional do seu lado.Suponha que você criou um trabalhador e queira ter dois canais paraComunicação com ele: um canal para troca de dados comum eUm canal para mensagens de alta prioridade.No tópico principal, você podeCrie um Messagechannel e ligue para PostMessage () no trabalhadorPara passar uma das porteiras de mensagem:Let Worker = New Worker ("Worker.js");Seja UrgentChannel = new Messagechannel ();Seja urgentport = urgentChannel.port1;trabalhador.PostMessage ({Command: "Seturgentport", Valor:urgentchannel.port2},[urgentChannel.port2]);// agora podemos receber mensagens urgentes do trabalhador comoesse

urgentport.addeventListener ("mensagem", handleUrgentMessage); urgentport.start ()// Comece a receber mensagens// e envie mensagens urgentes como esta urgentport.postMessage ("teste"); Messagechannels também são úteis se você criar dois trabalhadores e quiser permitir que eles se comuniquem diretamente entre si, em vez de exigindo o código no encadeamento principal para transmitir mensagens entre eles. O outro uso do segundo argumento para pós -magus () é para transferir matrizes entre trabalhadores sem precisar copiá -los. Este é um importante aprimoramento de desempenho para grandes matrizes como aqueles usados ??para manter dados de imagem. Quando um matriz é transferido em um Messageport, o ArrayBuffer se torna inutilizável no originalThread para que não haja possibilidade de acesso simultâneo ao seu conteúdo. Se o primeiro argumento a PostMessage () incluir um ArrayBuffer, ouQualquer valor (como uma matriz digitada) que tenha um matriz, então que o buffer pode aparecer como um elemento de matriz no segundo argumento postMessage (). Se aparecer, então será transferido sem copiar. Caso contrário, então o ArrayBuffer será copiado em vez de transferido. Exemplo 15-14 demonstrará o uso deste Técnica de transferência com ArrayBuffers. 15.13.6 Mensagens de origem cruzada com Postmessage () Existe outro caso de uso para o método PostMessage () no ClienteJavaScript lateral. Envolve janelas em vez de trabalhadores, mas há semelhanças suficientes entre os dois casos em que descreveremos o Método PostMessage () do objeto da janela aqui.

Quando um documento contém um elemento <frame>, esse elemento age como uma janela incorporada, mas independente. O objeto de elemento que representa o <frame> tem uma propriedade contentWindow que é o objeto de janela para o documento incorporado. E para scripts em execução dentro desse aninhado Iframe, a propriedade da janela contendo objeto de janela. Quando dois Windows exibem documentos com a mesma origem, os scripts em cada uma dessas janelas têm acesso para o conteúdo da outra janela. Mas quando os documentos têm origens diferentes, a política do mesmo origem do navegador impede o JavaScript de uma janela de acessar o conteúdo de outra janela. Para os trabalhadores, PostMessage () fornece uma maneira segura para dois tópicos independentes para se comunicar sem compartilhar memória. Para Windows, PostMessage () fornece uma maneira controlada para dois origens independentes para trocar mensagens com segurança. Mesmo que a mesma política de origem impede que seu script veja o conteúdo de outro janela, você ainda pode ligar para PostMessage () naquela janela e fazer isso com que um evento de "mensagem" seja acionado nessa janela, onde pode ser visto pelos manipuladores de eventos nos scripts dessa janela. O método PostMessage () de uma janela é um pouco diferente do método PostMessage () de um trabalhador, no entanto. O primeiro argumento ainda é uma mensagem arbitrária que será copiada pelo algoritmo de clone estruturado. Mas a lista de argumentos opcionais do segundo argumento são transferidos em vez de copiados se tornam um terceiro argumento opcional. O método PostMessage () de uma janela leva uma string como seu segundo argumento exigido. Este segundo argumento deve ser um origem (um protocolo, nome de host e porta opcional) que especifica quem você

Espera estar recebendo a mensagem.Se você passar pela string "Https://good.example.com" como o segundo argumento, mas a janelavocê está postando a mensagem para realmente contém conteúdo de "Https://malware.example.com", então a mensagem que você postou não será entregue.Se você estiver disposto a enviar sua mensagem para o conteúdo deQualquer origem, você pode passar o curinga ?*? como o segundo argumento.Código JavaScript em execução dentro de uma janela ou <frame> pode receberMensagens postadas nessa janela ou quadro definindo o OnMessagepropriedade dessa janela ou ligando para addEventListener () paraEventos de "mensagem".Como com os trabalhadores, quando você recebe uma "mensagem"Evento para uma janela, a propriedade de dados do objeto de evento é omensagem que foi enviada.Além disso, no entanto, eventos de "mensagem"Entregue no Windows também define as propriedades de origem e origem.A propriedade de origem especifica o objeto da janela que enviou o evento,e você pode usar o evento.source.postMessage () para enviar uma resposta.A propriedade Origin especifica a origem do conteúdo na fontejanela.Isso não é algo que o remetente da mensagem pode forjar,E quando você recebe um evento de "mensagem", você normalmente desejaVerifique se é de uma origem que você espera.15.14 Exemplo: o conjunto MandelbrotEste capítulo sobre JavaScript do lado do cliente culmina com um longo exemploque demonstra o uso de trabalhadores e mensagens para paralelizartarefas computacionalmente intensivas.Mas está escrito para ser um envolvente,Aplicativo da Web do mundo real e também demonstra um número doOutras APIs demonstradas neste capítulo, incluindo história

gerenciamento; uso da classe IMAGEDATA com um <Canvas>; e o uso de teclado, ponteiro e redimensionamento de eventos. Também demonstra recursos importantes de JavaScript, incluindo geradores e o uso sofisticado de promessas. O exemplo é um programa para exibir e explorar o Mandelbrot Set, um fractal complexo que inclui belas imagens como a mostrada na Figura 15-16. Figura 15-16. Uma parte do conjunto de Mandelbrot



O conjunto de Mandelbrot é definido como o conjunto de pontos no complexo plano, que, quando transmitido por um processo repetido de multiplicação e adição, produzem um valor cuja magnitude permanece limitada. Os contornos do conjunto são surpreendentemente complexos, e computação quais pontos são membros do conjunto e quais não são computacionalmente intensivo: produzir uma imagem de 500×500 pixels em sua imagem. E para verificar se o valor associado a cada pixel permanece delimitado, você pode ter que repetir o processo de multiplicação complexa 1.000 vezes ou mais. (Mais as iterações fornecem limites mais bem definidos para o conjunto; menos iterações produzem limites mais confusos.) Com até 250 milhões de etapas de aritmética complexa necessária para produzir uma imagem de alta qualidade do Conjunto de Mandelbrot, você pode entender por que o uso de trabalhadores é uma valiosa técnica. O exemplo 15-14 mostra o código do trabalhador que usaremos. Este arquivo é relativamente compacto: é apenas o músculo computacional bruto para o programa maior. Vale a pena notar duas coisas sobre isso: o trabalhador cria um objeto de `imageData` para representar uma grade retangular de pixels para os quais está computando o Conjunto de Mandelbrot. Mas em vez de armazenar valores reais de pixel na imagem, ele usa uma matriz de tinta personalizada para tratar cada pixel como um número inteiro de 32 bits. Ele armazena o número de iterações necessárias para cada pixel nesta matriz. Se a magnitude do número complexo calculado para cada pixel se torna maior que quatro, então é matematicamente garantido crescer. Sem limites a partir de então, e dizemos que "escapou". Então o valor que este trabalhador retorna para cada pixel é o número de iterações antes do valor escapar. Dizemos ao trabalhador o Número máximo de iterações deve tentar para cada valor, e pixels que atingem esse número máximo são considerados para

estar no conjunto.O trabalhador transfere o matriz associado aoImaginoutata de volta ao fio principal para que a memória associada com ele não precisa ser copiado.Exemplo 15-14.Código do trabalhador para regiões de computação do Mandelbrotdefinir// Este é um trabalhador simples que recebe uma mensagem de seu tópico pai,// executa o cálculo descrito por essa mensagem e depoisPublica o// resultado desse cálculo de volta ao thread pai.onMessage = function (mensagem) {// Primeiro, descompactemos a mensagem que recebemos:// - Tile é um objeto com propriedades de largura e altura.Especifica o// tamanho do retângulo de pixels para os quais estaremos computação// Mandelbrot Set Association.// - (x0, y0) é o ponto no plano complexo que corresponde ao// pixel superior no ladrilho.// - Perpixel é o tamanho do pixel no real e dimensões imaginárias.// - Maxiterations especifica o número máximo de iterações nós iremos// executa antes de decidir que um pixel está no conjunto.const {tile, x0, y0, perpixel, maxiterations} = message.data;const {largura, altura} = ladrilho;// Em seguida, criamos um objeto imagedata para representar oArray retangular// de pixels, obtenha seu ArrayBuffer interno e crie umVisualização de matriz digitada// desse buffer para que possamos tratar cada pixel como um único número inteiro em vez de// quatro bytes individuais.Guiaremos o número de iterações para cada um// Pixel nesta matriz de iterações.(As iterações serão transformadas em// cores de pixel reais no thread pai.)const imagedata = new IMAGEDATA (largura, altura);

```
const iterations = new Uint32Array(imagedata.data.buffer); // Agora começamos o cálculo. Existem três aninhados
paraloops aqui. // os dois loop externo sobre as linhas e colunas de pixels, e o interior // Loop itera cada pixel para
ver se "escapa" ou não. Os vários // As variáveis ??de loop são as seguintes: // - Linha e coluna são inteiros
representando o pixel coordenada. // - x e y representam o ponto complexo para cada pixel: x+ yi. // - índice é o
índice na matriz de iterações para opixel atual. // - n rastreia o número de iterações para cada pixel. // - Max e Min
rastreiam o maior e o menor número de iterações. // Vimos até agora para qualquer pixel no retângulo. Seja index =
0, max = 0, min = maxiterations; para (deixe linha = 0, y = y0; linha < altura; linha ++, y += perpixel) {para (deixe
coluna = 0, x = x0; coluna < largura; coluna ++, x += perpixel) { // Para cada pixel, começamos com o número
complexo c = x + yi. // então calculamos repetidamente o número complexo z (n+1) baseado em // Esta fórmula
recursiva: // z (0) = c // z (n + 1) = z (n)^2 + c // se | z (n) | (A magnitude de z (n)) é > 2, então // pixel não faz parte do
conjunto e paramos depois de n iterações. deixe n; // o número de iterações então distante. Seja r = x, i = y; // Comece
com z (0) definido como C para (n = 0; n < maxiterations; n++) {Seja rr = r * r, ii = i * i; // quadrado as duas partes de z
(n). if (rr + ii) > 4 { // if | z (n) | ^2 é > 4 então quebrar; // Nós escapamos e pode parar de iterando.
```

```
 }i = 2*r*i + y;// calcular imaginárioparte de z (n+1).r = rr - ii + x;// e a parte real dez (n+1).}iterações [index ++] = n;// Lembrar #iterações para cada pixel.if (n> max) max = n;// rastrear o máximonúmero que vimos.if (n <min) min = n;// e o mínimo comobem.}// Quando o cálculo estiver concluído, envie os resultados de volta para o pai// fio.O objeto de imagem será copiado, mas oArraybuffer gigante// ele contém será transferido para uma boa performanceimpulsionar.PostMessage ({Tile, IMAGEDATA, MIN, MAX},[imagedata.data.buffer]);};O aplicativo Mandelbrot Set Viewer que usa esse código do trabalhador émostrado no Exemplo 15-15.Agora que você quase chegou ao fim deEste capítulo, este longo exemplo é uma experiência de CapstoneIsso reúne um número importante e do lado do clienteRecursos de JavaScript e APIs.O código é completamente comentado, eEncorajo você a lê -lo com cuidado.Exemplo 15-15.Um aplicativo da web para exibir e explorar oMandelbrot Conjunto/** Esta classe representa um sub -rangão de uma tela ou imagem.Usamos ladrilhos para* Divida uma tela em regiões que podem ser processadasindependentelemente pelos trabalhadores.*/
```

Classe Tile {construtor (x, y, largura, altura) {this.x = x;// as propriedades de umObjeto de ladrilhothis.y = y;// representa oposição e tamanhodois.width = width;// do ladrilho dentro de ummaiorthis.Height = altura;// retângulo.}// Este método estático é um gerador que divide umretângulo do// largura e altura especificadas no número especificado delinhas e// colunas e produz NumRows*Numcols Tile Objects para cobriro retângulo.telhas estáticas *(largura, altura, numrows, numcols) {Deixe ColumnWidth = Math.ceil (largura / numcols);Let RowHeight = Math.ceil (Hight / NumRows);for (let linha = 0; linha <numrows; linha++) {Deixe TileHeight = (linha <numrows-1)?ROWHEILE // Alturada maioria das linhas: altura - altura da linha * (numrows -1);// alturada última linhapara (let col = 0; col <numcols; col++) {Deixe TileWidth = (col <numcols-1)?ColumnWidth // Largurada maioria das colunas: largura - largura de coluna * (numcols -1);// eÚltima colunaRendimento novo ladrilho (Col * largura de coluna, linha *ROWHEILE,larwidth, telhado);}}}}

```
/** Esta classe representa um pool de trabalhadores, todos executando o mesmo código. O* Código do trabalhador que você especificar deve responder a cada mensagem recebida por* executar algum tipo de computação e depois postar uma mensagem única com* o resultado desse cálculo.** Dado um trabalhador e uma mensagem que representa o trabalho para ser realizado, simplesmente* Call addwork (), com a mensagem como argumento. Se houver um trabalhador* inativo atualmente, a mensagem será postada para aquele trabalhador* imediatamente. Se não houver objetos de trabalhador ocioso, a mensagem será* na fila e será postada para um trabalhador quando alguém se tornar disponível.** addwork () retorna uma promessa, que resolverá com a mensagem recebida* do trabalho, ou rejeitará se o trabalhador jogar um erro não atendido.*/  
class Workerpool {  
    constructor (NumWorkers, Workersource) {  
        this.idleworkers = [];// trabalhadores que não são ativamente  
        trabalhando  
        this.workQueue = [];// trabalho não atualmente sendo processado  
        this.WorkerMap = new Map ()//  
        mapeia os trabalhadores para resolver e rejeitar funções// Crie o número especificado de trabalhadores, adicione mensagens e erros// Os manipuladores salvam -os na matriz inativa.  
        para (vamos i = 0; i < NumWorkers; i ++)  
        {  
            Deixe o trabalhador = novo trabalhador (Workersource);  
            trabalhador.onmessage = mensagem =>  
            {this._workerDone (trabalhador, null, message.data);};  
            trabalhador.onerror = erro => {this._workerDone (trabalhador, erro, nulo);};  
        }  
    }  
}
```

```
this.idleworkers [i] = trabalhador;}// Este método interno é chamado quando um trabalhador terminar trabalhando
também// enviando uma mensagem ou lançando um erro._WorkerDone (trabalhador, erro, resposta) {// Procure as
funções resolve () e rejeitar () para este trabalhador// e depois remova a entrada do trabalhador do mapa.deixe
[resolver, rejeitora] = this.workerMap.get (trabalhador);this.WorkerMap.Delete (trabalhador);// Se não houver
trabalho na fila, coloque esse trabalhador de volta// A lista de trabalhadores ociosos.Caso contrário, tire o trabalho
de fila// e envie para este trabalhador.if (this.workQueue.length === 0) {this.idleworkers.push (trabalhador);} outro
{Deixe [trabalhar, resolver, rejeitar] =this.workQueue.shift ();this.workermap.set (trabalhador, [resolvedor,
rejeitora]);trabalhador.PostMessage (trabalho);}// Finalmente, resolva ou rejeite a promessa associada com o
trabalhador.erro === null?Resolver (resposta): rejeitora (erro);}// Este método adiciona trabalho ao pool de
trabalhadores e retorna umPrometa isso// será resolvido com a resposta de um trabalhador quando o trabalho
forfeito.O trabalho// é um valor a ser transmitido a um trabalhador com pós -Message ().Se houver um//
Trabalhador ocioso, a mensagem de trabalho será enviada imediatamente.Caso contrário, isso// será na fila até
que um trabalhador esteja disponível.addwork (trabalho) {retornar nova promessa ((resolver, rejeitar) => {
```

```
if (this.idleworkers.length > 0) {Let Worker = this.idleworkers.pop ();this.workermap.set (trabalhador, [resolver, rejeitar]);trabalhador.PostMessage (trabalho);} outro {this.workQueue.push ([{trabalho, resolver, rejeitar}]);}}};}/**  
Esta classe detém as informações estaduais necessárias para renderizar umMandelbrot Conjunto.* As  
propriedades CX e Cy dão o ponto no plano complexoesse é o* centro da imagem.A propriedade Perpixel  
especifica comomuito o real e* partes imaginárias desse número complexo mudam para cadapixel da imagem.* A  
propriedade Maxiterations especifica o quanto trabalhamos paraCalcule o conjunto.* Números maiores requerem  
mais computação, mas produzem mais nítidosimagens.* Observe que o tamanho da tela não faz parte do  
estado.Dado CX, CY, e* Perpixel, simplesmente renderizamos qualquer parte do Mandelbroto conjunto se encaixa*  
A tela em seu tamanho atual.** Objetos deste tipo são usados ??com histórico.pushstate () esão usados ??para  
ler* O estado desejado de um URL marcado ou compartilhado.*/classe pagestate {// Este método de fábrica  
retorna um estado inicial para exibiro conjunto inteiro.estático initialState () {Seja s = new Pagestate ();s.cx =  
-0,5;s.cy = 0;s.perpixel = 3/window.innerHeight;s.maxiterations = 500;retorno s;
```

```
// Este método de fábrica obtém estado de um URL ou retornanulo se// Um ??estado válido não pôde ser lido no
URL.estático deurl (url) {Seja s = new Pagestate ();Deixe u = novo URL (URL); // inicialize o estado doParams de
pesquisa da URL.s.cx = parseFloat (u.searchparams.get ("cx"));s.cy = parseFloat (u.searchparams.get
("cy"));s.perpixel = parseFloat (u.searchparams.get ("pp"));s.maxiterations = parseInt (u.searchparams.get ("it"));//
Se obtivemos valores válidos, retorne o objeto Pagestate,caso contrário, nulo.Return (isnan (s.cx) || isnan (s.cy) ||
isnan (s.perpixel)||isnan (s.maxiterations))?nulo: s;}// Este método de instância codifica o estado atual noprocurar//
parâmetros do local atual do navegador.Tourl () {deixe u = novo url (window.location);USearchParams.Set ("CX",
this.cx);USearchParams.set ("cy", this.cy);USearchParams.set ("pp", this.perpixel);USearchParams.set ("it",
this.maxiterations);retornar u.href;}// Essas constantes controlam o paralelismo do MandelbrotDefina a
computação.// Pode ser necessário ajustá -los para obter o melhor desempenho emseu computador.const linhas =
3, cols = 4, NumWorkers =Navigator.hardwareCurrency ||2;// Esta é a classe principal do nosso programa
Mandelbrot Set.Simplesmente
```

invocar o// Função construtora com o elemento <latavas> para renderizarem.O programa// pressupõe que este elemento <Canvas> seja estilizado para que sejasempre tão grande// como a janela do navegador.classe MandelbrotCanvas {construtor (tela) {// Armazene a tela, obtenha seu objeto de contexto elnicialize um Workerpoolthis.Canvas = Canvas;this.Context = Canvas.getContext ("2D");this.workerpool = new Workerpool (NumWorkers,"Mandelbrotworker.js");// define algumas propriedades que usaremos mais tardethis.tiles = null;// sub-regiões da telathis.pendingRender = null;// não estamos atualmente renderizaçãothis.wantsRerender = false;// Nenhuma renderização está atualmente solicitadothis.resizetimer = null;// nos impede de redimensionar com muita frequênciathis.colortable = null;// para converter dados brutospara valores de pixel// Configure nossos manipuladores de eventosthis.Canvas.AddeventListener ("Pointerdown", e =>this.HandlePointer (e));window.addeventListener ("keydown", e =>this.HandleKey (e));window.addeventListener ("redimensionar", e =>this.HandleResize (e));window.addeventListener ("popstate", e =>this.setState (e.state, false));// inicialize nosso estado a partir do URL ou comece com o estado inicial.this.state =Pagestate.Fromurl (Window.Location)||Pagestate.initState ()// Salve este estado com o mecanismo de história.

```
history.Replacestate (this.state, "",this.state.Tourl());// Defina o tamanho da tela e obtenha uma variedade de  
ladrilhos quecubra.this.SetSize ()// e renderize o Mandelbrot definido na tela.this.render ()// Defina o tamanho da  
tela e inicialize uma matriz de ladrilhosobjetos.Esse// o método é chamado do construtor e também  
peloHandleResize ()// Método quando a janela do navegador é redimensionada.setSize () {this.width =  
this.canvas.width = window.innerWidth;this.Height = this.Canvas.Height = Window.innerHeight;this.tiles = [...  
tile.tiles (this.width, this.Height,Linhas, cols)];}// Esta função faz uma alteração no Pagestate e depoisrenderiza o//  
Mandelbrot conjunto usando esse novo estado e também salva onovo estado com// history.pushstate ().Se o  
primeiro argumento é uma funçãoessa função// será chamado com o objeto de estado como seu argumento edeve  
fazer// muda para o estado.Se o primeiro argumento é umobjeto, então nós simplesmente// copie as propriedades  
desse objeto no estadoobjeto.Se opcional// O segundo argumento é falso, então o novo estado não  
serásalvo.(Nós// Faça isso ao chamar SetState em resposta a um PopStateevento.)SetState (f, salvar = true) {// Se  
o argumento for uma função, chame para atualizar oestado// caso contrário, copie suas propriedades na  
correnteestado.if (typeof f === "function") {
```

f (this.state);} outro {para (deixe a propriedade em f) {this.state [propriedade] = f [propriedade];}}// Em ambos os casos, comece a renderizar o novo estado o mais rápido possível.this.render ()// Normalmente, salvamos o novo estado.Exceto quando estamos chamado com// um segundo argumento de false que fazemos quando obtemos umEvento PopState.se (salvar) {history.pushstate (this.state, "",this.state.Tourl ())};} Este método desenha de forma assíncrona a parte do Mandelbrot Conjunto// especificado pelo objeto Pagestate na tela.Isso é chamado por// O construtor, por setState () quando o estado muda,e pelo// Redimensione o manipulador de eventos quando o tamanho da tela muda.render () {}// às vezes o usuário pode usar o teclado ou mouse para solicitar renderizações// mais rapidamente do que podemos executá -los.Nós não queremos para enviar tudo// as renderizações para o pool de trabalhadores.Em vez disso, se somos Renderização, nós vamos// apenas anote que é necessária uma nova renderização e Quando a corrente// Render completa, renderizaremos o estado atual,possivelmente pulando// vários estados intermediários.se (this.pendingrender) {}// se já estivermos renderização,this.wantsRerender = true;// anotaReerender mais tarde

retornar;// e não faça qualquer coisa mais agora.}// Obtenha nossas variáveis ??de estado e calcule o complexoNúmero para o// canto superior esquerdo da tela.Seja {cx, cy, perpixel, maxiterations} = this.state;Seja x0 = cx - perpixel * this.width/2;Seja y0 = cy - perpixel * this.Height/2;// Para cada uma de nossas linhas*cols telhas, ligue para Addwork () com uma mensagem// para o código em mandelbrotworker.js.Colete opromessa resultante// objetos em uma matriz.deixe promessas = this.tiles.map (ladrilho =>this.workerpool.addwork ({telha: ladrilho,x0: x0 + tile.x * perpixel,y0: y0 + tile.y * perpixel,perpixel: perpixel,Maxiterations: Maxiterations}));// use Promise.all () para obter uma variedade de respostas dea matriz de// promessas.Cada resposta é o cálculo para um de nossos ladrilhos// Lembre -se de Mandelbrotworker.js que cada resposta inclui o// objeto de ladrilho, um objeto imagedata que inclui iteração conta// em vez de valores de pixels, e o mínimo e o máximo iterações// para esse ladrilho.this.pendingRender =Promessa.all (promessas). Então (respostas => {// Primeiro, encontre as iterações máximas e min sobre todos os ladrilhos// precisamos desses números para que possamos atribuir cores aos pixels.Seja min = maxiterations, max = 0;

Erro ao traduzir esta página.

```
}} outro {// no caso normal em que Min e Max estãoDiferente, use a// Escala logarítmica para atribuir cada um
possívelIteração conta um// opacidade entre 0 e 255 e depois use omudança para a esquerda// Operador para
transformá -lo em um valor de pixel.Seja maxlog = math.log (1+max-min);para (vamos i = min; i <= max; i++)
{this.Colortable [i] =(Math.ceil (Math.log (1+i-min)/maxlog *255) << 24);};// agora traduz os números de iteração em
cadaRespostas// fotografata para as cores do colortável.para (deixe r de respostas) {Deixe iterações =
novoUint32Array (R.Imagedata.data.buffer);para (vamos i = 0; i <iterations.length; i++) {iterações [i]
=this.colortable [iterações [i]];}};// Finalmente, renderize todos os objetos de imagem emdeles// ladrilhos
correspondentes da tela usandoputImagedata ().// (primeiro, porém, remova qualquer transformação de CSS
notela que pode// foram definidos pelo manipulador de eventos Pointerdown.)this.Canvas.style.Transform = "";para
(deixe r de respostas) {this.Context.putImagedata (r.imagedata,R.Tile.X, R.Tile.Y);}}
```

.CATCH ((Razão) => {// Se alguma coisa deu errado em qualquer uma de nossas promessas,Vamos registrar// um erro aqui.Isso não deve acontecer, mas isso vai ajudar com// Depuração se isso acontecer.console.error ("Promise rejeitada em render ():",razão);}).Finalmente (() => {// Quando terminamos de renderizar, limpe oBandeiras pendentesthis.pendingRender = null;// e se os pedidos de renderização chegaram enquanto estávamosOcupado, reproduzido agora.if (this.wantsRerender) {this.wantsRerender = false;this.render ();}});// Se o usuário redimensionar a janela, essa função será chamado repetidamente// redimensionar uma tela e renderizar o conjunto de mandelbrot é um caro// operação que não podemos fazer várias vezes por segundo, entãoUsamos um cronômetro// para adiar o manuseio do redimensionamento até que 200 ms tenham decorridoDesde o último// O evento redimensionou foi recebido.HandleResize (evento) {// Se já estivéssemos adiando um redimensionamento, limpe -o.if (this.resizetimer) clearTimeout (this.resizetimer); // e adie este redimensionamento.this.resizetimer = setTimeout (() => {this.resizetimer = null;// Observe que o redimensionamento tem sido tratadothis.setSize ();// redimensione a tela e azelejosthis.render ()// rerender no novo tamanho}, 200);

```
// Se o usuário pressionar uma tecla, este manipulador de eventos será chamado.// chamamos o setState () em  
resposta a várias chaves e setState () renderiza// o novo estado, atualiza o URL e salva o estado em História do  
navegador.HandleKey (evento) {switch (event.key) {caso "escape": // digite fuga para voltar ao estado  
initialthis.setState (pagestate.initialState ());quebrar;caso "+": // tipo + para aumentar o número  
de iterações;this.setState (s => {s.maxiterations =Math.Round (S.Maxiterations*1.5);});quebrar;caso " -": // tipo - para  
diminuir o número de iterações;this.setState (s => {s.maxiterations =Math.Round (S.Maxiterations/1.5);if  
(s.maxiterations <1) s.maxiterations = 1;});quebrar;caso "O": // tipo O para aumentar o zoom;this.setState (s =>  
s.perpixel *= 2);quebrar;case "Arrowup": // seta para cima para rolar para cima;this.setState (s => s.cy -=  
this.Height/10 *s.perpixel);quebrar;case "Arrowdown": // seta para baixo para rolar para baixo;this.setState (s =>  
s.cy += this.Height/10 *s.perpixel);quebrar;case "Arrowleft": // seta para a esquerda para rolar para a  
esquerda;this.setState (s => s.cx -= this.width/10 *
```

```
s.perpixel);quebrar;case "Arrowright": // seta direita para rolar para a direitathis.setState (s => s.cx += this.width/10 *s.perpixel);quebrar;}// Este método é chamado quando obtemos um evento de ponteiro ema tela// O evento Pointerdown pode ser o início de um zoomgesto (um clique ou// toque) ou um gesto de pan (um arrasto).Este manipulador registramanipuladores para// Os eventos de ponteiro e ponteiro para responderpara o resto// do gesto.(Esses dois manipuladores extras são removidosQuando o gesto// termina com um ponteiro.)handlepointer (evento) {// as coordenadas do pixel e o tempo da inicialponteiro para baixo// porque a tela é tão grande quanto a janela, estesCoordenadas de eventos// também são coordenadas de tela.const x0 = event.clientX, y0 = event.clientY, t0 =Date.now();// Este é o manipulador para os eventos de movimentação.Const PointermoveHandler = Evento => {// Quanto nos mudamos e quanto tempo tempassou?Deixe dx = event.clientX-x0, dy = event.clientY-y0,dt = date.now ()-t0;// se o ponteiro se moveu o suficiente ou o suficientejá passou isso// Este não é um clique regular e, em seguida, use CSS para pana tela// (nós o renderia de verdade quando conseguirmos oEvento de Pointerup.)if (dx> 10 || dy> 10 || dt> 500) {this.Canvas.style.Transform =
```

```
`traduzir ($ {dx} px, $ {dy} px`);}// Este é o manipulador para eventos de ponteiraconst pointerupHandler = evento
=> {// Quando o ponteiro sobe, o gesto acaba, então remova// Os manipuladores de movimento e subir até o
próximo gesto.this.Canvas.RemoveEventListener
("Pointermove",PointermoveHandler);this.Canvas.RemoveEventListener ("Pointerup",PointerupHandler);// quanto o
ponteiro se moveu e quanto tempo passou?const dx = event.clientX-x0, dy = event.clientY-y0,dt = date.now ()-t0;//
Depacote o objeto de estado em indivíduoconstantes.const {cx, cy, perpixel} = this.state;// se o ponteiro se moveu
o suficiente ou se o suficienteO tempo passou, então// Este foi um gesto de pan e precisamos mudar estando para
mudar// O ponto central.Caso contrário, o usuário clicou ou bateu em um// apontar e precisamos centralizar e
aumentar o zoomapontar.if (dx> 10 || dy> 10 || dt> 500) {// O usuário bateu a imagem por (dx, dy)pixels// converte
esses valores em compensações no plano complexo.this.setState ({cx: cx - dx*perpixel, cy: cy - dy*perpixel});} outro
{// O usuário clicou.Calcule quantos pixelsO centro se move.Seja cdx = x0 - this.width/2;Seja cdy = y0 -
this.height/2;// Use CSS para ampliar rápida e temporariamente
```

```
this.Canvas.style.Transform =`tradução ($ {-CDX*2} px, $ {-Cdy*2} px)scala (2)`;// Defina as coordenadas complexas do novoponto central e// zoom por um fator de 2.this.setState (s => {s.cx += cdx * s.perpixel;s.cy += cdy * s.perpixel;s.perpixel /= 2;});}// Quando o usuário inicia um gesto, registramos manipuladorespara o// Eventos de Pointermove e Pointerup a seguir.this.Canvas.addeventListener ("Pointermove",PointermoveHandler);this.canvas.addeventListener ("ponterupp",PointerupHandler);}// Finalmente, eis como configuramos a tela.Observe que issoArquivo javascript// é auto-suficiente.O arquivo html precisa incluir isso apena sum <script>.Let Canvas = Document.CreateElement ("Canvas");// Crie aelemento de teladocument.body.append (Canvas);// insira -ono corpodocument.body.style = "margem: 0";// Sem margem para o <body>canvas.style.width = "100%";// faz telaTão largo quanto o corpo canvas.style.Height = "100%";// e tão altocomo o corpo.New MandelbrotCanvas (Canvas);// e inicieRenderizando nele!
```

15.15 Resumo e sugestões para Leitura adicional

Este capítulo longo cobriu os fundamentos do lado do cliente da Programação JavaScript: Como os scripts e os módulos JavaScript estão incluídos nas páginas da web e como e quando eles são executados. JavaScript assíncrono, do lado do cliente, orientado a eventos é o modelo de programação. O Modelo de Objeto do Documento (DOM) que permite JavaScript código para inspecionar e modificar o conteúdo HTML do documento que ele está incorporado. Esta API DOM é o coração de toda a programação JavaScript do lado do cliente. Como o código JavaScript pode manipular os estilos CSS que são aplicados ao conteúdo dentro do documento. Como o código JavaScript pode obter as coordenadas dos elementos do documento na janela do navegador e dentro do documento em si. Como criar componentes da web? da interface do usuário reutilizados com JavaScript, HTML e CSS usando os elementos personalizados e Shadow DOM APIs. Como exibir e gerar dinamicamente gráficos com SVG e o elemento HTML <canvas>. Como adicionar efeitos sonoros com script (tanto gravados quanto sintetizado) em suas páginas da web. Como o JavaScript pode fazer o navegador carregar novas páginas, vá para trás e para frente no histórico de navegação do usuário, e até adicione novas entradas ao histórico de navegação.

Como os programas JavaScript podem trocar dados com servidores da Web usando os protocolos HTTP e WebSocket. Como os programas JavaScript podem armazenar dados no navegador do usuário. Como os programas JavaScript podem usar threads de trabalhadores para alcançar uma forma segura de simultaneidade. Este tem sido o capítulo mais longo do livro, de longe. Mas não pode chegar perto de cobrir todas as APIs disponíveis para os navegadores da Web. A plataforma da web é ampla e sempre evoluindo, e meu objetivo para isso. O capítulo era introduzir as APIs principais mais importantes. Com o conhecimento que você tem neste livro, você está bem equipado para aprender a usar novas APIs conforme você precisa delas. Mas você não pode aprender sobre um novo API se você não sabe que existe, então as seções curtas a seguir terminam o capítulo com uma lista rápida de recursos da plataforma da web que você pode querer investigar no futuro.

15.15.1 HTML e CSS

A Web é construída sobre três tecnologias principais: HTML, CSS e JavaScript e o conhecimento do JavaScript podem levá-lo apenas até onde um desenvolvedor da Web, a menos que você também desenvolva sua experiência com HTML e CSS. É importante saber como usar o JavaScript para manipular elementos HTML e estilos CSS, mas esse conhecimento é muito mais útil se você também souber quais elementos HTML e quais estilos CSS para usar. Então, antes de começar a explorar mais APIs de JavaScript, eu encorajaria você para investir algum tempo para dominar as outras ferramentas em uma webkit de ferramentas do desenvolvedor. HTML forma e elementos de entrada, por exemplo, têm comportamento sofisticado que é importante para entender, e o Flexbox

E os modos de layout da grade no CSS são incrivelmente poderosos. Dois tópicos que valem a pena prestar atenção nessa área são acessibilidade (incluindo atributos ARIA) e internacionalização (incluindo suporte para instruções de escrita direita para a esquerda). 15.15.2 Desempenho Depois de escrever um aplicativo da web e lançá-lo para o mundo, a busca interminável para fazer isso rápido começa. É difícil otimizar coisas que você não pode medir, no entanto, então vale a pena familiarizar você mesmo com as APIs de desempenho. A propriedade de desempenho do objeto da janela é o ponto de entrada principal para esta API. Inclui o desempenho de fonte de tempo de alta resolução `.now()` e métodos `performance.mark()` e `performance.measure()` parametrizando pontos críticos em seu código e medindo o tempo decorrido entre eles. Chamar esses métodos cria objetos de desempenho que você pode acessar com `performance.getEntries()`. Navegadores adicionam seus próprios objetos de desempenho sempre que o navegador carrega uma nova página ou busca um arquivo sobre a rede e estes automaticamente. Os objetos de desempenho criados incluem detalhes de tempo granular do desempenho da rede do seu aplicativo. O relacionado à classe `performanceObserver` permite que você especifique uma função para ser invocada quando novos objetos de desempenho são criados. 15.15.3 Segurança Este capítulo introduziu a ideia geral de como se defender contra vulnerabilidades de segurança de script de sites cruzados (XSS) em seus sites, mas

Não entramos em muitos detalhes. O tópico da segurança da web é um importante, e você pode querer passar algum tempo aprendendo mais sobre isso. Além do XSS, vale a pena aprender sobre o conteúdo Cabeçalho HTTP da política de segurança e entender como CSP permite que você peça ao navegador da web que restrinja os recursos que concede Código JavaScript. Entendendo os CORs (recurso de origem cruzada Compartilhar) também é importante.

15.15.4 WebAssembly

WebAssembly (ou "WASM") é um bytecode de máquina virtual de baixo nível projetado para integrar bem com intérpretes de JavaScript em navegadores da web. Existem compiladores que permitem compilar C, C++, e programas de ferrugem para WebAssembly Bytecode e para executar aqueles programas em navegadores da web perto da velocidade nativa, sem quebrar a caixa de areia ou modelo de segurança do navegador. WebAssembly pode exportar funções que podem ser chamadas pelos programas JavaScript. Um caso de uso típico para WebAssembly, seria compilar o Zlib padrão da língua C Biblioteca de compressão para que o código JavaScript tenha acesso a alta velocidade Algoritmos de compressão e descompressão. Saiba mais em <https://webassembly.org>.

15.15.5 Mais recursos de documentos e janelas

A janela e os objetos do documento têm vários recursos que não foram cobertos neste capítulo: O objeto de janela define alert(), confirm() e Métodos de promoto() que exibem diálogos modais simples para o usuário. Esses métodos bloqueiam o encadeamento principal. O

Confirm () Método retorna de forma síncrona um valor booleano,e prompt () retorna de maneira síncrona uma sequência de entrada do usuário.Estes não são adequados para uso da produção, mas podem ser úteis paraProjetos e protótipos simples.As propriedades do navegador e da tela da janelao objeto foi mencionado na passagem no início deste capítulo,Mas os objetos de navegador e tela que eles fazem referência têmAlguns recursos que não foram descritos aqui que você pode encontrarútil.O método solicitfullscreen () de qualquer elementoObjeto solicita que esse elemento (a <dife> ou <IVAs>elemento, por exemplo) ser exibido no modo de tela cheia.Oo método de saída do documento retorna paraModo de exibição normal.O método requestanimationframe () da janelaObjeto assume uma função como seu argumento e executará quefunção quando o navegador está se preparando para renderizar o próximoquadro.Quando você está fazendo mudanças visuais (especialmenterepetidos ou animados), envolvendo seu código em uma chamadapara requestanimationframe () pode ajudar a garantir queas mudanças são renderizadas suavemente e de uma maneira que éotimizado pelo navegador.Se o usuário selecionar o texto em seu documento, você poderá obterDetalhes dessa seleção com o método da janelagetSelection () e obtenha o texto selecionado comgetSelection (). ToString ()Em alguns navegadores,Navigator.clipboard é um objeto com uma API assíncronapara ler e definir o conteúdo da área de transferência do sistema paraAtivar interações de cópia e colar com aplicativos forado navegador.Um recurso pouco conhecido dos navegadores da web é que html

elementos com um atributo contentedável = "true"Permita que seu conteúdo seja editado.ODocument.ExecCommand () Método permite o texto ricoRecursos de edição para conteúdo editável.Um mutationObserver permite que o JavaScript monitore as alterações,ou abaixo, um elemento especificado no documento.Crie aMutationObServer com o mutationObServer ()construtor, passando a função de retorno de chamada que deve ser chamadoQuando as alterações são feitas.Em seguida, ligue para o método Observe ()do mutationObserver para especificar quais partes das quaiselemento deve ser monitorado.Um intersectionObserver permite que o JavaScript determine qualOs elementos do documento estão na tela e que estão próximosestar na tela.É particularmente útil para aplicaçõesque desejam carregar dinamicamente o conteúdo sob demanda como o usuárioRolls.15.15.6 EventosO grande número e a diversidade de eventos apoiados pela webA plataforma pode ser assustadora.Este capítulo discutiu uma variedade de eventosTipos, mas aqui estão alguns mais que você pode achar útil:Os navegadores disparam eventos "online" e "offline" na janelaObjeto quando o navegador ganha ou perde uma conexão com a Internet.Os navegadores disparam um evento "VisiblityChange" no documentoobjeto quando um documento se torna visível ou invisível (geralmenteporque um usuário alterou as guias).JavaScript pode verificardocument.VisibilityState para determinar se o seuO documento está atualmente "visível" ou "oculto".Os navegadores suportam uma API complicada para suportar arrastar e soltar

UIs e para apoiar a troca de dados com aplicativos fora do navegador. Esta API envolve vários eventos, incluindo "Dragstart", "Dragover", "Dragend" e "Drop". Esta API é Complicado de usar corretamente, mas útil quando você precisar. É um API importante para saber se você deseja permitir que os usuários Arraste os arquivos do desktop para o seu aplicativo da web. A API de bloqueio de ponteiro permite que JavaScript oculte o mouseponteiro e obtenha eventos de mouse bruto como movimento relativovalores em vez de posições absolutas na tela. Isso é Normalmente útil para jogos. Chamada RequestPointerlock () No elemento que você deseja que todos os eventos do mouse sejam direcionados. Depois você faz isso, eventos "mousemove" entregues a esse elemento terá propriedades de movimento e movimento. A API GamePad adiciona suporte para controladores de jogo. Usar Navigator.getgamepads () para obter gamepad conectado objetos e ouça para eventos "gamepadconnected" no objeto de janela a ser notificado quando um novo controlador é conectado. O objeto gamepad define uma API para consultar o estado atual dos botões no controlador.

15.15.7 Aplicativos da Web progressivos e trabalhadores de serviço

O termo aplicativos progressivos da Web, ou PWAs, é uma palavra da moda que Descreve aplicativos da Web que são criados usando algumas tecnologias importantes. A documentação cuidadosa dessas tecnologias -chave exigiria um livro por conta própria, e eu não os cobri neste capítulo, mas você deveria Esteja ciente de todas essas APIs. Vale a pena notar que moderno poderoso APIs como essas são normalmente projetadas para funcionar apenas em https seguros conexões. Sites que ainda estão usando http:// URLs não serão capaz de tirar proveito disso:

Um serviço de serviço é um tipo de fio de trabalhador com a capacidade de interceptar, inspecionar e responder a solicitações de rede do Aplicativo da Web que "Serviços". Quando um aplicativo da web registra um trabalhador de serviço, esse código do trabalhador se torna persistente no armazenamento local do navegador e quando o usuário visita o site associado novamente, o trabalhador de serviço é reativado. Os trabalhadores de serviço podem armazenar respostas de rede de cache (incluindo arquivos de código JavaScript), o que significa que a WebAplicações que usam os trabalhadores de serviço podem instalar efetivamente eles mesmos no computador do usuário para startup rápido e uso offline. O Livro de receitas de trabalhadores de serviço em <https://serviceworker.rs> é um recurso valioso para aprender sobre trabalhadores de serviço e suas tecnologias relacionadas. A API de cache foi projetada para uso por trabalhadores de serviço (mas também disponível para o código JavaScript regular fora dos trabalhadores). Funciona com os objetos de solicitação e resposta definidos pelo Fetch () API e implementa um cache de solicitação/resposta pares. A API de cache permite que um trabalhador de serviço cache os scripts e outros ativos do aplicativo da web que ele serve e também pode ajudar a ativar o uso offline do aplicativo da web (o que é particularmente importante para dispositivos móveis). Um manifesto da web é um arquivo formatado em JSON que descreve uma webaplicação incluindo um nome, um URL e links para ícones em vários tamanhos. Se o seu aplicativo da web usar um trabalhador de serviço ele inclui uma tag <link rel = "manifest"> que faz referência a um WebManifest File, então navegadores (principalmente navegadores em dispositivos móveis) pode fornecer a opção de adicionar um ícone para o aplicativo da web para sua área de trabalho ou tela inicial. A API de notificações permite que os aplicativos da Web exibam notificações Usando o sistema de notificação do SO nativo no celular e dispositivos de mesa. As notificações podem incluir uma imagem e texto, e seu código pode receber um evento se o usuário clicar no

notificação. Usar esta API é complicada pelo fato de você deve primeiro solicitar a permissão do usuário para exibir notificações. A API PUSH permite aplicativos da Web que tenham um serviço trabalhador (e que tem a permissão do usuário) para se inscrever notificações de um servidor e para exibir essas notificações mesmo quando o aplicativo em si não está em execução. Empurrar notificações são comuns em dispositivos móveis, e o empurrão API aproxima os aplicativos da web para apresentar paridade com aplicativos nativos no celular.

15.15.8 APIs de dispositivo móvel

Existem várias APIs da Web que são principalmente úteis para aplicativos da Web executando em dispositivos móveis. (Infelizmente, várias dessas APIs só trabalham em dispositivos Android e não em dispositivos iOS.) A API de geolocalização permite JavaScript (com o usuário permissionado) para determinar a localização física do usuário. Está bem suportado em desktop e dispositivos móveis, incluindo iOS dispositivos. Usar `navigator.geolocation.getCurrentPosition()` para solicitar a posição atual do usuário e usar `navigator.geolocation.watchPosition()` para registrar um retorno de chamada para ser chamado quando a posição do usuário muda. O método `navigator.vibrate()` causa um celular dispositivo (mas não iOS) para vibrar. Muitas vezes isso só é permitido em resposta a um gesto do usuário, mas chamar esse método permitirá que o aplicativo forneça feedback silencioso de que um gesto foi reconhecido. A API da Screenorientation permite que um aplicativo da Web consulte

a orientação atual de uma tela de dispositivo móvel e também para que o dispositivo se oriente de retrato. Os eventos de "DeviceOrientation" e "DeviceOrientation" no Relatório de objeto de janela Dados de acelerômetro e magnetômetro para o dispositivo, permitindo que você determine como o dispositivo é acelerando e como o usuário está orientando-o no espaço. (Esses eventos funcionam no iOS.) A API do sensor ainda não é amplamente suportada além do Chrome em dispositivos Android, mas permite o acesso a JavaScript ao conjunto de sensores de dispositivos móveis, incluindo acelerômetro, Giroscópio, magnetômetro e sensor de luz ambiente. Esses sensores permitem que o JavaScript determine em qual direção o usuário está enfrentando ou para detectar quando o usuário sacode o telefone, para exemplo.

15.15.9 APIs binárias

Matrizes digitadas, matrizes e a classe DataView (todos cobertos em §11.2) Permitir que o JavaScript trabalhe com dados binários. Conforme descrito anteriormente neste capítulo, a API Fetch () permite que os programas JavaScript carreguem dados binários sobre a rede. Outra fonte de dados binários são arquivos no sistema de arquivos local do usuário. Por razões de segurança, JavaScript não pode ler arquivos locais. Mas se o usuário selecionar um arquivo para fazer upload (usando um <input type = "FILE"> elemento do formulário) ou arrastar e soltar para o seu aplicativo da web, o JavaScript pode acessar o arquivo como um objeto de arquivo. O arquivo é uma subclasse de blob e, como tal, é uma representação opaca de um pedaço de dados. Você pode usar uma classe FileReader para obter assíncrono o conteúdo de um arquivo como um ArrayBuffer ou String. (Em alguns navegadores,

Você pode pular o FileReader e, em vez disso, usar a promessa baseada em promessamétodos de texto () e ArrayBuffer () definidos pela classe BLOB, ou o método stream () para transmitir acesso ao conteúdo do arquivo.) Ao trabalhar com dados binários, especialmente transmitir dados binários, você pode precisar decodificar bytes em texto ou codificar o texto como bytes. As classes TextEncoder e TextDecoder ajudam nessa tarefa.

15.15.10 APIs de mídia

A função Navigator.mediaDevices.getUserMedia () permite que o JavaScript solicite acesso ao microfone do usuário e/ou Câmera de vídeo. Uma solicitação bem-sucedida resulta em um objeto MediaStream. Os fluxos de vídeo podem ser exibidos em uma tag <video> (definindo a propriedade srcObject para o fluxo). Os quadros do vídeo ainda podem ser capturados em uma tela fora de tela <VAS> com a função drawImage () resultando em uma resolução relativamente baixa de fotografia. Fluxos de áudio e vídeo devolvidos por getUserMedia () podem ser gravados e codificados em uma bolha com um objeto mediarecorder. A API WebRTC mais complexa permite a transmissão e recepção de MediaStreams sobre a rede, permitindo que ponto a ponto videoconferência, por exemplo.

15.15.11 Criptografia e APIs relacionadas

A propriedade criptográfica do objeto da janela expõe um método getRandomValues ??() para criptograficamente seguros números pseudorandomos. Outros métodos para criptografia, descriptografia, chave

geração, assinaturas digitais e assim por diante estão disponíveis através de Crypto.subtle. O nome desta propriedade é um aviso para todo mundo que usa esses métodos que usam adequadamente o criptográfico. Os algoritmos são difíceis e que você não deve usar esses métodos, a menos que você realmente saiba o que está fazendo. Além disso, os métodos de Crypto.subtle estão disponível apenas para o código JavaScript em execução dentro de documentos que foram carregados em uma conexão HTTPS segura. A API de gerenciamento de credenciais e a API de autenticação da Web permitem que o JavaScript gere, armazene e recupere a chave pública (e outros tipos de) credenciais e permitem a criação e login de contas sem senhas. A API JavaScript consiste principalmente nas funções Navigator.credentials.create () e Navigator.credentials.get (), mas a infraestrutura substancial é necessário no lado do servidor para fazer esses métodos funcionarem. Essas APIs ainda não são universalmente apoiadas, mas têm o potencial de revolucionar a maneira como efetuamos login nos sites. A API de solicitação de pagamento adiciona suporte ao navegador para fazer cartão de crédito pagamentos na web. Ele permite que os usuários armazenem seus detalhes de pagamento com segurança no navegador para que eles não precisem digitar seu cartão de crédito número cada vez que eles fazem uma compra. Aplicativos da Web que desejam solicitar um pagamento Crie um objeto PaymentRequest e ligue para o show () Método para exibir a solicitação ao usuário. As edições anteriores deste livro tiveram uma extensa seção de referência que cobriu o JavaScript Biblioteca padrão e APIs da web. Foi removido na sétima edição porque o MDN tem tornado -o obsoleto: hoje, é mais rápido procurar algo no MDN do que para virar através de um livro, e meus ex-colegas da MDN fazem um trabalho melhor em manter seus online Documentação atualizada do que este livro jamais poderia.

2Algumas fontes, incluindo a especificação HTML, fazem uma distinção técnica entre Manipuladores e ouvintes, com base na maneira como estão registrados.Neste livro, nós Trate os dois termos como sinônimos.3Se você usou a estrutura do React para criar interfaces de usuário do lado do cliente, isso pode Surpreenda você.O React faz várias alterações pequenas no modelo de evento do lado do cliente e Um deles é que, no React, os nomes de propriedades do manipulador de eventos são escritos no CamelCase: OnClick, OnMouseOver, e assim por diante.Ao trabalhar com a plataforma da web nativamente, No entanto, as propriedades do manipulador de eventos são escritas inteiramente em minúsculas.4A especificação de elemento personalizada permite a subclassificação de <butto> e outros específicos classes de elemento, mas isso não é suportado no safari e uma sintaxe diferente é necessária para Use um elemento personalizado que estenda qualquer coisa que não seja HTMLELEMENT.

CAPÍTULO 16. SERVIDO DO SERVIDOJavaScript com nóO nó é JavaScript com ligações ao sistema operacional subjacente,possibilitando escrever programas JavaScript que leem e escrevemArquivos, execute processos filhos e se comunique pela rede.Esse Torna o nó útil como um:Alternativa moderna aos scripts de concha que não sofrem deA sintaxe arcana de Bash e outras conchas Unix.Linguagem de programação de uso geral para executar confiançaprogramas, não sujeitos às restrições de segurança impostas porNavegadores da Web em código não confiável.Ambiente popular para escrever eficiente e altamenteServidores da Web simultâneos.A característica definidora do nó é o seu evento único baseado em eventosConcorrência ativada por uma API assíncrona por defesa.Se você temprogramado em outras línguas, mas não fiz muito JavaScriptCodificação, ou se você é um programador JavaScript do lado do cliente experienteacostumado a escrever código para navegação na web, usar o nó será um pouco deAjuste, assim como qualquer nova linguagem ou ambiente de programação.Esse o capítulo começa explicando o modelo de programação do nó, com umênfase na simultaneidade, a API do Node para trabalhar com streamingDados e o tipo de buffer do Node para trabalhar com dados binários.EssesAs seções iniciais são seguidas por seções que destacam e demonstram

algumas das APIs de nó mais importantes, incluindo as de trabalhar com arquivos, redes, processos e threads. Um capítulo não é suficiente para documentar todas as APIs do nó, mas minhas esperanças é que este capítulo explique o suficiente dos fundamentos para tornar você produtivo com o nó e confiante de que você pode dominar qualquer nova API que você precisa. Instalação do nó O nó é um software de código aberto. Visite <https://nodejs.org> para baixar e instalar o Node para Windows e Mac OS. No Linux, você poderá instalar o nó com o seu gerenciador de pacotes normal, ou você pode visitar <https://nodejs.org/en/download> para baixar os binários diretamente. Se você trabalha em container software, você pode encontrar imagens oficiais do Node Docker em <https://hub.docker.com>. Além do executável do nó, uma instalação de nós também inclui o NPM, um gerente de pacotes que permite fácil acesso a um vasto ecossistema de ferramentas e bibliotecas JavaScript. Os exemplos neste capítulo usarão apenas os pacotes internos do Node e não exigirão NPM ou bibliotecas externas. Por fim, não ignore a documentação oficial do nó, disponível em <https://nodejs.org/api/> e <https://nodejs.org/docs/guides/>. Eu achei que é bem organizado e bem escrito.

16.1 Programação do Nó básico

Começaremos este capítulo com uma rápida olhada em como os programas de nó são estruturados e como eles interagem com o sistema operacional.

16.1.1 Saída do console

Se você está acostumado a programar JavaScript para navegadores da web, um dos as surpresas sobre o nó é que o `console.log()` não é apenas para depuração, mas é a maneira mais fácil de exibir uma mensagem para o usuário, ou, de maneira mais geral, para enviar a saída para o fluxo STDOUT. Aqui está o programa clássico "Hello World" em Node:

console.log ("Hello World!"); Existem maneiras de escrever em nível mais baixo, mas sem mais sofisticadas maneira oficial do que simplesmente chamar o console.log (). Nos navegadores da web, console.log (), console.warn () e console.error () normalmente exibe pequenos ícones ao lado de sua saída no console do desenvolvedor para indicar a variedade da mensagem de log. O nó não faz isso, mas a saída exibida com console.error () distingue-se da saída exibida com console.log () Porque console.error () grava no fluxo Stderr. Se você é Usando o nó para escrever um programa projetado para ter stdio redirecionado para um arquivo ou um tubo, você pode usar o console.error () para Exibir texto no console onde o usuário o verá, mesmo que o texto impresso com console.log () está oculto.

16.1.2 Argumentos e ambiente da linha de comando

Variáveis

Se você já escreveu programas de estilo Unix projetados para serem invocados de um terminal ou outra interface da linha de comando, você sabe que esses programas normalmente recebem sua entrada principalmente de comando Argumentos de linha e secundariamente das variáveis ?? ambientais. O nó segue essas convenções Unix. Um programa de nós pode ler seu Argumentos da linha de comando do processo de Strings.argv. O primeiro elemento desta matriz é sempre o caminho para o nó executável. O segundo argumento é o caminho para o arquivo de JavaScript Código que esse nó está executando. Quaisquer elementos restantes nesta matriz são

os argumentos separados por espaço que você passou na linha de comando Quando você chamou o nó. Por exemplo, suponha que você salve este programa de nó muito curto no arquivo argv.js: console.log (process.argv); Você pode executar o programa e ver a saída como esta: \$ Node-TRACE-ANGUGADO ARGV.JS --arg1 --arg2 FileName ['/usr/local/bin/nó','/private/tmp/argv.js','--arg1','--arg2','nome do arquivo'] Há algumas coisas a serem observadas aqui: O primeiro e o segundo elementos do processo.argv serão Caminhos de sistema de arquivos totalmente qualificados para o executável do nó e o arquivo de javascript que está sendo executado, mesmo que você não Digite -os dessa maneira. Argumentos da linha de comando que são destinados e interpretados pelo próprio nó executável é consumido pelo nó executável e não aparecem no processo.argv. (O --O argumento da linha de comando de rastreio não é realmente fazendo qualquer coisa útil no exemplo anterior; está apenas lá para demonstrar que não aparece na saída.) Qualquer argumentos (como --arg1 e nome do arquivo) que aparecem Após o nome do arquivo JavaScript, aparecerá em process.argv.

Os programas de nó também podem obter informações do ambiente de estilo Unixvariáveis.Nó os disponibiliza pelo processo.envobjeto.Os nomes de propriedades deste objeto são variáveis ??de ambientenomes e os valores da propriedade (sempre strings) são os valores daquelesvariáveis.Aqui está uma lista parcial de variáveis ??de ambiente no meu sistema:\$ node -p -e 'process.env'{Shell: '/bin/bash',Usuário: 'David',Caminho: '/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin',PWD: '/tmp',Lang: 'en_us.utf-8',Home: '/Usuários/David',}Você pode usar o nó -h ou o nó - -help para descobrir o que o -p e-e argumentos da linha de comando.No entanto, como uma dica, observe que vocêpoderia reescrever a linha acima como nó --eval 'process.env' --imprimir.16.1.3 Ciclo de vida do programaO comando do nó espera um argumento da linha de comando que especificao arquivo do código JavaScript a ser executado.Este arquivo inicial normalmente importaoutros módulos do código JavaScript e também podem definir suas próprias classese funções.Fundamentalmente, no entanto, o Node executa o JavaScriptCódigo no arquivo especificado de cima para baixo.Alguns programas de nós saemquando terminar de executar a última linha de código no arquivo.Muitas vezes,No entanto, um programa de nó continuará sendo executado muito tempo depois do arquivo inicialfoi executado.Como discutiremos nas seções a seguir, nó

Os programas geralmente são assíncronos e baseados em retornos de chamada e eventomanipuladores.Os programas de nó não saem até que sejam feitos de execução do arquivo inicial e até que todos os retornos de chamada tenham sido chamados e não haja mais eventos pendentes.Um programa de servidor baseado em nó que ouveAs conexões de rede recebidas terão teoricamente para sempre porqueSempre estará esperando por mais eventos.Um programa pode se forçar a sair chamando process.Exit ().Os usuários geralmente podem encerrar um programa de nós digitando Ctrl-C naJanela do terminal onde o programa está em execução.Um programa pode ignorarCtrl-C registrando uma função de manipulador de sinal comprocess.on ("sigint", () => {}).Se o código em seu programa lançar uma exceção e nenhuma cláusula de capturaPega, o programa imprimirá um rastreamento e saída de pilha.Por causa deA natureza assíncrona do nó, exceções que ocorrem em retornos de chamada ouOs manipuladores de eventos devem ser tratados localmente ou não tratados, quesignifica que lidar com exceções que ocorrem nas partes assíncronas deSeu programa pode ser um problema difícil.Se você não quer issoexceções para fazer com que seu programa trava completamente, registre um globalFunção de manipulador que será invocada em vez de travar:process.setuncaughtexceptionCaptureCallback (e => {console.error ("Exceção não capturada:", e);});Uma situação semelhante surge se uma promessa criada pelo seu programa forrejeitado e não há invocação .catch () para lidar com isso.A partir do nó13, este não é um erro fatal que faz com que seu programa sai, masImprima uma mensagem de erro detalhada no console.Em alguma versão futura de

Nó, espera -se que as rejeições de promessa não atendidas se tornem fatais. Se você não deseja rejeições não tratadas, imprimir mensagens de erro ou encerrar seu programa, registre uma função de manipulador global: Process.on ("UNHLEDRIECEJAÇÃO", (Razão, Promise) => { // A razão é qualquer valor que teria sido passado para um.CACK () função // promessa é o objeto de promessa que rejeitou}); 16.1.4 Módulos de nós

Capítulo 10 Sistemas de módulos JavaScript documentados, cobrindo os dois Módulos de nós e módulos ES6. Porque o nó foi criado antes, o JavaScript tinha um sistema de módulo, o Node teve que criar seu próprio. No sistema de módulo usa a função requer () para importar valores para um módulo e o objeto de exportação ou a propriedade Module.Exports para exportar valores de um módulo. Estes são uma parte fundamental do Modelo de programação do nó e eles são cobertos em detalhes no §10.2. O nó 13 adiciona suporte para módulos ES6 padrão, bem como requisitos Módulos baseados (que o nó chama de "módulos Commonjs"). Os dois sistemas de módulos não são totalmente compatíveis, então isso é um tanto complicado de fazer. O nó precisa saber - antes de carregar um módulo - seja esse módulo usará requer () e module.exports ou se ele estará usando importação e exportação. Quando o nó carrega um arquivo de Código JavaScript como um módulo Commonjs, ele define automaticamente o requer () função junto com exportações e módulo de identificadores. E não permite as palavras-chave de importação e exportação. No outro lado, quando o nó carrega um arquivo de código como um módulo ES6, ele deve ignorar as declarações de importação e exportação, e não deve definir

Identificadores extras como exigir, módulo e exportações. A maneira mais simples de dizer a nó que tipo de módulo está carregando é codificar essas informações na extensão do arquivo. Se você salvar seu Código JavaScript em um arquivo que termina com .mjs, então o nó sempre carregará o módulo ES6, esperando que ele use importação e exportação, e não fornecerá uma função requer (). E se você salvar o seu Código em um arquivo que termina com .cjs, então o nó sempre tratará como um Módulo CommonJS, fornecendo uma função requer () e irá jogar um SyntaxError se usar declarações de importação ou exportação. Para arquivos que não possuem uma extensão .mjs ou .cjs explícito, o nó procura por um arquivo chamado package.json no mesmo diretório que o arquivo e depois em cada um dos diretórios que contêm. Uma vez que o arquivo package.json mais próximo é encontrado, verifica o nó para uma propriedade do tipo de nível superior no JSON objeto. Se o valor da propriedade do tipo for "módulo", o nó carrega o arquivo como um módulo ES6. Se o valor dessa propriedade for "CommonJS", o nó carrega o arquivo como um módulo CommonJS. Observe que você não precisa ter um arquivo package.json para executar programas de nó: quando não tais arquivos são encontrados (ou quando o arquivo é encontrado, mas não tem um tipo de propriedade), os padrões do nó usam os módulos CommonJS. Esse package.json só se torna necessário se você quiser usar ES6 módulos com o nó e não deseja usar a extensão do arquivo .mjs. Porque há uma enorme quantidade de código de nó existente escrito usando o formato do módulo CommonJS, o nó permite que os módulos ES6 carreguem módulos CommonJS usando a palavra-chave import. O inverso não é verdadeiro, no entanto: um módulo CommonJS não pode usar requer () para carregar

um módulo ES6.16.1.5 O gerenciador de pacotes do nó Quando você instala o nó, você normalmente recebe um programa chamado NPM com o bem. Este é o gerenciador de pacotes do Node e ajuda você a baixar e gerenciar bibliotecas das quais seu programa depende. O NPM mantém o controle dessas dependências (bem como outras informações sobre o seu programa) em um arquivo chamado package.json no diretório raiz do seu projeto. Este arquivo package.json criado pelo npm é onde você adicionaria "Tipo": "Módulo" se você quiser usar os módulos ES6 para o seu projeto. Este capítulo não cobre o NPM com detalhes (mas consulte §17.4 para um pouco mais de profundidade). Estou mencionando isso aqui porque, a menos que você escreva programas que não usam bibliotecas externas, você quase certamente estará usando NPM ou uma ferramenta como esta. Suponha, por exemplo, que você seja desenvolvendo um servidor da web e planeje usar a estrutura expressa (<https://expressjs.com>) para simplificar a tarefa. Para começar, você pode Crie um diretório para o seu projeto e, em seguida, nesse tipo de diretório NPM init. NPM solicitará o nome do seu projeto, número da versão, etc., e então criará um arquivo package.json inicial com base no seu resposta. Agora, para começar a usar o Express, você digita o NPM Install Express. Isso diz ao NPM para baixar a biblioteca expressa junto com todos os seus dependências e instale todos os pacotes em um node_modules local/diretório: \$ npm Install Express

O Aviso do NPM criou um arquivo de bloqueio como package-lock.json. Você deve comprometer este arquivo. NPM avise my-server@1.0.0 sem descrição. NPM avise my-server@1.0.0 Nenhum campo de repositório. + express@4.17.1 Adicionado 50 pacotes de 37 colaboradores e auditado 126 pacotes em 3.058 sencontrou 0 vulnerabilidades Quando você instala um pacote com o NPM, o NPM registra esta dependência - que seu projeto depende do Express - no arquivo package.json. Com Esta dependência registrada no package.json, você pode dar outro programador uma cópia do seu código e seu package.json, e ele poderia simplesmente digitar o NPM instalar para baixar automaticamente elnstale todas as bibliotecas que seu programa precisa para executar. 16.2 O nó é assíncrono por padrão. JavaScript é uma linguagem de programação de uso geral, então é perfeitamente possível escrever programas intensivos em CPU que multipliquem grandes matrizes ou realizam análises estatísticas complicadas. Mas o nó era projetado e otimizado para programas - como servidores de rede - que são E/S intensivo. E em particular, o nó foi projetado para tornar possível Para implementar facilmente servidores altamente simultâneos que podem lidar com muitos solicitações ao mesmo tempo. Ao contrário de muitas linguagens de programação, no entanto, o nó não consegue simultaneidade com threads. A programação multithreaded é notoriamente difícil de fazer corretamente e difícil de depurar. Além disso, tópicos são um Abstração relativamente pesada e se você quiser escrever um servidor que podem lidar com centenas de solicitações simultâneas, usando centenas de

Os threads podem exigir uma quantidade proibitiva de memória. Então o nó adota o modelo de programação JavaScript de thread único que a web usa. E isso acaba sendo uma vasta simplificação que torna a criação de servidores de rede uma habilidade de rotina em vez de uma mistura arcana. Parallelismo verdadeiro com nós. Os programas de nó podem executar vários processos do sistema operacional e o nó 10 e o trabalhador desempenha posteriormente objetos (§16.11), que são um tipo de tópico emprestado dos navegadores da Web. Se você usar múltiplos processos ou criar um ou mais tópicos de trabalhadores e executar seu programa em um sistema com mais de uma CPU, então seu programa não será mais um thread único e seu programa será realmente executando vários fluxos de código em paralelo. Essas técnicas podem ser valiosas para a CPU intensiva operações, mas não são comumente usadas para programas intensivos em E/O, como servidores. Vale a pena notar, no entanto, que os processos e os trabalhadores do nó evitam a complexidade típica de programação multithreaded porque a comunicação de interprocesso e trabalho de trabalho é via mensagem passando e eles não podem compartilhar facilmente a memória entre si. O nó alcança altos níveis de simultaneidade, mantendo um único modelo de programação rosqueada, tornando sua API assíncrona e não bloqueio por padrão. Node adota muito sua abordagem sem bloqueio sério e é um extremo que pode surpreendê-lo. Você provavelmente espere funções que leiam e escreva para a rede seja assíncrono, mas o nó vai além e define não bloqueio funções assíncronas para ler e escrever arquivos do localFilesystem. Isso faz sentido, quando você pensa sobre isso: a API do nó foi projetado nos dias em que os discos rígidos girando ainda eram a norma. E realmente havia milissegundos de bloquear "busca tempo" enquanto esperava o disco girando antes que uma operação de arquivo possa começar. E em datacenters modernos, o sistema de arquivos "local" pode realmente serem toda a rede em algum lugar com latências de rede em cima da unidade de latências. Mas mesmo se ler um arquivo de forma assíncrona parece normal a você, o nó leva ainda mais: as funções padrão para iniciar um

conexão de rede ou para procurar um tempo de modificação de arquivo, paraExemplo, também são não bloqueadores. Algumas funções na API do Node são síncronas, mas não bloqueando: elasCorra até a conclusão e retorne sem nunca precisar bloquear. Mas a maioria das funções interessantes executam algum tipo de entrada ou saída, e essas são funções assíncronas para que possam evitar até o menor quantidade de bloqueio. O nó foi criado antes do JavaScript ter uma promessaClasse, as APIs de nó assíncronas são baseadas em retorno de chamada. (Se você não temNo entanto, leia ou já esqueci o capítulo 13, isso seria um bomHora de pular de volta para esse capítulo.) Geralmente, o último argumento que vocêPassar para uma função de nó assíncrona é um retorno de chamada. Nó usa erro-Os primeiros retornos de chamada, que normalmente são invocados com dois argumentos. O primeiro argumento para um retorno de chamada de erro é normalmente nulo no caso onde nenhum erro ocorreu, e o segundo argumento são os dados ouA resposta foi produzida pela função assíncrona original que vocêchamado. O motivo para colocar o argumento de erro primeiro é fazê-lo impossível para você omiti-lo, e você sempre deve verificar um nãovalor nulo neste argumento. Se for um objeto de erro, ou mesmo um número inteiroCódigo de erro ou mensagem de erro da string, então algo deu errado. NestaCaso, o segundo argumento para sua função de retorno de chamada provavelmente será nulo. O código a seguir demonstra como usar o não bloqueioFunção readfile () para ler um arquivo de configuração, analisá-lo como JSON, e depois passe o objeto de configuração analisada para outro retorno de chamada: const fs = requer ("fs"); // requer o módulo do sistema de arquivos // Leia um arquivo de configuração, analisou seu conteúdo como JSON e passe

o// Valor resultante para o retorno de chamada.Se algo der errado,// Imprima uma mensagem de erro para Stderr e invoque o retorno de chamada com nulofunção readconfigfile (caminho, retorno de chamada) {fs.readFile (caminho, "utf8", (err, text) => {se (err) {// algo deu errado lendo o arquivoconsole.error (err);retorno de chamada (nulo);retornar;}deixe dados = nulo;tentar {dados = json.parse (texto);} catch (e) {// algo deu errado analisando o Conteúdo do arquivoconsole.error (e);}retorno de chamada (dados);});}O nó antecede as promessas padronizadas, mas porque é bastante consistenteSobre seus retornos de chamada em primeiro lugar, é fácil criar promessas baseadas em promessasvariantes de suas APIs baseadas em retorno de chamada usando o util.promisify ()invólucro.Veja como podemos reescrever o readConfigfile ()função para retornar uma promessa:const util = requer ("util");const fs = requer ("fs");// requer o módulo do sistema de arquivosconst pfs = {// variantes baseadas em promessas de algunsfunções fsReadFile: util.promisify (fs.readFile)};função readConfigfile (Path) {Retorne pfs.readFile (Path, "UTF-8"). Então (texto => {

Erro ao traduzir esta página.

este código e escreva uma versão puramente síncrona do nosso função `ReadConfigFile()`. Em vez de invocar um retorno de chamada ou retornando uma promessa, essa função simplesmente retorna o JSON analisado valor ou joga uma exceção:
const fs = requer ("fs");
função `readconfigfilesync` (path) {
deixe texto = `fs.readfilesync` (caminho,
"utf-8");
retornar `json.parse` (texto);}
Além de seus retornos de chamada de dois argumentos de erro, o Node também possui um número de APIs que usam a assincronia baseada em eventos, normalmente paramanusear dados de streaming. Cobriremos os eventos do nó com mais detalhes mais tarde. Agora que discutimos a API agressivamente sem bloqueio de Node, vamosVolte ao tópico da simultaneidade. Não bloqueio embutido do Node funções funcionam usando a versão de retornos de chamada do sistema operacional e Manipuladores de eventos. Quando você chama uma dessas funções, o nó levaação para iniciar a operação e depois registra algum tipo de eventomanipulador com o sistema operacional para que seja notificado quando oOperação está completa. O retorno de chamada que você passou para a função do nó é armazenado internamente para que o nó possa invocar seu retorno de chamada quando oO sistema operacional envia o evento apropriado para o nó. Esse tipo de simultaneidade é frequentemente chamado de concorrência baseada em eventos. NoSeu núcleo, nó tem um único thread que executa um "loop de eventos". Quando aO programa de nó é iniciado, ele executa o código que você disse para executar. EsseCódigo presumivelmente chama pelo menos uma função não bloqueadora, causando umRetorno de chamada ou manipulador de eventos a ser registrado no sistema operacional.(Se

Não, então você escreveu um programa de nós síncronos e um nós simplesmente sai quando chegar ao fim.) Quando o nó chega ao fim de seu programa, ele bloqueia até que um evento aconteça, quando o sistema operacional inicia correndo novamente. Ele mapeia o evento do sistema operacional para o JavaScript. O retorno de chamada que você se registrou é chamado dessa função. Seu retorno de chamada da função pode invocar mais funções de nó não bloqueador, causando mais os manipuladores de eventos do sistema operacional a serem registrados. Uma vez que sua função de retorno de chamada for feita, o Node volta a dormir novamente e o ciclo se repete. Para servidores da web e outros aplicativos intensivos em E/S que gastam a maior parte de seu tempo esperando por entrada e saída, esse estilo de baseado em eventos é simultâneo e eficiente. Um servidor da web pode simultaneamente lidar com solicitações de 50 clientes diferentes sem precisar de 50 diferentes tópicos, desde que usem APIs não bloqueadoras e exista algum tipo de mapeamento interno de soquetes de rede para funções de JavaScript paralelo que ocorre nesses soquetes.

16.3 Buffers

Um dos tipos de dados que você provavelmente usará com frequência no nó - especialmente ao ler dados de arquivos ou da rede - é a classe de buffer. Um buffer é muito parecido com uma corda, exceto que é uma sequência de bytes em vez de uma sequência de caracteres. O nó foi criado antes do Core JavaScript suportado matrizes digitadas (ver §11.2) e não havia `UINT8Array` para representar uma variedade de bytes não assinados. Ele definiu a classe de buffer para preencher essa necessidade. Agora que `uint8array` faz parte do Javascript Language, a classe de buffer do Node é uma subclasse do `UINT8Array`. O que distingue o buffer de sua superclasse `uint8Array` é que é

Projetado para interoperar com Strings JavaScript: os bytes em um buffer pode ser inicializado a partir de seqüências de caracteres ou convertido em personagens. Um personagem que codifica mapeia cada personagem em algum conjunto de personagens para um número inteiro. Dado uma série de texto e um personagem codificação, podemos codificar os caracteres na string em uma sequência de bytes. E dada uma sequência (adequadamente codificada) de bytes e uma codificação de caracteres, podemos decodificar esses bytes em uma sequência de caracteres. A classe de buffer do nó tem métodos que executam os dois codificações e decodificações, e você pode reconhecer esses métodos porque eles esperam um argumento codificador que especifica que a codificação é usada. As codificações no nó são especificadas pelo nome, como strings. O suporte a codificações são: "UTF8" Este é o padrão quando nenhuma codificação é especificada e é o Unicode codificando você é mais provável de usar. "UTF16LE" Caracteres unicode de dois bytes, com pedidos pouco endianos. Os pontos de código acima \ uffff são codificados como um par de dois bytes. Sequências. A codificação "UCS2" é um pseudônimo. "Latin1" A codificação ISO-8859-1 de um byte por caractere que define um conjunto de personagens adequado para muitos idiomas da Europa Ocidental. Porque há um mapeamento individual entre bytes e latim-1 Personagens, essa codificação também é conhecida como "binário".

"ASCII" A codificação ASCII somente em inglês de 7 bits, um subconjunto estrito do "UTF8" codificação."Hex" Esta codificação converte cada byte em um par de ASCII hexadecimais dígitos."Base64" Esta codificação converte cada sequência de três bytes em uma sequência de quatro caracteres ASCII.Aqui está algum código de exemplo que demonstra como trabalhar com buffers e como se converter de e para as cordas:Seja b = buffer.From ([0x41, 0x42, 0x43]); // <buffer41 42 43>b.ToString () // =>"ABC"; padrão "utf8" B.ToString ("Hex") // =>"414243" deixe computador = buffer.from ("IBM3111", "ASCII"); // converter string para buffer para (vamos i = 0; i < Computer.Length; i++) { // UseBuffer como matriz de bytes computador [i]; // buffers são mutáveis} Computer.ToString ("ASCII") // =>"HAL2000" Computer.Subarray (0,3) .Map (x => x+1) .ToString () // => "IBM" // Crie novos buffers "vazios" com buffer.Alloc () Seja zeros = buffer.Alloc (1024); // 1024 zeros Seja o ONS = Buffer.Alloc (128, 1); // 128

Let Dead = Buffer.Alloc (1024, "Deadbeef", "Hex");//Padrão de repetição de bytes// buffers têm métodos para ler e escrever multi-bytesvalores// de e para um buffer em qualquer deslocamento especificado.Dead.readUInt32be (0) // => 0xdeadbeefDead.readUInt32be (1) // => 0xadbeefdeDead.readbiguint64be (6) // => 0xbeefdeadbeefdeadnDead.readUInt32LE (1020) // => 0xEfbeaddeSe você escrever um programa de nós que realmente manipula dados binários, vocêPode se encontrar usando a classe buffer extensivamente.Por outromão, se você está apenas trabalhando com texto que é lido ou escrito para umarquivo ou rede, então você pode encontrar apenas buffer como umRepresentação intermediária de seus dados.Uma série de APIs de nó podemPegue a saída de entrada ou retorno como strings ou objetos de buffer.Tipicamente,Se você passar por uma string ou espera que uma string seja devolvida, de um dessesAPIs, você precisará especificar o nome da codificação de texto que desejausar.E se você fizer isso, talvez não precise usar um objeto buffer emtodos.16.4 Eventos e EventEmitterComo descrito, todas as APIs do Node são assíncronas por padrão.ParaMuitos deles, essa assincronia assume a forma de erro de dois argumentos-Primeiros retornos de chamada que são invocados quando a operação solicitada écompleto.Mas algumas das APIs mais complicadas são baseadas em eventosem vez de.Este é normalmente o caso quando a API é projetada em torno de umobjeto em vez de uma função, ou quando uma função de retorno de chamada precisa serinvocado várias vezes, ou quando houver vários tipos de retorno de chamadafunções que podem ser necessárias.Considere a classe Net.erver, para

Exemplo: um objeto desse tipo é um soquete de servidor usado para aceitar conexões recebidas de clientes. Emite um evento de "escuta" quando o primeiro começa a ouvir conexões, um evento de "conexão" toda vez que o cliente se conecta, e um evento "próximo" quando foi fechado e não está ouvindo mais. No nó, objetos que emitem eventos são casos de EventEmitter ou uma subclasse do EventEmitter:
const EventEmitter = require ("eventos"); // O nome do módulo faz NOME DO MAIOR CLASSE NOME
const net = require ("net");
Deixe servidor = new net.Server (); // Crie um servidor
objetoInstância do servidor de EventEmitter // => true: servidores
A principal característica dos observadores de eventos é que eles permitem que você se registre. O manipulador de eventos funciona com o método on (). Os observadores de eventos podem emitir vários tipos de eventos e tipos de eventos são identificados por nome. Para registrar um manipulador de eventos, ligue para o método on (), passando o nome do tipo de evento e a função que deve ser invocada quando um evento desse tipo ocorre. Os observadores de eventos podem invocar as funções do manipulador com qualquer número de argumentos e você precisa ler a documentação para um tipo específico de evento de uma empresa de eventos específica para saber o que os argumentos que você deve ser aprovado:
const net = require ("net");
Deixe servidor = new net.Server (); // Crie um servidor
objeto server.on ("conexão", soquete => { // Ouça eventos de "conexão" // Os eventos de "conexão" do servidor passam um objeto de soquete // para o cliente que acabou de conectar. Aqui enviamos alguns

Erro ao traduzir esta página.

Você mantém o seu manipulador de eventos funciona sem bloqueio e rápido. Se você precisa fazer muito cálculo quando ocorrer um evento, geralmente é melhor usar o manipulador para agendar esse cálculo de forma assíncrona usando `setTimeout()` (consulte §11.10). O nó também define `setImmediate()`, que agenda uma função a ser invocada imediatamente depois de todos os retornos de chamada e eventos pendentes, foram tratados. A classe `EventEmitter` também define um método `emit()` que causa as funções de manipulador de eventos registradas a serem invocadas. Isso é útil se você estiver definindo sua própria API baseada em eventos, mas não é comumente usado. Quando você está apenas programando com APIs existentes, `emit()` deve ser invocado com o nome do tipo de evento como seu primeiro argumento. Qualquer argumentos adicionais que são passados ?? para `emit()` se tornarem argumentos para as funções do manipulador de eventos registradas. As funções do manipulador também são invocadas com o valor este definido para o próprio objeto de `EventEmitter`, o que geralmente é conveniente. (Lembre -se, porém, essa flecha funciona. Sempre use o valor deste contexto em que eles são definidos, e eles não podem ser invocados com nenhum outro esse valor. No entanto, as funções de seta geralmente são a maneira mais conveniente de escrever um evento manipuladores.) Qualquer valor retornado por uma função de manipulador de eventos é ignorado. Se um evento A função manipuladora lança uma exceção, no entanto, se propaga de uma chamada `emit()` e impede a execução de qualquer função de manipulador que foram registrados após o que lançou a exceção. Lembre-se de que as APIs baseadas em retorno de chamada do Node usam retornos de chamada de erro e é importante que você sempre verifique o primeiro argumento de retorno de chamada para ver se

ocorreu um erro.Com APIs baseadas em eventos, o equivalente é "erro"eventos.Como as APIs baseadas em eventos são frequentemente usadas para networking e outrasformas de streaming de E/S, elas são vulneráveis ??a imprevisíveisErros assíncronos, e a maioria dos apresentadores de eventos define um evento de "erro"que eles emitem quando ocorre um erro.Sempre que você usa um evento baseado em eventoAPI, você deve ter o hábito de registrar um manipulador para eventos de "erro".Os eventos de "erro" recebem tratamento especial da classe EventEmitter.Seemit () é chamado para emitir um evento de "erro" e se não houver manipuladoresRegistrado para esse tipo de evento, uma exceção será lançada.DesdeIsso ocorre de forma assíncrona, não há como você lidar com oExceção em um bloco de captura, então esse tipo de erro normalmente causa o seu programa para sair.16.5 fluxosAo implementar um algoritmo para processar dados, é quase sempremais fácil de ler todos os dados na memória, fazer o processamento e depoisEscreva os dados.Por exemplo, você pode escrever uma função de nó paraCopie um arquivo como este.const fs = requer ("fs");// um assíncrono, mas semamaming (e, portanto,Função ineficiente).Função CopyFile (SourceFileName, DestinationFilename,ligar de volta) {fs.readFile (sourcefilename, (err, buffer) => {if (err) {retorno de chamada (err);} outro {fs.writeFile (DestinationFilename, Buffer,ligar de volta);1

});}Esta função copyfile () usa funções assíncronas eretornos de chamada, para que não bloqueie e seja adequado para uso em simultâneoprogramas como servidores.Mas observe que deve alocar memória suficientePara manter todo o conteúdo do arquivo na memória de uma só vez.Isso pode ser bem em alguns casos de uso, mas começa a falhar se os arquivos a serem copiados forem muito grande, ou se o seu programa for altamente simultâneo e pode haver muitos arquivos sendo copiados ao mesmo tempo.Outra falha dissoA implementação copyfile () é que ele não pode começar a escrever o novoArquivo até terminar de ler o arquivo antigo.A solução para esses problemas é usar algoritmos de streaming ondeOs dados ?fluem? para o seu programa são processados ??e depois fluem para fora deseu programa.A idéia é que seu algoritmo processe os dadosPequenos pedaços e o conjunto de dados completo nunca são mantidos na memória de uma só vez.Quando as soluções de streaming são possíveis, elas são mais eficientes em memória e também pode ser mais rápido.As APIs de rede da Node são baseadas em fluxos eO módulo de sistema de arquivos do Node define o streaming de APIs para leitura e escrevendo arquivos, é provável que você use uma API de streaming em muitos dosProgramas de nós que você escreve.Veremos uma versão de streaming doCopyFile () Função em "Modo de fluxo".O nó suporta quatro tipos básicos de fluxo:LegívelOs fluxos legíveis são fontes de dados.O fluxo retornou por

`fs.createReadStream()`, por exemplo, é um fluxo de que o conteúdo de um arquivo especificado pode ser lido. `Process.stdin` é outro fluxo legível que retorna dados da entrada padrão. Graváveis Fluxos graváveis ??são sumidouros ou destinos para dados. O valor de retorno de `fs.createWriteStream()`, por exemplo, é uma `gravidadeStream`: permite que os dados sejam gravados em pedaços e produz todos desses dados para um arquivo especificado. Duplex Os fluxos duplex combinam um fluxo legível e um fluxo gravável em um objeto. Os objetos de soquete retornados por `net.connect()` e outras APIs de rede de nó, por exemplo, são fluxos duplex. Se você escrever em um soquete, seus dados serão enviados em toda a rede para Qualquer computador ao qual o soquete esteja conectado. E se você ler de um soquete, você acessa os dados escritos por esse outro computador. Transformar Os fluxos de transformação também são legíveis e graváveis, mas eles diferem de fluxos duplex de uma maneira importante: dados escritos para um fluxo de transformação se tornam legíveis - geralmente em alguma forma - do mesmo fluxo. A função `zlib.createGzip()`, por exemplo, retorna um fluxo de transformação que comprime (com o algoritmo GZIP) os dados gravados para ele. De maneira semelhante, a função `Crypto.createCipheriv()` retorna uma transformação que criptografa ou descriptografa dados que são gravados para eles. Por padrão, fluxos de leitura e gravação de buffers. Se você ligar para o método `setEncoding()` de um fluxo legível, ele retornará strings decodificados para você em vez de objetos de buffer. E se você escrever um

string a um buffer gravável, ele será codificado automaticamente usando oA codificação padrão do buffer ou qualquer codificação que você especificar.NóA API de stream também suporta um "modo de objeto" em que os fluxos leem eEscreva objetos mais complexos do que buffers e cordas.Nenhum dos nóAs APIs principais usam este modo de objeto, mas você pode encontrá -lo em outrosBibliotecas.Fluxos legíveis precisam ler seus dados de algum lugar eFluxos graváveis ??precisam escrever seus dados em algum lugar, então todo fluxo tem duas extremidades: uma entrada e uma saída ou uma fonte e umdestino.O complicado das APIs baseadas em fluxo é que os doisAs extremidades do fluxo quase sempre fluem em velocidades diferentes.TalvezO código que lê de um fluxo deseja ler e processar dados maisRapidamente do que os dados está realmente sendo escrito no fluxo.Ou oreverso: talvez os dados sejam gravados para um fluxo mais rapidamente do que pode serLeia e saiu do riacho da outra extremidade.FluxoAs implementações quase sempre incluem um buffer interno para manter dadosIsso foi escrito, mas ainda não lido.Buffers ajuda a garantir quehá dados disponíveis para ler quando solicitados e que háespaço para manter dados quando estiver escrito.Mas nenhuma dessas coisas podeSempre garantido, e é a natureza da programação baseada em fluxoque os leitores às vezes terão que esperar que os dados sejam escritos (porqueo buffer de fluxo está vazio), e os escritores às vezes precisam esperar pordados a serem lidos (porque o buffer de fluxo está cheio).Em ambientes de programação que usam simultaneidade baseada em roscas,As APIs de fluxo geralmente têm chamadas de bloqueio: uma chamada para ler dados nãoRetorne até que os dados cheguem no fluxo e uma chamada para escrever blocos de dadosaté que haja espaço suficiente no buffer interno do fluxo para

Erro ao traduzir esta página.

função PipeFileToSocket (nome do arquivo, soquete) {fs.createReadStream (nome do arquivo) .pipe (soquete);}A função de utilidade a seguir tubula um fluxo para outro e invoca um retorno de chamada quando feito ou quando ocorre um erro:Pipe de função (legível, gravável, retorno de chamada) {// Primeiro, configure o manuseio de erros função handleerror (err) {readerable.Close ();writable.close ();retorno de chamada (err);}// Definir o tubo e lidar com a terminação normal caso legível.On ("Erro", HandleError).pipe (gravável).On ("Erro", HandleError).On ("acabamento", retorno de chamada);}Os fluxos de transformação são particularmente úteis com tubos e criam oleodutos que envolvem mais de dois fluxos.Aqui está um exemplo função que comprime um arquivo:const fs = requer ("fs");const zlib = requer ("zlib");função gzip (nome do arquivo, retorno de chamada) {// Crie os fluxos deixa-se fonte = fs.createReadStream (nome do arquivo);Let Destination = Fs.CreateWriteStream (nome do arquivo + ".gz");deixe zipper = zlib.createGzip ();

```
// Configure o pipeline
fonte.on ("Erro", retorno de chamada) // Ligue para o retorno de chamada em leitura
aerro.pipe (gzipper).pipe (destino).on ("Erro", retorno de chamada) // Ligue para o retorno de chamada na gravação
oerro.On ("acabamento", retorno de chamada); // Ligue para o retorno de chamada quando Escrever está completo}
Usando o método Pipe () para copiar dados de um fluxo legível para um O fluxo gravável é fácil, mas na prática, muitas vezes você precisa processar o Dados de alguma forma, enquanto fluem através do seu programa. Uma maneira de fazer isso é implementar seu próprio fluxo de transformação para fazer esse processamento e Esta abordagem permite que você evite ler manualmente e escrever os fluxos. Aqui, por exemplo, é uma função que funciona como o Unix Utilitário Grep: Ele lê linhas de texto de um fluxo de entrada, mas escreve apenas as linhas que correspondem a uma expressão regular especificada:
const stream = requer ("stream");
classe Grepstream estende Stream.Transform {
    construtor (padrão) {
        Super ({DecodeStrings: false});
        // não converte Strings de volta aos buffer
        this.pattern = padrão;
        // o regularexpressão que queremos combinar
        this.incompleteTeline = "";
        // qualquer remanescente do último pedaço de dados
    }
    Este método é invocado quando há uma string pronta para ser transformado. Deve passar dados transformados para o especificado
    Função de retorno de chamada. Esperamos entrada de string, então esta
```

o fluxo deve// só pode ser conectado a fluxos legíveis que tiveram// setEncoding () chamou neles._Transform
(Chunk, codificação, retorno de chamada) {if (typeof chunk! == "string") {retorno de chamada (novo erro ("Esperava
uma string, mas obteve umbuffer "));retornar;}// Adicione o pedaço a qualquer linha anteriormente incompletae
quebrar// tudo em linhasdeixe linhas = (this.incompleteLeline +chunk) .split ("\ n");// O último elemento da matriz é o
novolinha incompletathis.incompleTeline = lines.pop ()// Encontre todas as linhas correspondentesDeixe a saída =
linhas // comece comTodas as linhas completas,.filter (l => this.pattern.test (l)) // filtre -ospara partidas,.Join ("\ n");//
e junte -seelas de volta// Se alguma coisa corresponde, adicione uma nova linha finalif (output) {saída += "\ n";}//
sempre ligue para o retorno de chamada, mesmo que não hajaídareturno de chamada (nulo, saída);}// Isso é
chamado logo antes do fechamento do fluxo// É a nossa chance de escrever qualquer último dados._flush (retorno
de chamada) {// se ainda tivermos uma linha incompleta, e elapartidas// Passe para o retorno de chamadaif
(this.pattern.test (this.incompleteline)) {

retorno de chamada (nulo, this.incompleteline + "\n");}}}// Agora podemos escrever um programa como 'Grep' com esta classe.deixe padrão = novo regexp (process.argv [2]);// Obtenha um regexp da linha de comando.process.stdin // comece comentada padrão,.setEncoding ("utf8") // leia -o como Strings Unicode.,pipe (novo Grepstream (padrão) // Pipa para o nossoGrepstream.,pipe (process.stdout) // e pico que para padrão..on ("erro", () => process.exit ());// Saia graciosamenteSe o stdout fechar.16.5.2 iteração assíncronaNo nó 12 e posterior, fluxos legíveis são iteradores assíncronos,o que significa que dentro de uma função assíncrona você pode usar umpara/aguarda loop para ler string ou buffer pedaços de um fluxo usandoCódigo estruturado como código síncrono seria.(Veja §13.4 paramais sobre iteradores assíncronos e loops para/aguardam.)Usar um iterador assíncrono é quase tão fácil quanto usar o tubo ()método, e provavelmente é mais fácil quando você precisa processar cada pedaçoVocê lê de alguma forma.Veja como poderíamos reescrever o programa Grepna seção anterior usando uma função assíncrona e um para/aguardarlaço:// leia linhas de texto do fluxo de origem e escreva qualquerlinhas// que corresponde ao padrão especificado com o destino

```
fluxo.Função assíncrona grep (fonte, destino, padrão,coding = "utf8") { // Configure o fluxo de origem para ler strings, nãoBuffersfonte.SetEncoding (codificação); // Defina um manipulador de erros no fluxo de destino, caso padrão // a saída se fecha inesperadamente (quando a tubulação de saída para `Head` , por exemplo) Destination.On ("Error", err => process.exit ()) ; // Os pedaços que lemos provavelmente terminam com uma nova linha,Então cada vontade// provavelmente tem uma linha parcial no final.Acompanhe isso aqui deixa incompleteline = "";// use um loop for/aguardar para ler assíncronos do fluxo de entrada para aguardar (Let Chunk of Source) { // dividiu o final do último pedaço e este em linhas Let lines = (incompleto Linear + Chunk) .split ("\ n"); // A última linha está incompleta incompleteLine = lines.pop () ; // agora percorre as linhas e escreva qualquer correspondência para o destino para (deixe a linha de linhas) {if (padrony.test (line)) {Destination.write (linha + "\ n", codificação);}} } // Finalmente, verifique se há uma correspondência em qualquer texto à direita.if (padring.test (incompleto)) {Destination.Write (IncompleteLeline + "\ n", codificação); } } deixe padrão = novo regexp (process.argv [2]); // Obtenha um regexp da linha de comando.
```

```
grep (process.stdin, process.stdout, padrão) // ligue para o função assíncrona grep ()..catch (err => {//
manipulaexceções assíncronas.console.error (err);process.Exit ();});16.5.3 Escrevendo para fluxos e
manuseioBackpressureA função Async Grep () no exemplo de código anterior demonstrou como usar um fluxo
legível como um assíncrono iterador, mas também demonstrou que você pode escrever dados para um
gravadorfluxo simplesmente passando -o para o método write ().O write ()O método leva um buffer ou string como
o primeiro argumento.(Fluxos de objetosEspere outros tipos de objetos, mas estão além do escopo deste
capítulo.)Se você passar por um buffer, os bytes desse buffer serão escritos diretamente.Sevocê passa uma corda,
ela será codificada para um buffer de bytes antes de serescrito.Fluxos graváveis ??têm uma codificação padrão
que é usada quandoVocê passa uma string como o único argumento a escrever ().O padrãoA codificação é
tipicamente "UTF8", mas você pode defini -lo explicitamente ligandosetDefaultEncoding () no fluxo
gravável.Alternativamente,Quando você passa uma string como o primeiro argumento a escrever (), você pode
passarum nome codificante como o segundo argumento.Write () opcionalmente assume uma função de retorno de
chamada como seu terceiro argumento.Isso será invocado quando os dados tiverem sido escritos e não formais
tempo no buffer interno do fluxo gravável.(Este retorno de chamada também podeser chamado se ocorrer um erro,
mas isso não for garantido.Você deveRegistre um manipulador de eventos de "erro" no fluxo gravável para
detectar
```

erros.) O método write () possui um valor de retorno muito importante. Quando você ligue para Write () em um fluxo, ele sempre aceita e amortece o pedaço de dados que você passou. Então ele retorna verdadeiro se o buffer interno for ainda não está cheio. Ou, se o buffer agora estiver cheio ou muito cheio, ele retornará falso. Esse valor de retorno é consultivo e você pode ignorá-lo - fluxos de escritórios vai ampliar seu buffer interno o máximo necessário se você continuar ligando escrever(). Mas lembre-se de que o motivo de usar uma API de streaming na primeira vez é evitar o custo de manter muitos dados na memória em uma vez. Um valor de retorno de false do método write () é uma forma de Backpressure: uma mensagem do fluxo que você escreveu dados mais rapidamente do que pode ser tratado. A resposta adequada a esse tipo de contrapressão é parar de chamar a gravação () até que o fluxo emite um Evento de "drenagem", sinalizando que há mais uma vez espaço no buffer. Aqui, por exemplo, é uma função que grava em um fluxo e depois invoca um retorno de chamada quando não há problema em escrever mais dados para o fluxo:

```
função write (stream, chunk, retorno de chamada) {  
    // Escreva o pedaço especificado para o fluxo especificado  
    // e deixe hasMoreRoom = stream.write (chunk);  
    // Verifique o valor de retorno do método write ():  
    if (hasMoreRoom) {  
        // se ele retornar é verdade que setImmediate (retorno de chamada);  
        // Invoco o retorno de chamada assíncrono.  
    } else {  
        // se retornar falso, então stream.once ("drenagem", retorno de chamada);  
        // Invocar o retorno de chamada em Evento de drenagem.  
    }  
}
```

}}O fato de que às vezes não há problema em chamar Write () várias vezes em umlinha e às vezes você tem que esperar por um evento entre as gravaçõescria algoritmos desajeitados.Esta é uma das razões que usamO método Pipe () é tão atraente: quando você usa Pipe (), NÓlida com a contrapressão para você automaticamente.Se você está usando aguardar e assíncrono em seu programa, e está tratandoFluxos legíveis como iteradores assíncronos, é diretoImplementar uma versão baseada em promessa da função de utilidade write ()acima para manipular corretamente a contrapressão.Na função Async Grep ()Nós apenas olhamos, não lidamos com a contrapressão.A cópia assíncrona ()A função no exemplo a seguir demonstra como pode ser feito corretamente.Observe que esta função apenas copia pedaços de uma fontetransmita para um fluxo de destino e cópia de chamada (fonte,destino) é como ligarfonte.pipe (destino):// Esta função escreve o pedaço especificado para o especificadostream e// retorna uma promessa que será cumprida quando estiver bomEscreva novamente.// Como retorna uma promessa, pode ser usado com aguardar.função write (stream, chunk) { // Escreva o pedaço especificado para o fluxo especificadodeixe hasmoreroom = stream.write (chunk);if (hasMoreroom) { // se o buffer forNão está cheio, retorneReturn Promise.Resolve (NULL); // jáObjeto de promessa resolvida} outro {

retornar nova promessa (resolve => {// caso contrário, devolver uma promessa que stream.once ("drenagem", resolver); // resolve no Evento de drenagem.});};// Copie dados do fluxo de origem para o fluxo de destino// respeitando a contrapressão do fluxo de destino.// É como chamar a fonte.pipe (destino).cópia da função assíncrona (fonte, destino) {// Defina um manipulador de erros no fluxo de destino, casopadrão// a saída se fecha inesperadamente (quando a tubulação de saída para `Head`, por exemplo)Destination.On ("Error", err => process.exit());// use um loop for/aguardar para ler assíncronos do fluxo de entrada para aguardar (Let Chunk of Source) {// Escreva o pedaço e espere até que haja mais espaço no buffer.aguardar escrita (destino, pedaço);};// Copiar entrada padrão para saída padrão copy (process.stdin, process.stdout);Antes de concluirmos esta discussão sobre escrever para fluxos, observe novamente que não responder à contrapressão pode fazer com que seu programa use mais memória do que deveria quando o buffer interno de uma gravação transborda e cresce cada vez maior.Se você está escrevendo um Servidor de rede, isso pode ser um problema de segurança remotamente explorável.Suponha que você escreva um servidor HTTP que entregue arquivos pela rede,Mas você não usou Pipe () e não levou um tempo para lidarBackpressure do método write ().Um atacante poderia escrever um

Cliente HTTP que inicia solicitações de arquivos grandes (como imagens), mas Nunca realmente lê o corpo do pedido. Como o cliente não é ler os dados sobre a rede e o servidor não está respondendo a Backpressure, os buffers do servidor vão transbordar. Com o suficiente Conexões simultâneas do atacante, isso pode se transformar em uma negação de serviço que diminui o servidor ou até trava.

-o.16.5.4 Lendo fluxos com eventos

Os fluxos legíveis do Node têm dois modos, cada um dos quais tem seu próprio API para leitura. Se você não pode usar tubos ou iteração assíncrona em seu programa, você precisará escolher uma dessas duas APIs baseadas em eventos para lidar com fluxos. É importante que você use apenas um ou outro e não misture as duas APIs.

Modo de fluxo

No modo de fluxo, quando os dados legíveis chegam, eles são imediatamente emitidos na forma de um evento de "dados". Para ler de um fluxo neste modo, basta registrar um manipulador de eventos para eventos de "dados", e o fluxo irá empurrar pedaços de dados (buffers ou cordas) para você assim que eles se tornarem disponíveis. Observe que não há necessidade de chamar o método `read()` em Modo de fluxo: você só precisa lidar com eventos "dados". Observe isso recentemente. Os fluxos criados não começam no modo de fluxo. Registrando um "dados" O manipulador de eventos alterna um fluxo para o modo de fluxo. Convenientemente, isso significa que um fluxo não emite eventos de "dados" até você registrar o primeiro manipulador de eventos "dados". Se você estiver usando o modo de fluxo para ler dados de um fluxo legível, processe-o e depois escreva em um fluxo gravável, então você pode precisar

Manuseie a contrapressão do fluxo gravável. Se a gravação () o método retorna false para indicar que o buffer de gravação está cheio, você pode ligar para pausa () no fluxo legível para interromper temporariamente os dados de eventos. Então, quando você obtém um evento de "dreno" do fluxo gravável, você pode ligar para o currículo () no fluxo legível para iniciar os "dados" eventos fluindo novamente. Um fluxo no modo de fluxo emite um evento "final" quando o fim do fluxo é alcançado. Este evento indica que não há mais eventos de "dados" sempre emitido. E, como em todos os fluxos, um evento de "erro" é emitido se um erro ocorre. No início desta seção em transmissões, mostramos um uso não transportadoCopyFile () função e prometeu uma versão melhor que está por vir. OA seguir, o código mostra como implementar um streaming copyfile () Função que usa a API do modo de fluxo e lida com a contrapressão. Isso teria sido mais fácil de implementar com uma chamada de tubo (), mas serve aqui como uma demonstração útil dos múltiplos manipuladores de eventos que são usados ?? para coordenar o fluxo de dados de um fluxo para o outro.

```
const fs = require('fs'); // Uma função de cópia do arquivo de streaming, usando "modo de fluxo". // copia o conteúdo do arquivo de origem nomeado para o nomeado arquivo de destino. // No sucesso, invoca o retorno de chamada com um argumento nulo. Sobre erro, // chama o retorno de chamada com um objeto de erro. Função CopyFile (SourceFileName, DestinationFilename, ligar de volta) { Deixe input = fs.createReadStream (SourceFileName); deixe output = fs.createWriteStream (destinofilename);
```

```
input.on ("dados", (chunk) => {// quando obtivemos novos dados, deixe hasroom = output.write (chunk); // Escreva para o fluxo de saída. if (! hasroom) {// se a saída do fluxo está cheia, input.Pause () // então pausa o fluxo de entrada.}); input.on ("end", () => {// quando chegarmos ao fim da entrada, output.end () // Diga à saída stream para terminar.}); input.on ("erro", err => {// se conseguirmos um erro na entrada, retorno de chamada (err); // Ligue para o retorno de chamada com o erro process.Exit () // e desiste.}); output.on ("dreno", () => {// quando a saída não está mais cheia, input.Resume () // retomar dadosEventos na entrada}); output.on ("erro", err => {// se obtivermos um erro na saída, retorno de chamada (err); // Ligue para o retorno de chamada com o erro process.Exit () // e desiste.}); output.on ("acabamento", () => {// quando a saída é totalmente escrita, retorno de chamada (nulo); // Ligue para o retorno de chamada sem erro.}); // Aqui está um utilitário simples da linha de comando para copiar arquivos. deixe de = process.argv [2], para = process.argv [3];
```

console.log(`copiando arquivo \${de} para \${para} ...`);copyfile(de, para, err => {if (err) {console.error(err);} outro {console.log("feito");}});Modo pausadoO outro modo para fluxos legíveis é "modo pausado".Este é oModo que os fluxos começam. Se você nunca registrar um manipulador de eventos de "dados"e nunca chame o método Pipe (), então um fluxo legível permanece emmodo pausado.No modo pausado, o fluxo não empurra dados para você ema forma de eventos de "dados".Em vez disso, você puxa dados do fluxo porchamando explicitamente o método read ().Esta não é uma chamada de bloqueio e seNão há dados disponíveis para leitura no fluxo, eles retornarão NULL.Como não há uma API síncrona para esperar pelos dados, o modo pausadoA API também é baseada em eventos.Um fluxo legível no modo pausado emiteEventos "legíveis" quando os dados ficam disponíveis para ler no fluxo.Em resposta, seu código deve chamar o método read () para ler quedados.Você deve fazer isso em um loop, chamando read () repetidamente atéretorna nulo.É necessário drenar completamente o buffer do fluxoAssim para acionar um novo evento "legível" no futuro.Se vocêpare de chamar read () Embora ainda haja dados legíveis, você não receberáOutro evento ?legível? e seu programa provavelmente pendurarão.Fluxos no modo pausado emitem eventos "end" e "error", assim comofluxos de modo de fluxo fazem.Se você está escrevendo um programa que lê dadosDe um fluxo legível e o escreve para um fluxo gravável, depois parado

O modo pode não ser uma boa escolha. Para manusear corretamente Backpressure, você só quer ler quando o fluxo de entrada é legível e o fluxo de saída não é backup. No modo pausado, isso significa lendo e escrevendo até read() retornar null ou write() retorna false, e depois começando a ler ou escrever novamente em um legível ou Evento de drenagem. Isso é desleixante, e você pode achar que o modo de fluxo (ou tubos) é mais fácil neste caso. O código a seguir demonstra como você pode calcular um sha256hash para o conteúdo de um arquivo especificado. Ele usa um fluxo legível em modo pausado para ler o conteúdo de um arquivo em pedaços e depois passa cada Shunk para o objeto que calcula o hash. (Observe que no nº 12 mais tarde, seria mais simples escrever esta função usando um para/aguardar laço.)

```
const fs = require("fs");
const crypto = require("crypto");

function sha256(filename) {
    const input = fs.createReadStream(filename);
    const hasher = crypto.createHash("sha256");
    let chunk;
    input.on("data", () => {
        if (chunk) hasher.update(chunk);
        chunk = Buffer.alloc(1024);
        input.read(chunk);
    });
    input.on("end", () => {
        console.log(`SHA256 of ${filename} is ${hasher.digest("hex")}`);
    });
}
```

```
} // e continue em loopaté não ser legível});input.on ("end", () => {// no final do fluxo, deixe hash = hashher.digest  
("hexadecimal");// Calcule o hash, retorno de chamada (nulo, hash);// e passe para o retorno de chamada.});input.on  
("erro", retorno de chamada);// em erro, ligue ligar de volta}// Aqui está um utilitário simples da linha de comando  
para calcular o hash de um arquivo sha256 (process.argv [2], (err, hash) => {// Passe o nome do arquivo da linha de  
comando.se (err) {// se conseguirmos um erro no console.error (err.toString ());// imprima -o como erro.} else {// caso  
contrário,console.log (hash);// Imprima o hash corda.}});16.6 Processo, CPU e sistema operacionalDetalhesO objeto  
de processo global possui várias propriedades úteis e funções que geralmente se relacionam com o estado do nó  
atualmente em execução do processo. Consulte a documentação do nó para obter detalhes completos, mas aqui só  
algumas propriedades e funções que você deve estar ciente:process.argv // uma variedade de linha de  
comando argumentos.process.arch // a arquitetura da CPU: "x64", para
```

exemplo.process.cwd () // retorna o funcionamento atualdiretório.process.chdir () // define o funcionamento atualdiretório.process.cpuusage () // relata o uso da CPU.process.env // um objeto de ambientevariáveis.process.ExecPath // O caminho do sistema de arquivos absoluto para o nó executável.process.exit () // encerra o programa.process.exitcode // Um ??código inteiro a ser relatadoQuando o programa sai.Process.GetTid () // Retornar o ID do usuário do UNIX dousuário atual.process.hrtime.bigint () // retorna uma "alta resolução" Nanosegund Timestamp.process.kill () // Envie um sinal para outro processo.process.memoryusage () // retorna um objeto com uso de memória detalhes.process.NextTick () // como setImmediate (), invocar uma função em breve.process.pid // O ID do processo da corrente processo.process.ppid // O ID do processo pai.Process.platform // OS: "Linux", "Darwin" ou "Win32", por exemplo.process.ResourceUSAGE () // Retornar um objeto com recursoDetalhes de uso.process.setUid () // define o usuário atual, por id ownname.process.Título // o nome do processo que aparece em `PS` listagens.process.umask () // defina ou retorne o padrãoPermissões para novos arquivos.process.uptime () // retorna o tempo de atividade do nó em segundos.Process.Version // String de versão do Node.Process.versions // Strings de versão para as bibliotecasO nó depende.O módulo "OS" (que, diferentemente do processo, precisa ser explicitamente

Carregado com requer () fornece acesso a nível semelhanteDetalhes sobre o computador e o sistema operacional que o nó está em execuçõesobre.Você pode nunca precisar usar nenhum desses recursos, mas vale a penaSaber que o nó os disponibiliza:const os = requer ("os");os.arch () // retorna arquitetura da CPU."x64" ou"Arm", por exemplo.OS.CONSTANTS // Constantes úteis, comoOS.CONSTANTS.Signals.SIGINT.os.cpus () // dados sobre núcleos de CPU do sistema,incluindo tempos de uso.os.endianness () // o nativo endianness da CPU "be" ou"Le".OS.EOL // O Terminador de Linha Nativa do OS: "\ n"ou "\ r \ n".os.freemem () // retorna a quantidade de RAM livre embytes.OS.GetPriority () // Retorna a prioridade de agendamento do sistema operacionalde um processo.os.homedir () // retorna a casa do usuário atualdiretório.os.hostname () // retorna o nome do host docomputador.os.loadavg () // retorna os 1, 5 e 15 minutosMédias de carga.OS.NetworkInterfaces () // Retorna detalhes sobre o disponívelrede.conexões.os.platform () // retorna os: "linux", "darwin" ou"Win32", por exemplo.os.release () // retorna o número da versão doOS.os.setPriority () // tenta definir o agendamentoprioridade para um processo.os.tmpdir () // retorna o padrão temporáriodiretório.OS.Totalmem () // retorna a quantidade total de RAM embytes.os.type () // retorna os: "linux", "darwin" ou"Windows_nt", por exemplo

Erro ao traduzir esta página.

descriptor ?como o primeiro argumento em vez de um caminho.Essas variantes têmNomes que começam com a letra "f".Por exemplo, fs.truncate ()trunca um arquivo especificado por caminho, e fs.ftruncate () truncam um arquivo especificado pelo descriptor de arquivo.Há uma promessa baseadafs.promises.truncate () que espera um caminho e outroVersão baseada em promessa que é implementada como um método de umObjeto FileHandle.(A classe FileHandle é o equivalente a um arquivodescriptor na API baseada em promessa.) Finalmente, há um punhado defunções no módulo "FS" que têm variantes cujos nomes sãoprefixado com a letra "l".Essas variantes "L" são como a função basemas não siga links simbólicos no sistema de arquivos e operediretamente nos próprios ligações simbólicas.16.7.1 Caminhos, descritores de arquivos e trabalhos de arquivoPara usar o módulo "FS" para trabalhar com arquivos, você primeiro precisa sercapaz de nomear o arquivo com o qual você deseja trabalhar.Os arquivos são mais frequentementeespecificado por caminho, o que significa o nome do próprio arquivo, além doHierarquia de diretórios nos quais o arquivo aparece.Se um caminho é absoluto,Isso significa que os diretórios até a raiz do sistema de arquivos estãoespecificado.Caso contrário, o caminho é relativo e só é significativo emrelação com algum outro caminho, geralmente o diretório de trabalho atual.Trabalhar com caminhos pode ser um pouco complicado, porque a operação diferenteOs sistemas usam caracteres diferentes para separar nomes de diretórios, é fácilpara dobrar accidentalmente esses caracteres separadores ao concatenarcaminhos e porque ../ segmentos de caminho do diretório pai precisam de especialmanuseio.Módulo "Path" do Node e alguns outros nó importantesAjuda dos recursos:

```
// Alguns caminhos importantes
process.cwd() // caminho absoluto do trabalho atual
dirname // caminho absoluto do arquivo que mantém o código atual.
filename // caminho absoluto do diretório que se segura
__filename.os.homedir() // o diretório inicial do usuário.
const caminho = requer ("caminho"); path.sep //, "/" ou
"\\" Dependendo do seu sistema operacional // O módulo de caminho tem funções de análise simples
Seja p =
"src/pkg/test.js"; // Um exemplo de caminho
Path.basename (p) // => "test.js"
path.extName (p) // =>
".js"
path.dirname (p) // => "src/pkg"
Path.basename (path.dirname (p)) // => "pkg"
path.dirname (path.dirname (p)) // => "src"
// normalize () limpa os caminhos: path.Normalize ("a/b/c ../../ d/") // => "a/b/d/"
manipula
.. / segmentos
path.Normalize ("a ./ b") // => "a/b": tira "."
.. / segmentos
path.Normalize ("// a // b //") // => "/a/b/":
removeduplicate // junção () combina segmentos de caminho, adicionando separadores e
depoisnormalize
path.join ("src", "pkg", "t.js") // => "src/pkg/t.js"
// resolve () leva um ou mais segmentos de caminho
e retorna um absoluto // caminho. Começa com o último argumento e funciona para trás, parando // quando construiu
um caminho absoluto ou resolvendo contra
process.cwd().path.resolve () // => process.cwd().path.resolve ("t.js")
// => Path.Join (process.cwd(), "t.js")
path.resolve ("/tmp", "t.js") // => "/tmp/t.js"
```

path.resolve ("/a", "/b", "t.js") // => "/b/t.js" Observe que Path.Normalize () é simplesmente uma manipulação de stringfunção que não tem acesso ao sistema de arquivos real.OFunções Fs.RealPath () e F.RealPathSync () executamCanonicalização consciente do sistema de arquivos: eles resolvem links simbólicos eInterprete os nomes relativos de caminho em relação ao diretório de trabalho atual.Nos exemplos anteriores, assumimos que o código está em execução em umOS e Path.Sep baseados em UNIX são "/".Se você quiser trabalhar com Unix-Caminhos de estilo mesmo quando em um sistema Windows, depois use Path.Posixem vez de caminho.E inversamente, se você quiser trabalhar com o WindowsCaminhos mesmo quando em um sistema Unix, Path.Win32.Path.Posix ePath.win32 Defina as mesmas propriedades e funções que o próprio caminho.Algumas das funções "fs" que abordaremos nas próximas seçõesEspere um descritor de arquivo em vez de um nome de arquivo.Os descritores de arquivos sãoOs números inteiros usados ??como referências no nível do SO aos arquivos "abertos".Você obtém umdescritor para um determinado nome chamando o fs.open () (oufunção fs.opensync ()).Os processos só podem ter umNúmero limitado de arquivos abertos ao mesmo tempo, por isso é importante que você liguefs.close () nos descritores de arquivos quando você terminar com eles.Você precisa abrir arquivos se quiser usar o nível mais baixoFunções fs.read () e fs.write () que permitem que você puleEm torno de um arquivo, lendo e escrevendo bits em momentos diferentes.Existem outras funções no módulo "FS" que usam descritores de arquivos,Mas todos eles têm versões baseadas em nomes, e isso só faz sentidoPara usar as funções baseadas em descritores se você fosse abrir o arquivo

para ler ou escrever de qualquer maneira.Finalmente, na API baseada em promessa definida pelo FSS.equivalente a fs.open () é fs.promises.open (), queRetorna uma promessa que resolve para um objeto FileHandle.Este arquivo de arquivoO objeto serve ao mesmo objetivo que um descritor de arquivo.Novamente, no entanto,A menos que você precise usar o nível mais baixo read () e write ()Métodos de um arquivo de arquivo, realmente não há razão para criar um.E seVocê cria um arquivo de arquivo, lembre -se de ligar para o seu fechamento ()método depois de terminar com isso.16.7.2 Leitura de arquivosO nó permite ler o conteúdo do arquivo de uma só vez, através de um fluxo ou comA API de baixo nível.Se seus arquivos forem pequenos ou se o uso e o desempenho da memória não forem omaior prioridade, então é muitas vezes mais fácil ler todo o conteúdo de umArquivo com uma única chamada.Você pode fazer isso de maneira síncrona, com um retorno de chamada,ou com uma promessa.Por padrão, você receberá os bytes do arquivo como umBuffer, mas se você especificar uma codificação, receberá uma corda decodificadaem vez de.const fs = requer ("fs");deixe buffer = fs.readfilesync ("test.data");//Síncrono, retorna bufferdeixe texto = fs.readfilesync ("data.csv", "utf8");//Síncrono, Retorna String// Leia os bytes do arquivo de forma assíncronafs.readfile ("test.data", (err, buffer) => {if (err) {

```
// lide com o erro aqui} outro {// Os bytes do arquivo estão em buffer});// Leia assíncrona baseada em
promessafs.promises.readFile ("data.csv", "utf8").TENHEN (ProcessfileText).catch (Handlereaderror);// ou use a
API de promessa com aguardar dentro de uma função assíncronaFunção assíncrona ProcessText (nome do
arquivo, coding = "utf8") {Deixe o texto = aguarda fs.promises.readFile (nome do arquivo,codificação);// ... Processe
o texto aqui ...}Se você é capaz de processar o conteúdo de um arquivo sequencialmente e não precisa ter todo o
conteúdo do arquivo na memória ao mesmo tempo,Em seguida, ler um arquivo por meio de um fluxo pode ser a
abordagem mais eficiente.Cobrimos riachos extensivamente: aqui está como você pode usar umstream e o
método Pipe () para escrever o conteúdo de um arquivo parasaída padrão:função printfile (nome do arquivo,
coding = "utf8") {fs.creteReadStream (nome do arquivo,codificação) .pipe (process.stdout);}Finalmente, se você
precisa de controle de baixo nível sobre exatamente quais bytes você lêDe um arquivo e quando você os lê, você
pode abrir um arquivo para obter um arquivodescriptor e depois use fs.read (), fs.readsync () ou fs.promises.read ()
para ler um número especificado de bytes de um
```

Localização da fonte especificada do arquivo em um buffer especificado noPosição de destino especificada:const fs = requer ("fs");// lendo uma parte específica de um arquivo de dadosfs.open ("dados", (err, fd) => {if (err) {}// Relatório Erro de alguma formaretornar;}tentar {// leia bytes 20 a 420 em um recém -alocadobuffer.fs.read (fd, buffer.alloc (400), 0, 400, 20, (err, n,b) => {// err é o erro, se houver// n é o número de bytes realmente lido// b é o buffer que eles foram lidosem.});}finalmente {// use uma cláusula finalmentefs.close (FD);// Fechar o descriptor de arquivo aberto}});A API read () baseada em retorno de chamada é estranha de usar se você precisar lerMais de um pedaço de dados de um arquivo.Se você pode usar oAPI síncrona (ou a API baseada em promessa com aguardar), torna-seFácil de ler vários pedaços de um arquivo:const fs = requer ("fs");função readData (nome do arquivo) {Seja fd = fs.opensync (nome do arquivo);tentar {// Leia o cabeçalho do arquivo

```
deixe o cabeçalho = buffer.Alloc (12); // Um ??buffer de 12 bytes
fs.readsync (FD, cabeçalho, 0, 12, 0); // Verifique o
número mágico do arquivo
Deixe Magic = Header.readInt32LE (0);
if (mágica! == 0xdadadafeed) {lançar um novo erro
("o arquivo é do tipo errado");}
// Agora obtenha o deslocamento e o comprimento dos dados
docabeçalhosoltar
offset = header.readInt32LE (4);
deixe comprimento = cabeçalho.readInt32LE (8); // e leia esses bytes do
arquivodeixe dados = buffer.Alloc (comprimento);
fs.readsync (fd, dados, 0, comprimento, deslocamento);
retornar
dados; } finalmente { // sempre feche o arquivo, mesmo que uma exceção seja jogado acima
fs.closesync (FD); }}
```

16.7.3 Escrevendo arquivos

Escrever arquivos no nó é como lê -los, com alguns detalhes extras que você precisa saber.

Um desses detalhes é que a maneira como você Criar um novo arquivo é simplesmente escrevendo para um nome de arquivo que não já existem.

Como na leitura, existem três maneiras básicas de escrever arquivos no nó.

Se Você tem todo o conteúdo do arquivo em uma string ou buffer, você pode Escreva a coisa toda em uma chamada com `fs.writeFile ()` (retorno de chamada-baseado), `fs.writeFileSync ()` (síncrono), ou

fs.promises.writeFile () (baseado em promessa):fs.writeFileSync (path.resolve (__ Dirname, "Settings.json"),Json.Stringify (Configurações));Se os dados que você está escrevendo no arquivo é uma string e você deseja usar umcodificação diferente de "utf8", passe a codificação como um terceiro opcionalargumento.As funções relacionadas fs.appendfile (),fs.appendFileSync () e fs.promises.appendfile ()são semelhantes, mas quando o arquivo especificado já existe, eles anexam seusdados até o final, em vez de substituir o conteúdo do arquivo existente.Se os dados que você deseja escrever em um arquivo não for tudo em um pedaço, ou se forNem todos na memória ao mesmo tempo, então usar um fluxo gravável é umboa abordagem, assumindo que você planeja escrever os dados deComeçando a terminar sem pular no arquivo:const fs = requer ("fs");deixe output = fs.createWriteStream ("números.txt");para (vamos i = 0; i <100; i ++) {output.Write (`\$ {i} \ n`);}output.end ();Finalmente, se você deseja escrever dados em um arquivo em vários pedaços, e vocêdeseja poder controlar a posição exata dentro do arquivo em queCada pedaço é escrito, então você pode abrir o arquivo com fs.open (),fs.opensync () ou fs.promises.open () e depois use oDescriptor de arquivo resultante com o fs.write () ou

funções `fs.writeFileSync()`. Essas funções vêm em diferentes formas para cordas e buffers. A variante da string leva um descriptor de arquivo, uma string e a posição do arquivo para escrever essa string (com uma codificação como um quarto argumento opcional). A variante de buffer leva um descriptor de arquivo, um buffer, um deslocamento e um comprimento que especificam um pedaço de dados dentro do buffer e uma posição de arquivo para escrever os bytes de aquele pedaço. E se você tiver uma variedade de objetos buffers que deseja escrever, você pode fazer isso com um único `fs.writeFileSync()`. Existem funções de baixo nível semelhantes para escrever buffers e strings usando `fs.promises.open()` e o objeto `FileHandle` que produz strings de modo de arquivo. Vimos os métodos `fs.open()` e `fs.openSync()` antes de usar a API de baixo nível para ler arquivos. Nesse caso de uso, foi suficiente passar o nome do arquivo para a função aberta. Quando você quiser escrever um arquivo, no entanto, você também deve especificar um segundo argumento de string que especifica como você pretende usar o descriptor de arquivo. Algumas das strings de bandeira disponíveis são as seguintes: "c" Abra o arquivo para escrever; "w+" Aberto para escrever e ler; "wx" Aberto para criar um novo arquivo; falha se o arquivo nomeado já existir; "wx+" Aberto para criação e também permitir a leitura; falha se o arquivo nomeado já existir; "um" Abra o arquivo para anexar; O conteúdo existente não será substituído.

"A+" Aberto para anexar, mas também permita a leituraSe você não passa uma dessas seqüências de bandeira para fs.open () ou fs.opensync (), eles usam o padrãoSinalizador ?R?, tornando o descritor de arquivo somente leitura. Observe que também pode ser útil passar essasbandeiras para outroMétodos de redação de arquivos:// escreva em um arquivo em uma chamada, mas anexar qualquer coisa que já estejalá// Isso funciona como fs.appendfilesync ()fs.writefilesync ("messages.log", "hello", {flag: "a"}); // Abra um fluxo de gravação, mas faça um erro se o arquivo já existir// Não queremos substituir accidentalmente algo!// Observe que a opção acima é "sinalizador" e é "sinalizadores" aqui fs.createwritestream ("mensagens.log", {sinalizadores: "wx"}); Você pode cortar o final de um arquivo com fs.truncate (), fs.truncatesync (), ou fs.promises.truncate (). EssesAs funções seguem um caminho como seu primeiro argumento e um comprimento como seusegundo e modifique o arquivo para que ele tenha o comprimento especificado. Se vocêOmita o comprimento, zero é usado e o arquivo fica vazio. Apesar do Nome dessas funções, eles também podem ser usados ??para estender um arquivo: se vocêEspecifique um comprimento maior que o tamanho atual do arquivo, o arquivo éestendido com zero bytes ao novo tamanho. Se você já abriuO arquivo que você deseja modificar, você pode usar ftruncate () ou ftruncatesync () com o descritor de arquivo ou o FileHandle. As várias funções de redação de arquivos descritas aqui retornam ou invocam seus retorno de chamada ou resolva sua promessa quando os dados foram "escritos" em A sensação de que o nó o entregou ao sistema operacional. Mas isso não significa necessariamente que os dados foram realmente escritos para armazenamento persistente ainda: pelo menos alguns de seus dados ainda podem ser bobs em algum lugar do sistema operacional ou em um driver de dispositivo esperando para estar

Erro ao traduzir esta página.

// Este retorno de chamada será chamado quando terminar. Em erro, err será não nulo.}); // Este código demonstra a versão baseada em promessa da função copyfile. // Dois sinalizadores são combinados com o bit a bit ou o OPERADOR |. O bandeiras significam isso. // Os arquivos existentes não serão substituídos e que se o FileSystem suporta // isso, a cópia será um clone de cópia em redação do original arquivo, significado // que nenhum espaço de armazenamento adicional será necessário até ou o original // ou a cópia é modificada. fs.promises.copyFile ("Dados importantes", `Dados importantes \${novoDate ()}. ToisSotring ()} "fs.constants.copyfile_excl |fs.constants.copyfile_ficclone).then (() => {console.log ("backup completo");}); .catch (err => {console.error ("backup falhou", err);}); A função fs.rename () (junto com o habitual síncrono e Variantes baseadas em promessas) move e / ou renomeia um arquivo. Chame com o Caminho atual para o arquivo e o novo caminho desejado para o arquivo. Não há Argumento de sinalizadores, mas a versão baseada em retorno de chamada leva um retorno de chamada como o Terceiro argumento: fs.renameSync ("CH15.Bak", "Backups/CH15.Bak"); Observe que não há bandeira para impedir a renomeação de substituir um arquivo existente. Lembre -se também de que os arquivos só podem ser renomeados em um FileSystem.

As funções `fs.link()` e `fs.symlink()` e suas variantes tem as mesmas assinaturas que `fs.rename()` e se comportam como `fs.copyfile()`, exceto que eles criam links rígidos e simbólicos links, respectivamente, em vez de criar uma cópia. Finalmente, `fs.unlink()`, `fs.unlinkSync()` e `fs.promises.unlink()` são funções do nó para excluir um arquivo. (A nomeação não intuitiva é herdada do Unix, onde a exclusão de um arquivo é basicamente o oposto de criar um link difícil para ele.) Chame essa função com a string, buffer ou caminho de URL para o arquivo a ser excluído e passar um retorno de chamada se você estiver usando a versão baseada em retorno de chamada:

```
const fs = require('fs');
let stats = fs.statSync('livro/CH15.md');
if(stats.isFile() // => true: este é um arquivo
  || stats.isDirectory() // => false: não é um diretório
) {
  console.log(`Tamanho do arquivo em bytes: ${stats.size}`);
  console.log(`Tempo de acesso: ${stats.atime}`);
  console.log(`Tempo de modificação: ${stats.mtime}`);
  console.log(`ID do usuário do proprietário do arquivo: ${stats.uid}`);
  console.log(`ID do grupo do proprietário do arquivo: ${stats.gid}`);
  console.log(`Permissões do arquivo, como um octal: ${stats.mode.toString(8)}`);
}
```

O objeto de estatísticas retornadas contém outras propriedades mais obscuras emétodos, mas este código demonstra aqueles que você provavelmente deveusar.fs.Istat () e suas variantes funcionam como fs.stat (), exceto queSe o arquivo especificado for um link simbólico, o nó retornará metadados para oLink em si em vez de seguir o link.Se você abriu um arquivo para produzir um descritor de arquivo ou um arquivo de arquivoObjeto, então você pode usar fs.fstat () ou suas variantes para obter metadadosinformações para o arquivo aberto sem ter que especificar o nome do arquivode novo.Além de consultar metadados com fs.stat () e todos os seusVariantes, também existem funções para a mudança de metadados.fs.chmod (), fs.lchmod () e fs.fchmod () (junto conversões síncronas e baseadas em promessas) defina o "modo" oupermissões de um arquivo ou diretório.Os valores de modo são inteiros nos quaisCada bit tem um significado específico e é mais fácil de pensar em octalnotação.Por exemplo, para fazer um arquivo somente leitura para seu proprietário elnecessível a todos os outros, use 0o400:fs.chmodsync ("CH15.MD", 0O400);// Não excluaaccidentalmente!fs.chown (), fs.lchown () e fs.fchown () (junto conversões síncronas e baseadas em promessas) defina o proprietário e o grupo (comoIDs) para um arquivo ou diretório.(Estes são importantes porque eles interagem com oPermissões de arquivo definidas por fs.chmod (.)

Por fim, você pode definir o tempo de acesso e o tempo de modificação de um arquivo ou diretório com `fs.utimes()` e `fs.futimes()` e seus variantes.

16.7.6 Trabalhando com diretórios

Para criar um novo diretório no nó, use `fs.mkdir()`, `fs.mkdirsSync()`, ou `fs.promises.mkdir()`. O primeiro argumento é o caminho do diretório a ser criado. O segundo argumento opcional pode ser um número inteiro que especifica o modo (bits de permissões) para o novo diretório. Ou você pode passar um objeto com modo opcional e propriedades recursivas. Se recursivo é verdadeiro, então isso criará qualquer diretório no caminho que ainda não existe:// Verifique se existem dist/ e dist/ lib/ ambos.

```
fs.mkdirsSync ("dist/lib", {recursive: true});
```

`fs.mkdtemp()` e suas variantes tomam um prefixo de caminho que você fornece, anexar alguns caracteres aleatórios a ele (isso é importante para a segurança). Crie um diretório com esse nome e retorne (ou passe para um retorno de chamada) o caminho do diretório para você. Para excluir um diretório, use `fs.rmdir()` ou uma de suas variantes. Observação que os diretórios devem estar vazios antes que possam ser excluídos:// Crie um diretório temporário aleatório e faça seu caminho, então// Exclua quando terminarmos

```
Deixe tempdirpath; tentar {tempdirpath = fs.mkdtempSync (path.join (os.tmpdir (), "D"))};
```

```
// Faça algo com o diretório aqui} finalmente // exclua o diretório temporário quando terminarfs.rmdirsync  
(tempdirpath);}O módulo "FS" fornece duas APIs distintas para listar o conteúdo de um diretório.Primeiro, fs.readdir()  
, fs.readdirSync () efs.promises.readdir () Leia o diretório inteiro de uma só vez e lhe fornece uma variedade de  
cordas ou uma variedade de objetos diretos que especificamos nomes e tipos (arquivo ou diretório) de cada  
item.Nomes de arquivos retornadosPor essas funções, são apenas o nome local do arquivo, não o caminho  
inteiro.Aqui estão exemplos:deixe tempfiles = fs.readdirSync ("/tmp");// retorna uma matriz de cordas// Use a API  
baseada em promessa para obter uma matriz direta e depois// Imprima os caminhos dos  
subdiretórios.fs.promises.readdir ("/tmp", {withfiletypes: true}).then (entradas => {entradas.Filter (Entrada =>  
Entrada.isDirectory ()).Map (entrada => Entry.name).ForEach (nome => console.log (path.join  
("/tmp/", nome));}).catch (console.error);Se você prever a necessidade de listar diretórios que podem ter  
milhares de entradas, você pode preferir a abordagem de streaming def.s opendir () e suas variantes.Essas funções  
retornam um objeto Dir representando o diretório especificado.Você pode usar o read () ou métodos ReadSync () do  
objeto Dir para ler um Dirent de cada vez.Se você passar por uma função de retorno de chamada para ler (), ela  
chamará o retorno de chamada.
```

E se você omitir o argumento de retorno de chamada, ele retornará uma promessa. Quando não há mais entradas de diretório, você ficará nulo em vez de um objeto. A maneira mais fácil de usar objetos dir é como iteradores assíncronos com um para/aguarda loop. Aqui, por exemplo, é uma função que usa o API de streaming para listar entradas de diretório, chama Stat () em cada entrada, e impressões de nomes e tamanhos de arquivo e diretórios:

```
const fs = requer ("fs"); const caminho = requer ("caminho");
Função ASYNC ListDirectory (Dirpath) {deixe dir para aguardar (deixe a entrada de dir) {Deixe o nome = entrada.name;if (Entry.isDirectory ()) {nome += "/"; Adicione uma barra à direita a subdiretórios} Deixe estatísticas = aguarda fs.promises.stat (path.join (dirpath, nome)); deixa tamanho = estatismo.size; console.log (string (tamanho) .padstart (10), nome);}}
```

16.8 clientes e servidores HTTPS módulos "http", https "e" http2 "são completos, mas implementações de nível relativamente baixo dos protocolos HTTP. Eles definem APIs abrangentes para implementar clientes HTTP e servidores. Porque as APIs são relativamente baixas, não há espaço em

Este capítulo para cobrir todos os recursos. Mas os exemplos que se seguem demonstram como escrever clientes e servidores básicos. A maneira mais simples de fazer um http básico solicitar é com `http.get()` ou `https.get()`. O primeiro argumento para estes funções é o URL a buscar. (Se for um `http:// url`, você deve usar o módulo "http" e, se for um `https:// url`, você deve usar o módulo `https`.) O segundo argumento é um retorno de chamada que será invocado com um objeto de entrada de entrada quando a resposta do servidor chegar. Quando o retorno de chamada é chamado, o status HTTP e os cabeçalhos estão disponíveis, mas o corpo ainda não está pronto. O objeto de entrada de Message é um fluxo legível e você pode usar técnicas demonstradas anteriormente neste capítulo para ler a resposta no corpo dele. A função `getJSON()` no final do §13.2.6 usou `ohttp.get()` função como parte de uma demonstração da promessa (`constructor`). Agora que você sabe sobre fluxos de nó e o modelo de programação de maneira mais geral, vale a pena revisar esse exemplo para ver como `http.get()` é usado. `http.get()` e `https.get()` são variantes ligeiramente simplificadas de `http.request()` e `https.request()` funções. A função `PostJson()` a seguir demonstra como fazer uso de `https.request()` para fazer uma solicitação de post `https` que inclui um corpo de solicitação JSON. Como a função `getJSON()` no Capítulo 13, espera uma resposta JSON e retorna uma promessa de que a versão analisada dessa resposta:

```
const https = requer ("https");/* Converta o objeto corporal em uma corda JSON e depois https postpara o* Ponto de extremidade da API especificado no host especificado.Quando oA resposta chega,* analise o corpo de resposta como JSON e resolva o retornoPromessa com* que analisou o valor.*/função postjson (host, endpoint, corpo, porto, nome de usuário,senha) {// Retornar um objeto de promessa imediatamente e depois chame a resoluçaoou rejeitar// Quando a solicitação HTTPS é bem -sucedida ou falha.retornar nova promessa ((resolver, rejeitar) => {// converte o objeto do corpo em uma stringDeixe BodyText = JSON.Stringify (Body);// Configure a solicitação HTTPSDeixe requestOptions = {Método: "post", // ou "get", "put","Delete", etc.host: host, // o host para se conectararcaminho: endpoint, // o caminho da URLcabeçalhos: {// cabeçalhos http para osolicitar"Tipo de conteúdo": "Aplicativo/JSON", "Length-thength": buffer.byteLength (BodyText)};se (porta) {// se uma porta for especificado,requestOptions.port = porta;// use -o para osolicitar.}// Se as credenciais forem especificadas, adicione uma autorizaçãocabeçalho.if (nome de usuário && senha) {requestOptions.auth = `\$ {nome de usuário}: \$ {senha}`;}
```

```
// agora crie a solicitação com base na configuração do objeto deixa solicitação = https.request (requestOptions);//
Escreva o corpo da solicitação de postagem e termine os solicitar.request.Write (BodyText);request.end ()// falha
nos erros de solicitação (como nenhuma redeconexão)request.on ("erro", e => rejeitar (e));// lide com a resposta
quando começar a chegar.request.on ("resposta", resposta => {if (Response.statuscode! == 200) {rejeite (novo erro
(`status http$ {Response.statuscode} `));// Não nos importamos com o corpo de resposta em Este caso, mas// Não
queremos que fique em umBuffer em algum lugar, então// colocamos o fluxo no modo de fluxo sem se registrar// 
um manipulador de "dados" para que o corpo seja descartado.Response.Resume ();retornar;}// queremos texto,
não bytes.Estamos assumindo o texto será// JSON-Formatted, mas não está se preocupando em verificar// 
Cabeçalho de conteúdo-type.Response.setEncoding ("UTF8");// O nó não tem um analisador JSON de streaming,
então Lemos o// Todo o corpo de resposta em uma corda.Deixe Body = "";resposta.on ("dados", chunk => {body +=
chunk;});
```

// e agora lida com a resposta quando estiver completo.resposta.on ("end", () => {// quando oA resposta está feita,tente {// tentearanalisá -lo como JSONresolve (json.parse (corpo));// eResolva o resultado.} catch (e) {// ou, seTudo dá errado,rejeitar (e);// rejeitarcom o erro}});});});Além de fazer solicitações HTTP e HTTPS, o "http" eOs módulos "https" também permitem escrever servidores que respondem a elessolicitações.A abordagem básica é a seguinte:Crie um novo objeto de servidor.Ligue para o método Listen () para começar a ouvir solicitações emuma porta especificada.Registre um manipulador de eventos para eventos de "solicitação", use quemanipulador para ler a solicitação do cliente (particularmente oPropriedade do request.url) e escreva sua resposta.O código a seguir cria um servidor HTTP simples que serve estáticoarquivos do sistema de arquivos local e também implementa uma depuraçãoendpoint que responde à solicitação de um cliente ecoando essa solicitação// Este é um servidor http estático simples que serve arquivos deum especificado// diretório.Ele também implementa um especial /teste /espelho

```
endpoint isto// ecoa a solicitação de entrada, que pode ser útil quandodepurar clientes.const http = requer ("http");//  
Use "https" se você tiver umCertificadoconst url = requer ("url");// para analisar URLsconst caminho = requer  
("caminho");// para manipularCaminhos do sistema de arquivosconst fs = requer ("fs");// Para leitura de arquivos//  
serve arquivos do diretório raiz especificado por meio de um httpservidor isso// escuta na porta especificada.função  
servir (rootdirectory, porta) {Deixe servidor = novo http.server ()// Crie um novo httpservidorServer.Listen (porta);//  
Ouça noporta especificadaconsole.log ("escuta na porta", porta);// Quando os pedidos chegarem, lide com esta  
funçãoServer.on ("Solicitação", (solicitação, resposta) => {// Obtenha a parte do caminho do URL da solicitação,  
ignorando// Quaisquer parâmetros de consulta que sejam anexados a ele.deixe o endpoint = url.parse (request.url)  
.pathname;// Se a solicitação foi para "/teste/espeelho", envie de volta o pedido// literalmente.Útil quando você  
precisa ver o pedidoCabeçalhos e corpo.if (endpoint === "/teste/espeelho") {// Definir cabeçalho de  
respostaResponse.setheader ("Content-Type", "Text/Plain;charset = utf-8 ");// Especifique o código de status da  
respostaResponse.Writehead (200);// 200 ok// Comece o corpo de resposta com o pedidoResponse.Write ('$  
{request.method} $ {request.url}Http/$ {request.httpversion} \ r \ n`);
```

Erro ao traduzir esta página.

deixe Stream = fs.createReadStream (nome do arquivo);stream.once ("readável", () => {// Se o fluxo ficar legível, então defina// Content-Type Cabeçalho e um status de 200 OK.Em seguida, pague o// Stream do leitor de arquivos para a resposta.OWill de tubo// Ligue automaticamente a resposta.end () quando oTerminos de fluxo.Response.setheader ("Tipo de conteúdo", tipo);Response.writeHead (200);stream.pipe (resposta);});stream.on ("erro", (err) => {// em vez disso, se recebermos um erro tentando abriro fluxo// então o arquivo provavelmente não existe ou não é legível// Envie uma resposta 404 não encontrada em texto simples com o// Mensagem de erro.Response.setheader ("Content-Type", "Texto/simples; charset = utf-8");Response.writeHead (404);resposta.END (Err.Message);});});};// Quando somos invocados da linha de comando, ligue para o saque ()funçãoservir (process.argv [2] || "/tmp", parseint (process.argv [3]) ||8000);Os módulos embutidos do Node são tudo o que você precisa para escrever HTTP simples e Servidores HTTPS.Observe, no entanto, que os servidores de produção não são normalmente construído diretamente sobre esses módulos.Em vez disso, mais não trivialOs servidores são implementados usando bibliotecas externas - como o expresso

estrutura-que fornecem ?middleware? e outros utilitários de nível superiorOs desenvolvedores da Web de back-end esperavam.16.9 Servidores de rede não-HTTP e ClientesServidores e clientes da web se tornaram tão onipresentes que é fácil que é possível escrever clientes e servidores que não usamHttp.Mesmo que o nó tenha uma reputação como um bom ambiente paraEscrevendo servidores da web, o Node também tem suporte total para escrever outros tipos de servidores de rede e clientes.Se você se sentir confortável trabalhando com riachos, a rede é relativamente simples, porque os soquetes de rede são simplesmente um tipo de duplexfluxo.O módulo "Net" define classes de servidor e soquete.Para criar um servidor, ligue para `net.createServer ()`, depois ligue para a escuta ()método do objeto resultante para dizer ao servidor em que porta ouvir para conexões.O objeto do servidor gerará eventos de "conexão"Quando um cliente se conecta nessa porta, e o valor passou para o eventoO ouvinte será um objeto de soquete.O objeto de soquete é um fluxo duplex,e você pode usá-lo para ler dados do cliente e escrever dados para o cliente.Ligue para `end ()` no soquete para desconectar.Escrever um cliente é ainda mais fácil: passe um número de porta e nome de host para `net.createConnection ()` para criar um soquete para comunicar com qualquer servidor estiver em execução nesse host e ouvindo nessa porta.Em seguida, use esse soquete para ler e gravar dados de e para o servidor.O código a seguir demonstra como escrever um servidor com a "rede"

módulo. Quando o cliente se conecta, o servidor conta uma piada de knock-knock:// Um ??servidor TCP que entrega piadas interativas de knock-knockna porta 6789.// (por que seis tem medo de sete? Porque sete comeu nove!)const net = requer ("net");const line = requer ("readline");// Crie um objeto de servidor e comece a ouvir conexõesDeixe o servidor = net.createServer ();Server.Listen (6789, () => console.log ("Entregando risadas emporta 6789 "));// Quando um cliente se conectar, diga-lhes uma piada de knock-knock.Server.on ("conexão", soquete => {Telljoke (soquete).Then (() => Socket.end ()) // Quando a piada é feita,Feche o soquete..catch ((err) => {console.error (err); // registre todos os erros queocorrer,soquete.end ()// mas ainda fechar oSocket!});});// Estas são todas as piadas que conhecemos.const piadas = {"Boo": "Não chore ... é apenas uma piada!", "Alface": "Vamos entrar! Está congelando aqui!", "Uma velha senhora": "Uau, eu não sabia que você poderiaYodel! "};// realiza interativamente uma piada de knock sobre este soquete,sem bloquear.função assíncrona Telljoke (soquete) {}// Escolha uma das piadas aleatoriamenteLet RandomElement = A => a [Math.floor (Math.random () *A.Length)];Let Who = RandomElement (object.Keys (piadas));Deixe Punchline = piadas [quem];

```
// Use o módulo ReadLine para ler a entrada do usuário linha de cada vez.Let LineReader =
readLine.CreateInterface ({Entrada: soquete,saída: soquete,Prompt: ">>"});// uma função de utilidade para gerar
uma linha de texto para o cliente// e então (por padrão) exibe um prompt.saída de função (texto, prompt = true)
{Socket.WriteLine (`$ {text} \r \n`);if (prompt) lineReader.prompt ();}// As piadas de knock-knock têm uma estrutura de
chamada e resposta// Esperamos informações diferentes do usuário em diferentes estágios e// Tome medidas
diferentes quando tivermos essa entrada em estágios diferentes.deixe o estágio = 0;// Comece a piada de
knock-knock da maneira tradicional.saída ("Knock Knock!");// Agora leia as linhas de forma assíncrona do cliente
até a piada está feita.para aguardar (deixe o inputline do lineReader) {if (estágio === 0) {if (inputline.toLowerCase () 
== "Quem está aí?") {// se o usuário der a resposta certa em estágio 0// então diga a primeira parte da piada eva
para o estágio 1.saída (quem);estágio = 1;} outro //, caso contrário, ensine o usuário a fazer knock-bata
piadas.saída ('Por favor, digite "quem está aí?".');}
```

```
} else if (estágio === 1) {if (inputline.toLowerCase () ===`$ {who.toLowerCase ()} quem?`) {// se a resposta do usuário estiver correta no estágio1, então// entregar a linha de soco e retornar desdeA piada está pronta.saída ('$ {Punchline}', false);retornar;} outro {// Faça o usuário jogar junto.saída (' por favor digite" $ {quem} quem? ".');}}}}Servidores simples baseados em texto como esse normalmente não precisam de um personalizadocliente.Se o utilitário NC ("NetCat") estiver instalado no seu sistema, você podeUse -o para se comunicar com este servidor da seguinte forma:$ nc localhost 6789Knock!>> Quem está lá?Uma velha senhora>> Uma senhora velhinha que?Uau, eu não sabia que você poderia Yodel!Por outro lado, escrever um cliente personalizado para o servidor de piadas é fácil emNó.Acabamos de nos conectar ao servidor e, em seguida, transmitir a saída do servidor paraStdout e Pipe Stdin para a entrada do servidor:// Conecte -se à porta de piada (6789) no servidor nomeado nolinha de comandoDeixe soquete = requer ("net"). CreateConnection (6789,process.argv [2]);Socket.pipe (process.stdout);// Dados de tubulação deo soquete para stdoutprocess.stdin.pipe (soquete);// Dados de tubulação de
```

stdin para o soqueteSocket.on ("Close", () => process.exit ());// desistir quando oO soquete fecha.Além de suportar servidores baseados em TCP, o módulo "Net" do Node tambémsuporta a comunicação interprocessante sobre "soquetes de domínio unix" quesão identificados por um caminho do sistema de arquivos e não por um número da porta.Nós somosnão vai cobrir esse tipo de soquete neste capítulo, mas o nóA documentação tem detalhes.Outros recursos do nó que não temosEspaço a cobrir aqui inclui o módulo "dgram" para clientes baseados em UDPe servidores e o módulo "TLS" que é "net" como "https" é "http".As classes TLS.Server e Tls.tlSSocket permitem a criaçãode servidores TCP (como o servidor de piada de knock-knock) que usam SSL-Conexões criptografadas como os servidores HTTPS.16.10 Trabalhando com processos filhosAlém de escrever servidores altamente simultâneos, o Node também funciona bempara escrever scripts que executam outros programas.No nóO módulo ?Child_Process? define uma série de funções para executarOutros programas como processos infantis.Esta seção demonstra alguns dosessas funções, começando com o mais simples e se movendo para o maiscomplicado.16.10.1 Execsync () e ExecFilesync ()A maneira mais fácil de executar outro programa é comChild_process.execsync ().Esta função leva o comando para correr como seu primeiro argumento.Ele cria um processo infantil, executa uma concha nessaProcessar e usa o shell para executar o comando que você passou.Então isso

Bloqueia até que o comando (e o shell) saia. Se o comando sair com um erro, o ExecSync () lança uma exceção. De outra forma, ExecSync () retorna qualquer saída que o comando grava para o seu Stdout Stream. Por padrão, esse valor de retorno é um buffer, mas você pode especificar uma codificação em um segundo argumento opcional para obter uma string em vez de um buffer. Se o comando gravar qualquer saída para Stderr, essa saída é passada para o fluxo Stderr do processo pai. Por exemplo, se você está escrevendo um script e o desempenho não é uma preocupação, você pode usar child_process.execSync () para listar um diretório com um comando familiar de shell Unix, em vez de usar a função fs.readdirSync ():`const Child_process = require ("Child_Process");
Seja listing = child_process.execSync ("ls -l web/*.html", {codificação: "utf8"});`

O fato de invocar um shell completo do Unix significa que a string que você passa para ela pode incluir vários semicolons-separados comandos e pode aproveitar os recursos do shell, como o nome do arquivo Wildcards, tubos e redirecionamento de saída. Isso também significa que você deve ter cuidado para nunca passar um comando para execSync () se alguma parte desse comando é entrada do usuário ou vem de uma fonte não confiável semelhante. A sintaxe complexa dos comandos do shell pode ser facilmente subvertida. Permita que um invasor execute o código arbitrário. Se você não precisar dos recursos de uma concha, pode evitar a sobrecarga de iniciando um shell usando child_process.execFileSync (). Esta função executa um programa diretamente, sem invocar um shell. Mas como nenhuma concha está envolvida, não pode analisar uma linha de comando e você

deve passar o executável como o primeiro argumento e uma variedade deArgumentos da linha de comando como o segundo argumento:Seja listing = Child_process.execFileSync ("LS", ["-l", "Web/"],{codificação: "utf8"});Opções de processo da criançaExecsync () e muitas das outras funções Child_process têm um segundo ou terceiro opcionalArgumento que especifica detalhes adicionais sobre como o processo infantil deve ser executado.A codificaçãoA propriedade deste objeto foi usada anteriormente para especificar que gostaríamos que a saída do comandofosse entreguecomo uma string e não como um buffer.Outras propriedades importantes que você pode especificar incluem oA seguir (observe que nem todas as opções estão disponíveis para todas as funções do processo filho):A CWD especifica o diretório de trabalho do processo filho.Se você omitir isso, então a criançaProcesso herda o valor do processo.cwd ().ENV especifica as variáveis ??de ambiente às quais o processo filho terá acesso.PorPadrão, os processos filhos simplesmente herdam o processo.env, mas você pode especificar um objeto diferente se você quiser.entrada especifica uma string ou buffer de dados de entrada que devem ser usados ??como entrada padrão parao processo infantil.Esta opção está disponível apenas para as funções síncronas que nãodevolver um objeto de processo infantil.MaxBuffer especifica o número máximo de bytes de saída que serão coletados pelofunções executivas.(Não se aplica a Spawn () e Fork (), que usam fluxos.) Se uma criançaO processo produz mais saída do que isso, será morto e sairá com um erro.Shell especifica o caminho para um shell executável ou verdadeiro.Para o processo infantil funções queNormalmente executa um comando Shell, esta opção permite especificar qual shell usar.ParaFunções que normalmente não usam um shell, essa opção permite especificar que um shelldeve ser usado (definindo a propriedade como verdadeiro) ou para especificar exatamente qual shell usar.Tempo limite especifica o número máximo de milissegundos que o processo infantil deve serpermitido correr.Se não tivesse saído antes desse tempo, será morto, será morto e sairá comum erro.(Esta opção se aplica às funções executivas, mas não para spawn () ou fork ().)O UID especifica o ID do usuário (um número) sob o qual o programa deve ser executado.Se o paiO processo está em execução em uma conta privilegiada, ele pode usar esta opção para executar a criança comreduçãoprivilégios.16.10.2 EXEC () e EXECFILE ()As funções Execsync () e ExecFileSync () são, como seus

Nomes indicam, síncrono: eles bloqueiam e não retornam atéSaídas do processo infantil.Urar essas funções é como digitar Unixcomandos em uma janela de terminal: eles permitem que você execute uma sequência decomanda um de cada vez.Mas se você está escrevendo um programa que precisarealizar várias tarefas, e essas tarefas não dependem de cadaOutro de qualquer maneira, então você pode paralizá -los e correrVários comandos ao mesmo tempo.Você pode fazer isso com oFunções assíncronas child_process.exec () eChild_process.execfile ().exec () e execfile () são como suas variantes síncronas, excetoque eles retornam imediatamente com um objeto de processo infantil que representao processo infantil em execução e eles recebem um retorno de chamada de erro como seuargumento final.O retorno de chamada é invocado quando o processo infantil sair,E na verdade é chamado com três argumentos.O primeiro é o erro, sequalquer;Será nulo se o processo terminou normalmente.O segundoO argumento é a saída coletada que foi enviada para o padrão da criançafluxo de saída.E o terceiro argumento é qualquer saída que foi enviada para oFluxo de erro padrão da criança.O objeto de processo infantil retornado por EXEC () e Execfile ()permite que você encerre o processo filho e escreva dados para ele (queentão ele pode ler a partir de sua entrada padrão).Vamos abordar o processo infantil emMais detalhes quando discutirmos o Child_process.spawn ()função.Se você planeja executar vários processos infantis ao mesmo tempo, entãopode ser mais fácil usar a versão "promisificada" do EXEC () que

Retorna um objeto de promessa que, se o processo filho sair sem erro, resolve um objeto com propriedades STDOUT e STDERR. Aqui, para exemplo, é uma função que leva uma variedade de comandos de shell como seuEntrar e retornar uma promessa que resolve ao resultado de todos aqueles comandos:

```
const Child_process = require ("Child_Process");
const util = require ("util");
const Execp = util.promisify (Child_process.exec);
function parallelexec (comandos) { // Use a variedade de comandos para criar uma matriz de Promessas
  let Promises = Commands.Map (Command => Execp (Command,{codificação: "utf8"}));
  // retorna uma promessa que cumprirá uma matriz documprimento// valores de cada uma das promessas individuais.
  (Em vez de Retornando objetos// com propriedades stdout e stderr valor stdout.)
  return Promise.All (promessas).THEN (saídas => outputs.map (out => out.stdout));
}
module.exports = parallelexec;
```

16.10.3 Spawn () As várias funções executivas descritas até agora - tanto síncronas quanto assíncronas - são projetadas para serem usados ?? com processos infantis que executam rapidamente e não produz muita saída. Até o assíncrono EXEC () e EXECFILE () não são o seco: eles retornam o processoSaída em um único lote, somente após a saída do processo. A função Child_process.spawn () permite que você transmita

acesso à saída do processo infantil, enquanto o processo ainda está correndo. Ele também permite escrever dados para o processo filho (queverá esses dados como entrada em seu fluxo de entrada padrão): isso significa que é possível interagir dinamicamente com um processo infantil, enviando entradas com base na saída que ele gera. `Spawn()` não usa uma concha por padrão, então você deve invocar `comoexecfile()` com o executável a ser executado e uma variedade separada de argumentos da linha de comando para passar para ele. `Spawn()` retorna o objeto de processo infantil como `execfile()`, mas não leva um argumento de retorno de chamada. Em vez de usar uma função de retorno de chamada, você ouve eventos no objeto de processo infantil e em seus fluxos. O objeto de processo infantil retornado por `Spawn()` é um emissor de eventos. Você pode ouvir o evento de `?saída?` para ser notificado quando o processo infantil saídas. Um objeto de processo infantil também possui três propriedades de fluxo: `stdout`, `stderr` e `stdin`. São fluxos legíveis: quando o processo infantil grava para o seu `stdout` e seus fluxos `stderr`, essa saída se torna legível através dos fluxos de processo infantil. Observe a inversão dos nomes aqui. No processo infantil, `?stdout?` é um fluxo de saída gravável, mas no pai processo, a propriedade `STDOUT` de um objeto de processo infantil é um fluxo legível de entrada. Da mesma forma, a propriedade `stdin` do objeto de processo infantil é um stream escritos: qualquer coisa que você escrever para este fluxo fica disponível para o processo filho em sua entrada padrão. O objeto de processo infantil também define uma propriedade `PID` que especifica o

ID do processo da criança. E define um método de matar () que você pode usar para encerrar um processo infantil.

16.10.4 Fork ()

Child_process.fork () é uma função especializada para executar um módulo de código JavaScript em um processo filho do nó. fork () espera os mesmos argumentos que Spawn (), mas o primeiro argumento deve especificar o caminho para um arquivo de código JavaScript em vez de um arquivo binário executável. Um processo infantil criado com fork () pode se comunicar com o processo pai através de seus fluxos padrão de entrada e saída padrão, como descrito na seção anterior para Spawn (). Mas, além disso, fork () permite outro canal de comunicação muito mais fácil entre os processos pais e filhos. Quando você cria um processo infantil com fork (), você pode usar o Send () Método do objeto de processamento infantil retornado para enviar uma cópia de um objeto para o processo infantil. E você pode ouvir a "mensagem" Evento no documento infantil para receber mensagens da criança. O Código em execução no processo infantil pode usar o processo.send () para enviar uma mensagem para o pai e pode ouvir eventos de "mensagem" em processo para receber mensagens do pai. Aqui, por exemplo, há algum código que usa fork () para criar uma criança Processar, então envia uma mensagem a essa criança e aguarda uma resposta:

```
const Child_process = require ("Child_Process"); // Inicie um novo processo de nó executando o código no Child.js em
```

nosso diretórioDeixe Child = Child_process.fork (`\$ __dirname/Child.js`); // Envie uma mensagem para a criança
Child.send ({x: 4, y: 3}); // Imprima a resposta da criança quando chegar. Child.on ("mensagem", mensagem => {console.log (message.hoToteNuse); // Isso deve imprimir "5" // Como apenas enviamos uma mensagem, esperamos apenas uma resposta. // Depois de receber -lo, chamamos desconexão () para rescindir a conexão // entre pai e filho. Isso permite ambos os processos para sair de maneira limpa. criança.disconnect ();}); E aqui está o código que é executado no processo infantil: // Aguarde por mensagens do nosso processo pai. process.on ("mensagem", mensagem => { // Quando recebermos um, faça um cálculo e envie o resultado // de volta ao pai. process.send ({hipotenuse: math.hypot (message.x, mensagem.Y)});}); Iniciar processos infantis é uma operação cara, e a criança processo teria que estar fazendo ordens de magnitude mais computação. Antes faria sentido usar o Fork () e o Interprocess comunicação dessa maneira. Se você está escrevendo um programa que precisar ser muito receptivo aos eventos de entrada e também precisa realizar tempo consumindo cálculos, então você pode considerar usar um separado processo da criança para realizar os cálculos para que eles não bloqueie o. Faça um loop de eventos e reduza a capacidade de resposta do processo pai.

(Embora um tópico - veja §16.11 - pode ser uma escolha melhor do que uma criança processo neste cenário.) O primeiro argumento a se enviar () será serializado com `JSON.stringify()` e desaparecerá no processo infantil `comJSON.parse()`, então você deve incluir apenas valores que são suportados pelo formato `JSON.send()` tem um segundo argumento especial, no entanto, isso permite que você transfira objetos de socket e servidor (a partir da rede?módulo) para um processo infantil. Os servidores de rede tendem a ser ligados a IO, em vez de ser ligado a computação, mas se você escreveu um servidor que precisa fazer mais computação do que uma única CPU pode lidar e se você estiver executando esse servidor em uma máquina com várias CPUs, então você pode usar `fork()` para criar vários processos filhos para lidar com solicitações. No processo pai, você pode ouvir eventos de "conexão" em seu servidor objeto, então obtenha o objeto de socket desse evento de "conexão" e `send()` isso - usando o segundo argumento especial - para uma das crianças processos a serem tratados. (Observe que esta é uma solução improvável para um cenário incomum. Em vez de escrever um servidor que bate filhos processos, provavelmente é mais simples manter seu servidor único e implantar várias instâncias em produção para lidar com a carga.)

16.11 tópicos dos trabalhadores Conforme explicado no início deste capítulo, a simultaneidade do nó O modelo é de thread único e baseado em eventos. Mas na versão 10 e mais tarde, o nó permite a verdadeira programação multithreaded, com uma API que espelha de perto a API dos trabalhadores da web definida por navegadores da web (§15.13). A programação multithreaded tem uma reputação merecida

por ser difícil. Isso é quase inteiramente por causa da necessidade de sincronizar cuidadosamente o acesso por threads com a memória compartilhada. Mas os threads JavaScript (em nó e navegadores) não compartilham memória por padrão, os perigos e dificuldades de usar tópicos não se aplicam para esses "trabalhadores" em JavaScript. Em vez de usar a memória compartilhada, os threads de trabalhadores do JavaScript comunicam-se pela passagem de mensagens. O tópico principal pode enviar uma mensagem para um tópico de trabalhador chamando o método `postMessage()` do objeto de trabalhador que representa esse tópico. O tópico do trabalhador pode receber mensagens de seus pais ouvindo eventos de "mensagem". E os trabalhadores podem enviar mensagens para o tópico principal com seu próprio versão do `PostMessage()`, que o pai pode receber com seu manipulador de eventos de `?mensagem?`. O código de exemplo deixará claro como isso funciona. Há três razões pelas quais você pode querer usar tópicos de trabalhador em um aplicativo do nó: Se o seu aplicativo realmente precisar fazer mais computação do que um núcleo da CPU pode lidar, então os threads permitem que você distribua trabalho nos vários núcleos, que se tornaram comuns em computadores hoje. Se você está fazendo cálculos computacionais ou aprendizado de máquina ou processamento de gráficos em um só nó, então você pode querer usar threads simplesmente para jogar mais poder de computação em seu problema. Mesmo que seu aplicativo não esteja usando todo o poder de uma CPU, você ainda pode usar threads para manter a responsabilidade do tópico principal. Considere um servidor que lida com solicitações grandes, mas relativamente pouco frequentes. Suponha isso

recebe apenas um pedido um segundo, mas precisa gastar cerca de metade segundo da computação (bloqueando a CPU) para processar cada solicitar. Em média, ficará ocioso 50% do tempo. Mas quando dois pedidos chegam dentro de alguns milissegundos um do outro, o servidor nem será capaz de iniciar uma resposta a o segundo pedido até que o cálculo da primeira resposta seja completo. Em vez de executar o cálculo, o servidor pode iniciar a resposta a ambos solicitam imediatamente e proporcionam uma melhor experiência para os clientes do servidor. Supondo que o servidor tenha mais de um CPU Core, também pode calcular o corpo de ambas as respostas em paralelo, mas mesmo que exista apenas um núcleo, usando trabalhadores ainda melhora a capacidade de resposta. Em geral, os trabalhadores nos permitem tornar o bloqueio síncrono operações em operações assíncronas não bloqueadoras. Se você estiver escrevendo um programa que depende do código legado que é inevitavelmente síncrono, você poderá usar os trabalhadores para evitar bloquear quando precisar chamar esse código legado. Os tópicos dos trabalhadores não são tão pesados ?? quanto os processos infantis, mas eles não são leves. Geralmente não faz sentido criar um trabalhador, a menos que você tenha um trabalho significativo para isso. E, geralmente falando, se o seu programa não estiver ligado à CPU e não estiver tendo problemas de capacidade de resposta, então você provavelmente não precisa de trabalhadores. 16.11.1 Criando trabalhadores e passagens de mensagens O módulo do nó que define os trabalhadores é conhecido como "trabalhador_threads". Nesta seção, nos referiremos a ela com os threads de identificador: const threads = requer ("trabalhador_threads");

Este módulo define uma classe de trabalhador para representar um fio de trabalhador e Você pode criar um novo thread com o threads.worker ()construtor.O código a seguir demonstra o uso deste construtor para criar um trabalhador e mostra como passar mensagens do tópico principal para trabalhador e de trabalhador para o tópico principal.Também demonstra um truque isso permite que você coloque o código do encadeamento principal e o código do tópico do trabalhador no mesmo arquivo.const threads = requer ("trabalhador_threads");// O módulo Worker_threads exporta o Ismainthread booleano propriedade// Esta propriedade é verdadeira quando o Node está executando o tópico principal e é// Falso quando o nó está executando um trabalhador.Podemos usar este fato para implementar// Os threads principais e trabalhadores no mesmo arquivo.if (threads.ismhread) {// Se estamos correndo no tópico principal, tudo o que fazemos é exportar// uma função.Em vez de realizar um computacionalmente intensivo// tarefa no tópico principal, essa função passa a tarefa para um trabalhador// e retorna uma promessa que resolverá quando o trabalhador está feito.Module.Exports = Função Reticulatessplines (splines) {retornar nova promessa ((resolver, rejeitar) => {// Crie um trabalhador que carregue e execute o mesmo arquivo de código// Observe o uso do __filename especial variável.deixe o reticulador = new Threads.worker (__ nome do arquivo);// Passe uma cópia da matriz Splines para o trabalhador reticulador.postMessage (splines);// e depois resolva ou rejeite a promessa quando nós pegar// Uma mensagem ou erro do trabalhador.reticulador.on ("mensagem", resolver);2

Erro ao traduzir esta página.

módulo. Se você deseja um caminho em relação ao módulo atual, use algo como Path.Resolve (__dirname, 'Trabalhadores/Reticulador.js'). O construtor trabalhador () também pode aceitar um objeto como seu segundo argumento, e as propriedades desse objeto fornecem opções para o trabalhador. Vamos abordar várias dessas opções mais tarde, mas por enquanto observe que se você passar {avaliar: true} como o segundo argumento, então o primeiro argumento para trabalhador () é interpretado como um string de código JavaScript a ser avaliado em vez de um nome de arquivo: novo threads.worker () const threads = requer ("trabalhador_threads"); threads.parentport.postMessage (threads.ismhread); ` , {avaliar: true}). on ("mensagem", console.log); // isso vai imprimir "false" Node faz uma cópia do objeto passada para PostMessage () do que compartilha-lo diretamente com o tópico do trabalhador. Isso impede o tópico do trabalhador e o thread principal do compartilhamento de memória. Você pode esperar que essa cópia seja feita com JSON.stringify () e json.parse () (§11.6). Mas, de fato, o nó empresta uma técnica robusta conhecida como algoritmo de clone estruturado da webnavegadores. O algoritmo de clone estruturado permite a serialização da maioria dos JavaScript tipos, incluindo objetos de mapa, conjunto, data e regexp e matrizes digitadas. Mas não pode, em geral, os tipos de cópia definidos pelo host do ambiente, como soquetes e fluxos. Observe, no entanto, que esses buffer os objetos são parcialmente suportados: se você passar um buffer para

PostMessage () será recebido como um Uint8Array e pode serconvertido de volta em um buffer com buffer.from ().Leia maissobre o algoritmo de clone estruturado em ?O clone estruturadoAlgoritmo".16.11.2 O ambiente de execução do trabalhadorNa maioria das vezes, o código JavaScript em um thread de trabalhador de nós é executado apenasComo faria no tópico principal do Node.Existem algumas diferenças quevocê deve estar ciente, e algumas dessas diferenças envolvemPropriedades do segundo argumento opcional para o trabalhador ()construtor:Como vimos, threads.ismhhread é verdadeiro noTópico principal, mas é sempre falso em qualquer tópico de trabalhador.Em um tópico de trabalhador, você pode usarthreads.parentport.postMessage () para enviar ummensagem para o tópico pai ethreads.parentport.on para registrar manipuladores de eventos paramensagens do thread pai.No tópico principal,threads.parentport é sempre nulo.Em um tópico de trabalhador, threads.workerdata está definido como uma cópiada propriedade Workerdata do segundo argumento para oTrabalhador () construtor.No tópico principal, esta propriedade ésempre nulo.Você pode usar esta propriedade WorkerData parapasse uma mensagem inicial para o trabalhador que estará disponível comoassim que começa para que o trabalhador não precise esperar por umEvento de ?mensagem? antes de começar a fazer o trabalho.Por padrão, process.env em um tópico de trabalhador é uma cópia deprocess.env no thread pai.Mas o fio pai pode

Especifique um conjunto personalizado de variáveis ??de ambiente definindo oPropriedade Env do segundo argumento para o trabalhador ()construtor.Como um caso especial (e potencialmente perigoso), oO tópico pai pode definir a propriedade Env parathreads.share_env, que fará com que os dois tópicoscompartilhe um único conjunto de variáveis ??de ambiente para que uma mudança emUm tópico é visível no outro.Por padrão, o processo do processo.stdin em um trabalhador nuncapossui dados legíveis sobre ele.Você pode alterar esse padrão porPassando stdin: verdadeiro no segundo argumento para oTrabalhador () construtor.Se você fizer isso, então o stdinA propriedade do objeto trabalhador é um fluxo gravável.Quaisquer dadosque o pai escreve para trabalhador.stdin se torna legívelem process.stdin no trabalhador.Por padrão, o processo.stdout e process.stderrfluxos no trabalhador são simplesmente canalizados para o correspondentefluxos no thread pai.Isso significa, por exemplo, queconsole.log () e console.error () produzem saídaexatamente da mesma maneira em um fio de trabalhador que elestópico principal.Você pode substituir esse padrão passandostdout: verdadeiro ou stderr: true no segundo argumento paraO construtor trabalhador ().Se você fizer isso, então qualquer saída doO trabalhador escreve para esses fluxos se torna legível pelo paiThread no trabalhador.stdout e trabalhador.stderrtópicos.(Há uma inversão potencialmente confusa do fluxoinstruções aqui, e vimos a mesma coisa com o filhoprocessos no início do capítulo: os fluxos de saída de um trabalhadorOs threads são fluxos de entrada para o fio pai e a entradaO fluxo de um trabalhador é um fluxo de saída para o pai.)Se um thread trabalhador chama process.exit (), apenas o tópicosai, não todo o processo.

Os tópicos dos trabalhadores não têm permissão para alterar o estado compartilhado do processo da qual eles fazem parte. Funções como `process.chdir()` e `process.setUid()` lançarão exceções quando invocadas de um trabalhador. Os sinais do sistema operacional (como SIGINT e SIGTERM) são entregues apenas no fio principal; Eles não podem ser recebidos ou tratados em fios de trabalhador.

16.11.3 canais de comunicação e Messageports

Quando um novo tópico de trabalhador é criado, um canal de comunicação é criado junto com ele que permite que as mensagens sejam passadas entre o trabalhador e o tópico pai. Como vimos, o trabalhador thread usa `threads.parentPort` para enviar e receber mensagens para e partir do fio pai, e o tópico pai usa o `trabalhadorObjeto` -se a enviar e receber mensagens para e do tópico do trabalhador. A API do Thread Worker também permite a criação de custumes canais de comunicação usando a API `MessageChannel` definida por Navegadores da Web e cobertos em §15.13.5. Se você leu essa seção, Muito do que se segue parecerá familiar para você. Suponha que um trabalhador precise lidar com dois tipos diferentes de mensagens enviadas por dois módulos diferentes no encadeamento principal. Esses dois diferentes módulos poderiam compartilhar o canal padrão e enviar mensagens para o trabalhador. `PostMessage()`, mas seria mais limpo se cada módulo tivesse seu próprio canal privado para enviar mensagens ao trabalhador. Ou Considere o caso em que o tópico principal cria dois trabalhadores independentes. Um canal de comunicação personalizado pode permitir que os dois trabalhadores

para se comunicar diretamente entre si, em vez de ter que enviar tudosas mensagens através do pai.Crie um novo canal de mensagem com o Messagechannel ()construtor.Um objeto Messagechannel tem duas propriedades, nomeadoPORT1 e PORT2.Essas propriedades se referem a um par de mensagensobjetos.Chamar PostMessage () em um dos portos causará umEvento de ?mensagem? a ser gerado por outro com um clone estruturado deo objeto de mensagem:const threads = requer ("trabalhador_threads");deixe canal = new threads.MessageChannel ();canal.port2.on ("mensagem", console.log);// registrar qualquer ummensagens que recebemoscanal.port1.postMessage ("hello");// vai causar"Olá" para ser impressoVocê também pode ligar para o fechamento () em qualquer porta para quebrar a conexãoentre as duas portas e sinalizar que não serão mais mensagenstroca.Quando Close () é chamado em qualquer porta, um evento "fechado" éentregue em ambas as portas.Observe que o exemplo de código acima cria um par de objetos de Messageport então usa esses objetos para transmitir uma mensagem na principalfio.Para usar canais de comunicação personalizados com trabalhadores,Devemos transferir uma das duas portas do tópico em que estácriado para o tópico em que será usado.A próxima seção explicaComo fazer isso.16.11.4 Transferindo Messageports e digitadoMatrizes

A função PostMessage () usa o algoritmo de clone estruturado,E como observamos, ele não pode copiar objetos como ssockets e fluxos.Ele pode lidar com objetos Messageport, mas apenas como um caso especial usando umTécnica especial.O método PostMessage () (de um objeto de trabalhador,de threads.parentport, ou de qualquer objeto Messageport) leva umSegundo argumento opcional.Este argumento (chamado transferlist) éuma variedade de objetos que devem ser transferidos entre fios e nãosendo copiado.Um objeto Messageport não pode ser copiado pelo clone estruturadoAlgoritmo, mas pode ser transferido.Se o primeiro argumento paraPostMessage () incluiu uma ou mais porteiras (aninhadasarbitrariamente profundamente dentro do objeto de mensagem), então aqueles MessageportObjetos também devem aparecer como membros da matriz passados ??como o segundoargumento.Fazer isso diz ao Node que não precisa fazer uma cópia deo Messageport, e pode apenas dar o objeto existente aooutro tópico.A coisa principal a entender, no entanto, sobre a transferências os valores entre os threads são que, uma vez transferido um valor, ele não podeser usado mais no thread que chamado PostMessage ().Aqui está como você pode criar um novo MessageChannel e transferir umde suas porteiras para um trabalhador:// Crie um canal de comunicação personalizadoconst threads = requer ("trabalhador_threads");deixe canal = new threads.MessageChannel ();// Use o canal padrão do trabalhador para transferir uma extremidade deo novo// canal para o trabalhador.Suponha que quando o trabalhadorrecebe isso// mensagem para começar imediatamente a ouvir mensagens em

o novo canal.trabalhador.postMessage ({Command: "Changechannel", Data:canal.port1},[canal.port1]);// agora envie uma mensagem ao trabalhador usando nosso fim docanal personalizado canal.port2.postMessage ("Você pode me ouvir agora?");// e ouça as respostas do trabalhador também Channel.port2.on ("Mensagem", HandleMessagesFromWorker);Os objetos MessagePort não são os únicos que podem ser transferidos.Sevocê ligaUma mensagem que contém uma ou mais matrizes digitadas aninhadas arbitrariamente no fundo da mensagem), essa matriz digitada (ou aquelas matrizes digitadas) irá simplesmente ser copiado pelo algoritmo de clone estruturado.Mas matrizes digitadas pode ser grande;Por exemplo, se você estiver usando um tópico de trabalhador para fazer a imagemProcessando em milhões de pixels.Então, para eficiência, Postmessage ()Também nos dá a opção de transferir matrizes digitadas em vez de copiá-las.(Os threads compartilham a memória por padrão. Tentos do trabalhador em JavaScript geralmente evita a memória compartilhada, mas quando permitimos esse tipo de transferência controlada, pode ser feita com muita eficiência.) O que faz disso seguro é que quando uma matriz digitada é transferida para outro tópico, é retornada -se inutilizável no tópico que o transferiu.Na imagem-cenário de processamento, o tópico principal pode transferir os pixels de uma imagem para o fio do trabalhador, e então o tópico do trabalhador pode transferir os pixels processados ??de volta ao fio principal quando ele foi feito.OA memória não precisaria ser copiada, mas nunca seria acessível por dois tópicos de uma só vez.Para transferir uma matriz digitada em vez de copiá-la, inclua o ArrayBuffer

Isso apóia a matriz no segundo argumento para PostMessage ():Seja pixels = novo Uint32Array (1024*1024); // 4 megabytes dememória// Suponha que lemos alguns dados nesta matriz digitada e depoisTransfira o// pixels para um trabalhador sem copiar.Observe que não colocamos a matriz// na lista de transferências, mas o objeto buffer da matriz em vez de.trabalhador.PostMessage (pixels, [pixels.buffer]);Assim como no Messageports transferido, uma matriz digitada transferida se torna inutilizável uma vez transferido.Nenhuma exceção é jogada se você tentarUse um Messageport ou uma matriz digitada que foi transferida;essesOs objetos simplesmente param de fazer qualquer coisa quando você interage com eles.16.11.5 Compartilhando matrizes digitadas entre threadsAlém de transferir matrizes digitadas entre threads, é na verdade possível compartilhar uma matriz digitada entre os threads.Simplesmente crie umSharedArrayBuffer do tamanho desejado e depois use esse buffer para criar uma matriz digitada.Quando uma matriz digitada é apoiada por umSharedArrayBuffer é passado via PostMessage (), o subjacenteA memória será compartilhada entre os threads.Você não deve incluir o buffer compartilhado no segundo argumento para PostMessage () nestecaso.Você realmente não deve fazer isso, no entanto, porque o JavaScript nunca foi projetado com a segurança do tópico em mente e a programação multithreaded é muito difícil de acertar.(E é por isso que SharedArrayBuffer não foi coberto em §11.2: é um recurso de nicho que é difícil de acertar.) Mesmo

O operador simples ++ não é seguro para threads porque precisa ler um valor, aumentá-lo e escrever de volta. Se dois tópicos estão incrementando um valor ao mesmo tempo, geralmente será incrementado uma vez, como o código a seguir demonstra:

```
const threads = require("trabalhador_threads");
if (threads.ismhrthread) { // No tópico principal, criamos uma matriz digitada compartilhada com // Um elemento. Ambos os tópicos serão capazes de ler e escrever // SharedArray [0] ao mesmo tempo. Seja SharedBuffer = new SharedArrayBuffer (4); Let SharedArray = new Int32Array (SharedBuffer); // Agora crie um tópico de trabalhador, passando a matriz compartilhada para isso com // como seu valor inicial de dados trabalhadores, então não precisamos preocupar com // enviando e recebendo uma mensagem. Let Worker = new Threads.Worker (__ nome do arquivo, {WorkerData:sharedArray}); // Aguarde o trabalhador começar a correr e depois incremente o // Inteiro compartilhado 10 milhões de vezes. trabalhador.on ("online", () => {para (vamos i = 0; i <10_000_000; i++) sharedArray [0] ++; // Depois de terminar com nossos incrementos, começamos ouvindo para // Eventos de mensagens para que sabemos quando o trabalhador está pronto. trabalhador.on ("mensagem", () => { // Embora o número inteiro compartilhado tenha sido incrementado // 20 milhões de vezes, seu valor geralmente será muito menos. // No meu computador, o valor final é normalmente menos de 12 milhões. console.log (SharedArray [0]);});};
```

});} outro {// No tópico do trabalhador, obtemos a matriz compartilhada deWorkerData// e depois aumentam 10 milhôes de vezes.Let SharedArray = Threads.WorkerData;para (vamos i = 0; i <10_000_000; i++) sharedArray [0] ++;// Quando terminarmos o incremento, informe o tópico principalthreads.parentport.postMessage ("feito");}Um cenário em que pode ser razoável usar umSharedArrayBuffer é quando os dois threads operam totalmente separadosseções da memória compartilhada.Você pode aplicar isso criando doismatrizes digitadas que servem como visualizações de regiões não sobrepostas doBuffer compartilhado e, em seguida, seus dois threads usem esses dois separadosmatrizes digitadas.Uma classificação de mesclagem paralela pode ser feita assim: um tópicoclassifica a metade inferior de uma matriz e o outro tópico classifica a metade superior,por exemplo.Ou alguns tipos de algoritmos de processamento de imagens também sãoAdequado para esta abordagem: vários tópicos que trabalham em regiões descontsda imagem.Se você realmente deve permitir que vários tópicos acessem a mesma região de ummatrix compartilhada, você pode dar um passo em direção à segurança do tópico com ofunções definidas pelo objeto Atomics.Atomics foi adicionado aJavaScript quando SharedArrayBuffer deveria definir operações atômicasSobre os elementos de uma matriz compartilhada.Por exemplo, o atomics.add ()A função lê o elemento especificado de uma matriz compartilhada, adiciona um especificadovvalor para ele e escreve a soma de volta à matriz.Isso faz issoatomicamente como se fosse uma única operação, e garante que nenhum outroO thread pode ler ou escrever o valor enquanto a operação está ocorrendo.Atomics.add () nos permite reescrever o código de incremento paralelo nós

apenas olhou e obtenha o resultado correto de 20 milhões de incrementos de um elemento de matriz compartilhada:
const threads = requer ("trabalhador_threads");
if (threads.ismharethread) {Seja SharedBuffer = new SharedArrayBuffer (4);
Let SharedArray = new Int32Array (SharedBuffer);
Let Worker = new Threads.Worker (_ nome do arquivo, {WorkerData:sharedArray});
trabalhador.on ("online", () => {para (vamos i = 0; i <10_000_000; i++) {Atomics.add (SharedArray, 0, 1); // ThreadSafeincremento atômico}
trabalhador.on ("mensagem", (mensagem) => {// Quando ambos os threads estiverem concluídos, use um threadSafefunção// para ler a matriz compartilhada e confirmar quetem o// valor esperado de 20.000.000.console.log (atomics.load (sharedArray, 0));});});}
outro {Let SharedArray = Threads.WorkerData;
para (vamos i = 0; i <10_000_000; i++) {Atomics.add (SharedArray, 0, 1); // ThreadSafeincremento atômico}
threads.parentport.postMessage ("feito");}
Esta nova versão do código imprime corretamente o número 20.000.000. Mas é cerca de nove vezes mais lento que o código incorreto que substitui. Isto seria muito mais simples e muito mais rápido para fazer todos os 20 milhões

incrementos em um thread. Observe também que as operações atômicas podem poder para garantir a segurança dos threads para algoritmos de processamento de imagens para os quais cada um O elemento de matriz é um valor totalmente independente de todos os outros valores. Mas em A maioria dos programas do mundo real, vários elementos de matriz estão frequentemente relacionados a um do outro e algum tipo de sincronização de roscas de nível superior é obrigatório. Os atomics de baixo nível. wait () e A função atomics. Notify () pode ajudar com isso, mas uma discussão sobre seu uso está fora de escopo deste livro.

16.12 Resumo

Embora o JavaScript tenha sido criado para ser executado em navegadores da web, o Node possui transformou o JavaScript em uma linguagem de programação de uso geral. Isso é particularmente popular para implementar servidores da web, mas é profundamente ligadas ao sistema operacional significam que também é uma boa alternativa para scripts de conchas. Os tópicos mais importantes abordados neste longo capítulo incluem: APIs assíncronas por antecedentes do Node e seus threads únicos, retorno de chamada e estilo de concorrência baseado em eventos. Tipos, buffers e fluxos fundamentais do Node. Os módulos "Fs" e "Path" do Node para trabalhar com o FileSystem. Os módulos "http" e "https" do Node para escrever clientes HTTP para servidores. Módulo "Net" do Node para escrever clientes que não usam http para servidores.

Módulo "Child_process" do Node para criar ecomunicação com processos infantis.Módulo "Worker_threads" do Node para verdadeiro multithreadedProgramação usando passagem de mensagens em vez de compartilhadomemória.1O nó define uma função fs.copyfile () que você realmente usaria na prática.2Muitas vezes, é mais limpo e mais simples definir o código do trabalhador em um arquivo separado.Mas essetruquede ter dois tópicos executando seções diferentes do mesmo arquivo me impressionaram quando eu primeiroencontrou -o para a chamada do sistema unix fork ().E eu acho que vale a pena demonstrarEsta técnica simplesmente por sua estranha elegância.

Capítulo 17. Ferramentas JavaScript e extensões

Parabéns por chegar ao capítulo final deste livro. Se você temLeia tudo o que vem antes, agora você tem um detalhadocompreensão da linguagem JavaScript e sabe como usá -la emNó e nos navegadores da web. Este capítulo é um tipo de graduaçãopresente: introduz um punhado de ferramentas importantes de programação queMuitos programadores JavaScript acham útil e também descrevem doisExtensões amplamente usadas para a linguagem JavaScript central. Seja ou nãoVocê escolhe usar essas ferramentas e extensões para seus próprios projetos, vocêtem quase certeza de vê -los usados ??em outros projetos, então é importantepelo menos saber o que são. As ferramentas e extensões de idiomas abordadas neste capítulo são:Eslint para encontrar possíveis insetos e problemas de estilo em seu código. Mais bonito para formatar seu código JavaScript em um padronizadocaminho. Jogue como uma solução completa para escrever testes de unidade JavaScript. NPM para gerenciar e instalar as bibliotecas de software queSeu programa depende. Ferramentas de Bundling de Código-como Webpack, Rollup e Parcel-queConverta seus módulos de código JavaScript em um único pacotepara uso na web.

Babel para traduzir o código JavaScript que usa novidadesRecursos de linguagem (ou que usam extensões de linguagem) emCódigo JavaScript que pode ser executado nos navegadores da web atuais.A extensão da linguagem JSX (usada pela estrutura do React)Isso permite que você descreva as interfaces de usuário usando JavaScriptExpressões que se parecem com a marcação HTML.A extensão da linguagem de fluxo (ou o tipo de texto semelhanteextensão) que permite anotar seu código JavaScriptcom tipos e verifique seu código para obter a segurança do tipo.Este capítulo não documenta essas ferramentas e extensões em nenhum maneira abrangente.O objetivo é simplesmente explicá -los o suficienteprofundidade que você pode entender por que eles são úteis e quando você pode querer usá -los.Tudo coberto neste capítulo é amplamente usado emO mundo da programação JavaScript, e se você decidir adotar uma ferramentaOu extensão, você encontrará muita documentação e tutoriais online.17.1 LINT COM ESLINTNa programação, o termo fiapos refere -se a codificar isso, enquanto tecnicamenteCorreto, é desagradável, ou um possível bug, ou abaixo do ideal de alguma forma.UMLinter é uma ferramenta para detectar fiapos no seu código, e o linhagem é o processo de executar um linhador em seu código (e depois consertar seu código para remover o fiapo para que o linhador não reclame mais).O linter mais comumente usado para JavaScript hoje é Eslint.Se vocêExecute -o e depois reserve um tempo para realmente corrigir os problemas que aponta, ele tornará seu código mais limpo e menos provável de ter erros.Considere o Código seguinte:

var x = 'não utilizado';Função de exportação Fatorial (x) {if (x == 1) {retornar 1;} outro {Retornar X * Fatorial (X-1)}}Se você executar Eslint neste código, poderá obter saídas assim:\$ eslint Code/CH17/linty.js código/CH17/linty.js1: 1 Erro Var inesperado, use Let ou constnão-var1: 5 Erro 'x' é atribuído um valor, mas nunca usadonão-used-vars1: 9 Strings de aviso devem usar o DoubleQuotecitações4:11 Erro esperado '====' e, em vez disso, viu '==eqeqeq5: 1 Erro recuo esperado de 8 espaços, mas encontrou 6recuar7:28 Erro faltando semicolonsemi? 6 problemas (5 erros, 1 aviso)3 erros e 1 aviso potencialmente fixável com o '--fix` opção.Às vezes, os linteres podem parecer nitpicky.Realmente importa se nósusou citações duplas ou citações únicas para nossas cordas?Por outro lado,Obter o recuo correto é importante para a legibilidade e o uso =====vamos, em vez de == e Var o protege de bugs sutis.EVariáveis ??não utilizadas têm peso morto em seu código - não há razão paraMantenha aqueles por perto.

Eslint define muitas regras de linha e possui um ecossistema de plug-ins que acrescentam muito mais. Mas Eslint é totalmente configurável, e você pode definir um arquivo de configuração que tira Eslint para aplicar exatamente as regras que você quer e apenas essas regras.

17.2 Javascript Formating com mais bonito

Uma das razões pelas quais alguns projetos usam linters é fazer cumprir um estilo de codificação consistente para que, quando uma equipe de programadores está funcionando em uma base de código compartilhada, eles usam convenções de código compatível. Esse inclui regras de indentação de código, mas também pode incluir coisas como o tipo de aspas preferidas e se deve haver um espaço entre a palavra-chave e os parênteses abertos que seguem.

Uma alternativa moderna para aplicar regras de formatação de código por meio de um linter é usar uma ferramenta como o mais bonito para analisar e reformar automaticamente seu código. Suponha que você tenha escrito a seguinte função, que funciona, mas é formatado de forma não convencional:

```
Função Fatorial (X){if (x === 1) {return 1}else {return x*fatorial (x-1)}}A execução mais bonita neste código corrige o recuo, adiciona faltaSemicolons, adiciona espaços em torno de operadores binários e insere quebras de linha
```

depois {e antes}, resultando em muito mais convencional código:\$ Prettier factorial.js função factorial (x) {if (x === 1) {retornar 1;} outro {retornar x * factorial (x - 1);}}Se você invocar mais bonito com a opção - -write, ele simplesmente reformate o arquivo especificado em vigor em vez de imprimir um reformado versão.Se você usar o Git para gerenciar seu código -fonte, poderá invocar Mais bonito com a opção -write em um gancho de comprometimento para que o código seja formatado automaticamente antes de ser marcado.Mais bonito é particularmente poderoso se você configurar seu editor de código para executarEle automaticamente toda vez que você salva um arquivo.Eu acho libertador escrever código desleixado e veja -o corrigido automaticamente para mim.Mais bonito é configurável, mas possui apenas algumas opções.Você pode selecionar o comprimento máximo da linha, a quantidade de recuo, seja semicolons deve ser usado, se as strings devem ser únicas ou duplas,E algumas outras coisas.Em geral, as opções padrão de Prettier são bastante razoável.A idéia é que você apenas adote mais bonito para o seu projeto eEntão nunca mais preciso pensar em formatar o código novamente.Pessoalmente, eu realmente gosto de usar projetos mais bonitos em JavaScript.Eu não tenho housei para o código neste livro, no entanto, porque em grande parte do meu códigoEu confio em uma formatação cuidadosa para alinhar meus comentários verticalmente, e

Mais bonito os estraga.17.3 Teste de unidade com JESTOs testes de escrita são uma parte importante de qualquer programação não trivialprojeto.Línguas dinâmicas como JavaScript Suport Testing FrameworksIsso reduz drasticamente o esforço necessário para escrever testes e quaseTorne a escrita de teste divertida!Existem muitas ferramentas de teste e bibliotecas paraJavaScript, e muitos são escritos de maneira modular para que seja possível escolher uma biblioteca como seu corredor de teste, outra biblioteca para asserções,e um terceiro para zombar.Nesta seção, no entanto, descreveremos o JEST,que é uma estrutura popular que inclui tudo o que você precisa em um pacote único.Suponha que você tenha escrito a seguinte função:

```
const getjson = require("./getjson.js");
/***
 * gettemperature () toma o nome de uma cidade como sua entrada,e retorna* Uma promessa que resolverá a temperatura atual de aquela cidade,* Em graus Fahrenheit.Ele se baseia em um serviço (falso) da webque retorna* Temperaturas mundiais em graus Celsius.*/
Module.Exports = Função Async GetTemperature (City) { // Obtenha a temperatura em Celsius do serviço da webSeja c = aguarda getjson (`https://globaltemps.example.com/api/city/${City.toLowerCase()}`); // converte em Fahrenheit e retorne esse valor.
    return (C * 5 /9) + 32; // TODO: Verifique duas
```

Erro ao traduzir esta página.

```
// Este segundo teste verifica que GetTemperature ()convertidos// Celsius para Fahrenheit corretamente teste
("converte c a f corretamente", async () => {getjson.mockResolVedValue (0); // Se getjson retorna 0cEspere
(agenda gettemperature ("x")). Tobe (32); // NósEspere 32f// 100c deve se converter para
212fgetjson.mockResolVedValue (100); // Se getjson retorna 100cEspere (agenda gettemperature ("x")). Tobe
(212); // NósEspere 212f});}); Com o teste escrito, podemos usar o comando jest para executá-lo, e nósDescubra
que um de nossos testes falha:$ jest gettemperatureFalha ch17/gettemperature.test.jsgetTemperature ()? Invoca a
API correta (4ms)? converte C para F corretamente (3ms)? gettemperature () ?converte c para f
corretamenteEspere (recebido) .ToBe (esperado) // object.is igualdadeEsperado: 212Recebido:
87.55555555555629 // 100c deve se converter para 212f30 |getjson.mockResolVedValue (100); // Se getjson
retorna 100c> 31 |Espere (agenda gettemperature ("x")). Tobe (212); // Espere 212f|^32 |);33 |);34 |no objeto.
<Anonymous>
```

(CH17/gettemperature.test.js: 31: 43) Suítes de teste: 1 falha, 1 totalTestes: 1 falhou, 1 passou, 2 totalInstantâneos: 0 TotalTempo: 1.403s Run todas as suítes de teste correspondentes /gettemperature /i. Nossa implementação getTemPerature () está usando o errado Fórmula para converter C a F. do que multiplicar por 9 e dividir por 5. Se corrigirmos o código e executarmos a brincadeira novamente, podemos ver os testes passarem. E, como um bônus, se adicionarmos o -argumento de cobertura quando invocarmos o JEST, ele vai calcular e exibir a cobertura do código para nossos testes: \$ jest -Cobertura gettemperaturePasse ch17/gettemperature.test.js getTemperature ()? Invoca a API correta (3ms)? Converte C em F corretamente (1ms)----- | ----- | ----- | ----- | ----- | ----- | Arquivo |% Stmts |% Ramificação |% Funcs |% Linhas |Linha descoberta #s |----- | ----- | ----- | ----- | ----- | Todos os arquivos |71.43 |100 |33.33 |83.33 ||getJSON.js |33.33 |100 |0 |50 |2 |getTemperature.js |100 |100 |100 |100 ||----- | ----- | ----- | ----- | ----- | ----- | Suítes de teste: 1 aprovado, 1 totalTestes: 2 passados, 2 totalInstantâneos: 0 TotalTempo: 1.508s Run todas as suítes de teste correspondentes /gettemperature /i.

Executar nosso teste nos deu 100% de cobertura de código para o módulo que estávamos Teste, que é exatamente o que queríamos. Só nos deu parcial cobertura de `getjson()`, mas zombamos desse módulo e não éramos Tentando testá-lo, o que é esperado.

17.4 Gerenciamento de pacotes com NPM

No desenvolvimento moderno de software, é comum para qualquer não trivial Programa que você escreve para depender de bibliotecas de software de terceiros. Se Você está escrevendo um servidor da web no nó, por exemplo, você pode estar usando a estrutura expressa. E se você está criando uma interface de usuário para ser Exibido em um navegador da web, você pode usar uma estrutura de front-end como Reagem ou litement ou angular. Um gerente de pacotes facilita a Encontre e instale pacotes de terceiros como esses. Tão importante quanto ainda, um O gerenciador de pacotes acompanha o que o seu código depende de dependee salva essas informações em um arquivo para que quando alguém quiser Para experimentar o seu programa, eles podem baixar seu código e sua lista dedependências e, em seguida, use seu próprio gerenciador de pacotes para instalar todos os Pacotes de terceiros que seu código precisa.

NPM é o gerente de pacotes que é empacotado com nó e foi introduzido em §16.1.5. É igualmente útil para JavaScript do lado do cliente Programação como é para programação do lado do servidor com o nó, no entanto. Se você está experimentando o projeto JavaScript de outra pessoa, então um dos As primeiras coisas que você costumam fazer depois de baixar o código deles é digitar `NPM Instale`. Isso lê as dependências listadas no `package.json` arquivo e baixar os pacotes de terceiros que o As necessidades do projeto e salvam -as em um `Node_modules/` Diretório.

Você também pode digitar o NPM Install <name> para instalar um pacote específico para o node_modules/ diretório do seu projeto:\$ npm Install ExpressAlém de instalar o pacote nomeado, o NPM também faz um registro da dependência no arquivo package.json para o projeto.Gravação das dependências dessa maneira é o que permite que outras pessoas instalem dependências simplesmente digitando a instalação do NPM.O outro tipo de dependência está em ferramentas de desenvolvedor que são necessárias para desenvolvedores que querem trabalhar em seu projeto, mas na verdade não são precisadas para executar o código.Se um projeto usa mais bonito, por exemplo, para garantir que todo o seu código é consistentemente formatado, então mais bonito é um ?dev dependência ?e você pode instalar e gravar um deles com -salvar-dev:\$ npm install-salve-dev mais bonitoÀs vezes, você pode querer instalar ferramentas de desenvolvedor globalmente para que elas são acessíveis em qualquer lugar, mesmo para código que não faz parte de um formal Projeto com um arquivo package.json e um diretório node_modules/.Para que você pode usar a opção -g (para global):\$ npm install -g eslint jest/usr/local/bin/eslint ->/usr/local/lib/node_modules/eslint/bin/eslint.js/usr/local/bin/jest ->/usr/local/lib/node_modules/jest/bin/jest.js+ jest@24.9.0+ eslint@6.7.2Adicionado 653 pacotes de 414 colaboradores em 25.596s\$ qual eslint

Erro ao traduzir esta página.

Erro ao traduzir esta página.

diferenciado com base em quão configuráveis ??eles são ou em quão fáceis eles são para usar. Webpack já existe há muito tempo, tem um grande ecossistema de plug-ins, é altamente configurável e pode suportar mais antigoBibliotecas não módulos. Mas também pode ser complexo e difícil de configurar. No outro extremo do espectro está o pacote que se destina a zero-Alternativa de configuração que simplesmente faz a coisa certa. Além de executar o pacote básico, as ferramentas do Bundler também podem fornecer alguns recursos adicionais: Alguns programas têm mais de um ponto de entrada. Uma webAplicação com várias páginas, por exemplo, poderia ser escrita com um ponto de entrada diferente para cada página. Aparecedores em geralPermita que você crie um pacote por ponto de entrada ou crie um Pacote único que suporta vários pontos de entrada. Os programas podem usar o import () em sua forma funcional (§10.3.6) em vez de sua forma estática para carregar módulos dinamicamente quando Eles são realmente necessários em vez de carregá -los estaticamente em Hora de inicialização do programa. Fazer isso geralmente é uma boa maneira de Melhore o tempo de inicialização do seu programa. Ferramentas de Bundler isso suporte de suporte () pode ser capaz de produzir múltiplos resultados pacotes: um para carregar no horário de inicialização e um ou mais que são carregados dinamicamente quando necessário. Isso pode funcionar bem se houver apenas algumas chamadas para importar () em seu programa e elasMódulos de carga com conjuntos de dependências relativamente disjuntos. Se Os módulos carregados dinamicamente compartilham dependências e depoistorna -se complicado descobrir quantos pacotes produzirem. E é provável que você tenha que configurar manualmente seu empuxo para resolver isso. Muitas vezes geralmente podem produzir um arquivo de mapa de origem que define um mapeamento entre as linhas de código no pacote e o

linhas correspondentes nos arquivos de origem original. Isso permite Ferramentas de desenvolvedor de navegador para exibir automaticamente JavaScript Erros em seus locais originais não recrutados. Às vezes, quando você importa um módulo para o seu programa, você Use apenas alguns de seus recursos. Uma boa ferramenta de Bundler pode analisar o código para determinar quais partes não são utilizadas e podem ser omitidas dos feixes. Este recurso passa pelo capricho Nome de "Troca de árvores". Muitos pacotes normalmente têm uma arquitetura baseada em plug-in e Suporte plug-ins que permitem importar e agrupar módulos que na verdade não são arquivos do código JavaScript. Suponha que isso Seu programa inclui um grande bloco de dados compatível com JSON estrutura. Os pacotes de código podem ser configurados para permitir que você Move essa estrutura de dados para um arquivo JSON separado e depois importe-o para o seu programa com uma declaração como importação widgets de "./big-widget-list.json". Da mesma forma, desenvolvedores da web que incorporam CSS em seus programas JavaScript podem usar plug-ins de Bundler que lhes permitem Importar arquivos CSS com uma diretiva de importação. Nota, no entanto, que se você importar algo que não seja um arquivo JavaScript, você estará usando uma extensão JavaScript sem padrões e fazendo o seu Código dependente da ferramenta Bundler. Em um idioma como JavaScript que não requer Compilação, executar uma ferramenta de Bundler parece uma compilação passo, e é frustrante ter que correr um empate depois de cada Código Editar antes que você possa executar o código no seu navegador. Aparecedores normalmente suportam observadores do sistema de arquivos que detectam e editam para qualquer arquivo em um diretório de projeto e automaticamente regenerar os feixes necessários. Com este recurso no lugar, você normalmente pode salvar seu código e depois recarregar imediatamente a janela do seu navegador da web para experimentá-lo. Alguns pacotes também suportam um modo de "substituição do módulo quente"

Para desenvolvedores onde cada vez que um pacote é regenerado, é carregado automaticamente no navegador. Quando isso funciona, é uma experiência mágica para desenvolvedores, mas existem alguns truques. Continuando debaixo do capô para fazê-lo funcionar, e não é adequado para todos os projetos. 17.6

Transpilação com Babel: Babel é uma ferramenta que compila JavaScript escrita usando a linguagem moderna para JavaScript que não usa aquela linguagem moderna com suas características. Porque compila JavaScript ao JavaScript, Babel às vezes é chamado de "transpiler". Babel foi criado para que os desenvolvedores web possam usar os novos recursos de idioma do ES6 e mais tarde enquanto segmentando navegadores da Web que suportavam apenas o ES5. Recursos de linguagem, como o `**` operador de exponenciação e a setaAs funções podem ser transformadas relativamente facilmente em `math.pow()` expressões de função. Outros recursos de linguagem, como a classe `palavra-chave`, requer transformações muito mais complexas e, em geral, A saída de código da Babel não deve ser legível por humanos. Como Bundler Tools, no entanto, Babel pode produzir mapas de origem que mapeiam locais de código transformados de volta aos seus locais originais de fonte. Isso ajuda dramaticamente ao trabalhar com código transformado. Os fornecedores do navegador estão fazendo um trabalho melhor em acompanhar a evolução da linguagem JavaScript, e há muito menos necessidade hoje para compilar funções de seta e declarações de classe. Babel ainda pode ajudar quando você deseja usar os recursos mais recentes, como sublinhado separadores em literais numéricos.

Como a maioria das outras ferramentas descritas neste capítulo, você pode instalar Babel com NPM e execute -o com NPX. Babel lê um .Babelrc arquivo de configuração que informa como você gostaria do seu código JavaScript transformado. Babel define ?Predefinições? que você pode escolher dependendo de quais extensões de idiomas você deseja usar e como. Agressivamente, você deseja transformar os recursos de linguagem padrão. Um de As predefinições interessantes de Babel são para compactação de código por minificação (retirando comentários e espaço em branco, renomeando variáveis ??e assim por diante). Se você usar Babel e uma ferramenta de construção de código, poderá configurar o Código Código para executar automaticamente Babel em seus arquivos JavaScript como. Ele constrói o pacote para você. Nesse caso, essa pode ser uma opção conveniente. Porque simplifica o processo de produção de código executável. Webpack, Por exemplo, suporta um módulo "carregador de babel" que você pode instalar e configurar para executar o Babel em cada módulo JavaScript, pois é empacotado. Mesmo que haja menos necessidade de transformar o JavaScript central para o idioma hoje, Babel ainda é comumente usado para apoiar extensões para o idioma, e descreveremos dois desses idiomas Extensões nas seções a seguir.

17.7 JSX: Expressões de marcação em JavaScript

JSX é uma extensão do JavaScript central que usa a sintaxe no estilo HTML para definir uma árvore de elementos. JSX está mais intimamente associado ao reagireestrutura para interfaces de usuário na web. Em reação, as árvores de Os elementos definidos com JSX são renderizados em um navegador da web como html. Mesmo se você não tiver planos de usar reagir, é

Popularidade significa que é provável que você veja o código que usa o JSX. EsseA seção explica o que você precisa saber para entender dela. (EsseA seção é sobre a extensão da linguagem JSX, não sobre reação, eExplica apenas o suficiente de reagir para fornecer contexto para a sintaxe JSX.) Você pode pensar em um elemento JSX como um novo tipo de expressão de JavaScriptsintaxe.Javascript String literais são delimitados com aspas,e a expressão regular literais é delimitada com barras.No mesmoManeira, a expressão de JSX literais é delimitada com colchetes de ângulo.Aqui estámuito simples:deixe a linha = <hr/>;Se você usar o JSX, precisará usar Babel (ou uma ferramenta semelhante) aCompilar expressões JSX em JavaScript regular.A transformação ésimples o suficiente para que alguns desenvolvedores optem por usar o React sem usarJSX.Babel transforma a expressão JSX nesta declaração de atribuiçãoem uma chamada de função simples:deixe a linha = react.createElement ("hr", nulo);A sintaxe JSX é como HTML e, como elementos HTML, elementos de reaçãoopode ter atributos como estes:Deixe imagem = ;Quando um elemento tem um ou mais atributos, eles se tornam propriedades deUm objeto passou como o segundo argumento para createElement ():Deixe a imagem = react.createElement ("img", {src: "logo.png",

alt: "o logotipo JSX", oculto: verdadeiro}); Como elementos html, elementos JSX podem ter cordas e outros elementos quando criancas. Assim como os operadores aritméticos de JavaScript podem ser usados para escrever expressões aritméticas de complexidade arbitrária, JSX elementos também podem ser aninhados arbitrariamente profundamente para criar árvores de elementos: Deixe a barra lateral = (<div className = "barra lateral"><H1> Título </h1><hr/><p> Este é o conteúdo da barra lateral </p></div>); Expressões regulares de chamadas de função javascript também podem ser aninhadas arbitrariamente profundamente, e essas expressões JSX aninhadas se traduzem em um conjunto de chamadas createElement () aninhadas. Quando um elemento JSX tem Crianças, aquelas criancas (que normalmente são cordas e outras JSX elementos) são aprovados como os terceiros e subsequentes argumentos: Deixe a barra lateral = react.createElement ("Div", {className: "Sidebar"}, // esta chamada externa cria um <div>React.createElement ("h1", null, // este é o primeiro filho do <div>"Título"), // e seu próprio primeiro criancas. React.createElement ("hr", null), // o segundo filho de <div>.React.createElement ("P", NULL, // e o terceiro filho."Este é o conteúdo da barra lateral"));

Erro ao traduzir esta página.

Expressões arbitrárias de JavaScript podem ser incluídas porque a funçãoAs invocações também podem ser escritas com expressões arbitrárias. EsseO código de exemplo é traduzido por Babel para o seguinte:Barra lateral da função (ClassName, Title, Content, Drawline = true) {return react.createElement ("div", {className: ClassName},React.createElement ("h1", null,título),drawline &&React.CreateElement ("HR", NULL),React.createElement ("p", null,contente));}Este código é fácil de ler e entender: os aparelhos encaracolados se foram eO código resultante passa os parâmetros de função que recebem paraReact.createElement () de maneira natural. Observe o truque interessanteque fizemos aqui com o parâmetro da linha drawl e o curtoCircuiting && Operator. Se você ligar para a barra lateral () com apenas trêsArgumentos, então drawline padrões para true, e o quarto argumentopara a chamada CreateElement () externo é o elemento <hr/>. Mas se você passa falsa como o quarto argumento para a barra lateral (), então oQuarto argumento para a chamada de CreateElement ()Falso, e nenhum elemento <hr/> é criado. Este uso do &&O operador é um idioma comum no JSX para incluir ou excluir condicionalmenteum elemento filho, dependendo do valor de alguma outra expressão.(EsseO idioma trabalha com reação porque o react simplesmente ignora crianças que sãofalso ou nulo e não produz nenhuma saída para eles.) Quando você usa expressões JavaScript nas expressões JSX, você énão se limitando a valores simples, como os valores de string e booleanos no

Exemplo anterior.Qualquer valor JavaScript é permitido.Na verdade, é bastante comum na programação do React para usar objetos, matrizes e funções.Considere a seguinte função, por exemplo:// recebeu uma variedade de strings e uma função de retorno de chamada retorna um elemento JSX// representando uma lista HTML com uma matriz de elementos como filho.Lista de funções (itens, retorno de chamada) {retornar (<ul style = {{Padding: 10, Border: "Solid Red 4px"}>{items.map ((item, index) => {<li onclick = {() => retorno de chamada (index)} key = {index}>{item} }));}Esta função usa um objeto literal como o valor do atributo de estilono elemento . (Observe que os aparelhos duplos encaracolados são necessários aqui.) O elemento tem uma única criança, mas o valor dessa criançaé uma matriz.A matriz infantil é a matriz criada usando o mapa ()função na matriz de entrada para criar uma matriz de elementos . (Esse trabalha com reação porque a biblioteca react divina os filhos de um elemento quando os torna.Um elemento com uma criança da matriz é o mesmo que esse elemento com cada um desses elementos da matriz quando crianças.)Finalmente, observe que cada um dos elementos aninhados tem um OnClickAtributo do manipulador de eventos cujo valor é uma função de seta.O código JSXcompila com o seguinte código JavaScript puro (que eu tenhoformatado com mais bonito):Lista de funções (itens, retorno de chamada) {

Retornar react.createElement ("Ul",{style: {preenchimento: 10, borda: "sólido 4px"}},items.map ((item, índice) =>React.createElement ("Li",{OnClick: () => retorno de chamada (índice), chave: índice},item));}Outro uso de expressões de objetos em JSX é com o objeto SPLPINCOperador (§6.10.4) para especificar vários atributos de uma só vez.Suponha que issoVocê se vê escrevendo muitas expressões JSX que repetem umconjunto comum de atributos.Você pode simplificar suas expressões pordefinindo os atributos como propriedades de um objeto e ?espalhando -osem "seus elementos JSX:Seja hebraico = {lang: "He", dir: "rtl"};// Especifique a linguageme direçãoSeja Shalom = ???? ;Babel compila isso para usar uma função _extends () (omitida aqui)que combina esse atributo de nome de classe com os atributos contidosno objeto hebraico:Seja Shalom = React.CreateElement ("Span",_extends ({className:"ênfase"}, hebraico),"\\ u05e9 \\ u05dc \\ u05d5 \\ u05dd");Finalmente, há mais uma característica importante da JSX que não temos

coberto ainda.Como você já viu, todos os elementos JSX começam com um identificadorimediatamente após o suporte do ângulo de abertura.Se a primeira letra dissoidentificador é minúsculo (como tem sido em todos os exemplos aqui), entãoO identificador é passado para createElement () como uma string.Mas se oA primeira letra do identificador é a maçaneta, então é tratada como um realidentificar, e é o valor JavaScript desse identificador que é passadocomo o primeiro argumento a createElement ().Isso significa que o JSXExpressão <Math /> compila com o código JavaScript que passa oObjeto de matemática global para react.createElement ().Para reagir, essa capacidade de passar valores de não coragem como o primeiro argumento acreateElement () permite a criação de componentes.UMO componente é uma maneira de escrever uma expressão JSX simples (com umNome do componente em maiúsculas) que representa um mais complexoexpressão (usando nomes de tags HTML em minúsculas).A maneira mais simples de definir um novo componente no React é escrever umfunção que pega um "objeto de adereços" como seu argumento e retorna um JSXexpressão.Um objeto de adereços é simplesmente um objeto JavaScript que representavalores de atributo, como os objetos que são aprovados como o segundo argumentopara createElement ().Aqui, por exemplo, é outra opinião sobre o nossoFunção da barra lateral ():barra lateral da função (adereços) {retornar (<div><H1> {props.title} </h1>{props.Drawline && <hr/>}<p> {props.content} </p></div>);

}Esta nova função da barra lateral () é muito parecida com a barra lateral anterior ()função.Mas este tem um nome que começa com uma letra maiúscula eleva um único argumento de objeto em vez de argumentos separados.Esse torna um componente de reação e significa que ele pode ser usado no lugar deUm nome de tag html nas expressões JSX:Deixe a barra lateral = <barra lateral title = "Something Snappy"content = "algo sábio"/>;Este elemento <barra lateral/> compila como este:Deixe a barra lateral = react.createElement (barra lateral, {Título: "Something Snappy",Conteúdo: "Algo sábio"});É uma expressão JSX simples, mas quando o react renderiza, ele passará pelo segundo argumento (o objeto de adereços) para o primeiro argumento (oFunção da barra lateral ()) e usará a expressão JSX retornada por essa função no lugar da expressão <apara>.17.8 Verificação do tipo com fluxoO fluxo é uma extensão de linguagem que permite anotar seuCódigo JavaScript com informações de tipo e uma ferramenta para verificar seuCódigo JavaScript (anotado e não anotado) para erros de tipo.ParaUse o fluxo, você começa a escrever código usando a extensão da linguagem de fluxo paraAdicione anotações de tipo.Então você executa a ferramenta de fluxo para analisar seu código e relatar erros de tipo.Depois de consertar os erros e está pronto para

Execute o código, você usa Babel (talvez automaticamente como parte do códigoprocesso de agrupamento) para retirar as anotações do tipo de fluxo fora do seu código.(Uma das coisas legais sobre a extensão da linguagem de fluxo é que lánão é uma nova sintaxe que o fluxo precisa compilar ou transformar.Você usaa extensão da linguagem de fluxo para adicionar anotações ao código e a todosBabel precisa fazer é retirar essas anotações para devolver seu código paraJavaScript padrão.)TypeScript versus FlowO TypeScript é uma alternativa muito popular ao fluxo.TypeScript é uma extensão do JavaScript que adicionaTipos e outros recursos de linguagem.O compilador TypeScript "TSC" compila TypeScriptprogramas em programas JavaScript e, no processoda mesma forma que o fluxo faz.O TSC não é um plug -in Babel: é seu próprio compilador independente.As anotações de tipo simples no TypeScript são geralmente escritas de forma idêntica às mesmas anotações nofluxo.Para uma digitação mais avançada, a sintaxe das duas extensões diverge, mas a intenção e o valor doDuas extensões são as mesmas.Meu objetivo nesta seção é explicar os benefícios das anotações do tipo eAnálise de código estático.Eu farei isso com exemplos com base no fluxo, mas tudo demonstrado aquiTambém pode ser alcançado com o TypeScript com alterações de sintaxe relativamente simples.O TypeScript foi lançado em 2012, antes do ES6, quando JavaScript não tinha uma palavra -chave de classe ouumpara/de loop ou módulos ou promessas.O fluxo é uma extensão de linguagem estreita que adiciona tipoanotações para JavaScript e nada mais.TypeScript, por outro lado, foi muito projetado como umnova linguagem.Como o próprio nome indica, adicionar tipos ao javascript é o principal objetivo do TypeScript,E é a razão pela qual as pessoas o usam hoje.Mas os tipos não são o único recurso que o datilografia adicionaJavaScript: a linguagem TypeScript possui palavras -chave enum e namespace que simplesmente não existem emJavaScript.Em 2020, o TypeScript tem melhor integração com IDEs e editores de código (particularmenteO VSCode, que, como o TypeScript, é da Microsoft) do que o Flow.Por fim, este é um livro sobre JavaScript, e estou cobrindo o fluxo aqui em vez de datilografado porqueNão quero tirar o foco do JavaScript.Mas tudo o que você aprende aqui sobre adicionar tipos aO JavaScript será útil para você se você decidir adotar o TypeScript para seus projetos.Uso o fluxo requer compromisso, mas eu descobri isso para o meioE grandes projetos, o esforço extra vale a pena.Leva tempo extra para adicionarDigite anotações para o seu código, para executar o fluxo toda vez que você editar o código e para corrigir os erros de tipo que ele relata.Mas, em troca, o fluxo

Aplicar uma boa disciplina de codificação e não permitirá que você corte os cantos! Isso pode levar a bugs. Quando eu trabalhei em projetos que usam fluxo, eu ficaram impressionados com o número de erros encontrados em meu próprio código. Ser capaz de corrigir esses problemas antes de se tornarem insetos é um ótimo sentimento e me dá confiança extra de que meu código está correto. Quando comecei a usar o fluxo, achei que às vezes era difícil para entender por que estava reclamando do meu código. Com alguma prática, porém, cheguei a entender suas mensagens de erro e encontrei que geralmente era fácil fazer pequenas alterações no meu código para fazê-lo mais seguro e para satisfazer o fluxo. Eu não recomendo usar fluxo se você ainda sente que está aprendendo JavaScript. Mas uma vez confiante com o idioma, a adição de fluxo aos seus projetos JavaScript empurrará você para levar suas habilidades de programação para o próximo nível. E isso, realmente, é por isso que estou dedicando a última seção deste livro a um tutorial de fluxo: Porque aprender sobre os sistemas de tipo JavaScript oferece um vislumbre de outro nível, ou outro estilo, de programação. Esta seção é um tutorial e não tenta cobrir o fluxo abrangente. Se você decidir tentar fluir, você quase certamente irá acabar gastando tempo lendo a documentação em <https://flow.org>. Sobre por outro lado, você não precisa dominar o sistema de tipo de fluxo antes de você poder começar a fazer uso prático em seus projetos: os usos simples de fluxo descrito aqui levará um longo caminho.

17.8.1 Instalando e executando o fluxo

Como as outras ferramentas descritas neste capítulo, você pode instalar o fluxo ferramenta de verificação de tipo usando um gerenciador de pacotes, com um comando como:

NPM Install -g Flow-Bin ou NPM Instale-Save-devFluxo-barracão.Se você instalar a ferramenta globalmente com -g, poderá executá -lacom fluxo.E se você instalá-lo localmente em seu projeto com--Save-Dev, então você pode executá -lo com o fluxo NPX.Antes de usar o fluxo para fazerVerificação de tipo, a primeira vez que o executa como fluxo -init na raizDiretório do seu projeto para criar um arquivo de configuração .flowconfig.Você pode nunca precisar adicionar nada a este arquivo, mas o fluxo precisaSaiba onde está o seu projeto.Quando você executa o fluxo, ele encontrará todo o código -fonte JavaScript em seuprojeto, mas ele relatará apenas erros de tipo para os arquivos que ?optaramem ?para digitar a verificação adicionando um comentário de A // @flow na parte superior do arquivo.Esse comportamento de opção é importante porque significa que você podeAdote fluxo para projetos existentes e depois comece a converter seu códigoum arquivo de cada vez, sem ser incomodado por erros e avisos emarquivos que ainda não foram convertidos.O fluxo pode encontrar erros em seu código, mesmo que tudo o que você faça seja optarcom um comentário // @flow.Mesmo se você não usar a linguagem de fluxoExtensão e não adicione nenhuma anotações de tipo ao seu código, o tipo de fluxoA ferramenta de verificador ainda pode fazer inferências sobre os valores em seu programae alertá -lo quando você os usa inconsistentemente.Considere a seguinte mensagem de erro de fluxo:Erro
?????????????????????????????????????VariBlebreASSignment.js: 6: 3Não é possível atribuir 1 a I.R
porque:? A propriedade R está ausente no número [1].

2? Let i = {r: 0, i: 1}// o número complexo 0+1i[1] 3? para (i = 0; i <10; i ++) {// oops!A variável loopsubstitui i4?
console.log (i);5?}6? I.R = 1;// O fluxo detecta o erro aquiNesse caso, declaramos a variável i e atribuímos um
objeto a ele.EntãoUsamos i novamente como uma variável de loop, substituindo o objeto.Avisos de fluxoIsso e
sinaliza um erro quando tentamos usar i como se ele ainda tivesse um objeto.(Uma correção simples seria
escrever para (vamos i = 0; fazendo o loopvariável local para o loop.)Aqui está outro erro que o fluxo detecta
mesmo sem anotações de tipo:Erro??size.js: 3:
14Não pode obter x.length porque o comprimento da propriedade está ausente emNúmero [1].1? // @flow2?
Tamanho da função (x) {3? retornar x.Length;4?}[1] 5? Vamos s = tamanho (1000);Flow vê que a função Size ()
leva um único argumento.Não saber o tipo desse argumento, mas pode ver que o argumento éEspera -se ter uma
propriedade de comprimento.Quando vê esse tamanho ()a função sendo chamada com um argumento numérico,
ele sinaliza corretamenteum erro porque os números não têm propriedades de comprimento.17.8.2 Usando
anotações de tipo

Quando você declara uma variável JavaScript, você pode adicionar um tipo de fluxoanotação a ele seguindo o nome da variável com um colón e o tipo: Deixe a mensagem: String = "Hello World"; Deixe a bandeira: booleano = false; Seja n: número = 42; O fluxo saberia os tipos dessas variáveis, mesmo que você não tivesse Anote -os: pode ver quais valores você atribui a cada variável e Ele mantém o controle disso. Se você adicionar anotações de tipo, no entanto, flu conhece o tipo de variável e que você expressou a intenção de que a variável sempre seja desse tipo. Então, se você usar o Anotação do tipo, o fluxo sinalizará um erro se você atribuir um valor de um Tipo diferente para essa variável. As anotações de tipo para variáveis ?? também são particularmente útil se você tende a declarar todas as suas variáveis ?? no topo de uma função antes de serem usados. Anotações de tipo para argumentos de função são como anotações para Variáveis: siga o nome do argumento da função com um colón e o nome do tipo. Ao anotar uma função, você normalmente também adiciona uma anotação para o tipo de retorno da função. Isso vai entre o Perteria próxima e a cinta aberta aberta do corpo da função. Funções que retornam nada use o tipo de fluxo vazio. No exemplo anterior, definimos uma função de tamanho () que esperava uma discussão com uma propriedade de comprimento. Aqui está como poderíamos mudar essa função especificamente para esperar explicitamente que uma matriz é passada para a função, mesmo que a função funcione nesse caso:

Erro??size2.js: 5: 18Não pode chamar o tamanho com a matriz literal ligada a s porque a matrizliteral [1]é incompatível com string [2].[2] 2? Tamanho da função (s: string): número {3? Retornar S. comprimento;4?}{1} 5? console.log (tamanho ([1,2,3]));Usar anotações de tipo com funções de seta também é possível, emborapode transformar essa sintaxe normalmente sucinta em algo mais detalhado:const tamanho = (s: string): número => s.Length;Uma coisa importante a entender sobre o fluxo é que o javascriptvalor nulo tem o tipo de fluxo nulo e o valor JavaScriptindefinido tem o tipo de fluxo vazio.Mas nenhum desses valores é umMembro de qualquer outro tipo (a menos que você o adicione explicitamente).Se você declarararum parâmetro de função para ser uma string, então deve ser uma string e é umerro de passar nulo ou passar indefinido ou omitir o argumento(que é basicamente a mesma coisa que a passagem indefinida):Erro??size3.js: 3: 18Não pode chamar tamanho com nulo ligado a s porque nulo [1] éincompatívelcom string [2].1? // @flow[2] 2? const tamanho = (s: string): número => s.Length;[1] 3? console.log (tamanho (nulo));

Se você deseja permitir nulo e indefinido como valores legais para um Argumento de variável ou função, basta prefixar o tipo com uma perguntamarca.Por exemplo, use? String ou? Número em vez de string ou número.Se mudarmos nossa função de tamanho () para esperar um argumento dedigite? string, então o fluxo não se queixa quando passamos nulos paraa função.Mas agora tem outra coisa para reclamar:Erro??size4.js: 3: 14Não pode obter o comprimento S., porque o comprimento da propriedade está ausente emnulo ou indefinido [1].1? // @flow[1] 2? Tamanho da função (S :? String): Número {3? Retornar S. comprimento;4?}5? console.log (tamanho (nulo));O que o fluxo está nos dizendo aqui é que não é seguro escrever s. comprimentoPorque, neste lugar em nosso código, S pode ser nulo ou indefinido,e esses valores não têm propriedades de comprimento.É aqui que fluiGaranta que não cortamos nenhum canto.Se um valor puder ser nulo, fluaWill insistirá que verificamos esse caso antes de fazer qualquer coisa quedepende do valor não ser nulo.Nesse caso, podemos corrigir o problema alterando o corpo da função do seguinte modo:Tamanho da função (S :? String): Número { // Neste ponto do código, S pode ser uma string ou nulaou indefinido.if (s === null || s === indefinido) {

// Neste bloco, Flow sabe que S é nulo ou indefinido.retornar -1;} outro {// e neste bloco, o Flow sabe que S é uma string.Retornar S. Length;}}Quando a função é chamada pela primeira vez, o parâmetro pode ter mais de um tipo.Mas adicionando código de verificação de tipo, criamos um bloco dentro do Código onde o Flow sabe com certeza que o parâmetro é uma string.Quando utilizamos o comprimento S. dentro desse bloco, o fluxo não reclama.ObservaçãoEsse fluxo não exige que você escreva código detalhado como este.Fluxo também ficaria satisfeito se apenas substituíssemos o corpo do tamanho ()função com retorno s?S. Length: -1;.A sintaxe do fluxo permite um ponto de interrogação antes de qualquer especificação de tipoindique que, além do tipo especificado, nulo e indefinidosão permitidos também.Pontos de interrogação também podem aparecer após um parâmetronome para indicar que o próprio parâmetro é opcional.Então, se mudarmosA declaração dos parâmetros S de S :? String para S?:string, isso significaria que não há problema em chamar tamanho () sem argumentos(ou com o valor indefinido, que é o mesmo que omitindo), mas que se o chamarmos de um parâmetro que não seja indefinido, então issoO parâmetro deve ser uma string.Nesse caso, Null não é um valor legal.Até agora, discutimos tipos primitivos string, número, booleano,nulo e vazio e demonstraram como você pode usar usá -los com declarações variáveis, parâmetros de função e retorno de funçãovalores.As subseções a seguir descrevem alguns tipos mais complexos

suportado por fluxo. 17.8.3 Tipos de classe Além dos tipos primitivos que o fluxo conhece, também sabe sobre todas as classes internas do JavaScript e permite que você use a classenome como tipos. A função a seguir, por exemplo, usa o tipo anotações para indicar que deve ser invocado com um objeto de uma data e um objeto regexp:// @fluxo// retorna true se a representação ISO do especificado data// corresponde ao padrão especificado, ou falso caso contrário.// por exemplo: const IsTodayChristmas = DataMatches (new Date (), /\d{4}-12-25t/); Função de exportação Datematches (D: Data, P: Regexp): Boolean { retorno p.test (d.toISOString ());} Se você definir suas próprias aulas com a palavra -chave da classe, essas classes tornam-se automaticamente tipos de fluxo válidos. Para fazer isso funcionar, No entanto, o fluxo exige que você use anotações de tipo na classe. Em particular, cada propriedade da classe deve ter seu tipo declarado. Aqui é uma classe numérica complexa simples que demonstra o seguinte:// @fluxo Exportar o complexo da classe padrão // O fluxo requer uma sintaxe de classe estendida que inclua anotações de tipo// para cada uma das propriedades usadas pela classe.i: número;r: número; estático i: complexo; construtor (r: número, i: número) {

// Quaisquer propriedades inicializadas pelo construtor devem tem tipo de fluxo// anotações acima.this.r = r;this.i = i;}Adicione (isso: complexo) {retornar novo complexo (this.r + that.r, this.i + that.i);}}// Esta tarefa não seria permitida por fluxo se houvessemão a// Tipo anotação para i dentro da classe.Complex.i = novo complexo (0,1);17.8.4 Tipos de objetosO tipo de fluxo para descrever um objeto se parece muito com um objeto literal,Exceto que os valores da propriedade são substituídos pelos tipos de propriedades.Aqui, para exemplo, é uma função que espera um objeto com x e y numéricas propriedades:// @fluxo// recebeu um objeto com propriedades X e Y numéricas, retorneo// distância da origem ao ponto (x, y) como um número.Exportar distância padrão da função (ponto: {x: número,y: número}): número {retornar math.hypot (Point.x, Point.Y);}Neste código, o texto {x: número, y: número} é um tipo de fluxo, apenas como string ou data é.Como em qualquer tipo, você pode adicionar uma perguntaMarque na frente para indicar que nulo e indefinido também deve ser permitido.

Erro ao traduzir esta página.

"Seattle": {longitude: 47.6062, latitude: -122.3321},// TODO: Se houver outras cidades importantes, adicione elas aqui.};exportar locações de cidades padrão;17.8.5 Aliases do tipoObjetos podem ter muitas propriedades e o tipo de fluxo que descreveEsse objeto será longo e difícil de digitar.E mesmo relativamenteTipos de objetos curtos podem ser confusos porque se parecem muito com objetos literais.Uma vez que vamos além dos tipos simples, como número e?string, muitas vezes é útil poder definir nomes para o nosso fluxotipos.E, de fato, o Flow usa a palavra -chave TIPE para fazer exatamente isso.Siga a palavra -chave de tipo com um identificador, um sinal igualTipo de fluxo.Depois de fazer isso, o identificador será um alias para o tipo.Aqui, por exemplo, é como poderíamos reescrever a distância ()função da seção anterior com um ponto explicitamente definido tipo:// @fluxoPonto de tipo de exportação = {x: número,Y: Número};// Dado um objeto de ponto, retorne sua distância da origemExportar distância padrão da função (ponto: ponto): número {retornar math.hypot (Point.x, Point.Y);}Observe que este código exporta a função distance () e também exporta o tipo de ponto.Outros módulos podem usar o tipo de importação

Ponto de './distance.js' se eles querem usar esse tipo de definição. Lembre-se, porém, que esse tipo de importação é um fluxo extensão do idioma e não uma diretiva de importação JavaScript real. Tipos de importações e exportações são usadas pelo verificador do tipo fluxo, mas como todas as outras extensões de linguagem de fluxo, elas são despojadas do código antes de correr. Finalmente, vale a pena notar que, em vez de definir um nome para um fluxo tipo de objeto que representa um ponto, provavelmente seria mais simples e mais limpo para definir apenas uma classe pontual e usar essa classe como o tipo.

17.8.6 Tipos de matriz O tipo de fluxo para descrever uma matriz é um tipo de composto que também inclui o tipo de elementos da matriz. Aqui, por exemplo, é uma função que espera uma variedade de números e o erro que o fluxo relata se você tentar chamar a função com uma matriz que possui elementos não numéricos:

Erro

??mádia.js: 8: 16 Não pode chamar a média com a matriz literal ligada a dados porque String [1] é incompatível com o número [2] no elemento da matriz.[2] 2? Média da função (Dados: Array <número>) {3? deixe soma = 0;4? Para (deixe x de dados) soma += x;5? Retornar soma / data.length;6?}7?[1] 8? média ([1, 2, "três"]); O tipo de fluxo para uma matriz é seguido pelo tipo de elemento em suportes de ângulo. Você também pode expressar um tipo de matriz seguindo o

Erro ao traduzir esta página.

função cinza (nível: número): cor {retornar [nível, nível, nível, 1];}Função Fade ([R, G, B, A]: Cor, Factor: Número): Cor {retornar [r, g, b, a/fator];}Seja [r, g, b, a] = desbotamento (cinza (75), 3);Agora que temos uma maneira de expressar o tipo de matriz, vamos voltar aa função size () de anteriores e modificá -lo para esperar uma matrizargumento em vez de um argumento de string.Queremos que a função seja capazPara aceitar uma matriz de qualquer comprimento, um tipo de tupla não é apropriado.MasNão queremos restringir nossa função a trabalhar apenas para matrizes ondeTodos os elementos têm o mesmo tipo.A solução é o tipoArray <fixed>://
@fluxoTamanho da função (S: Array <fixed>): Número {Retornar S. Length;}console.log (tamanho ([1, true, "três"]));O tipo de elemento misto indica que os elementos da matriz podem ser de qualquer tipo.Se nossa função realmente indexou a matriz e tentouPara usar qualquer um desses elementos, o fluxo insistiria que usamos o tipo deverificações ou outros testes para determinar o tipo do elemento antesexecutando qualquer operação insegura nela.(Se você estiver disposto a desistirVerificação de tipo, você também pode usar qualquer um em vez de misto: permite que vocêfazer o que quiser com os valores da matriz sem garantirque os valores são do tipo que você espera.)

17.8.7 Outros tipos parametrizados Vimos isso quando você anota um valor como uma matriz, fluexige que você também especifique o tipo de elementos de matriz dentro do ânguloSuportes. Este tipo adicional é conhecido como parâmetro de tipo e matriznão é a única classe JavaScript que é parametrizada. A classe set de JavaScript é uma coleção de elementos, como uma matriz é, eVocê não pode usar o conjunto como um tipo por si só, mas precisa incluir um tipoparâmetro entre colchetes de ângulo para especificar o tipo de valorescontido no conjunto.(Embora você possa usar misto ou qualquer um, se o conjuntopode conter valores de vários tipos.) Aqui está um exemplo:// @fluxo// retorna um conjunto de números com membros que são exatamente duas vezes esses// do conjunto de números de entrada.função dupla (s: set <número>): set <número> {Seja duplado: set <MumM> = new Set ();para (vamos n de s) dobrar.Add (n * 2);o retorno dobrou;}console.log (duplo (novo conjunto ([1,2,3])));// imprime "Set {2, 4,6} "O mapa é outro tipo parametrizado.Nesse caso, existem dois tiposparâmetros que devem ser especificados;o tipo de chaves e os tipos deos valores:// @fluxoimportar tipo {color} de "./color.js";Deixe Colornames: mapa <string, cor> = novo mapa ([[{"vermelho": [1, 0, 0, 1]}, {"verde": [0, 1, 0, 1]}],

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Garanta que um objeto ou matriz não possa ser modificado (verObject.freeze () em §14.2 se você deseja objetos verdadeiros somente leitura), masPorque permite que você pegue bugs causados ??por não intencionalmodificações.Se você escrever uma função que leva um objeto ou matrizargumento e não muda nenhuma das propriedades do objeto ou oelementos da matriz, então você pode anotar o parâmetro de função comUm dos tipos somente leitura do Flow.Se você fizer isso, o fluxo relatará umErro se você esquecer e modificar acidentalmente o valor de entrada.Aqui estãoDois exemplos:// @fluxotipo de ponto = {x: número, y: número};// Esta função leva um objeto de ponto, mas promete nãomodifique -oDistância da função (p: \$ readonly <point>): número {Return Math.HyPot (P.X, P.Y);}Seja p: ponto = {x: 3, y: 4};Distância (P) // => 5// Esta função leva uma variedade de números que não vaimodificarMédia da função (Dados: \$ readonlyArray <número>): Número {deixe soma = 0;para (vamos i = 0; i <data.length; i ++) soma+= dados [i];retorno soma/data.length;}Deixe os dados: Array <Number> = [1,2,3,4,5];média (dados) // => 317.8.9 Tipos de funçãoVimos como adicionar anotações de tipo para especificar os tipos de um

Os parâmetros da função e seu tipo de retorno. Mas quando um dosParâmetros de uma função é uma função, precisamos ser capazes deEspecifique o tipo desse parâmetro de função. Para expressar o tipo de função com fluxo, escreva os tipos de cada parâmetro, separe -os com vírgulas, inclua -os entre parênteses, e depois siga isso com um tipo de seta e tipo de retorno da função. Aqui está uma função de exemplo que espera ser aprovada um retorno de chamada função. Observe como definimos um alias de tipo para o tipo de função de retorno de chamada:// @fluxo// O tipo da função de retorno de chamada usada em fetchText ()abaixoTipo de exportação

```
FetchTextCallback = (? Erro ,? Número ,? String) =>vazio;exportar função padrão fetchText (url: string, retorno de chamada:FetchTextCallback) {deixe status = nulo;buscar (url).then (resposta => {status = resposta.status;RETORNO DE REPORTAÇÃO.TEXT ()}).Then (Body => {retorno de chamada (nulo, status, corpo);}).catch (erro => {retorno de chamada (erro, status, nulo);});}17.8.10 Tipos de sindicatos
```

Vamos retornar mais uma vez à função Size (). Na verdade não faz sentido ter uma função que não faça nada além de devolver o comprimento de uma matriz. Matrizes têm uma propriedade de comprimento perfeitamente boa para que. Mas size () pode ser útil se puder levar qualquer tipo de objeto de coleta (uma matriz ou conjunto ou mapa) e retorne o número de elementos da coleção. Em JavaScript regular não é fácil de escrever uma função de tamanho () como essa. Com fluxo, precisamos de uma maneira para expressar um tipo que permite matrizes, conjuntos e mapas, mas não permite valores de qualquer outro tipo. Tipos de chamadas de fluxo como esse tipo de sindicato e permite expressá-los simplesmente listando os tipos desejados e separando-os com verticalPersonagens de bar://

```
@fluxoTamanho da função (coleção:Array <fixed> | set <fixed> | mapa <mixed, mixed>): número {if (Array.isArray(coleção)) {Return collection.Length;} outro {Return collection.size;}}tamanho ([1, verdadeiro, "três"]) + tamanho (novo conjunto ([true, false])) // => 5 Os tipos de sindicatos podem ser lidos usando a palavra "ou" - "uma matriz ou um conjunto ou umMapa ? - o fato de essa sintaxe de fluxo usa a mesma barra verticalO personagem como JavaScript ou operadores é intencional. Vimos anteriormente que colocar um ponto de interrogação antes de um tipo permite nulos e valores indefinidos. E agora você pode ver isso? prefixo é
```

Simplesmente um atalho para adicionar um sufixo | nulo | vazio a um tipo. Em geral, quando você anota um valor com um tipo de união, o fluxo não vai permitir que você use esse valor até fazer testes suficientes para descobrir qual é o tipo do valor real. No exemplo () exemplo, apenas observamos, precisamos verificar explicitamente se o argumento é uma matriz. Antes de tentarmos acessar a propriedade de comprimento do argumento. Observação: que não precisamos distinguir um argumento definido de um mapa argumento, no entanto: ambas essas classes definem uma propriedade de tamanho, então o código na cláusula else é seguro, desde que o argumento não seja um variedade.

17.8.11 tipos enumerados e discriminados

O fluxo permite que você use literais primitivos como tipos que consistem em um único valor. Se você escrever, deixe x: 3; então o fluxo não permitirá que você atribua qualquer valor a essa variável que não seja 3. Não é frequentemente útil para definir tipos que têm apenas um único membro, mas uma união de tipos literais podem ser úteis. Você provavelmente pode imaginar um uso para tipos assim, por exemplo:

```
Digite Answer = "Sim" | "não";
```

Digit de tipo = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;

Se você usa tipos compostos de literais, você precisa entender isso apenas. Valores literais são permitidos:

```
Deixe A: Responder = "Sim".ToLowerCase();
```

// Erro: não posso atribuir string para responder

```
Seja D: Digit = 3+4;
```

// Erro: não posso atribuir

Erro ao traduzir esta página.

Outro uso importante para tipos literais é a criação de discriminadossindicatos.Quando você trabalha com tipos de sindicatos (composto de realmentetipos diferentes, não de literais), você normalmente precisa escrever código para discriminar entre os tipos possíveis.Na seção anterior, nósescreveu uma função que poderia levar uma matriz ou um conjunto ou um mapa como seuargumento e tive que escrever código para discriminar a entrada da matriz de set ouEntrada de mapa.Se você deseja criar uma união de tipos de objetos, você pode fazerEsses tipos fáceis de discriminar usando um tipo literal em cada um dosos tipos de objetos individuais.Um exemplo deixará isso claro.Suponha que você esteja usando um tópico de trabalhadorno nó (§16.11) e estão usando PostMessage () e "Mensagem"eventos para enviar mensagens baseadas em objetos entre o tópico principal eo tópico do trabalhador.Existem vários tipos de mensagens que oO trabalhador pode querer enviar para o tópico principal, mas gostaríamos de escrever umTipo de união de fluxo que descreve todas as mensagens possíveis.Considere issocódigo:// @fluxo// O trabalhador envia uma mensagem desse tipo quando é feito// reticulando as splines que o enviamos.Tipo de exportação ResultMessage = {messageType: "resultado",Resultado: Array <CeticuledSpline>, // assume que esse tipo édefinido em outro lugar.};// O trabalhador envia uma mensagem desse tipo se seu código falhoucom uma exceção.Tipo de exportação errorMessage = {MessageType: "Erro",Erro: erro,};

```
// O trabalhador envia uma mensagem desse tipo para relatar o usoEstatística.Tipo de exportação
StatisticsMessage = {messageType: "estatísticas",splinesReticulules: Number,splinespersegund: número};//
Quando recebermos uma mensagem do trabalhador, será umWorkerMessage.Tipo de exportação workermessage
= resultMessage |ErrorMessage |StatisticsMessage;// O encadeamento principal terá uma função de manipulador
de eventos queé passado// Um ??Workermessage.Mas porque definimos cuidadosamente cada umdo// Tipos de
mensagem para ter uma propriedade MessageType com umtipo literal,// O manipulador de eventos pode
facilmente discriminar entre osMensagens possíveis:função handleMessageFromReticulator (Mensagem:
WorkerMessage){if (message.MessageType === "resultado") {// apenas o resultadoeste valor// Então Flow sabe
que é seguro usarmessage.Result aqui// e fluxo irá reclamar se você tentar usar qualquer
outropropriedade.console.log (message.result);} else if (message.MessageType === "Error") {// Somente
errorMessage tem uma propriedade Messagetype comvalor "erro"// SONTE SABE QUE É SEGURO USAR
MENSAGEM.Error aqui.lança message.error;} else if (message.MessageType === "Stats") {// Somente
StatisticsMessage possui uma propriedade Messagetypecom valor "estatísticas"// então sabe que é seguro
usarmessage.splinespersegund aqui.console.info (message.splinespersecond);}
```

Erro ao traduzir esta página.

ÍndiceSímbolos!(Boolean não operador), lógico não (!)! = (Operador de desigualdade não estrito)Expressões relacionais, igualdade e operadores de desigualdadeDigite conversões, conversões especiais de operadoras de casos! == (operador de desigualdade)valores booleanos, valores booleanosVisão geral de operadores de igualdade e desigualdadeComparação de strings, trabalhando com strings"(citações duplas), literais de cordas\$(sinal de dólar), identificadores e palavras reservadas% (Operador Modulo), aritmética em JavaScript, expressões aritméticas& (bitwise e operador), operadores bitwise&& (booleano e operador), valores booleanos, lógicos e (&&)'(citações únicas), literais de cordas* (operador de multiplicação), aritmética em javascript, expressões eOperadores, expressões aritméticas** (operador de exponenciação), aritmética em JavaScript, aritméticaExpressões

+ (mais sinal)Operador de adição e atribuição (+=), atribuição comOperaçãoOperador de adição, aritmética em JavaScript, o operador +concatenação de string, literais de string, trabalhando com strings, o +OperadorDigite conversões, conversões especiais de operadoras de casosOperador aritmético unário, operadores aritméticos unários++ (operador de incremento), operadores aritméticos unários, (operadora de vírgula), o operador de vírgula (,)- (sinal de menos)Operador de subtração, aritmética em JavaScript, aritméticaExpressõesOperador aritmético unário, operadores aritméticos unários- (Operador de decrescimento), operadores aritméticos unários.(Operador de pontos), um tour de JavaScript, Propriedades de consulta e definição/ (operadora de divisão), aritmética em JavaScript, expressões aritméticas/ * */ personagens, comentários// (barras duplas), um tour de JavaScript, um tour de JavaScript,ComentáriosGráficos 3D, gráficos em uma <VAS>;(Semicolon), semicolons opcionais semicolons-opcionais<(menos que o operador)

Visão geral dos operadores de comparaçãoComparação de strings, trabalhando com stringsDigite conversões, conversões especiais de operadoras de casos<< (Operador esquerdo de mudança), operadores bitwise<= (menor ou igual ao operador)Visão geral dos operadores de comparaçãoComparação de strings, trabalhando com stringsDigite conversões, conversões especiais de operadoras de casos= (operador de atribuição), um tour de JavaScript, igualdade e desigualdadeOperadores, expressões de atribuição== (operador de igualdade)Visão geral de operadores de igualdade e desigualdadeDigite conversões, visão geral e definições, conversões elgualdade, conversões especiais de operadoras de casos==== (operador de igualdade rigoroso)valores booleanos, valores booleanosVisão geral de operadores de igualdade e desigualdadeComparação de strings, trabalhando com stringsDigite conversões, visão geral e definições, conversões elgualdade=> (setas), um tour de JavaScript, semicolons opcionais, setaFunções> (maior que o operador)

Visão geral dos operadores de comparaçãoComparação de strings, trabalhando com stringsDigite conversões, conversões especiais de operadoras de casos> = (maior ou igual ao operador)Visão geral dos operadores de comparaçãoComparação de strings, trabalhando com stringsDigite conversões, conversões especiais de operadoras de casos>> (Mudar bem com o operador de sinal), operadores bitwise>>> (desligue bem com o operador de preenchimento zero), operadores bitwise?.(Operador de acesso condicional), um tour de JavaScript, funçãoInvocação?: (Operador condicional), o operador condicional (? :)?(Operador primeiro definido), primeiro definido (??)[] (Suportes quadrados), um passeio de JavaScript, trabalhando com strings,Objetos e matrizes inicializadores, consultas e configurações de propriedades, leiturae escrevendo elementos de matriz, cordas como matrizes\ (barragem), seqüências de literais de string-escape em literais de string\ n (newline), literais de string, sequências de fuga em literais de string\ U (UNICODE CARACHAR ESCAPE), UNICODE ESCAPE SEQUÊNCIÓES, ESCAPESequências em literais de string\ xa9 (símbolo de direitos autorais), sequências de fuga em literais de cordas\ `(backtick ou apóstrofe) Escape, sequências de fuga em literais de cordas^ (Operador Bitwise XOR), operadores bitwise

_ (sublinhado), identificadores e palavras reservadas_ (sublinhado, como separadores numéricos), literais de ponto flutuante` (backtick), literais de cordas, literais de modelo{} (aparelho encaracolado), um passeio de JavaScript, objeto e matriz inicializadores|| (Booleano ou operador), valores booleanos, lógicos ou (||)~ (bitwise não operador), operadores bitwise? (Bitwise ou operador), operadores bitwise? (Operador espalhado), operador espalhado, o operador de espalhamento, oEspalhe o operador para chamadas de função, iteradores e geradoresUMClasses abstratas, hierarquias de classe e classes abstratas-Summaryacelerômetros, APIs de dispositivo móvelPropriedades do acessador, getters de propriedades e settersMétodo addEventListener (), addEventListener ()Operador de adição (+), aritmético em JavaScript, o operador +Recursos avançadosextensibilidade do objeto, extensibilidade do objetoVisão geral da metaprogramaçãoAtributos da propriedade, atributos de propriedades atributosAtributo do protótipo, o atributo do protótipoObjetos de proxy, objetos de proxy-proxy invariantesReflita API, a API refletida-refletir API

Tags de modelos, tags de modelo de modelo tagssímbolos bem conhecidosSímbolos de correspondência de padrões, símbolos de correspondência de padrõesSymbol.asynciterator, símbolo.iterator eSymbol.asynciteratorSymbol.hasinsance, símbolo.hasinstanceSymbol.iscoNcatsPreadable, symbol.iscoNcatsPreadableSymbol.iterator, símbolos conhecidosSymbol.spécies, símbolo.species-symbol.speciesSymbol.Toprimitive, símbolo.ToprimitivoSymbol.ToStringTag, symbol.ToStringTagSymbol.UNSCOPABLES, SYMBOL.UNSCOPABLESalfabetização, comparando stringscaracteres âncora, especificando a posição de correspondênciaApostróficos, literais de cordasMétodo Aplicar (), Invocação Indireta, Métodos Call () e Aplicar ()Método arc (), curvasMétodo arcto (), curvasargumentostipos de argumento, tipos de argumentoObjeto de argumentos, o objeto de argumentosDefinição de termo, funções

Destructar a função argumentos nos parâmetros, destruindoFunção Argumentos na função de destruturação de parâmetrosArgumentos nos parâmetroslistas de argumentos de comprimento variável, parâmetros de descanso e variável-Listas de argumentos de comprimentoOperadores aritméticos, um passeio de JavaScript, aritmético em JavaScript-Aritmética em javascript, operadoras de expressões aritméticas-bitwiseíndice de matriz, leitura e escrita de elementos de matrizMétodos do iterador da matrizcada () e alguns (), cada () e outros ()filtro (), filtro ()Find () e FindIndex (), Find () e FindIndex ()foreach (), foreach ()map (), map ()Visão geral dos métodos do iterador de matrizReduce () e ReduceRight (), Reduce () e ReduceRight ()matriz literais, objetos e matrizes inicializadores, matrizes literaisArray () construtor, o construtor da matriz ()Array.from () função, Array.from (), funções estáticas da matrizFunção de Array.ISArray (), Funções de Array estáticoArray.of () função, array.of (), funções estáticas de matrizArray.prototype, matrizes, objetos semelhantes a matrizesMétodo Array.sort (), funciona como valores

Método ArrayBuffer (), analisando os corpos de respostas adicionando e excluindo, adicionando e excluindo elementos de matriz comprimento da matriz, comprimento da matriz Métodos de matriz Adicionando matrizes, adicionando matrizes com concat () Matriz para conversões de string, matriz para conversões de string Arrays achatados, achatando matrizes com plano () e plangmap () Aplicação genérica de matrizes Métodos de iterador, Matriz de Matray Methods-Reduce () e Reduteright () Visão geral dos métodos de matriz Métodos de pesquisa e classificação, pesquisa de matrizes-reverter() pilhas e filas, pilhas e filas com push (), pop (), shift (), e não dividido () Funções de matriz estática, funções de matriz estáticas subarrays, subarrays with slice (), splice (), preenche () e CopyWithin () Objetos semelhantes a matrizes, objetos semelhantes a matrizes-objetos semelhantes a matrizes Matrizes associativas, introdução a objetos, objetos como associativa Matrizes Criando, criando matrizes-array.from () Definição de termo, visão geral e definições

Expressões inicializador, um tour de JavaScript, objeto e matrizInicializadoresmatrizes de iteração, matrizes de iteraçãoMatrizes multidimensionais, matrizes multidimensionaisInicializadores aninhados, objetos e matrizesVisão geral de, um tour de JavaScript, matrizesprocessamento com funções, processamento de matrizes com funçõesLEITURA E ESCREVER ELEMENTOS DE ARAY, LEITURA E ESCREVER ARRAYElementosMatrizes esparsas, matrizes esparsascordas como matrizes, cordas como matrizesmatrizes digitadasCriando, criando matrizes digitadasDataView e Endianness, DataView e Endiannessmétodos e propriedades, métodos de matriz digitados ePropriedadesVisão geral de matrizes digitadas e dados bináriosTipos de matriz digitados, tipos de matriz digitadosUsando, usando matrizes digitadasFunções de seta, um tour de JavaScript, funções definidoras, flechasFunçõessetas ($=>$), um tour de javascript, semicolons opcionais, setaFunçõesCaracteres de controle ASCII, o texto de um programa JavaScript

Afirações, um passeio de JavaScriptOperador de atribuição (=), um tour de JavaScript, igualdade e desigualdadeOperadores, expressões de atribuiçãoMatrizes associativas, introdução a objetos, objetos como associativaMatrizesAssociatividade, Associatividade do OperadorPalavra-chave assíncrona, ASYNC e AWAIT-IMPLEMENTATION DetalhesProgramação assíncrona (ver também nó)assíncrono e aguardar palavras-chave, assíncronas e implementaçãoDetalhesiteração assíncronageradores assíncronos, geradores assíncronositeradores assíncronos, iteradores assíncronosPara/aguarda loops, iteração assíncrona com/aguardar, Iteração assíncronaimplementação, implementando iteradores assíncronos-Implementando iteradores assíncronosretornos de chamadareturnos de chamada e eventos no nó, retornos de chamada e eventos no nóDefinição de termo, programação assíncrona comRetornos de chamadaeventos, eventosEventos de rede, eventos de redeTimers, temporizadores

Definição de termo, javascript assíncronoJavascript Suporte para JavaScript assíncronoPromessasencadear promessas, encadear promessas de promessaslidar com erros com, lidar com erros com promessas, mais sobrePromessas e erros-a captura e finalmente métodosfazendo promessas, fazendo promessas-promessas em sequênciaVisão geral de, promessasoperações paralelas, promessas em paraleloPromessas em sequência, promessas em promoções de sequência emSequênciaResolvendo promessas, resolvendo promessas-mais nas promessase errosretornando dos retornos de chamada de promessa, o problema e finalmenteMétodsterminologia, lidando com erros com promessasUsando, usando erros de manipulação de promessas com promessasAPIs de áudioConstrutor de áudio (), o construtor Audio ()Visão geral de APIs de áudioAPI Webaudio, a API WebaudioAguarde detalhes-chave, assíncronos e detalhes de implementaçãoaguardar operador, o operador aguardar

Erro ao traduzir esta página.

Booleano ou operador (||), valores booleanos, lógicos ou (||)valores booleanos, valores
booleanos-booleanosFunção booleana (), conversões explícitasquebrar declarações, quebreFerramentas de
desenvolvimento do navegador, explorando JavaScriptHistória de navegaçãoGerenciando com eventos de
hashchange, gerenciamento de história comHashChange EventsGerenciando com PushState (), Gerenciamento
de História com PushState ()Visão geral da história de navegaçãoAlgoritmo de clone estruturado, gerenciamento
de história com pushState ()Classe de buffer (nó), buffersCAPI de cache, aplicativos da Web progressivos e
trabalhadores de serviçocalendários, datas de formatação e horáriosCall () Método, Invocação Indireta, Métodos
Call () e Aplicação ()retornos de chamadareturnos de chamada e eventos no nó, retornos de chamada e eventos
no nóDefinição de termo, programação assíncrona com retornos de chamadaeventos, eventosEventos de rede,
eventos de redeTimers, temporizadores

Erro ao traduzir esta página.

manipulação de pixels, manipulação de pixelsSensibilidade ao caso, o texto de um programa JavaScriptPegue cláusulas, tentativas/finalmente declarações de Catch/FinalmentePegue declarações, aulas de erroMétodo .Catch (), a captura e finalmente métodos-a captura e finalmenteMétodosClasses de personagens (expressões regulares), classes de personagensHistogramas de frequência do caractere, exemplo: frequência do personagemHistogramas-SummaryMétodo Charat (), Strings como matrizesfunção checkScope (), fechamentosProcessos Infantis (Nó), Trabalhando com Processos Infantis-Fork ()benefícios de trabalhar com processos infantisExec () e Execfile (), Exec () e Execfile ()execsync () e execfilesync (), execsync () e execfilesync ()fork (), garfo ()Opções, Execsync () e ExecFileSync ()Spawn (), Spawn ()Declaração de classe, classePalavra-chave da classe, aulas com a palavra-chave-exemplo de classe: um complexoClasse de númeroMétodos de classe, métodos estáticosclasses

Adicionando métodos às classes existentes, adicionando métodos aos existentesClassesclasses e construtores, classes e construtores-Propriedade do construtorpropriedade construtora, a propriedade do construtorconstrutores, identidade de classe e instância, construtores,Identidade de classe e instanceofExpressão, classes e construtores do New.Targetclasses e protótipos, classes e protótiposAulas com palavra-chave de classe, classes com a palavra-chave de classe-Exemplo: uma classe de números complexosExemplo de classe de número complexo, exemplo: um complexoExemplo de classe de número: uma classe de números complexosgetters, setters e outros formulários de método, getters, setters eOutros formulários de métodoCampos públicos privados e estáticos, públicos, privados e estáticosCampos-campos, áreas públicas, privadas e estáticasMétodos estáticos, métodos estáticosprogramação modular com, módulos com classes, objetos eFechamentosNomeação, palavras reservadasVisão geral de, visão geral e definições, classessubclasseshierarquias de classe e classes abstratas, hierarquias de classe eClasses abstratas-Summary

delegação versus herança, delegação em vez de Herança Visão geral de, subclasses e protótipos, subclasses e protótipos com cláusula de estendências, subclasses com extensões e super-Subclasses com extensões e super-JavaScript do lado do cliente, JavaScript em navegadores da Web Armazenamento do lado do cliente, armazenamento recorte, recorte Método mais próximo (), selecionando elementos com seletores CSS fechamentos Combinando com getters e setters de propriedades, fechamentos erros comuns, fechamentos Definição de termo, fechamento Regras de escopo lexicais e fechamentos programação modular com, automatizando baseado em fechamento Modularidade Fechamentos de funções aninhadas, fechamentos Estado privado compartilhado, fechamentos Bundling de código, agrupamento de código Exemplos de código Comentário Sintaxe, um tour pelo JavaScript obtenção e uso, código de exemplo

Erro ao traduzir esta página.

Funções definidas por, a API do console-a API do consolesuporte para, a API do consoleFunção Console.log (), Hello World, Saída do Consolepalavra -chave const, declarações com let e constconstantesdeclarando, visão geral e definições, declarações com let econst, const, let e varDefinição de termo, declaração e atribuição variáveisNomeação, palavras reservadasconstrutoresArray () construtor, o construtor da matriz ()Construtor de áudio (), o construtor Audio ()Aulas e, aulas e construtores-a propriedade do construtorPropriedade do construtor, a propriedade do construtor-Propriedade do construtorconstrutores, identidade de classe e instância, construtores,Identidade de classe e instanceofExpressão, classes e construtores do New.TargetInvocação construtora, invocação construtoraDefinição de termo, funçõesexemplos de, criando objetos com novoFunction () construtor, o construtor function ()SET () construtor, a classe Set

Erro ao traduzir esta página.

Folhas de estilo CSSEstilos CSS comuns, Scripts CSSEstilos computados, estilos computadosAnimações e eventos CSS, animações e eventos CSSClasses CSS, aulas CSSSyntax do seletor CSS, selecionando elementos com seletores CSSEstilos embutidos, estilos embutidosConvenções de nomeação, estilos em linhafolhas de estilo de script, folhas de estilo de scriptObjeto CSSStyleDeclaration, estilos em linhaBrace Curly ({}), um passeio de JavaScript, Objeto e Array Inicializadoresmoeda, números de formataçãocurvas, curvasDPropriedades de dados, getters de propriedades e settersClasse DataView, DataView e EndianessTipo de data, visão geral e definições, datas e horáriosdatas e temposdata aritmética, data aritméticaFormatação e análise Data de cadeias, formatação e análiseCordasformatação para internacionalização, formatação e datas

Datas e tempos de formataçãoTimestamps de alta resolução, registro de data e horaVisão geral de, datas e horários, datas e horáriosTIMESTAMPSTIMESTAMPSDeclarações Debugger, Debuggerdeclarações classe, classeconst, let e var, const, let e varfunção, funçãoimportar e exportar, importar e exportarVisão geral de, declaraçõesfunção decodeuri (), funções de URL legadofunção decodeuricomponent (), funções de URL legadoOperador de decrementos (-), operadores aritméticos unáriosdelegação, delegação em vez de herançaExcluir operador, o operador de exclusão, excluindo propriedadesataques de negação de serviço, escrevendo para fluxos e manuseioBackpressureatribuição de destruição, destruição de tarefasAtribuição, destruição da função argumentos em parâmetros-Destructar os argumentos da função em parâmetrosFerramentas de desenvolvimento, explorando JavaScriptEvento de Devicemotion, APIs de dispositivos móveis

Erro ao traduzir esta página.

geração dinamicamente tabelas de conteúdo, exemplo: gerando um índice de elementos iframe, documentar coordenadas e viewport Coordenadas Modificando conteúdo, conteúdo do elemento como HTML Modificação da estrutura, criação, inserção e exclusão de nós Visão geral de documentos de script consulta e configuração de atributos, atributos Selecionando elementos do documento, selecionando elementos de documentos Shadow Dom, Shadow Dom-Shadow Dom API Nós de documentário, usando componentes da web documentos, carregando novos, carregando novos documentos \$ sinal de dólar (\$), identificadores e palavras reservadas Evento DomContentLoaded, execução de programas JavaScript, Cliente-Linha do tempo do JavaScript lateral Operador de pontos (.), Um passeio de JavaScript, Propriedades de consulta e definição Citações duplas (""), literais de cordas Double Shashes (//), um tour de JavaScript, um tour de JavaScript, Comentários função drawImage (), APIs de mídia Operações de desenho curvas, curvas imagens, imagens

retângulos, retângulo texto, textoMatrizes dinâmicas, matrizesEPadrão ECMA402, a API de internacionalizaçãoECMAScript (s), Introdução ao JavaScriptMétodo elementFromPoint (), coordenadas de documentos e viewportCoordenadaselementoselementos da matrizDefinição de termo, matrizesLeitura e escrita, leitura e escrita de elementos de matrizelementos do documentoelementos personalizados, elementos personalizadosdeterminando elemento em um ponto, determinando o elemento em umApontariframe, documentar coordenadas e coordenadas de viewportconsulta geometria de elementos, consultando a geometria deum elementoSelecionando, selecionando elementos do documentoMétodo Ellipse (), curvascaso contrário, se declarações, senão seemojis, unicode, sequências de fuga em literais de string

declarações vazias, declarações compostas e vaziasStrings vazios, textofunção codeuri (), funções do URL do legadofunção codeuricomponent (), funções de URL legadoContrações em inglês, literais de cordasatributo enumerável, introdução a objetos, atributos de propriedadeOperador de igualdade (==)Visão geral de operadores de igualdade e desigualdadeDigite conversões, visão geral e definições, conversões elgualdade, conversões especiais de operadoras de casosOperadores de igualdade, um tour pelo JavaScriptClasses de erro, classes de erromanuseio de errosUsando promessas, lidando com erros com promessas, mais sobre promessase errosAmbiente de host do navegador da web, erros de programaES2016Operador de exponenciação (**), aritmética em JavaScript, aritméticaExpressõesInclui () método, inclui ()ES2017, Palavras -chave assíncronas e aguardam, o operador aguardando,Detalhes assíncronos de JavaScript, Async e Agud-ImplementationES2018

Iterador assíncrono, iteração assíncrona com para/aguardar, Iteração assíncrona Destrução com parâmetros de descanso, função de destruição Argumentos nos parâmetros. Finalmente () Método, a captura e finalmente métodos expressões regulares ASSEGURÇÕES LOLHEBEHIND, especificando a posição de correspondência nomeados grupos de captura, alternância, agrupamento e referências Bandeira, sinalizadores Classes de personagens unicode, classes de personagens Operador espalhado (?), operador espalhado, função de destruição Argumentos em parâmetros, iteradores e geradores ES2019 cláusulas de captura nua, tentar/capturar/finalmente Arrays achatados, achatando matrizes com plano () e plangmap () ES2020? Operador, primeiro definido (??) Bigint Type, números inteiros de precisão arbitrária com bigint BigInt64Array (), tipos de matriz digitados Biguint64Array (), tipos de matriz digitados Operador de acesso condicional (?.), Um passeio de JavaScript, propriedade Erros de acesso, invocação de funções invocação condicional, invocação condicional

Erro ao traduzir esta página.

Erro ao traduzir esta página.

encadear promessas, encadear promessas de promessasManipulação de erros com, mais sobre promessas e erros-a capturae finalmente métodosfazendo promessas, fazendo promessas-promessas em sequênciaVisão geral de, promessasoperações paralelas, promessas em paraleloPromessas em sequência, promessas em promoções de sequência emSequênciaResolvendo promessas, resolvendo promessas-mais nas promessase errosretornando dos retornos de chamada de promessa, o problema e finalmenteMétodosUsando, usando erros de manipulação de promessas com promessasOrdem de enumeração da propriedade, ordem de enumeração de propriedadeliberação de, introdução ao javascriptDefinir e mapas a classes, para/de com set e mapaMétodos abreviados, métodos abreviadosOperador espalhado (?), o operador de propagaçãoStrings delimitadas com backsticks, literais de cordas, literais de modeloSubclasses com cláusula estends, subclasses com estendências eSuper-subclasses com extensões e superTipo de símbolo, visão geral e definiçõesSímbolos como nomes de propriedades, símbolos como nomes de propriedadesMatrizes digitadas, matrizes

Erro ao traduzir esta página.

Registrando manipuladores de eventos, registrando manipuladores de eventosEventos enviados ao servidor, eventos enviados ao servidorRecursos da plataforma da web para investigar, eventosEventEmitter Class, Events and EventEmittertodo () método, todo () e alguns ()exceções, jogando e pegando, joguemétodo EXEC (), EXEC ()notação exponencial, literais de ponto flutuanteOperador de exponenciação (**), aritmética em JavaScript, aritméticaExpressõesExportar declaração, importação e exportaçãoExportar palavras -chave, módulos em ES6declarações de expressão, declarações de expressãoexpressõesExpressões aritméticas, expressões aritméticas-Bitwise OperadoresExpressões de atribuição, assinatura de expressões de atribuiçãocom operaçãoDefinição de termo, expressões e operadoresIncorporação em literais de cordas, literais de cordasExpressões de avaliação, Expressões de avaliação Strict Eval ()formando -se com operadores, um tour por JavaScript, expressões eOperadoresExpressões de definição de função, expressões de definição de função

expressões de função, expressões de função, funções comoNamespacesExpressão inicializadora, um passeio de JavaScript, objeto e matrizInicializadoresExpressões de invocação, expressões de invocação condicionaisInvocação, invocação de invocação InvocaçãoExpressões lógicas, expressões lógicas não (!)Expressão, classes e construtores do New.TargetInicializadores de objetos e matrizes, Inicializadores de objetos e matrizesExpressões de criação de objetos, expressões de criação de objetosExpressões primárias, expressões primáriasExpressões de acesso à propriedade, expressões de acesso à propriedade-Acesso à propriedade condicionalexpressões relacionais, expressões relacionais-a instânciaOperadorversus declarações, um tour de javascript, declaraçõesextensibilidade, extensibilidade do objetoFFunção () Função, declarações de função, invocação de funçõesFunções de fábrica, aulas e protótiposvalores falsamente, valores booleanosfunção fetch (), eventos de redeMétodo Fetch ()

Erro ao traduzir esta página.

Lendo arquivos, leitura de arquivosEscrevendo arquivos, escrevendo arquivosmétodo de preenchimento (),
preenchimento ()Método filtro (), filtro ().Finalmente () método, a captura e finalmente métodos-a captura e
finalmenteMétodosNúmeros de contas financeiras, armazenamentoMétodo Find (), Find () e FindIndex ()Método
FindIndex (), Find () e FindIndex ()Ferramentas de desenvolvedor do Firefox, explorando JavaScriptoperador
primeiro definido (??), primeiro definido (??)Método plano (), achatando matrizes com planos () e plangmap
()Método Flatmap (), achatando matrizes com Flat () e Flatmap ()Literais de ponto flutuante, literais de ponto
flutuante, ponto flutuante binárioe erros de arredondamentoExtensão da linguagem de fluxo, verificação de tipo
com fluxo de fluxoTipos e sindicatos discriminadosTipos de matriz, tipos de matrizTipos de classe, tipos de
classeTipos enumerados e sindicatos discriminados, tipos enumeradose sindicatos discriminadosTipos de funções,
tipos de funçãoInstalando e executando, instalando e executando fluxo

tipos de objetos, tipos de objetoOutros tipos parametrizados, outros tipos parametrizadosVisão geral de, verificação de tipo com fluxoTipos somente leitura, tipos somente leituraTipo Aliases, Aliases de TipoTypeScript versus Flow, Verificação de tipo com fluxoTipos de sindicatos, tipos de sindicatosUsando anotações de tipo, usando anotações de tipopara loops, para, matrizes de iteraçãoopara/aguarda loops, iteração assíncrona com/aguardar, assíncronaalteraçãoopara/em loops, para/in, enumerando propriedadesPara/de loops, texto, para/de OF/in, matrizes de iteração, iteradores eGeradoresMétodo foreach (), matrizes de iteração, foreach ()Método formato (), números de formataçãofrações, números de formataçãoMétodo FromData (), analisando os corpos de respostaJavaScript de front-end, JavaScript em navegadores da webMódulo FS (nó), trabalhando com arquivos que trabalham com diretóriosDeclaração da função, funçãoExpressões de funções, expressões de função, funções como espaço para nomeFunção Palavra -chave, Definindo funções

Function () construtor, o construtor function ()função* palavra -chave, geradoresfunçõesFunções de seta, um tour de JavaScript, funções definidoras, flechasFunçõesSensibilidade ao caso, o texto de um programa JavaScriptfechamentos, fechamentos-closõesDefinindo, definindo funções aninhadas funçõesDefinindo suas próprias propriedades de função, definindo a sua própriaPropriedades da funçãoFunções de fábrica, aulas e protótiposFunções argumentos e parâmetrosTipos de argumento, tipos de argumentoObjeto de argumentos, o objeto de argumentosDestructar os argumentos da função em parâmetros,Destructuring Function argumentos em parâmetros-Destructar os argumentos da função em parâmetrosparâmetros e padrões opcionais, parâmetros opcionais ePadrõesVisão geral dos argumentos e parâmetros de funçãoparâmetros de descanso, parâmetros de descanso e comprimento de variávelListas de argumentosespalhe o operador para chamadas de função, o operador de spread paraChamadas de função

listas de argumentos de comprimento variável, parâmetros de descanso e variável-Listas de argumentos de comprimentoExpressões de definição de função, expressões de definição de funçãoinvocação de funções, criando objetos com novoPropriedades da função, métodos e construtor, funçãoPropriedades, métodos e construtor para o construtor ()Método Bind (), o método Bind ()Call () e Aplicar () Métodos, os métodos Call () e Apply ()Function () construtor, o construtor function ()propriedade de comprimento, a propriedade de comprimentoPropriedade do nome, a propriedade NomePropriedade do protótipo, a propriedade do protótipo Método ToString (), o método ToString ()Programação funcionalExplorando, programação funcionalFunções de ordem superior, funções de ordem superiorMemomando, memóriasAplicação parcial de funções, aplicação parcial deFunçõesesprocessamento de matrizes com função, processando matrizes comFunçõesFunciona como namespaces, funciona como espaços de nomefunciona como valores, funciona como valores que definem o seu próprioPropriedades da função

invocando abordagens para, invocando funções Invocação construtora, invocação construtora Exemplos, um tour pelo JavaScript Invocação de função implícita, invocação de função implícita in invocação indireta, invocação indireta Expressões de invocação, invocação de funções Invocação de método, invocação de método Method Invocation Nomeação, palavras reservadas Visão geral de, visão geral e definições, funções Funções recursivas, invocação de funções Sintaxe abreviada para, um tour pelo JavaScript Funções de matriz estática, funções de matriz estática G Coleta de lixo, visão geral e definições Funções do gerador, geradores, o valor de retorno de um gerador Função (veja também iteradores e geradores) API de geolocalização, APIs de dispositivo móvel Método GetBoundingClientRect (), coordenadas de documentos e Coordenadas de viewport Método getRandomValues ??(), criptografia e APIs relacionadas Métodos Getter, Getters e Setters de Propriedade, Getters, Setters e outros Formulários de método

Global Eval (), Global Eval ()Objeto global, o objeto global, o objeto global nos navegadores da webvariáveis ??globais, variável e escopo constantegradientes, cores, padrões e gradientesGráficos3D, gráficos em um <IVAs>API de tela, gráficos em uma manipulação de <Canvas> -pixeldimensões e coordenadas de tela, dimensões de tela eCoordenadasrecorte, recorteTransformações do sistema de coordenadas, sistema de coordenadasExemplo de transformação de transformaçãoOperações de desenho, operações de desenho de IonaAtributos gráficos, atributos gráficosVisão geral de, gráficos em um <IVAs>Caminhos e polígonos, caminhos e polígonosmanipulação de pixels, manipulação de pixelssalvar e restaurar o estado de gráficos, salvar e restaurarestado gráficoGráficos vetoriais escaláveis ??(SVG), SVG: Graphics de vetor escalável-Criando imagens SVG com JavaScriptmaior que o operador (>)Visão geral dos operadores de comparação

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

métodos de instância, métodos estáticosInstância do operador, a instância do operador, construtores, classeldentidade e instanceofLiterais inteiros, literais inteirosAPI de internacionalizaçãoClasses incluídas na API de internacionalizaçãoComparando cordas, comparando seqüências de cordasdatas e horários de formatação, datas de formatação e temposDatas e tempos de formataçãoNúmeros de formatação, formatação de números de formataçãoApoio ao Node, a API de internacionalizaçãotexto traduzido, a API de internacionalizaçãointerpolação, literais de cordasIntl.DateFormat Classe, Datas de formatação e Formatação de TemposDatas e temposClasse Intl.NumberFormat, Números de formatação de números de formataçãoExpressões de invocaçãoinvocação condicional, invocação condicional, funçãoInvocaçãoInvocação de método, expressões de invocação, invocação de métodoVisão geral de, invocação de funçõesfunção isfinite (), aritmética em javascriptfunção isnan (), aritmética em javascript

iteradores e geradores (veja também métodos de iterador de matriz) Recursos avançados do gerador Valor de retorno das funções do gerador, o valor de retorno de um função do gerador Métodos de retorno () e Throw (), o retorno () e o arremesso () Métodos de um gerador valor das expressões de rendimento, o valor de uma expressão de rendimento Iteradores assíncronos e assíncronos que implementam Iteradores assíncronos fechando iteradores, ?fechando? um iterador: o método de retorno geradores benefícios de uma nota final sobre geradores criando, geradores Definição de termo, geradores exemplos de exemplos de geradores rendimento* e geradores recursivos, rendimento* e recursivo Geradores Como os iteradores funcionam, como os iteradores funcionam implementando objetos iteráveis, implementando objetos iteráveis-Implementando objetos iteráveis Visão geral de iteradores e geradores J JavaScript

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Kpalavras -chavePalavra-chave assíncrona, ASYNC e AWAIT-IMPLEMENTATION DetalhesAguarde detalhes-chave, assíncronos e detalhes de implementaçãoSensibilidade ao caso, o texto de um programa JavaScriptPalavra-chave da classe, aulas com o exemplo de palavra-chave da classe: umClasse de números complexospalavra -chave const, declarações com let e constExportar palavras -chave, módulos em ES6Função Palavra -chave, Definindo funçõesfunção* palavra -chave, geradoresImportar palavra -chave, módulos em ES6Deixe a palavra -chave, um passeio de JavaScript, declarações com LET econst, const, let e varnova palavra -chave, criando objetos com nova invocação construtorapalavras reservadas, palavras reservadas, expressões primáriasEsta palavra -chave, um passeio de JavaScript, expressões primárias,Invocação de funçõesPalavra -chave var, declarações variáveis ??com var, const, let e varrendimento* palavra -chave, rendimento* e geradores recursivosKoch Snowflakes, Exemplo de transformaçãol

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Propriedade LocalStorage, LocalStorage e SessionStorage propriedade de localização, localização, navegação e história Operadores lógicos, um tour de JavaScript, expressões lógicas-lógicas NÃO (!) ASSERÇÕES LOLHEBEHIND, especificando a posição de correspondência de declarações de loop Faça/enquanto loops, faça/while para loops, para, matrizes de iteração Para/aguarda loops, iteração assíncrona com/aguardar, Iteração assíncrona para/em loops, para/in, enumerando propriedades para/de loops, para/para/in, matrizes de iteração, iteradores e Geradores objetivo de, declarações es en quanto loops, en quanto value, operando e tipo de resultado MMagnetômetros, APIs de dispositivo móvel Mandelbrot Set, Exemplo: o Set-Summary e Mandelbrot e Sugestões para leitura adicional Classe de mapa, para/de com set e mapa, a classe de mapa-a classe de mapa Objetos de mapa, visão geral e definições, a classe de mapa Método Map (), map ()

marechaling, serialização de JSON e análiseMétodo Match (), Match ()Método matchall (), matchall ()Método Matches (), selecionando elementos com seletores CSSFunção Math.Pow, expressões aritméticasoperações matemáticas, aritmética em JavaScript-aritmético emJavaScriptSite da MDN, PrefácioAPIs de mídia, APIs de mídiaMemomando, memóriasGerenciamento de memória, visão geral e definiçõesEventos de mensagens, modelo de encadeamento JavaScript do lado do cliente, eventos, eventos,Eventos enviados ao servidor, objetos trabalhadores-objeto global em trabalhadores,Execução do trabalhador Model-PostMessage (), MessagePorts eMessagechannels, mensagens de origem cruzada com PostMessage (),Fork ()-tópicos de trabalhadores, canais de comunicação e mensagens de mensagens,Tipos enumerados e sindicatos discriminadosMessagechannels, PostMessage (), Messageports eMessagechannelsMessagePort Objects, PostMessage (), MessagePorts eMessagechannels, canais de comunicação e mensagens de mensagensmensagensWebSocket APIrecebendo mensagens, recebendo mensagens de um WebSocketEnviando mensagens, enviando mensagens sobre um WebSocket

Tópicos de trabalhadores e mensagens, fios de trabalhadores e mensagens-Mensagens de origem cruzada com pós-maquiagem ()Mensagens de origem cruzada, mensagens de origem cruzada comPostMessage (), mensagens de origem cruzada com Postmessage ()Modelo de execução, modelo de execução do trabalhadorimportar código, importar código para um trabalhadorExemplo de conjunto de Mandelbrot, exemplo: o conjunto de Mandelbrot-Resumo e sugestões para leitura adicionalmódulos, importar código para um trabalhadorVisão geral de fios de trabalhador e mensagensPostMessage (), MessagePorts e Messagechannels,PostMessage (), MessagePorts e MessagechannelsObjetos de trabalhador, objetos de trabalhadorObjeto Workerglobalscope, o objeto global em trabalhadoresmetaprogramação, metaprogramaçãoMétodosAdicionando métodos às classes existentes, adicionando métodos aos existentesClassesMétodos de matrizAplicação genérica de matrizesVisão geral dos métodos de matrizMétodos de classe versus instância, métodos estáticosCriando, um tour de JavaScriptDefinição de termo, funções, invocação de método

encadeamento de método, invocação de método, expressões de invocação, invocação de método-Invocação do métodoMétodos de abreviação, getters, setters e outros formulários de métodoSintaxe abreviada, métodos abreviadosMétodos estáticos, métodos estáticosMétodos de matriz digitados, métodos de matriz digitados e propriedadesSign de menos (-)Operador de subtração, aritmética em JavaScript, aritméticaExpressõesOperador aritmético unário, operadores aritméticos unáriosAPIs de dispositivo móvel, APIs de dispositivo móvelmódulosautomatizando a modularidade baseada em fechamento, automatizando baseado em fechamento baseado emfechamentoModularidadeem ES6importações dinâmicas com importação (), importações dinâmicas comimportar()exportações, es6 exportaçõesImport.Meta.url, import.meta.urlimportações, ES6 importações-ES6 importaçõesimportações e exportações com renomeação, importações e exportações comRenomearMódulos JavaScript na web, módulos JavaScript no

Erro ao traduzir esta página.

Erro ao traduzir esta página.

corpos de resposta de streaming, corpos de resposta de streamingVisão geral de, networkingEventos enviados ao servidor, eventos enviados ao servidorWebsocket API, WebSocketsAPI xmlHttPrequest (xhr), busca ()nova palavra -chave, criando objetos com nova invocação construtoraExpressão, classes e construtores do New.Targetnewline (\ n), literais de string, sequências de fuga em literais de stringNewlines, semicolons opcionais semicolons opcionaisUsando para formatação de código, o texto de um programa JavaScriptNóiteração assíncrona, o loop for/aguardar, o nó éO nó assíncrono por padrão é assíncrono por padrãoBenefícios da Introdução ao JavaScript, JavaScript do lado do servidorcom nóBigint Type, números inteiros de precisão arbitrária com bigintbuffers, buffersretornos de chamada e eventos, retornos de chamada e eventos no nóProcessos infantis, trabalhando com processos infantis-Fork ()Definindo recurso de JavaScript do lado do servidor com nóEventos e EventEmitter, Eventos e EventEmitterManuseio de arquivos, trabalhando com arquivos que trabalham com diretóriosdiretórios, trabalhando com diretórios

metadados do arquivo, metadados do arquivoStrings de modo de arquivo, escrevendo arquivosOperações de arquivo, operações de arquivoVisão geral de, trabalhando com arquivoscaminhos, descritores de arquivos e trabalhos de arquivo, caminhos, arquivoDescritores e trabalhos de arquivoLendo arquivos, leitura de arquivosEscrevendo arquivos, escrevendo arquivosClients e servidores HTTP, clientes HTTP e servidores-
httpClients e servidoresInstalando, explorando JavaScript, JavaScript do lado do servidor comNóAPI INTL, a API de internacionalizaçãoMódulos em, módulos em módulos de nó no nó na webServidores e clientes de rede não http, rede não httpServidores e clientesParalelismo com, Node é assíncrono por padrãoDetalhes do processo, processo, CPU e detalhes do sistema operacionalProgramação básica, básico de programação do nó-o
nóGerente de pacotesArgumentos da linha de comando, argumentos da linha de comando eVariáveis ??de ambienteSaída do console, saída do consolevariáveis ??de ambiente, argumentos da linha de comando e

Variáveis ??de ambiente módulos, módulos de nó Gerenciador de pacotes, o gerenciador de pacotes de nó Ciclo de vida do programa, ciclo de vida do programa Documentação de referência, prefácio fluxos, modo de fluxo siteração assíncrona, iteração assíncrona Visão geral de, fluxos tubos, tubos Lendo com eventos, lendo fluxos com eventos tipos de fluxos escrevendo e lidando com a contrapressão, escrevendo para fluxos e Manuseio de contrapressão Tópicos dos trabalhadores, tópicos de trabalhadores compartilhando matrizes digitadas entre Tópicos canais de comunicação e por portes de mensagem, comunicação Canais e mensagens de mensagens criando trabalhadores e passando mensagens, criando trabalhadores e mensagens passando Visão geral de tópicos de trabalhador Compartilhando matrizes digitadas entre threads, compartilhando matrizes digitadas Entre threads transferindo portões de mensagem e matrizes digitadas, transferindo Messageports e matrizes digitadas

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

extensibilidade do objeto, extensibilidade do objetoMétodos de objeto, métodos de objeto-método tojson ()Visão geral de, um tour de javascript, visão geral e definiçõesVisão geral e definiçõesconsulta e configuração de propriedades, consultas e configurações de propriedade-Erros de acesso à propriedadeobjetos serializando, objetos serializandoPropriedades de teste, propriedades de testeEvento OnMessage, recebendo mensagens de um WebSocket, trabalhadorObjetos-objeto global em trabalhadores, PostMessage (), MessagePorts,e Messagechannels, mensagens de origem cruzada com Postmessage ()operadoresoperadores aritméticos, um tour de JavaScript, aritmético emJavaScript-aritmético em JavaScript, Expressões aritméticas-Operadores bitwiseoperadores de tarefas, designação de expressões de atribuição comOperaçãooperadores binários, número de operandoOperadores de comparação, operadores de comparaçãoOperadores de igualdade e desigualdade, igualdade e desigualdadeOperadoresOperadores de igualdade, um tour pelo JavaScriptformando expressões com, um tour de JavaScript, expressões eOperadoresOperadores lógicos, um passeio de JavaScript, expressões lógicas-

Lógico não (!)operadores diversosaguardar operador, o operador aguardaroperador de vírgula (,), o operador de vírgula (,)operador condicional (? :), o operador condicional (? :)Excluir operador, o operador de exclusãooperador primeiro definido (??), primeiro definido (??)TIPEOF OPERADOR, O TIPEOF OPERADORoperador vazio, o operador vazioNúmero de operandos, número de operandosoperando e tipo de resultado, operando e tipo de resultadoAssociatividade do Operador, Associatividade do Operadorprecedência do operador, precedência do operadorEfeitos colaterais do operador, efeitos colaterais do operadorOrdem de avaliação, ordem de avaliaçãoVisão geral da visão geral do operadorOperadores de pós -fix, semicolons opcionaisOperadores relacionais, um tour de JavaScript, expressões relacionais-A instância do operadorTabela de, Visão geral do operadoroperadores ternários, número de operandoSemicolons opcionais, semicolons opcionais semicolons opcionais

Erro ao traduzir esta página.

Erro ao traduzir esta página.

pixels, coordenadas de documentos e coordenadas de viewport, pixelManipulação mais sinal (+) Operador de adição e atribuição (+=), atribuição com Operação Operador de adição, aritmética em JavaScript, o operador + concatenação de string, literais de string, trabalhando com strings, o + Operador Digite conversões, conversões especiais de operadoras de casos Operador aritmético unário, operadores aritméticos unários polígonos, caminhos e polígonos e polígonos Método pop (), pilhas e filas com push (), pop (), shift () e NIFT () PopState Event, Categorias de Eventos, Gerenciamento de História compushState () - Networking zero positivo, aritmética em javascript Posse, literais de cordas Operadores de pós - fix, semicolons opcionais Método PostMessage (), PostMessage (), Messageports e Messagechannels Formatação mais bonita, JavaScript com mais bonito Expressões primárias, expressões primárias Tipos primitivos Valores da verdade booleana, valores booleanos-booleanos

valores primitivos imutáveis, valores primitivos imutáveis ??eReferências de objetos mutáveisTipo de número, números e horáriosVisão geral e definições, visão geral e definiçõesTipo de string, literais de modelo marcado com textoPrintProps () função, declarações de funçãoCampos privados, campos públicos, privados e estáticosprocedimentos, funçõesesprogramasManuseio de erros, erros de programaExecução de JavaScript, execução de programas JavaScript- Client-Linha do tempo do JavaScript lateralModelo de encadeamento do lado do cliente, encadeamento JavaScript do lado do clientemodeloLinha do tempo do lado do cliente, linha do tempo JavaScript do lado do clienteEntrada e saída, entrada e saída de programasAplicativos da Web progressivos (PWAs), aplicativos da Web progressivos e serviçoTrabalhadoresPromessa cadeias, promessas, encadeamento de promessas promessasFunção promey.all (), promessas em paraleloPromessasencadear promessas, encadear promessas de promessaslidar com erros com, lidar com erros com promessas, mais sobrePromessas e erros-a captura e finalmente métodos

Erro ao traduzir esta página.

herdando, herançanomeação, símbolos, introdução a objetos, símbolos como propriedadeNomespropriedades não herdadas, introdução aos objetoserros de acesso à propriedade, erros de acesso à propriedadeExpressões de acesso à propriedade, expressões de acesso à propriedadeAtributos da propriedade, introdução a objetos, atributos de propriedade-Atributos da propriedadeDescritores de propriedades, atributos de propriedadePropriedade Getters e Setters, Property Getters e SettersConsulta e cenário, consulta e configuração de propriedades-PropertyErros de acessoTestes, propriedades de testePropriedades da matriz digitada, métodos de matriz digitados e propriedadesMétodo PropertyInumerable (), Propriedades de testeherança prototípica, introdução a objetos, herançaCadeias de protótipo, protótiposprotótipos, protótipos, herança, propriedade do protótipo, classes e protótipos, o atributo do protótipoinvariantes de procuração, invariantes de procuraçãoObjetos de proxy, objetos de proxy-proxy invariantesnúmeros de pseudorandom, criptografia e APIs relacionadasCampos públicos, campos públicos, privados e estáticos

Empurre API, aplicativos da Web progressivos e trabalhadores de serviço com método push (), um tour de javascript, pilhas e filas com push (), pop (), shift () e não dividido () QM Método quadraticcurve () , curvas Método querySelector () , selecionando elementos com seletores CSS Método querySelectorAll () , selecionando elementos com seletores CSS Marcas de citação Citações duplas (""), literais de corda citações únicas (''), literais de cordas R React, JSX: Expressões de marcação em JavaScript retângulos, retângulos Funções recursivas, invocação de funções geradores recursivos, rendimento* e geradores recursivos Reduce () Método, Reduce () e ReduceRight () Método ReduceRight (), Reduce () e ReduceRight () Tipos de referência, valores primitivos imutáveis ?? e objeto mutável Referências Reflita API, a API refletida-refletir API Reflete.ownskeys () função, enumerando propriedades Classe regexp

método EXEC (), EXEC ()LastIndex Property and Regexp Reutily, EXEC ()Visão geral da classe RegexpPropriedades regexp, propriedades regexpmétodo test (), teste ()Tipo de regexp, visão geral e definições, correspondência de padrões, padrãoCombinando com expressões regulares (ver também correspondência de padrões)Expressões regulares, correspondência de padrões com expressões regulares(Veja também correspondência de padrões)Expressões relacionais, expressões relacionais-a instância do operadorOperadores relacionais, um tour pelo JavaScriptMétodo Substituir (), trabalhando com Stringsrequer () função, importações de nopalavras reservadas, palavras reservadas, expressões primáriasParâmetros de descanso, parâmetros de descanso e listas de argumentos de comprimento variávelRetornar declarações, retornarRetornar valores, funçõesMétodo de retorno (), ?fechando? um iterador: o método de retorno, oMétodos de retorno () e Throw () de um geradorMétodo reverso (), um tour de javascript, reverse ()Erros de arredondamento, ponto flutuante binário e erros de arredondamentoS

Erro ao traduzir esta página.

Política da mesma origem, a política da mesma origemRecursos da plataforma da web para investigar, segurançaSemicolon (;), semicolons opcionais semicolons opcionaisInformações sensíveis, armazenamentoAPI do sensor, APIs de dispositivo móvelserialização, serializando objetos, serialização e análise JSON,Gerenciamento de história com pushState ()Eventos enviados ao servidor, eventos de eventos de servidor-servidor-servidoresJavaScript do lado do servidor, JavaScript em navegadores da Web, servidorJavaScript com nóTrabalhadores de serviço, aplicativos da Web progressivos e trabalhadores de serviçoPropriedade da sessão, LocalStorage e SessionStorageSet Class, para/de com set e mapa, a classe Set-a classe SetDefina objetos, visão geral e definiçõesSET () construtor, a classe Setfunção setInterval (), temporizadoresconjuntos e mapasDefinição de conjuntos, a classe setClasse de mapa, a classe de mapa-a classe de mapaVisão geral de, sets e mapasClasse definida, a classe Set-The Set ClassClasses fracas e fracas, fraco e fracoMétodos Setter, Getters e Setters de Propriedade, Getters, Setters e outros

Formulários de método função `setTimeout()`, temporizadores, temporizadores
Método `setTransform()`, transforma o sistema de coordenadas
Shadow Dom, Shadow Dom-Shadow Dom API Sombras, sombras
Operador esquerdo do turno (`<<`), operadores bitwise Mudar à direita com o operador de sinal (`>>`), operadores bitwise Mudar à direita com o operador de preenchimento zero (`>>>`), operadores bitwise
Método Shift (), pilhas e filas com push (), pop (), shift () e NIFT ()
Métodos abreviados, métodos abreviados, getters, setters e outros
Formulários de método Efeitos colaterais, efeitos colaterais do operador
citações únicas ('), literais de cordas
Método Slice (), Slice ()
Alguns () método, todo () e outros ()
Classificar ordem, comparando strings
Método Sort (), Invocação Condicional, Sort ()
Matrizes esparsas, matrizes, matrizes esparsas
Método Splice (), Splice ()
Método Split (), Split ()
Operador espalhado (?), operador espalhado, o operador de espalhamento, o
Espalhe o operador para chamadas de função,
iteradores e geradores

Suportes quadrados ([]), um tour de JavaScript, trabalhando com strings, Objetos e matrizes inicializadores, consultas e configurações de propriedades, leitura e escrevendo elementos de matriz, cordas como matrizes Biblioteca padrão (consulte Biblioteca Padrão JavaScript) Blocos de declaração, declarações compostas e vazias declarações (ver também declarações) declarações compostas e vazias, compostas e vazias Declarações de declarações condicionais, declarações, comutação condicionais Estruturas de controle, um passeio de JavaScript - um passeio de JavaScript Declarações de expressão, declarações de expressão versus expressões, um passeio de JavaScript Declaração se/else, valores booleanos Jump declarações, declarações, saltos-trinta/captura/finalmente quebras de linha e semicolons opcionais semicolons opcionais Lista de declarações de JavaScript Loops, declarações, loops for/in declarações diversas Declarações Debugger, Debugger Use diretiva estrita, "Use rigoroso" com declarações, declarações diversas Visão geral de, declarações

Separando com semicolons, semicolons opcionais-opcionaisSemicolonsJogue declarações, jogueExperimente/Finalmente as instruções, tente/capture/finalmente-try/Catch/Finalmente declarações de rendimento, rendimento, o valor de uma expressão de rendimentoCampos estáticos, campos públicos, privados e estáticosMétodos estáticos, métodos estáticosArmazenamento, IndexedDB de armazenamentoCookies, biscoitosIndexedDB, indexedDBLocalStorage e SessionStorage, LocalStorage e SessionStorageVisão geral de, armazenamentoSegurança e privacidade, armazenamentofluxos (nó), modo de fluxos de fluxositeração assíncrona, iteração assíncronaVisão geral de, fluxostubos, tubosLendo com eventos, lendo fluxos com eventostipos de fluxosescrevendo e lidando com a contrapressão, escrevendo para fluxos eManuseio de contrapressãooperador estrito de igualdade (==)valores booleanos, valores booleanos

Erro ao traduzir esta página.

Matriz para conversões de string, matriz para conversões de stringcaracteres e pontos de código, textoMétodos para correspondência de padrõesMatch (), Match ()matchall (), matchall ()substituir (), substituir ()pesquisa (), métodos de string para correspondência de padrõesSplit (), Split ()Visão geral de, textoLiterais de cordas, literais de cordascordas como matrizes, cordas como matrizestrabalhando comAcessando caracteres individuais, trabalhando com stringsAPI para, trabalhando com stringsComparando, trabalhando com cordas, comparando stringsconcatenação, trabalhando com stringsDeterminando o comprimento, trabalhando com stringsimutabilidade, trabalhando com cordasAlgoritmo de clone estruturado, gerenciamento de história com pushState ()subarrays, subarrays with slice (), splice (), preench () e copywithin ()subclasseshierarquias de classe e classes abstratas, hierarquias de classe e

Classes abstratas-Summary delegação versus herança, delegação em vez de herança Visão geral de, subclasses protótipos e subclasses e protótipos com cláusula de estendências, subclasses com extensões e super-Subclasses com extensões e supersub -rotinas, funções Operador de subtração (-), aritmético em JavaScript pares substitutos, textoSVG (ver Graphics Scalable Vector (SVG)) Switch Declarações, switch-switchSymbol.asynciterator, símbolo.iterator e símbolo.asynciteratorSymbol.hasinstance, símbolo.hasinstanceSymbol.iscoNcatsPreadable, symbol.iscoNcatsPreadableSymbol.iterator, símbolos conhecidosSymbol.spécies, símbolo.species-symbol.speciesSymbol.Toprimitive, símbolo.ToprimitiveSymbol.ToStringTag, symbol.ToStringTagSymbol.UNSCOPABLES, SYMBOL.UNSCOPABLESSímbolos Definição de extensões de linguagem, visão geral e definições nomes de propriedades, símbolos, símbolos como nomes de propriedades

símbolos conhecidos, símbolos conhecidosExecução de scripts síncronos, quando os scripts são executados: assíncronos e adiadossintaxeEstruturas de controle, um passeio de JavaScript-um passeio de JavaScriptDeclarando variáveis, um tour pelo JavaScriptComentários em inglês, um passeio de JavaScript, um tour de JavaScriptOperadores de igualdade e relacional, um tour pelo JavaScriptexpressõesformando com operadores, um tour pelo JavaScriptExpressão inicializadora, um tour de JavaScriptestendido para literais de objetos, sintaxe literal de objeto estendido-Propriedade Getters and Settersfunções, um tour de javascriptEstrutura lexical, estrutura lexical-sumárioSensibilidade ao caso, o texto de um programa JavaScriptComentários, comentáriosIdentificadores, o texto de um Programa JavaScript-Identificadores ePalavras reservadasquebras de linha, o texto de um programa JavaScriptLiterais, literaispalavras reservadas, palavras reservadas, expressões primáriasSemicolons, semicolons opcionais semicolons opcionais

Erro ao traduzir esta página.

Literais de cordas, literais de cordas
Tipo de string representando, texto
Literais de modelos, literais de modelo
Trabalhando com cordas, trabalhando com cordas
Editores de textoNormalização, normalização
unicodeUsando com o nó, olá mundial
Estilos de texto, estilos de texto.then () método, usando promessas,
encadeamento de promessas, mais sobre promessase erros
Esta palavra -chave, um tour de javascript,
expressões primárias, função
Invocação Threading, fios de trabalhadores e mensagens, aplicativos da Web
progressivos e Trabalhadores de serviço (ver também API do trabalhador)
Gráficos 3D, gráficos em uma
<VAS>lançar declarações, arremesso, classes de erro
Método Throw (), os métodos de retorno () e Throw () de um
gerador
Fusos horários, datas de formatação e horários
Timers, temporizadores, temporizadores
Timestamps, datas e horários, registro de data e hora
Método TodATestring (), formatação e análise de data
Método ToExponencial (), conversões explícitas
Método tofixed (), conversões explícitas

Método `ToISOString()`, Formatação e Parsing Data Strings, JSONPersonalizaçõesMétodo `ToJSON()`, o método `ToJSON()`, JSON CustomizationsTOLOCALEDATESTRING () Método, formatação e análise de datas de análise,Datas e tempos de formataçãoMétodo `toLocaleString()`, o método `toLocaleString()`, matriz para stringConversões, formatação e análise de datas de análiseTOLOCALETIMESTRING () Método, formatação e análise de datas de análise,Datas e tempos de formataçãoFerramentas e extensões, ferramentas de JavaScript e extensões-imensasTipos e sindicatos discriminadosBundling de código, agrupamento de códigoFormatação de javascript com formatação mais bonita e javascript comMais bonitoExtensão da linguagem JSX, JSX: Expressões de marcação em JavaScript-JSX: Expressões de marcação em JavaScriptlinhando com Eslint, linhando com EslintVisão geral de ferramentas e extensões JavaScriptGerenciamento de pacotes com NPM, gerenciamento de pacotes com NPMTranspilação com Babel, transpilação com BabelTipo de verificação com fluxo, verificação de tipo com o fluxo, enumeradoTipos e sindicatos discriminadosTipos de matriz, tipos de matrizTipos de classe, tipos de classeTipos enumerados e sindicatos discriminados, enumerados

Tipos e sindicatos discriminadosTipos de funções, tipos de funçãoInstalando e executando, instalando e executando fluxoTipos de objetos, tipos de objetoOutros tipos parametrizados, outros tipos parametrizadosVisão geral de, verificação de tipo com fluxoTipos somente leitura, tipos somente leituraTipo Aliases, Aliases de TipoTypeScript versus Flow, Verificação de tipo com fluxoTipos de sindicatos, tipos de sindicatosUsando anotações de tipo, usando anotações de tipoTeste de unidade com JEST, teste de unidade com JESTMétodo toprecision (), conversões explícitasmétodo tostring (), valores booleanos, conversões explícitas, ométodos tostring () e valueof (), o operador +, igualdade com tipoconversão, o método tostring (), o método tostring (), formataçãoe strings de data de análiseTOTIMEString () Método, formatação e análise de datas de análiseMétodo touppercase (), trabalhando com stringsMétodo toutcString (), formatação e análise de datas de análiseTransformações, o sistema de coordenadas transforma a transformaçãoexemploMétodo traduz (), transforma o sistema de coordenadas

translucidez, translucidez e composição transpilação, transpilação com Babel valores verdadeiros, valores booleanos Experimente/Finalmente as instruções, tente/capture/finalmente-try/Catch/Finalmente Verificação de tipo, verificação de tipo com tipos anumerados por fluxo e Sindicatos discriminados Tipos de matriz, tipos de matriz Tipos de classe, tipos de classe tipos enumerados e sindicatos discriminados, tipos enumerados se sindicatos discriminados Tipos de funções, tipos de função Instalando e executando o fluxo, instalando e executando o fluxo tipos de objetos, tipos de objeto Outros tipos parametrizados, outros tipos parametrizados Visão geral de, verificação de tipo com fluxo Tipos somente leitura, tipos somente leitura Tipo Aliases, Aliases de Tipo TypeScript versus Flow, Verificação de tipo com fluxo Tipos de sindicatos, tipos de sindicatos Usando anotações de tipo, usando anotações de tipo Digite conversões igualdade e, conversões e igualdade, igualdade com tipo conversão

conversões explícitas, conversões explícitasDados financeiros e científicos, conversões explícitasconversões implícitas, conversões explícitasObjeta -se a conversões primitivasalgoritmos para, objeta-se a conversões primitivas, objeto aAlgoritmos de conversão primitivaConversões objeto a boolean e objeto a booleanoConversões de objeto para número, objeto a númeroConversões de objetos para cordas, objeto para cordãoConversões especiais de operador de caso, operador de caso especialconversõesmétodos `ToString()` e `ValueOf()`, o `ToString()` eMétodos `ValueOf()`Visão geral das conversões de tipomatrices digitadasCriando, criando matrizes digitadasDataView e Endianness, DataView e EndiannessMétodos e propriedades, métodos de matriz digitados e propriedadesVisão geral de matrizes digitadas e dados bináriosversus matrizes regulares, matrizesCompartilhando entre threads, compartilhando matrizes digitadas entre threadsTipos de matriz digitados, tipos de matriz digitadosUsando, usando matrizes digitadas

Erro ao traduzir esta página.

subfluxo, aritmética em javascriptsublinhado (_), identificadores e palavras reservadassublinhou, como separadores numéricos (_), literais de ponto flutuanteUNESCAPE () FUNÇÃO, FUNÇÕES DE URL LEGADOEvento de rejeição não entrega, erros de programaConjunto de caracteres unicodeSequências de fuga, sequências de escape unicode, sequências de escapeem literais de cordasStrings JavaScript, textoNormalização, normalização unicodeVisão geral de, unicodeCombinação de padrões, classes de personagenscaracteres espaciais, o texto de um programa JavaScriptTeste de unidade, teste de unidade com JESTMétodo de Netift (), pilhas e filas com push (), pop (), shift () eNIFT ()URL APIs, URL APIS-LEGACY URL FunctionsUse diretiva estritaAplicação padrão do modo rigoroso, classes com a palavra -chave de classe,Módulos em ES6, JavaScript Módulos na WebExcluir operador e, o operador de exclusãoEVALL () Função, EVALdeclarações de função, declarações de função

Erro ao traduzir esta página.

Definições tipos de, um tour de javascriptPalavra-chave var, declarações variáveis ??com var, const, let e varvarargs, parâmetros de descanso e listas de argumentos de comprimento variávelfunções variáveis ??de arity, parâmetros de descanso e comprimento de variávelListas de argumentosvariáveisSensibilidade ao caso, o texto de um programa JavaScriptdeclaração e atribuiçãodeclarações com let and const, declarações com let econst-declarações e tiposdeclarações com VAR, declarações variáveis ??com VARatribuição de destruição, destruiçãoAtribuição de destruiçãoVisão geral de, um tour de JavaScriptvariáveis ??não declaradas, declarações variáveis ??com VARDefinição de termo, declaração e atribuição variáveisdeclarações variáveis ??e içadas com VARNomeação, palavras reservadasVisão geral de, tipos, valores e variáveis-overview eDefiniçõesescopo de escopo variável e constante, funções aninhadasFunções variádicas, parâmetros de descanso e argumento de comprimento variávelListas

fluxos de vídeo, APIs de mídiaViewport, coordenadas de documentos e coordenadas de viewport, viewportTamanho, tamanho do conteúdo e posição de rolagemoperador vazio, o operador vazioClasse de mapa fraco, mapa fraco e conjunto fracoClasse fraca classe, mapa fraco e conjunto fracoAPI de autenticação na web, criptografia e APIs relacionadasambiente de host de navegador da webAPIs assíncronas, eventosAPIs de áudio, APIs de áudio-a API WebaudioBenefícios do JavaScript, JavaScript em navegadores da WebAPI de tela, gráficos em uma manipulação de <Canvas> -pixeldimensões e coordenadas de tela, dimensões de tela eCoordenadasrecorte, recorteTransformações do sistema de coordenadas, sistema de coordenadasExemplo de transformação de transformaçãoOperações de desenho, operações de desenho de lonaatributos gráficos, atributos gráficos, salvando e restaurandoestado gráficoVisão geral de, gráficos em um <IVAs>Caminhos e polígonos, caminhos e polígonos

manipulação de pixels, manipulação de pixelsDocumentar geometria e rolagem, geometria de documentos eTamanho da Scrolling-ViewPort, tamanho de conteúdo e posição de rolagemPixels CSS, coordenadas de documentos e viewportCoordenadasdeterminando elemento em um ponto, determinando o elemento em umApontarDocumentar coordenadas e coordenadas de viewport, documentoCoordena e coordenadas de viewportconsulta geometria de elementos, consultando a geometria de um elementoRolando, rolandoTamanho da viewport, tamanho do conteúdo e posição de rolagem, viewportTamanho, tamanho do conteúdo e posição de rolagemEventos, eventos para eventos personalizadosDespacha eventos personalizados, despachando eventos personalizadosCancelamento de eventos, cancelamento de eventoscategorias de eventos, categorias de eventosInvocação de manipulador de eventos, invocação de manipulador de eventospropagação de eventos, propagação de eventosVisão geral dos eventosRegistrando manipuladores de eventos, registrando manipuladores de eventosAPIs legadas, JavaScript em navegadores da weblocalização, navegação e história, localização, navegação e

Gerenciamento de história da história com pushState ()História de navegação, história de navegaçãoCarregando novos documentos, carregando novos documentosVisão geral de localização, navegação e históriaExemplo de conjunto de Mandelbrot, exemplo: o Set-Summary Mandelbrote sugestões para leitura adicionalnavegadores com reconhecimento de módulos, módulos JavaScript na webNetworking, Networking-Protocol NegociaçãoMétodo Fetch (), Fetch ()Visão geral de, networkingEventos enviados ao servidor, eventos enviados ao servidorWebsocket API, WebSocketsVisão geral de, JavaScript em navegadores da webGráficos vetoriais escaláveis ??(SVG), SVG: Graphics de vetor escalável-Criando imagens SVG com JavaScriptCriando imagens SVG com JavaScript, criando imagens SVGcom javascriptVisão geral de, SVG: gráficos vetoriais escaláveisScripts SVG, Scripts SVGSVG em HTML, SVG em HTMLScripts CSS, Scripts CSS-CSS Animações e eventosEstilos CSS comuns, Scripts CSSEstilos computados, estilos computados

Animações e eventos CSS, animações e eventos CSSClasses CSS, aulas CSSEstilos embutidos, estilos embutidosConvenções de nomeação, estilos em linhafolhas de estilo de script, folhas de estilo de scriptDocumentos de script, documentos de script-Exemplo: gerando umíndiceestrutura de documentos e travessia, estrutura de documentos eTraversalgeração dinamicamente tabelas de conteúdo, exemplo:Gerando uma tabela de índiceModificando conteúdo, conteúdo do elemento como HTMLModificação da estrutura, criação, inserção e exclusão de nósVisão geral de documentos de scriptconsulta e configuração de atributos, atributosSelecionando elementos do documento, selecionando elementos de documentosArmazenamento, IndexedDB de armazenamentoCookies, biscoitosIndexedDB, indexedDBLocalStorage e SessionStorage, LocalStorage eSessionStorageVisão geral de, armazenamentoSegurança e privacidade, armazenamento

Componentes da Web, componentes da Web-Exemplo: A <search-box>Componente da Webelementos personalizados, elementos personalizadosNós de documentário, usando componentes da webModelos HTML, modelos HTMLVisão geral de, componentes da webExemplo de caixa de pesquisa, exemplo: uma web <search-box>ComponenteShadow Dom, Shadow DomUsando, usando componentes da webRecursos da plataforma da web para investigarAPIs binárias, APIs bináriasAPIs de criptografia e segurança, criptografia e relacionadasAPIseventos, eventosHTML e CSS, HTML e CSSAPIs de mídia, APIs de mídiaAPIs de dispositivo móvel, APIs de dispositivo móvelAPIs de desempenho, desempenhoAplicativos da Web progressivos e trabalhadores de serviço, progressivosAplicativos da Web e trabalhadores de serviçoSegurança, segurançaWebAssembly, WebAssembly

Erro ao traduzir esta página.

WebSocket APIcriando, conectando e desconectando websockets, criação,Conectando e desconectando WebSocketsVisão geral de, websocketsNegociação de protocolo, negociação de protocolorecebendo mensagens, recebendo mensagens de um WebSocketEnviando mensagens, enviando mensagens sobre um WebSocketenquanto loops, enquanto com declarações, declarações diversasAPI do trabalhadorMensagens de origem cruzada, mensagens de origem cruzada comPostMessage ()erros, erros em trabalhadoresModelo de execução, modelo de execução do trabalhadorimportar código, importar código para um trabalhadorExemplo de conjunto de Mandelbrot, exemplo: o Set-Summary Mandelbrote sugestões para leitura adicionalmódulos, importar código para um trabalhadorVisão geral de fios de trabalhador e mensagensPostMessage (), MessagePorts e Messagechannels,PostMessage (), MessagePorts e MessagechannelsObjetos de trabalhador, objetos de trabalhadorObjeto Workerglobalscope, o objeto global em trabalhadores

atributo gravável, introdução a objetos, atributos de propriedadeXAPI XMLHttpRequest (xhr), busca ()XSS (scripts cross-sites), a política da mesma origemYdeclarações de rendimento, rendimento, o valor de uma expressão de rendimentorendimento* palavra -chave, rendimento* e geradores recursivosZzerozero negativo, aritmética em javascriptzero positivo, aritmética em javascriptMatrizes baseadas em zero, matrizes

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Parque Nacional, em Java, Indonésia. Esta estratégia parece estar ajudando a sobrevivência desses rinocerontes por enquanto, como um censo de 1967 contou apenas 25. Muitos dos animais em capas de O'Reilly estão em perigo; todos eles são importantes para o mundo. A ilustração colorida na capa é de Karen Montgomery, com base em uma gravura em preto e branco de animais de Dover. As fontes de capa são Gilroy e Guardian Sans. A fonte de texto é Adobe Minion Pro; a fonte do cabeçalho é um míríade de Adobe condensado; e a fonte do código é Dalton Ubuntu Mono de Maag.