Erro ao traduzir esta página.

ii.4.7.2 Operando e tipo de resultado
iii.4.7.3 Efeitos colaterais do operador
4.4.7.4 Precedência do operador
v. 4.7.5 Associatividade do operador
vi.4.7.6 Ordem de avaliação
h.4.8 Expressões aritméticas
eu.4.8.1 O operador +
ii.4.8.2 Operadores aritméticos unários
iii.4.8.3 Operadores bitwise
eu.4.9 Expressões relacionais
eu.4.9.1 Operadores de igualdade e desigualdade
ii.4.9.2 Operadores de comparação
iii.4.9.3 O operador no
4.4.9.4 A instância do operador
j.4.10 Expressões lógicas
eu.4.10.1 lógico e (&&)
ii.4.10.2 Lógico ou (||)
iii.4.10.3 Lógico não (!)
k.4.11 Expressões de atribuição
eu.4.11.1 Atribuição com operação
l.4.12 Expressões de avaliação
eu.4.12.1 Eval ()

v. 8.2.5 Invocação de função implícita
c.8.3 Argumentos e parâmetros de função
eu.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
4.8.3.4 O operador de propagação para função
Chamadas
v. 8.3.5 Argumentos de função de destruição
em parâmetros
vi.8.3.6 Tipos de argumento
d.8.4 Funções como valores
eu.8.4.1 Definindo sua própria função
Propriedades
e.8.5 Funções como namespaces
f.8.6 fechamentos
g.8.7 Propriedades, métodos e métodos da função
Construtor
eu.8.7.1 A propriedade de comprimento
ii.8.7.2 A propriedade Nome
iii.8.7.3 A propriedade do protótipo
4.8.7.4 Os métodos Call () e Apply ()
v. 8.7.5 O método bind ()
vi.8.7.6 O método ToString ()

um.11.1 conjuntos e mapas
eu.11.1.1 A classe definida
ii.11.1.2 A classe do mapa
iii.11.1.3 Frawmap e fraco
b.11.2 Matrizes digitadas e dados binários
eu.11.2.1 Tipos de matriz digitados
ii.11.2.2 Criando matrizes digitadas
iii.11.2.3 Usando matrizes digitadas
4.11.2.4 Métodos de matriz digitados e
Propriedades
v. 11.2.5 DataView e Endianness
c.11.3 Combinação de padrões com expressões regulares
eu.11.3.1 Definindo expressões regulares
ii.11.3.2 Métodos de string para padrão
Correspondência
iii.11.3.3 A classe Regexp
d.11,4 datas e horários
eu.11.4.1 Timestamps
ii.11.4.2 Data aritmética
iii.11.4.3 Data de formatação e análise
Cordas
e.11.5 Classes de erro
f.11.6 Serialização e análise de JSON JSON

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 EventEmitter
e.16.5 fluxos
eu.16.5.1 Tubos
ii.16.5.2 iteração assíncrona
iii.16.5.3 Escrevendo para fluxos e manuseio Backpressure
4.16.5.4 Lendo fluxos com eventos
f.16.6 Process, CPU e detalhes do sistema operacional
g.16.7 trabalhando com arquivos
eu.16.7.1 caminhos, descritores de arquivos e FileHandles
ii.16.7.2 Leitura de arquivos
iii.16.7.3 Escrevendo arquivos
4.16.7.4 Operações de arquivo
v. 16.7.5 metadados do arquivo
vi.16.7.6 Trabalhando com diretórios
h.16.8 clientes e servidores HTTP
eu.16.9 Servidores e clientes de rede não-HTTP
j.16.10 Trabalhando com processos filhos

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 e recursos emergentes) serão bem servidos por um profundo e reflexivo Viaje por este livro abrangente e definitivo. ?

?Brian Sletten, presidente da Bosatsu Consulting

JavaScript: o definitivo
Guia
Sétima edição
Domine a programação mais usada do mundo
Linguagem
David Flanagan

JavaScript: The Definitive Guide, Sétima Edição
Por David Flanagan

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 vendas
uso promocional.Edições online também estão disponíveis para a maioria dos títulos
(http://oreilly.com).Para mais informações, entre em contato com o nosso
Departamento de Vendas Corporativas/Institucionais: 800-998-9938 ou
corporate@oreilly.com.
Editor de aquisições: Jennifer Pollock
Editor de desenvolvimento: Angela Rufino
Editor de produção: Deborah Baker
CopyEditor: Holly Bauer Forsyth
Revisor: Piper Editorial, LLC
Indexador: Judith McConville
Designer de interiores: David Futato
Designer de capa: Karen Montgomery

Ilustrador: Rebecca DeMarest
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.

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

Dedicação

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

Prefácio

Este livro cobre a linguagem JavaScript e as APIs JavaScript
implementado por navegadores da web e por nó.Eu escrevi para leitores com
alguma experiência de programação anterior que deseja aprender JavaScript e
também para programadores que já usam JavaScript, mas querem levar seus
compreensão de um novo nível e realmente dominar o idioma.Meu objetivo
com este livro é documentar a linguagem JavaScript de forma abrangente
e definitivamente e para fornecer uma introdução aprofundada ao máximo
APIs importantes do lado do cliente e do lado do servidor disponíveis para JavaScript
programas.Como resultado, este é um livro longo e detalhado.Minha esperança,
No entanto, é que ele recompensará um estudo cuidadoso e que o tempo você
Passe a leitura será facilmente recuperado na forma de maior
produtividade de programação.

As edições anteriores deste livro incluíram uma referência abrangente
seção.Não sinto mais que faz sentido incluir esse material em
formulário impresso quando é tão rápido e fácil de encontrar referência atualizada
material online.Se você precisar procurar algo relacionado ao núcleo ou
JavaScript do lado do cliente, recomendo que você visite o site da MDN.E
Para APIs do nó do lado do servidor, recomendo que você vá diretamente para a fonte
e consulte a documentação de referência Node.js.

Convenções usadas neste livro

Eu uso as seguintes convenções tipográficas neste livro:

itálico
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, e
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_defgd7

Este 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 do
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 de O'Reilly oferece acesso sob demanda ao vivo cursos de treinamento, caminhos de aprendizagem detalhados, codificação interativa ambientes e uma vasta coleção de texto e vídeo de 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 erratas, 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 nosso
Site em http://www.oreilly.com.
Encontre -nos no Facebook: http://facebook.com/oreilly
Siga -nos no Twitter: http://twitter.com/oreillymedia
Assista ao YouTube: http://www.youtube.com/oreillymedia
Agradecimentos
Muitas pessoas ajudaram na criação deste livro.Eu gostaria de
Graças à minha editora, Angela Rufino, por me manter no caminho certo e por ela
Paciência com meus prazos perdidos.Obrigado também ao meu técnico
Revisores: Brian Sletten, Elisabeth Robson, Ethan Flanagan,
Maximiliano Firtman, Sarah Wachs e Schalk Neethling.Deles
Comentários e sugestões fizeram deste um livro melhor.
A equipe de produção de O'Reilly fez o seu bom trabalho habitual: Kristen
Brown gerenciou o processo de produção, Deborah Baker foi o
A editora de produção, Rebecca DeMarest, atraiu as figuras, e Judy
McConville criou o índice.
Editores, revisores e colaboradores de edições anteriores deste livro
incluiu: Andrew Schulman, Angelo Sirigos, Aristóteles Pagaltzis,
Brendan Eich, Christian Heilmann, Dan Shafer, Dave C. Mitchell, Deb
Cameron, 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 meu ausências.

David Flanagan, março de 2020

Capítulo 1. Introdução a JavaScript

JavaScript é a linguagem de programação da web.O avassalador
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 nó significa que JavaScript agora também é a linguagem de programação mais usada entre desenvolvedores de software.Se você está começando do zero ou é
Já 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 relacionado.JavaScript deriva suas funções de primeira classe do esquema e 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 um superficial semelhança sintática, JavaScript é completamente diferente do Java

linguagem de programação.E JavaScript há muito tempo superou seu
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 - o
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
chama o idioma JavaScript.Este livro usa o nome "ecmascript" e a abreviação "es" para
Consulte o padrão de idioma e as versões desse padrão.
Na maior parte dos 2010, a versão 5 do padrão Ecmascript foi suportada por toda a web
navegadores.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 grave, de uso geral
idioma adequado para engenharia de software em larga escala.Desde o ES6, a especificação do ECMAScript tem mudou -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 quão 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 ?Use
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
disponível nesses contextos.Este livro abordará os recursos legados do JavaScript, mas é cuidadoso para
aponte 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
A 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, e
Este ainda é o ambiente de execução mais comum para JavaScript
código.O ambiente do navegador da web permite que o código JavaScript obtenha
entrada do mouse e teclado do usuário e fazendo http
solicitações.E permite que o código JavaScript exiba saída para o usuário
com HTML e CSS.
Desde 2010, outro ambiente host está disponível para JavaScript
código.Em vez de restringir o JavaScript para trabalhar com as APIs
Fornecido por um navegador da web, o nó dá acesso a JavaScript a todo
sistema operacional, permitindo que os programas JavaScript leiam e gravem arquivos,
Envie e receba dados sobre a rede e faça e sirva HTTP
solicitações.O nó é uma escolha popular para implementar servidores da web e
também uma ferramenta conveniente para escrever scripts simples de utilitário como alternativa
scripts de conchas.
A maior parte deste livro está focada na própria linguagem JavaScript.
Capítulo 11 documenta a biblioteca padrão JavaScript, capítulo 15
Apresenta o ambiente do host do navegador da web e o capítulo 16
Apresenta o ambiente do host do nó.
Este livro abrange os fundamentos de baixo nível primeiro e depois se baseia neles
a abstrações mais avançadas e de nível superior.Os capítulos são
destinado a ser lido mais ou menos em ordem.Mas aprender um novo
A linguagem de programação nunca é um processo linear, e descrevendo um
A linguagem também não é linear: cada recurso de idioma está relacionado a outros
Recursos, e este livro está cheio de referências cruzadas-às vezes

para trás e às vezes para a frente - para material relacionado.Esse
O capítulo introdutório 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
capítulo 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 Firefox

Outra maneira de experimentar o código JavaScript é baixar e instalar o nó de 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

sessão interativa de javascript como esta:

```
$ node
Bem -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 arquivo
Pressione ^c para abortar a expressão atual, ^d para sair do repl
> Seja x = 2, y = 3;
indefinido
> x + y
5
> (x === 2) && (y === 3)
verdadeiro
> (x> 3) ||(y <3)
falso
```

1.2 Hello World

Quando 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.js
```

Se 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 um
Hello.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 0
x // => 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 Curly
aparelho 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 indexado
listas) 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 no
variedade.
primos [prime.length-1] // => 7: o último elemento do
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 propriedades
Trial1: [[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 código

Você 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 e verifica 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 um 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 Brace Curly é conhecido como expressão inicializadora, e é apenas um dos os tópicos do capítulo 4. Uma expressão é uma frase de javascript que pode ser avaliado 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 um expressão.

Uma das maneiras mais comuns de formar expressões em JavaScript é

Use operadores:
// Os operadores agem sobre valores (os operando) para produzir um novo valor.
// Os operadores aritméticos são alguns dos mais simples:
3 + 2 // => 5: adição
3 - 2 // => 1: Subtração
3 * 2 // => 6: multiplicação
3/2 // => 1.5: Divisão
pontos [1] .x - pontos [0] .x // => 1: operando mais complicado
Também trabalhe
"3" + "2" // => "32": + adiciona números,
Concatenados
// javascript define alguns operadores aritméticos abreviados
deixe count = 0;// Defina uma variável
contagem ++;// incremento a variável
contar--;// diminuir a variável
contagem += 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: igualdade
x! == y // => true: desigualdade
x <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 diferente
"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
valor
Se as expressões JavaScript são como frases, as declarações JavaScript
são como frases completas.Declarações são o tópico do capítulo 5. Aproximadamente,
Uma expressão é algo que calcula um valor, mas não faz
Qualquer coisa: não altera o estado do programa de forma alguma.Declarações, em
Por outro lado, não tem um valor, mas eles alteram o estado.Você tem
Declarações variáveis ??e declarações de atribuição acima.O outro
ampla categoria de declaração são estruturas de controle, como condicionais
e loops.Você verá exemplos abaixo, depois de cobrirmos funções.
Uma função é um bloco nomeado e parametrizado de código JavaScript que
Você define uma vez e pode invocar repetidamente.Funções
não são cobertos formalmente até o capítulo 8, mas como objetos e matrizes,
Você os verá muitas vezes antes de chegar a esse capítulo.Aqui estão
Alguns exemplos simples:
// As funções são blocos parametrizados de código JavaScript que
Podemos invocar.
função plus1 (x) {// define uma função chamada "plus1"
com o parâmetro "X"
retornar x + 1;// retorna um valor maior que
O valor passou em
} // As funções são fechadas em Curly
aparelho ortodôntico
Plus1 (y) // => 4: y é 3, então este
Retornos de invocação 3+1
Deixe Square = function (x) {// As funções são valores e podem ser
atribuído a vars

retornar x * x;// Calcule o valor da função
};// semicolon marca o fim do
atribuição.
quadrado (mais1 (y)) // => 16: Invoque duas funções em
uma expressão
No 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
Passe uma função sem nome como argumento para outra função.O
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 + 1
const square = x => x * x;// A entrada X mapeia para a saída
x * x
Plus1 (y) // => 4: A invocação da função é
o mesmo
quadrado (mais1 (y)) // => 16
Quando 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 vazia
a.push (1,2,3);// o método push () adiciona elementos
para uma matriz
a. reverse ();// Outro método: reverter o
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 pontos
Array de antes.
pontos.dist = function () {// define um método para calcular

distância entre pontos
Seja p1 = this [0];// Primeiro elemento da matriz somos
invocou
Seja P2 = este [1];// segundo elemento do "isto"
objeto
Seja a = p2.x-p1.x;// diferença em coordenadas x
Seja b = p2.y-p1.y;// diferença nas coordenadas y
Retornar Math.sqrt (A*A + // O Teorema Pitagórico
b*b);// math.sqrt () calcula o quadrado
raiz
};
Points.dist () // => Math.sqrt (2): Distância
Entre nossos 2 pontos
Agora, como prometido, aqui estão algumas funções cujos corpos demonstram
Declarações de estrutura de controle JavaScript comum:
// declarações JavaScript incluem condicionais e loops usando
a sintaxe
// de C, C ++, Java e outros idiomas.
função abs (x) {// uma função para calcular o
valor absoluto.
if (x> = 0) {// a instrução if ...
retornar x;// executa este código se o
A comparação é verdadeira.
} // Este é o fim do IF
cláusula.
else {// a cláusula opcional mais
executa seu código se
retornar -x;// A comparação é falsa.
} // aparelho encaracolado opcional quando 1
declaração por cláusula.
} // Nota As declarações de retorno aninhadas
dentro se/else.
ABS (-10) === ABS (10) // => true
Função Sum (Array) {// Calcule a soma dos elementos
de uma matriz
deixe soma = 0;// Comece com uma soma inicial de 0.
para (deixe x de matriz) {// loop sobre a matriz, atribuindo cada
elemento para x.

```
soma += x;;// Adicione o valor do elemento ao
soma.
} // Este é o fim do loop.
soma de retorno;// retorna a soma.
}
soma (primos) // => 28: soma dos primeiros 5
Prima 2+3+5+7+11
função fatorial (n) {// uma função para calcular
fatoriais
deixe o produto = 1;// Comece com um produto de 1
while (n> 1) {// repita declarações em {} while
expr in () é verdadeiro
produto *= n;// atalho para produto = produto
* n;
n--;// atalho para n = n - 1
} // Fim do loop
produto de retorno;// retorna o produto
}
Fatorial (4) // => 24: 1*4*3*2
função fatorial2 (n) {// outra versão usando um
loop diferente
Deixe eu, produto = 1;// Comece com 1
para (i = 2; i <= n; i ++) // incremento automaticamente i de
2 até n
produto *= i;// Faça isso cada vez.{} não
necessário para loops de 1 line
produto de retorno;// retorna o fatorial
}
Fatorial2 (5) // => 120: 1*2*3*4*5
```

JavaScript suporta um estilo de programação orientado a objetos, mas é significativamente diferente da programação "clássica" orientada a objetos idiomas.O capítulo 9 abrange a programação orientada a objetos em JavaScript em detalhes, com muitos exemplos.Aqui está um muito simples exemplo que demonstra como definir uma classe JavaScript para representar Pontos geométricos 2D.Objetos que são casos desta classe têm um

método único, denominado distância (), que calcula a distância do
ponto da origem:
classe Point {// por convenção, os nomes da classe são
capitalizado.
construtor (x, y) {// função construtora
Inicialize novas instâncias.
this.x = x;// essa palavra -chave é o novo objeto
sendo inicializado.
this.y = y;// armazenar argumentos de função como
Propriedades do objeto.
} // Nenhum retorno é necessário em
Funções do construtor.
distance () {// Método para calcular a distância de
origem para apontar.
Retornar Math.Sqrt (// Retornar a raiz quadrada de x^2 +
y^2.
this.x * this.x + // refere -se ao ponto
objeto em qual
this.y * this.y // o método de distância é
invocado.
);
}
}
// Use a função do construtor Point () com "novo" para criar
Objetos pontuais
Seja p = novo ponto (1, 1);// O ponto geométrico (1,1).
// Agora use um método do objeto Point P
p.distance () // => math.sqrt2
Este passeio introdutório da sintaxe fundamental de JavaScript e
Os recursos termina aqui, mas o livro continua com independente
Capítulos 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
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 e
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 cobre 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 nó
Apresenta o ambiente do host do nó, cobrindo o fundamental
modelo de programação e as estruturas de dados e APIs que são mais
importante para entender.
Capítulo 17, Ferramentas e extensões JavaScript
Cobre ferramentas e extensões de idiomas que valem a pena saber sobre
Porque eles são amplamente utilizados e podem torná -lo um mais produtivo
programador.
1.4 Exemplo: Frequência do personagem
Histogramas
Este 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.js
T: ########## 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 estar
garantiu 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 do
maioria
* caracteres usados ??com frequência.Requer o nó 12 ou superior a
correr.
*
* 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
classe DefaultMap estende o mapa {
construtor (defaultValue) {
super();// Invoque a Superclass
construtor
this.defaultValue = defaultValue;// lembre -se do
valor padrão
}
Get (key) {
if (this.has (key)) {// se a chave for
já no mapa
retornar super.get (chave);// retorna seu valor
da superclass.
}
outro {
retornar this.defaultValue;// de outra forma retornar
o valor padrão
}
```

```
}
}
// Esta classe calcula e exibe histogramas de frequência de carta
classe Histograma {
construtor () {
this.LetterCounts = new DefaultMap (0);// mapa de
cartas para contar
this.ToTalletters = 0;// Quantos
letras em tudo
}
// Esta função atualiza o histograma com as letras de
texto.
add (text) {
// remova o espaço em branco do texto e converta para
maiúscula
text = text.replace (/\ s/g, "") .toupPercase ();
// agora percorre os personagens do texto
para (deixe o caráter do texto) {
Let count = this.letterCounts.get (caractere);//
Obtenha uma contagem antiga
this.letterCounts.set (caractere, contagem+1);//
Incrementá -lo
this.Totalletters ++;
}
}
// converte o histograma em uma string que exibe um ASCII
gráfico
tostring () {
// converte o mapa em uma matriz de matrizes [chave]
deixe entradas = [... this.letterCounts];
// Classifique a matriz por contagem e depois em ordem alfabética
entres.sort ((a, b) => {// uma função para
Defina a ordem de classificação.
if (a [1] === b [1]) {// se as contagens
são iguais
retornar A [0] <b [0]?-1: 1;// organizar
alfabeticamente.
} else {// se as contagens
```

```
diferir
retornar b [1] - a [1];// classificar por maior
contar.
}
});
// converte as contagens em porcentagens
para (deixe a entrada de entradas) {
entrada [1] = entrada [1] / this.ToTalletters*100;
}
// soltar qualquer entradas inferiores a 1%
entradas = entradas.Filter (Entrada => Entrada [1]> = 1);
// agora converte cada entrada em uma linha de texto
Deixe linhas = entradas.map (
([l, n]) => `$ {l}: $ {"#". Repita (math.round (n))}
$ {n.tofixed (2)}%`
);
// e retorne as linhas concatenadas, separadas por
personagens newline.
retorno linhas.Join ("\ n");
}
}
// Esta função Async (Promise Returning) cria um histograma
objeto,
// lê de forma assíncrona pedaços de texto da entrada padrão e
adiciona esses pedaços a
// o histograma.Quando chega ao final do fluxo, ele
Retorna este histograma
Função assíncreada histogramfromstdin () {
process.stdin.setEncoding ("UTF-8");// Leia Unicode
cordas, não bytes
Seja 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 (histogram.toString ());});
1.5 Resumo
Este livro explica JavaScript de baixo para cima.Isso significa que nós
Comece com detalhes de baixo nível, como comentários, identificadores, variáveis ??e
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 um
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
e domine os detalhes depois de ter um conhecimento prático do
linguagem como um todo.

Capítulo 2. Estrutura lexical

A estrutura lexical de uma linguagem de programação é o conjunto de
regras elementares que especificam como você escreve programas nisso
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 linha
Comentários
Literais
Identificadores e palavras reservadas
Unicode
Semicolons opcionais

2.1 O texto de um programa JavaScript

JavaScript é uma linguagem sensível ao caso.Isso significa que a linguagem
Palavras -chave, variáveis, nomes de funções e outros identificadores devem sempre
ser digitado 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 um
exceção).Porque você pode usar espaços e linhas de novo livremente em seu
programas, você pode formatar e recuar seus programas em um elegante e
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 A //
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
aninhado.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 javascript
Identificador deve começar com uma carta, um sublinhado (_) ou um sinal de dólar
($).Caracteres subsequentes podem ser cartas, dígitos, sublinhados ou dólar
Sinais.(Os dígitos não são permitidos como o primeiro caractere para que JavaScript
pode facilmente distinguir identificadores dos números.) Estes são todos legais
Identificadores:
eu
my_variable_name
v13
_fictício
$ str
Como qualquer idioma, JavaScript se reserva certos identificadores para uso pelo
linguagem em si.Essas "palavras reservadas" não podem ser usadas como regular
identificadores.Eles estão listados na próxima seção.

2.4.1 Palavras reservadas

As seguintes palavras fazem parte da linguagem JavaScript.Muitos de estes (como se, enquanto e para) são palavras -chave reservadas que devem não ser usado como nomes de constantes, variáveis, funções ou classes (embora todos possam ser usados ??como nomes de propriedades dentro de um objeto).Outros (como de, de, obter e se estabelecer) são usados ??em limitados contextos sem ambiguidade sintática e são perfeitamente legais como identificadores.Ainda outras palavras -chave (como Let) não podem ser totalmente reservadas para manter a compatibilidade com versões anteriores com programas mais antigos e, portanto, Existem regras complexas que governam quando podem ser usadas como identificadores e quando não podem.(vamos ser usados ??como um nome de variável se declarado com var fora de uma aula, por exemplo, mas não se declarado dentro de uma aula ou com const.) O curso mais simples é evitar usar qualquer 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 nulo
vazio
Assync continuam se estende se disso
enquanto
Aguarde o depurador False Import Return Throw
com
quebrar o padrão finalmente no set true
colheita
Excluir casos, por exemplo, de tentativa estática
Catch Do de Let Super Typeof
Classe mais função

JavaScript também se reserva ou restringe o uso de certas palavras -chave que são atualmente não usado pelo idioma, mas isso pode ser usado no futuro versões:

Enum implementos Pacote de interface Protegido Private

público
Por razões históricas, argumentos e avaliação não são permitidos como
identificadores em certas circunstâncias e são melhor evitados inteiramente.
2.5 Unicode
Os 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 Unicode
Alguns hardware e software de computador não podem exibir, entrar ou
Processe 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 \ u
e 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 um
Nome da variável que inclui este personagem:
Deixe Café = 1;// Defina uma variável usando um caractere unicode
caf \ u00e9 // => 1;Acesse a variável usando uma fuga
Sequência
caf \ u {e9} // => 1;Outra forma da mesma fuga
Sequência
As primeiras versões do JavaScript suportaram apenas a fuga de quatro dígitos
Sequência.A versão com aparelho encaracolado foi introduzido em ES6 para
apoiar melhor os pontos de codepates unicode que requerem mais de 16 bits, como
Como emoji:
console.log ("\ u {1f600}");// imprime um rosto sorridente emoji
O Unicode Escapes também pode aparecer nos comentários, mas desde os comentários
são ignorados, eles são simplesmente tratados como caracteres ASCII nesse contexto
e não interpretado como unicode.

2.5.2 Normalização unicode

Se você usa caracteres não-ASCII em seus programas JavaScript, você
deve estar ciente de que o Unicode permite mais de uma maneira de codificar o
mesmo personagem.A string "é", por exemplo, pode ser codificada como o
caractere unicode único \ u00e9 ou como um ASCII regular ?e? seguido
pelo sotaque agudo combinando Mark \ U0301.Essas duas codificações
normalmente parece exatamente o mesmo quando exibido por um editor de texto, mas
Eles têm codificações binárias diferentes, o que significa que são consideradas
Diferente por JavaScript, que pode levar a programas muito confusos:

```
const Café = 1;// Esta constante é nomeada "Caf \ u {e9}"
const Café = 2;// Esta constante é diferente: "Cafe \ u {301}"
Café // => 1: esta constante tem um valor
Café // => 2: Esta constante indistinguível tem um
valor diferente
```

O padrão Unicode define a codificação preferida para todos os personagens e especifica um procedimento de normalização para converter texto em um canônico forma adequada para comparações.JavaScript assume que o código -fonte está interpretando já foi normalizada e não faz nenhum normalização por conta própria.Se você planeja usar caracteres unicode em seu Programas JavaScript, você deve garantir que seu editor ou algum outro A ferramenta executa a normalização unicode do seu código -fonte para prevenir você de acabar com diferente, mas visualmente indistinguível 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 o 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 marque explicitamente os fins das declarações, mesmo onde elas não estão obrigatório.Outro estilo é omitir semicolons sempre que possível, usando eles apenas nas poucas situações que os exigem.Qualquer estilo que você

Escolha, existem alguns detalhes que você deve entender sobre opcional
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 um
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 a
poderia ficar sozinho como a afirmação A;, mas JavaScript não trata o

Segunda linha quebra como um semicolon porque pode continuar analisando o
Declaração mais longa a = 3;.
Essas regras de rescisão de declaração levam a alguns casos surpreendentes.Esse
O 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 um
Invocação de função de F a partir da primeira linha, e JavaScript interpreta
O código como este:
Seja y = x + f (a + b) .ToString ();
É mais provável que não, essa não é a interpretação pretendida pelo
autor do código.Para trabalhar como duas declarações separadas, um
O semicolon explícito é necessário neste caso.
Em geral, se uma declaração começa com (, [, /, +, ou -, há uma chance
que isso poderia ser interpretado como uma continuação da declaração antes.
Declarações que começam com /, +e - são bastante raras na prática, mas
declarações começando com (e [não são incomuns, pelo menos em
Alguns estilos de programação JavaScript.Alguns programadores gostam de colocar
um semicolon defensivo no início de qualquer declaração para que
continuará a funcionar corretamente, mesmo que a declaração antes seja
Modificado e um semicolon de terminação anteriormente removido:
Seja x = 0 // semicolon omitido aqui
; [x, x+1, x+2] .foreach (console.log) // defensivo;mantém isso
declaração separada

Existem três exceções à regra geral que o JavaScript interpreta
A linha quebra como semicolons quando não pode analisar a segunda linha como um
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 em um não óbvio
maneira que é difícil de depurar.
A segunda exceção envolve os operadores ++ e - (§4.8).Esses
Os operadores podem ser operadores de prefixo que aparecem antes de uma expressão ou
operadores pós -fix que aparecem após 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 no nível 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 representado e manipulado em uma linguagem de programação são conhecidos como tipos, e uma das características mais fundamentais de um A linguagem de programação é o conjunto de tipos que ele suporta.Quando um programa precisa manter um valor para uso futuro, ele atribui o valor a (ou "lojas" o valor em) uma variável.Variáveis ??têm nomes e permitem o uso de Esses nomes em nossos programas para se referir a valores.A maneira como as variáveis O trabalho 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 linguagem
Extensões 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, um
Símbolo, nulo ou indefinido é um objeto.Um objeto (isto é, um
membro do objeto de tipo) é uma coleção de propriedades onde cada um
a 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 nome
valores.O idioma também define um tipo especial de objeto, conhecido como um
Array, que representa uma coleção ordenada de valores numerados.O
A 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.UM
O objeto de mapa representa um mapeamento de chaves para valores.Vários ?digitados
tipos de matriz ?facilitam operações em matrizes de bytes e outros binários
dados.O tipo regexp representa padrões textuais e permite
Combinação sofisticada, pesquisando e substituindo operações em strings.
O tipo de data representa datas e horários e suporta rudimentar
data 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 e
As aulas não fazem apenas parte da sintaxe da linguagem: elas são elas mesmas
Valores que podem ser manipulados por programas JavaScript.Como qualquer
Valor de javascript que não é um valor primitivo, funções e classes são
um tipo especializado de objeto.Eles são cobertos em detalhes nos capítulos 8
e 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
o programa não tem mais maneira de se referir a ele - o intérprete sabe disso
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 ??- e
Portanto, 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
trabalhando 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 apenas
Objetos 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 ser
invocou.
Os tipos de objetos de JavaScript são mutáveis ??e seus tipos primitivos são
imutável.Um valor de um tipo mutável pode mudar: um javascript
O 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 um
número, 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 explorados
Mais adiante em §3.8.
JavaScript converte liberalmente valores de um tipo para outro.Se a
o programa espera uma string, por exemplo, e você dá um número, vai
Converta automaticamente o número em uma string para você.E se você usar um
valor não-booleano em que é esperado um booleano, o JavaScript será convertido
de acordo.As regras para conversão de valor são explicadas no §3.9.
As regras de conversão de valor liberal do Javascript afetam sua definição de
igualdade e o operador de igualdade == executa conversões de tipo
descrito em §3.9.1.(Na prática, no entanto, o operador de igualdade == é
descontinuado a favor do estrito operador de igualdade ===, que não faz
Digite 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 declarado com LET (ou com var no código JavaScript mais antigo).JavaScript Constantes e variáveis ??não são criadas: as declarações não especificam o que tipo de valores serão atribuídos.Declaração e atribuição variáveis são cobertos 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 representado e manipulado em JavaScript.Começaremos mergulhando certo nos 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 IEEE 754 padrão, o que significa que pode representar números tão grandes quanto ± 1,7976931348623157 × 10 e tão pequeno quanto ± 5 × 10 .

O formato do número JavaScript permite que você represente exatamente todos Inteiros entre -9.007.199.254.740.992 (-2) e 9.007.199.254.740.992 (2), inclusive.Se você usar valores inteiros maiores Do que isso, você pode perder precisão nos dígitos à direita.Nota, no entanto, que certas operações em JavaScript (como indexação de matrizes e o Os operadores bitwise descritos no capítulo 4) são realizados 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

1

308

-324

53

53

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:

```
0
3
10000000
```

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 de
o número.
Os literais de ponto flutuante também podem ser representados usando exponencial
Notação: um número real seguido pela letra e (ou e), seguida por um
Sinal opcional Plus ou Minus, seguido de um expoente inteiro.Esse
a notação representa o número real multiplicado por 10 ao poder de
o expoente.
Mais sucintamente, a sintaxe é:
[dígitos] [. dígitos] [(e | e) [(+|-)] dígitos]
Por exemplo:
3.14
2345.6789
.3333333333333333333
6.02E23 // 6,02 × 10^{23}
1.4738223E-32 // 1.4738223 × 10?^{32}
Separadores em literais numéricos
Você 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 como 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
padronizado como parte do JavaScript.Mas eles estão nos estágios avançados da padronização
processo e são implementados por todos os principais navegadores e por nó.

3.2.3 aritmética em javascript

Os 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.Completo
Detalhes 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 o
potência 53
Math.Round (.6) // => 1,0: arredondado para o mais próximo
Inteiro
Math.ceil (.6) // => 1.0: arredondado para um número inteiro
Math.floor (.6) // => 0,0: Recupela para um número inteiro
Math.abs (-5) // => 5: valor absoluto
Math.max (x, y, z) // retorna o maior argumento
Math.min (x, y, z) // retorna o menor argumento
Math.random () // pseudo-random número x onde 0 <=
x <1,0
Math.pi // ?: circunferência de um círculo /
diâmetro
Math.e // e: a base do natural
logaritmo
Math.sqrt (3) // => 3 ** 0.5: a raiz quadrada de 3
Math.pow (3, 1/3) // => 3 ** (1/3): a raiz do cubo de 3
Math.sin (0) // Trigonometria: também Math.cos,
Math.atan, etc.
Math.log (10) // Logaritmo natural de 10
Math.log (100) /math.ln10 // base 10 logaritmo de 100
Math.log (512) /math.ln2 // base 2 logaritmo de 512
Math.exp (3) // math.e cubado
```

ES6 define mais funções no objeto de matemática:
Math.cbrt (27) // => 3: raiz de cubo
Math.hypot (3, 4) // => 5: raiz quadrada da soma dos quadrados de
todos os argumentos
Math.Log10 (100) // => 2: Logaritmo Base-10
Math.log2 (1024) // => 10: LOGARITHM BASE-2
Math.log1p (x) // log natural de (1+x);preciso para muito
pequeno x
Math.expm1 (x) // math.exp (x) -1;o inverso de
Math.log1p ()
Math.sign (x) // -1, 0 ou 1 para argumentos <, ==, ou> 0
Math.imul (2,3) // => 6: multiplicação otimizada de 32 bits
Inteiros
Math.clz32 (0xf) // => 28: Número de zero bits líderes em um
Inteiro de 32 bits
Math.trunc (3.9) // => 3: converter em um número inteiro truncando
parte fracionária
Math.Fround (X) // Round to mais próximo do número de flutuação de 32 bits
Math.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
O maior número negativo representável, o resultado é o infinito negativo, -
Infinidade.Os valores infinitos se comportam como seria de esperar: acrescentando,
subtrair, multiplicar ou dividi -los 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 quase
Completamente indistinguível de Zero e JavaScript regulares
Os 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 por
Zero não tem um valor bem definido e o resultado desta operação
é o valor especial, não um número, Nan.Nan também surge se você tentar
Para dividir o infinito pelo infinito, pegue a raiz quadrada de um número negativo,
ou use operadores aritméticos com operandos não numéricos que não podem ser
convertido em números.
JavaScript predefine as constantes globais infinitas e a nan para manter o
infinito positivo e valor não um número, e esses valores também são
Disponível como propriedades do objeto numérico:
Infinito // um número positivo muito grande para
representar
Número.positive_infinity // mesmo valor
1/0 // => infinito
Número.max_value * 2 // => infinito;transbordamento
-Infinity // um número negativo muito grande para
representar
Número.negative_infinity // o mesmo valor
-1/0 // => -infinity
-Number.max_value * 2 // => -infinity
Nan // o valor não-number
Número.nan // o mesmo valor, escrito
Outra maneira
0/0 // => nan
Infinito/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 ES6
Número.ParseInt () // O mesmo que o parseint global ()
função
Número.parsefloat () // O mesmo que o parsefloat global ()
função
Número.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 - 1
Número.epsilon // => 2 **-52: menor diferença
entre números
O 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 ().Isto
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 -Infinidade.O global
A 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
deixe negz = -0;// Zero negativo
zero === negz // => true: zero e zero negativo são
igual
1/zero === 1/negz // => false: infinito e -infinity são
não é igual
```

3.2.4 Ponto flutuante binário e erros de arredondamento

Existem infinitamente muitos números reais, mas apenas um número finito de
eles (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 de
O 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 o
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 o
Os valores x e y no código mostrado aqui estão muito próximos um do outro e
para o 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
Programas, 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, tem sido
implementado em Chrome, Firefox, Edge e Nó, e há um
Implementação em andamento no Safari.Como o nome indica, Bigint é um
Tipo numérico cujos valores são inteiros.O tipo foi adicionado a
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
Carta 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" .Resepat (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
Recunda (em direção a zero):
1000n + 2000n // => 3000n
3000N - 2000N // => 1000N
2000n * 3000n // => 6000000n
3000N / 997N // => 3N: O quociente é 3
3000n % 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
Digite BIGINT com operando de números regulares.Isso pode parecer confuso em
Primeiro, mas há uma boa razão para isso.Se um tipo numérico fosse mais
Geral que o outro, seria fácil definir aritmética em misto
operando para simplesmente retornar um valor do tipo mais geral.Mas também não
Tipo é mais geral que o outro: Bigint pode representar extraordinariamente
valores grandes, tornando -o mais geral que os números regulares.Mas Bigint
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 como
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 Datas e horários

JavaScript define uma classe de data simples para representar e
manipular os números que representam datas e horários.JavaScript
Datas 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
verá 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 um
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 ele 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 o caracteres da string dentro de um par correspondente de solteiro ou duplo citações ou backsks ('ou "ou`).
Backticks podem estar contidos em strings delimitados por uma única citação 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' ", ele disse.
Strings delimitadas com backsticks são uma característica do ES6 e permitir 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 +.Como
do ES5, no entanto, você pode quebrar uma corda literal em várias linhas por
terminando 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
Várias linhas e, neste caso, os terminadores de linha fazem parte do
String literal:

```
// Uma string representando 2 linhas escritas em uma linha:
'Duas \ nlines'
// 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
tenha cuidado com as contrações e possessivos em inglês, como não pode e
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
Apostróficos 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 seqüências de seqüências 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 Eventytler:
<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 apóstrofo 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
representar.Três seqüências 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 um
a 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
ence
Personagem representado
\ 0
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 carruagem (\ u000d)
\ "
Citação dupla (\ u0022)
\ '
Apóstrofo ou cotação única (\ u0027)
\\
Barragem (\ u005c)
\ xnn
O caractere unicode especificado pelos dois dígitos hexadecimais nn
\ Unn
nn
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 de
O idioma pode, 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 vários
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:

```
deixe 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 em
Exatamente a mesma sequência de valores de 16 bits.Strings também podem ser
comparado com os operadores <, <=,> e> = =.Comparação de string é
feito simplesmente comparando os valores de 16 bits.(Para localidade mais robusta-
Comparação e classificação de string cientes, consulte §11.7.3.)
Para determinar o comprimento de uma corda-o número de 16 bits o valor
contém - use a propriedade de comprimento da string:

```
S. Length
```

Além desta propriedade de comprimento, o JavaScript fornece uma API rica para
Trabalhando com strings:

```
Seja s = "Olá, mundo";// Comece com algum texto.
// obtendo partes de uma corda
s.substring (1,4) // => "ell": o 2º, 3º e 4º
caracteres.
s.slice (1,4) // => "ell": a mesma coisa
s.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 tarde
s.startswith ("inferno") // => true: a string começa com
esses
s.endswith ("!") // => false: s não termina com isso
s.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 no
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 a
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" .TriMend () // => "teste": remova os espaços em
certo.Também Trimright
// métodos de string diversos
s.concat ("!") // => "Olá, mundo!": Apenas use +
operador em vez disso
"<>". Repita (5) // => "<> <> <> <> <>": concatenar n
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 quadrado
Suportes 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 backsticks:

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 daqueles
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
sinal 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 Escape
sequência, e pelo menos quatro novas linhas (os valores de expressão podem incluir
NEWLINES também):

Deixe errorMessage = `\
\ U2718 Falha no teste em $ {FileName}: $ {linenumber}:
$ {excepcion.message}

Stack Trace:
$ {excepcion.stack}
`;
A barra de barriga no final da primeira linha aqui escapa do inicial
newline para que a string resultante comece com o unicode ?
Personagem (\ U2718) em vez de uma nova linha.
Literais de modelo marcados
Um 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 modelo
Literais são passados ??para a função.O valor deste ?modelo marcado
literal ?é o valor de retorno da função.Isso pode ser usado, para
exemplo, para aplicar HTML ou SQL escapando para os valores antes
substituindo -os no texto.
O ES6 possui uma função de tag embutida: string.raw ().Ele retorna o texto
Dentro de backticks sem nenhum processamento de backslash escapas:
`\ n`.length // => 1: a string tem um único
personagem newline
String.raw` \ n`.length // => 2: um caractere de barriga e o
Carta n
Observe que, embora a parte da tag de um modelo marcado literal seja um
Função, não há parênteses usados ??em sua invocação.Neste mesmo
Caso específico, os caracteres de backtick substituem o aberto e o fechamento
parênteses.
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 regular
Os 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
/\ bjavascript \ 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
deixe padrão = /\ d+ /g;// corresponde a todas as instâncias de
um ou mais dígitos
```

Pattern.test (texto) // => True: Existe uma correspondência
text.search (padrão) // => 9: posição do primeiro
corresponder
text.match (padrão) // => ["1", "2", "3"]: matriz
de todas as partidas
text.replace (padrão, " #") // => "teste: #, #, #"
text.split (/\ d+/) // => ["", "1", "2", "3"]:
dividido em não "
3.4 valores booleanos
Um valor booleano representa a verdade ou a falsidade, dentro ou fora, sim ou não.
Existem apenas dois valores possíveis desse tipo.As palavras reservadas
Verdadeiro e falso avaliar para esses dois valores.
Os valores booleanos são geralmente o resultado de comparações que você faz em
Seus programas JavaScript.Por exemplo:
a === 4
Este código testa para ver se o valor da variável A é igual ao
número 4. Se for, o resultado dessa comparação é o valor booleano
verdadeiro.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 um
ação se um valor booleano for verdadeiro e outra ação se o valor for
falso.Você geralmente combina uma comparação que cria um booleano
valor 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, é
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, funcionam
Tipo, falso:
indefinido
nulo
0
-0
Nan
"" // a corda vazia
Todos os outros valores, incluindo todos os objetos (e matrizes) se convertem e trabalham
Tipo, 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 false
E 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 um se
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 se
Somente se O não for falso ou qualquer valor falsamente (como nulo ou
indefinido).Qual afirmação se for apropriada para o seu programa
realmente depende de quais valores você espera ser atribuído a o.Se você
precisa distinguir nulo de 0 e "", então você deve usar um
comparação explícita.
Os valores booleanos têm um método tostring () que você pode usar
converte -os para as cordas "verdadeiras" ou "falsas", mas elas não têm nenhum
Outros métodos úteis.Apesar da API trivial, existem três importantes
operadores booleanos.
O operador && executa o booleano e a operação.Ele avalia
um 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 ou
operação: ele avalia um valor verdadeiro se um (ou ambos) de seus
operands é verdade e avalia um valor falsamente se ambos os operando forem
falsidade.Finalmente, o unário!O operador executa o booleano não
Operação: Avalia -se para TRUE se seu operando for falsamente e avaliar
Falso 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 O operador no NULL retorna a string "Objeto", indicando que o NULL pode Seja 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 um 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 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 retorne 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 um distinção importante na prática) que é inicializada para o indefinido valor.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 distingui -los.) Ambos são valores falsamente: eles se comportam
Como false quando um valor booleano é necessário.Nem nulo nem
indefinido têm propriedades ou métodos.De fato, usando.ou [] para
Acesse uma propriedade ou método desses valores causa um TypeError.
Considero indefinido 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
indefinido 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 um
função, eu geralmente uso nulo.Alguns programadores se esforçam para evitar nulos
inteiramente e use indefinido em seu lugar onde puder.

3.6 Símbolos

Os 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
deixe 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 símbolo
Seja o = {};// Crie um novo objeto
o [strname] = 1;// Defina uma propriedade com um
Nome da string
o [symname] = 2;// Defina uma propriedade com um
Nome do símbolo
o [strname] // => 1: Acesse a string-
propriedade nomeada
o [symname] // => 2: Acesse o símbolo-
propriedade nomeada
O tipo de símbolo não possui uma sintaxe literal.Para obter um símbolo
valor, você chama a função Symbol ().Esta função nunca retorna
O mesmo valor duas vezes, mesmo quando chamado com o mesmo argumento.Esse
significa que, se você chama Symbol () para obter um valor de símbolo, você pode
use com segurança esse valor como nome de propriedade para adicionar uma nova propriedade a um
objeto e não precisa se preocupar com o fato de você estar substituindo um
Propriedade existente com o mesmo nome.Da mesma forma, se você usa simbólico
nomes de propriedades e não compartilham esses símbolos, você pode estar confiante
que outros módulos de código em seu programa não acidentalmente
substitua suas propriedades.
Na prática, os símbolos servem como um mecanismo de extensão de linguagem.Quando
O ES6 introduziu os objetos for/of loop (§5.4.4) e iterável
(Capítulo 12), precisava definir o método padrão que as classes poderiam
implementar para se tornar iterável.Mas padronizando qualquer
nome de string específico para este método de iterador teria quebrado
Código existente, então um nome simbólico foi usado.Como veremos em
Capítulo 12, Symbol.iterator é um valor de símbolo que pode ser usado
como 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 incluído 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 entrará 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
Para 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. para () 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
s.toString () // => "Símbolo (compartilhado)"
Symbol.key para (t) // => "compartilhado"
3.7 O objeto global
As seções anteriores explicaram os tipos primitivos de JavaScript e
valores.Tipos de objetos - objetos, matrizes e funções - são cobertos em
capítulos próprios mais tarde neste livro.Mas há um muito importante
Valor do objeto que devemos cobrir agora.O objeto global é regular
Objeto 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 nan
Funções globais como isnan (), parseint () (§3.9.2) e
Eval () (§4.12)
Funções de construtor como date (), regexp (), string (),
Objeto () e Array () (§3.9.2)
Objetos globais como Math e Json (§6.8)
As propriedades iniciais do objeto global não são palavras reservadas, mas
Eles 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
Código JavaScript contido na janela do navegador que ele representa.Esse
O 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 o
Propriedades globais centrais, mas também define alguns outros globais que
são específicos para navegadores da Web e JavaScript do lado do cliente.Trabalhador da web
threads (§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
o objeto global em qualquer contexto.No início de 2020, esse recurso foi
implementado por todos os navegadores modernos e por nó.
3.8 valores primitivos imutáveis ??e
Referências de objetos mutáveis
Existe uma diferença fundamental no JavaScript entre primitivo
valores (indefinidos, nulos, booleanos, números e cordas) e
objetos (incluindo matrizes e funções).Primitivos são imutáveis:
Não há como mudar (ou "mutatar") 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 capaz
Para alterar o caractere em qualquer índice especificado.De fato, JavaScript não
Permitir isso, e todos os métodos de string que parecem retornar uma string modificada
estão, 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 S
s // => "Hello": a string original não tem
mudado
Os primitivos também são comparados pelo valor: dois valores são os mesmos apenas se
Eles têm o mesmo valor.Isso parece circular para números, booleanos,
nulo e indefinido: não há outra maneira de eles poder
comparado.Novamente, no entanto, não é tão óbvio para as cordas.Se dois
Os 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 índice
é o mesmo.
Objetos são diferentes dos primitivos.Primeiro, eles são mutáveis ??- seu
Os valores podem mudar:
Seja o = {x: 1};// Comece com um objeto
O.x = 2;// sofre -o alterando o valor de um
propriedade
O.Y = 3;// sofre novamente adicionando um novo
propriedade
Seja a = [1,2,3];// matrizes também são mutáveis
a [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 distintos
matrizes não são iguais, mesmo que tenham os mesmos elementos no mesmo
ordem:

```
Seja o = {x: 1}, p = {x: 1};// dois objetos com o mesmo
propriedades
o === p // => false: objetos distintos
nunca são iguais
Seja a = [], b = [];// duas matrizes distintas e vazias
a === b // => false: matrizes distintas são
nunca igual
```

Os objetos às vezes são chamados de tipos de referência para distingui -los de
Tipos primitivos de JavaScript.Usando essa terminologia, os valores dos objetos são
referências, e dizemos que os objetos são comparados por referência: dois
Os valores dos objetos são os mesmos se e somente se eles se referirem ao mesmo
objeto 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 visível
variável 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 um
A variável simplesmente atribui a referência: não cria uma nova cópia de
o objeto.Se você quiser fazer uma nova cópia de um objeto ou matriz, você
deve copiar explicitamente as propriedades do objeto ou os elementos do
variedade.Este exemplo demonstra o uso de um loop for (§5.4.3):

```
deixe A = ["A", "B", "C"];// Uma matriz que queremos
cópia
Seja b = [];// Uma matriz distinta nós vamos
```

copiar em
para (vamos i = 0; i <A.Length; i ++) {// para cada índice de um []
b [i] = a [i];// copie um elemento de um
em b
}
Seja c = Array.From (B);// em ES6, copiar matrizes
com Array.From ()
Da mesma forma, se queremos comparar dois objetos ou matrizes distintos, nós
deve comparar suas propriedades ou elementos.Este código define uma função
Para comparar duas matrizes:
função igualArrays (a, b) {
if (a === b) retorna true;// idêntico
Matrizes são iguais
if (a.Length! == B.Length) retornar FALSO;// Diferente-
Matrizes de tamanho não são iguais
para (vamos i = 0; i <A.Length; i ++) {// loop
todos os elementos
if (a [i]! == b [i]) retorna false;// se houver
diferem, matrizes não são iguais
}
retornar true;// De outra forma
Eles são iguais
}
3.9 Conversões de tipo
O JavaScript é muito flexível sobre os tipos de valores necessários.Nós temos
Vi isso para booleanos: quando JavaScript espera um valor booleano, você
pode fornecer um valor de qualquer tipo, e JavaScript o converterá como
necessário.Alguns valores (valores "verdadeiros") se convertem para verdadeiros e outros
(Valores ?falsamente?) convertem para false.O mesmo vale para outros tipos: se
JavaScript quer uma string, ele converterá qualquer valor que você der em um
corda.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 convertidos
para uma string
"7" * "4" // => 28: Ambas as strings se convertem em números
Seja n = 1 - "x";// n == nan;String "X" não pode se converter 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
Encontre surpreendente.Células vazias indicam que nenhuma conversão é necessária
e nenhum é realizado.
Tabela 3-2.Conversões do tipo JavaScript
Valor
para string
para numerar
para booleano
indefinido
"indefinido"
Nan
falso
nulo
"nulo"
0
falso
verdadeiro
"verdadeiro"
1
falso
"falso"
0
"" (string vazia)
0
falso
"1.2" (não vazio, numérico)
1.2
verdadeiro
"One" (não vazio, não numérico)
Nan
verdadeiro
0
"0"
falso

-0
"0"
falso
1 (finito, diferente de zero)
"1"
verdadeiro
Infinidade
"Infinidade"
verdadeiro
-Infinidade
"-Infinidade"
verdadeiro
Nan
"Nan"
falso
{} (qualquer objeto)
Veja §3.9.3
Veja §3.9.3
verdadeiro
[] (matriz vazia)
""
0
verdadeiro
[9] (um elemento numérico)
"9"
9
verdadeiro
['a'] (qualquer outra matriz)
Use o método junção ()
Nan
verdadeiro
function () {} (qualquer função)
Veja §3.9.3
Nan
verdadeiro

As 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 analisadas
À medida que os números se convertem para esses números.Os espaços de liderança e trilha são
permitido, mas quaisquer caracteres de líder ou não -espaço que não sejam
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, e
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 operando
ser 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
Esses 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
Funções booleanas (), number () e string ():
Número ("3") // => 3
String (false) // => "false": ou use false.toString ()
Booleano ([]) // => true
Qualquer valor que não seja nulo ou indefinido tem um toque ()
método, e o resultado desse método geralmente é o mesmo
retornado pela função String ().
Como um aparte, observe que os booleanos (), number () e string ()
As funções também podem ser invocadas - com novo - como construtor.Se você usa
Eles 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 um
sobras 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 UNUY + converte seu operando em um número.E o
UNARY!O 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 computador
programas e JavaScript possui funções e métodos especializados que
Forneça controle mais preciso sobre o número a cordas e a string-to-
conversões numéricas.
O método tostring () definido pela classe numérica aceita um
Argumento opcional que especifica um radix ou base, para a conversão.Se
Você não especifica o argumento, a conversão é feita na base 10.
No entanto, você também pode converter números em outras bases (entre 2 e
36).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 querer
converter números em strings de maneiras que lhe dão controle sobre o
número de locais decimais ou o número de dígitos significativos no
saída, ou você pode querer controlar se a notação exponencial é
usado.A classe numérica define três métodos para esses tipos de
conversões de número a cordas.tofixed () converte um número em um
string com um número especificado de dígitos após o ponto decimal.Nunca
usa notação exponencial.toExponencial () converte um número para
uma string usando notação exponencial, com um dígito antes do decimal
ponto e um número especificado de dígitos após o ponto decimal (que
significa 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
parte inteira do número.Observe que todos os três métodos ao redor do
dígitos ou almofada à direita com zeros, conforme apropriado.Considere o seguinte
Exemplos:

```
Seja n = 123456.789;
n.tofixado (0) // => "123457"
n.tofixado (2) // => "123456.79"
n.tofixado (5) // => "123456.78900"
n.toexponencial (1) // => "1.2e+5"
n.toexponencial (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, o
Classe Intl.NumberFormat define uma mais geral, internacionalizada
Método de formatação por número.Veja §11.7.1 para obter detalhes.
Se você passar uma string para a função de conversão número (), ela tenta
Para analisar essa corda 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
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 () Skip
liderando 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") // => 0,1
parseint ("0,1") // => 0
parseint (". 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 primitivas

As 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 JavaScript
conversões é que alguns tipos de objetos têm mais de um primitivo
representação.Objetos de data, por exemplo, podem ser representados como strings
ou como registro de data e hora numéricos.A especificação JavaScript define três
Algoritmos fundamentais para converter objetos em valores primitivos:
preferência string
Este algoritmo retorna um valor primitivo, preferindo um valor de string,
Se uma conversão para string for possível.
número preferido
Este algoritmo retorna um valor primitivo, preferindo um número, se
Essa conversão é possível.
sem preferência
Este algoritmo não expressa preferência sobre que tipo de
o valor primitivo é desejado e as classes podem definir seus próprios
conversões.Dos tipos de javascript embutidos, todos exceto a data
Implemente este algoritmo como número preferido.A classe de data
implementa esse algoritmo como string de preferência.
A implementação desses algoritmos de conversão de objeto para princípios
é explicado no final desta seção.Primeiro, no entanto, explicamos como
Os algoritmos são usados ??no JavaScript.
Conversões objeto para boolean
As 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 novo
Booleano (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 e
Conversões de objeto para número descritas anteriormente.
O operador + em JavaScript executa adição numérica e string
concatenação.Se qualquer um de seus operandos for um objeto, JavaScript converte
eles para valores primitivos usando o algoritmo de não preferência.Uma vez que tem
Dois valores primitivos, ele verifica seus tipos.Se um argumento for um
String, 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 um
maneira frouxa que permite conversões de tipo.Se um operando é um objeto e
o outro é um valor primitivo, esses operadores convertem o objeto para
primitivo usando o algoritmo sem preferência e compare os dois
valores primitivos.
Finalmente, os operadores relacionais <, <=,> e> = compare a ordem de
seus operandos e podem ser usados ??para comparar números e strings.Se
Qualquer um operando é um objeto, é convertido em um valor primitivo usando o
Algoritmo de número preferido.Observe, no entanto, que diferente do objeto para-
Conversão de número, os valores primitivos retornados pelo número preferido
A conversão não é então convertida em números.
Observe que a representação numérica dos objetos de data é significativamente
Comparável com <e>, mas a representação da string não é.Para data
objetos, o algoritmo de não preferência se converte em uma string, então o fato
Esse javascript usa o algoritmo de número preferido para esses operadores
significa 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 () do
A 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
existe.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 de 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
tempo)
D.Valueof () // => 1262332800000
Algoritmos de conversão de objeto a princípios
Com os métodos ToString () e ValueOf ()
agora explique aproximadamente como os três objeto a princípio
Os algoritmos funcionam (os detalhes completos são adiados até §14.4.7):
O algoritmo preferido primeiro tenta o toque ()
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 preferida
algoritmo, exceto que tenta valueof () primeiro e
ToString () Segundo.
O algoritmo sem preferência depende da classe do
objeto sendo convertido.Se o objeto for um objeto de data, então
O JavaScript usa o algoritmo preferido.Para qualquer outro
Objeto, JavaScript usa o algoritmo de número preferido.
As regras descritas aqui são verdadeiras para todos os tipos de javascript embutidos e
são as regras padrão para todas as classes que você se define.§14.4.7
Explica como você pode definir sua própria conversão de objeto para primitiva
Algoritmos para as classes que você define.
Antes de deixarmos este tópico, vale a pena notar que os detalhes do
Conversão de número preferido Explique por que as matrizes vazias se convertem para o
Número 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 resultante
valor primitivo para um número.O algoritmo do número preferido tenta
Valueof () primeiro e depois recorre ao ToString ().Mas a matriz
Classe herda o método ValueOf () padrão, que não retorna um
valor primitivo.Então, quando tentamos converter uma matriz em um número, nós
acaba invocando o método tostring () da matriz.Matrizes vazias
converter para a string vazia.E a corda vazia se converte para o
número 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 no
programas que escrevemos.Quando fazemos isso, normalmente dizemos que somos
atribuindo um valor a uma variável.O termo "variável" implica que novo
Os 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 -lo.No ES6 e mais tarde, isso é feito com o Let and Const
Palavras -chave, que explicamos a seguir.Antes do ES6, variáveis ??eram
declarado com var, que é mais idiossincrático e é explicado mais tarde
em nesta 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 Let
declaração, a variável é declarada, mas seu valor é indefinido até
Seu 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.Um
A abordagem é usar const apenas para valores que são fundamentalmente imutáveis, como
as 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 de
As chamadas variáveis ??em nosso programa nunca mudam como nosso programa
corre.Nesta abordagem, declaramos tudo com const, e então se encontrarmos isso
Na verdade, queremos permitir que o valor varie, mudamos a declaração para deixar.
Isso pode ajudar a prevenir bugs descartando mudanças acidentais 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 primeiro
abordagem em meu próprio código.
No capítulo 5, aprenderemos sobre o para, para/in e para/de loop
Declarações em JavaScript.Cada um desses loops inclui uma variável de loop
Isso recebe um novo valor atribuído a ele em cada iteração do loop.
O JavaScript nos permite declarar a variável de loop como parte do loop
sintaxe 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 do
O loop não reatribui um novo valor.Nesse caso, a declaração const
está apenas dizendo que o valor é constante durante a duração de um loop
iteração:
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 eles são definidos apenas dentro
O 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 eles
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
e nos módulos JavaScript do lado do cliente (consulte o capítulo 10), o escopo de um
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 html
documento no qual é definido.Isto é: se um <cript> declara um
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).

Declarações repetidas
É um erro de sintaxe usar o mesmo nome com mais de um let ou
Declaração const no mesmo escopo.É legal (embora seja melhor uma prática
evitado) para declarar uma nova variável com o mesmo nome em um aninhado
escopo:
const x = 1;// declarar x como uma constante global
if (x === 1) {
Seja x = 2;// dentro de um bloco x pode se referir a um
valor diferente
console.log (x);// imprime 2
}
console.log (x);// Imprima 1: estamos de volta ao global
escopo agora
Seja x = 3;// Erro!Erro de sintaxe tentando voltar
declarar x
Declarações e tipos
Se você está acostumado a idiomas estaticamente digitados como C ou Java, você pode
acho que o objetivo principal das declarações variáveis ??é especificar o
tipo de valores que podem ser atribuídos a uma variável.Mas, como você tem
visto, não há tipo associado à variável de JavaScript
declarações.Uma variável JavaScript pode conter um valor de qualquer tipo.Para
por exemplo, é perfeitamente legal (mas geralmente mau estilo de programação) em
Javascript para atribuir um número a uma variável e depois atribuir um
string para essa variável:
deixe i = 10;
i = "ten";
3.10.2 declarações variáveis ??com VAR
2

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.O
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 escoposos no corpo da função que contém não
importa o quão 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 um
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 o
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 loop
corpo, cada um desses loops é (inofensivamente) re-declaração e re-
inicializando a mesma variável.
Uma das características mais incomuns das declarações VAR é
conhecido como iça.Quando uma variável é declarada com var, o
a declaração é levantada (ou "içada") para o topo do
Função de anexo.A inicialização da variável permanece
onde você escreveu, mas a definição da variável se move para
o topo da função.Portanto, as variáveis ??declaradas com var podem ser
Usado, sem erro, em qualquer lugar da função de anexo.Se o
O código de inicialização ainda não foi executado, o valor do
A variável pode ser indefinida, mas você não receberá um erro se
Você usa a variável antes de ser inicializada.(Este pode ser um
fonte de bugs e é um dos malfeilhos importantes que
Deixe corrigir: se você declarar uma variável com LET HABLE TENTE
Para usá -lo antes da execução da declaração let, você receberá um verdadeiro
erro em vez de apenas ver um valor indefinido.)
Usando variáveis ??não declaradas
No modo rigoroso (§5.6.3), se você tentar usar uma variável não declarada, você receberá um
Erro de referência ao executar seu código.Fora do modo rigoroso, no entanto, se você
atribua 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 agora
profundamente aninhado nas funções e bloqueia seu código, o que é quase certamente
Não é o que você quer, é propenso a insetos e é uma das melhores razões para usar rigorosamente
modo!
As variáveis ??globais criadas dessa maneira acidental são como variáveis ??globais declaradas
com VAR: eles definem propriedades do objeto global.Mas, diferente das propriedades
Definidos por declarações adequadas do VAR, essas propriedades podem ser excluídas com o
Excluir operador (§4.13.4).

3.10.3 Atribuição de destruição
ES6 implementa um tipo de declaração e atribuição compostas
Sintaxe conhecida como atribuição de destruição.Em uma destruição
atribuição, 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 e
Sintaxe literal do objeto.Quando ocorre uma tarefa de destruição, um ou
Mais valores são extraídos ("destruturados") do valor à direita
e armazenado nas variáveis ??nomeadas à esquerda.Destruição
A atribuição talvez seja mais comumente usada para inicializar variáveis ??como
parte 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 pode
Também 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, teta] 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/4
Seja [x, y] = tocartesiano (r, teta);// [x, y] == [1,0, 1,0]
Vimos que variáveis ??e constantes podem ser declaradas como parte de
Vários de Javascript para loops.É possível usar variável
Destrutura nesse contexto também.Aqui está um código que atravessa o
Nome/valor pares de todas as propriedades de um objeto e usa a destruição
atribuição para converter esses pares de matrizes de dois elementos em
variáveis ??individuais:
Seja o = {x: 1, y: 2};// o objeto que vamos fazer
for (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 faz
não precisa corresponder ao número de elementos de matriz à direita.Extra
Variáveis ??à esquerda são definidas como indefinidas e valores extras no
à direita são ignorados.A lista de variáveis ??à esquerda pode incluir extra
vírgulas 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 == 4
Se você deseja coletar todos os valores não utilizados ou restantes em um único
variável ao destruir uma matriz, use três pontos (...) antes do
Último 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 ==
2.5
Uma característica poderosa da destruição de matrizes é que ela realmente não
requer uma matriz!Você pode usar qualquer objeto iterável (capítulo 12) no
Lado RightHands 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 vírgula
nomes variáveis ??dentro de aparelhos encaracolados:
Seja transparente = {r: 0.0, g: 0,0, b: 0,0, a: 1,0};// A RGBA
cor
Seja {r, g, b} = transparente;// r == 0,0;g == 0,0;b == 0.0
O 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.tan
const {sin, cos, tan} = matemática;
Observe no código aqui que o objeto de matemática tem muitas propriedades outras
do que os três que são destruídos em variáveis ??individuais.Aqueles que
não são nomeados são simplesmente ignorados.Se o lado esquerdo deste
A tarefa incluiu uma variável cujo nome não era uma propriedade de
Matemática, essa variável seria simplesmente designada indefinida.
Em cada um desses exemplos de destruição de objetos, escolhemos
nomes variáveis ??que correspondem aos nomes de propriedades do objeto que somos
destruição.Isso mantém a sintaxe simples e fácil de entender, mas
não é necessário.Cada um dos identificadores no lado esquerdo de um
A atribuição de destruição de objetos também pode ser um par de separação de cólon de
identificadores, onde o primeiro é o nome da propriedade cujo valor é
ser 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 ser
Útil 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, em
ambos os literais de objeto e à esquerda de um objeto destrutivo
atribuição.
A tarefa de destruição se torna ainda mais complicada quando é
usado com objetos aninhados, ou matrizes de objetos, ou objetos de matrizes, mas
é legal:

deixe pontos = [{x: 1, y: 2}, {x: 3, y: 4}];// uma variedade de
dois objetos de pontos
Seja [{x: x1, y: y1}, {x: x2, y: y2}] = pontos;//
Destruturado em 4 variáveis.
(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => true
Ou, 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 matriz
Seja {p1: [x1, y1], p2: [x2, y2]} = pontos;//
Destruturado em 4 VARs
(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => true
Sintaxe de destruição complexa como essa pode ser difícil de escrever e difícil para
Leia, e você pode estar melhor apenas escrevendo suas tarefas
explicitamente com código tradicional como let x1 = pontos.p1 [0];.
Compreendendo a destruição complexa
Se 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 um
A atribuição de destruição parece uma matriz literal ou um objeto literal (§6.2.1 e §6.10).Depois do
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 por
Tentando usar o lado esquerdo do lado do campo de outra expressão de atribuição:
// Comece com uma estrutura de dados e uma destruição complexa
deixe 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 destruição
Seja pontos2 = [{x: x1, y: y1}, {x: x2, y: y2}];// pontos2 == pontos

3.11 Resumo

Alguns pontos -chave a serem lembrados sobre este capítulo:

Como escrever e manipular números e seqüências de texto em
JavaScript.

Como trabalhar com outros tipos primitivos de JavaScript:
Booleanos, símbolos, nulos e indefinidos.

As diferenças entre tipos primitivos imutáveis ??e
Tipos de referência mutáveis.

Como o JavaScript converte valores implicitamente de um tipo para
Outro e como você pode fazê -lo explicitamente em seus programas.

Como declarar e inicializar constantes e variáveis
(inclusive com atribuição de destruição) e o lexical
Escopo das variáveis ??e constantes que você declara.

1
Este é o formato para números de tipo duplo em java, c ++ e mais moderno
linguagens de programação.

2
Existem extensões de JavaScript, como TypeScript e Flow (§17.8), que permitem que tipos
ser 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 avaliado para produzir um valor.Uma constante Incorporado 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 do 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
linguagem que usa sintaxe no estilo C, você descobrirá que a sintaxe da maioria dos
As 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
Stail sozinho - eles 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 um
A 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 o
Valores das expressões separadas por vírgula:
[] // Uma matriz vazia: sem expressões dentro de colchetes
significa não elementos
[1+2,3+4] // Uma matriz de 2 elementos.O primeiro elemento é 3, segundo
é 7
As expressões de elementos em um inicializador de matriz podem ser mesmas
Inicializadores, o que significa que essas expressões podem criar aninhadas
Matrizes:
Let Matrix = [[1,2,3], [4,5,6], [7,8,9]];
As expressões de elemento em um inicializador de matriz são avaliadas a cada vez
O inicializador da matriz é avaliado.Isso significa que o valor de uma matriz
A expressão inicializadora pode ser diferente cada vez que é avaliada.
Elementos indefinidos podem ser incluídos em uma matriz literal por simplesmente
omitindo um valor entre vírgulas.Por exemplo, a seguinte matriz
Contém cinco elementos, incluindo três elementos indefinidos:
Seja SparsarArray = [1 ,,,, 5];
Uma única vírgula à direita é permitida após a última expressão em uma matriz
Inicializador e não cria um elemento indefinido.No entanto, qualquer um
Expressão de acesso à matriz para um índice após o da última expressão
será necessariamente avaliado como indefinido.
Expressões de inicializador de objetos são como expressões de inicializador de matriz, mas
Os suportes quadrados são substituídos por colchetes encaracolados, e cada
A subexpressão é prefixada com um nome de propriedade e um cólon:

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 o mesmo
propriedades como p
No ES6, os literais de objetos têm uma sintaxe muito mais rica em recursos (você pode
Encontre 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
O valor dessa expressão é a função recém -definida.Em certo sentido, um
A 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 um
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) em
aparelho 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, função
As expressões podem usar uma nova sintaxe compacta "Função de seta".Completo
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 [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 propriedade
o acesso 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 = [O, 4, [5, 6]];// um exemplo de matriz que contém
o objeto
O.x // => 1: Propriedade X de expressão
o
O.Y.Z // => 3: Propriedade z da expressão
O.Y
o ["x"] // => 1: propriedade x do objeto O
a [1] // => 4: elemento no índice 1 de
expressão a
a [2] ["1"] // => 6: elemento no índice 1 de
expressão a [2]
```

a [0] .x // => 1: Propriedade x de expressão
A [0]
Com qualquer tipo de expressão de acesso à propriedade, a expressão antes
o .ou [é primeiro avaliado.Se o valor for nulo ou indefinido, o
A expressão lança um TypeError, já que esses são os dois JavaScript
valores que não podem ter propriedades.Se a expressão do objeto for seguida
por um ponto e um identificador, o valor da propriedade nomeada por isso
O identificador é procurado e se torna o valor geral da expressão.
Se a expressão do objeto for seguida por outra expressão no quadrado
Suportes, essa segunda expressão é avaliada e convertida em uma corda.
O valor geral da expressão é então o valor da propriedade
nomeado por essa string.Em ambos os casos, se a propriedade nomeada não
Existir, 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
o acesso tem um nome que é um identificador legal e quando você sabe o
Nome quando você escreve o programa.Se o nome da propriedade incluir
espaços ou caracteres de pontuação, ou quando é um número (para matrizes),
Você deve usar a notação de suporte quadrado.Suportes quadrados também são usados
Quando o nome da propriedade não é estático, mas é o resultado de um
Computação (consulte §6.3.1 para um exemplo).
Objetos e suas propriedades são abordados em detalhes no capítulo 6 e
Matrizes e seus elementos são abordados no capítulo 7.
4.4.1 Acesso à propriedade condicional
O ES2020 adiciona dois novos tipos de expressões de acesso à propriedade:

expressão?.identificador
expressão?. [Expressão]
Em JavaScript, os valores nulos e indefinidos são os únicos dois
valores que não têm propriedades.Em um acesso regular à propriedade
expressão usando.ou [], você obtém um TypeError se a expressão no
A 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 o
A expressão avalia para indefinida sem qualquer tentativa de acessar o
Propriedade b.Se A é algum outro valor, então a? .B avalia para qualquer coisa
A.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 ?opcional
encadeamento ?porque também funciona para acesso de propriedade mais longo? encadeado ?
expressões como esta:
Seja a = {b: null};
a.b? .c.d // => indefinido
A é um objeto, então A.B é uma expressão de acesso à propriedade válida.Mas o
O 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 valor
indefinido.Mas - e esta é uma parte muito importante de ?opcional
encadeamento ? - a.b? .c.d (sem parênteses) simplesmente avalia

indefinido e não apresenta um erro.Isso é porque propriedade
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
ocorrer 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?. [] curto
circuitos
A [index ++] //! TypeError: Não é possível indexar indefinidos.

Acesso à propriedade condicional 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
seguido de um parêntese aberta, 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 o
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
avaliado primeiro, e depois as expressões de argumento são avaliadas para
produzir uma lista de valores de argumento.Se o valor da função
A expressão 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 um
expressão antes dos parênteses abertos.Se essa expressão for uma propriedade
Expressão de acesso, então a invocação é conhecida como um método
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) opera 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?. () Em vez de ().
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 o
expressão à esquerda do?.avalia para nulo ou indefinido,
então toda a expressão de invocação avalia para indefinido e não
A 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
elementos.Antes do ES2020, se você quisesse escrever um método como
classin () que leva um argumento de função opcional, você normalmente
Use 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 opcional
if (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 exceção
é jogado
}
f?. (x ++) // => indefinido: f é nulo, mas sem exceção
jogado
x // => 1: o incremento é ignorado por causa de curta
circuito
Expressões de invocação condicional com?. () Funcionam tão bem para
Métodos como fazem para funções.Mas porque a invocação de método também
envolve acesso à propriedade, vale a pena levar um momento para ter certeza de
Entenda as diferenças entre as seguintes expressões:
o.m () // acesso regular à propriedade, invocação regular
o?. .m () // Acesso à propriedade condicional, invocação regular
O.M?. () // Acesso regular à propriedade, Invocação condicional
Na primeira expressão, O deve ser um objeto com uma propriedade M e o
O valor dessa propriedade deve ser uma função.Na segunda expressão, se o
é nulo ou indefinido, então a expressão avalia para
indefinido.Mas se O tiver algum outro valor, deve ter um
Propriedade m cujo valor é uma função.E na terceira expressão, O
não deve ser nulo ou indefinido.Se não tiver uma propriedade m, ou
Se o valor dessa propriedade for nulo, toda a expressão
Avalia como indefinido.
Invocação condicional com?. () É uma das características mais recentes de
JavaScript.A partir dos primeiros meses de 2020, esta nova sintaxe é suportada
nas 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 um
função (chamada de construtor) para inicializar as propriedades desse objeto.
Expressões de criação de objetos são como expressões de invocação, exceto que
Eles são prefixados 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
excluir e instanceof.Os operadores de palavras -chave são operadores regulares,
Assim como os expressos com pontuação;eles simplesmente têm menos
Sintaxe sucinta.
A Tabela 4-1 é organizada pela precedência do operador.Os operadores listados

+=, -=, & =, ^=,
| =,
<< =, >> =, >>> =
, Assim,
Descarte o 1º operando, retornar
2º
L
2
qualquer, qualquer ? qualquer
4.7.1 Número de operandos
Os operadores podem ser categorizados com base no número de operandos que eles
Espere (sua arity).A maioria dos operadores de javascript, como a * multiplicação
operador, são operadores binários que combinam duas expressões em um
expressão única e mais complexa.Ou seja, eles esperam dois operando.
JavaScript também suporta vários operadores unários, que convertem um
expressão única em uma expressão única e mais complexa.O -
O operador na expressão ?x é um operador unário que executa o
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
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 seus
operando conforme necessário.O operador de multiplicação * espera numérico

operando, mas a expressão "3" * "5" é legal porque JavaScript
pode converter os operando 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
Espere que os operando booleanos funcionem com um operando de qualquer tipo.
Alguns operadores se comportam de maneira diferente, dependendo do tipo de
operando usados ??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
Encomende, dependendo do tipo de operandos.As descrições de
operadores individuais explicam suas dependências de tipo e especificam o que
Digite conversões que eles executam.
Observe que os operadores de atribuição e alguns dos outros operadores
Listado na Tabela 4-1 Espere um operando do tipo LVAL.lvalue 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 lvalues.

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
não seja 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 um 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 Os 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.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 aqueles 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 estão concluídos.

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 maior
Precedência do que qualquer um dos operadores listados na Tabela 4-1.Considere isso
expressão:
// meu é um objeto com uma propriedade chamada funções cujo
O valor é um
// Matriz de funções.Invocamos o número da função X, passando
argumento
// y, e então solicitamos o tipo de valor retornado.
tipo de my.fuções [x] (y)
Embora o tipo de seja um dos operadores de maior prioridade, o
Tipoof Operação é realizada no resultado do acesso à propriedade,
índice de matriz e invocação de funções, todos com maior prioridade
do que operadores.
Na prática, se você não tiver certeza sobre a precedência de seu
operadores, a coisa mais simples a fazer é usar parênteses para fazer o
Ordem de avaliação explícita.As regras que são importantes a saber são
estes: multiplicação e divisão são realizadas antes da adição e
subtração, e a atribuição tem muito baixa precedência e é quase
Sempre executado por último.
Quando novos operadores são adicionados ao JavaScript, eles nem sempre se encaixam
naturalmente nesse esquema de precedência.O ??Operador (§4.13.2) é
Mostrado na tabela como menor precedência que ||e &&, mas, de fato, seu
Precedência em relação a esses operadores não está definida e ES2020
Requer 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 e
você deve usar parênteses ao combinar a negação com
Expeneção.

4.7.5 Associatividade do operador

Na Tabela 4-1, a coluna rotulou como especifica 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 precedência
são realizados.Associatividade da esquerda para a direita significa que as operações são
realizado 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 = ~ -y;
w = x = y = z;
P = a? B: C? D: E? F: G;
```

são equivalentes a:

```
y = (a ** (b ** c));
x = ~ (-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
avaliado 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 ou
propriedade especificada pela expressão w.Adicionando parênteses ao
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 avaliado 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 própria
Porque também pode executar concatenação de string e tem alguns incomuns
Tipo Regras de conversão.Os operadores unários e os operadores bitwise
também 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, como
desde 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 depois
Calcule o poder, produto, quociente, restante ou diferença.Não-
operando numéricos que não podem se converter em números convertidos para a nan
valor.Se qualquer um opera é (ou converter para) nan, o resultado do
a operação é (quase sempre) nan.
O operador ** tem maior precedência que *, /e % (que por sua vez
tem 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 de
a relativa precedência de unário menos e exponenciação, que
A expressão pode significar (-3) ** 2 ou -(3 ** 2).Diferentes idiomas
lidar com isso de maneira diferente e, em vez de escolher lados, JavaScript simplesmente
torna um erro de sintaxe omitir parênteses neste caso, forçando você a
Escreva uma expressão inequívoca.** é a mais nova aritmética de JavaScript

Operador: foi adicionado ao idioma com ES2016.O
A função Math.pow () está disponível desde as primeiras versões
de JavaScript, no entanto, e 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
números de ponto, você pode esperar obter um resultado inteiro quando você
Divida 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 a NAN: nenhum desses casos
levanta um erro.
O operador % calcula o primeiro módulo de operando o segundo operando.
Em outras palavras, ele retorna o restante após a divisão de número inteiro de
o primeiro operando pelo segundo operando.O sinal do resultado é o
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 a string
operando:

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 realizada
Depende da conversão realizada.As regras de conversão para + dê
prioridade à concatenação da string: se qualquer um dos operandos for uma string ou
um objeto que se converte em uma string, o outro operando é convertido em um
String e concatenação são realizadas.A adição é realizada apenas se
Nenhuma 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 um
primitivo 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 maioria
Objetos não têm um método de valor útil (), então são
convertido via ToString () também.
Após a conversão de objeto para primitivo, se qualquer um operando for um
string, a outra é convertida em uma corda e a concatenação é
realizado.
Caso contrário, ambos os operando são convertidos em números (ou para
Nan) e adição é realizada.
Aqui estão alguns exemplos:
1 + 2 // => 3: adição
"1" + "2" // => "12": concatenação
"1" + 2 // => "12": concatenação após número a-

corda
1 + {} // => "1 [objeto objeto]": concatenação após
objeto para cordas
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
dependem 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
associatividade, então os dois números são adicionados primeiro e sua soma é
concatenado 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
usado 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 em Nan) e retornos que convertiam valor.Quando usado com um operando Isso 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 alterar o sinal do resultado.
Incremento (++)
O operador ++ incrementos (ou seja, adiciona 1 a) seu único operando, que deve ser um lvalue (uma variável, um elemento de uma matriz ou um 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 de sua posição parente para o operando.Quando usado antes do operando, onde é conhecido como O operador de pré-incremento, ele aumenta o operando e avalia ao valor incrementado desse operando.Quando usado após o operando, onde é conhecido como operador pós-incremento, ele incrementa seu operando, mas avalia o valor não descmentado de aquele operando.Considere a diferença entre essas duas linhas de código:
Seja i = 1, j = ++ i;// eu e j são 2
Seja n = 1, m = n ++;// n é 2, m é 1

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 JavaScript
Inserção, você não pode inserir uma quebra de linha entre o pós-incremento
operador e o operando que o precede.Se você fizer isso, JavaScript
tratará o operando como uma declaração completa por si só e inserirá um
Semicolon 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 de
o operando a um número, subtrai 1 e atribui o decrementado
Valor de volta ao operando.Como o operador ++, o valor de retorno de
- depende de sua posição em relação ao operando.Quando usado
Antes do operando, ele diminui e retorna o decrementado
valor.Quando usado após o operando, diminui o operando, mas
Retorna o valor indecremido.Quando usado após seu operando, não
A quebra de linha é permitida entre o operando e o operador.
4.8.3 Operadores bitwise
Os operadores bitwee
Representação binária de números.Embora eles não realizem
Operações aritméticas tradicionais, elas são classificadas como aritmética
operadores aqui porque operam em operandos numéricos e retornam um
valor numérico.Quatro desses operadores realizam álgebra booleana no
Bits individuais dos operandos, se comportando como se cada um pouco em cada operando

eram um valor booleano (1 = true, 0 = false).Os outros três bit
Os 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
Complemento Representação de números inteiros negativos, você provavelmente pode pular
esta seção.
Os operadores bitwee esperam operando inteiros e se comportar
Os valores foram representados como números inteiros de 32 bits em vez de flutuação de 64 bits
valores pontuais.Esses operadores convertem seus operando em números, se
necessário e depois coagir os valores numéricos a números inteiros de 32 bits por
soltando qualquer parte fracionária e quaisquer bits além do 32º.A mudança
Os operadores exigem um operando do lado direito entre 0 e 31. Depois
convertendo este operando em um número inteiro de 32 bits não assinado, eles abandonam qualquer
Bits 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 um booleano e operação em cada bit de
seus argumentos inteiros.Um pouco está definido no resultado apenas se o
O bit correspondente é definido nos dois operandos.Por exemplo, 0x1234 &
0x00FF Avalia para 0x0034.
Bit nessa ou (|)
O |O operador executa um booleano 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 ^ 0xf0f0
Avalia 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
0xffffff0, 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
bit), o segundo bit de a se torna o terceiro, etc. Um zero é usado para
O novo primeiro bit e o valor do 32º bit são perdidos.Mudando a
O valor deixado 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 o número de lugares especificados no segundo operando (um número inteiro entre 0 e 31).Bits que são deslocados para a direita são perdidos.O 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 e -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 Os operadores testam um relacionamento (como "iguais", "menos que" ou ?Propriedade de?) entre dois valores e retorno verdadeiro ou falso Dependendo se esse relacionamento existe.Expressões relacionais sempre avalie 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 o
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 mesmice.Ambos os operadores aceitam
operando de qualquer tipo, e ambos retornam verdadeiro se seus operando forem o
O mesmo e falso se forem diferentes.O operador === é conhecido como
o estrito operador de igualdade (ou às vezes o operador de identidade), e ele
Verifica se seus dois operandos são "idênticos" usando uma definição estrita
de mesmice.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! = E! == Operadores testam exatamente o oposto do == e
=== operadores.O! = O operador de desigualdade retorna false se dois
Os valores são iguais um ao outro de acordo com == e retorna verdadeiro
de outra forma.O! == O operador retorna false se dois valores forem estritamente
igual um ao outro e retorna verdadeiro de outra forma.Como você verá em §4.10,
o !O operador calcula a operação booleana e não.Isso faz isso
fácil de lembrar disso! = e! == significa "não igual a" e "não
estritamente igual a. ?

O =, == e === operadores

JavaScript suporta =, == 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 o mesmo
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.
Igualdade estrita
O Strito Operador de Igualdade === 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 os
Valor booleano Falso, eles são iguais.
Se um ou ambos os valores forem nan, eles não são iguais.(Isso é
surpreendente, mas o valor da nan nunca é igual a nenhum outro
valor, inclusive a si mesmo!Para verificar se um valor x é nan, use
x! == x, ou a função global isnan ().)
Se ambos os valores forem números e têm o mesmo valor, eles são
igual.Se um valor for 0 e o outro é -0, eles também são
igual.
Se ambos os valores forem strings e conter exatamente os mesmos 16 bits
valores (veja a barra lateral em §3.3) nas mesmas posições, eles são
igual.Se as cordas diferirem em comprimento ou conteúdo, elas não são

igual.Duas cordas podem ter o mesmo significado e o mesmo
aparência visual, mas ainda ser codificada usando diferente
Sequências de valores de 16 bits.JavaScript não executa o Unicode
normalização, 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, eles
são iguais.Se eles se referirem a objetos diferentes, eles não são iguais,
Mesmo que ambos os objetos tenham propriedades idênticas.
Igualdade com conversão de tipo
O 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, ele
tenta 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, eles
são 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 e
Digite 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 comparação
Novamente, usando o valor convertido.
Se qualquer um dos valores for verdadeiro, converta -o em 1 e tente o
comparação novamente.Se qualquer um dos valores for falso, converta -o
para 0 e tente a comparação novamente.
Se um valor é um objeto e o outro é um número ou
string, 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
O mesmo, a comparação retorna true.
4.9.2 Operadores de comparação
Os operadores de comparação testam a ordem relativa (numérica ou
alfabético) 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 (<=)
O operador <= 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 (> =)
O operador> = 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
O método tostring () é usado.
Se, após qualquer conversão de objeto para princípio necessária, ambos
operando são strings, as duas cordas são comparadas, usando
ordem alfabética, onde ?ordem alfabética? é definida por
a 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
ou 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 seqüências 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 fizer
não 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.TOUPPERCASE ().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
a 11.
"um" <3 // => false: comparação numérica, "um" convertido
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
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
O operador in espera um operando do lado esquerdo que é uma corda, símbolo ou
valor que pode ser convertido em uma string.Espera 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
chamado "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 cordas
3 em dados // => false: nenhum elemento 3
4.9.4 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
O operador avalia como verdadeiro se o objeto do lado esquerdo for uma instância do
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
Instância de deve ser uma função.Aqui estão exemplos:
Seja d = new Date ();// Crie um novo objeto com a data ()
construtor
d instância de data // => true: d foi criado com date ()
D instância do objeto // => true: Todos os objetos são instâncias de
Objeto
D instância do número // => false: D não é um objeto numérico
Seja a = [1, 2, 3];// Crie uma matriz com matriz literal
sintaxe
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 de
considera as "superclasses" ao decidir se um objeto é um
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 um
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 o
Instância de F, JavaScript avalia F.Prototype e, em seguida,
Procura esse valor na cadeia de protótipos de O.Se encontrar, então O é
Uma 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 são
frequentemente 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 totalmente
entenda -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.No
nível mais simples, quando usado com operandos booleanos, && executa o
Booleano e operação nos dois valores: ele retorna true se e somente
Se 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 operando 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 operando devem ser falsamente, e o
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 causar 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
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
Avalie 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 && 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 o
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 veja o código que explora propositalmente esse comportamento para
Execute o código condicionalmente.Por exemplo, as duas linhas a seguir de
O código 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, decréscimos ou função
invocações) no lado direito de &&.Se esses efeitos colaterais
Ocorre depende do valor do lado esquerdo.
Apesar da maneira um tanto complexa que esse operador realmente funciona, é
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 seu
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 padrão
Valores para parâmetros:

```
// copie as propriedades de O para P e retorne P
cópia da função (o, p) {
P = P ||{};// Se nenhum objeto foi aprovado para P, use um recém
objeto criado.
// O corpo da função vai aqui
}
```

No ES6 e mais tarde, no entanto, esse truque não é mais necessário porque o
O valor do parâmetro padrão pode ser simplesmente escrito na função
Definição em si: função cópia (o, p = {}) {...}.

4.10.3 Lógico não (!)

O !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, se
X é verdade ,! X avalia como falso.Se x é falsamente, então! X é verdadeiro.
Ao contrário do && e ||operadores, o!operador converte seu operando para
um valor booleano (usando as regras descritas no capítulo 3) antes
invertendo o valor convertido.Isso significa isso!sempre retorna verdadeiro
ou falso e que você pode converter qualquer valor x em seu equivalente
Valor booleano aplicando este operador duas vezes: !! x (consulte §3.9.2).
Como um operador unário ,!tem alta precedência e se liga firmemente.Se você
deseja inverter o valor de uma expressão como p && q, você precisa usar
Parênteses :! (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 e
q
! (p || q) === (! p &&! q) // => true: para todos os valores de p e
q
```

4.11 Expressões de atribuição

JavaScript 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 tarefa Expressão é o valor do operando do lado direito.Como efeito colateral, o = O operador atribui o valor à direita à variável ou propriedade no à 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 veja o valor de uma expressão de atribuição usada como parte de um expressão maior.Por exemplo, você pode atribuir e testar um valor no mesma expressão com código como este:

```
(a = b) === 0
```

Se 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 para
ser 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, eles
são avaliados da direita para a esquerda.Assim, você pode escrever código como este para
Atribua um único valor a várias variáveis:
i = j = k = 0;// Inicialize 3 variáveis ??para 0
4.11.1 Atribuição com operação
Além 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 string
operandos, executa concatenação e atribuição.
Os operadores semelhantes incluem -=, *=, & = e assim por diante.A Tabela 4-2 os lista
todos.

Tabela 4-2.Operadores de atribuição

Operador	Exemplo	Equivalente
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
** =	a ** = b	a = a ** b
<< =	a << = b	a = a << b
>> =	a >> = b	a = a >> b
>>> =	a >>> = b	a = a >>> b
& =	a & = b	a = a & b
| =	a | = b	a = a |b
^=	a ^= b	a = 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 apenas 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
foi 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
e 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 = aval;
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 um
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
analise 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
indefinido se a última expressão ou declaração não tivesse valor.Se o
String avaliada lança uma exceção, que a exceção se propaga de
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
os valores das variáveis ??e define novas variáveis ??e funções no
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 local
variável.Se ele chama de avaliar ("x = 1"), altera o valor do local
variável.E se a função chama avaliar ("var y = 3;"),

declara uma nova variável local y.Por outro lado, se avaliado
Usos 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 usá -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 o
A 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 chama
Eval ().Mas o que um intérprete de javascript deve fazer, no entanto, se um
o script define um alias para avaliar () e depois chama essa função por

Outro nome?A especificação JavaScript declara que quando quando
avaliar () 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 pode
Defina novas variáveis ??globais ou funções globais, e pode definir global
variáveis, mas não usarão ou modificarão nenhuma variável local para a chamada
Função 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 indireto
As chamadas podem definir novas variáveis ??apenas com var.Usos de let e const
Dentro de uma string avaliada, crie variáveis ??e constantes que são locais
para a avaliação e não altere o chamado ou ambiente global.)
O código a seguir demonstra:

```
const geval = avaliação;// usando outro nome faz
Uma avaliação global
Seja x = "global", y = "global";// duas variáveis ??globais
função f () {// Esta função faz um
Eval local
Seja x = "local";// define uma variável local
avaliar ("x += 'alterado';");// Conjuntos de avaliação direta Local
variável
retornar x;// retorno alterado local
variável
}
função g () {// Esta função faz um
Global Eval
Seja y = "local";// Uma variável local
```

geval ("y += 'alterado';");// Conjuntos de avaliação indireta
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 tremendamente útil recurso que permite executar seqüências 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 aval.

4.12.3 Eval rigoroso ()

Modo rigoroso (ver §5.6.3) impõe restrições adicionais ao comportamento de a função EVALL () e mesmo no uso do identificador "avaliar".

Quando avaliar () é chamado do código de modo rigoroso ou quando a sequência de O código a ser avaliado em si começa com uma diretiva de "uso rigoroso", então Eval () 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 local escopo.

Além disso, o modo rigoroso torna Eval () ainda mais parecido com o operador por efetivamente transformar "avaliar" em uma palavra reservada.Você não tem permissão para substitua a função Eval () com um novo valor.E você não é

permitido declarar uma variável, função, parâmetro de função ou captura
Bloquear 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 apareça
Assim no código.Porque este operador tem três operandos, o
Primeiro vai antes do?, O segundo vai entre o?e o: e
O terceiro vai depois do :.É usado assim:
x> 0?x: -x // o valor absoluto de x
Os operandos do operador condicional podem ser de qualquer tipo.O primeiro
Operando é avaliado e interpretado como um booleano.Se o valor do
O primeiro operando é verdadeiro, então o segundo operando é avaliado e seu
o valor é retornado.Caso contrário, se o primeiro operando for falsamente, então o terceiro
Operando é avaliado e seu valor é retornado.Apenas um dos segundo
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 típico
uso, que verifica para ter certeza de que uma variável é definida (e tem um
valor 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 se
declaração:
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: se
Seu 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 se
O primeiro operando avalia como nulo ou indefinido.Se a expressão
a 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 o
Primeiro operando definido em vez do primeiro operando de verdade.Embora ||
é nominalmente um lógico ou operador, também é usado idiomaticamente para
Selecione o primeiro operando não-falsy com código como este:
// 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;
O problema com esse uso idiomático é que zero, a corda vazia e
false são todos valores falsamente que podem ser perfeitamente válidos em alguns
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
indefinido) 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
definido
options.n ??10 // => 10: A propriedade é nula
Observe que o tempo limite, título e expressões detalhadas aqui
teria valores diferentes se usássemos ||em vez de ??.

O ??O operador é semelhante ao && e ||operadores, mas não
têm 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 // SyntaxeRor: 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 "coalescante nulo", mas eu
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 de
o operando.A Tabela 4-3 especifica o valor do operador do tipo de
Qualquer valor JavaScript.

Tabela 4-3.Valores retornados pelo operador TypeOf

x	Tipo de x
indefinido	"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
teste 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 matriz
Além 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 instância
operador (ver §4.9.4), o atributo de classe (ver §14.4.3) ou o
Propriedade 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
variedade
2 em 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
existe.Tentando ler uma propriedade inexistente retorna indefinida,
Mas você pode testar a existência real de uma propriedade com o in
operador (§4.9.3).A exclusão de um elemento de matriz deixa um "buraco" na matriz
e 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
O operador não toma ação e retorna verdadeiro.Caso contrário, exclua
Tentativas de excluir o LValue especificado.excluir retorna verdadeiro se for
Exclui com sucesso o LValue especificado.Nem todas as propriedades podem ser
Excluído, 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 função Parâmetro: ele só funciona quando o operando é um acesso à propriedade expressão (§4.4).O modo rigoroso também especifica que o delete levanta um TypeError se solicitado a excluir qualquer não confundível (isto é, nãolável) 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 sintaxerror em
modo rigoroso.
excluir o;
// Propriedade nãolável: retorna falsa, ou TypeError em
modo rigoroso.
excluir object.prototype;
```

Veremos o operador de exclusão novamente no §6.4.

4.13.5 O operador aguardado

aguardar foi introduzido no ES2017 como uma maneira de tornar assíncrono Programação mais natural em JavaScript.Você precisará ler

Capítulo 13 para entender este operador.Resumidamente, no entanto, aguarde
espera 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 aguardar
Operador 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 completo
detalhes.

4.13.6 O operador vazio

Void 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 seta
Sintaxe de atalho (ver §8.1.3) onde o corpo da função é um único
expressão que é avaliada e devolvida.Se você está avaliando o
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 () // => indefinido
contador // => 1
```

4.13.7 O operador de vírgula (,)

O operador de vírgula é um operador binário cujos operandos podem ser de qualquer tipo.Ele avalia seu operando esquerdo, avalia seu operando correto e Em 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 quando A expressão esquerda tem efeitos colaterais.A única situação em que o O operador de vírgula é comumente usado é com um loop (§5.4.3) que possui Variáveis ??de loop múltiplas:

```
// O primeiro vírgula abaixo faz parte da sintaxe do Let
declaração
// O segundo vírgula é o operador de vírgula: nos permite espremer
2
// 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 ?curto
Circuando ?comportamento 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
executado 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
o intérprete executar ou pular outras declarações, dependendo do valor

de uma expressão
Loops
Declarações como enquanto e para esse executar outras declarações
repetidamente
Saltos
Declarações como quebrar, retornar e jogar que causam o
intérprete para pular para outra parte do programa
As seções a seguir descrevem as várias declarações em JavaScript
e 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
estão familiarizados com as declarações de JavaScript, você pode começar a escrever
Programas JavaScript.
5.1 declarações de expressão
Os tipos mais simples de declarações em JavaScript são expressões que
tem efeitos colaterais.Esse tipo de afirmação foi mostrado no capítulo 4.
As declarações de atribuição são uma categoria principal de expressão
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 um
Propriedade 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 como
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 um
Declaração de atribuição.Por exemplo, você não apenas calcula um
Cosseno e descarte o resultado:
Math.cos (x);
Mas você pode calcular o valor e atribuí -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
eles.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
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):
// inicialize uma matriz a
para (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
O 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
Comente 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 Condicionais

Declarações condicionais executam ou pularem outras declarações, dependendo de
o valor de uma expressão especificada.Essas declarações são a decisão
Pontos do seu código, e eles também são conhecidos como "ramos".
Se você imagina um intérprete de javascript seguindo um caminho através do seu
Código, as declarações condicionais são os lugares onde o código é ramifica
em dois ou mais caminhos e o intérprete deve escolher qual caminho
seguir.
As subseções a seguir explicam o condicional básico de JavaScript, o
If/else declaração, e também o switch de capa, mais complicado,
Declaração de Multiway Branch.

5.3.1 se

A declaração IF é a declaração de controle fundamental que permite
JavaScript para tomar decisões, ou, mais precisamente, para executar declarações
condicionalmente.Esta afirmação tem duas formas.O primeiro é:
se (expressão)
declaração
Nesta 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 ou
indefinido,
nome de usuário = "John Doe";// Defina

Ou da mesma forma:
// Se o nome de usuário for nulo, indefinido, falso, 0, "" ou nan, dê
é um novo valor
if (! Nome de usuário) nome de usuário = "John Doe";
Observe que os parênteses ao redor da expressão são uma parte necessária de
a 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 para
Combine várias declarações em uma.Portanto, a declaração se também pode
Parece isso:
if (! endereço) {
endereço = "";
mensagem = "Especifique um endereço de correspondência.";
}
A segunda forma da declaração IF apresenta uma cláusula que é
executado quando a expressão é falsa.Sua sintaxe é:
se (expressão)
Declaração1
outro
Declaração2
Esta 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 cautela
é necessário para garantir que a cláusula elsei
declaração.Considere as seguintes linhas:

```
i = j = 1;
k = 2;
if (i === J)
if (j === k)
console.log ("eu igual a k");
outro
console.log ("Eu não igual a J");// ERRADO!!
```

Neste exemplo, a instrução IF interna forma a instrução única
permitido pela sintaxe da instrução IF Exterior.Infelizmente, é
não claro (exceto da dica dada pelo recuo) que se o
mais vai com.E neste exemplo, o recuo está errado,
Porque um intérprete de javascript realmente interpreta o anterior
Exemplo como:

```
if (i === j) {
if (j === k)
console.log ("eu igual a k");
outro
console.log ("Eu não igual a J");// opa!
}
```

A regra em JavaScript (como na maioria das linguagens de programação) é que por
Padrão Uma cláusula else faz parte da instrução IF mais próxima.Fazer
Este 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 prestio 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
declaração anterior.Usando o else se o idioma for preferível e
mais legível do que, escrevendo essas declarações
Formulário 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 Switch
Uma 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 dependem
no valor da mesma expressão.Nesse caso, é desperdiçado
Avalie 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 de
Código em aparelhos encaracolados:

```
switch (expressão) {
declarações
}
```

No entanto, a sintaxe completa de uma declaração de interruptor é mais complexa do que
esse.Vários locais no bloco de código são rotulados com o caso
Palavra -chave seguida por uma expressão e um cólon.Quando um interruptor
executa, ele calcula o valor da expressão e depois procura um
rótulo de caso cuja expressão avalia o mesmo valor (onde
A semelhança é determinada pelo operador ===).Se encontrar um, começa
executando o bloco de código na instrução rotulada pelo caso.Se isso
não encontra um caso com um valor correspondente, ele procura uma declaração
Padrão rotulado:.Se não houver padrão: etiqueta, o interruptor
A declaração pula o bloco de código completamente.
Switch é uma declaração confusa para explicar;sua operação se torna
Muito mais claro com um exemplo.A seguinte declaração do interruptor é
equivalente às declarações repetidas if/else mostradas no anterior
seção:

```
Switch (n) {
Caso 1: // Comece aqui se n === 1
// Executar o bloco de código #1.
quebrar;// Pare aqui
Caso 2: // Comece aqui se n === 2
// Executar o bloco de código #2.
quebrar;// Pare aqui
Caso 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 interruptor
declaração e continue com a declaração que a segue.O caso
cláusulas em uma declaração de interruptor especificam apenas o ponto de partida do
código desejado;Eles não especificam nenhum ponto final.Na ausência de
Declarações de quebra, uma declaração de interruptor começa a executar seu bloco de
código na etiqueta do caso que corresponde ao valor de sua expressão e
continua executando declarações até chegar ao final do bloco.Sobre
ocasiões raras, é útil escrever um código como esse que "cai"
de uma etiqueta de caso para a seguinte, mas 99% do tempo você deve ser
Cuidado para encerrar todos os casos com uma declaração de quebra.(Ao usar
Mudar dentro de uma função, no entanto, você pode usar uma declaração de retorno
em vez de uma declaração de quebra.Ambos servem para encerrar o interruptor
declaração e impedir a execução de cair para o próximo
caso.)
Aqui está um exemplo mais realista da instrução Switch;ele converte
Um 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 um
Inteiro hexadecimal
retornar x.toString (16);
case "string": // retorna a string fechada
em 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
seguido de literais de número e cordas, respectivamente.É assim que o
A declaração de switch é mais frequentemente usada na prática, mas observe que o
O padrão ECMAScript permite que cada caso seja seguido por um arbitrário
expressão.
A instrução Switch primeiro avalia a expressão que segue o
alternar a palavra -chave e depois avaliar as expressões de caso, no
Ordem em que eles aparecem, até encontrar um valor que corresponda.O
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 caso
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

Exemplos mostrados, o padrão: rótulo aparece no final do
Mudar o corpo, seguindo todos os rótulos da caixa.Este é um lógico e
lugar comum para isso, mas pode realmente aparecer em qualquer lugar dentro do
corpo da declaração.

5.4 Loops

Para entender as declarações condicionais, imaginamos o JavaScript
intérprete seguindo um caminho de ramificação através do seu código -fonte.O
declarações em loop são aquelas que dobram esse caminho
Repita 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.Um
O uso comum para loops é iterar sobre os elementos de uma matriz.§7.6
discute esse tipo de loop em detalhes e abrange métodos de loops especiais
definido pela classe da matriz.

5.4.1 enquanto

Assim como a declaração IF é condicional básico de JavaScript, o tempo todo
A declaração é o loop básico de JavaScript.Tem a seguinte sintaxe:

```
enquanto (expressão)
declaração
```

Para executar um tempo, a declaração, o intérprete avalia primeiro
expressão.Se o valor da expressão for falsamente, então o intérprete
ignora a afirmação que serve como o corpo do loop e segue para
A próxima declaração no programa.Se, por outro lado, a expressão
é verdade, o intérprete executa a declaração e repete, saltando

de volta ao topo do loop e avaliando a expressão novamente.Outro
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
tempo através do loop.Além disso, se a variável em mudança ou
Variáveis ??estão envolvidas na expressão, o valor da expressão pode
Seja 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 o
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
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 usado como contadores de loop, embora você deva usar mais
Nomes descritivos se tornar seu código mais fácil de entender.

5.4.2 Do/while

O loop do DO/While é como um loop de tempo, exceto que o loop
A expressão é testada na parte inferior do loop e não na parte superior.Esse
significa que o corpo do loop é sempre executado pelo menos uma vez.O
Sintaxe é:

fazer
declaração
enquanto (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 um
Loop 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/enquanto
loop e o loop comum.Primeiro, o loop do DO requer ambos
faça palavras -chave (para marcar o início do loop) e enquanto
palavra -chave (para marcar o final e introduzir a condição do loop).Além disso, o
O loop deve sempre ser encerrado com um ponto de vírgula.O loop while
Não precisa de um semicolon se o corpo do loop estiver fechado em aparelhos encaracolados.

5.4.3 para

A declaração for fornece uma construção em loop que geralmente é mais conveniente do que a declaração do tempo.A declaração for simplifica Loops que seguem um padrão comum.A maioria dos loops tem um contador Variável de algum tipo.Esta variável é inicializada antes do início do loop e é testado antes de cada iteração do loop.Finalmente, o balcão A variável é incrementada ou atualizada no final do loop corpo, pouco antes da variável ser testada novamente.Nesse tipo de loop, o inicialização, teste e atualização são os três cruciais manipulações de uma variável de loop.A declaração for codifica cada um de essas três manipulações como expressão e fazem aquelas Expressões Uma parte explícita da sintaxe do loop:

para (inicializar; teste; incremento)
declaração

inicializar, teste e incremento são três expressões (separadas por semicolons) que são responsáveis ??por inicializar, testar e incrementando a variável de loop.Colocando todos eles na primeira linha do O loop facilita o entendimento do que um loop está fazendo e evita erros como esquecer de inicializar ou incrementar o loop variável.

A maneira mais simples de explicar como funciona um loop é para mostrar o equivalente enquanto loop:

```
inicializar;
while (teste) {
declaração
incremento;
}
```

Em outras palavras, a expressão inicializa é avaliada uma vez, antes do
Loop começa.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 antes
cada iteração e controla se o corpo do loop é executado.Se
O teste avalia a um valor verdadeiro, a afirmação que é o corpo do
O 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
seguindo.Contrastá -lo com o equivalente enquanto o loop mostrado no
Seção anterior:

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

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

```
deixe eu, j, soma = 0;
para (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 um
Para 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 o
para (; o.next; o = o.next) / * vazio * /;// atravessar enquanto
O.Next é verdade
retornar o;
}
```

Observe que este código não possui expressão inicializa.Qualquer um dos três
Expressões podem ser omitidas de um loop para o loop, mas os dois semicolons
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
o 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
Iterável: eles representam uma sequência ou conjunto de elementos que você pode fazer
ou iterar usando um loop for/of.

Aqui, por exemplo, é como podemos usar para/de para fazer um loop através do
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 para
A palavra -chave é seguida por parênteses que contêm detalhes sobre o que o
Loop 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 um
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 do
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
iterado 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 por
adicionar os dados da linha.push (soma);dentro do corpo do loop, então nós
criar 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 um
Objeto 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
iterável
console.log (elemento);
}
```

Se você deseja iterar através das propriedades de um objeto, você pode usar
o loop for/in (introduzido no §5.4.5), ou uso para/de com o
Método 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 propriedades
Nomes para um objeto, e as matrizes são iteráveis ??com/of.Nota também
que essa iteração das chaves de um objeto não é viva como a matriz
Exemplo acima foi - mudanças para o objeto O feito no corpo do loop
não terá efeito na iteração.Se você não se importa com as chaves de
um objeto, você também pode iterar através dos valores correspondentes como
esse:

```
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 objeto
Propriedades, você pode usar para/de object.entries () e
atribuição 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 interior
Array representa um par de chaves/valor para uma propriedade do objeto.Nós usamos
Descrutamento de destruição neste exemplo de código para descompactar os internos
Matrizes em duas variáveis ??individuais.
Para/de strings
As cordas são iteráveis ??de caractere por caracteres 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-16
personagem.A string ?eu ?
?Tem um comprimento de 5 (porque os dois
Cada um dos caracteres emoji exige dois caracteres UTF-16 para representar).Mas
Se você itera essa corda com para/de, o corpo do loop executará três
vezes, uma vez para cada um dos três pontos de código "eu", "?" e "
. ?

Para/de com set e mapa

As classes de conjunto e mapa do ES6 embutidas são iteráveis.Quando você itera um
Defina 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 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 teclas do mapa ou os valores do mapa, mas pares de chave/valor.Cada
Tempo através da iteração, o iterador retorna uma matriz cujo primeiro
O 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/aguardar

O ES2018 apresenta um novo tipo de iterador, conhecido como um assíncrono
iterador, e uma variante no loop for/de
Para/aguarda o loop que funciona com iteradores assíncronos.

Você precisará ler os capítulos 12 e 13 para entender o
para/aguarda loop, mas aqui está como ele fica no código:

```
// leia pedaços de um fluxo de maneira assíncrona e
Imprima -os
Função assíncreada printStream (stream) {
para aguardar (Let Chunk of Stream) {
console.log (pedaço);
}
}
```

5.4.5 para/in

A for/in loop se parece muito com um loop para/de uma palavra -chave
alterado 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 JavaScript
Desde o começo (e é por isso que tem o mais natural
Sintaxe de som).
O para/in declaração circula através dos nomes de propriedades de um
objeto especificado.A sintaxe se parece com a seguinte:

```
para (variável no objeto)
declaração
```

A variável normalmente nomeia uma variável, mas pode ser uma declaração variável
ou qualquer coisa adequada como o lado esquerdo de uma expressão de atribuição.
Objeto é uma expressão que avalia para um objeto.Como sempre, declaração
é 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 p
console.log (O [P]);// Imprima o valor de cada propriedade
}
Para executar uma declaração para/in, o intérprete JavaScript primeiro
Avalia a expressão do objeto.Se avaliar para nulo ou
indefinido, o intérprete pula o loop e passa para o próximo
declaração.O intérprete agora executa o corpo do loop uma vez para
cada 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 arbitrário
expressão, desde que avalie para algo adequado para o lado esquerdo
de uma tarefa.Esta expressão é avaliada a cada vez através do
Loop, 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 todos
Propriedades 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 matriz
Índices são propriedades de objetos que podem ser enumerados com um para/in
laço.Por exemplo, seguindo o código anterior com esta linha
enumera 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 vez
de for/in.
O loop for/in não enumora todas as propriedades de um
objeto.Não enumora propriedades cujos nomes são símbolos.
E das propriedades cujos nomes são strings, ele apenas se arrasta sobre o
propriedades enumeráveis ??(ver §14.1).Os vários métodos internos
Definido pelo JavaScript Core não é enumerável.Todos os objetos têm um
Método ToString (), por exemplo, mas o for/in loop não
Enumere esta propriedade ToString.Além de métodos internos,
Muitas outras propriedades dos objetos embutidos não são adequados.Todos
Propriedades e métodos definidos pelo seu código são enumeráveis, por
padrão.(Você pode torná-los não-enumeráveis ??usando técnicas
explicado no §14.1.)
Propriedades herdadas enumeráveis ??(ver §6.3.2) também são enumeradas por
o loop for/in.Isso significa que se você usar para/em loops e também
Use o código que define propriedades herdadas por todos os objetos, então
Seu loop pode não se comportar da maneira que você espera.Por esse motivo, muitos
Os 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 foi
enumerado, essa propriedade não será enumerada.Se o corpo do
Loop define novas propriedades no objeto, essas propriedades podem ou podem
não 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 saltos

Outra categoria de declarações de JavaScript são as declarações de salto.Como o
O nome indica, eles fazem com que o intérprete JavaScript salte para um novo
Localização no código -fonte.A declaração de quebra faz o
INTERPRETER SULT até o final de um loop ou outra declaração.continuar
faz com que o intérprete pule o resto do corpo de um loop e pule de volta
até o topo de um loop para iniciar uma nova iteração.JavaScript permite
declarações a serem nomeadas ou rotuladas, e quebrar e continuar podem
Identifique o loop de destino ou outro rótulo de instrução.

A declaração de retorno faz o intérprete saltar de uma função
invocação de volta ao código que o invocou e também fornece o valor
para a invocação.A declaração de arremesso é uma espécie de retorno provisório
de uma função de gerador.A declaração de arremesso levanta, ou joga, um
exceção e foi projetada para trabalhar com a tentativa/captura/finalmente
Declaração, que estabelece um bloco de código de manuseio de exceção.Esse
é um tipo complicado de declaração de salto: quando uma exceção é lançada,
O intérprete salta para o manipulador de exceção de anexo mais próximo, que
pode estar na mesma função ou na pilha de chamadas em uma invocação
função.

Detalhes sobre cada uma dessas declarações de salto estão nas seções que
seguir.

5.5.1 Declarações rotuladas

Qualquer declaração pode ser rotulada precedindo -a com um identificador e um
cólon:
Identificador: declaração
Ao rotular uma declaração, você dá um nome que você pode usar para se referir
em outros lugares do seu programa.Você pode rotular qualquer declaração, embora ela
é útil apenas para rotular declarações que têm corpos, como loops e
condicionais.Ao dar um nome a um loop, você pode usar quebra e
Continue as declarações dentro do corpo do loop para sair do loop ou para
Salte diretamente para o topo do loop para iniciar a próxima iteração.quebrar
e continuar são as únicas declarações JavaScript que usam a declaração
etiquetas;Eles são abordados nas seguintes subseções.Aqui está um
Exemplo de um rotulado enquanto loop e uma declaração continuada que
usa o rótulo.
MAINLOOP: while (token! == null) {
// Código omitido ...
Continue Mainloop;// pule para a próxima iteração do
Nomeado loop
// mais código omitido ...
}
O identificador que você usa para rotular uma declaração pode ser qualquer javascript legal
Identificador que não é uma palavra reservada.O espaço para nome para rótulos é
diferente do espaço para nome para variáveis ??e funções, então você pode
Use 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 qual
Eles se aplicam (e dentro de suas sub -subestimações, é claro).Uma declaração pode
não tem o mesmo rótulo que uma declaração que o contém, mas dois
As 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 cólon):

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 o
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
em algum lugar
// Agora soma todos os números na matriz.
Deixe Sum = 0, Sucesso = Falso;
// Comece com uma declaração rotulada de que podemos sair se
Erros ocorrem
Computesum: 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 estávamos
dado.
// caso contrário, a soma contém a soma de todas as células do
matriz.
```

Finalmente, observe que uma declaração de quebra, com ou sem rótulo, não pode Transferir controle através dos limites da função.Você não pode rotular um declaração de definição de função, por exemplo, e depois use esse rótulo dentro da função.

5.5.3 Continue

A declaração continua é semelhante à declaração de quebra.Em vez de de sair de um loop, no entanto, continue reinicia um loop no próximo iteração.A sintaxe da declaração continua é tão simples quanto a Declaração 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 O 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 de O 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 o A expressão do teste é testada novamente para determinar se outra iteração deve ser feito.

Em um para/de ou para/em loop, o loop começa com o Próximo valor iterado ou o próximo nome da propriedade sendo atribuído ao variável especificada.

Observe a diferença de comportamento da declaração continuação 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 continuam 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 loop sozinho.

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
Ocorre erro:

```
para (vamos i = 0; i <data.length; i ++) {
se (! dados [i]) continuar;// não pode prosseguir com indefinidamente
dados
total += 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 sua
LabelName.

5.5.4 Retorno

Lembre -se de que as invocações de função são expressões e que todos
Expressões têm valores.Uma declaração de retorno dentro de uma função
Especifica o valor das invocações dessa função.Aqui está a sintaxe de
A 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
A declaração 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 um
Declaração de retorno
```

quadrado (2) // => 4
Sem declaração de retorno, uma invocação de funções simplesmente executa
cada uma das declarações no corpo da função, por sua vez, até chegar ao
Fim da função e depois retorna ao seu chamador.Nesse caso, o
A expressão de invocação é avaliada como indefinida.O retorno
A declaração geralmente aparece como a última declaração em uma função, mas precisa
não ser o último: uma função retorna ao seu chamador quando uma declaração de retorno é
executado, mesmo que haja outras declarações restantes na função
corpo.
A declaração de retorno também pode ser usada sem expressão para
Faça a função retornar indefinida ao seu chamador.Por exemplo:
Função DisplayObject (O) {
// retorna imediatamente se o argumento for nulo ou
indefinido.
se (! O) retornar;
// O restante da função vai aqui ...
}
Por causa da inserção automática de semicolon do JavaScript (§2.6), você
não pode incluir uma quebra de linha entre a palavra -chave de retorno e o
expressão que a segue.
5.5.5 Rendimento
A declaração de rendimento é muito parecida com a declaração de retorno, mas é usada
Somente nas funções do gerador ES6 (consulte §12.3) para produzir o próximo valor
Na 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 e
geradores, que não serão cobertos até o capítulo 12. O rendimento é
Incluído 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 Jogue

Uma exceção é um sinal que indica que algum tipo de excepcional
condição ou erro ocorreu.Dar uma exceção é sinalizar tal
um erro ou condição excepcional.Pegar uma exceção é lidar com isso
- para tomar as ações necessárias ou apropriadas para se recuperar de
a exceção.Em JavaScript, as exceções são jogadas sempre que um tempo de execução
o erro ocorre e sempre que o programa joga explicitamente um usando o
Declaração de arremesso.Exceções são capturadas com o
Tente/Catch/Finalmente declaração, descrita no próximo
seção.

A declaração de arremesso tem a seguinte sintaxe:

```
lançar expressão;
```

A expressão pode avaliar um valor de qualquer tipo.Você pode jogar um
número que representa um código de erro ou uma string que contém um humano
Mensagem de erro legível.A classe de erro e suas subclasses são usadas
Quando 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) // => 24
```

Quando 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 um
a 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 JavaScript
mecanismo de manuseio.A cláusula de tentativa desta afirmação simplesmente define
O bloco de código cujas exceções devem ser tratadas.O bloco de tentativa
é seguido por uma cláusula de captura, que é um bloco de declarações que são
Invocado quando ocorre uma exceção em qualquer lugar dentro do bloco de tentativa.
A cláusula de captura é seguida por um bloco finalmente contendo
Código de limpeza que é garantido para ser executado, independentemente do que
acontece no bloco de tentativa.Tanto a captura quanto finalmente os blocos são
opcional, mas um bloco de tentativa deve ser acompanhado por pelo menos um desses
blocos.A tentativa, a captura e finalmente os blocos começam e terminam com
aparelho encaracolado.Esses aparelhos são uma parte necessária da sintaxe e não podem
Seja omitido, mesmo que uma cláusula contenha apenas uma única declaração.
O código a seguir ilustra a sintaxe e o objetivo do
Tente/Catch/Finalmente declaração:

```
tentar {
// normalmente, esse código é executado do topo do bloco para
o fundo
// sem problemas.Mas às vezes pode jogar um
exceção,
// seja diretamente, com uma declaração de arremesso, ou
Indiretamente, ligando
// Um ??método que lança uma exceção.
}
Catch (e) {
// As declarações neste bloco são executadas se, e somente
Se, a tentativa
// Block lança uma exceção.Essas declarações podem usar
a variável local
// e para se referir ao objeto de erro ou outro valor que foi
jogado.
// Este bloco pode lidar com a exceção de alguma forma, pode
ignore o
// exceção fazendo nada, ou pode repensar o
exceção com arremesso.
```

}
finalmente {
// Este bloco contém declarações que são sempre
executado, 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 captura
Cláusula acima
// 4) com uma exceção não capturada que ainda é
propagação
}
Observe que a palavra -chave de captura geralmente é seguida por um identificador em
parênteses.Este identificador é como um parâmetro de função.Quando um
Exceção é capturada, o valor associado à exceção (um erro
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 e
saída:
tentar {
// Peça ao usuário para inserir um número
Seja n = número (prompt ("insira um número inteiro positivo",
""));
// calcular o fatorial do número, assumindo
A entrada é válida
Seja f = fatorial (n);
// exibe o resultado
alerta (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 for
executado, 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 do
tente 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 um
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 processo
de ser jogado.Se uma cláusula finalmente emitir uma declaração de devolução, o
o 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
declarações 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 nua
Ocasionalmente, 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 ES2019
E mais tarde, você pode omitir os parênteses e o identificador e usar a palavra -chave de captura nua neste
caso.Aqui está um exemplo:

```
// como json.parse (), mas retorne indefinido em vez de jogar um erro
função parsejson (s) {
tentar {
retornar json.parse (s);
} pegar {
// algo deu errado, mas não nos importamos com o que foi
retornar indefinido;
}
}
```

5.6 Declarações diversas

Esta seção descreve as três declarações restantes de JavaScript
- com, depurador e "use rigoroso".

5.6.1 com

A declaração com coma um bloco de código como se as propriedades de um
Objeto especificado foi variável no escopo para esse código.Tem o
Após a sintaxe:

```
com (objeto)
declaração
```

Esta afirmação cria um escopo temporário com as propriedades do objeto
como variáveis ??e, em seguida, executa a declaração dentro desse escopo.
A declaração com com
ser considerado depreciado no modo não rigoroso: evite usá-lo sempre que
possível.O código JavaScript que usa é difícil de otimizar e é
Provavelmente correr significativamente mais lentamente do que o código equivalente escrito
sem 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, para
Por 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 pode
Use 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 mais
Para prefixar cada nome da propriedade do formulário com document.forms [0].Isso é
Tão simples, é claro, para evitar a declaração com e escrever o
Código 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 ou
constante dentro do corpo de uma declaração, cria um comum
variável e não define uma nova propriedade dentro do objeto especificado.

5.6.2 Depurador

A declaração do depurador normalmente não faz nada.Se, no entanto, a
O programa depurador está disponível e está em execução, depois uma implementação
pode (mas não é necessário) executar algum tipo de ação de depuração.Em
Prática, esta afirmação age como um ponto de interrupção: execução de javascript
O 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 seja
obtendo uma exceção em sua função f () porque está sendo chamada
com um argumento indefinido, e você não consegue descobrir onde está essa chamada
vindo de.Para ajudá -lo a depurar esse problema, você pode alterar
f () para que comece assim:

```
função f (o) {
if (o === indefinido) depurador;// linha temporária para
fins de depuração
... // o resto da função
vai aqui.
}
```

Agora, quando f () é chamado sem argumento, a execução vai parar e
Você pode usar o depurador para inspecionar a pilha de chamadas e descobrir onde
Esta chamada incorreta está vindo.

Observe que não basta ter um depurador disponível: o depurador
A declaração não começará o depurador para você.Se você está usando uma web
navegador e ter o console de ferramentas de desenvolvedor aberto, no entanto, este
A 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
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
O corpo da função, antes que quaisquer declarações reais aparecessem.
O objetivo de uma diretiva "uso rigoroso" é indicar que o código
A 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
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
corpo (capítulo 9) ou em um módulo ES6 (§10.3) é automaticamente rigoroso
código.Isso significa que se todo o seu código JavaScript for escrito como
módulos, então tudo é automaticamente rigoroso, e você nunca precisará
Use 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
adicionando 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
Este valor é exatamente o valor passado como o primeiro argumento para
ligue () ou aplique ().(No modo não estrito, nulo 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
Jogue 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) em 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 nomeados
Ambos se referem ao mesmo valor.
No modo rigoroso, um SyntaxError é jogado se a exclusão
O operador é seguido por um identificador não qualificado, como um
Parâmetro variável, função ou função.(No modo não
Essa expressão de exclusão não faz nada e avalia para
falso.)
No modo rigoroso, uma tentativa de excluir uma propriedade não configurável
joga um TypeError.(No modo não rito, a tentativa falha e
A 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 para
Tenha dois ou mais parâmetros com o mesmo nome.(Em não
modo 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 seus
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 e
Propriedades 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
a 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 faça muito acontecer, mas fornecendo nomes para
valores eles, em um sentido importante, definem o significado do outro
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.O
declarações em um programa não "correm" da mesma maneira: em vez disso, elas
Defina a estrutura do próprio programa.Vagamente, você pode pensar em
declarações como as partes do programa que são processadas antes do
O código começa 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
declarações.Todos estão cobertos com muito mais detalhes em outros lugares em

este livro.

5.7.1 const, let e var

As declarações Const, Let e ??VAR são cobertas em §3.10.Em ES6
e mais tarde, Const declara constantes e deixe declara variáveis.Anterior
Para ES6, a palavra -chave VAR era a única maneira de declarar variáveis ??e
Não havia como declarar constantes.Variáveis ??declaradas com var são
escopo para a função contendo, em vez do bloco contendo.Esse
pode ser uma fonte de bugs e, no javascript moderno, não há realmente não
Razão para usar o VAR em vez de Let.

```
const tau = 2*math.pi;
Deixe o raio = 3;
var circunferência = tau * raio;
```

5.7.2 Função

A declaração de função é usada para definir funções, que são
coberto 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 um
declaração 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 ao
Nome especificado - Área neste exemplo.Em outros lugares do nosso programa, nós
pode se referir à função - e executar o código dentro dela - usando isso
nome.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 a
a função se observa em todo o bloco.Dizemos essa função
As declarações são "içadas" porque é como se todas tivessem sido movidas
até 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 código
que 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 com
um asterisco.§13.3 descreve funções assíncronas, que também são
declarado usando a palavra -chave da função, mas é prefixado com o
palavra -chave assíncrona.

5.7.3 Classe

No ES6 e mais tarde, a declaração de classe cria uma nova classe e dá
É um nome que podemos usar para referir a ele.As aulas são descritas em detalhes em
Capítulo 9. Uma declaração simples de classe pode ser assim:

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

Ao contrário das funções, as declarações de classe não são içadas e você não pode usar
Uma classe declarada dessa maneira no código que aparece antes da declaração.

5.7.4 Importação e exportação

As declarações de importação e exportação são usadas juntas para fazer

valores definidos em um módulo de código JavaScript disponível em outro
módulo.Um módulo é um arquivo de código JavaScript com seu próprio global
namespace, completamente independente de todos os outros módulos.A única
como um valor (como função ou classe) definido em um módulo pode
ser usado em outro módulo é se o módulo definidor o exportar
exportar e o uso do módulo importa com importação.Módulos são
o assunto do capítulo 10 e a importação e exportação são cobertas em
Detalhes em §10.3.
Diretivas de importação são usadas para importar um ou mais valores de outro
Arquivo do código JavaScript e dê -lhes nomes no módulo atual.
As diretrizes de importação vêm em algumas formas diferentes.Aqui estão alguns
Exemplos:

```
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 importados
em outros módulos, a menos que tenham sido exportados explicitamente.O
A Diretiva de Exportação faz isso: declara que um ou mais valores definidos
no módulo atual são exportados e, portanto, disponíveis para importação
por outros módulos.A diretiva de exportação tem mais variantes do que o
A Diretiva de Importação faz.Aqui está um deles:

```
// geometria/constants.js
const 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 um constante, variável, função ou classe e a exporta ao mesmo tempo.
E quando um módulo exporta apenas um valor, isso geralmente é feito
Com 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 omitida
aqui */ }
```

5.8 Resumo das declarações JavaScript

Este capítulo introduziu cada uma das declarações da linguagem JavaScript, que estão resumidos na Tabela 5-1.

Tabela 5-1.Declaração JavaScript Sintaxe

Declaração	Propósito
quebrar	Saia do loop ou interruptor mais íntimo ou do nome nomeado
declaração	
caso	Rotule uma declaração dentro de um interruptor
aula	Declarar uma aula
const	Declarar e inicializar uma ou mais constantes
continuar	Comece a próxima iteração do loop mais interno ou o loop nomeado
Depurador	Ponto de interrupção do depurador
padrão	Rotule a instrução padrão em um interruptor
faça/while	Uma alternativa ao loop while

exportar
Declarar valores que podem ser importados para outros módulos
para
Um loop fácil de usar
para/aguardar
Iteram de forma assíncrona os valores de um iterador assíncrono
para/in
Enumerar os nomes de propriedades de um objeto
para/de
Enumerar os valores de um objeto iterável, como uma matriz
função
Declarar uma função
se/else
Executar uma declaração ou outra dependendo de uma condição
importar
Declarar nomes para valores definidos em outros módulos
rótulo
Dê um nome para uso com quebra e continue
deixar
Declare e inicialize uma ou mais variáveis ??escassas de blocos (Novo
sintaxe)
retornar
Retornar um valor de uma função
trocar
Filial Multiway para Case ou Padrão: Rótulos
lançar
Jogue uma exceção
tente/capturar/final
ly
Lidar com exceções e limpeza de código
?Use rigoroso?
Aplique restrições de modo rigoroso para script ou função
var
Declare e inicialize uma ou mais variáveis ??(sintaxe antiga)
enquanto
Uma construção de loop básico
com
Estender a cadeia de escopo (depreciada e proibida no modo rigoroso)
colheita
Fornecer um valor a ser iterado;usado apenas nas funções do gerador
1
O fato de as expressões de caso serem avaliadas em tempo de execução faz do JavaScript
Declaração 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: agrega vários valores (primitivo
valores 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 um Principais características do JavaScript.

Os objetos JavaScript são dinâmicos - as propriedades geralmente podem ser adicionadas e excluídos - mas eles podem ser usados ??para simular os objetos estáticos e ?Estruturas? de idiomas típicos estaticamente.Eles também podem ser usados ??(por ignorando 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 Seja 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 criá -los e definir, Consulta, exclua, teste e enumerar suas propriedades.Estes fundamentais As 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 qualquer
Valor 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 propriedades
definido diretamente em um objeto e aqueles que são herdados de um
Objeto de protótipo.JavaScript usa o termo propriedade própria para se referir a não
propriedades herdadas.
Além de seu nome e valor, cada propriedade possui três propriedades
Atributos:
O atributo gravável especifica se o valor do
A propriedade pode ser definida.
O atributo enumerável especifica se o nome da propriedade
é devolvido por um loop for/in.
O atributo configurável especifica se a propriedade pode
ser excluído e se seus atributos podem ser alterados.
Muitos dos objetos internos de JavaScript têm propriedades que são lidas
Somente, não inebriante, ou não confundível.Por padrão, no entanto, todos
As propriedades dos objetos que você cria são graváveis, enumeráveis ??e
configurável.§14.1 explica técnicas para especificar não-defasa
Valores do atributo de propriedade para seus objetos.
6.2 Criando objetos
Objetos podem ser criados com literais de objeto, com a nova palavra -chave, e
com 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
Lista separada de nome separado pelo cólon: pares de valor, fechados dentro
aparelho encaracolado.Um nome de propriedade é um identificador JavaScript ou uma string
literal (a corda vazia é permitida).Um valor de propriedade é qualquer javascript
expressão;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 deste
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 e
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 do
A 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 isso
está coberto no capítulo 9.

6.2.3 Protótipos

Antes 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 um
Invocação do construtor Use o valor da propriedade do protótipo de
o 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 novo
Array () usa Array.prototype como seu protótipo e o objeto
Criado 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 com
propriedades 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 normais
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 por
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 retornado
por {} ou novo objeto ()), aprove o object.prototype:
Seja o3 = object.create (object.prototype);// o3 é como {} ou
novo objeto ().
A capacidade de criar um novo objeto com um protótipo arbitrário é um
poderoso, e usaremos object.create () em vários
lugares ao longo deste capítulo.(Object.create () também leva um
segundo argumento opcional que descreve as propriedades do novo
objeto.Este segundo argumento é um recurso avançado coberto no §14.1.)
Um uso para object.create () é quando você deseja se proteger
Modificação não intencional (mas não maliciosa) de um objeto por uma biblioteca
função que você não tem controle.Em vez de passar o objeto
Diretamente para a função, você pode passar um objeto que herda dela.Se
A função lê propriedades desse objeto, verá o herdado
valores.Se definir propriedades, no entanto, essas gravações não afetarão o
objeto original.

```
Seja o = {x: "Não altere este valor"};
Library.function (object.create (o));// se protege contra
modificações acidentais
```

Para entender por que isso funciona, você precisa saber como são as propriedades
consultado e definido em JavaScript.Estes são os tópicos da próxima seção.

6.3 Propriedades de consulta e definição

Para 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, o
O lado arete deve ser um identificador simples que nomeia a propriedade.Se
Usando suportes quadrados, o valor dentro dos colchetes deve ser um
expressão que avalia a uma string que contém a propriedade desejada
nome:

```
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 faria
para consultar a propriedade, mas coloque -os no lado esquerdo de um
Expressão de atribuição:

```
book.edition = 7;// Crie uma "edição"
propriedade do livro.
livro ["título principal"] = "ecmascript";// Alterar o "principal
título "Propriedade.
```

Ao usar a notação de suporte quadrado, dissemos que a expressão
Dentro dos suportes quadrados devem avaliar uma corda.Um mais preciso
declaração é que a expressão deve avaliar uma string ou um valor que
pode ser convertido em uma corda ou em um símbolo (§6.10.3).No capítulo 7, para
Por exemplo, veremos que é comum usar números dentro do quadrado
Suportes.

6.3.1 Objetos como matrizes associativas

Conforme explicado na seção anterior, os dois JavaScript a seguir

Expressões têm o mesmo valor:
Object.Property
Objeto ["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
a uma matriz indexada por strings e não por números.Esse tipo de
A 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 destes
As 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
manipulado 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.Cordas
são 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 matriz
Para acessar as propriedades de um objeto com expressões de string.Este código
pode ser reescrito usando a notação de pontos, mas há casos em que
Somente a notação da matriz serve.Suponha, por exemplo, que você seja
Escrever um programa que usa recursos de rede para calcular o atual
Valor dos investimentos no mercado de ações do usuário.O programa permite o
Usuário 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 Portfolio
Para manter essas informações.O objeto possui uma propriedade para cada estoque.
O nome da propriedade é o nome do estoque e a propriedade
O valor é o número de ações dessas ações.Então, por exemplo, se um usuário
detém 50 ações da IBM, a propriedade portfolio.ibm
o valor 50.
Parte deste programa pode ser uma função para adicionar um novo estoque ao
Portfólio:
função addstock (portfólio, nome da estoque, ações) {
Portfolio [SockName] = ações;
}
Como o usuário insere nomes de estoque em tempo de execução, não há como você
pode 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
use 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 o
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
o 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 um 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 deve considerar o acesso à propriedade com mais detalhes.Os exemplos neste seção Use a função object.create () para criar objetos com Protótipos especificados.Veremos no capítulo 9, no entanto, que toda vez Você cria uma instância de uma classe com nova, você está criando um objeto Isso herda as propriedades de um objeto de protótipo.

Suponha que você consulte a propriedade x no objeto o.Se O não tiver um propriedade própria com esse nome, o protótipo objeto de O é consultado para a propriedade x.Se o protótipo 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 de onde as propriedades são herdadas:

```
Seja o = {};// o herda métodos de objeto de
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,
e...
q.z = 3;// ... object.prototype e tem um
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
escondido 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 leitura
Nomeado X, por exemplo, a atribuição não é permitida.(Detalhes
sobre quando uma propriedade pode ser definida está em §6.3.3.) Se a tarefa for
permitido, no entanto, sempre cria ou define uma propriedade no original
Objeto e nunca modifica objetos na cadeia de protótipos.O fato disso
A herança ocorre ao consultar propriedades, mas não quando as definir
é 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
r
c.x = 1;c.y = 1;// c define dois
propriedades próprias
c.r = 2;// c substitui seu
propriedade herdada
unitcircle.r // => 1: o protótipo é
não 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 setter
Método (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 o
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.Esse
a seção explica as coisas que podem dar errado quando você consulta ou define um
propriedade.

Não é um erro consultar uma propriedade que não existe.Se a propriedade
x não é encontrado 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
Nosso objeto 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,
E 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 livro. Autora são realmente definidos.Aqui estão

Duas maneiras de se proteger contra esse tipo de problema:

```
// Uma técnica detalhada e explícita
Seja sobrenome = indefinido;
if (livro) {
if (book.author) {
Sobrenome = book.author.surname;
}
}
// Uma alternativa concisa e idiomática para obter sobrenome ou nulo
ou indefinido
Sobrenome = book && book.author && book.author.surname;
```

Para entender por que essa expressão idiomática funciona para prevenir
Exceções TypeError, você pode querer revisar o curto-circuito
Comportamento do operador && em §4.10.1.
Conforme descrito em §4.4.1, o ES2020 suporta acesso à propriedade condicional
com?., que nos permite reescrever a tarefa anterior
expressão como:

```
Deixe o sobrenome = livro? .Author? .Surname;
```

Tentar definir uma propriedade em nulo ou indefinido também causa um
TypeError.Tentativas de definir propriedades em outros valores nem sempre
Ter sucesso, também: algumas propriedades são somente leitura e não podem ser definidas, e
Alguns objetos não permitem a adição de novas propriedades.Em rigoroso
Modo (§5.6.3), um TypeError é jogado sempre que uma tentativa de definir um
A propriedade falha.Fora do modo rigoroso, essas falhas geralmente são silenciosas.
As regras que especificam quando uma atribuição de propriedade é bem -sucedida e quando
As 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
Defina 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
definido 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 excluí -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 um
expressão que não é uma expressão de acesso à propriedade:
Seja o = {x: 1};// O tem propriedade própria x e herda
Propriedade ToString
Exclua O.x // => true: exclui a propriedade x
Exclua o.x // => true: não faz nada (x não existe)
Mas é verdade de qualquer maneira
excluir o.toString // => true: não faz nada (a tostragem não é
uma propriedade própria)
exclua 1 // => true: bobagem, mas verdade de qualquer maneira
Delete não remove propriedades que têm um atributo configurável
de falso.Certas propriedades de objetos embutidos não são confundíveis,
assim como as propriedades do objeto global criado pela Declaração Variável
e declaração de função.No modo rigoroso, tentando excluir um não
A propriedade configurável causa um TypeError.No modo não estrito, exclua
Simplesmente avalia o FALSE neste caso:
// No modo rigoroso, todas essas deleções jogam TypeError
Em 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 isso
propriedade
função f () {} // declarar uma função global
Exclua globalthis.f // => false: Não é possível excluir isso
propriedade também
Ao excluir propriedades configuráveis ??do objeto global no não rito
modo, 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
excluído
```

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
excluir 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 no operador, com o HasownProperty () e PropertyisEnumerable () métodos, ou simplesmente consultando o propriedade.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 um 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 isso
O objeto possui uma propriedade própria com o nome fornecido.Retorna falsa para
Propriedades herdadas:
Seja o = {x: 1};
O.HasownProperty ("x") // => true: o tem um próprio
Propriedade x
O.HasownProperty ("y") // => false: o não tem um
Propriedade y
O.HasownProperty ("ToString") // => false: ToString é um
propriedade herdada
O PropertyIsenumerable () refina o
teste HASOWNPROPERTY ().Ele retorna verdadeiro apenas se o nomeado
A propriedade é uma propriedade própria e seu atributo enumerável é verdadeiro.
Certas propriedades embutidas não são enumeráveis.Propriedades criadas por
O código JavaScript normal é enumerável, a menos que você tenha usado um dos
Técnicas mostradas no §14.1 para torná-las que não são inebriantes.
Seja o = {x: 1};
O.Propertyisenumerable ("x") // => true: o tem um próprio
propriedade enumerável x
O.Propertyisenumerable ("ToString") // => false: não um próprio
propriedade
Object.prototype.propertyisenumerable ("tostring") // =>
Falso: não enumerável
Em vez de usar o operador em In, muitas vezes é suficiente simplesmente consultar
a propriedade e o uso! == para garantir que não seja indefinido:
Seja o = {x: 1};
o.x! == indefinido // => true: o tem uma propriedade x
o.y! == indefinido // => false: o não tem um
Propriedade y
O.ToString! == indefinido // => true: o herda uma toque

propriedade
Há uma coisa que o operador pode fazer que a propriedade simples
A técnica de acesso mostrada aqui não pode fazer.pode distinguir entre
propriedades 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 indefinido
O.Y! == indefinido // => Falso: Propriedade nem mesmo
existe
"X" em o // => true: a propriedade existe
"y" em o // => false: a propriedade não
existe
excluir o.x;// Exclua a propriedade x
"x" em o // => false: não existe
não mais
6.6 Propriedades de enumeração
Em vez de testar a existência de propriedades individuais, nós
às vezes quero 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.Embutido
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
enumerável
para (vamos P in O) {// percorrer o
propriedades
console.log (P);// imprime x, y e z,
mas não tostring
}
Para se proteger contra a enumitação 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/deExistem quatro funções que você pode usar para obter um
Array de nomes de propriedades:
Object.Keys () retorna uma matriz dos nomes do
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 própria
propriedades cujos nomes são símbolos, sejam eles
enumerável.
Reflete.ownkeys () retorna todos os próprios nomes de propriedades, ambos
enumerável e não enumerável, e símbolo e símbolo.
(Veja §14.6.)
Existem exemplos do uso de object.keys () com um para/de
Loop em §6.7.
6.6.1 Ordem de enumeração da propriedade
ES6 define formalmente a ordem em que as próprias propriedades de um
objeto 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 não
propriedades ou propriedades enumeráveis ??cujos nomes são strings ou
Símbolos:
Propriedades de string cujos nomes são inteiros não negativos são
Listado primeiro, em ordem numérica do menor ao maior.Esta regra
significa que matrizes e objetos semelhantes a matrizes terão seu
propriedades enumeradas em ordem.
Afinal, todas as propriedades que parecem índices de matriz estão listadas, todas
As propriedades restantes com nomes de strings estão listadas (incluindo
propriedades que parecem números negativos ou ponto flutuante
números).Essas propriedades estão listadas na ordem em que
Eles 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 o
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
Execute 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 próprio
Propriedades 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
o 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 alvo
Objeto 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 e
você deseja copiar essas propriedades padrão em outro objeto se um
A 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 e 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 RESTRITCI () pode excluir propriedades de um objeto se não aparecerem em outro objeto de modelo, por exemplo.Ou uma função subtract () pode remover toda a propriedades de um objeto de outro objeto.

6.8 Objetos serializados

A 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 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
objeto
Seja s = json.Stringify (O);// s == '{"x": 1, "y": {"z":
[Falso, 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.Data Os objetos são serializados para as seqüências de datas formatadas iso (ver o Date.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 o 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 JSON.PARSE () aceitam opcional Segundo 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
criado 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 o
Métodos HASOWNPROPERTY () e PROPRIEDISENUMELE (),
por exemplo.(E também já cobrimos alguns estáticos
funções definidas 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
converta 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 do
elementos da matriz, eles mesmos se converteram em uma corda, e quando um
A função é convertida em uma string, você obtém o código -fonte para o
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 um
tolocalestring ().O objetivo deste método é retornar um
Representação de string localizada do objeto.O padrão
O método tolocalestring () definido pelo objeto não faz nenhum
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 ()
métodos em vez de seus métodos tostring ().Você pode fazer o
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)": Nota
```

milhares de separadores
As aulas de internacionalização documentadas no §11.7 podem ser úteis
Ao 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 JavaScript
Este método automaticamente se um objeto for usado em um contexto em que um
É necessário valor primitivo.O método padrão de valueof () faz
Nada 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 desejado
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 mova -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 // => 3
```

Com esta nova sintaxe, os parênteses quadrados delimitam um arbitrário
Expressão de JavaScript.Essa expressão é avaliada e o resultante
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
nomes.Em vez disso, você pode achar que torna seu código mais robusto para
Use 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
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
Use esse símbolo como um nome de propriedade usando a propriedade computada
sintaxe:

```
Extensão const = símbolo ("meu símbolo de extensão");
Seja 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 a
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 corda
para symbol (), e essa string é usada quando seu símbolo é convertido
para uma string.Mas este é apenas um auxílio de depuração: dois símbolos criados com
O mesmo argumento de string ainda é diferente um do outro.
O ponto dos símbolos não é segurança, mas definir uma extensão segura
Mecanismo para objetos JavaScript.Se você receber um objeto de terceiros
código que você não controla e precisa adicionar alguns dos seus próprios
propriedades para esse objeto, mas querem ter certeza de que suas propriedades vão
não conflito com nenhuma propriedade que já possa existir no objeto,
Você pode usar com segurança símbolos como nomes de propriedades.Se você fizer isso, você
também pode estar confiante de que o código de terceiros não será acidentalmente
Altere suas propriedades simbolicamente nomeadas.(Esse código de terceiros poderia,
Obviamente, use o object.getownPropertySymbols () para descobrir
Os símbolos que você está usando e podem alterar ou excluir seu
propriedades.É por isso que os símbolos não são um mecanismo de segurança.)

6.10.4 Operador de espalhamento

No ES2018 e mais tarde, você pode copiar as propriedades de um objeto existente
em um novo objeto usando o "operador de espalhamento" ... dentro de um objeto
literal:

```
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 // => 175
```

Neste 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
objetos literais.(Três pontos são usados ??para outros propósitos em outros
Contextos de javascript, mas literais de objeto são o único contexto em que o
Três pontos causam esse tipo de interpolação de um objeto em outro
um.)
Se o objeto que é espalhado e 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
Trabalhe 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
funcione 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 faria
Defina qualquer outra propriedade de um objeto:

```
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 cólon 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

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
o 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 métod_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.Em
a fim de tornar um objeto iterável (para que possa ser usado com um para/de
loop), você deve definir um método com o nome simbólico
Symbol.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, JavaScript
Invoca o método getter (não transmitindo argumentos).O valor de retorno de
Este método se torna o valor da expressão de acesso à propriedade.
Quando um programa define o valor de uma propriedade acessadora, JavaScript
chama o método do setter, passando o valor do lado direito do
atribuição.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/gravação
propriedade.Se possui apenas um método getter, é uma propriedade somente leitura.E
Se 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 sempre
avaliar como indefinido.
As propriedades do acessador podem ser definidas com uma extensão do objeto
Sintaxe literal (ao contrário das outras extensões ES6 que vimos aqui, Getters
e os setters foram introduzidos no ES5):
Seja o = {
// Uma propriedade de dados comum
DataProp: 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 comuns
definido usando a taquigrafia ES6, exceto esse getter e setter
As 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 do
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
Em §8.2.2.
As propriedades do acessador são herdadas, assim como as propriedades de dados, então você pode
Use o objeto P definido acima como um protótipo para outros pontos.Você pode
Dê aos 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
getters e setters
q.x = 3;Q.Y = 4;// Crie propriedades de dados de Q
q.r // => 5: o acessador herdado
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 escreve 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 aumentá -lo
obtenha a seguir () {return this._n ++;},

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Capítulo 7. Matrizes

Este capítulo documenta as matrizes, um tipo de dados fundamental em javascript
e na maioria das outras linguagens de programação.Uma matriz é uma ordem
Coleção de valores.Cada valor é chamado de elemento e cada elemento
tem uma posição numérica na matriz, conhecida como seu índice.JavaScript
Matrizes não são criadas: um elemento de matriz pode ser de qualquer tipo e diferente
Elementos da mesma matriz podem ser de tipos diferentes.Elementos da matriz
pode até ser objetos ou outras matrizes, o que permite criar
estruturas 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 de
O primeiro elemento é 0 e o índice mais alto possível é 4294967294
(2 ?2), para um tamanho máximo da matriz de 4.294.967.295 elementos.
As matrizes de JavaScript são dinâmicas: elas crescem ou encolhem conforme necessário, e
Não há necessidade de declarar um tamanho fixo para a matriz quando você a cria
ou realocá -lo quando o tamanho mudar.Matrizes de JavaScript podem ser
Esparso: os elementos não precisam ter índices contíguos, e pode haver
ser lacunas.Cada matriz JavaScript possui uma propriedade de comprimento.Para não parse
Matrizes, esta propriedade especifica o número de elementos na matriz.Para
Matrizes esparsas, o comprimento é maior que o índice mais alto de qualquer elemento.
Matrizes de JavaScript são uma forma especializada de objeto JavaScript e Array
Os índices são realmente pouco mais do que nomes de propriedades que são
Inteiros.Falaremos mais sobre as especializações de matrizes em outros lugares
Neste capítulo.As implementações normalmente otimizam as matrizes para que
O acesso a elementos de matriz numericamente indexados é geralmente significativamente

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
Os métodos são genéricos, o que significa que eles funcionam corretamente não apenas para
Matrizes verdadeiras, mas para qualquer "objeto de matriz".Discutiremos como matriz
Objetos 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 ?digitado
Matrizes. ?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 elementos
Seja os primos = [2, 3, 5, 7, 11];// Uma matriz com 5 numérico
elementos
Seja Misc = [1.1, verdadeiro, "A",];// 3 elementos de vários
Tipos + vírgula à direita
Os valores em uma matriz literal não precisam ser constantes;eles podem ser
Expressões 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 valor
Entre, a matriz é escassa (ver §7.3).Elementos de matriz para os quais
Os valores são omitidos não existem, mas parecem indefinidos se você
Consulte -os:
deixe count = [1, 3];// elementos nos índices 0 e 2. Não
Elemento no índice 1
deixe undefs = [,,];// uma matriz sem elementos, mas um comprimento
de 2
A sintaxe literal da matriz permite uma vírgula opcional, então [,,] tem um
comprimento de 2, não 3.
7.1.2 O operador de propagação
No ES6 e mais tarde, você pode usar o "operador de espalhamento" ..., para incluir
Os 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 duplicado
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 ().Você
Pode invocar este construtor de três maneiras distintas:
Chame sem argumentos:
deixe a = new Array ();
Este método cria uma matriz vazia sem elementos e é
equivalente à matriz literal [].
Chame com um único argumento numérico, que especifica um
comprimento:
deixe a = nova matriz (10);
Esta técnica cria uma matriz com o comprimento especificado.Esse
A forma do construtor Array () pode ser usada para pré -alocar
uma matriz quando você souber com antecedência quantos elementos vão
ser necessário.Observe que nenhum valores é armazenado na matriz e o
Propriedades do índice de matriz ?0?, ?1? e assim por diante nem são definidas
para a matriz.
Especificar explicitamente dois ou mais elementos de matriz ou um único não
Elemento 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 invocado
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 matriz
Elementos:
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 um
variedade:
Seja copy = Array.From (original);
Array.from () também é importante porque define uma maneira de fazer um
cópia de matriz verdadeira de um objeto semelhante a uma matriz.Objetos semelhantes a matrizes não são
objetos que têm uma propriedade de comprimento numérico e valores armazenados com
propriedades cujos nomes são inteiros.Ao trabalhar com
JavaScript do lado do cliente, os valores de retorno de alguns métodos do navegador da web
são parecidos com a matriz e pode ser mais fácil trabalhar com eles se você primeiro
converte -os em verdadeiros matrizes:
Deixe Truearray = Array.From (matriz);
Array.From () também aceita um segundo argumento opcional.Se você
Passe uma função como o segundo argumento, então como a nova matriz está sendo
construído, cada elemento do objeto de origem será passado para o
função que você especificar e o valor de retorno da função será armazenado
na matriz em vez do valor original.(Isso é muito parecido com o
Método de mapa de matriz () que será introduzido posteriormente no capítulo, mas
é mais eficiente para realizar o mapeamento enquanto a matriz está sendo construída
do 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 referência
Para a matriz, deve aparecer à esquerda dos colchetes.Um arbitrário
expressão 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 0
a [1] = 3,14;// Escreva o elemento 1
deixe i = 2;
a [i] = 3;// Escreva elemento 2
a [i + 1] = "Olá";// Escreva elemento 3
a [a [i]] = a [0];// Leia os elementos 0 e 2, escreva
```

Elemento 3
O 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 automaticamente
Mantém o valor da propriedade de comprimento para você.No anterior,
Por exemplo, criamos uma matriz A com um único elemento.Nós então
valores atribuídos nos índices 1, 2 e 3. A propriedade de comprimento do
Array mudou como nós, então:

```
A.Length // => 4
```

Lembre -se de que as matrizes são um tipo especializado de objeto.O quadrado
Suportes usados ??para acessar os elementos da matriz funcionam como o quadrado
Suportes 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 o
String "1" - então usa essa string como nome de propriedade.Não há nada
Especial sobre a conversão do índice de um número em uma string:
Você também pode fazer isso com objetos regulares:

```
Seja o = {};// Crie um objeto simples
o [1] = "um";// indexá -lo com um número inteiro
```

o ["1"] // => "One";nomes numéricos e de propriedades de string
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 justo
tão verdadeiro para matrizes quanto para objetos:

```
deixe um = [verdadeiro, falso];// Esta matriz tem elementos nos índices
0 e 1
```

a [2] // => indefinido;nenhum elemento nisso
índice.
a [-1] // => indefinido;Sem propriedade com isso
nome.
7.3 Matrizes esparsas
Uma matriz esparsa é aquela em que os elementos não têm contíguo
índices começando em 0. Normalmente, a propriedade de comprimento de uma matriz
Especifica o número de elementos na matriz.Se a matriz for escassa, o
O valor da propriedade de comprimento é maior que o número de elementos.
Matrizes esparsas podem ser criadas com o construtor Array () ou simplesmente
atribuindo 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 e
comprimento = 0.
a [1000] = 0;// A atribuição adiciona um elemento, mas define
comprimento a 1001.
Veremos mais tarde que você também pode fazer uma matriz escassa com o
Excluir operador.
Matrizes suficientemente escassos são tipicamente implementadas em um
maneira mais lenta e com eficiência de memória do que as densas matrizes, e olhando
elementos para cima em tal matriz levarão tanto tempo quanto regular
Pesquisa de propriedade do objeto.
Observe que quando você omita um valor em uma matriz literal (usando repetidos
vírgulas como em [1 ,, 3]), a matriz resultante é escassa e o omitido
Elementos simplesmente não existem:

Seja A1 = [,];// Esta matriz não tem elementos e
comprimento 1
Seja A2 = [indefinido];// Esta matriz tem um indefinido
elemento
0 em A1 // => false: A1 não tem elemento com
ÍNDICE 0
0 em A2 // => true: A2 tem o indefinido
valor no índice 0
Entender matrizes esparsas é uma parte importante do entendimento do
Natureza verdadeira das matrizes JavaScript.Na prática, no entanto, a maioria dos javascripts
Matrizes com quem você trabalhará não será escasso.E, se você tiver que
Trabalhe com uma matriz esparsa, seu código provavelmente o tratará exatamente como
trataria uma matriz não par com elementos indefinidos.
7.4 Comprimento da matriz
Cada matriz tem uma propriedade de comprimento, e é essa propriedade que faz
Matrizes diferentes dos objetos JavaScript regulares.Para matrizes que são
Denso (isto é, não escasso), a propriedade de comprimento especifica o número de
elementos na matriz.Seu valor é mais do que o maior índice em
a matriz:
[] .Length // => 0: A matriz não tem elementos
["A", "B", "C"]. Comprimento // => 3: O índice mais alto é 2, o comprimento é
3
Quando uma matriz é escassa, a propriedade de comprimento é maior que o
número de elementos, e tudo o que podemos dizer sobre isso é que o comprimento é
Garantido ser maior que o índice de todos os elementos da matriz.Ou,
Em outras palavras, uma matriz (esparsa ou não) nunca terá um elemento
cujo 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 um
Matriz 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.Altera 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 iteração
A partir do ES6, a maneira mais fácil de percorrer cada um dos elementos de um
matriz (ou qualquer objeto iterável) é com o loop for/de
coberto em detalhes em §5.4.4:
Deixe letras = [... "Hello World"];// Uma variedade de letras
deixe string = "";
para (deixe a carta das letras) {
string += letra;
}
String // => "Hello World";Remontizamos o texto original
O iterador de matriz embutido que o loop for/of usa retorna o
elementos de uma matriz em ordem ascendente.Não tem comportamento especial para
matrizes esparsas e simplesmente retorna indefinidas para qualquer elementos de matriz
que não existem.
Se você deseja usar um loop for/de uma matriz e precisa saber o
Índice 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 em
até índices
}
todos os outros // => "hlowrd"
Outra boa maneira de iterar as matrizes é com a foreach ().Isso não é um
nova forma do loop for, mas um método de matriz que oferece um funcional
abordagem para a iteração de matriz.Você passa uma função para o foreach ()
Método de uma matriz, e foreach () invoca sua função uma vez
cada elemento da matriz:
Seja uppercase = "";
letters.foreach (letra => {// Nota Sintaxe da função de seta
aqui
uppercase += letter.ToupPercase ();
});
uppercase // => "Hello World"
Como seria de esperar, foreach () itera a matriz em ordem, e ela
Na verdade, passa o índice de matriz para sua função como um segundo argumento,
o que é ocasionalmente útil.Ao contrário do loop for/of, o
foreach () está ciente das matrizes esparsas e não invoca o seu
função para elementos que não estão lá.
§7.8.1 documenta o método foreach () com mais detalhes.Essa seção
também abrange métodos relacionados, como map () e filtro () que
Realize 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 matriz
Deixe a letra = letras [i];// Obtenha o elemento
Nesse índice
se (//eiou ]/.test(Letter)) {// use um regular
teste de expressão
vogais += letra;// se for um
vogal, lembre -se disso
}
}
vogais // => "EOO"
```

Em loops aninhados, ou outros contextos em que o desempenho é crítico, você
às 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.Ambos
dos seguintes formas de loop são idiomáticas, embora não particularmente
Comum, e com intérpretes javascript modernos, não é de todo claro
que eles têm algum impacto de desempenho:

```
// salve o comprimento da matriz em uma variável local
para (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 início
para (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 elementos
contêm dados válidos.Se não for esse o caso, você deve testar a matriz
elementos 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 o A matriz variável é uma matriz de matrizes de números.Cada elemento em Matrix [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 mesa:

```
// Crie uma matriz multidimensional
deixe tabela = nova matriz (10);// 10 linhas do
mesa
para (vamos i = 0; i <table.length; i ++) {
tabela [i] = nova matriz (10);// Cada linha tem 10
colunas
}
// inicialize a matriz
for (let linha = 0; linha <table.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 matriz

Os 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, o
A função que você passa não é invocada para elementos inexistentes.Na maioria
casos, 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
ignore 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
invocou como se fosse um método deste segundo argumento.Isto é, o segundo
argumento que você passa se torna o valor dessa palavra -chave dentro de
A função que você passa como o primeiro argumento.O valor de retorno do
A função que você passa geralmente é importante, mas métodos diferentes lidam
o valor de retorno de maneiras diferentes.Nenhum dos métodos descritos aqui
modifique a matriz em que eles são invocados (embora a função você
Pass pode 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 do
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) obras
particularmente bem com esses métodos, e nós o usaremos nos exemplos
isso a seguir.

Foreach ()

O método foreach () itera através de uma matriz, invocando um
função que você especifica para cada elemento.Como descrevemos, você passa
a função como o primeiro argumento para foreach ().foreach () então
Invoca sua função com três argumentos: o valor da matriz
elemento, o índice do elemento da matriz e a própria matriz.Se você apenas
se preocupar com o valor do elemento da matriz, você pode escrever uma função com
Apenas um parâmetro - os argumentos adicionais serão ignorados:

```
deixe dados = [1,2,3,4,5], soma = 0;
// calcular a soma dos elementos da matriz
data.foreach (value => {sum += value;});// soma ==
15
// agora incrementa cada elemento da matriz
data.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 iteração
Antes de todos os elementos serem passados ??para a função.Isto é, existe
Nenhum equivalente à declaração de quebra que você pode usar com um regular para
laço.

MAPA()

O método map () passa cada elemento da matriz em que é
invocado para a função que você especificar e retorna uma matriz contendo o
valores 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 mapa () é invocada da mesma maneira que um
função passada para foreach ().Para o método map (), no entanto, o
A função que você passa deve retornar um valor.Observe que o map () retorna um
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 escassa da mesma maneira que a matriz original:
Ele terá o mesmo comprimento e os mesmos elementos ausentes.
FILTRO()
O método filtro () retorna uma matriz contendo um subconjunto do
elementos da matriz em que é invocado.A função que você passa
Deve ser predicado: uma função que retorna verdadeira ou falsa.O
O predicado é chamado como para foreach () e map ().Se o retorno
O valor é verdadeiro, ou um valor que se converte para verdadeiro, então o elemento
passado para o predicado é um membro do subconjunto e é adicionado ao
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
valor
Observe que o filtro () 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 são
até.
O método de alguns () é como o quantificador matemático "existe"
?: ele retorna verdadeiro se houver pelo menos um elemento na matriz para
que o predicado retorna verdadeiro e retorna false se e somente se o
Retornos 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 como
Assim que eles sabem que valor retornar.alguns () retorna verdadeiro o
Primeira vez que seu predicado retorna <code> true </code> e apenas itera
Durante toda a matriz, se o seu predicado sempre retornar FALSE.
todo () é o oposto: retorna falsa na primeira vez que você
Retornos predicados falsos e apenas itera todos os elementos se o seu
O predicado sempre retorna verdadeiro.Observe também que, por matemática
Convenção, todo () retorna verdadeiro e alguns retornam falsos quando
invocado em uma matriz vazia.
Reduce () e ReduceRight ()
Os métodos ReduCer () e ReduceRight () combinam o
elementos de uma matriz, usando a função que você especifica, para produzir um
valor único.Esta é uma operação comum na programação funcional

e também passa pelos nomes "injetar" e "dobrar".Exemplos de ajuda
ilustrar como funciona:
Seja a = [1,2,3,4,5];
a.Reduce ((x, y) => x+y, 0) // => 15;a soma do
valores
a.Reduce ((x, y) => x*y, 1) // => 120;o produto de
os valores
a.Reduce ((x, y) => (x> y)? X: y) // => 5;o maior de
os valores
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 para
retornar esse valor reduzido.Nos exemplos que mostramos aqui, o
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 reduz () são diferentes das funções usadas
com foreach () e map ().O valor familiar, índice e matriz
Os valores são passados ??como os segundo, terceiro e quarto argumentos.O primeiro
O argumento é o resultado acumulado da redução até agora.No primeiro
Chamada para a função, este primeiro argumento é o valor inicial que você passou como
o segundo argumento a reduzir ().Nas chamadas subsequentes, é o
valor retornado pela invocação anterior da função.No primeiro
Por exemplo, a função de redução é chamada primeiro com os argumentos 0 e 1.
Ele adiciona e retorna 1. É então chamado novamente com os argumentos 1 e
2 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
reduzir().

Você deve ter notado que a terceira chamada para reduzir () neste exemplo
Tem apenas um argumento: não há valor inicial especificado.Quando
você chama Reduce () como este sem valor inicial, ele usa o primeiro
elemento da matriz como o valor inicial.Isso significa que a primeira chamada para
a função de redução terá o primeiro e o segundo elementos de matriz como
seus primeiros e segundos argumentos.Nos exemplos de soma e produto, nós
poderia ter omitido o argumento do valor inicial.
Chamando Reduce () em uma matriz vazia sem argumento de valor inicial
Causa um TypeError.Se você ligar com apenas um valor - uma matriz
com um elemento e nenhum valor inicial ou uma matriz vazia e um inicial
valor - simplesmente retorna esse valor sem nunca chamar o
função de redução.
ReduceRight () funciona como Reduce (), exceto que
processa a matriz do índice mais alto para o mais baixo (da direita para a esquerda), mas sim
do que do mais baixo ao mais alto.Você pode querer fazer isso se a redução
A operação tem associatividade da direita para a esquerda, por exemplo:
// calcular 2^(3^4).A exponenciação tem o direito para a esquerda
precedência
Seja a = [2, 3, 4];
A. Reduteright ((ACC, Val) => Math.pow (val, acc)) // =>
2.4178516392292583E+24
Observe que nem reduz () nem reduteright () aceita um
argumento opcional que especifica esse valor em que o
A função de redução deve ser invocada.O argumento de valor inicial opcional
toma seu lugar.Consulte o método function.bind () (§8.7.5) se você
Precisa 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 reduteright () 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 um
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
elementos 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 achatada 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 achata para nada no
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
Concatenado, não a própria matriz.Observe, no entanto, que concat () faz
Não achatar 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

achatado
A.Concat (4, [5, [6,7]]) // => [1,2,3,4,5, [6,7]];mas não
matrizes aninhadas
a // => [1,2,3];A matriz original é
não modificado
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
Eles 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 achate os argumentos da matriz.O
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
Implemente 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 achate uma matriz que você passa para ele, mas se você
quero 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
a matriz, mudando todos os elementos subseqüentes para baixo para ocupar
O espaço recém -vago no início da matriz.Você poderia usar
não definido () e shift () para implementar uma pilha, mas seria menos
eficiente do que usar push () e pop () porque os elementos da matriz
precisa ser deslocado 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 um
estrutura de dados da fila usando push () para adicionar elementos no final de um
Array 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 terminam na matriz em uma ordem diferente da que seria se você inserisse eles 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 () e CopyWithin ()

Matrizes definem vários métodos que funcionam em regiões contíguas, ou subarrays ou "fatias" de uma matriz.As seções a seguir descrevem Métodos para extrair, substituir, preencher e copiar fatias.

FATIAR()

O método slice () retorna uma fatia, ou subarray, do especificado variedade.Seus dois argumentos especificam o início e o final da fatia para ser retornou.A matriz retornada contém o elemento especificado pelo primeiro argumento e todos os elementos subsequentes até, mas não incluindo, o elemento especificado pelo segundo argumento.Se apenas um argumento for Especificado, a matriz retornada contém todos os elementos desde o início posição até o final da matriz.Se um argumento for negativo, é Especifica um elemento de matriz em relação ao comprimento da matriz.Um argumento de ?1, por exemplo, especifica o último elemento na matriz e Um argumento de ?2 especifica o elemento antes disso.Observe que Slice () não modifica a matriz em que é invocada.Aqui estão Alguns 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 remover
elementos de uma matriz.Ao contrário do slice () e concat (),
Splice () modifica a matriz em que é invocada.Observe que
Splice () e Slice () têm nomes muito semelhantes, mas executam
operações substancialmente diferentes.
Splice () pode excluir elementos de uma matriz, inserir novos elementos
em uma matriz ou execute as duas operações ao mesmo tempo.Elementos de
a matriz que vem após o ponto de inserção ou exclusão tem seu
os índices aumentaram ou diminuíram conforme necessário para que permaneçam
contíguo com o restante da matriz.O primeiro argumento a Splice ()
especifica a posição da matriz na qual a inserção e/ou exclusão é para
começar.O segundo argumento especifica o número de elementos que
deve ser excluído de (emendado) da matriz.(Observe que isso é
Outra diferença entre esses dois métodos.O segundo argumento
Para Slice () é uma posição final.O segundo argumento para Splice () é
um comprimento.) Se este segundo argumento for omitido, todos os elementos da matriz de
O elemento de partida para o final da matriz é removido.emenda ()
Retorna uma matriz dos elementos excluídos, ou uma matriz vazia se não
Os 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.

ÍNDIC
preenchido.Se esse argumento for omitido, a matriz será preenchida desde o início
índice até o fim.Você pode especificar índices em relação ao final do
Array passando números negativos, assim como você pode para Slice ().
CopyWithin ()
copyWithin () copia uma fatia de uma matriz para uma nova posição dentro
a matriz.Ele modifica a matriz no lugar e retorna a matriz modificada,
Mas não mudará o comprimento da matriz.O primeiro argumento
Especifica o índice de destino para o qual o primeiro elemento será
copiado.O segundo argumento especifica o índice do primeiro elemento para
ser copiado.Se esse segundo argumento for omitido, 0 será usado.O terceiro
O argumento especifica o fim da fatia dos elementos a serem copiados.Se
omitido, o comprimento da matriz é usado.Elementos do índice de início
Até, entre outros, o índice final será copiado.Você pode especificar
índices 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 matriz
up um
a.Copywithin (2, 3, 5) // => [1,1,3,4,4]: Copie os últimos 2 elementos
para indexado 2
a.Copywithin (0, -2) // => [4,4,3,4,4]: compensações negativas
trabalho também
copyWithin () é destinado a um método de alto desempenho que é
particularmente útil com matrizes digitadas (consulte §11.2).É modelado após o
função memmove () da biblioteca padrão C.Observe que a cópia
funcionará corretamente, mesmo que haja sobreposição entre a fonte e

regiões de destino.

7.8.6 Métodos de pesquisa e classificação de matrizes

Matrizes implementar indexof (), lastIndexOf () e
inclui () métodos semelhantes aos métodos de mesmo nome de
cordas.Também existem métodos de classificação () e reverso () para
reordenando os elementos de uma matriz.Esses métodos são descritos no
subseções 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 tipo
encontrado, ou -1 se nenhum for encontrado.indexOf () pesquisa a matriz de
Começando ao fim, e LastIndexOf () pesquisas de ponta a
começo:

```
Seja a = [0,1,2,1,0];
A.IndexOF (1) // => 1: a [1] é 1
A.LastIndexOf (1) // => 3: a [3] é 1
A.IndexOF (3) // => -1: nenhum elemento tem valor 3
```

indexOf () e lastIndexOf () comparam seu argumento com o
elementos de matriz usando o equivalente ao operador ===.Se sua matriz
contém objetos em vez de valores primitivos, esses métodos verificam para ver
Se duas referências se referem exatamente ao mesmo objeto.Se você quiser
Na 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
o primeiro.

```
// encontra todas as ocorrências de um valor x em uma matriz A e retorne
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
de
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
variedade
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 negativo
O argumento é tratado como zero.
Inclui ()
O ES2016 inclui () o método leva um único argumento e
Retorna true se a matriz contiver esse valor ou false caso contrário.Isto
Não informa o índice do valor, apenas se ele existe.O
Inclui () 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 usar
um objeto definido real (§11.1.1).
O método inclui () é ligeiramente diferente do indexOF ()
método de uma maneira importante.indexOf () testa a igualdade usando o
mesmo algoritmo que o operador ===
considera o valor não-número de um número diferente de todos os outros
valor, incluindo a si mesmo.inclui () usa uma versão ligeiramente diferente
de igualdade que considera Nan igual a si mesmo.Isso significa isso
indexOf () não detectará o valor da nan em uma matriz, mas
inclui () Will:
Seja a = [1, verdadeiro, 3, nan];
a.includes (verdadeiro) // => true
a.includes (2) // => false
a.includes (nan) // => true
A.IndexOF (NAN) // => -1;Indexof não consegue encontrar nan
ORGANIZAR()
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 strings
Para 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, o
A função de comparação deve retornar um número menor que zero.Se o primeiro
o argumento deve aparecer após o segundo na matriz classificada, o
A função deve retornar um número maior que zero.E se os dois
Os 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.sort (função (a, b) {// passa uma função comparadora
retornar a-b;// retorna <0, 0 ou> 0, dependendo
em ordem
});// a == [4, 33, 222, 1111];numérico
ordem
a.sort ((a, b) => b-a);// a == [1111, 222, 33, 4];reverter
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 caso
organizar
REVERTER()
O método reverso () reverte a ordem dos elementos de um
matriz e retorna a matriz invertida.Faz isso no lugar;em outro
palavras, ele não cria uma nova matriz com os elementos reorganizados, mas
Em 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 string
A classe da matriz define três métodos que podem converter matrizes para
Strings, que geralmente é algo que você pode fazer ao criar log
e mensagens de erro.(Se você deseja salvar o conteúdo de uma matriz em
forma textual para reutilização posterior, serialize a matriz com
Json.stringify () [§6.8] em vez de usar os métodos descritos
aqui.)

O método junção () converte todos os elementos de uma matriz em strings
e concatena -os, retornando a string resultante.Você pode especificar
uma sequência opcional que separa os elementos na sequência resultante.Se
Nenhuma sequência de separador é especificada, uma vírgula é usada:
Seja 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 não
elementos
B.Join ("-") // => "---------": uma sequência de 9
hífens
O 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 ().Para
Uma matriz, esse método funciona como o método junção () sem
Argumentos:
[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 tipo
de delimitador em torno do valor da matriz.
tolocalestring () é a versão localizada do tostring ().Isto
converte cada elemento da matriz em uma string chamando o
TOLOCALESTRING () Método do elemento, e então
concatena as seqüências resultantes usando um local específico (e

String de separador definida por implementação).

7.8.8 Funções de matriz estática

Além dos métodos de matriz que já documentamos, a matriz
A classe também define três funções estáticas que você pode invocar através do
Construtor de matrizes em vez de matrizes.Array.of () e
Array.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 ([]) // => true
Array.isarray ({}) // => false
```

7.9 Objetos semelhantes a matriz

Como vimos, as matrizes de javascript têm alguns recursos especiais que outros
objetos não têm:
A propriedade de comprimento é atualizada automaticamente como nova
Os elementos são adicionados à lista.
A configuração do comprimento para um valor menor trunca a matriz.
Matrizes herdam métodos úteis do Array.prototype.
Array.esArray () 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 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 é somente leitura ou se eles pelo menos deixam o comprimento da matriz inalterado.
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
são
// 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 adicional! == 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 matriz
```

A segunda linha deste código chama a fatia de matriz () método em um objeto semelhante a uma matriz para copiar os elementos disso Objeto 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 matrizes

Strings JavaScript se comportam como matrizes somente leitura do UTF-16 Unicode caracteres.Em vez de acessar caracteres individuais com o Método 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.esArray () retorna false se você passar um
corda.
O principal benefício das seqüências indexíveis é simplesmente que podemos substituir
chamadas para charat () com colchetes, que são mais concisos e
legível 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 a
S 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 (), classin (), 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 esotérico
Detalhes sobre matrizes esparsas e objetos semelhantes a matrizes.Os principais pontos para
Tire deste capítulo é:
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. Funções

Este capítulo abrange as funções JavaScript.Funções são fundamentais
Bloco de construção para programas JavaScript e um recurso comum em
Quase todas as linguagens de programação.Você já pode estar familiarizado com
o 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
parametrizado: uma definição de função pode incluir uma lista de identificadores,
conhecidos como parâmetros, que funcionam como variáveis ??locais para o corpo do
função.As invocações de função fornecem valores ou argumentos para o
parâmetros da função.Funções geralmente usam seus valores de argumento para
Calcule 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 do
essa palavra -chave.

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
função.Funções projetadas para inicializar um objeto recém -criado são
construtores chamados.Os construtores foram descritos em §6.2 e serão
coberto 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.função* define gerador
funções (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 um
variável, e o objeto de função recém -definido é atribuído a
a variável.
Um par de parênteses em torno de uma lista se 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
executado 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 calcula
fatoriais
// Lembre -se que x!é o produto de x e tudo positivo
Inteiros 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 função é esse o nome da função se torna uma variável cujo valor é o função em si.As declarações de declaração de função são "içadas" ao topo do script, função ou bloqueio da anexo para que funções definidas em Dessa 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 dos o código nesse bloco.

As funções distance () e fatorial () que descrevemos são projetado 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 pare executando 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
invocado:
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 omita.Uma declaração de função realmente declara uma variável e
Atribui um objeto de função a ele.Uma expressão de função, por outro
mão, não declara uma variável: cabe a você atribuir o recém
objeto de função definida para uma constante ou variável se você estiver indo para
Precisa consultar várias vezes.É uma boa prática usar const
com expressões de função para que você não substitua acidentalmente seu
funções atribuindo novos valores.
Um nome é permitido para funções, como a função fatorial, que precisam
consulte a si mesmos.Se uma expressão de função incluir um nome, o local
o escopo da função para essa função incluirá uma ligação desse nome a
o 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, o
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
existe até que a expressão que os defina seja realmente avaliada.
Além disso, para invocar uma função, você deve ser capaz de se referir a
e você não pode se referir a uma função definida como uma expressão até que seja
atribuído a uma variável, portanto as funções definidas com expressões não podem ser
invocados antes de serem definidos.

8.1.3 Funções de seta

No ES6, você pode definir funções usando uma sintaxe particularmente compacta
conhecido como "funções de seta".Esta sintaxe é uma reminiscência de
notação matemática e usa um => "seta" para separar a função
parâmetros do corpo da função.A palavra -chave da função não é
usado, 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 um
A função de seta é uma lista de parâmetros separados por vírgula entre parênteses,
seguido pela seta => seguida pelo corpo da função em Curly
aparelho ortodôntico:

```
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
aparelho e escreva o corpo da função como a expressão cuja
o 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 variável Regras: eles podem acessar os parâmetros e variáveis ??da função (ou funções) eles são aninhados dentro.No código mostrado aqui, por exemplo, A função interna Square () pode ler e escrever os parâmetros a e b definido pela função externa hipotenuse ().Essas regras de escopo pois as funções aninhadas são muito importantes, e nós as consideraremos novamente em §8.6.

8.2 Funções de invocação

O código JavaScript que compõe o corpo de uma função não é executado quando a função é definida, mas sim quando é invocada. As funções JavaScript podem ser invocadas de cinco maneiras:

Como funções
Como métodos
Como construtores
Indiretamente através de seus métodos de chamada () e Aplicação ()
Implicitamente, via recursos de linguagem JavaScript que não aparecem como invocações de função normal

8.2.1 Invocação da função

As funções são invocadas como funções ou métodos com uma invocação expressão (§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 argumento
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 o
argumentos para a função.Esses valores são atribuídos aos parâmetros
nomeado na definição da função.No corpo da função, um
A 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 um
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 (o
este valor) é o objeto global.No modo rigoroso, no entanto, o
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
modo rigoroso.
const strict = (function () {return! this;} ());
Funções recursivas e a pilha
Uma função recursiva é uma, como a função fatorial () 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
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
Esses 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 um propriedade de um objeto.Se você tem uma função f e um objeto o, você pode Defina 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 um Expressão da função O.M e duas expressões de argumento, x e y.O A expressão da função é em si uma expressão de acesso à propriedade, e este significa que a função é invocada como um método e não como um regular função.

Os argumentos e o valor de retorno de uma invocação de método são tratados Exatamente como descrito para invocação regular de função.Método as invocações diferem das invocações de funções de uma maneira importante, No entanto: o contexto de invocação.Expressões de acesso à propriedade consistem em Duas partes: um objeto (neste caso O) e um nome de propriedade (M).Em um Expressão de vocação de métodos como esta, o objeto O se torna o contexto de invocação, e o corpo da função pode se referir a esse objeto por usando 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
esta 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
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
f (). M ();// Invocar o método m () em
Valor de retorno de f ()
Métodos e essa palavra-chave são centrais para os objetos orientados a objetos
paradigma 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 nessa
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
execute 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 do
operação.
Encadeamento de método
Quando os métodos retornam objetos, você pode usar o valor de retorno de uma invocação de método como parte de um
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), para
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
invocado 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, seu

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 definido em um método e invocado como uma função pode usar isso para Obtenha 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 o
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
precisa ser movido dentro do método m para que ele apareça antes que seja
invocado.
Outra solução alternativa é invocar o método bind () do aninhado
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 nova,
Então é uma invocação do construtor.(As invocações construtivas foram
Introduzido em §4.6 e §6.2.2, e os construtores serão cobertos em mais
Detalhes no Capítulo 9.) As invocações do construtor diferem de regular
invocações de função e método no manuseio 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 um
Invocação do construtor.As duas linhas a seguir, por exemplo, são
equivalente:
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
Objeto criado é usado como contexto de invocação, então o construtor
A função pode se referir a ela com a palavra -chave esta.Observe que o novo
O objeto é usado como contexto de invocação, mesmo que o construtor
A 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 inicialize o novo objeto e depois retorne implicitamente quando eles
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 indireta

As funções JavaScript são objetos e, como todos os objetos JavaScript, eles tem métodos.Dois desses métodos, Call () e Apply (), Invoke a 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 implícita

Existem vários recursos de linguagem JavaScript que não parecem Invocações da função, mas que fazem com que as funções sejam invocadas.Ser extra cuidadoso 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 eles estão 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, lá
são 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
tipos iteráveis.
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 o
parâmetros de função e invocações de função não fazem nenhum tipo
Verificando os 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
invocado 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
variedade
para (deixe a propriedade em O) a.push (propriedade);
retornar a;
}
// getPropertyNames () pode ser invocado com um ou dois
Argumentos:
Seja o = {x: 1}, p = {y: 2, z: 3};// dois objetos para teste
deixe a = getPropertyNames (o);// a == ["x"];Obtenha O's
propriedades em uma nova matriz
getPropertyNames (P, A);// a == ["x", "y", "z"];Adicione P's
propriedades para isso
```

Em 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 se
Esse argumento é verdade e retorna seu segundo argumento.Em
Este caso, se algum objeto for passado como o segundo argumento, a função
usará esse objeto.Mas se o segundo argumento for omitido (ou nulo ou
Outro 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
argumentos) 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
vazio, 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
Os 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 repouso foram introduzidos no JavaScript no ES6.Antes disso
Versão do idioma, as funções varargs foram escritas usando o
Objeto de argumentos: dentro do corpo de qualquer função, o identificador
Argumentos refere -se ao objeto de argumentos para essa invocação.O
O objeto de argumentos é um objeto semelhante a uma matriz (ver §7.9) que permite o
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 e
Lembrando, o maior.
para (vamos i = 0; i <argumentos.Length; i ++) {
if (argumentos [i]> maxvalue) maxvalue = argumentos [i];
}
// retorna o maior
retornar maxvalue;
}
max (1, 10, 100, 2, 3, 1000, 4, 5, 6) // => 1000
O objeto de argumentos remonta aos primeiros dias do JavaScript e
carrega consigo alguma bagagem histórica estranha que a torna ineficiente
e difícil de otimizar, especialmente fora do modo rigoroso.Você ainda pode
Encontre código que usa o objeto de argumentos, mas você deve evitar
Usando -o em qualquer novo código que você escreve.Ao refatorar o código antigo, se você
encontrar uma função que use argumentos, você pode substituí -lo
com um ... parâmetro de repouso args.Parte do infeliz legado do
O objeto de argumentos é que, no modo rigoroso, os argumentos são tratados como um
palavra reservada, e você não pode declarar um parâmetro de função ou um local
variável com esse nome.
8.3.4 O operador de propagação para chamadas de função
O operador de spread ... é usado para descompactar ou ''espalhar'', o
elementos de uma matriz (ou qualquer outro objeto iterável, como seqüências) em um
contexto em que os valores individuais são esperados.Nós vimos a propagação
Operador usado com literais de matriz em §7.1.2.O operador pode ser usado, em
Da 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 dois
Argumentos vetoriais em parâmetros mais claramente nomeados:
função vetoradd ([x1, y1], [x2, y2]) {// descompactar 2 argumentos
em 4 parâmetros
retornar [x1 + x2, y1 + y2];
}
Vectoradd ([1,2], [3,4]) // => [4,6]
Da mesma forma, se você está definindo uma função que espera um objeto
argumento, você pode destruir parâmetros desse objeto.Vamos usar um
Exemplo de vetor novamente, exceto desta vez, vamos supor que representamos
vetores como objetos com parâmetros x e y:
// multiplique o vetor {x, y} por um valor escalar
função 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 dois
Parâmetros é bastante claro porque os nomes de parâmetros que usamos
Combine os nomes de propriedades do objeto de entrada.A sintaxe é mais
Verboso e mais confuso quando você precisa destruir as propriedades
com um nome em parâmetros com nomes diferentes.Aqui está o vetor
Exemplo de adição, implementado para vetores baseados em objetos:
função vetoradd (
{x: x1, y: y1}, // descompacte o 1º objeto em x1 e y1
params
{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 espera
vá em um objeto literal.Portanto, os nomes de propriedades estão sempre à mão esquerda
lado do cólon, e os nomes de parâmetros (ou variáveis) estão no
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 escalar
função 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 um
função com argumentos especificados em nome = formulário de valor, que é
conveniente quando há muitos argumentos opcionais ou quando o
A lista de parâmetros é longa o suficiente para que seja difícil lembrar o correto
ordem.JavaScript não permite isso diretamente, mas você pode se aproximar
Destruir 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
ter 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,
de Index = 0, toIndex = 0}) {
Deixe valuestocópia = de.slice (FromIndex, FromIndex + N);
to.splice (toIndex, 0, ... valuestocópia);
retornar a;
}
Seja a = [1,2,3,4,5], b = [9,8,7,6,5];
Arraycopy ({de: a, n: 3, para: b, toIndex: 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
elementos restantes
// são armazenados na matriz Coords.E quaisquer argumentos depois
a primeira matriz
// são embalados na matriz REST.
função f ([x, y, ... coords], ... descanso) {
retornar [x+y, ... descansar, ... 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 tipo
A verificação é executada nos valores que você passa para uma função.Você pode
Ajude a tornar seu código auto-documentação escolhendo nomes descritivos
para argumentos de função e documentá -los cuidadosamente no
Comentários para cada função.(Alternativamente, consulte §17.8 para um idioma
Extensão que permite que você verifique o tipo de verificação em cima de
Javascript.)
Conforme descrito no §3.9, o JavaScript realiza a conversão do tipo liberal como
necessário.Então, se você escrever uma função que espera um argumento de string e
Então chame essa função com um valor de algum outro tipo, o valor que você
Passado será simplesmente convertido em uma string quando a função tentar para
use -o como uma string.Todos os tipos primitivos podem ser convertidos em cordas, e tudo
Objetos possuem métodos tostring () (se não necessariamente úteis),
Portanto, um erro nunca ocorre neste caso.
Isso nem sempre é verdade, no entanto.Considere novamente o Arraycopy ()
Método mostrado anteriormente.Espera um ou dois argumentos de matriz e irão
Falha se esses argumentos forem do tipo errado.A menos que você esteja escrevendo um
função 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 quando
passou valores ruins do que começar a executar e falhar mais tarde com um erro
mensagem que provavelmente não é clara.Aqui está um exemplo de função que
Executa 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 (tipoof 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 invocou.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 Eles podem ser atribuídos a variáveis, armazenadas nas propriedades dos objetos ou Os elementos das matrizes passaram como argumentos para as funções e assim por diante. Para entender como as funções podem ser dados de JavaScript, bem como Sintaxe JavaScript, 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.O A função pode ser atribuída a outra variável e ainda funciona da mesma forma caminho:

Erro ao traduzir esta página.

§7.8.6.
Exemplo 8-1 demonstra os tipos de coisas que podem ser feitas quando
As funções são usadas como valores.Este exemplo pode ser um pouco complicado, mas
Os comentários explicam o que está acontecendo.
Exemplo 8-1.Usando funções como dados

```
// Definimos algumas funções simples aqui
função 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 operandos
função 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 simples
funciona 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 predefinidas
também
};
// Esta função leva o nome de um operador, olha para cima que
operador
// 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) // => 100
8.4.1 Definindo suas próprias propriedades de função
Funções não são valores primitivos em JavaScript, mas um tipo especializado
de objeto, o que significa que as funções podem ter propriedades.Quando a
função precisa de uma variável "estática" cujo valor persiste
invocações, muitas vezes é conveniente usar uma propriedade da função
em si.Por exemplo, suponha que você queira 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
Acompanhe os valores que ele já retornou e esta informação
deve persistir entre invocações de função.Você poderia armazenar isso
informações em uma variável global, mas isso é desnecessário, porque o
As informações são usadas apenas pela própria função.É melhor armazenar o
informações em uma propriedade do objeto de função.Aqui está um exemplo
Isso retorna um número inteiro único sempre que é chamado:
// Inicialize a propriedade Contador do objeto de função.
// declarações de função são içadas 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 para
ser devolvido.
function ÚnicoInteger () {
retornar exclusivoInteger.counter ++;// retornar e incremento
contra -propriedade
}
Únicanteger () // => 0
ÚnicoInteger () // => 1
```

Como outro exemplo, considere a seguinte função fatorial ()
que usa propriedades de si mesma (tratando -se como uma matriz) para armazenar em cache
Resultados previamente calculados:

```
// calcular fatoriais e resultados de cache como propriedades do
função em si.
função fatorial (n) {
if (number.isInteger (n) && n> 0) {// positivo
Apenas números inteiros
if (! (n em fatorial)) {// se não
resultado em cache
fatorial [n] = n * fatorial (n-1);// calcular
e cache
}
retornar fatorial [n];// Retornar
o resultado em cache
} outro {
Nan de retorno;// se entrada
foi ruim
}
}
fatorial [1] = 1;// inicialize o cache para manter esta base
caso.
Fatorial (6) // => 720
fatorial [5] // => 120;a chamada acima do armazenamento em cache este valor
```

8.5 Funções como namespaces

Variáveis ??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
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
Ele cria entrará em conflito com variáveis ??criadas pelos programas que usam
isto.A solução é colocar o pedaço de código em uma função e depois
Invoque a função.Dessa forma, variáveis ??que teriam sido globais
Torne -se local para a função:

```
função chunkNamespace () {
// pedaço de código vai aqui
// Quaisquer variáveis ??definidas no pedaço são locais para isso
função
// em vez de atrapalhar o espaço para nome global.
}
ChunkNamespace ();// mas não se esqueça de invocar o
função!
```

Este código define apenas uma única variável global: o nome da função
ChunkNamespace.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 um
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
o código pode redefinir o contador ou defini -lo para um não -inteiro, causando o
Função exclusiva () para violar o "único" ou o "número inteiro"
parte de seu contrato.Os fechamentos capturam as variáveis ??locais de um único
Invocação da função e pode usar essas variáveis ??como estado privado.Aqui está
Como poderíamos reescrever o exclusivo () usando um imediatamente
Expressão da função invocada para definir um espaço para nome e um fechamento que
Usos esse espaço para nome para manter seu estado privado:

```
Deixe exclusivo
deixe o contador = 0;// estado privado de
função abaixo
Return function () {retorna contador ++;};
} ());
Únicanteger () // => 0
ÚnicoInteger () // => 1
```

Para entender esse código, você deve lê -lo com cuidado.Inicialmente
Olhar, 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 a
Único.Agora, se estudarmos o corpo da função, vemos
que seu valor de retorno é outra função.É esse objeto de função aninhado
Isso 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 outro
O código pode ver a variável contador: a função interna tem exclusivo
acesso a ele.

Variáveis ??privadas como o contador não precisam ser exclusivas de um único
Fechamento: é perfeitamente possível para duas ou mais funções aninhadas serem
definido 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 contadores
c.count () // => 0
D.Count () // => 0: eles contam
independentemente
c.reset ();// reset () e count ()
Métodos compartilham estado
c.count () // => 0: porque redefinimos
c
D.Count () // => 1: D não foi redefinido
```

A função do contador () retorna um objeto "contador".Este objeto tem
Dois métodos: count () retorna o próximo número inteiro e redefinir () redefinir
o estado interno.A primeira coisa a entender é que os dois métodos
Compartilhe o acesso à variável privada n.A segunda coisa a entender
é que cada invocação de contador () cria um novo escopo -
independente dos escopos usados ??por invocações anteriores - e um novo
Variável 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 no
outro.

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 GetName e SetName ()
// Verifique se apenas os valores da string são permitidos
addPrivateProperty (O, "Nome", X => TIPOF x === "String");
o.setName ("Frank");// Defina o valor da propriedade
o.getName () // => "Frank"
o.setName (0);//! TypeError: tente definir um valor de
o tipo errado
```

Agora vimos vários exemplos em que dois fechamentos são definido no mesmo escopo e compartilhe acesso à mesma variável privada ou variáveis.Esta é uma técnica importante, mas é igualmente importante para reconhecer quando os fechamentos compartilham inadvertidamente o acesso a uma variável que Eles não devem compartilhar.Considere o seguinte código:

```
// Esta função retorna uma função que sempre retorna V
função 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] () // => 5
```

Ao trabalhar com código como este que cria vários fechamentos usando um Loop, 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, Arrow
funções 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 valor
função, 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
O operador do tipo de retorna a string "função" quando aplicado a um
função, mas as funções são realmente um tipo especializado de javascript
objeto.Como as funções são objetos, eles podem ter propriedades e
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 do
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 o
fins desta propriedade de comprimento.

8.7.2 A propriedade Nome

A 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 nome
A expressão foi atribuída a 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
discutido no §6.2.3 e será coberto novamente no capítulo 9.

8.7.4 Os métodos Call () e Apply ()

Call () e Apply () permita 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 é
ser invocado;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 um
Função de seta, o primeiro argumento é efetivamente ignorado.
Quaisquer argumentos a serem chamados () após o primeiro argumento de contexto de invocação
são os valores que são passados ??para a função que é invocada (e estes
Argumentos 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 o
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 o maior = math.max.apply (matemática, Arrayofnumbers);
A função Trace () definida a seguir é semelhante ao
função 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 o
mesmos argumentos e o mesmo esse 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);// Registro
mensagem.
deixe o resultado = original.Apply (isto, args);//
Invoque o original.
console.log (new Date (), "saindo:", m);// Registro
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 um
Objeto O, o método retorna uma nova função.Invocando a nova função
(em função) Invoca a função original f como um método de O.Qualquer
Argumentos que você passa para a nova função são passados ??para o original
função.Por exemplo:

```
função f (y) {return this.x + y;} // Esta função precisa
para ser amarrado
Seja o = {x: 1};// um objeto que vamos vincular
para
Seja g = f.bind (o);// chamando g (x) chama
f () em o
g (2) // => 3
Seja p = {x: 10, g};// invocar G () como um
Método deste objeto
p.g (2) // => 3: g ainda é
ligado a O, não p.
```

Funções de seta herdam seu valor a partir do ambiente em
que eles são definidos, e esse valor não pode ser substituído com
bind (), portanto, se a função f () no código anterior foi definida como
Uma função de seta, a ligação não funcionaria.O uso mais comum
Caso para chamadas bind () é fazer com que as funções não se comportem como
Funções 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 você
Passe 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 equivalente
Para uma função definida com a sintaxe familiar:
const f = função (x, y) {return x*y;};
O construtor function () espera qualquer número de string
argumentos.O último argumento é o texto do corpo da função;pode
conter declarações arbitrárias de javascript, separadas uma da outra
Semicolons.Todos os outros argumentos para o construtor são strings que
Especifique os nomes dos parâmetros para a função.Se você está definindo um
função 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 argumento
Isso 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 JavaScript
ser criado dinamicamente e compilado em tempo de execução.
O construtor da função () analisa o corpo da função e
cria um novo objeto de função cada vez que é chamado.Se a chamada
para o construtor aparece dentro de um loop ou dentro de um frequentemente
Função chamada, esse processo pode ser ineficiente.Por outro lado,

Erro ao traduzir esta página.

eles mesmos particularmente bem para um estilo de programação funcional.O
Seções a seguir demonstram técnicas para funcional
Programação em JavaScript.Eles são destinados a uma expansão da mente
Exploração do poder das funções do JavaScript, não como uma receita
Para um bom estilo de programação.

8.8.1 Matrizes de processamento com funções

Suponha que tenhamos uma variedade de números e queremos calcular o
Desvio médio e padrão desses valores.Podemos fazer isso em
Estilo 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 elementos
Deixe total = 0;
para (vamos i = 0; i <data.length; i ++) total+= dados [i];
deixe mean = total/data.length;// média == 3;A média do nosso
Os 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 ==
2
```

Podemos executar esses mesmos cálculos em estilo funcional conciso
usando os métodos de matriz map () e reduz () como este (consulte §7.8.1 para
Revise esses métodos):

```
// Primeiro, defina duas funções simples
const sum = (x, y) => x+y;
const square = x => x*x;
// então use essas funções com métodos de matriz para calcular
Média e Stddev
deixe dados = [1,1,3,5,5];
deixe mean = data.reduce (sum) /data.length;// média == 3
Deixe os desvios = data.map (x => x-mean);
Seja stddev =
Math.sqrt (desvioss.map (quadrado) .Reduce (soma)/(data.length-
1));
stddev // => 2
```

Esta nova versão do código parece bem diferente do primeiro, mas ainda está chamando métodos em objetos, por isso tem algum objeto Convenções orientadas restantes.Vamos escrever versões funcionais do MAP () e Reduce () Métodos:

```
const map = function (a, ... args) {return a.map (... args);};
const reduz = função (a, ... args) {return
A.Reduce (... args);};
```

Com estes MAP () e Reduce () funções definidas, nosso código para Calcule 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 // => 2
```

8.8.2 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
função
Let 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 mesmo
const 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 estranha
```

Esta 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 e G, 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 retorno desligado.
// ambos F e G são invocados com o mesmo valor que H foi invocado com.
função compõe (f, g) {
Função de retorno (... args) {
// usamos Call para F porque estamos passando um único valor e
// solicita -se para G porque estamos passando uma variedade de valores.
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 soma
```

As funções parciais () e memorize () definidas nas seções

A seguir, são duas funções mais importantes de ordem superior.

8.8.3 Aplicação parcial de funções

O método bind () de uma função f (ver §8.7.5) retorna um novo função que chama F em um contexto especificado e com um conjunto especificado

de argumentos.Dizemos que isso liga a função a um objeto e
aplica parcialmente os argumentos.O método bind () aplica parcialmente
Argumentos à esquerda - isto é, os argumentos que você passa para vincular () são
colocado no início da lista de argumentos que é passada para o original
função.Mas também é possível aplicar parcialmente argumentos no
certo:

```
// Os argumentos para esta função são passados ??à esquerda
função parcialleft (f, ... Outterargs) {
Retornar função (... innerargs) {// retorna esta função
Seja args = [... Outterargs, ... innerargs];// construa o
Lista de argumentos
retornar F.Apply (isto, args);// Então
Invoque F com isso
};
}
// Os argumentos para esta função são passados ??à direita
function parcialright (f, ... Outterargs) {
Retornar função (... innerargs) {// retorna esta função
Seja args = [... innerargs, ... Outterargs];// construa o
Lista de argumentos
retornar F.Apply (isto, args);// Então
Invoque F com isso
};
}
// Os argumentos a essa função servem como um modelo.
Valores indefinidos
// Na lista de argumentos, são preenchidos com valores do
Conjunto interno.
função parcial (f, ... Outterargs) {
Função de retorno (... Innerargs) {
Seja args = [... Outterargs];// cópia local da externa
Modelo de args
Deixe InnerIndex = 0;// que arg interno é o próximo
// percorre os args, preenchendo valores indefinidos
do args interno
para (vamos i = 0; i <args.length; i ++) {
```

Erro ao traduzir esta página.

Também podemos usar composição e aplicação parcial para refazer nossa média
e cálculos de desvio padrão em estilo funcional extremo:

```
// As funções Sum () e Square () são definidas acima.Aqui estão
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 dados
Deixe 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 interessante
Para ver como o código JavaScript profundamente funcional pode ser.

8.8.4 MEMOIZAÇÃO

Em §8.4.1, definimos uma função fatorial 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 e
Retorna uma versão memorada da função:

```
// retorna uma versão memorada de f.
// só funciona se os argumentos para f todos tiverem string distinta
representações.
Função Memoize (f) {
const cache = new map ();// cache de valor armazenado no
encerramento.
Função de retorno (... args) {
// Crie uma versão de string dos argumentos para usar como
uma 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 e
atribui esse objeto a uma variável local para que seja privada (no
fechamento de) a função retornada.A função retornada converte seu
Argumentos Array em uma string e usa essa string como um nome de propriedade para
o 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 estes
Argumentos, armazenam em cache esse valor e o devolve.Aqui está como podemos usar
Memoize ():

```
// retorna o maior divisor comum de dois números inteiros usando
o 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 certos 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 Propriedades herdadas (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 introduzido
uma sintaxe totalmente nova (incluindo uma palavra-chave de classe) que a torna uniforme
mais fácil de criar classes.Essas novas classes JavaScript funcionam na mesma
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
explicou esses fundamentos, mudaremos e começaremos a usar o novo,
Sintaxe da definição de classe simplificada.
Se você está familiarizado com a programação de objetos fortemente digitada
idiomas como Java ou C ++, você notará que as classes JavaScript são
Muito diferente das classes nessas línguas.Existem alguns
semelhanças sintáticas, e você pode imitar muitos recursos de "clássico"
Aulas em JavaScript, mas é melhor entender antecipadamente que
Classes de JavaScript e mecanismo de herança baseado em protótipo são
substancialmente diferente das classes e herança baseada em classes
Mecanismo de Java e idiomas semelhantes.

9.1 classes e protótipos

Em JavaScript, uma classe é um conjunto de objetos que herdam propriedades do
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 cria
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
instância 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
do
// Objeto de protótipo definido abaixo.O objeto de protótipo é
armazenado como
// uma propriedade desta função e define o compartilhado
Métodos (comportamento)
// para todos os objetos de intervalo.
Seja r = object.create (range.methods);
// Armazene os pontos de partida e final (estado) desta nova linha
objeto.
// estas são propriedades não herdadas que são exclusivas para
este objeto.
r.from = de;
r.to = para;
// finalmente retorna o novo objeto
retornar r;
}
// Este objeto de protótipo define métodos herdados por toda a faixa
objetos.
range.methods = {
// 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.
// 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
r.includes (2) // => true: 2 está no intervalo
r.toString () // => "(1 ... 3)"
[... r] // => [1, 2, 3];converter em uma matriz
via iterador
```

Há algumas coisas que valem a pena notar no Código do Exemplo 9-1:
Este código define um intervalo de função de fábrica () para criar
novos objetos de intervalo.
Ele usa a propriedade Métodos desta Função ()
local conveniente para armazenar o objeto de protótipo que define o
aula.Não há nada de especial ou idiomático em colocar o
Objeto de protótipo aqui.
A função range () define de e para propriedades em
cada objeto de intervalo.Estes são os não -compartilhados, não herdados
propriedades que definem o estado único de cada indivíduo
Objeto de alcance.
O objeto Range.Methods usa a sintaxe de abreviação ES6
para definir métodos, e é por isso que você não vê o
Função de palavra -chave em qualquer lugar.(Ver §6.10.5 para revisar o objeto
Sintaxe 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
o método é prefixado com *, o que indica que é um
Função 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 com
o loop for/of e com o ... Spread Operator.
Os métodos compartilhados e herdados definidos em range.methods
Todos usam as propriedades de e para as propriedades que foram inicializadas no
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 o novo
Palavra -chave conforme descrito em §8.2.3.Invocações construtoras usando novo
Crie automaticamente o novo objeto, para que o próprio construtor só precisa
Para inicializar o estado desse novo objeto.A característica crítica de
Invocações de construtor é que a propriedade do protótipo do
O construtor é usado como o protótipo do novo objeto.§6.2.3
apresentou protótipos e enfatizou que, embora quase todos os objetos
Tenha um protótipo, apenas alguns objetos têm uma propriedade de protótipo.
Finalmente, podemos esclarecer isso: são objetos de função que têm um
Propriedade do protótipo.Isso significa que todos os objetos criados com o
A mesma função do construtor herda do mesmo objeto e são

Portanto, membros da mesma classe.Exemplo 9-2 mostra como nós
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 o
maneira idiomática de criar uma classe em versões de javascript que não
Apoie a palavra -chave da classe ES6.Mesmo que a classe esteja bem
Suportado agora, ainda há muitos código JavaScript mais antigos em torno disso
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
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
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 intervalo
ToString: function () {return "(" + this.from + "..." +
this.to + ")";}
};
// Aqui estão os usos de exemplo desta nova classe
Seja r = novo intervalo (1,3);// Crie um objeto de intervalo;Observe o
uso de novo
r.includes (2) // => true: 2 está no intervalo
r.toString () // => "(1 ... 3)"
[... r] // => [1, 2, 3];converter em uma matriz
via iterador
```

Vale a pena comparar exemplos 9-1 e 9-2 com bastante cuidado e observar
As 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 comum
Convenção: as funções do construtor definem, em certo sentido, classes e
As aulas têm nomes que (por convenção) começam com letras maiúsculas.
Funções e métodos regulares têm nomes que começam com minúsculas
cartas.
Em seguida, observe que o construtor range () é invocado (no final de
o exemplo) com a nova palavra -chave enquanto a fábrica range ()
A função foi invocada sem ela.Exemplo 9-1 usa função regular
invocação (§8.2.1) para criar o novo objeto e o exemplo 9-2 usa
Invocação do construtor (§8.2.3).Porque o construtor range () é
invocado com novo, ele não precisa chamar object.create () ou
Tome qualquer ação para criar um novo objeto.O novo objeto é automaticamente
criado 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.O
fato 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
Isso mantém as funções do construtor distintas das funções regulares ajudam
Os 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 o
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 o
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 o
A 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 descritivo, mas

arbitrário.No segundo exemplo, o protótipo é
Range.prototype, e esse nome é obrigatório.Uma invocação de
o construtor range () usa automaticamente range.prototype como
o 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
soletra 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
funciona ao definir construtores ou métodos.Lembre -se de §8.1.3
essas funções definidas dessa maneira não têm uma propriedade de protótipo
e assim não pode ser usado como construtores.Além disso, as funções de seta herdam o
essa palavra -chave do contexto em que são definidos em vez de
defini -lo com base no objeto através do qual eles são invocados, e este
os 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
Definindo 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 um
Classe: Dois objetos são casos da mesma classe se e somente se eles
herdar do mesmo objeto de protótipo.A função do construtor que
Inicializa o estado de um novo objeto não é fundamental: dois construtores
funções podem ter propriedades de protótipo que apontam para o mesmo
Objeto de protótipo.Então, ambos os construtores podem ser usados ??para criar
Instâncias da mesma classe.

Embora os construtores não sejam tão fundamentais quanto protótipos, o
O construtor serve como face pública de uma classe.Mais obviamente, o
o nome da função do construtor é geralmente adotado como o nome do
aula.Dizemos, por exemplo, que o construtor range () cria
Objetos de alcance.Mais fundamentalmente, no entanto, os construtores são usados ??como
o operando areia do operador da instância ao testar
objetos para associação a uma classe.Se tivermos um objeto r e queremos
Saiba se é um objeto de intervalo, podemos escrever:

```
r Instância de intervalo // => true: r herda de
Range.prototype
```

A instância do operador foi descrita em §4.9.4.A esquerda
operando deve ser o objeto que está sendo testado, e o caminho
Operando deve ser uma função construtora que nomeia uma classe.O
expressão o instância de c avalia como verdadeiro se o herdar
C.Prototipo.A herança não precisa ser direta: se o herdar de
um objeto que herda de um objeto que herda
C.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 foi
realmente necessário para criar um novo objeto.Qualquer javascript regular
função (excluindo funções de seta, funções geradoras e assíncronas
funções) podem ser usadas como construtor e invocações de construtor
Precisa de uma propriedade de protótipo.Portanto, todo JavaScript regular
Função automaticamente possui uma propriedade de protótipo.O valor disso
Propriedade é um objeto que possui um único construtor e não enumerável
propriedade.O valor da propriedade do construtor é a função
objeto:

```
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 seu
propriedade construtora significa que os objetos normalmente herdam um
Propriedade do construtor que se refere ao seu construtor.Desde
Construtores servem como identidade pública de uma classe, este construtor
A 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 o

Erro ao traduzir esta página.

assim:
// estende o range predefinido.prototipo 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 classe
As aulas fazem parte do JavaScript desde a primeira versão do
linguagem, mas no ES6, eles finalmente obtiveram sua própria sintaxe com o
Introdução da palavra -chave da classe.Exemplo 9-3 mostra o que nosso
A 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
Semelhante aos literais de objetos, eles não são a mesma coisa.Em
em particular, eles não suportam a definição de propriedades com
pares de nome/valor.)
O construtor de palavras -chave é usado para definir o construtor
função para a classe.A função definida não é realmente
nomeado "construtor", no entanto.A declaração de classe
A instrução define um novo intervalo de variáveis ??e atribui o valor
desta função especial do construtor nessa variável.
Se sua classe não precisar fazer nenhuma inicialização, você pode
omita a palavra -chave do construtor e seu corpo, e um vazio
A 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 classe
Palavra -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
comprimento
A 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 e
Explique as extensas 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
pode 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 o
Função de seta abreviação), em programação JavaScript, definição de classe
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
Declare uma variável antes de usá -la.

Ao contrário das declarações de função, as declarações de classe não são
"Içado."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
De algumas maneiras, eles não compartilham esse comportamento de iça
não 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
definido 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 (), não
Range.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
invocado 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 gerador
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 Iterable:
*[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 Faça 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 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 {
construtor () {
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
inicializado 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 do
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 melhor
Para 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
foi removido.Esta sintaxe de campo - com é igual a sinais e
semicolons em vez de telas e vírgulas - deixa claro que a aula
Os corpos não são iguais aos literais do objeto.

Erro ao traduzir esta página.

esse:
estático integgerRangePattern = /^\(\d+)\.\.\.(\d+)\)$/;
análise estática (s) {
Seja Matches = S.Match (Range.IntegerRangePattern);
if (! Matches) {
Jogue o novo TypeError (`não pode analisar o alcance de
"$ {s}". `)
}
Retornar nova faixa (parseint (correspondências [1]), correspondências [2]);
}
Se quiséssemos que esse campo estático fosse acessível apenas dentro da classe, nós
poderia torná -lo privado usando um nome como #pattern.
9.3.4 Exemplo: uma classe de números complexos
Exemplo 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.Inclui
Alguns comentaram o código demonstrando como podemos usar o não-
Sintaxe ainda padrão para definir campos de instância e campos estáticos dentro
o 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 o
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 existentes

JavaScript?s prototype-based inheritance mechanism is dynamic: an
Objeto herda as propriedades de seu protótipo, mesmo que as propriedades de
A mudança de protótipo após a criação do objeto.Isso significa que podemos
Aumentar as classes JavaScript simplesmente adicionando novos métodos a seus
Objetos de protótipo.

Aqui, por exemplo, é o código que adiciona um método para calcular o
Conjugado complexo com a complexa classe do Exemplo 9-4:

```
// retorna um número complexo que é o conjugado complexo de
Este.
Complex.prototype.conj = function () {return novo
Complexo (this.r, -This.i);};
```

O protótipo objeto de classes JavaScript embutido também é aberto como
Isso, o que significa que podemos adicionar métodos a números, cordas, matrizes,
funções, e assim por diante.Isso é útil para implementar um novo idioma
Recursos nas versões mais antigas do idioma:

```
// se o novo método da string startSwith () ainda não estiver
definido ...
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 o
Número da iteração
// por exemplo, para imprimir "Hello" 3 vezes:
// deixe n = 3;
// n.times (i => {console.log (`hello $ {i}`);});
Número.prototype.Times = function (f, context) {
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
objetos.Mas isso nunca é uma coisa boa a fazer porque as propriedades adicionadas
para objet.Prototype são visíveis para/in loops (embora você
pode evitar isso usando object.DefineProperty () [§14.1] para
tornar a nova propriedade não anexerável).
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 seu próprio
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
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 de superclass a () para garantir que as instâncias sejam
completamente inicializado.

Erro ao traduzir esta página.

```
// Verifique se o protótipo de extensão herda do intervalo
protótipo
Span.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.prototipo para herdar do range.prototype.A chave
linha 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 e
para as propriedades que o construtor range () faz e, portanto, não precisa
Para 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 robusta
O mecanismo precisa permitir que as classes invocem os métodos e
construtor de sua superclasse, mas antes do ES6, JavaScript não
Tenha 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 super
No ES6 e mais tarde, você pode criar uma superclasse simplesmente adicionando um
estende a cláusula a uma declaração de classe, e você pode fazer isso mesmo para
Aulas internas:

```
// 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 a = new EZARRAY ();
uma instância de ezarray // => true: a é uma instância da subclasse
uma instância de matriz // => true: a também é uma superclasse
exemplo.
a.push (1,2,3,4);// A.Length == 4;Podemos usar herdado
Métodos
a.pop () // => 4: outro método herdado
a. primeiro // => 1: primeiro getter definido por
subclasse
A.Last // => 3: Último getter definido por
subclasse
a [1] // => 2: Sintaxe de acesso regular à matriz
ainda funciona.
Array.isarray (a) // => true: a instância da subclasse realmente é
uma matriz
Ezarray.isarray (a) // => true: subclasse herda estática
```

Métodos também!
Esta subclasse EZARRAY define dois métodos simples de getter.Instâncias de
Ezarray se comportará 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.prototype.isprototypeof (ezarray.prototype) // => true
// e ezarray herda métodos e propriedades estáticas porque
// EZARRAY herda da Array.Esta é uma característica especial de
o
// estende a palavra -chave e não é possível antes do ES6.
Array.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
e os valores do mapa são dos tipos especificados (de acordo com
typeof).É importante ressaltar que este exemplo demonstra o uso do
super palavra -chave 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 tipedmap 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 o
Tipos de chave e valor desejados.Estas devem ser strings, como "número"
e "booleano", que o operador do tipo de retorna.Você também pode especificar
Um terceiro argumento: uma matriz (ou qualquer objeto iterável) de [chave, valor]
Matrizes que especificam as entradas iniciais no mapa.Se você especificar algum
entradas iniciais, então a primeira coisa que o construtor faz é verificar que
seus tipos estão corretos.Em seguida, o construtor chama a superclasse
Construtor, usando a super palavra -chave como se fosse um nome de função.
O construtor map () leva um argumento opcional: um objeto iterável
de matrizes [chave, valor].Então o terceiro argumento opcional do
O construtor typeDMap () é o primeiro argumento opcional para o mapa ()
construtor, e passamos a esse construtor de superclasse com
super (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 por
definindo this.KeyType e this.valuetype para o especificado
tipos.Ele precisa definir essas propriedades para que possa usá -las novamente em
o método set ().
Existem algumas regras importantes que você precisará saber sobre o uso
super () nos construtores:
Se você definir uma classe com a palavra -chave estende, então o
construtor 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
Inicialize -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
foi 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 um
nova entrada para o mapa.Dizemos que este método set () no typeDmap
substitui o método set () de sua superclasse.Este mapa 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 superclass 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
A abordagem 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 comporte
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
Considere o conjunto de subclassificação e adicionar um método count ().Por outro
mão, uma vez que começamos a pensar em como podemos implementar isso
método count (), 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
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 remover () como você faria para um
Conjunto, e
* Call count () para descobrir quantas vezes um determinado valor tem
foi adicionado.
* O iterador padrão produz os valores que foram adicionados
pelo menos
* uma vez.Use entradas () se você deseja iterar [valor, contagem]
pares.
*/
classe Histograma {
```

```
// 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
tem (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 parcial
implementação que todas as subclasses herdam e compartilham.As subclasses,
Então, só precisa definir seu próprio comportamento único, implementando
os métodos abstratos definidos - mas não implementados - por
Superclass.Observe que o JavaScript não tem nenhuma definição formal de
Métodos abstratos ou classes abstratas;Estou simplesmente usando esse nome aqui
para métodos não implementados e classes implementadas incompletamente.
O exemplo 9-8 é bem comentado e permanece por conta própria.Eu encorajo
Você lê -lo como um exemplo capstone para este capítulo sobre as classes.O
A 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 membros
de alguns
* outro conjunto.Porque é definido em termos de outro conjunto
não é
* gravável, e porque tem membros infinitos, não é
enumerável.
* 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 herdamos
tem (x) {return this.set.has (x);}
// e também substituímos este método de objeto
tostring () {return `{x |x ? $ {this.set.toString ()}} `;}
}
/**
* O conjunto de alcance é uma subclasse concreta do AbstractSet.Seus membros
são
* 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}} `;
}
}
/*
* AbstractEnumerablesetet é uma subclasse abstrata de
AbstractSet.Define
* um getter abstrato que retorna o tamanho do conjunto e também
define um
* Iterador abstrato.E então implementa concreto
isEmpty (), tostring (),
* e iguais () métodos em cima deles.Subclasse isso
implementar o
* iterador, o tamanho do getter e o método has () obtêm estes
concreto
* Métodos gratuitamente.
*/
```

Erro ao traduzir esta página.

```
*[Symbol.iterator] () {rende this.Member;}
}
/*
* AbstractWritableset é uma subclasse abstrata de
AbstractEnumerabableset.
* Define os métodos abstratos insert () e remove () que
insira e
* Remova os elementos individuais do conjunto e depois implementa
concreto
* Métodos Add (), Subract () e Intersect () em cima deles.
Observe que
* Nossa API diverge aqui do conjunto de javascript padrão
aula.
*/
classe AbstractWritableset 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 abstrumwritableset com
```

Erro ao traduzir esta página.

```
e bit
deixe bit = x % 8;
if (! this._has (byte, bit)) {// se esse bit for
ainda não está definido
this.data [byte] | = bitset.bits [bit];// então
defina
this.n ++;// e
tamanho do conjunto de incrementos
}
} outro {
jogue novo TypeError ("Elemento de conjunto inválido:" + x);
}
}
Remova (x) {
if (this._valid (x)) {// se o valor for
válido
deixe byte = math.floor (x / 8);// Calcule o byte
e bit
deixe bit = x % 8;
if (this._has (byte, bit)) {// se esse bit for
já definido
this.data [byte] & = bitset.masks [bit];// então
sem serem
this.n--;// e
tamanho decrescente
}
} outro {
jogue novo TypeError ("Elemento de conjunto inválido:" + x);
}
}
// um getter para retornar o tamanho do conjunto
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 () e
Remover () métodos
Bitset.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 Resumo

Este capítulo explicou os principais recursos das classes JavaScript:
Objetos que são membros da mesma classe herdam propriedades
do mesmo objeto de protótipo.O objeto de protótipo é o
Principais características das classes JavaScript, e é possível definir
Aulas com nada mais do que o objeto.create ()
método.
Antes do ES6, as aulas eram mais tipicamente definidas pelo primeiro
definindo uma função construtora.Funções criadas com o
A palavra -chave da função tem uma propriedade de protótipo e a
O valor desta propriedade é um objeto usado como protótipo
de todos os objetos criados quando a função é invocada com novo
como um construtor.Ao inicializar este objeto de protótipo, você pode
Defina os métodos compartilhados da sua classe.Embora o
Objeto de protótipo é o principal recurso da classe, o construtor
A função é a identidade pública da classe.
ES6 apresenta uma palavra -chave de classe que facilita a
Definir classes, mas sob o capô, construtor e protótipo
O mecanismo permanece o mesmo.
As subclasses são definidas usando a palavra -chave Extends em uma classe

Erro ao traduzir esta página.

Capítulo 10. Módulos

O objetivo da programação modular é permitir que grandes programas sejam
montado usando módulos de código de autores e fontes díspares
e para que todo esse código seja executado corretamente, mesmo na presença de código
que os vários autores do módulo não anteciparam.Como prático
matéria, modularidade é principalmente sobre encapsular ou ocultar particular
A implementação detalha 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.requer ()-módulos baseados em
parte fundamental do ambiente de programação do nó, mas foram
Nunca adotado como parte oficial da linguagem JavaScript.Em vez de,
O ES6 define módulos usando palavras -chave de importação e exportação.Embora
Importação e exportação fazem parte do idioma há anos, eles
foram implementados apenas por navegadores da Web e nó 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 fechamentos
Módulos de nó usando requer ()
Módulos ES6 usando exportação, importação e importação ()
10.1 módulos com classes, objetos e
Fechamentos
Embora possa ser óbvio, vale ressaltar que um dos
características importantes das classes é que elas atuam como módulos para seus
Métodos.Pense no exemplo 9-8.Esse exemplo definiu um número
de diferentes classes, todas que tinham um método chamado ().Mas você
não teria nenhum problema em escrever um programa que usasse vários conjuntos
Classes desse exemplo: não há perigo que a implementação
de has () de singletonset substituirá o método has () de
Bitset, por exemplo.
A razão pela qual os métodos de uma classe são independentes do
Métodos de outras classes não relacionadas é que os métodos de cada classe são
definido como propriedades de objetos de protótipo independentes.A razão disso
As classes são modulares é que os objetos são modulares: definir uma propriedade em um
O objeto JavaScript é como declarar uma variável, mas adicionando propriedades
para objetos não afeta o espaço de nome global de um programa, nem
afeta as propriedades de outros objetos.JavaScript define alguns
funções e constantes matemáticas, mas em vez de definir todos elas
Globalmente, 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 de
Definindo classes globais com nomes como singletonset e bitset, que
Exemplo 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
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
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 do
módulo 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
função 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 no
Exemplo anterior, você pode imaginar escrever código como o seguinte para
Faça uso desses módulos:

```
// Obtenha referências aos módulos (ou ao conteúdo do módulo) que
Nós precisamos
const stats = requer ("stats.js");
const bitset = requer ("sets.js"). bitset;
// agora escreva código usando esses módulos
Seja s = novo bitset (100);
S.Insert (10);
S.Insert (20);
S.Insert (30);
deixe a média = estatísticas.mean ([... s]);// A média é de 20
```

Este código é um esboço aproximado de como as ferramentas de Bundling de código (como
webpack e pacote) para os navegadores da web funcionam, e também é um simples
Introdução à função requer () como a usada no nó
programas.

10.2 Módulos no nó

Na programação de nós, é normal dividir programas em tantos arquivos
como parece natural.Esses arquivos de código JavaScript são assumidos para todos os vivos
em 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 abstrumwritableset {
// implementação omitida
};
O valor padrão do Module.Exports é o mesmo objeto que
Exportações refere -se a.No módulo de estatísticas anteriores, poderíamos ter
atribuído a função média ao module.exports.mean em vez de
exports.Mean.Outra abordagem com módulos como as estatísticas
módulo é exportar um único objeto no final do módulo, em vez de
Exportando funções uma a uma enquanto você avança:
// define todas as funções, público e privado
const 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 públicos
module.exports = {média, stddev};
10.2.2 Importações de nós
Um módulo de nó importa outro módulo chamando o requer ()
função.O argumento desta função é o nome do módulo a ser
importado, 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 funções
const 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 idiomática
atribuição para importar
// exatamente as funções que queremos diretamente no local
namespace:
const {stdDev} = requer ('./ stats.js');
// Isso é bom e sucinto, embora tenhamos um pouco de contexto
// sem o prefixo 'estatísticas' como um namspace para o stddev ()
função.
Seja sd = stddev (dados);
```

10.2.3 módulos no estilo de nó na web

Módulos com um objeto de exportação e uma função requer () são construídos no nó.Mas se você estiver disposto a processar seu código com um agrupamento ferramenta como webpack, também é possível usar esse estilo de módulos Para 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 oficial Módulos 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
são exportados de um módulo estão disponíveis para uso em módulos que
importá -los explicitamente.Os módulos ES6 diferem dos módulos de nó no
sintaxe usada para exportar e importar e também da maneira que
Os módulos são definidos em navegadores da Web.As seções a seguir explicam
essas coisas em detalhes.

Primeiro, porém, observe que os módulos ES6 também são diferentes dos regulares
Javascript "scripts" de algumas maneiras importantes.O mais óbvio
A diferença é a própria modularidade: em scripts regulares, nível superior
Declarações de variáveis, funções e classes entram em um único global
Contexto 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.E isso significa que
O código nos módulos não pode usar a instrução com os argumentos ou os argumentos
objeto ou variáveis ??não 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
código de nível.(Por outro lado, scripts em navegadores da web e nó definem
o objeto global.)

Módulos ES6 na web e no nó
Os 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 disso
Escrevendo, no entanto, os módulos ES6 são finalmente suportados nativamente por todos os navegadores da web
Além do Internet Explorer.Quando usados ??nativamente, os módulos ES6 são adicionados em
Páginas html com uma tag <script type = "módulo">, descrito posteriormente
Neste capítulo.
Enquanto isso, tendo pioneiro na modularidade JavaScript, o nó se encontra no
posição desajeitada de ter que apoiar dois módulos não totalmente compatíveis
sistemas.O nó 13 suporta módulos ES6, mas por enquanto, a grande maioria do nó
Os programas ainda usam módulos de nó.
10.3.1 ES6 Exportações
Exportar 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 e
aulas como você normalmente faria, sem declaração de exportação, e então
(normalmente no final do seu módulo) Escreva uma única declaração de exportação
Isso declara exatamente o que é exportado em um único local.Então, em vez de

Escrevendo três exportações individuais no código anterior, poderíamos ter
escrito equivalentemente uma única linha no final:
exportação {círculo, degreestoradians, pi};
Esta sintaxe parece a palavra -chave de exportação seguida de um objeto
literal (usando notação abreviada).Mas neste caso, os aparelhos encaracolados fazem
na verdade não define um objeto literal.Esta sintaxe de exportação simplesmente requer
Uma lista separada por vírgula de identificadores dentro de aparelhos encaracolados.
É comum escrever módulos que exportam apenas um valor (normalmente um
função ou classe) e, neste caso, geralmente usamos o padrão de exportação
em 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 exportação
facilita 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 qualquer
expressão, incluindo expressões de função anônima e anônimas
expressões de classe.Isso significa que, se você usar o padrão de exportação, você
pode exportar literais de objetos.Então, diferente da sintaxe de exportação, se você vir
aparelho encaracolado após o padrão de exportação, é realmente um objeto literal que
está 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 padrão
Exportar, 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 um
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
Seja o mesmo 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
a 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 pode aparecer no nível superior de um módulo e não é permitido dentro
Aulas, funções, loops ou condicionais.Por quase universal
Convenção, as importações necessárias para um módulo são colocadas no início de

o módulo.Curiosamente, no entanto, isso não é necessário: como função
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 constante
String 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
Use 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 deve
Seja um caminho absoluto começando com "/", ou um caminho relativo começando com "./"
ou "../", ou um URL completo com protocolo e nome de host.O ES6
A 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
um, 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 módulo
Isso os define.Em vez disso, fornecemos um nome local quando importamos
Esses valores.Mas as exportações não-defensivas de um módulo têm nomes no
exportando módulo, e quando importamos esses valores, nos referimos a eles
por esses nomes.O módulo de exportação pode exportar qualquer número de
valor nomeado.Uma declaração de importação que faz referência a que o módulo pode
importar qualquer subconjunto desses valores simplesmente listando seus nomes dentro
aparelho encaracolado.Os aparelhos encaracolados fazem desse tipo de declaração de importação
Parece algo como uma tarefa de destruição e destruição
A tarefa é realmente uma boa analogia para o que esse estilo de importação é
fazendo.Os identificadores dentro de aparelhos encaracolados são todos içados no topo de
o módulo de importação e se comporta como constantes.
Os guias de estilo às vezes recomendam que você importe explicitamente a cada
Símbolo que seu módulo usará.Ao importar de um módulo que
define muitas exportações, no entanto, você pode importar facilmente tudo com
Uma declaração de importação como esta:
importação * como estatísticas de "./stats.js";
Uma declaração de importação como essa cria um objeto e o atribui a um
constante estatísticas nomeadas.Cada uma das exportações não padrão do módulo
Ser importado se torna uma propriedade desse objeto de estatísticas.Não-defasa
As exportações sempre têm nomes, e esses são usados ??como nomes de propriedades
dentro 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 importado
mean () e stddev () funcionam através do objeto STATS,
invocando -os como estatísticas.mean () e stats.stddev ().
Os módulos geralmente definem uma exportação padrão ou múltiplo nomeado
exportações.É legal, mas um tanto incomum, para um módulo usar os dois
exportar e exportar inadimplência.Mas quando um módulo faz isso, você
pode importar o valor padrão e os valores nomeados com um
Declaração 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 um
outra forma da declaração de importação que é usada com módulos que
não têm exportações.Para incluir um módulo de não exportações em seu
Programa, 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 subsequente
as importações não fazem nada.) Um módulo que apenas define funções é só útil
se exportar pelo menos uma dessas funções.Mas se um módulo executar alguns
Código, pode ser útil importar mesmo sem símbolos.Um
Módulo de análise para um aplicativo da web pode executar o código para se registrar
vários manipuladores de eventos e depois usam esses manipuladores de eventos para enviar
Dados de telemetria de volta ao servidor em horários apropriados.O módulo é
independente e não precisa exportar nada, mas ainda precisamos

Erro ao traduzir esta página.

fornece outra maneira de importar de módulos que definem ambos
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
Use 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 objeto
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 ??neste código.Se não estamos sendo seletivos com um reexport e simplesmente queremos exportar todos os valores nomeados de outro módulo, podemos usar um curinga:

exportação * de "./stats/mean.js";
exportação * de "./stats/stddev.js";

Reexportar sintaxe permite renomear com a mesma importação regular e Declarações 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 exportação em vez de padrão de exportação.De fato, no entanto, uma vez que estes são módulos com apenas uma única exportação, teria feito sentido definir eles com o padrão de exportação.Se tivéssemos feito isso, então o reexport A sintaxe é um pouco mais complicada porque precisa definir um nome para 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 o Exportação padrão do seu módulo, você pode fazer uma importação seguida por um 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 módulo
exportar {significa como padrão} de "./stats.js"
E finalmente, para reexportar a exportação padrão de outro módulo como o
exportação padrão do seu módulo (embora não esteja claro por que você faria
quero fazer isso, já que os usuários podem simplesmente importar o outro módulo
diretamente), você pode escrever:
// O módulo médio.js simplesmente reexporta as estatísticas/mean.js
exportação padrão
exportar {default} de "./stats/mean.js"
10.3.5 Módulos JavaScript na Web
As seções anteriores descreveram os módulos ES6 e sua importação
e exportar declarações de maneira um tanto abstrata.Nesta
seção e na próxima, discutiremos como eles realmente funcionam em
navegadores da web e se você ainda não é uma web experiente
Desenvolvedor, você pode encontrar o resto deste capítulo mais fácil de entender
Depois de ler o capítulo 15.
No início de 2020, o código de produção usando módulos ES6 ainda é geralmente
Pacotado com uma ferramenta como Webpack.Existem trocas para fazer isso,
Mas, no geral, o Bundling de código tende a dar melhor desempenho.Que
pode muito bem mudar no futuro à medida que a velocidade da rede cresce e o navegador
Os fornecedores continuam otimizando suas implementações de módulos ES6.
Mesmo que as ferramentas de agrupamento ainda possam ser desejáveis ??na produção, elas
não são mais necessários no desenvolvimento, pois todos os navegadores atuais

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
quero usar de maneira nativamente as diretivas de importação em um navegador da web, você deve
Diga 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
importado 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 um
A 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
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
apontar 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
O módulo estará pronto para uso antes de qualquer código do seu módulo
começa a correr.
Na web, o código deve ser transferido por uma rede em vez de ser
Leia 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 - faça muito sentido.
É comum que os aplicativos da web carreguem inicialmente apenas o suficiente de seus
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
a quantidade 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 <cript> 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 de
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
aguarde:
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 como
Um que você usaria com uma diretiva de importação estática.Mas com importação (),
Você não está limitado a usar uma string constante literal: qualquer expressão
que avalia uma string na forma correta servirá.
Dinâmico importar () parece uma invocação de funções, mas na verdade é
não.Em vez disso, importar () é um operador e os parênteses são um
parte necessária da sintaxe do operador.A razão para esse pouco incomum de
A sintaxe é que o import () precisa ser capaz de resolver especificadores de módulo
como URLs em relação ao módulo atualmente em execução, e isso requer um
um pouco de magia de implementação que não seria legal para colocar em um
Função de JavaScript.A função versus a distinção do operador raramente
faz a diferença na prática, mas você notará se tentar escrever
código 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.O
A maneira mais direta de usar um patrimônio de código é dizer o principal
ponto de entrada para o seu programa e deixe -o encontrar toda a importação estática
diretivas e monte tudo em um arquivo grande.Por estrategicamente
Usando chamadas dinâmicas de importação (), no entanto, você pode quebrar esse
Monolítico Bachar em um conjunto de feixes menores que podem ser carregados
Sob demanda.

10.3.7 Import.Meta.url

Há um recurso final do sistema de módulos ES6 para discutir.Dentro de
um 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 () faz
é 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 o
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 LocalStringsurl (Locale) {
retornar novo URL (`L10N/$ {Locale} .json`, import.meta.url);
}
```

10.4 Resumo

O objetivo da modularidade é permitir que os programadores ocultem o
Detalhes da implementação de seu código para que pedaços de código de
Várias fontes podem ser montadas em grandes programas sem
preocupando -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
alcançado 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 () e
Defina 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 pode
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 JavaScript
que podemos considerá -los parte da própria linguagem.Este capítulo
abrange outras APIs importantes, mas menos fundamentais, que podem ser pensadas
de definir a ?biblioteca padrão? para JavaScript: estes são úteis
classes e funções que são incorporadas ao JavaScript e disponíveis para todos
Programas JavaScript nos navegadores da Web e no nó.

As seções deste capítulo são independentes uma da outra, e você
pode 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 tipedarrays que representam matrizes
de dados binários, juntamente com uma classe relacionada para extrair valores
de dados binários que não são de teatro.

Expressões regulares e a classe Regexp, que define
padrões textuais e são úteis para processamento de texto.Esta seção
Também abrange a sintaxe de expressão regular em detalhes.

A classe de data para representar e manipular datas e
vezes.

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

O objeto JSON, cujos métodos apóiam a serialização e
Deserialização de estruturas de dados JavaScript compostas de
objetos, matrizes, cordas, números e booleanos.
O objeto INTL e as classes que ele define que podem ajudá -lo
Localize seus programas JavaScript.
O objeto de console, cujos métodos produzem seqüências de seqüências de maneiras de maneiras
que são particularmente úteis para programas de depuração e
registrar o comportamento desses programas.
A classe URL, que simplifica a tarefa de analisar e
Manipulando URLs.Esta seção também abrange funções globais
para codificar e decodificar URLs e suas partes componentes.
setTimeout () e funções relacionadas para especificar o código para
ser executado após o intervalo de tempo especificado.
Algumas das seções deste capítulo - principalmente, as seções sobre digitadas
matrizes e expressões regulares - são bastante longas porque há
informações de fundo significativas que você precisa entender antes de você
pode usar esses tipos de maneira eficaz.Muitas das outras seções, no entanto,
são curtos: eles simplesmente apresentam uma nova API e mostram alguns exemplos
de seu uso.
11.1 conjuntos e mapas
O tipo de objeto de JavaScript é uma estrutura de dados versátil que pode ser usada para
Strings de mapa (nomes de propriedades do objeto) para valores arbitrários.E
Quando o valor que está sendo mapeado é algo fixo como verdadeiro, então o
O 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 com
nomes como "ToString", que normalmente não se destinam a fazer parte do
mapa ou conjunto.
Por esse motivo, o ES6 apresenta as classes verdadeiras de conjunto e mapa, que vamos
Cubra 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
O valor é um membro de um conjunto ou não é um membro;não é possível
para 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 remova os elementos a qualquer momento com add (), delete () e
claro().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 vazio
s.size // => 0
S.Add (1);// Adicione um número
s.size // => 1;agora o conjunto tem um membro
S.Add (1);// Adicione o mesmo número novamente
s.size // => 1;O tamanho não muda
s.Add (verdadeiro);// Adicione outro valor;Observe que é
Tipos finos para misturar
s.size // => 2
s.Add ([1,2,3]);// Adicione um valor de matriz
s.size // => 3;A matriz foi adicionada, não a sua
elementos
s.delete (1) // => true: elemento excluído com sucesso
1
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
s.size // => 1: Ainda existe uma matriz em
o conjunto
s.clear ();// Remova tudo do conjunto
s.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 individual
elementos.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 o
Os valores são objetos (ou matrizes ou funções), eles também são
comparado como se com ===.É por isso que não conseguimos
Exclua o elemento da matriz do conjunto neste código.Adicionamos um
matriz para o conjunto e depois tentou remover essa matriz passando um
matriz diferente (embora com os mesmos elementos) para o
Método delete ().Para que isso funcione, teríamos
teve que 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 o
A troca é que os conjuntos de python apenas permitem membros imutáveis, como tuplas, e fazer
Não permita que listas e ditos sejam adicionados aos conjuntos.
Na prática, a coisa mais importante que fazemos com os conjuntos não é adicionar e
Remova os elementos deles, mas para verificar se um especificado
O valor é um membro do conjunto.Fazemos isso com o método Has ():
Let OneDigitPries = novo conjunto ([2,3,5,7]);
onedigitPries.has (2) // => true: 2 é um prime 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
otimizado 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 o tempo
tomadas 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
primos
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 o
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 tecla
tem outro valor associado a (ou "mapeado para").Em certo sentido, um mapa
é como uma matriz, mas em vez de usar um conjunto de números inteiros seqüenciais como o
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
é.
Crie um novo mapa com o construtor map ():

Seja m = novo map ();// Crie um novo mapa vazio
Seja n = novo mapa ([// um novo mapa inicializado com teclas de string
mapeado para números
["One", 1],
["dois", 2]
]);
O argumento opcional para o construtor map () deve ser um iterável
Objeto 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 um
Matriz de matrizes.Mas você também pode usar o construtor map () para copiar
outros mapas ou para copiar os nomes e valores de propriedades de um existente
objeto:
Deixe copiar = novo mapa (n);// um novo mapa com as mesmas chaves e
valores como mapa n
Seja o = {x: 1, y: 2};// um objeto com duas propriedades
Seja 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 valor
associado a uma determinada chave com get () e pode adicionar uma nova chave/valor
Emparelhar com set ().Lembre -se, porém, que um mapa é um conjunto de chaves, cada um
dos quais tem um valor associado.Isso não é exatamente o mesmo que um conjunto de
pares de chave/valor.Se você ligar para o set () com uma chave que já existe no
mapa, você mudará o valor associado a essa chave, não adicionar um novo
Mapeamento de chave/valor.Além de obter () e set (), a classe de mapa
Define também métodos que são como métodos definidos: o uso tem () para verificar
se um mapa inclui a chave especificada;use delete () para remover um
chave (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 descobrir
Quantas chaves um mapa contém.

```
Seja m = novo map ();// Comece com um mapa vazio
M.Size // => 0: Mapas vazios não têm chaves
M.Set ("One", 1);// mapeie a chave "um" para o valor 1
M.set ("dois", 2);// e a chave "dois" para o valor 2.
M.Size // => 2: O mapa agora tem duas chaves
m.get ("dois") // => 2: retorna o valor associado
com chave "dois"
m.get ("três") // => indefinido: esta chave não está no
definir
M.set ("One", verdadeiro);// altere o valor associado a um
chave existente
M.Size // => 2: O tamanho não muda
m.has ("um") // => true: o mapa tem uma chave "
m.has (verdadeiro) // => false: o mapa não tem uma chave
verdadeiro
M.Delete ("One") // => True: a chave existia e a exclusão
conseguiu
M.Size // => 1
M.Delete ("três") // => false: Falha ao excluir um
chave inexistente
m.clear ();// Remova todas as chaves e valores do
mapa
```

Como o método add () de set, o método set () de mapa pode ser
acorrentado, o que permite que os mapas sejam inicializados sem usar matrizes de
Matrizes:

```
Seja m = novo map (). set ("um", 1) .set ("dois", 2) .set ("três",
3);
M.Size // => 3
m.get ("dois") // => 2
```

Como no conjunto, qualquer valor de JavaScript pode ser usado como uma chave ou um valor em um
Mapa.Isso inclui nulo, indefinido e nan, além de referência

Tipos como objetos e matrizes.E, como na classe Set, o mapa compara
chaves por identidade, não por igualdade, portanto, se você usar um objeto ou matriz como um
chave, 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 o
Número 2.
M.Size // => 2: Existem duas chaves neste mapa
m.get ({}) // => indefinido: mas este objeto vazio
não é uma chave
M.set (M, indefinido);// mapeia o mapa em si para o valor
indefinido.
m.has (m) // => true: m é uma chave em si mesma
m.get (m) // => indefinido: o mesmo valor que obteríamos se
M não foi uma chave
Os objetos de mapa são iteráveis ??e cada valor iterado é uma matriz de dois elementos
onde o primeiro elemento é uma chave e o segundo elemento é o valor
associado a essa chave.Se você usar o operador de spread com um mapa
objeto, você receberá uma variedade de matrizes como as que passamos para o
Map () construtor.E quando itera um mapa com um loop para/de
é idiomático para usar a atribuição de destruição para atribuir a chave e o valor
para 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 irá
ser 1
// Na segunda iteração, a chave será "y" e valor
será 2
}

Como a classe Set, a classe do mapa itera em ordem de inserção.O primeiro
O par de teclas/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 de um
Mapeie, use os métodos Keys () e valores (): estes retornam iterável
Objetos 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
seja "x"
// Na segunda invocação, o valor será 2 e a chave será
seja "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 de 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" e
não 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, por
Contraste, mantém referências "fracas" aos seus principais valores para que não sejam
alcançável através do mapa fraco, e sua presença no mapa faz
não impedir que sua memória seja recuperada.
O construtor frAchapMap () é exatamente como o construtor map (), mas
Existem algumas diferenças significativas entre o Frafmap e o mapa:
As teclas de mapa fraco devem ser objetos ou matrizes;Os valores primitivos são
não está sujeito a coleta de lixo e não pode ser usado como chaves.
FrawMap implementa apenas o get (), set (), tem () e
Métodos delete ().Em particular, o mapa fraco não é iterável
e não define as chaves (), valores () ou foreach ().Se
Frafmap era iterável, então suas chaves seriam acessíveis e
Não seria fraco.
Da mesma forma, o Frafmap não implementa a propriedade de tamanho
Porque o tamanho de um mapa fraco pode mudar a qualquer momento como
Objetos 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
estão escrevendo uma função que pega um argumento de objeto e precisa
Execute algum cálculo demorado nesse objeto.Para
Eficiência, você deseja cache o valor calculado para a reutilização posterior.Se
Você usa um objeto de mapa para implementar o cache, você impedirá qualquer um dos
os objetos de sempre sendo recuperados, mas usando um mapa fraco, você
Evite esse problema.(Muitas vezes você pode obter um resultado semelhante usando um
propriedade de símbolo privado para cache o valor calculado diretamente no
objeto.Veja §6.10.3.)
Frawset implementa um conjunto de objetos que não impedem aqueles
Objetos de lixo coletados.O construtor fracos ()
funciona como o construtor set (), mas os objetos fracos diferem do conjunto
Objetos da mesma maneira que os objetos fracos diferem do mapa
objetos:
O BRACHST não permite valores primitivos como membros.
Fraqueza implementa apenas o add (), tem () e
Métodos delete () e não é iterável.
O BRACHST não possui uma propriedade de tamanho.
O fraco não é usado com frequência: seus casos de uso são como os para
Map fraco.Se você deseja marcar (ou "marca") um objeto como tendo alguns
Propriedade ou tipo especial, por exemplo, você pode adicioná -lo a um conjunto fraco.
Então, em outros lugares, quando você quiser verificar essa propriedade ou tipo, você
pode testar a associação a esse conjunto fraco.Fazendo isso com um conjunto regular
impediria que todos os objetos marcados fossem coletados de lixo, mas
Isso não é uma preocupação 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 muitos 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 próximos das matrizes de baixo nível desses idiomas. Matrizes digitadas não são tecnicamente matrizes (Array.isArray () Returns falso para eles), mas eles 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 armazenado na matriz.

Você deve especificar o comprimento de uma matriz digitada quando criar e 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 digitados

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

Construtor

Tipo numérico

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 Uint
Segure 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 requer
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 menos 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 menos
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_ELEMENT
propriedade com o valor 1, 2, 4 ou 8, dependendo do tipo.
11.2.2 Criando matrizes digitadas
A maneira mais simples de criar uma matriz digitada é chamar o apropriado
construtor com um argumento numérico que especifica o número de
Elementos que você deseja na matriz:
Let Bytes = novo UINT8Array (1024);// 1024 bytes
Let Matrix = new Float64Array (9);// uma matriz 3x3
Let Point = new Int16Array (3);// Um ??ponto no espaço 3D
Seja RGBA = novo UINT8ClampedArray (4);// Um ??pixel de 4 bytes RGBA
valor
Seja sudoku = novo int8array (81);// um quadro de 9x9 Sudoku
Quando você cria matrizes digitadas dessa maneira, os elementos da matriz são todos
garantido a ser inicializado para 0, 0n ou 0,0.Mas se você conhece o
Valores que você deseja em sua matriz digitada, você também pode especificar esses valores
Quando você cria a matriz.Cada um dos construtores de matriz tipados tem
estático de () e () métodos de fábrica que funcionam como
Array.From () e Array.Of ():

Seja branco = uint8clampedArray.of (255, 255, 255, 0);// rgba
branco opaco
Lembre-se de que o método da fábrica do Array.From () espera um matriz
ou objeto iterável como seu primeiro argumento.O mesmo vale para os digitados
variantes de matriz, exceto que o objeto iterável ou parecido com uma matriz também deve
tem elementos numéricos.Strings são iteráveis, por exemplo, mas seria
Não faça sentido em passá -los para o método de fábrica de () de um digitado
variedade.
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 para
A função construtora, que se comporta exatamente a mesma.Observe que
Tanto o construtor quanto o método de fábrica de () permitem que você
Copie as matrizes digitadas existentes, ao mesmo tempo em que altera o tipo:
deixe ints = uint32Array.From (branco);// os mesmos 4 números,
mas como ints
Quando você cria uma nova matriz digitada de uma matriz existente, iterável, ou
objeto semelhante a uma matriz, os valores podem ser truncados para ajustar o tipo
restrições da sua matriz.Não há avisos ou erros quando isso
acontece:
// flutua truncado para ints, mais longos truncados para 8 bits
Uint8array.of (1.23, 2,99, 45000) // => novo uint8Array ([1, 2,
200])
Finalmente, há mais uma maneira de criar matrizes digitadas que envolvem o
Tipo de matriz.Um ArrayBuffer é uma referência opaca a um pedaço de
memória.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 memória
A classe ArrayBuffer não permite que você leia ou escreva nenhum dos
bytes que você alocou.Mas você pode criar matrizes digitadas que usam
a memória do buffer e isso permite que você leia e escreva isso
memória.Para fazer isso, chame o construtor de matriz digitado com um
ArrayBuffer 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 matriz
buffer.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 digitado
Construtor de matriz: as matrizes devem estar alinhadas com a memória, portanto, se você especificar um
Deslocamento de byte, o valor deve ser um múltiplo do tamanho do seu tipo.O
O 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
bytes
Let Asints = new Int32Array (buffer);// Visto como
INTS assinado de 32 bits
deixe o lastk = novo uint8Array (buffer, 1023*1024);// Durar
Kilobyte como bytes
Seja 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 o
Número que você especifica.O código é exatamente o mesmo que seria com um
Array JavaScript regular, mas usando uint8array () em vez de
Array () faz o código funcionar mais de quatro vezes mais rápido e usar
Oito vezes menos memória nos meus testes.
Matrizes digitadas não são matrizes verdadeiras, mas reimpleem a maioria das matrizes
Métodos, para que você possa usá -los praticamente como você usaria regularmente
Matrizes:
deixe ints = new int16Array (10);// 10 números inteiros curtos
ints.fill (3) .map (x => x*x) .Join ("") // => "99999999999"
Lembre -se de que as matrizes digitadas têm comprimentos fixos, então o comprimento
A 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 conteúdo
de 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 mesmo
digite o que eles são chamados.
11.2.4 Métodos e propriedades de matriz digitada
Além dos métodos de matriz padrão, as matrizes digitadas também implementam um
poucos métodos próprios.O método set () define vários elementos
de uma matriz digitada de uma só vez, copiando os elementos de um regular ou digitado
Matriz em uma matriz digitada:

Let Bytes = novo UINT8Array (1024);// um buffer de 1k
deixe padrão = novo uint8array ([0,1,2,3]);// Uma matriz de 4
bytes
bytes.set (padrão);// copie -os para o início de outro
Array de bytes
bytes.set (padrão, 4);// copie -os novamente em um diferente
desvio
bytes.set ([0,1,2,3], 8);// ou apenas copiar valores diretos de um
Array regular
bytes.slice (0, 12) // => novo
Uint8array ([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
a 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 digitado
Array 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 buffer
bytes.buffer [1] // => 255: Isso apenas define um regular
Propriedade JS
bytes [1] // => 0: a linha acima não definiu
o byte
Vimos anteriormente que você pode criar um ArrayBuffer com o
ArrayBuffer () construtor e, em seguida, crie matrizes digitadas que usam
Esse buffer.Outra abordagem é criar uma matriz digitada inicial, então
Use o buffer dessa matriz para criar outras visualizações:
Let Bytes = novo UINT8Array (1024);// 1024 bytes
deixe ints = new uint32Array (bytes.buffer);// ou 256
Inteiros
Deixe Floats = new Float64Array (Bytes.Buffer);// ou 128
duplos
11.2.5 DataView e Endianness
Matrizes digitadas permitem que você visualize a mesma sequência de bytes em pedaços
de 8, 16, 32 ou 64 bits.Isso expõe o "endianness": a ordem em
Quais bytes são organizados em palavras mais longas.Para eficiência, digitada
As matrizes usam a endianidade nativa do hardware subjacente.Em Little-
Endian Systems, os bytes de um número são organizados em um ArrayBuffer
Do menos significativo ao mais significativo.Em plataformas grandes endianas, o
Os bytes são organizados de mais significativos e menos significativos.Você pode
determinar a endianidade da plataforma subjacente com código como
esse:
// Se o número inteiro 0x00000001 estiver organizado na memória como 01 00
00 00, então
// Estamos em uma plataforma pouco endiana.Em um grande endiano
Plataforma, 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.Muitos
Protocolos de rede e alguns formatos de arquivo binário exigem grande e-e-e-e-uns
pedidos de byte, no entanto.Se você estiver usando matrizes digitadas com dados que
veio da rede ou de um arquivo, você não pode apenas assumir que o
A plataforma Endianness corresponde à ordem de byte dos dados.Em geral,
Ao trabalhar com dados externos, você pode usar o Int8Array e
Uint8array para ver os dados como uma variedade de bytes individuais, mas você
Não deve usar as outras matrizes digitadas com tamanhos de palavras multibyte.
Em vez disso, você pode usar a classe DataView, que define métodos para
ler e escrever valores de um ArrayBuffer explicitamente
Pedido de byte especificado:
// Suponha que tenhamos uma variedade digitada de bytes de dados binários para
processo.Primeiro,
// criamos um objeto DataView para que possamos ler de maneira flexível e
escrever
// valores desses bytes
Let View = new DataView (bytes.buffer,
bytes.byteoffset,
bytes.byTeLength);
deixe int = view.getInt32 (0);// leia Big-Endian assinado int
de byte 0
int = view.getInt32 (4, false);// a seguir também é grande
Endian
int = view.getuint32 (8, true);// o próximo int é pouco endiano
e não assinado
view.setUint32 (8, int, false);// Escreva de volta em Big-
Formato Endiano
DataView define 10 métodos GET para cada uma das 10 matrizes digitadas
Aulas (excluindo o UINT8ClampedArray).Eles têm nomes como
getInt16 (), getUint32 (), 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
omitida ou é falsa, a ordem de bytes big-endian é usada.Se o segundo
O 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
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 fornece todas as ferramentas necessárias para
Processar dados binários e permitir que você escreva programas JavaScript que
fazer 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
Realize funções poderosas de correspondência de padrões e pesquisa e pesquisa

no texto.Para usar a API regexp de maneira eficaz, no entanto, você deve
Aprenda também a descrever padrões de texto usando a expressão regular
gramática, que é essencialmente uma mini linguagem de programação de seu
ter.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
Invista 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
Explique 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,
Expressão regular 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
mas 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 um
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 bandeira
Personagens que afetam como eles funcionam.As bandeiras são especificadas após o
segundo personagem de barra em literais regexp, ou como uma segunda sequência
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

Chara cter Partidas	Alfão Umérico Charact er	Em si
\ 0	O caractere nu (\ u0000)	
\ t	Guia (\ u0009)	
\ n	Newline (\ u000a)	
\ v	Guia vertical (\ u000b)	
\ f	Formulário de formulário (\ u000c)	
\ r	Retorno de carruagem (\ 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 regular
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
Esses personagens de pontuação literalmente em uma expressão regular, você deve
precede -os com um \.Outros personagens de pontuação, como citação
Marcas e @, não têm significado especial e simplesmente combine
eles mesmos literalmente em uma expressão regular.
Se você não consegue se lembrar exatamente de quais personagens de pontuação precisam ser
Escapou com uma barra de barra
caráter de pontuação.Por outro lado, observe que muitas letras e
Os números têm significado especial quando precedidos por uma barra de barriga, então qualquer
letras ou números que você deseja combinar literalmente não deve ser
escapou 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 negado
As classes também podem ser definidas;estes correspondem a qualquer personagem, exceto aqueles
contido 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 unicode
caráter de espaço em branco;\ S corresponde a qualquer personagem que não seja unicode
espaço em branco.A Tabela 11-2 lista esses personagens e resume
Sintaxe da classe de caracteres.(Observe que vários desses caráters
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

ara
cte
r
Partidas
[.
..
]
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 o
Regexp usa a bandeira S, então um período corresponde a qualquer caractere, incluindo linha
Terminadores.
\c
Qualquer personagem de palavra ascii.Equivalente a [A-ZA-Z0-9_].
\C
Qualquer personagem que não seja um personagem da palavra ascii.Equivalente a [^a-za-z0
-9_].
\ s
Qualquer caractere de espaço em branco Unicode.
\ S
Qualquer personagem que não seja Unicode WhiteSpace.
\ d
Qualquer dígito ASCII.Equivalente a [0-9].
\ D
Qualquer 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 quadrado
Suportes.\ s corresponde a qualquer caractere de espaço em branco e \ d corresponde a qualquer
Digit, SO / [\ S \ d] / corresponde a qualquer caractere ou dígito em branco.
Observe que há um caso especial.Como você verá mais tarde, o \ b escape
tem um significado especial.Quando usado em uma classe de personagem, no entanto, é
representa o caractere de backspace.Assim, para representar um backspace
personagem literalmente em uma expressão regular, use a classe de personagem com

Um elemento: /[\ b] /.
Classes de caracteres Unicode
No 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 nó, 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 dos
Os sistemas de escrita do mundo, você pode usar /\ p {decimal_number} /u.E se você quiser corresponder a algum
Um 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 e
Numerais romanos, você pode usar \ p {número}.Observe que "decimal_number" e "número" não são
específico para JavaScript ou para a Gramática de Expressão regular: é o nome de uma categoria de caracteres
definido pelo padrão Unicode.
A classe de caracteres \ w funciona apenas para texto ascii, mas com \ p, podemos aproximar um
Versão internacionalizada como esta:
/[\ p {alfabético} \ p {decimal_number} \ p {mark}]/u
(Embora seja totalmente compatível com a complexidade dos idiomas do mundo, precisamos realmente adicionar
As categorias também ?Connector_Puntuation? e ?junção_control?.)
Como exemplo final, a sintaxe \ p também nos permite definir expressões regulares que correspondem aos caracteres
De um determinado alfabeto ou script:
Seja greekLetter = /\ p {script = grego} /u;
Seja CyrillicLetter = /\ p {script = Cirillic} /u;
REPETIÇÃO
Com a sintaxe de expressão regular que você aprendeu até agora, você pode
descreva 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, um
Número que pode ter qualquer número de dígitos ou uma sequência de três letras
seguido de um dígito opcional.Esses padrões mais complexos usam regulares
Expressão 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 expressão

Char ACTER	Significado
{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 = /\ d {2,4} /;// combina entre dois e quatro dígitos
r = /\ w {3} \ d? /;// corresponde exatamente a três caracteres de palavras e
um dígito opcional
r = /\ s+java \ s+ /;// combina "Java" com um ou mais espaços
antes e depois
```

r = /[^(]* /; // corresponde a zero ou mais caracteres que são
não aberto parens
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
Defina um grupo com parênteses, que são explicados no seguinte
seções.
Tenha cuidado ao usar o * e?Personagens de repetição.Desde estes
Os personagens podem corresponder a zero instâncias de qualquer que seja os 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 regular
expressão para corresponder.Dizemos que essa repetição é "gananciosa".É também
Possível especificar que a repetição deve ser feita de maneira não-greedora.
Basta seguir o personagem ou personagens de repetição com uma pergunta
Mark: ??, +?, *?, ou mesmo {1,5}?.Por exemplo, o regular
expressão / 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 como
poucos caracteres conforme necessário.Quando aplicado à mesma string, este
O 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 o
partes da sequência de destino que correspondiam a qualquer parênteses em particular
Subpaterno.(Você verá como essas substringas correspondentes são obtidas
mais tarde nesta seção.) Por exemplo, suponha que você esteja procurando um ou
Mais letras minúsculas seguidas por um ou mais dígitos.Você pode usar
o padrão /[a-z]+\ d+ /.Mas suponha que você realmente se importe com o
dígitos no final de cada partida.Se você colocar essa parte do padrão em
parênteses (/[a-z]+(\ d+)/), você pode extrair os dígitos de qualquer
Matches 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 a
a posição da subexpressão entre parênteses dentro do regular
expressão.Por exemplo, \ 1 refere -se à primeira subexpressão e
\ 3 refere -se ao terceiro.Observe que, porque as subexpressões podem ser aninhadas
Dentro de outros, é a posição dos parênteses esquerdos que são contados.Em
a seguinte expressão regular, por exemplo, o aninhado
A subexpressão (crívio [SS]) é referida como \ 2:
/([Jj] ava ([ss] crip)?) \ Sis \ 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 um
Restrições que partes separadas de uma corda contêm exatamente o mesmo
caracteres.Por exemplo, a seguinte expressão regular corresponde a zero
ou mais caracteres dentro de citações únicas ou duplas.No entanto, não
requer as cotações de abertura e fechamento para combinar (ou seja, ambos solteiros
citações 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 fechamento
Citação corresponde à cotação de abertura.Esta expressão regular não permite
Citações únicas em seqüências de cordas duplas ou vice-versa.(Não é legal
Para usar uma referência em uma classe de caracteres, para que você não possa escrever:
/(['"]) [^\ 1]*\ 1/.)
Quando cobrimos a API regexp mais tarde, você verá que esse tipo de
Referência a uma subexpressão entre parênteses é uma característica poderosa de
Operações 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
C
h
um
r
um
c
t
e
r
Significado
|
Alternância: corresponda à subexpressão à esquerda ou à subexpressão ao
certo.
(
.
.
.
)
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.

\
n
Corresponder 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
formado com (?: não estão numerados.
Nomeados grupos de captura
O ES2018 padroniza um novo recurso que pode tornar as expressões regulares mais autodocumentadoras e
mais fácil 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 alguém
ler o código para entender mais facilmente o objetivo dessa parte da expressão regular.Como
No início de 2020, esse recurso é implementado em nó, 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+) (? <sate> [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 replipl () 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 correspondesse a uma string única ou dupla, onde as citações abertas e fechadas
teve que combinar.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 um
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 um
posição específica na sequência de pesquisa.A âncora mais usada
Os elementos são ^, que ligam 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
procure por "java" como uma palavra por si só (não como um prefixo, como está em
"JavaScript"), você pode experimentar o padrão /\ sjava \ s /, que requer um
espaço antes e depois da palavra.Mas há dois problemas com isso
solução.Primeiro, ele não corresponde a "java" no início ou no final de um
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) palavra
limites com \ b.A expressão resultante é /\ bjava \ b /.O
elemento \ b ancora a partida em um local que não é uma palavra
limite.Assim, o padrão / \ b [ss] crívio / 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 um
afirmação lookahead, e especifica que os personagens fechados devem
Combine, sem realmente combiná -los.Por exemplo, para corresponder ao
nome de uma linguagem de programação comum, mas apenas se for seguida por
um cólon, você poderia usar /[jj] ava ([ss] crívio)? (? = \ :) /.Esse
padrão corresponde à palavra "javascript" em "javascript: o definitivo
Guia ", mas não corresponde a" java "em" java em poucas palavras "porque
não é seguido por um cólon.
Se você apresentar uma afirmação com (?!, É um negativo
Afirmação 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 regular
C
h
ar
um
Ct
er
Significado
^
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.
\
b
Combine 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.)

\
B
Combine 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 LOLHEBEHIND
O ES2018 estende a sintaxe de expressão regular para permitir as afirmações "Lookbehind".Estes são como
LookaHead
Asserções, mas consulte o texto antes da posição atual da correspondência.No início de 2020, estes são
implementados
no nó, 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 um
zip de 5 dígitos
Código, 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 um
Afirmação negativa do LookBehind como esta:
/(?<!
Bandeiras
Toda expressão regular pode ter uma ou mais bandeiras associadas a ele
para alterar seu comportamento correspondente.JavaScript define seis bandeiras possíveis,
cada um dos quais é representado por uma única letra.As bandeiras são especificadas depois
o segundo / caráter de uma expressão regular literal ou como uma corda
passou como o segundo argumento para o construtor regexp ().O

Bandeiras suportadas e seus significados são:

g

A 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
do que 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 corresponde () e o regexp
método EXEC () de maneiras importantes.

eu

A bandeira I especifica que a correspondência de padrões deve ser caso
insensível.

m

A bandeira M especifica que a correspondência deve ser feita em "Multiline"
modo.Diz que o regexp será usado com cordas multilinas
e 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.

s

Como a bandeira M, a bandeira S também é útil ao trabalhar com texto
Isso inclui novas linhas.Normalmente, um "".em uma expressão regular
corresponde a qualquer caractere, exceto um terminador de linha.Quando a bandeira S é
usado, no entanto, "".vai corresponder a qualquer personagem, incluindo linha
Terminadores.A bandeira S foi adicionada ao JavaScript no ES2018 e, como
do início de 2020, é apoiado em nó, cromo, borda e safari, mas
Não Firefox.

u

A bandeira U significa Unicode e faz a expressão regular

Erro ao traduzir esta página.

11.3.2 Métodos de string para correspondência de padrões

Até 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 o
Métodos de string que usam expressões regulares para executar o padrão
Operações de correspondência e pesquisa e substituição.As seções que se seguem
Este continua a discussão sobre o padrão de correspondência com JavaScript
Expressões 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) // => -1

Se 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 regular
argumento 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 necessária
corresponde ao padrão especificado.Se a expressão regular tiver o g
Conjunto 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 muito
recurso 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
Calcule 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
capturado por esses grupos é passado como argumentos.O próximo argumento
é a posição dentro da sequência na qual a partida foi encontrada.Depois
Isso, a sequência inteira que substitui () foi chamada é 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 os
texto correspondente.Como exemplo, aqui está o código que usa uma substituição
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)) // => "f
vezes 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 (/\ 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 do
Array retornado é a string correspondente e quaisquer elementos restantes são
as substringas que correspondem aos grupos de captura entre parênteses
expressão regular.Assim, se a correspondência () retornar uma matriz A, a [0]
contém a correspondência completa, a [1] contém a substring que corresponde
A 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
O mesmo que US $ 2, e assim por diante.
Por exemplo, considere analisar um URL com o seguinte código:

```
// Um ??URL muito simples parsing regexp
vamos url = /(\w+):\/\/([\w. ]+)\/(\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"
}
```

Neste caso não global, a matriz retornada por match () também tem alguns
Propriedades do objeto Além dos elementos de matriz numerados.O
A propriedade de entrada refere -se à string na qual correspondência () foi chamada.
A propriedade do índice é a posição dentro daquela string na qual o
a partida começa.E se a expressão regular contiver captura nomeada
grupos, então a matriz retornada também tem uma propriedade de grupos cuja
valor é um objeto.As propriedades deste objeto correspondem aos nomes do
Grupos nomeados e os valores são o texto correspondente.Poderíamos reescrever
O 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 // => texto
match.index // => 17
match.groups.protocol // => "http"
match.groups.host // => "www.example.com"
match.groups.path // => "~ David"
Vimos que a correspondência () se comporta de maneira bastante diferente, dependendo de
se o regexp tem o conjunto de sinalizadores G ou não.Também existem importantes
Mas diferenças menos dramáticas no comportamento quando o sinalizador Y é definido.Lembrar
que a bandeira y faz uma expressão regular "pegajosa" restringindo
onde nas correspondências da string pode começar.Se um regexp tem o G e
Y Sinalizadores definidos, depois corresponde () retorna uma variedade de cordas correspondentes, exatamente como
Faz quando G é definido sem y.Mas a primeira partida deve começar no
início da string, e cada partida subsequente deve começar no
Personagem 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 do
corda.Você pode alterar esta posição de início de correspondência padrão, no entanto, por
Definindo a propriedade LastIndex do objeto regexp no índice em
em que você deseja combinar.Se uma partida for encontrada, então este LastIndex
será atualizado automaticamente para o primeiro caractere após a partida, então se
você liga para match () novamente, neste caso, ele procurará um subsequente
corresponder.(Lastindex pode parecer um nome estranho para uma propriedade que
Especifica a posição na qual iniciar a próxima partida.Vamos ver isso
Novamente quando cobrimos o método regexp exec (), e seu nome pode
faça mais sentido nesse contexto.)

```
Seja vogal = /[aeiou] /y;// partida de vogal pegajosa
"teste" .match (vogal) // => null: "teste" não começa
com uma vogal
vogal.LastIndex = 1;// Especifique uma correspondência diferente
posição
"teste" .Match (vogal) [0] // => "e": encontramos uma vogal em
Posição 1
vogal.lastindex // => 2: o LastIndex foi automaticamente
atualizado
"Teste" .Match (vogal) // => nulo: nenhuma vogal na posição 2
vogal.lastindex // => 0: LastIndex é redefinido após
falhou correspondência
```

Vale 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
método exec () da expressão regular: a matriz retornada e seu
As 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 o
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 {alfabético}+\ 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 e
Isso torna muito menos a probabilidade de causar bugs em seu código.

DIVIDIR()

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 substringas, 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.Aqui
Nós o chamamos com um separador que inclui uma quantidade arbitrária de
Espaço 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 o
A expressão regular inclui capturar grupos, depois o texto que corresponde
Os 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 Regexp
Esta seção documenta o construtor regexp (), as propriedades de
Instâncias 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 um
string 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, é
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 o
Expressão regular sintaxe literal.Por exemplo, para procurar uma string
Digido 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
Passe também um objeto regexp.Isso permite que você copie um regular
expressão e altere suas bandeiras:
Let ExactMatch = /JavaScript /;
Deixe CaseInsensitive = novo regexp (exatmatch, "i");
Propriedades regexp
Os objetos regexp têm as seguintes propriedades:

fonte
Esta propriedade somente leitura é o texto de origem da expressão regular:
Os personagens que aparecem entre as barras em um literal regexp.
bandeiras
Esta propriedade somente leitura é uma string que especifica o conjunto de letras
que representam os sinalizadores para o regexp.
global
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador G estiver definido.
ignorecase
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador I estiver definido.
Multilina
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador M estiver definido.
Dotall
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador S estiver definido.
unicode
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador U estiver definido.
pegajoso
Uma propriedade booleana somente leitura que é verdadeira se o sinalizador Y estiver definido.
LastIndex
Esta propriedade é um número inteiro de leitura/gravação.Para padrões com o g ou y
bandeiras, 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 um
expressão regular.É preciso um único argumento de string e retorna verdadeiro
Se 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 se
EXEC () 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 Regexp
Reutilize ?para mais detalhes.
Exec ()
O método regexp EXEC () é a maneira mais geral e poderosa de
Use 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 partida
é encontrado, no entanto, ele retorna uma matriz exatamente como a matriz devolvida pelo
Match () Método para pesquisas não globais.Elemento 0 da matriz
contém a string que correspondia à expressão regular e qualquer
Os elementos de matriz subsequentes contêm as substringas que correspondiam a qualquer
captura de grupos.A matriz devolvida também possui propriedades: o
A 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 que
mantém as substringas que correspondem aos grupos de captura qualquer nome.

Ao contrário do método String Match (), EXEC () retorna o mesmo tipo de
Array 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 global
expressão 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 o
Conjunto de bandeira Y Sticky, ele consulta a propriedade LastIndex do regexp
Objeta para determinar onde começar a procurar uma correspondência.(E se o y
Flag está definido, também restringe a partida para começar nessa posição.) Para um
Objeto regexp recém -criado, LastIndex é 0 e a pesquisa começa
No início da string.Mas cada vez executiva () encontra com sucesso um
corresponder, ele atualiza a propriedade LastIndex para o índice do
Personagem imediatamente após o texto correspondente.Se Exec () não encontrar um
corresponde, ele redefine o LastIndex para 0. Este comportamento especial permite que você
ligue para o Exec () repetidamente para percorrer todo o regular
A expressão corresponde a uma string.(Embora, como descrevemos, em
ES2020 e, posterior, o método matchall () da string é uma maneira mais fácil
Para fazer uma pancada em todas as partidas.) Por exemplo, o loop no seguinte
O 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 do
A 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 infinito
posições.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 um
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
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 DoublletterWords = [];
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 no
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 particular 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 horários

A 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 atual
```

Se você passar um argumento numérico, a data () interpreta o construtor

Esse argumento como o número de milissegundos desde a época de 1970:

```
Seja epoch = nova data (0);// meia -noite, 1º de janeiro de 1970, GMT
```

Se você especificar dois ou mais argumentos inteiros, eles são interpretados como

O ano, mês, dia de mês, hora, minuto, segundo e milissegundos

no seu fuso horário local, como no seguinte:

```
Let Century = New Date (2100, // Ano 2100
0, // janeiro
1, // 1º
2, 3, 4, 5);// 02: 03: 04.005, local
tempo
```

Uma 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, o

DATE () Constructor Padrates todos para 0, definindo o tempo para

meia-noite.

Observe que, quando invocado com vários números, a data ()

construtor os interpreta usando qualquer fuso horário o local
O 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 o
Date.utc ().Este método estático leva os mesmos argumentos que o
Date () construtor, os interpreta no UTC e retorna um
Milissegund Timestamp que você pode passar para o construtor DATE ():
// meia -noite na Inglaterra, 1 de janeiro de 2100
Let Century = New Date (Date.utc (2100, 0, 1));
Se você imprimir uma data (com console.log (século), por exemplo), ele
Por padrão, será impresso no seu fuso horário local.Se você quiser
exibir uma data no UTC, você deve convertê -lo explicitamente em uma string com
toutcString () ou ToisSotring ().
Finalmente, se você passar uma string para o construtor date (), ele tentará
Para analisar essa string como especificação de data e hora.O construtor pode
datas de parse especificadas nos formatos produzidos pelo tostring (),
Métodos ToutcString () e ToisSoString ():
Let Century = New Date ("2100-01-01T00: 00: 00Z");// um ISO
Data de formato
Depois de ter um objeto de data, vários métodos de obtenção e definição permitem que você
consultar e modificar o ano, mês, dia de mês, hora, minuto,
segundo, e milissegundos campos da data.Cada um desses métodos tem
duas formas: uma que recebe ou conjunta usando a hora local e uma que recebe ou
conjuntos usando o tempo UTC.Para obter ou definir o ano de um objeto de data, para
Por exemplo, você usaria o getlyear (), getutclyear (),
setlyear () ou setutclyear ():

Seja d = new Date ();// Comece com o
data atual
d. sinerlyear (d.getlyear () + 1);// incremento o ano
Para obter ou definir os outros campos de uma data, substitua "totalmente" no
Nome do método com "mês", "data", "horas", "minutos", "segundos",
ou "milissegundos".Alguns dos métodos conjuntos de data permitem que você defina mais
do que um campo de cada vez.setlyear () e
Setutclyear () também permite opcionalmente que você defina o mês e
Dia de mês também.E sethours () e setutchours ()
Permita que você especifique os campos de atas, segundos e milissegundos em
adição ao campo de horas.
Observe que os métodos para consultar o dia do mês são GetDate ()
e getutcdate ().As funções mais naturais
getDay () e getutcday () retornam o dia da semana (0 para domingo
até 6 para sábado).O dia da semana é somente leitura, então não há um
Método SetDay () correspondente.
11.4.1 Timestamps
JavaScript representa datas internamente como números inteiros que especificam o
Número 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 de
270.000 anos.
Para qualquer objeto de data, o método gettime () retorna este interno
Value e o método setTime () o define.Então você pode adicionar 30 segundos
Para uma data com código como este, por exemplo:

D.setTime (D.GetTime () + 30000);
Esses valores de milissegundos às vezes são chamados de timestamps, e é
às vezes útil para trabalhar com eles diretamente, e não com a data
objetos.O método estático date.now () retorna a hora atual como um
Timestamp e é útil quando você deseja medir quanto tempo
O código leva para executar:
Seja starttime = date.now ();
reticulatesplines ();// Faça alguma operação demorada
deixe endtime = date.now ();
Console.log (`reticulação spline levou $ {EndTime -
startTime} ms.`);
Timestamps de alta resolução
Os registros de data e hora retornados por date.now () são medidos em milissegundos.Um milissegundo é na verdade um
relativamente longo para um computador e às vezes você pode medir o tempo decorrido com maior
precisão.A função performance.now () permite isso: também retorna um milissegundo baseado
Timestamp, mas o valor de retorno não é um número inteiro, por isso inclui frações de um milissegundo.O valor
Retornado por performance.now () não é um registro de data e hora absoluto como o valor DAT.now ().
Em vez
O processo do nó foi iniciado.
O objeto de desempenho faz parte de uma API de desempenho maior que não é definida pelo ecmascript
padrão, mas é implementado por navegadores da web e por nó.Para usar o objeto de desempenho
No 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, então
Os navegadores (principalmente o Firefox) podem reduzir a precisão do desempenho.now () por padrão.Como uma web
Desenvolvedor, você deve ser capaz de reativar o tempo de alta precisão de alguma forma (como definir
Privacy.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
datas.(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 hora
Como 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 eles
têm 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 e
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 nós
Defina o dia do mês como um valor maior que o número de dias em
O 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 data
classe define vários métodos diferentes para converter objetos de data
para cordas.Aqui estão alguns exemplos:
Seja d = nova data (2020, 0, 1, 17, 10, 30);// 17:10:30 no novo
Dia do ano 2020
d.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' Locale
d.tolocaletimestring () // => "17:10:30": 'en-us' Locale
D.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 é
apropriado 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 geral
Técnica de formatação para propósito e data de conhecimento e local.
Finalmente, além desses métodos que convertem um objeto de data em um
String, há também um método estático de data.parse () que leva uma string
Como argumento, tentativas de analisá -lo como uma data e hora, e retorna um
Timestamp representando essa data.Date.parse () é capaz de analisar o
mesmas cordas que o construtor DAT () pode e é garantido
Capaz 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 o
classe de erro base, cada uma dessas subclasses tem um construtor que leva um
argumento de mensagem única.E instâncias de cada uma dessas subclasses
Tenha 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 melhor
encapsular as condições de erro do seu próprio programa.Observe que você
não estão limitados ao nome e propriedades da mensagem.Se você criar um
Subclasse, você pode definir novas propriedades para fornecer detalhes de erro.Se você
estão escrevendo um analisador, por exemplo, você pode achar útil definir um
Classe de parseerror com propriedades de linha e coluna que especificam o
Localização exata da falha da análise.Ou se você estiver trabalhando com http
Solicitações, você pode querer definir uma classe httperror que tenha um
propriedade de status que contém o código de status HTTP (como 404 ou 500)
da solicitação fracassada.
Por exemplo:
classe httperror estende o erro {
construtor (status, statustext, url) {
super (`$ {status} $ {statustext}: $ {url}`);
this.status = status;
this.statustustext = statustext;

this.url = url;
}
get name () {return "httperror";}
}
Deixe erro = novo httperror (404, "não encontrado",
"http://example.com/");
error.status // => 404
error.message // => "404 não encontrado:
http://example.com/ "
error.name // => "httperror"
11.6 Serialização e análise de JSON JSON
Quando um programa precisa salvar dados ou transmitir dados em um
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 original
estruturas de dados de memória.Este processo de conversão de estruturas de dados em
fluxos de bytes ou caracteres são conhecidos como serialização (ou marechaling
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
e 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
Coberto brevemente em §6.8.Dado um objeto ou matriz (aninhado arbitrariamente
profundamente) que não contém valores não -semerializáveis ??como regexp
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]} '
Seja copy = json.parse (s);// copy == {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
prefixo uma string json com var dados = e passe o resultado para avaliar (), você vai
Obtenha 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 execute seu código.É mais rápido e seguro usar apenas json.parse () para
Decode dados formatados por JSON.
Às vezes, o JSON é usado como um formato de arquivo de configuração legível pelo homem.Se você
Encontre-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
Incluído 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
descrito a seguir.Json.stringify () também leva um terceiro opcional
argumento que discutiremos primeiro.Se você gostaria do 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
O terceiro argumento é uma série de espaço em branco (como '\ t'), ele usará isso
string 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
Rigifica o valor de retorno no lugar do valor original.Objetos de data
implementar tojson (): ele retorna a mesma string que
O método ToisSotring () faz.Isso significa que se você serializar um 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 que o objeto original 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
invocado uma vez para cada valor primitivo (mas não os objetos ou matrizes que
contêm esses valores primitivos) analisado na sequência de entrada.O
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 objeto com essa palavra -chave.
O valor de retorno da função Reviver se torna o novo valor do
propriedade nomeada.Se retornar seu segundo argumento, a propriedade irá

permanecer inalterado.Se retornar indefinido, então a propriedade nomeada
será excluído do objeto ou matriz antes de json.parse ()
retorna 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
converta -o em uma data.
if (typeof value === "string" &&
/^\ d \ 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, o
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
Especificado, a função Replacer é invocada para que cada valor seja
straciificado.O primeiro argumento para a função Replacer é o objeto
nome da propriedade ou índice de matriz do valor dentro desse objeto, e o
O segundo argumento é o próprio valor.A função substituta é invocada como
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
serializá -los em
Deixe o texto = json.stringify (endereço,
["cidade", "estado", "país"]);
// Especifique uma função de substituição que omite o valor regexp
propriedades
Seja json = json.stringify (o, (k, v) => v instância de regexp?
indefinido: v);
As duas chamadas JSON.Stringify () aqui usam o segundo argumento em
uma maneira benigna, produzindo saída serializada que pode ser desserializada
sem exigir uma função de reviver especial.Em geral, porém, se você
Defina um método tojson () para um tipo, ou se você usar um substituto
função que na verdade substitui valores não -semerializáveis ??por serializável
o outro, então você normalmente precisará usar uma função de reviver personalizada com
Json.parse () para recuperar sua estrutura de dados original.Se você fizer
Você deve entender que está definindo um formato de dados personalizado
e sacrificar a portabilidade e compatibilidade com um grande ecossistema de

Erro ao traduzir esta página.

chamado de "números árabes" 0 a 9 são usados ??em muitos idiomas, este
não é universal, e os usuários em alguns países esperam ver números
escrito usando os dígitos de seus próprios scripts.
A classe intl.numberFormat define um método format () que leva
Todas essas possibilidades de formatação em consideração.O construtor toma
dois argumentos.O primeiro argumento especifica o local que o número
deve ser formatado e o segundo é um objeto que especifica mais
Detalhes sobre como o número deve ser formatado.Se o primeiro argumento
é omitido ou indefinido, então o local do sistema (que assumimos
para ser o local preferido do usuário) será usado.Se o primeiro argumento for um
string, especifica um local desejado, como "en-us" (inglês como usado
Nos Estados Unidos), "FR" (francês) ou "Zh-Hans-Cn" (chinês,
usando o sistema de escrita Han simplificado, na China).O primeiro argumento
também pode ser uma variedade de cordas de localidade e, neste caso,
Intl.NumberFormat escolherá o mais específico que está bem
suportado.
O segundo argumento para o construtor intl.numberFormat ()
especificado, deve ser um objeto que defina um ou mais dos seguintes
propriedades:
estilo
Especifica o tipo de formatação numérica necessária.O
O padrão é "decimal".Especifique "porcentagem" para formatar um número
como uma porcentagem ou especificar "moeda" para especificar um número como um
quantidade de dinheiro.
moeda

Se o estilo for "moeda", esta propriedade é necessária para especificar
O código de moeda ISO de três letras (como "USD" para dólares americanos
ou "GBP" para libras britânicas) da moeda desejada.
MoedaDisplay
Se o estilo é "moeda", esta propriedade especifica como o
A moeda é exibida.O valor padrão "símbolo" usa um
Símbolo da moeda se a moeda tiver uma.O valor "código" usa
O código ISO de três letras e o valor "nome" soletram o
nome da moeda em forma longa.
useGrouping
Defina esta propriedade como false se você não quiser que os números tenham
milhares de separadores (ou seus equivalentes apropriados para localidade).
MinimintegerDigits
O número mínimo de dígitos a serem usados ??para exibir a parte inteira de
o número.Se o número tiver menos dígitos do que isso, será
acolchoado à esquerda com zeros.O valor padrão é 1, mas você pode
Use valores até 21.
MINIMUMFRACTIONDDIGITS, MAXIMUMFRACTIONDDIGITS
Essas duas propriedades controlam a formatação da parte fracionária de
o número.Se um número tiver menos dígitos fracionários do que o
Mínimo, ele será acolchoado com zeros à direita.Se tiver mais
do que o máximo, então a parte fracionária será arredondada.Jurídico
Os valores para ambas as propriedades estão entre 0 e 20. O padrão
o mínimo é 0 e o máximo padrão é 3, exceto quando
formatando quantidades monetárias, quando o comprimento do fracionário
A parte varia dependendo da moeda especificada.
MinimumsignificantDigits,
MaximumsInSignificAntDDigits

Essas propriedades controlam o número de dígitos significativos usados ??quando
formatando um número, tornando -os adequados ao formatar
Dados científicos, por exemplo.Se especificado, essas propriedades substituem
as propriedades inteiras e de dígitos fracionários listadas anteriormente.Jurídico
Os valores estão entre 1 e 21.
Depois de criar um objeto intl.NumberFormat com o desejado
Local 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, espanhol
formatação
Deixe Pounds = intl.NumberFormat ("EN", {style: "Moutrency",
moeda: "GBP"});
libras.format (1000) // => "£ 1.000,00": mil libras,
Formatação em inglês
Uma característica útil do intl.NumberFormat (e as outras classes INTL
Além disso) é que seu método formato () está ligado ao número de formação
objeto ao qual pertence.Então, em vez de definir uma variável que se refere
para o objeto de formatação e depois invocar o método format ()
isso, você pode apenas atribuir o método format () a uma variável e usá -lo
Como 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-US
Alguns idiomas, como o árabe, usam seu próprio script para decimal
dígitos:
Seja árabe = intl.NumberFormat ("AR", {useGrouping:
false}). formato;
árabe (1234567890) // => "??????????"
Outros idiomas, como o hindi, usam um script que tem seu próprio conjunto de
dígitos, mas tendem a usar os dígitos ASCII 0?9 por padrão.Se você quiser
substituir o script padrão usado para dígitos, adicione -u-nu- ao local
e siga -o com um nome de script abreviado.Você pode formatar números
com agrupamento de estilo indiano e dígitos de devanagari como este, para
exemplo:
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 curto
Para Devanagari.O padrão da API INTL define nomes para vários
outros sistemas de numeração, principalmente para os idiomas indicados do sul e
Sudeste Asiático.
11.7.2 Datas e horários de formatação
A classe Intl.DateTimeFormat é muito parecida com o intl.numberFormat
aula.O construtor intl.dateTeMormat () leva o mesmo
Dois argumentos que o intl.numberFormat () faz: um local ou matriz
de locais e um objeto de opções de formatação.E a maneira como você usa um

Erro ao traduzir esta página.

mês
Use "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 altamente
Nome abreviado como "J" que não é garantido para ser único.
dia
Use "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 semana
Use "Long" para um nome completo como "segunda -feira", "curto" para um
nome abreviado como "seg" e "estreito" para um altamente
Nome abreviado como "M" que não é garantido para ser único.
era
Esta propriedade especifica se uma data deve ser formatada com um
Era, como CE ou BCE.Isso pode ser útil se você estiver formatando
datas de muito tempo atrás ou se você estiver usando um calendário japonês.
Os valores legais são "longos", "curtos" e "estreitos".
hora, minuto, segundo
Essas propriedades especificam como você gostaria de ser exibido.Usar
"numérico" para um campo de um ou dois dígitos ou "2 dígitos" para forçar
números de um dígito a serem acolchoados à esquerda com 0.
fuso horário
Esta propriedade especifica o fuso horário desejado para o qual a data
deve ser formatado.Se omitido, o fuso horário local é usado.
As implementações sempre reconhecem "UTC" e também podem reconhecer
Nomes 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 isso
propriedade.
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
Especifique 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
Essa 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.dateTimeFormat ("en-us"). Formato (d) // => "1/2/2020"
Intl.dateTimeFormat ("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.dateTimeFormat ("en-us", opts) .Format (d) // => "Quinta-feira, quinta-feira,

2 de janeiro de 2020 "
Intl.DateTimeFormat ("es-es", opts) .Format (d) // => "Jueves, 2
de Enero de 2020 "
// O tempo em Nova York, para um canadense de língua francesa
opts = {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 do
Calendário Juliano Padrão com base na era cristã.Embora alguns
Os locais podem usar um calendário não-cristão por padrão, você sempre pode
Especifique explicitamente o calendário a ser usado adicionando -u-ca- ao local
e seguindo isso com o nome do calendário.Calendário possível
Os nomes incluem "budista", "chinês", "copta", "etiópico", "gregory",
"Hebraico", "Indiano", "Islâmico", "Iso8601", "Japonês" e "Persa".
Continuando o exemplo anterior, podemos determinar o ano em
Vários calendários não-cristãos:
Seja opts = {ano: "numérico", ERA: "Short"};
Intl.dateTimeFormat ("en", opts) .Format (d) //
=> "2020 AD"
Intl.dateTimeFormat ("en-u-ca-isto8601", opts) .Format (d) //
=> "2020 AD"
Intl.dateTimeFormat ("en-u-ca-hebrew", opts) .Format (d) //
=> "5780 AM"
Intl.dateTimeFormat ("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.dateTimeFormat ("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 alfabéticos) 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, para
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 ch
e 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 encontrarão
Natural, não é suficiente usar o método sTEL () 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 o local-
Classificação apropriada das cordas.Os objetos intl.Collator podem ser
configurado para que o método compare () execute o caso insensível
comparações ou mesmo comparações que apenas consideram a letra base e
Ignore detalhes e outros diacríticos.

Como intl.numberFormat () e intl.dateTimeFormat (),
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:

uso
Esta propriedade especifica como o objeto Collator deve ser usado.O
O 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.
sensibilidade
Esta propriedade especifica se o colatador é sensível à letra
Case 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
Os idiomas consideram certos personagens acentuados como base distinta
cartas.) "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 o
A sensibilidade padrão depende do local.
ignorepuncção
Defina esta propriedade como True para ignorar espaços e pontuação quando
Comparando strings.Com esta propriedade definida como true, as cordas ?qualquer
Um "e" qualquer pessoa ", por exemplo, serão considerados iguais.
numérico
Defina 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
Especifique "Upper", então "A" classificará antes de "A".E se você
Especifique "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 do unicode
Ordenação lexicográfica (o comportamento padrão da matriz ()
método) em que todas as letras maiúsculas ascii vêm antes de todas
LETRAS ASCII PORTUMAS.O padrão para esta propriedade é localidade
dependentes e implementações podem ignorar esta propriedade e não
Permita 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 após a segunda sequência.E 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
liga automaticamente o método compare () à sua instância, então você
pode passar diretamente para classificar () sem ter que escrever um invólucro
Função e invoco através do objeto Collator.Aqui estão alguns
Exemplos:
// Um ??comparador 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",
Ignorepunctionation: 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 pode ser selecionado através do argumento das opções do Intl.Collator, mas elas
pode 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 console

Você viu a função console.log () usada ao longo disso
Livro: Em navegadores da web, ele imprime uma string na guia "Console" do
Painel de ferramentas de desenvolvedor do navegador, que pode ser muito útil quando
depuração.No nó, console.log () é uma saída de uso geral
função e imprime seus argumentos para o fluxo de stdout do processo, onde
Normalmente, aparece para o usuário em uma janela de terminal como saída do programa.
A API do console define uma série de funções úteis, além de
console.log ().A API não faz parte de nenhum padrão ECMAScript,
mas é apoiado pelos navegadores e por nó e tem sido formalmente
Escrito 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 seu
Argumentos para as cordas e produzem para o console.Inclui
espaços entre os argumentos e inicia uma nova linha após a saída
todos 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
função 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 ?afirmação
falhou ?prefixo.Observe que, diferentemente das funções típicas assert (),
console.assert () não joga uma exceção quando um
A 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 nó
programas que precisam produzir saída que resume os dados.
console.table () tenta exibir seu argumento no tabular
formulário (embora, se não puder fazer isso, exiba -o usando regular
console.log () formatação).Isso funciona melhor quando o
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 um
Matriz 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
Torne -se 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 pode
Seja útil ao depurar um manipulador de eventos, por exemplo, se você
precisa acompanhar quantas vezes o manipulador de eventos tem sido
provocado.
console.countreset ()
Esta função leva um argumento de string e redefine o contador para isso
corda.
console.Group ()
Esta função imprime seus argumentos para o console como se tivessem sido
passou para console.log (), então define o estado interno do
console para que todas as mensagens subsequentes do console (até o próximo
console.groupend () chamada) será recuado em relação ao
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
entrou 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 web
Navegadores, o grupo será "colapso" por padrão e o
As 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 do
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
foi passado anteriormente para console.time (), então ele imprime
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
passou para console.log ().
console.TimeEnd ()
Esta função leva um único argumento de string.Se esse argumento tivesse
foi passado anteriormente para console.time (), então ele imprime
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 console

Funções 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, %O ou %C, então este primeiro argumento é tratado como
String de formato e os valores dos argumentos subsequentes são substituídos
na corda no lugar das seqüências de dois caracteres.

Os significados das seqüências são os seguintes:

%s

O argumento é convertido em uma string.

%i e %d

O argumento é convertido em um número e depois truncado para um
Inteiro.

%f

O argumento é convertido em um número

%O e %O.

O argumento é tratado como um objeto, e nomes de propriedades e
Os valores são exibidos.(Nos navegadores da web, essa tela é normalmente
interativo, e os usuários podem expandir e colapsar propriedades para explorar
uma estrutura de dados aninhada.) %O e %o Ambos os detalhes do objeto exibem.O
A variante da maçaneta usa um formato de saída dependente da implementação
Isso é considerado mais útil para desenvolvedores de software.

%c
Nos navegadores da web, o argumento é interpretado como uma série de CSS
estilos e usado para estilizar qualquer texto a seguir (até o próximo %c
Sequência ou o final da string).No nó, a sequência %C e seu
O argumento correspondente é simplesmente ignorado.
Observe que geralmente não é necessário usar uma string de formato com o
Funções de console: geralmente é fácil obter uma produção adequada por simplesmente
passando um ou mais valores (incluindo objetos) para a função e
permitindo que a implementação os exiba de uma maneira útil.Como um
exemplo, observe que, se você passar um objeto de erro para console.log (), ele
é impresso automaticamente junto com seu rastreamento de pilha.
11.9 URL APIs
Como JavaScript é tão comumente usado em navegadores da web e web
Servidores, é comum o código JavaScript precisar manipular URLs.
A classe URL analisa URLs e também permite modificação (adicionando
Parâmetros de pesquisa ou caminhos de alteração, por exemplo) dos URLs existentes.Isto
também lida adequadamente o tópico complicado de escapar e descontagem
os vários componentes de um URL.
A classe URL não faz parte de nenhum padrão ECMAScript, mas funciona em
Nó 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 absoluto
String de URL como o argumento.Ou passar um URL relativo como o primeiro
argumento e o URL absoluto de que é relativo como o segundo

argumento.Depois de criar o objeto URL, seus vários
Propriedades permitem que você consulte versões unespadas das várias partes
do 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 um
nome de usuário e senha, e a classe URL pode analisar esses URL
componentes 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 URL
Protocolo e host (incluindo a porta, se for especificado).Como tal, é um
propriedade somente leitura.Mas cada uma das outras propriedades demonstradas em
O exemplo anterior é lido/gravação: você pode definir qualquer uma dessas propriedades
Para definir a parte correspondente do URL:
vamos url = new url ("https://example.com");// Comece com nosso

servidor
url.pathname = "API/pesquisa";// Adicione um caminho para
um endpoint da API
url.search = "q = teste";// Adicione uma consulta
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
New String reencontre 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
consulte 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
caráter hash.Às vezes, é suficiente apenas tratar isso como um único
Propriedade da URL.Freqüentemente, no entanto, as solicitações HTTP codificam os valores de
vários campos de forma ou vários parâmetros da API na parte de consulta
de um URL usando o aplicativo/x-www-forma-urlicoded
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 que
Uma 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 consulta
parte de um URL, então a propriedade SearchParams será mais
útil que a propriedade de pesquisa.A propriedade de pesquisa é uma
Leia/escreva string que permite obter e defina toda a parte de consulta do
Url.A propriedade SearchParams é uma referência somente leitura a um
Objeto URLSearchParams, que tem uma API para obter, configurar,
Adicionando, excluindo e classificando os parâmetros codificados na consulta
parte do URL:
vamos url = new url ("https://example.com/search");
url.search // => "": sem consulta ainda
url.searchparams.append ("q", "termo");// Adicione uma pesquisa
parâmetro
url.search // => "? q = termo"
url.searchparams.set ("q", "x");// Alterar o valor de
este parâmetro
url.search // => "? q = x"
url.searchparams.get ("q") // => "x": consulte o
valor do parâmetro
url.searchparams.has ("q") // => true: existe um
q parâmetro
url.searchparams.has ("p") // => false: existe
sem parâmetro p
url.searchparams.append ("opts", "1");// Adicione outra pesquisa
parâmetro
url.search // => "? q = x & opts = 1"
url.searchparams.append ("opts", "&");// Adicione outro valor
Para o mesmo nome
url.search // => "?
Q = X & OPTS = 1 & OPTS =%26 ": Nota Escape
url.searchparams.get ("opts") // => "1": o primeiro
valor

Erro ao traduzir esta página.

funções de escape () e unescape (), que agora estão preteridas
mas ainda amplamente implementado.Eles não devem ser usados.
Quando Escape () e UNESCAPE () foram obsoletos, ecmascript
Introduziu dois pares de funções globais alternativas:
codeuri () e decodeuri ()
codeuri () pega uma string como seu argumento e retorna um novo
string 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 %xx
Sequência de fuga, onde xx são dois dígitos hexadecimais.Porque
codeuri () destina -se a codificar URLs inteiros, ele não
Escape URL Separator caracteres como /,? e #.Mas isso
significa que o codeuri () não pode funcionar corretamente para os URLs que
Tenha esses personagens em seus vários componentes.
codeuricomponent () e decodeuricomponent ()
Este par de funções funciona como o codeuri () e
decodeuri (), exceto que eles pretendem escapar do indivíduo
componentes de um URI, então eles também escapam de personagens como /,? e
que são usados ??para separar esses componentes.Estes são os mais
Útil das funções do URL legado, mas esteja ciente de que
codeuricomponent () escapará / caracteres em um caminho
Nome que você provavelmente não deseja escapar.E vai converter
espaços em um parâmetro de consulta para %20, embora os espaços sejam
deveria ser escapar com A + naquela parte de um URL.
O problema fundamental de todas essas funções legadas é que elas
procure 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
Use 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 dois
funções - setTimeout () e setInterval () - que permitem
programas 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 nó
e 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
decorre 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 chamado sem argumentos.Aqui, por exemplo, são três
setTimeout () chama as 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
retornando.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
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 o
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 dados simples
estruturas e o objeto de console para registrar mensagens.

1
Nem tudo documentado aqui é definido pela especificação do idioma JavaScript:
Algumas das classes e funções documentadas aqui foram implementadas pela primeira vez na web
navegadores e depois adotados por nó, tornando -os membros de fato do JavaScript
Biblioteca padrão.
2
Esta ordem de iteração previsível é outra coisa sobre os conjuntos de JavaScript que Python
Os programadores podem achar surpreendente.
3
Matrizes digitadas foram introduzidas pela primeira vez no JavaScript do lado do cliente quando os navegadores da Web adicionaram
Suporte para gráficos WebGL.O que há de novo no ES6 é que eles foram elevados a um
Recurso do idioma central.
4
Exceto dentro de uma classe de caracteres (colchetes quadrados), onde \ b corresponde ao backspace
personagem.
5
Analisar URLs com expressões regulares não é uma boa ideia.Veja §11.9 para um mais robusto
Analisador de URL.
6
C 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 acabou - 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 do
Pares, você pode usar os métodos Keys () e valores ():
[... m] // => [["One", 1], ["dois", 2]]: Padrão
iteração
[... M.Entries ()] // => [["One", 1], ["dois", 2]]: entradas ()
Método é o mesmo
[... M.Keys ()] // => ["One", "Two"]: Keys () Método
itera apenas as teclas de mapa
[... M.Values ??()] // => [1, 2]: Values ??() Método itera apenas
Valores do mapa
Finalmente, várias funções e construtores internos que são
comumente usado com objetos de matriz são realmente escritos (em ES6 e
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 o básico
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 iterável
objetos, mas vale a pena entender o que está realmente acontecendo com
fazer a iteração funcionar.Existem três tipos separados que você precisa
Entenda para entender a iteração em JavaScript.Primeiro, há o
Objetos iteráveis: esses são tipos como matriz, conjunto e mapa que podem ser
iterado.Segundo, existe o próprio objeto de iterador, que executa o
iteração.E terceiro, existe o objeto de resultado da iteração que mantém o
resultado de cada etapa da iteração.
Um objeto iterável é qualquer objeto com um método de iterador especial que
Retorna um objeto iterador.Um iterador é qualquer objeto com um próximo ()
Método que retorna um objeto de resultado da iteração.E um resultado de iteração
O objeto é um objeto com propriedades nomeadas e feitas.Para iterar
Um objeto iterável, você primeiro chama seu método de iterador para obter um iterador
objeto.Em seguida, você chama o método próximo () do objeto iterador
repetidamente até que o valor retornado tenha sua propriedade concluída definida como true.
O mais complicado disso é que o método do iterador de um iterável
O objeto não tem um nome convencional, mas usa o símbolo
Symbol.iterator como seu nome.Então, um simples para/de loop sobre um
objeto iterável iterável também pode ser escrito da maneira mais difícil, como
esse:

```
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
notação
tostring () {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 nosso
Localização no
// iteração.Começamos no primeiro número inteiro> = de.
deixe o próximo = math.ceil (this.from);// este é o próximo
valor que retornamos
deixe 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 iterador
iterável.
[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 útil
Defina funções que retornam valores iteráveis.Considere estes iteráveis-
alternativas baseadas nos métodos map () e filter () de
JavaScript Matrizes:
// retorna um objeto iterável que itera o resultado de
Aplicando f ()
// para cada valor da fonte iterável
mapa 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 um
variedade

```
[... mapa (novo intervalo (1,4), x => x*x)] // => [1, 4, 9, 16]
// retorna um objeto iterável que filtra o especificado
iterável,
// iterando apenas os elementos para os quais o predicado
retorna verdadeiro
filtro 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 inerentemente 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 poderia Basta 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 o Primeira palavra.E você acaba alocando muita memória para os devolvidos Array e todas as cordas dentro dela.Aqui está uma função que permite você para atingir preguiçosamente as palavras de uma corda sem mantê -las memória 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 fim
r.LastIndex = s.Match (/[^]/).// Comece a combinar
no primeiro não espaço
retornar {// retorna um
objeto iterador iterável
[Symbol.iterator] () {// Isso nos faz
iterável
devolver isso;
},
próximo () {// isso nos torna um
iterador
deixe iniciar = r.LastIndex;// retomar onde o
A última partida terminou
se (start <s.length) {// se não formos
feito
Seja Match = R.Exec (S);// corresponde ao próximo
limite da palavra
se (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 os
elementos de uma estrutura de dados, mas o resultado de uma computação.
Para criar um gerador, você deve primeiro definir uma função de gerador.UM
A função do gerador é sintaticamente como uma função JavaScript regular, mas
é definido 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
uma declaração de rendimento.O rendimento é novo no ES6 e é algo como um
Declaração 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 gerador
objeto.Chamando
rendimento 3;// O Método Next ()
Gerador é executado
rendimento 5;// o código até um rendimento
A declaração fornece
rendimento 7;// o valor de retorno para o
Método seguinte ().
}
// Quando invocamos a função do gerador, temos um gerador
deixe os primos = onedigitPries ();
// um gerador é um objeto de iterador que itera o
valores produzidos
Primes.Next (). Valor // => 2
Primes.Next (). Valor // => 3
Primes.Next (). Valor // => 5
Primes.Next (). Valor // => 7
Primes.next (). feito // => true
// Os geradores têm um método de símbolo.iterator para torná -los
iterável
primos [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 // => 17
```

Neste exemplo, usamos uma declaração de função* para definir um gerador.Como funções regulares, no entanto, também podemos definir geradores 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 o
Função de palavra -chave inteiramente quando definimos métodos.Para definir um
gerador neste contexto, simplesmente usamos um asterisco antes do método
Nome 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 deste
objeto
*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 seta
Função Sintaxe.

Os geradores geralmente facilitam a definição de classes iteráveis.
Podemos substituir o método [symbol.iterator] () mostrar em
Exemplo 12-1 com um muito mais curto *
[Symbol.iterator & rbrack; () Função do gerador que parece
assim:

```
*[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 gerador
função no contexto.

12.3.1 Exemplos de geradores

Os geradores são mais interessantes se realmente gerarem os valores
Eles produzem fazendo algum tipo de computação.Aqui, por exemplo, é um
Função 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 tem
Um loop infinito e produz valores para sempre sem retornar.Se isso
O gerador é usado com o ... Spread Operator, ele fará um loop até
A memória está esgotada e o programa trava.Com cuidado, é possível
Para 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) // => 10946
```

Esse tipo de gerador infinito se torna mais útil com uma tomada ()
gerador como este:

```
// produz os primeiros n elementos do iterável especificado
objeto
função* Take (n, iterable) {
deixe = iterable [symbol.iterator] ();// Obtenha iterador para
```

```
objeto iterável
while (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
o 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 recursivos

Além do gerador zip () definido no exemplo anterior,
Pode ser útil ter uma função de gerador semelhante que produz o
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", OneDigitPries ())] // =>
["A", "B", "C", 2,3,5,7]
```

Este processo de produzir os elementos de algum outro objeto iterável é
Comum o suficiente em funções geradoras que o ES6 possui sintaxe especial para
isto.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 dos
os 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 um
computação, produzir resultados intermediários e depois retomar o

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

12.4.1 O valor de retorno de uma função de gerador

As funções do gerador que vimos até agora não tiveram retorno
declarações, ou se tiverem, foram usadas para causar um início
retornar, 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
propriedade.Com iteradores e geradores típicos, se a propriedade do valor
é definido, 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
É verdade, indicando que não há mais valores para iterar.Esta final
O valor é ignorado pelo loop for/de
está disponível para codificar que itera manualmente com chamadas explícitas para
próximo():

```
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 novamente
generator.next () // => {value: indefinido, feito:
verdadeiro }
12.4.2 O valor de uma expressão de rendimento
Na discussão anterior, tratamos o rendimento como uma declaração que
leva 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 gerador
A 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 o
Valor de retorno do próximo () invocação.Neste ponto, o gerador
A função para de executar bem no meio da avaliação do rendimento
expressão.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
seu 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 seguir
O 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 == 1
console.log ("gerador rendido", n1.value);
Seja n2 = g.next ("b");// n2.value == 2
console.log ("gerador rendido", n2.value);
Seja n3 = g.next ("c");// n3.Value == 3
console.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 B
O gerador produziu 2
Em seguida () invocou uma terceira vez com o argumento C
O gerador produziu 3
Em seguida () invocou uma quarta vez com o argumento D
O 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 um Gerador

Vimos que você pode receber valores produzidos ou devolvidos por um Função do gerador.E você pode passar valores para um gerador em execução por passando 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 seu Métodos de return () e Throw ().Como os nomes sugerem, chamando Esses métodos em um gerador fazem com que retorne um valor ou jogue um exceção como se a próxima declaração no gerador fosse um retorno ou lançar.

Lembre -se de anterior ao capítulo de que, se um iterador define um Return () Método e iteração para mais cedo, depois o intérprete chama automaticamente o método Return () para dar uma chance ao iterador para fechar arquivos ou fazer outra limpeza.No caso de geradores, você não pode Defina um método de retorno () personalizado para lidar com a limpeza, mas você pode estruturar o código do gerador para usar uma declaração de tentativa/finalmente que garante que a limpeza necessária seja feita (no bloco finalmente) quando o gerador retorna.Forçando o gerador a retornar, o O método de retorno interno () do gerador garante que o código de limpeza é executado quando o gerador não será mais usado.

Assim como o método próximo () de um gerador nos permite passar arbitrário valores em um gerador em execução, o método throw () de um gerador nos 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 sempre
sequência crescente de números inteiros.Isso poderia ser escrito para que um
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 usa
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 nenhum
razão prática para fazê -lo.

12.4.4 Uma nota final sobre geradores

Os geradores são uma estrutura de controle generalizada muito poderosa.Eles dão
nós a capacidade de pausar um cálculo com rendimento e reiniciá -lo novamente em
Alguns arbitrários posteriormente com um valor de entrada arbitrária.É possível
Use geradores para criar um tipo de sistema de rosqueamento cooperativo dentro
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 função
são realmente assíncronos e dependem dos eventos da rede.
Tentar fazer essas coisas com geradores leva ao código que é a mente
Difícilmente 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 mais
Qualquer motivo para abusar dos geradores dessa maneira.

12.5 Resumo

Neste capítulo, você aprendeu:

O loop for/of e o ... o operador espalhado trabalham com
objetos 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 um
objeto de resultado da iteração.
Um objeto de resultado da iteração possui uma propriedade de valor que detém o
Próximo valor iterado, se houver um.Se a iteração tiver
Concluído, então o objeto de resultado deve ter uma propriedade feita
definido como true.
Você pode implementar seus próprios objetos iteráveis, definindo um
[Symbol.iterator] () Método que retorna um objeto
com um método próximo () que retorna objetos de resultado da iteração.
Você também pode implementar funções que aceitam o iterador
argumentos 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 do
A função não é executada imediatamente;Em vez disso, o valor de retorno é
um objeto iterador iterável.Cada vez que o próximo () método de
O iterador é chamado, outro pedaço da função do gerador
corre.
As funções do gerador podem usar o operador de rendimento para especificar o
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. Assíncrono JavaScript

Alguns programas de computador, como simulações científicas e máquina
Os modelos de aprendizado, são ligados a computação: eles correm continuamente, sem
Pause, até que tenham calculado seu resultado.O computador mais real do mundo real
Os programas, no entanto, são significativamente assíncronos.Isso significa isso
Eles geralmente precisam parar de calcular enquanto aguardam a chegada dos dados ou
Para que algum evento ocorra.Os programas JavaScript em um navegador da web são
normalmente orientado a eventos, o que significa que eles esperam o usuário clicar ou
Toque antes que eles realmente façam qualquer coisa.E servidores baseados em JavaScript
normalmente aguarda os pedidos do cliente chegarem pela rede antes de eles
faça qualquer coisa.

Esse tipo de programação assíncrona é comum em
JavaScript, e este capítulo documenta três idiomas importantes
Recursos que ajudam a facilitar o trabalho com código assíncrono.
Promessas, novas no ES6, são objetos que representam o que ainda não está disponível
resultado de uma operação assíncrona.As palavras -chave async e aguardam
foram introduzidos no ES2017 e fornecem uma nova sintaxe que simplifica
Programação assíncrona, permitindo que você estruture sua promessa
código baseado como se fosse síncrono.Finalmente, iteradores assíncronos
e o loop for/wait foi introduzido no ES2018 e permitir que você
Trabalhe com fluxos de eventos assíncronos usando loops simples que
Parece síncrono.

Ironicamente, embora o JavaScript forneça esses recursos poderosos para
Trabalhando 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
Invokes ("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 concreto
exemplos 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 é
Um intervalo de tempo medido em milissegundos.No código anterior, um
A função hipotética checkForupDates () será chamada de 60.000
milissegundos (1 minuto) após a chamada setTimeout ().
checkForupDates () é uma função de retorno de chamada que seu programa
pode definir, e setTimeout () é a função para a qual você invoca
Registre sua função de retorno de chamada e especifique sob o que assíncrono
Condições deve ser invocada.
setTimeout () chama a função de retorno de chamada especificada uma vez,
não transmitindo argumentos e depois esquece.Se você está escrevendo um
função que realmente verifica se há atualizações, você provavelmente deseja que ela seja executada
repetidamente.Você pode fazer isso usando setInterval () em vez de
setTimeout ():
// Ligue para o checkForupDates em um minuto e depois novamente a cada
minuto depois disso
Deixe 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 Eventos
Os programas JavaScript do lado do cliente são quase universalmente orientados a eventos:

Em vez de executar algum tipo de computação predeterminada, eles
normalmente espera o usuário fazer algo e depois responder ao
Ações do usuário.O navegador da web gera um evento quando o usuário
pressiona uma tecla no teclado, move o mouse, clica em um mouse
botão ou toca em um dispositivo de tela sensível ao toque.JavaScript orientado a eventos
Programas Registrar funções de retorno de chamada para tipos especificados de eventos em
Contextos 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 com
addEventListener ():

```
// Peça ao navegador da web para devolver um objeto que representa o
Html
// elemento <butter> que corresponde a este seletor CSS
Let Okk = Document.QuerySelector ('#confirmUpDatedialog
Button.OKay ');
// Agora registre uma função de retorno de chamada a ser invocada quando o
usuário
// clica nesse botão.
```

ok.
Neste exemplo, ApplUpDate () é uma função hipotética de retorno de chamada
que assumimos ser implementado em outro lugar.A chamada para
document.QuerySelector () retorna um objeto que representa um
Elemento especificado único na página da web.Nós chamamos
addEventListener () nesse elemento para registrar nosso retorno de chamada.
Então o primeiro argumento para addEventListener () é uma string que
Especifica o tipo de evento em que estamos interessados ??- um clique do mouse ou
Toque em tela sensível ao toque, neste caso.Se o usuário clicar ou torneiras nesse específico
Elemento da página da web, então o navegador invocará nosso
Função 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:

```
função 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 o
A resposta chega
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);
VersionCallback (NULL, CurrentVersion);
} outro {
// Caso contrário, relate um erro ao retorno de chamada
VersionCallback (Response.Statustext, NULL);
}
};
// 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 PLUS
Funções de retorno de chamada para fazer solicitações HTTP e manipular assíncrono
a resposta do servidor quando chega.O
GetCurrentVersionNumber () Função definida aqui (podemos
Imagine que é usado pelos hipotéticos checkForupDates ()
Função discutimos no §13.1.1) faz uma solicitação HTTP e define
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 web
APIs (incluindo este), os manipuladores de eventos podem ser definidos invocando
addEventListener () no objeto que gera o evento e
Passando 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 fazemos
Neste código de exemplo, atribuindo funções ao Onload, OnError,
e propriedades on -timeout.Por convenção, propriedades do ouvinte de eventos
Como 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 que
Nenhum outro código precisará registrar um ouvinte para o mesmo objeto e
Tipo de evento, pode ser mais simples simplesmente definir a propriedade apropriada como
Seu retorno de chamada.
Outra coisa a observar sobre o getCurrentVersionNumber ()
função neste exemplo, o código é que, porque faz um
solicitação assíncrona, ele não pode retornar síncrono o valor (o

número de versão atual) em que o chamador está interessado. Em vez disso, o
O 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 um
Função de retorno de chamada que espera dois argumentos.Se o xmlHttPrequest
Funciona corretamente, então getCurrentVersionNumber () invoca o
retorno de chamada com um primeiro argumento nulo e o número da versão como o
segundo argumento.Ou, se ocorrer um erro, então
getCurrentVersionNumber () chama o retorno de chamada com erro
detalhes no primeiro argumento e nulo como o segundo argumento.

13.1.4 retornos de chamada e eventos no nó

O ambiente JavaScript do lado do Node.js é profundamente
assíncrono e define muitas APIs que usam retornos de chamada e eventos.
A API padrão para ler o conteúdo de um arquivo, por exemplo, é
assíncrono e chama uma função de retorno de chamada quando o conteúdo do
O arquivo foi lido:

```
const fs = requer ("fs");// O módulo "FS" tem sistema de arquivos-
APIs relacionadas
deixe opções = {// um objeto para manter opções para
nosso programa
// Opções padrão iriam aqui
};
// Leia um arquivo de configuração e ligue para a função de retorno de chamada
fs.readfile ("config.json", "utf-8", (err, texto) => {
if (err) {
// Se houve um erro, exiba um aviso, mas
continuar
console.warn ("Não foi possível ler o arquivo de configuração:", err);
} outro {
// caso contrário, analise o conteúdo do arquivo e atribua a
o objeto de opções
Object.assign (options, json.parse (text));
```

```
}
// Em ambos os casos, agora podemos começar a executar o programa
startProgram (opções);
});
```

A função fs.readfile () do Node recebe um retorno de chamada de dois parâmetros como seu último argumento.Ele lê o arquivo especificado de forma assíncrona e depois Invoca o retorno de chamada.Se o arquivo foi lido com sucesso, ele passa o arquivo conteúdo como o segundo argumento de retorno de chamada.Se houve um erro, é passa o erro como o primeiro argumento de retorno de chamada.Neste exemplo, nós expressar o retorno de chamada como uma função de seta, que é um sucinto e Sintaxe natural para esse tipo de operação simples.

O Node também define uma série de APIs baseadas em eventos.A seguir A função mostra como fazer uma solicitação http para o conteúdo de um URL no nó.Tem duas camadas de código assíncrono tratado com ouvintes de eventos.Observe que o nó usa um método on () para registrar ouvintes de eventos em vez de addEventListener ():

```
const https = requer ("https");
// Leia o conteúdo de texto do URL e passa de forma assíncrona
para o retorno de chamada.
função getText (url, retorno de chamada) {
// Inicie um http get solicitação para o URL
solicitação = 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 resposta
foram recebidos
Seja httpstatus = resposta.statuscode;
// O corpo da resposta HTTP não foi
recebido ainda.
```

```
// Então, registramos mais manipuladores de eventos para serem chamados
Quando chegar.
Response.setEncoding ("UTF-8");// estamos esperando
Texto unicode
Deixe Body = "";// que vamos
acumule aqui.
// Este manipulador de eventos é chamado quando um pedaço do
O corpo está pronto
resposta.on ("dados", chunk => {body += chunk;});
// Este manipulador de eventos é chamado quando a resposta é
completo
Response.on ("end", () => {
if (httpstatus === 200) {// se o http
A 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 rede
request.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 assíncrono
computação.Esse resultado pode ou não estar pronto ainda, e a promessa
API é intencionalmente vaga sobre isso: não há como
obter o valor de uma promessa;você só pode pedir a promessa de ligar para um
Função de retorno de chamada quando o valor estiver pronto.Se você está definindo um
API assíncrona como a função getText ()
seção, mas quero torná-la baseada em promessa, omitir o retorno de chamada
argumento 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
Invocado 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
retornos de chamada, com linhas de código tão altamente recuadas que é difícil
ler.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 manuseio
difícil.Se uma função assíncrona (ou um invocado assíncrono
retorno de chamada) lança uma exceção, não há como essa exceção
Propagam de volta ao iniciador da operação assíncrona.Este é um
Fato fundamental sobre programação assíncrona: quebra
manuseio de exceção.A alternativa é para a pista meticulosamente e
propagar erros com argumentos de retorno de chamada e valores de retorno, mas isso é
tedioso e difícil de acertar.Promessas ajuda aqui padronizando um
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 assíncrono
cálculos.Eles não podem ser usados ??para representar repetidos assíncronos
cálculos, 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 um
Promessa em vez do manipulador de eventos de "carga" de um xmlhttprequest
Objeto, já que esse retorno de chamada é chamado apenas uma vez.Mas nós normalmente
não usaria uma promessa no lugar de um manipulador de eventos de "clique" de um
Objeto de botão HTML, já que normalmente queremos permitir que o usuário clique
um botão várias vezes.
As subseções a seguir:
Explique a terminologia da promessa e mostre o uso básico de promessa
Mostre como as promessas podem ser acorrentadas
Demonstrar 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 para
programação assíncrona, mas você precisa entendê -los profundamente para usá -los
corretamente e com confiança.Vale a pena dedicar um tempo para desenvolver tão profundo
Entendendo, 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
computação e cada função registrada com então () será
invocado apenas uma vez.Vale a pena notar que a função que você passa
então () é invocado 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 distinto
característica das promessas, e é idiomático anexar. então () diretamente
a invocação de funções que retorna a promessa, sem o
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
perfil
Função DisplayUserProfile (perfil) { /* Implementação
omitido */}
// 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 promessas
Operações 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
Isso ocorre depois que o objeto Promise for criado.Porque o
A computação é realizada após a devolução do objeto de promessa,
Não há como o cálculo tradicionalmente retornar um valor ou
Jogue uma exceção que podemos pegar.As funções para as quais passamos
então () fornecer alternativas.Quando um cálculo síncrono
Conclui normalmente, ele simplesmente retorna seu resultado ao seu chamador.Quando a
A computação assíncrona baseada em promessa é concluída normalmente, ele
passa seu resultado para a função que é o primeiro argumento para então ().
Quando algo dá errado em um cálculo síncrono, ele 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 com
Promessas.Para entender, primeiro considere o que acontece se Getjson ()
completa normalmente, mas ocorre um erro em
DisplayUserProfile ().Essa função de retorno de chamada é invocada
assíncrono 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 (h
andleprofileError);
Com este código, um resultado normal de getjson () ainda é passado para
displayUserProfile (), mas qualquer erro em getjson () ou em
DisplayUserProfile () (incluindo todas as exceções jogadas por
displayUserProfile) seja passado para
handleprofileError ().O método Catch () é apenas um
abreviação de ligar então () com um primeiro argumento nulo e o
O manipulador de erros especificado funciona como o segundo argumento.
Teremos mais a dizer sobre Catch () e este idioma de manipulação de erros
Quando discutirmos as cadeias prometidas na próxima seção.
Terminologia da promessa
Antes 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 rejeitado se e quando o segundo retorno de chamada for chamado.Se uma promessa não for cumprida nem Rejeitado, então está pendente.E uma vez que uma promessa é cumprida ou rejeitada, dizemos que está resolvida.Observação
que 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
Resultados desse código assíncrono.Se o código assíncrono executar normalmente (e a promessa é cumprida), então isso
O resultado é 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
Rejeitado, 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 resolvido.É fácil confundir este estado resolvido com o estado cumprido ou com o estado liquidado, mas é não é precisamente o mesmo que.Compreender o estado resolvido é uma das chaves para um profundo Compreensão 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 um
maneira natural de expressar uma sequência de operações assíncronas como um
Cadeia linear de invocações de método então (), sem ter que ninho
cada operação dentro do retorno de chamada do anterior.Aqui, para
Exemplo, é uma cadeia de promessas hipotéticas:

```
busca (documenturl) // faça um http
solicitar
.Then (Response => Response.json ()) // Peça o JSON
corpo da resposta
.then (document => {// quando obtemos o
Parsed JSON
render renderização (documento);// exibe o
documento para o usuário
```

})
.THEN (renderizado => {// quando obtemos o
documento renderizado
CacheIndatabase (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
Discuta 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
Faça uma solicitação HTTP no JavaScript.Aquele objeto estranhamente nomeado tem
uma API antiga e desajeitada, e foi amplamente substituída pelo
API de busca mais recente baseada em promessa (§15.11.1).Em sua forma mais simples, esta
A nova API HTTP é apenas a função busca ().Você passa por um URL, e
ele 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
if (Response.ok &&
Response.Headers.get ("Content-Type") ===
"Application/json") {
// O que podemos fazer aqui?Na verdade, não temos o
corpo de 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 de
A resposta ainda pode não ter chegado.Então esses texto () 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, como
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.Inicia
Um HTTP recebe solicitação para esse URL e retorna uma promessa.
Vamos chamar isso de solicitação http de "Tarefa 1" e chamaremos o
Promise "Promise 1".
2. Na segunda linha, invocamos o método então () de
Promise 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 é
invocado.
3. Na terceira linha, invocamos o método de promessa então ()
2, passando a função de retorno2 que queremos invocar quando
A promessa 2 é cumprida.Este método então () se lembra do nosso
retorno de chamada e retorna mais uma promessa.Vamos dizer essa "tarefa
3 ?começa quando o retorno de chamada2 é invocado.Nós podemos chamar isso
Última promessa "Promise 3", mas realmente não precisamos de um nome
Por isso, porque não o usaremos.
4. As três etapas anteriores acontecem de forma síncrona quando o
A expressão é executada pela primeira vez.Agora temos um assíncrono
pausa enquanto a solicitação HTTP iniciada na etapa 1 é enviada
na Internet.
5. Eventualmente, a resposta HTTP começa a chegar.O
parte assíncrona da chamada fetch () envolve o http
Status e cabeçalhos em um objeto de resposta e cumpre a promessa 1
com esse objeto de resposta como o valor.
6. Quando a promessa 1 é cumprida, seu valor (o objeto de resposta) é
Passado para a nossa função de retorno de chamada1 () e a tarefa 2 começa.O
O trabalho desta tarefa, dado um objeto de resposta como entrada, é obter
O 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 JSON
objeto.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 isso
Promessa 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 registramos.
Este objeto de resposta possui .text (), .json () e outros métodos para
Solicite o corpo da resposta HTTP de várias formas.Mas desde o
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 2
displayUserProfile (perfil);
}
Seja p1 = buscar ("/api/usuário/perfil");// Promise 1, Tarefa 1
Seja P2 = P1.THEN (C1);// Promise 2, Tarefa 2
Seja p3 = p2.then (c2);// Promise 3, Tarefa 3
Para que as cadeias de promessas funcionem de maneira útil, a saída da Tarefa 2 deve
Torne -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 um
Objeto json.Mas, como acabamos de discutir, o valor de retorno do retorno de chamada
C1 não é um objeto JSON, mas prometa P4 para esse objeto JSON.Esse
Parece uma contradição, mas não é: quando P1 é cumprido, C1 é
Invocado, 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 são
sobre o gerenciamento de tarefas assíncronas, afinal, e se a Tarefa 2 for
assíncrono (o que é, neste caso), então essa tarefa não será
Completo no momento em que o retorno de chamada retorna.
Agora estamos prontos para discutir os detalhes finais que você precisa para
Entenda para realmente dominar promessas.Quando você passa um retorno de chamada C para
O método então (), então () retorna uma promessa P e organiza
Invocar de maneira assíncrona C em algum momento posterior.O retorno de chamada é executado
Alguns computação e retorna um valor v. Quando o retorno de chamada retorna, P
é 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 .hen () 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 discuta isso com mais detalhes.Para prefaciar 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.O
A boa notícia é que o método .catch () facilita o manuseio
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 e
Porque o nome corresponde à cláusula de captura em uma tentativa/captura
declaração de manipulação de exceção.Como discutimos, exceções normais
Não trabalhe com código assíncrono.O método .catch () de
As promessas são uma alternativa que funciona para o código assíncrono.Quando
algo dá errado em código síncrono, podemos falar de um
Exceção ?borbulhando a pilha de chamadas? até encontrar um bloco de captura.
Com uma cadeia de promessas assíncronas, a metáfora comparável
pode 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 finalmente
Experimente/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 chamou
se estabelece.Seu retorno de chamada será chamado se a promessa cumprir ou rejeitar,
e não será aprovado nenhum argumento, então você não pode descobrir se
cumpriu ou rejeitou.Mas se você precisar executar algum tipo de limpeza
código (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.O
o valor de retorno de um retorno de chamada .Finalmente () é geralmente ignorado, e o
Promessa devolvida por .Finalmente () normalmente resolverá ou rejeitará com
o 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 prontos
if (! 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 nulo
if (perfil) {
displayUserProfile (perfil);
}
```

```
else {// se obtivemos um erro 404 acima e devolvemos
nulo nós acabamos aqui
displayLoggedoutProfilePage ();
}
})
.catch (e => {
if (e instanceof NetworkError) {
// fetch () pode falhar dessa maneira se a internet
A conexão está baixa
DisplayErRormessage ("Verifique sua internet
conexão.");
}
else if (e instanceof typeError) {
// isso acontece se jogarmos TypeError acima
DisplayErRormessage ("Algo está errado com nosso
servidor! ");
}
outro {
// Isso deve ser algum tipo de erro imprevisto
console.error (e);
}
});
```

Vamos analisar esse código olhando o que acontece quando as coisas vão errado.Usaremos o esquema de nomenclatura que usamos antes: p1 é o Promessa devolvida pela chamada fetch ().P2 é a promessa devolvida por A 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 o Retorno que passamos para a chamada .catch ().(Essa chamada retorna um Prometa, mas não precisamos nos referir a ele pelo nome.)

A primeira coisa que poderia falhar é a solicitação de busca ().Se o A conexão de rede está baixa (ou por algum outro motivo um http 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 como
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 .CACH () no
cadeia, resultando em código que pode ser assim:
startAsyncoperation ()
. então (Dostagetwo)
.catch (recuperação de estagiário)

. então (Dostagethree)
. então (DostageFour)
.catch (LogStagethReeAndFouRerRors);
Lembre -se de que o retorno de chamada que você passa para .catch () será apenas
Invocado se o retorno de chamada em um estágio anterior lançar um erro.Se o
Retorno de chamada retorna normalmente, então o retorno de chamada .catch () será
pulados, e o valor de retorno do retorno de chamada anterior se tornará o
Entrada 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 isso
O valor de retorno é usado para resolver e/ou cumprir a promessa associada, e
O erro para de se propagar.
Vamos ser concretos sobre isso: no exemplo do código anterior, se qualquer um
startasyncoperation () ou DostageTwo () lança um erro,
então a função recuperada de fromstagetwoerror () será
invocado.Se recuperar de questagetwoerror () retornar normalmente,
então seu valor de retorno será passado para Dostagethree () e o
A operação assíncrona continua normalmente.Por outro lado, se
recuperado de estagiário () não conseguiu se recuperar.
Por si só, eleva um erro (ou ele repete o erro que ele foi aprovado).Em
Este caso, nem Dostagethree () nem DostageFour () serão
invocado, e o erro lançado por
recuperado de estagiário () seria passado para
LOGSTAGETHREEANDFOURERRORS ().

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 falhar
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
do 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
Retorne 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 um
A 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 inócua:
.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 paralelo
Passamos 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 assíncronos 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 código como este:

```
// começamos com uma variedade de URLs
const urls = [ / * zero ou mais URLs aqui * /];
// e converta -o em uma variedade de objetos de promessa
promessas = urls.map (url => busca (url) .then (r => r.text ()));
// agora tem uma promessa de executar todas essas promessas em paralelo
Promise.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.O
A 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 promessa.all () rejeita quando qualquer um dos
As 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 entrada
Promete e retorna uma promessa, assim como a promessa.All () faz.Mas
Promete.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.E
Se o status for "rejeitado", o objeto também terá um motivo
propriedade que fornece o erro ou o valor de rejeição do correspondente
Promessa:
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 promete.race () em vez de promete.all ().Isto retorna uma promessa que é cumprida ou rejeitada quando o primeiro dos As promessas na matriz de entrada são cumpridas ou rejeitadas.(Ou, se houver algum valores 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 tem Também se baseou em funções hipotéticas de remuneração de promessas getjson () e espera ().Funções escritas para devolver promessas realmente são bastante Útil, 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 alguns 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 () assim:

```
função getjson (url) {
return fetch (url) .hen (resposta => resposta.json ());
}
```

O código é trivial porque o objeto de resposta da API busca ()
possui um método JSON () predefinido.O método json () retorna um
Promessa, 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 o
Promessa devolvida por getjson () resolve a promessa devolvida por
Response.json ().Quando essa promessa cumpre, a promessa
retornado por getjson () atende ao mesmo valor.Observe que há
Não há tratamento de erro nesta implementação getjson ().Em vez de
verificando a resposta.OK e o cabeçalho do tipo de conteúdo, em vez disso
Basta permitir que o método json () rejeite a promessa que retornou com um
SyntaxError se o corpo de resposta não puder ser analisado como JSON.
Vamos escrever outra função de retorno de promessa, desta vez usando
getjson () como fonte da promessa inicial:
função gethsscore () {
retornar getjson ("/api/user/perfil"). Então (perfil =>
perfil.highscore);
}
Estamos assumindo que essa função faz parte de algum tipo de Web baseada na Web
jogo e que o URL ?/API/User/Perfil? retorna um JSON formatado
Estrutura de dados que inclui uma propriedade Highscore.
Promessas baseadas em valores síncronos
Às vezes, pode ser necessário implementar um existente baseado em promessa
API e devolver uma promessa de uma função, mesmo que o
A computação a ser realizada não exige nenhum
operações 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: o
As promessas devolvidas por esses métodos estáticas ainda não são cumpridas ou
rejeitados quando forem devolvidos, mas eles cumprirão ou rejeitarão
Imediatamente após a atual parte síncrona de código terminou
correndo.Normalmente, isso acontece em alguns milissegundos, a menos que haja
são muitas tarefas assíncronas pendentes esperando para correr.)
Lembre -se do §13.2.3 de que uma promessa resolvida não é a mesma coisa que um
promessa cumprida.Quando chamamos de promete.resolve (), normalmente
Passe o valor do cumprimento para criar um objeto de promessa que em breve
cumprir com esse valor.O método não é nomeado
Prometo.fulfill (), no entanto.Se você passar uma promessa P1 para
Promise.Resolve (), ele retornará uma nova promessa P2, que é
imediatamente resolvido, 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
o valor é calculado de forma síncrona e devolvida de forma assíncrona com
Promise.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 com
Promete.reject ().(Você também pode simplesmente jogar um erro
síncrono nesse caso, mas isso é considerado uma forma ruim porque
Então o chamador da sua função precisa escrever um síncrono
Cláusula de captura e use um método .Catch () assíncrono para lidar
erros.) Finalmente, Promise.Resolve () às vezes é útil para criar
A promessa inicial em uma cadeia de promessas.Vamos ver alguns
Exemplos que o usam dessa maneira.

Promessas do zero

Para getjson () e gethighscore (), começamos
chamando uma função existente para obter uma promessa inicial e criada e
devolveu uma nova promessa chamando o método .hen () dessa inicial
Promessa.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 novo
Promise 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 esperar
Dois parâmetros, que, por convenção, devem ser nomeados resolvendo e
rejeitar.O construtor chama de síncrona sua função com
Argumentos de função para os parâmetros de resolução e rejeição.Depois
Chamando sua função, o construtor Promise () retorna o recém
promessa 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) => {// estes
Controle 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 um
Promessa 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 em
Nó, onde a API Fetch () não é incorporada. Lembre -se de que nós
Iniciou este capítulo com uma discussão sobre retornos de chamada assíncronos e
eventos.Este exemplo usa retornos de chamada e manipuladores de eventos e é um
boa demonstração, portanto, de como podemos implementar a promessa
APIs 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 promessa
retornar nova promessa ((resolver, rejeitar) => {
// Inicie um http get solicitação para o URL especificado
solicitação = http.get (url, resposta => {// chamado quando
A resposta começa
// rejeitar a promessa se o status HTTP estiver errado
if (Response.statuscode! == 200) {
rejeite (novo erro (`status http
$ {Response.statuscode} `));
Response.Resume ();// então não vazamos memória
}
// e rejeite se os cabeçalhos de resposta estiverem errados
caso 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 corpo
da resposta
Deixe Body = "";
Response.setEncoding ("UTF-8");
```

```
resposta.on ("dados", chunk => {body += chunk;
});
Response.on ("end", () => {
// Quando o corpo de resposta estiver completo, tente
para analisá -lo
tentar {
Seja analisado = json.parse (corpo);
// se for divulgado com sucesso, cumpra
a promessa
resolver (analisado);
} catch (e) {
// Se a análise falhou, rejeite o
Promessa
rejeitar (e);
}
});
}
});
// também rejeitamos a promessa se a solicitação falhar
antes de nós
// até obtenha uma resposta (como quando a rede é
abaixo)
request.on ("erro", erro => {
rejeitar (erro);
});
});
}
```

13.2.7 Promessas em sequência

Promey.all () facilita a execução de um número arbitrário de Promessas em paralelo.E 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 de comprimento arbitrário e conteúdo desconhecido, você não pode escrever uma promessa

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
corpos const = [];
// Aqui está uma função de denúncia de promessa que busca uma
corpo
função fetchOne (url) {
Retornar Fetch (URL)
.Then (Response => Response.Text ())
.Then (Body => {
// salvamos o corpo na matriz, e somos
propositalmente
// 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
a corrente
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 buscar
os URLs um de cada vez com código como o código de busca em paralelo
Usamos anteriormente para demonstrar Promise.all ():

```
FetchSequencialmente (URLs)
.Then (corpos => { /* Faça algo com a matriz de
Strings */})
.catch (e => console.error (e));
```

A função FetchSequencialmente () começa criando uma promessa
Isso cumprirá imediatamente após o retorno.Em seguida, constrói um longo e linear
Promessa se encaixa nessa promessa inicial e retorna a última promessa em
a corrente.É como montar uma fileira de dominó e depois bater o
primeiro um.
Há outra abordagem (possivelmente mais elegante) que podemos adotar.
Em vez de criar as promessas com antecedência, podemos ter o retorno de chamada
Para cada promessa, crie e retorne a próxima promessa.Isto é, em vez de
Criando e encadeando várias promessas, criamos promessas
Isso resolve outras promessas.Em vez de criar um dominó
Cadeia de promessas, estamos criando uma sequência de promessas
aninhado um dentro do outro como um conjunto de bonecas Matryoshka.Com isso
abordagem, nosso código pode retornar a primeira promessa (mais externa), sabendo
que acabará cumprindo (ou rejeitará!) Para o mesmo valor que o último
(Interior) promessa na sequência.O
Promisesequence () A função a seguir é escrita para ser genérica
e não é específico para buscar o URL.Está aqui no final do nosso
Discussão das promessas porque é complicada.Se você leu isso
Capítulo com cuidado, no entanto, espero que você consiga entender como é
funciona.Em particular, observe que a função aninhada dentro

Promisesequence () parece se chamar recursivamente, mas porque
A chamada "recursiva" é através de um método então (), na verdade não há
Qualquer recursão tradicional acontecendo:
// Esta função leva uma matriz de valores de entrada e um
Função "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 deixar
eles correm
// Paralelo, no entanto, o promission () executa apenas uma promessa
de cada vez
// e não chama o promissor () para obter um valor até o
promessa anterior
// cumpriu.
Função Promisesequence (Entradas, Promisemaker) {
// Faça uma cópia privada da matriz que podemos modificar
entrada = [... entradas];
// Aqui está a função que usaremos como uma promessa
ligar de volta
// Esta é a magia pseudorrecursiva que faz com que tudo
trabalhar.
função 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 corrente
Promessa
// com o valor futuro de uma nova promessa.
deixe nextInput = inputs.shift ();// Obtenha o próximo
valor de entrada,

```
Return Promesemaker (NextInput) // Calcule o
Próximo valor de saída,
// então crie uma nova matriz de saídas com o
novo 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
pode 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 URL
Promisesequence (URLs, Fetchbody)
.Then (corpos => { /* Faça algo com a matriz de
Strings */})
.catch (console.error);
```

13.3 assíncrono e aguardar

O ES2017 apresenta duas novas palavras -chave - Async and Aguard
Representar 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 jogue uma exceção da maneira como o código síncrono regular
pode.E 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 um
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
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 aguait 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 não
fazer com que seu programa bloqueie e literalmente não faça nada até que o especificado
Promessa se acalma.O código permanece assíncrono, e o aguardar
Simplesmente disfarça esse fato.Isso significa que qualquer código que use aguarda é
ele próprio assíncrono.

13.3.2 Funções assíncronas

Porque qualquer código que usa aguarda é assíncrono, há um
Regra crítica: você só pode usar a palavra -chave aguardar em funções que
foram declarados com a palavra -chave assíncrona.Aqui está uma versão do
GethighScore () função do início do capítulo, reescrito para
Use 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 função
será uma promessa, mesmo que nenhum código relacionado à promessa apareça no corpo
da função.Se uma função assíncrona parece retornar normalmente, então
O objeto de promessa que é o valor de retorno real da função será
Resolva para esse aparente valor de retorno.E se uma função assíncrona
parece fazer uma exceção, então o objeto de promessa que ele retorna
será rejeitado com essa exceção.
A função GethighScore () é declarada assíncrona, por isso retorna um
Promessa.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: o
A busca do segundo URL não começará até que a primeira busca seja concluída.
Se o segundo URL não depender do valor obtido do primeiro
URL, provavelmente devemos tentar buscar os dois valores no mesmo
tempo.Este é um caso em que a natureza baseada em promessa de Async
Funções mostra.Para aguardar um conjunto de execução simultaneamente
funções assíncronas, usamos o prometido.all () exatamente como faríamos se
Trabalhando com promessas diretamente:
Seja [Value1, Value2] = Aguarda Promise.all ([Getjson (URL1),
getjson (url2)]);
13.3.4 Detalhes da implementação
Finalmente, para entender como as funções assínconais funcionam, pode ajudar
Pensar 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 embrulhada
Ao 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 uma
marcador que quebra um corpo de função em separado, síncrono
pedaços.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 assíncrona

Começamos este capítulo com uma discussão sobre retorno de chamada e eventos baseados em eventos
assincronia, e quando introduzimos promessas, observamos que eles
foram úteis para cálculos assíncronos de tiro único
Adequado 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
os 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/wait

O nó 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 = requer ("fs");
Função assíncreada parsefile (nome do arquivo) {
Deixe Stream = fs.cretereadstream (nome do arquivo, {coding:
"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 um
Promessa e o loop for/aguardar aguardam essa promessa de cumprir,
atribui o valor de atendimento à variável de loop e executa o corpo de
o 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 usar
Promessa.all () para esperar que todas as promessas da matriz sejam
cumprido.Mas suponha que queremos os resultados da primeira busca assim que
Eles ficam disponíveis e não querem 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
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 lá
são 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
O 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.O
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 é
frequentemente para usar um gerador.O mesmo vale para iteradores assíncronos,
que podemos implementar com as funções do gerador que declaramos
assíncrono.Um gerador assíncrono tem as características das funções assíncronas e o
Recursos de geradores: você pode usar aguardar como faria em um regular
função assíncrona, e você pode usar o rendimento como faria em um regular
gerador.Mas os valores que você produz são automaticamente envolvidos
Promessas.Até a sintaxe para geradores assíncronos é uma combinação:
Função e função assíncronas * combinam -se em assíncronas
função *.Aqui está um exemplo que mostra como você pode usar um
gerador assíncrono e um loop para/aguardar para executar o código repetidamente
intervalos 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 podemos
use aguarda com.
// retorna uma promessa que cumpre o número especificado de
milissegundos
Função DastaDtime (ms) {
return nova promessa (resolve => setTimeout (resolve, ms));
}
// uma função do gerador assíncrono que incrementa um contador e
produz isso
// um número especificado (ou infinito)
intervalo.
função assíncrona* relógio (intervalo, max = infinito) {
para (let count = 1; count <= max; count ++) {// regular
para loop
aguardar tempo de pasteld (intervalo);// Espere
hora de passar
contagem de rendimentos;// produz o
contador
}
}
```

// uma função de teste que usa o gerador assíncrono com
para/aguardar
teste de função assíncrona () {// assíncro
pode usar para/aguardar
para aguardar (deixe o tick de relógio (300, 100)) {// loop 100
vezes a cada 300ms
console.log (tick);
}
}
13.4.4 Implementando iteradores assíncronos
Em vez de usar geradores assíncronos para implementar iteradores assíncronos,
Também é possível implementá -los diretamente, definindo um objeto
com um método symbol.asynciterator () que retorna um objeto
com um método próximo () que retorna uma promessa que resolve para um
objeto de resultado do iterador.No código a seguir, reimplementamos o
clock () função do exemplo anterior para que não seja um
gerador 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 usar
aguarde com.
// Observe que isso leva um tempo absoluto em vez de um
intervalo.
função até (horário) {
Retorne nova promessa (resolve => setTimeout (resolve,
tempo - data.now ()));
}
// devolver um objeto de maneira assíncrona
retornar {
StartTime: date.now (), // lembre -se de quando começamos
CONTA: 1, // Lembre -se de qual iteração
Estamos ligados

```
assíncrono next () {// o método next () faz
Este é um iterador
if (this.count> max) {// terminamos?
return {done: true};// resultado de iteração
indicando feito
}
// descobrir quando a próxima iteração deve
começar,
Deixe TargetTime = this.startTime + this.count *
intervalo;
// Espere até aquele momento,
aguarde até (TargetTime);
// e retorne o valor da contagem em uma iteração
objeto de resultado.
return {value: this.count ++};
},
// Este método significa que este objeto de iterador é
também um iterável.
[Symbol.asynciterator] () {return this;}
};
}
```

Esta versão baseada em iterador da função clock () corrige uma falha em a versão baseada em gerador.Observe que, neste código mais recente, temos como alvo o tempo absoluto em que cada iteração deve começar e subtrair o tempo atual disso para calcular o intervalo para o qual passamos setTimeout ().Se usarmos o relógio () com um loop for/aguardar, este A versão será executada de iterações de loop mais precisamente no intervalo especificado Porque é responsável pelo tempo necessário para realmente executar o corpo do laço.Mas essa correção não é apenas uma precisão de tempo.O para/aguardar Loop sempre espera pela promessa devolvida por uma iteração para ser cumprido antes de começar a próxima iteração.Mas se você usar um iterador assíncrono sem um loop para/aguardar, não há nada para impedir 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 promessas
tudo isso vai cumprir quase exatamente ao mesmo tempo, o que provavelmente é
não o que você quer.A versão baseada em iterador que implementamos aqui
não tem esse problema.
O benefício dos iteradores assíncronos é que eles nos permitem representar
fluxos de eventos ou dados assíncronos.A função do relógio ()
discutido anteriormente era bastante simples de escrever porque a fonte de
A assincronia foi as chamadas setTimeout () que estávamos fazendo
nós mesmos.Mas quando estamos tentando trabalhar com outros assíncronos
fontes, como o desencadeamento dos manipuladores de eventos, torna -se
substancialmente mais difícil de implementar iteradores assíncronos - normalmente
ter uma única função de manipulador de eventos que responde aos eventos, mas cada
Chamada para o método Next () do iterador deve retornar uma promessa distinta
objeto, e várias chamadas para a próxima () podem ocorrer antes do primeiro
Promessa resolve.Isso significa que qualquer método de iterador assíncrono
deve ser capaz de manter uma fila interna de promessas que ela resolve
em ordem como eventos assíncronos estão ocorrendo.Se encapsularmos isso
Comportamento de prometo em uma aula de assíncrona, então se torna
Muito mais fácil escrever iteradores assíncronos com base no assíncrono.
A aula de assíncrona a seguir
Métodos dequeue () como você esperava para uma aula de filas.O
o método dequeue () retorna uma promessa em vez de um valor real,
No entanto, o que significa que não há problema em chamar Dequeue () antes
ENQUEUE () já foi chamado.A classe Asyncqueue também é um
iterador assíncrono e deve ser usado com um para/aguardar
Loop cujo corpo corre uma vez cada vez que um novo valor é assíncrono

preso.(Asyncqueue tem um método Close (). Uma vez chamado, não
Mais valores podem ser envolvidos.Quando uma fila fechada está vazia, o
para o loop/aguardar vai parar de fazer o loop.)
Observe que a implementação do AsyncQueue não usa assíncrono ou
Aguarde e, em vez disso, trabalhe diretamente com promessas.O código é
um pouco complicado, e você pode usá -lo para testar sua compreensão de
O material que abordamos neste longo capítulo.Mesmo que você não
entender a implementação do assíncico, dê uma olhada no
Exemplo mais curto que o segue: implementa um simples, mas muito
Iterador assíncrono interessante no topo do Asyncqueue.

```
/**
* Uma classe de fila de maneira assíncrona.Adicione valores com
enquistar ()
* e remova -os com dequeue ().Dequeue () retorna a
Promessa, que
* significa que os valores podem ser desquedados antes de serem
preso.O
* a classe implementa
pode
* ser usado com o loop for/aguardar (que não terá terminado
até
* O método Close () é chamado.)
*/
classe Asyncqueue {
construtor () {
// valores que foram na fila, mas ainda não
são armazenados aqui
this.values ??= [];
// quando as promessas são desquedas antes de seus
Os valores correspondentes são
// na fila, os métodos de resolução para essas promessas são
armazenado aqui.
this.resolvers = [];
// Uma vez fechado, não há mais valores pode ser inserido e
Não 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 promessa
const resolve = this.resolvers.shift ();
resolver (valor);
}
outro {
// caso contrário, fila
this.values.push (valor);
}
}
dequeue () {
if (this.values.length> 0) {
// Se houver um valor na fila, retorne um resolvido
Promessa para isso
const valor = this.values.shift ();
return promey.resolve (valor);
}
else if (this.closed) {
// se não houver valores na fila e estamos fechados, retorne um
resolvido
// 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 posterior
retornar nova promessa ((resolve) => {
this.resolvers.push (resolve);});
}
}
fechar() {
// Uma vez que a fila estiver fechada, não serão mais valores
preso.
```

```
// resolva as promessas pendentes com o final de
marcador 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 EOS
Sentinel se formos
// fechado.Aqui, precisamos devolver uma promessa de que
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 de
fluxo "quando fechado
AsyncQueue.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 tratado 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 como
um fluxo de eventos
Função EventsTream (ELT, Type) {
const q = new AsyncQueue ();// Crie a
fila
ELT.AddeventListener (tipo, e => q.enqueue (e));// Enqueue
eventos
retornar q;
}
Função assíncreada HandleKeys () {
// Obtenha um fluxo de eventos de KeyPress e loop uma vez para cada
um
para aguardar (const Event of EventStream (documento,
"KeyPress")) {
console.log (event.key);
}
}
13.5 Resumo
Neste capítulo, você aprendeu:
A maioria da programação JavaScript no mundo real é assíncrona.
Tradicionalmente, a assincronia foi tratada com eventos e
Funções de retorno de chamada.Isso pode ficar complicado, no entanto,
Porque você pode acabar com vários níveis de retorno de chamada
aninhado dentro de outros retornos de chamada, e porque é difícil de fazer
Manuseio 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 usar
incorretamente), eles podem converter código assíncrono que
foram aninhados em cadeias lineares de então () chamadas onde
Um 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
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 itemerável de forma assíncrona
Objetos implementando um [Symbol.asynciterator]
() Método ou invocando uma função assíncrona *
Função do gerador.Iteradores assíncronos fornecem um
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.E
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 um
A propriedade pode mudar.
O atributo enumerável especifica se a propriedade é
enumerado pelo loop for/in e o object.keys ()
método.
O atributo configurável especifica se uma propriedade pode ser
excluído e também se os atributos da propriedade podem ser
mudado.
Propriedades definidas em literais de objeto ou por atribuição comum a um
O objeto é gravável, enumerável e configurável.Mas muitos dos
Propriedades definidas pela biblioteca padrão JavaScript não são.
Esta seção explica a API para consultar e definir a propriedade
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
determinado 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 um
A propriedade usa um objeto chamado Descritor de propriedade para representar o conjunto
de quatro atributos.Um objeto de descritor de propriedades possui propriedades com o
Os mesmos nomes que os atributos da propriedade descrevem.Assim, o
O objeto do descritor de propriedades de uma propriedade de dados possui propriedades nomeadas
valor, gravável, enumerável e configurável.E o
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 Set
Propriedades 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 leitura
const 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 herdadas
isso não existe.
Object.getownPropertyDescriptor ({}, "x") // =>
indefinido;Não tal Prop
Object.getownPropertyDescriptor ({}, "ToString") // =>
indefinido;herdado
Como o próprio nome indica, object.GetownPropertyDescriptor ()
Funciona apenas para propriedades próprias.Para consultar os atributos de herdado
Propriedades, você deve atravessar explicitamente a cadeia de protótipo.(Ver
Object.getProTypeOf () em §14.3);Veja também o semelhante
Reflete.getownPropertyDescriptor () Função em §14.6.)
Para definir os atributos de uma propriedade ou criar uma nova propriedade com o
atributos especificados, chamado object.DefineProperty (), passando o
objeto 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 entusiasmada
O.x // => 1
Object.Keys (O) // => []
// Agora modifique a propriedade x para que seja somente leitura
Object.DefineProperty (O, "X", {Writable: false});
// Tente alterar o valor da propriedade
O.x = 2;// falha silenciosamente ou joga TypeError em rigoroso
modo
O.x // => 1
// A propriedade ainda está configurável, para que possamos mudar seu
valor como este:
Object.DefineProperty (O, "X", {value: 2});
O.x // => 2
// agora mude x de uma propriedade de dados para uma propriedade de acessórios
Object.DefineProperty (O, "X", {get: function () {return 0;}
});
O.x // => 0
```

O descritor da propriedade que você passa para objeto.DefineProperty () não precisa incluir todos os quatro atributos.Se você está criando um novo Propriedade, os atributos omitidos são considerados falsos ou indefinido.Se você está modificando uma propriedade existente, então o Os atributos que você omitem simplesmente permanecem inalterados.Observe que este método altera uma propriedade própria existente ou cria uma nova propriedade própria, mas não alterará uma propriedade herdada.Veja também a função muito semelhante Reflete.DefineProperty () em §14.6.

Se você deseja criar ou modificar mais de uma propriedade por vez, use Object.DefineProperties ().O primeiro argumento é o objeto

Erro ao traduzir esta página.

Object.DefineProperties () atirar TypeError se a tentativa de
Criar ou modificar uma propriedade não é permitida.Isso acontece se você tentar
Para adicionar uma nova propriedade a um objeto não extensível (consulte §14.2).O outro
razões que esses métodos podem lançar o TypeError têm a ver com o
atributos.O atributo gravável governa as tentativas de
Altere o atributo de valor.E o atributo configurável governa
tenta mudar os outros atributos (e também especifica se um
A propriedade pode ser excluída).As regras não são completamente diretas,
no entanto.É possível alterar o valor de uma propriedade não escritor se
Essa propriedade é configurável, por exemplo.Além disso, é possível mudar
uma propriedade de gravidade a não escritor, mesmo que essa propriedade seja
Não Configurável.Aqui estão as regras completas.Chamadas para
Object.DefineProperty () ou
Object.DefineProperties () que tentam violá -los
Jogue um TypeError:
Se um objeto não for extensível, você pode editar seu próprio
Propriedades, mas você não pode adicionar novas propriedades.
Se uma propriedade não estiver configurável, você não pode alterar seu
atributos configuráveis ??ou enumeráveis.
Se uma propriedade acessadora não estiver configurável, você não poderá mudar
Seu método getter ou setter, e você não pode alterá -lo para um dados
propriedade.
Se uma propriedade de dados não estiver configurável, você não pode alterá -lo para
uma propriedade acessadora.
Se uma propriedade de dados não estiver configurável, você não pode alterar seu
atributo gravável de false para verdadeiro, mas você pode mudar
de 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 um
propriedade que é configurável, mas não escritor, no entanto
(porque isso seria o mesmo que torná -lo gravável, então
Alterar o valor e convertê -lo novamente em não escritura).
§6.7 descreveu a função object.assign () que copia a propriedade
valores de um ou mais objetos de origem em um objeto de destino.
Object.assign () apenas copia propriedades enumeráveis ??e propriedades
valores, não atributos de propriedade.Isso é normalmente o que queremos, mas
significa, por exemplo, que se um dos objetos de origem tiver um
Propriedade do acessador, é o valor retornado pela função getter que é
Copiado para o objeto de destino, não a própria função getter.Exemplo 14-1
demonstra como podemos usar
Object.GetownProperTyDescriptor () e
Object.DefineProperty () para criar uma variante de
Object.assign () que copia descritores inteiros de propriedades
do que apenas copiar valores de propriedades.
Exemplo 14-1.Copiar propriedades e seus atributos de um objeto
para outro
```
/*
* Definir um novo object.assignDescriptors () Função que funciona
como
* Object.assign (), exceto que ele copia os descritores de propriedades
de
* fonte de objetos no objeto de destino em vez de apenas
cópia
* valores de propriedade.Esta função copia todas as próprias propriedades,
ambos
* enumerável e não enumerável.E porque copia
descritores,
* Copia as funções getter de objetos de origem e
substitui o setter
* funciona no objeto de destino em vez de invocar aqueles
getters 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 o
Valores da propriedade
Seja q = object.assignDescriptors ({}, o);// copie o
Descritores de propriedades
p.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
incrementos.

14.2 Extensibilidade do objeto

O atributo extensível de um objeto especifica se novas propriedades
pode ser adicionado ao objeto ou não.Objetos JavaScript comuns são
extensível por padrão, mas você pode mudar isso com as funções
descrito 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
o tornaram não extensível.Observe também que chama
Object.PreventExtensions () afeta apenas a extensibilidade de
o 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, reflete.isExtensible () e
Reflete.PreventExtensions (), são descritos em §14.6.
O objetivo do atributo extensível é poder "travar"
objetos em um estado conhecido e impedem adulteração externa.O
O atributo extensível dos objetos é frequentemente usado em conjunto com o
Atributos configuráveis ??e graváveis ??de propriedades e javascript
Define funções que facilitam a definição desses atributos:
Object.Seal () funciona como
Object.PreventExtensions (), mas além de
Tornando o objeto não extensível, também faz com que todos
Propriedades desse objeto não configuráveis.Isso significa que novo
propriedades não podem ser adicionadas ao objeto e existente
As propriedades não podem ser excluídas ou configuradas.Propriedades existentes
que são graváveis ??ainda podem ser definidos, no entanto.Não há como
Desperte um objeto selado.Você pode usar object.iSiSealed ()
para determinar se um objeto está selado.
Object.freeze () trava objetos ainda mais firmemente.
Além de tornar o objeto não estetensivo e seu
Propriedades não confundíveis, também faz com que todos os objeto
Propriedades de dados próprios somente leitura.(Se o objeto tiver acessador
propriedades com métodos de setter, estes não são afetados e podem
ainda ser chamado por atribuição à propriedade.) Use
Object.isfrozen () para determinar se um objeto está congelado.
É importante entender que objeto.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 no
Cadeia de protótipo também.
Object.PreventExtensions (), object.Seal () e
Object.freeze () todos retornam o objeto que eles passam, que
significa que você pode usá -los em invocações de funções aninhadas:
// Crie um objeto selado com um protótipo congelado e um não
propriedade enumerável
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
modificando -os.Isso é fácil e conveniente, mas há comércio
OFFs: Objetos congelados podem interferir nos testes JavaScript comuns
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 dizem "o protótipo de O" em vez de "o protótipo
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
criado a partir de literais de objeto, use objeto.prototipo como seu
protótipo.Objetos criados com novo use o valor do protótipo
propriedade de sua função construtora como protótipo.E objetos
criado com object.create () use o primeiro argumento para isso
função (que pode ser nula) como seu protótipo.
Você pode consultar o protótipo de qualquer objeto, passando esse objeto para
Object.getProTypeOf ():
Object.getPrototypeOf ({}) // => object.prototype
Object.getPrototypeOf ([]) // => Array.prototype
Object.getPrototypeOf (() => {}) // => function.prototype
Uma função muito semelhante, reflete.getPrototypeof (), é
descrito em §14.6.
Para determinar se um objeto é o protótipo de (ou faz parte do
Cadeia de protótipo de) Outro objeto, use o ISPrototypeof ()
método:
Seja p = {x: 1};// Defina um protótipo
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.SetProTypeOf ():
Seja o = {x: 1};
Seja p = {y: 2};
Object.SetProTypeOf (O, P);// Defina o protótipo de O para P
O.y // => 2: o agora herda a propriedade y
Seja a = [1, 2, 3];
Object.SetProTypeOf (a, p);// defina o protótipo da matriz a
principal
A.Join // => indefinido: A não tem mais um método de junção ()
Geralmente não há necessidade de usar
Object.SetProTypeOf ().As implementações de JavaScript podem
fazer otimizações agressivas com base na suposição de que o
O protótipo de um objeto é fixo e imutável.Isso significa que se você
sempre ligue para object.SetProTypeOf (), qualquer código que use o
Objetos alterados podem ser executados muito mais lentos do que normalmente.
Uma função semelhante, reflet.setProtTypeOf (), 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.(Suportes 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.getProTypeOf () e
Object.SetProTypeOf ().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 P
```

14.4 Símbolos bem conhecidos

O tipo de símbolo foi adicionado ao JavaScript em ES6, e um dos
As 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
aprendi que você pode tornar uma classe iterável implementando um método
cujo "nome" é o símbolo símbolo.iterator.
Symbol.iterator é o exemplo mais conhecido do ?conhecido
Símbolos. ?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.O
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 símbolos.iterator e símbolo.asynciterator
permitir que objetos ou classes se tornem iteráveis ??ou assíncronos
iterável.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.HasInsinStance

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.Prototipo 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 do
Instância do operador.E, é claro, se o valor no caminho
lado não tem um método [símbolo.hasinstance], mas é um
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 instância
Seja uint8 = {
[Symbol.hasinstance] (x) {
retornar número.isinteger (x) && x> = 0 && x <= 255;
}
};
128 instância de uint8 // => true
256 instância de uint8 // => false: muito grande
Math.pi instância de uint8 // => false: não é um número inteiro
```

Observe que este exemplo é inteligente, mas confuso porque usa um Objeto não clássico onde uma classe normalmente seria esperada.Seria tão fácil - e mais claro para os leitores do seu código - para escrever um Isuint8 () função em vez de confiar nisso Symbol.Hasinsance Behavior.

14.4.3 Symbol.ToStringTag

Se 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 (/./) // => "[Objeto
Regexp] "
Object.prototype.toString.Call (() => {}) // => "[Objeto
Função]"
Object.prototype.toString.Call ("") // => "[Objeto
Corda]"
Object.prototype.toString.Call (0) // => "[Objeto
```

Número]"
Object.prototype.toString.Call (false) // => "[Objeto
Booleano] "
Acontece que você pode usar isso
Object.prototype.toString (). Call () Técnica com qualquer
Valor Javascript para obter o "atributo de classe" de um objeto que
Contém informações de tipo que não estão disponíveis de outra forma.A seguir
a função de classe () é indiscutivelmente mais útil do que o tipo de
Operador, 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"
classf (() => {}) // => "função"
Classof (novo mapa ()) // => "mapa"
Classof (new Set ()) // => "Set"
classe de (new Date ()) // => "Data"
Antes do ES6, esse comportamento especial do
Object.prototype.toString () Método estava disponível apenas para
Instâncias de tipos internos e se você chamou essa função de classe ()
Em uma instância de uma aula que você havia definido, simplesmente
retornar "objeto".Em ES6, no entanto,

Object.prototype.toString () procura uma propriedade com o
Nome simbólico Symbol.ToStringTag em seu argumento, e se tal
Existe 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.Spécies

Antes do ES6, o JavaScript não forneceu nenhuma maneira real de criar robusto
subclasses de classes internas como a matriz.Em ES6, no entanto, você pode
estender qualquer classe interna simplesmente usando a classe e estende
palavras -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 e
F.Last // => 9: F também é um ezarray com uma última propriedade
```

Array define métodos concat (), filtro (), map (), slice (),

e Splice (), que retornam matrizes.Quando criamos uma matriz
subclasse como ezarray que herda esses métodos, deve ser herdado
Instâncias de retorno do método de matriz ou instâncias de ezarray?Bom
Argumentos podem ser feitos para qualquer opção, mas a especificação ES6
diz que (por padrão) os cinco métodos de retorno de matriz retornarão
Instâncias 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 isso
O símbolo é usado como o nome de uma propriedade do construtor
função.A maioria dos outros símbolos bem conhecidos descritos
Aqui são usados ??como o nome dos métodos de um objeto de protótipo.)
Quando criamos uma subclasse com estendências, o resultante
O construtor de subclasse herda as propriedades da superclasse
construtor.(Isso é um acréscimo ao tipo normal de
herança, onde casos dos métodos de herdamento da subclasse de
a superclasse.) Isso significa que o construtor para cada
A subclasse da Array também possui uma propriedade herdada com nome
Symbol.Spécies.(Ou uma subclasse pode definir seu próprio
propriedade com este nome, se quiser.)
Métodos como map () e slice () que criam e retornam
Novas matrizes são ligeiramente aprimoradas no ES6 e mais tarde.Em vez de
Apenas criando uma matriz regular, eles (com efeito) invocam novos
this.Constructor [Symbol.Species] () para criar
a nova matriz.
Agora aqui está a parte interessante.Suponha que isso
Array [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 // => 3
F.Last // => indefinido: f é uma matriz regular sem último
getter
```

Criar subclasses úteis de matriz foi o principal caso de uso que
motivou a introdução de símbolo.
único local que este símbolo bem conhecido é usado.Aulas de matriz digitadas
Use o símbolo da mesma maneira que a classe da matriz.De forma similar,
O método Slice () de ArrayBuffer olha para o
Symbol.pécies bens this.Constructor em vez de
Simplesmente criando um novo ArrayBuffer.E promessa métodos como
então () que retornam novos objetos de promessa criam esses objetos por meio disso
Protocolo 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.iscoNcatsPreadable

O método da matriz concat () é um dos métodos descritos no
seção anterior que usa símbolo.pécies para determinar o que
construtor para usar para a matriz retornada.Mas concat () também usa
Symbol.iscoNcatsPreadable.Lembre -se do §7.8.3 de que o
O método concat () de uma matriz trata seu valor e sua matriz
Argumentos de maneira diferente dos seus argumentos não provocados: argumentos não marcantes
são simplesmente anexados à nova matriz, mas a matriz e qualquer

Os argumentos da matriz são achatados ou "espalhados" para que os elementos do
A matriz é concatenada em vez do próprio argumento da matriz.
Antes do ES6, concat () acabou de usar o Array.esArray () 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 tem uma propriedade com o nome simbólico
Symbol.iscoNcatsPreadable, então o valor booleano daquele
A propriedade é usada para determinar se o argumento deve ser
"espalhar."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 comportar como uma matriz real quando passada para concat (), você pode
Basta adicionar a propriedade simbólica ao seu objeto:
Deixe a matriz = {
Comprimento: 1,
0: 1,
[Symbol.iscoCatsPreadable]: 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:
1

```
classe nãopreadableArray estende a matriz {
get [symbol.isconcatspreadable] () {
retornar falso;}
}
Seja a = novo não adolescente (1,2,3);
[] .CONCAT (a) .Length // => 1;(seria 3
elementos 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 padrões operações 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 conhecido Símbolo: Symbol.Match, Symbol.Search, e assim por diante.

Os regexps são uma maneira geral e muito poderosa de descrever textual padrões, mas eles podem ser complicados e não adequados para confusos correspondência.Com os métodos generalizados de string, você pode definir seu Classes de padrões próprios usando os métodos de símbolo bem conhecidos para fornecer correspondência personalizada.Por exemplo, você pode executar comparações de string Usando o Intl.Collator (consulte o §11.7.3) para ignorar sotaques ao corresponder.Ou você pode definir uma classe de padrão baseada no algoritmo SoundEx para Combine as palavras com base em seus sons aproximados ou para combinar vagamente Strings 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 globs:

```
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.
// Globs 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 caractere
0
"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
convertido em valores primitivos.Para fazer isso, defina um método com isso
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 é
fazendo a conversão em um contexto em que esperaria ou
prefira (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 é
fazendo a conversão em um contexto em que esperaria ou
prefira (mas não exigir) um valor numérico.Isso acontece quando
você usa o objeto com um operador <ou> ou com aritmética
operadores como - e *.
Se o argumento for "padrão", significa que o javascript é
convertendo seu objeto em um contexto em que um numérico ou
O valor da string pode funcionar.Isso acontece com o +, == e! =
operadores.
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
comparável e classificável com <e>, então esse é um bom motivo para
Defina 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 com
um objeto e executa seu corpo de declaração como se estivesse em um escopo onde
As propriedades desse objeto eram variáveis.Isso causou compatibilidade
problemas quando novos métodos foram adicionados à aula de matriz, e isso
quebrou 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 calcula
Object.Keys (o [symbol.unscopables] || {}) e ignora
propriedades cujos nomes estão na matriz resultante ao criar o
escopo simulado para executar seu corpo.ES6 usa isso para adicionar novo
Métodos para Array.prototype sem quebrar o código existente em
a web.Isso significa que você pode encontrar uma lista da matriz mais recente
Métodos avaliando:

```
Deixe newArrayMethods =
Object.Keys (Array.prototype [symbol.unscopables]);
```

14.5 Tags de modelo

Strings 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 como
metaprogramação, porque os modelos marcados são frequentemente usados ??para definir
DSLS-idiomas específicos de domínio-e definir uma nova função de tag é
Como adicionar nova sintaxe ao JavaScript.Modelos marcados que os literais têm
foi 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 tag
funções como essas.
Não há nada de especial nas funções de tags: elas são comuns
As funções JavaScript e nenhuma sintaxe especial são necessárias para defini -las.
Quando uma expressão de função é seguida por um modelo literal, o
A função é invocada.O primeiro argumento é uma variedade de cordas, e isso é
seguido por zero ou mais argumentos adicionais, que podem ter valores
de qualquer tipo.
O número de argumentos depende do número de valores que são
interpolado no modelo literal.Se o modelo literal é simplesmente um
string constante sem interpolações, então a função de tag será
chamado com uma matriz dessa corda e nenhum argumento adicional.Se
O modelo literal inclui um valor interpolado, depois a tag
A função é chamada com dois argumentos.O primeiro é uma variedade de dois
Strings, e a segunda é o valor interpolado.As cordas nisso
matriz inicial são a string à esquerda do valor interpolado e o
String à sua direita, e qualquer um deles pode ser a corda vazia.Se o
Modelo literal inclui dois valores interpolados, depois a função da tag
é invocado com três argumentos: uma variedade de três cordas e os dois
valores interpolados.As três cordas (uma ou todas as quais podem ser
vazios) são o texto à esquerda do primeiro valor, o texto entre os dois
valores e o texto à direita do segundo valor.No caso geral,
Se o modelo literal tiver n valores interpolados, a função da tag
será invocado com n+1 argumentos.O primeiro argumento será um

Matriz de N+1 Strings, e os argumentos restantes são os n
Os valores interpolados, na ordem em que aparecem no modelo literal.
O valor de um modelo literal é sempre uma string.Mas o valor de um
O 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 executa
HTML escape em cada um dos valores antes de usá -lo para construir o final
corda:

```
função html (strings, ... valores) {
// converte cada valor em uma string e escape de HTML especial
caracteres
Deixe escapar = valores.map (v => string (v)
.place ("&", "& amp;")
.place ("<", "& lt;")
.place (">", "& gt;")
.place ("","" "")
.place ("'", "&#39;"));
// retorna as cordas concatenadas e valores escapados
deixe resultado = strings [0];
para (vamos i = 0; i <escapado.length; i ++) {
resultado += escape [i] +strings [i +1];
}
resultado de retorno;
}
deixe operator = "<";
html` <b> x $ {operator} y </b> `// =>" <b> x & lt;
```

y "
Let Kind = "Game", Name = "D&D";
html` <div class = "$ {Kind}"> $ {name} </div> `// => '<div
class = "Game"> d & amp; d '
Para um exemplo de uma função de tag que não retorna uma string, mas
Em vez disso, uma representação analisada de uma corda, pense no globo
Classe de padrão definida no §14.4.6.Como o construtor glob () leva um
argumento de string única, podemos definir uma função de tag para criar novos
Objetos Glob:
função glob (strings, ... valores) {
// monta as cordas e valores em uma única string
Seja s = strings [0];
para (vamos i = 0; i <valores.length; i ++) {
s += valores [i] +strings [i +1];
}
// retorna uma representação analisada dessa string
Retornar New Glob (s);
}
deixe root = "/tmp";
deixe filepattern = glob` $ {root}/*. html`;// um regexp
alternativa
"/tmp/test.html".match(FilePattern):1] // =>" teste "
Um dos recursos mencionados de passagem no §3.3.4 é o
Função de tag String.raw`` que retorna uma string em sua forma "RAW"
sem interpretar nenhuma das sequências de fuga de barragem.Isso é
implementado usando um recurso de invocação de funções de tag que temos
ainda não discutido.Quando uma função de tag é invocada, vimos que é
O primeiro argumento é uma variedade de cordas.Mas esta matriz também tem uma propriedade
chamado 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 bruta
Inclui strings nas quais as sequências de fuga não são interpretadas.Esse
O 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
Uso 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
imitar 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.E,
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 -a
como uma função sem nenhum valor se o for nulo) e passa o
valores na matriz args como argumentos.É equivalente a
F.Apply (O, Args).
Reflet.Construct (C, Args, NewTarget)
Esta função chama o construtor C como se a nova palavra -chave tivesse
foi usado e passa os elementos da matriz args como argumentos.
Se o argumento opcional newTarget for especificado, ele é usado como o
Valor de New.Target dentro da invocação do construtor.Se não
Especificado, o valor do novo.Target será c.
Reflete.DefineProperty (O, nome, descritor)
Esta função define uma propriedade no objeto O, usando o nome (a
string ou símbolo) como o nome da propriedade.O descritor
Objeto deve definir o valor (ou getter e/ou setter) e atributos
da propriedade.Reflete.DefineProperty () é muito semelhante
para object.DefineProperty (), mas retorna true no sucesso
e false em falhas.(Object.DefineProperty () retorna
o 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 (ou
se não existisse essa propriedade) e falsa se a propriedade não pudesse ser
excluído.Chamar esta função é semelhante a escrever delete
o [nome].
Reflete.get (o, nome, receptor)
Esta função retorna o valor da propriedade de O com o
Nome especificado (uma string ou símbolo).Se a propriedade for um acessador

método com um getter, e se o argumento do receptor opcional for
especificado, então a função getter é chamada de método de
receptor em vez de como método de o.Chamar esta função é
Semelhante a avaliar o [nome].
Reflet.GetownPropertyDescriptor (O, nome)
Esta função retorna um objeto de descritor de propriedade que descreve o
atributos da propriedade nomeada nome do objeto O, ou retorna
indefinido se não existir tal propriedade.Esta função é quase
idêntico a object.GetownPropertyDescriptor (),
exceto que o reflete a versão da API da função exige que o
O 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 o
Objeto não possui protótipo.Joga um TypeError se o é um primitivo
valor em vez de um objeto.Esta função é quase idêntica a
Object.getProTypeOf (), exceto isso
Object.getPrototypeOf () apenas joga um TypeError para
Argumentos nulos e indefinidos e coages outros primitivos
valores para seus objetos de wrapper.
Reflete.Has (o, nome)
Esta função retorna true se o objeto O tiver uma propriedade com o
Nome especificado (que deve ser uma string ou um símbolo).Chamando isso
A função é semelhante a avaliar o nome em o.
Reflete.isextensible (O)
Esta função retorna true se o objeto O for extensível (§14.2) e
Falso se não for.Ele lança um TypeError se O não for um objeto.
Object.isExtensible () é semelhante, mas simplesmente retorna
Falso 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 for
não extensível ou se a operação causaria um protótipo circular

corrente).Joga um TypeError se o não for um objeto ou se p não for
um 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 fará
Seu código mais lento, interrompendo o intérprete JavaScript
otimizações.

14.7 Objetos de proxy

A aula de proxy, disponível no ES6 e mais tarde, é mais
poderoso recurso de metaprogramação.Nos permite escrever código que
altera o comportamento fundamental dos objetos JavaScript.A API refletida
descrito no §14.6 é um conjunto de funções que nos dá acesso direto a um
Conjunto de operações fundamentais em objetos JavaScript.Que proxy
A classe faz é nos permite uma maneira de implementar esses fundamentais
operações de nós mesmos e criam objetos que se comportam de maneiras que não são
possível para objetos comuns.
Quando criamos um objeto de proxy, especificamos dois outros objetos, o alvo
objeto 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 um
propriedade, defina uma nova propriedade, procure o protótipo, invocar como um
função), ele despacha essas operações para o objeto de manipuladores ou para o
objeto 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 reflete.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 do
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 (accessthedatabase, {});
proxy () // => 42: o proxy dá acesso ao subjacente
função alvo
revogar();// 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 como
bem 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 certo dos manipuladores, o objeto alvo subjacente se torna essencialmente irrelevante.
No código a seguir, por exemplo, é como poderíamos implementar um objeto que parece ter um número infinito de propriedades somente leitura, onde o valor de cada propriedade é o mesmo que o nome do propriedade:

```
// usamos um proxy para criar um objeto que parece ter
todo
// Propriedade possível, com o valor de cada propriedade igual
para seu nome
Seja identidade = novo proxy ({}, {
// Cada propriedade tem seu próprio nome como seu valor
obtenha (o, nome, destino) {return name;},
// Cada nome de propriedade é definido
tem (o, nome) {return true;},
// Existem muitas propriedades para enumerar, então apenas nós
lançar
OwnKeys (O) {jogue novo RangeError ("Número Infinito de
propriedades ");},
// 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 definidas
set (o, nome, valor, destino) {return false;},
// Todas as propriedades não são confundíveis, então não podem ser
excluído
deleteProperty (o, nome) {return false;},
// todas as propriedades existem e não são confundíveis, então nós
Não posso definir mais
defineProperty (o, nome, desc) {return false;},
```

```
// Com efeito, isso significa que o objeto não é
extensível
isExtensible (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 Jogue 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 a
Faça isso
* aparecerá nas mensagens de log para identificar o objeto.Se
o tem próprio
* propriedades cujos valores são objetos ou funções, então se
você pergunta
* O valor dessas propriedades, você receberá um LoggingProxy
de 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
propriedades
// cujo valor é um objeto ou função, ele retorna um
Proxy, em vez disso
// do que retornar o próprio valor.
Get (Target, Property, Receiver) {
// Logre a operação Get
Console.log (`Handler
get ($ {objName}, $ {Property.toString ()}) `);
// Use a API refletir para obter o valor da propriedade
Deixe 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) &&
(tipoof valor === "objeto" || TIPOF 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 seguintes
Métodos:
// eles registram a operação e delegados ao alvo
objeto.
// eles são um caso especial simplesmente para que possamos evitar
registrando o
// objeto receptor que pode causar infinito
Recursão.
SET (Target, Prop, Valor, Receptor) {
Console.log (`Handler
set ($ {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 do
manipuladores.
// metaprograma ftw!
Reflete.ownskeys (refletir) .ForEach (HandlerName => {
if (! (nome do manipulador em manipuladores)) {
Manipuladores [nome do manutenção] = função (Target, ... args)
{
// registrar a operação
console.log (`manipulador $ {handlername}
($ {objName}, $ {args}) `);
// Delegar a operação
retorno reflete [handlername] (Target, ... args);
};
}
});
```

// retorna um proxy para o objeto usando esses registros
manipuladores
retornar novo proxy (O, manipuladores);
}
A função LoggingProxy () definida anteriormente cria proxies que
registre todas as maneiras pelas quais eles são usados.Se você está tentando entender como
Uma função não documentada usa os objetos que você passa, usando um registro
Proxy pode ajudar.
Considere os seguintes exemplos, que resultam em alguns genuínos
Insights sobre a iteração da matriz:
// define uma matriz de dados e um objeto com uma função
propriedade
deixe dados = [10,20];
deixe métodos = {quadrado: x => x*x};
// Crie proxies de registro para a matriz e o objeto
deixe proxydata = loggingproxy (dados, "dados");
deixe proxymethods = loggingproxy (métodos, "métodos");
// Suponha que queremos entender como o método da matriz.map ()
funciona
data.map (métodos.square) // => [100, 400]
// Primeiro, vamos tentar com uma matriz proxy de madeira
proxydata.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 proxy
data.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 o
Protocolo de iteração
para (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 o
Método Array.map () verifica explicitamente a existência de cada
elemento da matriz (fazendo com que o manipulador tenha () seja invocado) antes
Na verdade, lendo o valor do elemento (que aciona o manipulador get ()).
Presumivelmente, isso pode distinguir elementos de matriz inexistentes
de elementos que existem, mas indefinidos.
O segundo pedaço de saída de madeira pode nos lembrar que a função
Passamos para Array.map () é invocado com três argumentos: o
valor do elemento, o índice do elemento e a própria matriz.(Há um
Problema em nossa saída de registro: o método Array.toString ()
não inclui suportes quadrados em sua saída e as mensagens de log
seria 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/of
funciona procurando um método com nome simbólico

[Symbol.iterator].Também demonstra que a aula de matriz
A implementação deste método de iterador é cuidadoso para verificar a matriz
comprimento em cada iteração e não assume que o comprimento da matriz
permanece constante durante a iteração.

14.7.1 Invariantes de procuração

A função readOnlyProxy () definida anteriormente cria proxy
Objetos que são efetivamente congelados: qualquer tentativa de alterar um valor de propriedade
ou atributo de propriedade ou para adicionar ou remover propriedades
exceção.Mas enquanto o objeto de destino não estiver congelado, descobriremos que
Se pudermos consultar o proxy com reflete.isextensible () e
Reflet.GetownPropertyDescriptor (), e ele nos dirá
que devemos ser capazes de definir, adicionar e excluir propriedades.Então
ReadOnlyProxy () cria objetos em um estado inconsistente.Nós
poderia consertar isso adicionando isextensible () e
getOwnPropertyDescriptor () manipuladores, ou podemos simplesmente viver
com esse tipo de inconsistência menor.
A API de manipulador de proxy nos permite definir objetos com major
inconsistências, no entanto, e neste caso, a própria classe de procuração irá
impedir -nos de criar objetos de proxy que são inconsistentes em um mau
caminho.No início desta seção, descrevemos proxies como objetos com
nenhum comportamento próprio porque eles simplesmente encaminham todas as operações para
os manipuladores objeto e o objeto de destino.Mas isso não é totalmente verdadeiro:
Depois de encaminhar uma operação, a classe proxy realiza alguma sanidade
verificações no resultado para garantir que invariantes importantes de JavaScript não sejam
sendo violado.Se detectar uma violação, o proxy jogará um
TypeError em vez de deixar a operação prosseguir.

Como exemplo, se você criar um proxy para um objeto não extensível, o
Proxy 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: Invariant
```

violação
De acordo, objetos de proxy para alvos não extensíveis podem não ter um
getPrototypeOf () manipulador que retorna qualquer coisa que não seja o
Objeto de protótipo real do alvo.Além disso, se o objeto de destino tiver
Propriedades não escritas e não confundíveis, então a classe de proxy irá
Jogue um TypeError se o manipulador get () retornar qualquer coisa que não seja
O 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 destino
Proxy aplica vários invariantes adicionais, quase todos eles
tendo a ver com objetos-alvo não estetensivos e não-configurável
propriedades no objeto de destino.

14.8 Resumo

Neste capítulo, você aprendeu:
Os objetos JavaScript têm um atributo e objeto extensíveis
As 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 o
Cadeia de protótipo de um objeto e até para 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 um
A 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
O argumento da string de modelo permite incorporar DSLs dentro
Código JavaScript.As funções de tags também fornecem acesso a um cru,
forma não descontada de literais de cordas onde as barris 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).

1
Um bug no mecanismo V8 JavaScript significa que este código não funciona corretamente no nó 13.

Capítulo 15. JavaScript na Web

Navegadores

A linguagem JavaScript foi criada em 1994 com o propósito expresso de ativar o comportamento dinâmico nos documentos exibidos pela Web navegadores.O idioma evoluiu significativamente desde então, e no ao mesmo tempo, o escopo e as capacidades da plataforma da web cresceram explosivamente.Hoje, os programadores JavaScript podem pensar na web como um plataforma completa para desenvolvimento de aplicativos.Navegadores da web especialize -se na exibição de texto e imagens formatados, mas como nativo Sistemas operacionais, os navegadores também fornecem outros serviços, incluindo Gráficos, vídeo, áudio, networking, armazenamento e encadeamento.JavaScript é o idioma que permite que os aplicativos da web usem os serviços fornecido pela plataforma da web, e este capítulo demonstra como você pode 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) e Como o código JavaScript é acionado de forma assíncrona por eventos (§15.2). As seções que seguem este material introdutório documentam o núcleo JavaScript 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)
Trocar 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 cliente
Neste livro, e na web, você verá o termo "JavaScript do lado do cliente".O termo é simplesmente um
Sinônimo de JavaScript escrito para executar em um navegador da web e contrasta com o código "do lado do servidor",
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 o navegador 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 todos
Javascript APIs definidas pelos navegadores da web e, como resultado, este livro
Foi há muito tempo há uma década.O número e a complexidade das APIs da Web
continuou a crescer, e eu não acho mais que faz sentido tentar
Para cobrir todos eles em um livro.Na sétima edição, meu objetivo é
cubra a linguagem JavaScript definitivamente e para fornecer uma profundidade
Introdução 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 mais
importantes com detalhes suficientes para que você possa começar a usá -los certos
ausente.E, tendo aprendido sobre as APIs principais cobertas aqui, você
deve 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
O impacto do 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 definisse o
cor 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 em
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
Aprenda 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.

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 por
respondendo a eventos.

15.1.1 JavaScript em tags HTML <Script>

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 html
arquivo que inclui uma tag de script com código JavaScript que dinamicamente
atualiza um elemento do documento para fazê -lo se comportar como um digital
relógio:

```
<! Doctype html> <!-Este é um arquivo html5-
>
<html> <!-o elemento raiz->
<head> <!- título, scripts e estilos
pode ir aqui ->
<title> relógio digital </title>
<estilo> /* uma folha de estilo CSS para o
relógio */
#clock { /* estilos se aplicam ao elemento
com id = "relógio" */
Fonte: negrito 24px sem serrif;/ * Use uma fonte grande em negrito */
Antecedentes: #ddf;/* Em um leve cinza azulado
fundo.*/
preenchimento: 15px;/* Cercá -lo com alguns
espaço */
```

```
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 de
o documento.->
<h1> relógio digital </h1> <!- exibir um título.->
<span id = "relógio"> </span> <!- Vamos inserir o tempo em
este elemento.->
<Cript>
// Defina uma função para exibir o horário atual
Função DisplayTime () {
deixe clock = document.QuerySelector ("#relógio");// Pegar
elemento com id = "relógio"
deixe agora = new Date ();// Pegar
horário atual
clock.TextContent = agora.tolocaletimestring ();// Mostrar
Tempo no relógio
}
displaytime () // exibe o tempo certo
ausente
setInterval (DisplayTime, 1000);// e depois atualize -o a cada
segundo.
</script>
</body>
</html>
```

Embora o código JavaScript possa ser incorporado diretamente dentro de um Tag <cript>, 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 disso 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> ou
Qualquer outro html.Por convenção, os arquivos do código JavaScript têm nomes
Esse fim com .js.
Uma tag <cript> com o atributo A SRC se comporta exatamente como se o
O 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 é
Especificado: HTML não suporta uma tag <script/>.
Há várias vantagens em usar o atributo SRC:
Ele simplifica seus arquivos HTML, permitindo que você remova
Grandes blocos de código JavaScript deles - ou seja, ajuda
Mantenha 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 único
cópia desse código, em vez de ter que editar cada arquivo html
Quando o código muda.
Se um arquivo de código JavaScript for compartilhado por mais de uma página, ele
só 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 pode
Empregue código exportado por outros servidores da Web.Muita internet
A 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 usando
módulos (e não usaram uma ferramenta de construção de código para combinar todos os seus
módulos em um único arquivo não modular de javascript), então você deve
Carregue 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 você
Especificar 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 script
Nos primeiros dias da web, pensava -se que os navegadores poderiam alguns
Dia implementa idiomas que não sejam JavaScript e programadores
Atributos adicionados como linguagem = "javascript" e
type = "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 ali
São apenas dois motivos para usar um atributo de tipo em uma tag <cript>:
Para especificar que o script é um módulo
Para incorporar dados em uma página da web sem exibi -los (veja
§15.3.4)
Quando os scripts são executados: assíncronos e adiados
Quando o JavaScript foi adicionado aos navegadores da web, não havia API
para atravessar e manipular a estrutura e o conteúdo de um já
documento renderizado.A única maneira de o código JavaScript afetar o
O conteúdo de um documento era gerar esse conteúdo em tempo real, enquanto o
O 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 assíncicos funcionam enquanto carregam, o que significa que eles podem executar 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 o seu scripts no final do arquivo HTML.Dessa forma, o script pode correr
sabendo que o conteúdo do documento antes de ter sido analisado e é
pronto para ser manipulado.
Carregando scripts sob demanda
Às vezes, você pode ter código JavaScript que não é usado quando um
documentar as primeiras cargas e só 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 <cript> 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 </ititure>
</head>
<Body>
<H1> Um documento html </h1>
<p> Este é um documento simples .
</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
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 dom
A 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 é
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 texto
nós, e para inseri -los no documento como filhos de outros
Objetos de elemento.Também existem métodos para mover elementos dentro
o documento e para removê -los completamente.Enquanto é o servidor
o aplicativo pode produzir saída de texto simples escrevendo seqüências de strings com
Console.log (), um aplicativo JavaScript do lado do cliente pode produzir
Saída HTML formatada construindo ou manipulando a árvore de documentos
Documente 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 um
instância de htmlbodyelement e uma tag <able> 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 <mg>
tags, têm uma propriedade SRC que corresponde ao atributo src do
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 html (e faz com que o navegador

carregar e exibir uma nova imagem).A maioria das classes de elemento JavaScript
Basta refletir os atributos de uma tag html, mas alguns definem
Métodos.As classes HtmlaudioElement e HtmlVideoElement,
por exemplo, defina métodos como play () e pausa () para
Controlando a reprodução de arquivos de áudio e vídeo.

15.1.3 O objeto global nos navegadores da Web

Existe 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 - o
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 http
solicitações 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 global
em si.Isso significa que você pode simplesmente digitar janela para se referir ao
Objeto global no seu código do lado do cliente.Ao usar a janela específica
Recursos, 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 nome
Com módulos, as constantes, variáveis, funções e classes definidas
no nível superior (ou seja, fora de qualquer função ou definição de classe) do
O módulo é privado para o módulo, a menos que sejam exportados explicitamente, em
Que caso, eles podem ser importados seletivamente por outros módulos.(Observação
que essa propriedade dos módulos é homenageada por ferramentas de aglomeração de código como
bem.)
Com scripts não módulos, no entanto, a situação é completamente
diferente.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 outra
O script define uma classe C, então um terceiro script pode invocar a função e
Instanciar a classe sem precisar tomar nenhuma ação para importá -los.
Então, se você não estiver usando módulos, os scripts independentes em seu
documento compartilhe um único espaço para nome e se comportar como se fossem todos parte de
um único script maior.Isso pode ser conveniente para pequenos programas, mas
A necessidade de evitar a nomeação de conflitos pode se tornar problemática para maiores
Programas, especialmente quando alguns dos scripts são bibliotecas de terceiros.
Existem algumas peculiaridades históricas com a forma como este espaço de nome compartilhado
funciona.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
e o documento contido são carregados do mesmo servidor, o
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 dois
fases.Na primeira fase, o conteúdo do documento é carregado e o código
de <sCript> elementos (scripts embutidos e scripts externos) é
correr.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
Condicionais, loops e outras declarações de controle de JavaScript.Alguns
Scripts realmente não fazem nada durante esta primeira fase e, em vez disso, apenas
Defina funções e classes para uso na segunda fase.Outros scripts
pode fazer um trabalho significativo durante a primeira fase e depois não fazer nada em
o 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
orientado a eventos.Se um script vai participar desta segunda fase,
Então uma das coisas que deve ter feito durante a primeira fase é
registre pelo menos um manipulador de eventos ou outra função de retorno de chamada que será
invocou assíncrono.Durante esta segunda fase orientada a eventos, o
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 chamado 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
Eventos "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 documento
Recursos 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
Documente e faça o que for que o programa deve fazer.
Observe que é comum na programação JavaScript para um manipulador de eventos
Função como o manipulador de eventos "Load" descrito aqui para se registrar
Outros 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 o
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 Web
mecanismo.
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
contente.A propriedade Document.ReadyState tem o
valor ?carregamento? nesta fase.
2. Quando o analisador html encontra uma tag <cript> que
não possui nenhum dos assíncronos, adiados, ou
TIPE = atributos "módulo", ele acrescenta essa tag de script ao
Documente 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
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 <cript> e

Erro ao traduzir esta página.

objeto.
8. A partir deste momento, os manipuladores de eventos são invocados de forma assíncrona
Em resposta a eventos de entrada do usuário, eventos de rede, temporizador
Expira, e assim por diante.

15.1.6 Entrada e saída do programa

Como qualquer programa, os programas JavaScript do lado do cliente processam dados de entrada
Para produzir dados de saída.Há uma variedade de insumos disponíveis:
O conteúdo do próprio documento, que o código JavaScript pode
Acesso com a API DOM (§15.3).
Entrada do usuário, na forma de eventos, como cliques de mouse (ou
toques de tela de toque) em elementos html <button> ou texto
Entrou em elementos HTML <Textarea>, por exemplo.
§15.2 demonstra como os programas JavaScript podem responder a
Eventos de usuário como esses.
O URL do documento que está sendo exibido está disponível para
JavaScript do lado do cliente como document.url.Se você passar isso
string para o construtor url () (§11.9), você pode acessar facilmente
As seções do caminho, consulta e fragmento do URL.
O conteúdo do cabeçalho da solicitação de "cookie" http está disponível
para o código do lado do cliente como document.cookie.Cookies são
Geralmente 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) se
necessário.Consulte §15.12.2 para obter mais detalhes.
A propriedade Global Navigator fornece acesso a
Informações sobre o navegador da web, o sistema operacional está em execução
de e as capacidades de cada um.Por exemplo,
Navigator.UserAgent é uma string que identifica a web
Navegador, Navigator.Language é preferido pelo usuário

Erro ao traduzir esta página.

A função Window.onerror será invocada com três string
argumentos.O primeiro argumento para Window.onerror é uma mensagem
descrevendo o erro.O segundo argumento é uma string que contém o
URL 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
O OnError Handler retorna true, diz ao navegador que o manipulador
lidou com o erro e que nenhuma ação adicional é necessária - em outro
Palavras, o navegador não deve exibir sua própria mensagem de erro.
Quando uma promessa é rejeitada e não há função .Catch () para
lidar com isso, essa é uma situação como uma exceção não tratada: um
Erro imprevisto ou um erro lógico em seu programa.Você pode detectar
isso definindo uma janela.onunhandledReject Função ou
usando window.addeventListener () para registrar um manipulador para
Eventos de ?rejeição não entrega?.O objeto de evento passou para este manipulador
terá uma propriedade de promessa cujo valor é o objeto de promessa que
rejeitado e uma propriedade pela qual a propriedade cujo valor é o que teria sido
Passado para uma função .catch ().Como nos manipuladores de erros descritos
Antes, se você ligar para o PreventDefault () na rejeição não enfrentada
objeto de evento, ele será considerado tratado e não causará um erro
mensagem no console do desenvolvedor.
Não é frequentemente necessário definir o OnError ou
Os manipuladores de rejeição de OnUnHandled, mas podem ser bastante úteis como um
mecanismo de telemetria se você deseja relatar erros do lado do cliente ao
servidor (usando a função fetch () para fazer uma solicitação de postagem http,
por exemplo) para que você possa obter informações sobre erros inesperados
Isso acontece nos navegadores de seus usuários.

15.1.8 O modelo de segurança da web

O 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
trabalhei duro para equilibrar dois objetivos concorrentes:

Definindo APIs poderosas do lado do cliente para ativar a Web útil
Aplicações

Impedindo que o código malicioso leia ou altere seus dados,
comprometendo sua privacidade, enganando você ou desperdiçando seu
tempo

As subseções a seguir fornecem uma rápida visão geral da segurança
Restrições e questões que você, como um programador JavaScript, deveriam
estar ciente de.

O que JavaScript não pode fazer

A primeira linha de defesa dos navegadores da web contra código malicioso é que eles
Simplesmente não suporta certos recursos.Por exemplo, lado do cliente
O JavaScript não fornece nenhuma maneira de escrever ou excluir arquivos arbitrários ou
Liste os diretórios arbitrários no computador cliente.Isso significa a
O programa JavaScript não pode excluir dados ou plantar vírus.

Da mesma forma, o JavaScript do lado do cliente não tem uso geral
Recursos de rede.Um programa JavaScript do lado do cliente pode fazer
Solicitações HTTP (§15.11.1).E outro padrão, conhecido como
WebSockets (§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 e
Os servidores não podem ser gravados no JavaScript do lado do cliente.

A política da mesma origem
A política da mesma origem é uma restrição de segurança abrangente sobre qual web
O código JavaScript de conteúdo pode interagir.Normalmente entra em jogo
Quando uma página da web inclui elementos <frame>.Nesse caso, o
A política da mesma origem governa as interações do código JavaScript em um
enquadrar com o conteúdo de outros quadros.Especificamente, um script pode ler
Somente as propriedades de janelas e documentos que têm o mesmo
origem como o documento que contém o script.
A origem de um documento é definida como o protocolo, o host e o porto de
o URL a partir do qual o documento foi carregado.Documentos carregados
De diferentes servidores da Web têm origens diferentes.Documentos carregados
Através de diferentes portas do mesmo host, têm origens diferentes.E a
Documento carregado com o http: o protocolo tem uma origem diferente de
um carregado com o https: protocolo, mesmo que eles vierem do
mesmo servidor da web.Os navegadores normalmente tratam todos os arquivos: URL como um
origem separada, o que significa que se você estiver trabalhando em um programa que
Exibe mais de um documento do mesmo servidor, você não pode
ser capaz de testá -lo localmente usando o arquivo: URLs e terá que executar um
Servidor da Web estático durante o desenvolvimento.
É importante entender que a origem do próprio script não é
relevante para a política da mesma origem: o que importa é a origem do
documento no qual o script está incorporado.Suponha, por exemplo, que
Um 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 de
Esse script é o host B e o script tem acesso total ao conteúdo do
documento que o contém.Se o documento contiver um <frame> que

contém um segundo documento do Host B, e o script também tem completo
acesso ao conteúdo desse segundo documento.Mas se o nível superior
O documento contém outro <frame> que exibe um documento de
Host C (ou mesmo um do Host A), então a política da mesma origem vem
Em 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, mas
não permite que os scripts se comuniquem com outros servidores da web (a menos
Esses servidores da Web optam por CORS, como descrevemos a seguir).
A política da mesma origem apresenta problemas para grandes sites que usam
Vários subdomínios.Por exemplo, scripts com origem
orders.example.com pode precisar ler propriedades de documentos em
exemplo.com.Para suportar sites multidomânticos desse tipo, os scripts podem
Altere sua origem definindo document.Domain para um sufixo de domínio.
Então, um script com origem https://orders.example.com pode alterar sua origem
para 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 cruzada
Compartilhamento de Recursos de Origin, ou CORS, que permite aos servidores decidir
quais origens eles estão dispostos a servir.CORS estende http com um
Nova origem: Cabeçalho de solicitação e um novo controle de acesso
Cabeçalho de resposta de origem da autorização.Ele permite que os servidores usem um cabeçalho para
Liste 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 cors
cabeçalhos e não relaxam restrições da mesma origem, a menos que sejam
presente.
Script de câmara cruzada
Scripts 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 defender
Contra, scripts de sites cruzados.
Uma página da web é vulnerável a scripts cruzados se ela dinamicamente
gera conteúdo de documentos e bases esse conteúdo em substituição de usuário
dados sem primeiro "higienizar" esses dados removendo qualquer
Tags html a partir dele.Como exemplo trivial, considere a seguinte web
página 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 html
String na primeira tag <H1> no documento.Esta página pretende
ser 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/%3e
Quando 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 OnLoad
é executado.A função Global Alert () exibe um diálogo modal
caixa.Uma única caixa de diálogo é relativamente benigna, mas demonstra que
A execução do código arbitrária é possível neste site porque exibe
HTML não senitado.
Os ataques de script de sites são assim chamados porque mais de um site é
envolvido.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 no
Link, eles serão levados para o site A, mas esse site agora estará executando o código
No site B. Esse código pode definir a página ou causar
defeituoso.Mais perigosamente, o código malicioso poderia ler cookies
armazenado pelo site A (talvez números de conta ou outro pessoalmente
identificação de informações) e enviar esses dados de volta ao site B. o injetado
O código pode até rastrear as teclas do usuário e enviar esses dados de volta para
Local B.
Em geral, a maneira de impedir ataques XSS é remover tags HTML
De qualquer dados não confiáveis ??antes de usá -los para criar um documento dinâmico
contente.Você pode corrigir o arquivo greet.html mostrado anteriormente substituindo
caracteres HTML especiais na sequência de entrada não confiável com seus
Entidades HTML equivalentes:

nome = nome
.place (///g, "& amp;")
.Place (/</g, "& lt;")
.place (/>/g, "& gt;")
.Place (/"/g", "")
.place (/'/g, "'")
.place (/\ // g, "/")
Outra abordagem do problema do XSS é estruturar sua web
aplicativos para que o conteúdo não confiável seja sempre exibido em um
<frame> com o atributo Sandbox definido para desativar scripts e
Outros recursos.
O script entre sites é uma vulnerabilidade perniciosa cujas raízes vão profundamente
na arquitetura da web.Vale a pena entender isso
vulnerabilidade em profundidade, mas uma discussão mais aprofundada está além do escopo de
este livro.Existem muitos recursos on -line para ajudá -lo a se defender
Script de câmara cruzada.
15.2 Eventos
Os programas JavaScript do lado do cliente usam um evento assíncrono orientado
modelo de programação.Nesse estilo de programação, o navegador da web
gera um evento sempre que algo interessante acontece com o
documentar ou navegador ou para algum elemento ou objeto associado a ele.
Por exemplo, o navegador da web gera um evento quando terminar
Carregando um documento, quando o usuário move o mouse sobre um hiperlink,
ou quando o usuário atinge uma chave no teclado.Se um JavaScript
o aplicativo se importa com um tipo específico de evento, pode registrar um ou
Mais funções a serem invocadas quando ocorrem eventos desse tipo.Observe que
Isso não é exclusivo da programação da web: todos os aplicativos com gráfico

As interfaces de usuário são projetadas dessa maneira - elas ficam esperando para ser
interagiram com (ou seja, eles esperam os eventos ocorrerem) e depois eles
responder.
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.Nós
Comece esta seção com algumas definições importantes que ajudam a explicar
Esse modelo de evento:
Tipo de evento
Esta string especifica que 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
um 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, para
Exemplo, ou um evento de clique em um elemento <butter>.Janela,
Os 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
O 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.
objeto de evento
Este objeto está associado a um evento específico e contém detalhes
sobre esse evento.Os objetos de evento são passados ??como um argumento para o
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 o
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 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
pare 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 o
oportunidade de interceptar (ou "capturar") eventos antes de serem
entregue ao seu alvo real.Eventos borbulhando e captura são
coberto em detalhes em §15.2.4.
Alguns eventos têm ações padrão associadas a eles.Quando um clique
Evento 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
Evite 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 há como este 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", "Toukend", "Keydown" e "KeyUp".
Eventos de entrada independentes do dispositivo
Esses eventos de entrada não estão diretamente ligados a um dispositivo de entrada específico.
O evento "clique", por exemplo, indica que um link ou botão (ou
outro elemento de documento) foi ativado.Isso geralmente é feito via
um clique do mouse, mas também pode ser feito pelo teclado ou (no toque-
dispositivos sensíveis) com uma torneira.O evento de "entrada" é um dispositivo
Alternativa independente ao evento "KeyDown" e suporta
entrada do teclado, bem como alternativas como corte e colar
Métodos de entrada usados ??para scripts ideográficos.O "ponteiro down".
Os tipos de eventos "Pointermove" e "Pointerup" são independentes de dispositivos
alternativas para o mouse e tocar eventos.Eles trabalham para o tipo de mouse
Ponteiros, para telas de toque, e para entrada no estilo de caneta ou caneta
bem.
Eventos de interface do usuário
Eventos de interface do usuário são eventos de nível superior, geralmente em elementos de forma HTML
Isso define uma interface do usuário para um aplicativo da Web.Eles incluem o
Evento "Focus" (quando um campo de entrada de texto ganha foco no teclado), o
Evento de "mudança" (quando o usuário altera o valor exibido por um
elemento do formulário) e o evento "enviar" (quando o usuário clica em um
Enviar botão em um formulário).
Eventos de mudança de estado
Alguns eventos não são acionados diretamente pela atividade do usuário, mas por
atividade de rede ou navegador e indica algum tipo de ciclo de vida ou
mudança relacionada ao estado.Os eventos "Carregar" e "DomContentLoaded"
?Filado na janela e no documento objetos, respectivamente, no
Fim do carregamento de documentos - provavelmente é o mais comumente usado
Desses eventos (consulte ?Linha do tempo do JavaScript do lado do cliente?).Navegadores
Fire 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
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 com
proibido no modo rigoroso (§5.6.3), mas o código JavaScript em HTML
atributos nunca são rigorosos.Os manipuladores de eventos definidos dessa maneira são
executado 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
invocou 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
<butto> elemento.Observe as diferenças entre as duas técnicas
usado:

<botão id = "myButton"> clique em mim </button>
<Cript>
Seja b = document.QuerySelector ("#myButton");
B.OnClick = function () {Console.log ("Obrigado por clicar
meu!"); };
B.AddeventListener ("Click", () => {Console.log ("Obrigado
de novo!"); });
</script>
Chamando addEventListener () com "clique" como seu primeiro argumento
não afeta o valor da propriedade OnClick.Neste código, um
O clique do botão registrará duas mensagens no console do desenvolvedor.E se nós
chamado addEventListener () primeiro e depois definido oClick, nós
ainda registraria duas mensagens, exatamente na ordem oposta.Mais
É importante ressaltar que você pode ligar para addEventListener () várias vezes para
Registre mais de uma função de manipulador para o mesmo tipo de evento no
mesmo objeto.Quando um evento ocorre em um objeto, todos os manipuladores
Registrado para esse tipo de evento é invocado na ordem em que eles
foram registrados.Invocando addEventListener () mais de uma vez
no mesmo objeto com os mesmos argumentos não tem efeito - o manipulador
A função permanece registrada apenas uma vez e a invocação repetida
não altera a ordem em que os manipuladores são invocados.
addEventListener () é emparelhado com um
RemoneeventListener () Método que espera os mesmos dois
Argumentos (além de um terceiro opcional), mas remove um manipulador de eventos
função de um objeto em vez de adicioná -lo.Muitas vezes é útil
registre temporariamente um manipulador de eventos e remova -o em breve
depois.Por exemplo, quando você recebe um evento "Mousedown", você
pode registrar manipuladores de eventos temporários para ?mousemove? e
Eventos "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 isso
A 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 prevenirDefault () para cancelar
A ação padrão (consulte §15.2.5).Isso é particularmente importante para o toque
Eventos 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.Suave
A 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 explicitamente
Defina a propriedade passiva como falsa.
Você também pode passar em um objeto de opções para removerventListener (),
Mas a propriedade de captura é a única que é relevante.Não há
precisa especificar uma vez ou passivo ao remover um ouvinte e
Essas propriedades são ignoradas.

15.2.3 Invocação do manipulador de eventos

Depois 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
Especifique 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
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
Tenha o mesmo esse 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
O valor é tipicamente um sinal para o navegador de que não deve executar o
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
O 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 para 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 o
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 UI
As 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
objetar 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 a
O segundo argumento:

```
// Despacha um evento personalizado para que a interface do usuário saiba que estamos ocupados
document.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 do 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 do documento global refere -se
para o objeto do documento, e o objeto de documento tem cabeça e corpo
propriedades que se referem aos objetos do elemento para o <head> e
<body> tags, respectivamente.Mas um programa que deseja manipular um
O elemento incorporado mais profundamente no documento deve de alguma forma obter
ou selecione 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
descrevendo elementos ou conjuntos de elementos em um documento.O dom
Métodos querySelector () e querySelectorAll () permitem
nós para encontrar o elemento ou elementos em um documento que corresponda
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 id
atributo, ou as palavras em seu atributo de classe:
div // qualquer elemento
#nav // O elemento com id = "Nav"
.warning // qualquer elemento com "aviso" em seu
atributo de classe
O 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: <p
lang = "fr">
*[name = "x"] // qualquer elemento com um nome = "x"
atributo
Observe que esses exemplos combinam um seletor de nome de tag (ou a * tag
Nome Wildcard) com um seletor de atributos.Combinações mais complexas
também são possíveis:
span.fatal.error // qualquer com "fatal" e
"Erro" em sua classe
span [lang = "fr"]. Aviso // qualquer em francês com aula
"aviso"
Os seletores também podem especificar a estrutura do documento:
#log span // qualquer descendente do
elemento com id = "log"
#log> span // qualquer filho do elemento
com 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> e
é um irmão disso

Se dois seletores forem separados por vírgula, significa que selecionamos
Elementos que correspondem a um dos seletores:
botão, entrada [type = "button"] // all <butto> e <entrada
type = "Button"> elementos
Como você pode ver, os seletores de CSS nos permitem referir a elementos dentro de um
documento por tipo, id, classe, atributos e posição dentro do
documento.O método querySelector () leva um seletor CSS
string como seu argumento e retorna o primeiro elemento correspondente no
Documente que ele encontra ou retorna nulo se nenhum corresponde:
// Encontre o elemento de documento para a tag html com atributo
id = "Spinner"
deixe spinner = document.QuerySelector ("#spinner");
querySelectorAll () é semelhante, mas retorna todas as correspondências
Elementos no documento, em vez de apenas retornar o primeiro:
// Encontre todos os objetos de elemento para <H1>, <H2> e <H3> tags
Let Titles = Document.QuerySelectorAll ("H1, H2, H3");
O valor de retorno de querySelectorAll () não é uma variedade de
Objetos de elemento.Em vez disso, é um objeto semelhante a uma matriz conhecido como um
Nodelist.Os objetos nodelistas têm uma propriedade de comprimento e pode ser
indexados como matrizes, para que você possa percorrer -las com um tradicional para
laço.Os nodelistas também são iteráveis, para que você possa usá -los com/de
loops 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
Os 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
O 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
Registrou um manipulador de eventos em um nível alto na árvore de documentos.Se você
estão 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 do
A própria tag do Hyperlink.Seu manipulador de eventos pode procurar o
mais próximo contendo hiperlink assim:

```
// Encontre a etiqueta de gabinete mais próxima que tem um href
atributo.
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 HTML
elemento
função insidelist (e) {
retornar e.closest ("ul, ol, dl")! == null;
}
```

O método relacionado corresponde () não retorna ancestrais ou
Descendentes: simplesmente testa se um elemento é correspondido por um CSS
seletor e retorna true se for e false, caso contrário:

```
// retorna true se E for um elemento de cabeçalho HTML
função isheading (e) {
retornar e.matches ("H1, H2, H3, H4, H5, H6");
}
```

Outros métodos de seleção de elementos

Além de queryselector () e querySelectorAll (),
O DOM também define vários métodos de seleção de elementos mais antigos
que são mais ou menos obsoletos agora.Você ainda pode ver alguns deles
Métodos (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.Querysselectorall ("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, o
Os nodelistas retornados por esses métodos de seleção mais antigos são "vivos", que
significa que o comprimento e o conteúdo da lista podem mudar se o documento
Alterações de conteúdo ou estrutura.
Elementos pré -selecionados
Por razões históricas, a classe de documentos define propriedades de atalho
Para acessar certos tipos de nós.As imagens, formas e links
Propriedades, por exemplo, fornecem fácil acesso ao <mg>, <morm>,
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
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
precisam encontrar porções estruturalmente relacionadas (pai, irmãos, filhos) de
o 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
métodos;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 nodelas contém o elemento 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 apenas
Crianças de elemento) do nó.
FirstChild, LastChild
O primeiro e o último filho dos nós de um nó, ou nulo se o nó não tiver
crianças.
Nextsibling, anteriorsbling
Os próximos e anteriores nós de irmãos de um nó.Essas propriedades
Conecte nós em uma lista duplamente vinculada.
NodeType
Um número que especifica que tipo de nó é esse.Nós do documento
ter valor 9. Os nós do elemento têm valor 1. Os nós de texto têm valor
3. Os nós de comentários têm valor 8.
NodEvalue
O conteúdo textual de um nó de texto ou comentário.
Nodename
O nome da tag html de um elemento, convertido em maiúsculas.
Usando essas propriedades de nó, o segundo nó filho do primeiro filho de
O documento pode ser referido com expressões como estas:
Document.ChildNodes [0] .ChildNodes [1]
document.firstchild.firstchild.nextsibling
Suponha 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 um
NodeType 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
<fody> 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 TextContent
Função 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
nó
s += Child.NodEvalue;// Adicione o texto
conteúdo para nossa string.
} else if (type === 1) {// e se for um
Nó do elemento
s += 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 de
a imagem
image.id === "main_image" // => true;Nós procuramos a imagem
por id
Da 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 documento
f.action = "https://www.example.com/submit";// defina o URL
para enviá -lo para.
f.method = "post";// defina o http
Tipo de solicitação.
Para alguns elementos, como o elemento <input>, alguns html
Nomes de atributo Mapa para propriedades de nome diferente.O html
o atributo de valor de um <input>, por exemplo, é refletido pelo
Propriedade JavaScript DefaultValue.A propriedade JavaScript Value
do elemento <input> contém a entrada atual do usuário, mas muda
para a propriedade Value não afeta a propriedade DefaultValue nem
o atributo de valor.
Os atributos HTML não são sensíveis ao maiúsculas, mas nomes de propriedades JavaScript
são.Para converter um nome de atributo para a propriedade JavaScript, escreva -o em
minúsculo.Se o atributo for mais de uma palavra de comprimento, no entanto, coloque o
Primeira letra de cada palavra após a primeira em maiúsculas:
DefaultChecked e Tabindex, por exemplo.Manipulador de eventos
Propriedades como OnClick são uma exceção, no entanto, e são escritas em
minúsculo.

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 mantaria o valor de
o atributo Data-X.Mapas de atributos hifenizados para camelcase
Nomes de propriedades: o número de seção de dados do atributo se torna o
Propriedade 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 a string html ?Isso é um simples
documento".
O conteúdo é a sequência de texto simples ?Este é um simples
documento".
Ambas são respostas válidas, e cada resposta é útil em seus próprios
caminho.As seções a seguir explicam como trabalhar com o HTML
Representação e a representação de texto simples do conteúdo de um elemento.

Conteúdo do elemento como html
Ler 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 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 seus
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" a
o 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
ilustrado 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
Faça isso é com a propriedade TextContent:
Seja para = document.QuerySelector ("p");// primeiro <p> no
documento
deixe texto = para.textContent;// Obtenha o texto de
o parágrafo
para.TextContent = "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.

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
nós.A classe de documento define métodos para criar elemento
objetos 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
Documentar a classe e anexar seqüências de texto ou outros elementos a ele com
Seus métodos Append () e Prendend ():
Seja parágrafo = document.createElement ("p");// Crie um
Elemento vazio <p>

deixe ênfase = document.createElement ("em");// Crie um
Elemento vazio
ênfase.append ("mundo");// Adicionar texto a
o elemento
parágrafo.Append ("Hello", ênfase "!");// Adicionar texto e
 para <p>
parágrafo.Prepend ("¡");// Adicione mais texto
no início de <p>
paragraph.innerhtml // => "¡Hello
 mundo ! "
append () e precend () tome qualquer número de argumentos que
pode ser objetos ou strings do nó.Argumentos de string são automaticamente
convertido em nós de texto.(Você pode criar nós de texto explicitamente com
document.createTextNode (), mas raramente há motivo para
faça isso.) Append () adiciona os argumentos ao elemento no final do
Lista de crianças.Agenda () adiciona os argumentos no início da lista de crianças.
Se você deseja inserir um elemento ou nó de texto no meio do
contendo a lista infantil do elemento, depois nem appeê () ou
Apresença () funcionará para você.Nesse caso, você deve obter um
referência a um nó de irmão e ligue antes () para inserir o novo
conteúdo 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 depois
aquele cabeçalho
Saudações.
Como append () e precend (), depois () e antes () pegar
qualquer 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
necessário.

15.3.6 Exemplo: gerando um índice

Exemplo 15-1 mostra como criar dinamicamente um índice para
um documento.Demonstra muitos dos scripts de documentos
Técnicas descritas nas seções anteriores.O exemplo está bem
comentou 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 demitido
e
* gera automaticamente um índice para o
documento.
* Não define nenhum símbolo global, então não deve
conflito
* 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
no
* Início do documento.Em seguida, a função encontra tudo <H2>
através
* <H6> tags, os trata como títulos de seção e cria um
Tabela de
* Conteúdo dentro do elemento TOC.A função adiciona seção
números
* para cada seção cabeçalho e envolver os títulos em nomeado
âncoras
```

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

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
o 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 embutidos
Para continuar com o exemplo da dica de ferramenta anterior, suponha que o
O documento está estruturado com apenas um único elemento da dica de ferramenta, e queremos
Para posicioná -lo dinamicamente antes de exibi -lo.Em geral, não podemos
criar 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 ferramenta
Elemento para definir estilos embutidos específicos para esse elemento.O
Dom 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, o
Propriedade de estilo não é uma string.Em vez
Objeto: uma representação analisada dos estilos CSS que aparecem em textual
forma no atributo de estilo.Para exibir e definir a posição de nosso
Distima 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 JavaScript
Muitas 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 CSSSSTYLEDECLARATION são ligeiramente diferentes de
os nomes das propriedades reais do CSS.Se um nome de propriedade CSS contiver um ou mais hífens, o
O nome da propriedade cssstyleleDeclaration 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 cssstyledeclaration
Objeto, 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 para
A 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 estilo
usará quaisquer unidades e qualquer formato de propriedade de atalho
realmente usado no atributo html, e seu código pode ter que fazer
Alguns sofisticados analisando para interpretá -los.Em geral, se você quiser
Consulte os estilos de um elemento, você provavelmente deseja o estilo calculado,
que é discutido a seguir.

15.4.3 Estilos computados

O estilo calculado para um elemento é o conjunto de valores de propriedade que o
O navegador deriva (ou calcula) do estilo embutido do elemento, além de todos
Regras de estilo aplicáveis ??em todas as folhas de estilo: é o conjunto de propriedades
realmente usado para exibir o elemento.Como estilos embutidos, estilos computados
são representados com um objeto CSSStyleleDeclaration.Ao contrário de embutido
Os estilos, no entanto, os estilos computados são somente leitura.Você não pode definir isso
Estilos, mas o objeto CSSSTyleDECLARATION CULUTADO para um elemento
Permite determinar quais propriedades de estilo valores o navegador usou quando
renderizando esse elemento.
Obter o estilo calculado para um elemento com o

getComputedStyle () Método do objeto de janela.O primeiro
Argumento para este método é o elemento cujo estilo calculado é
desejado.O segundo argumento opcional é usado para especificar um CSS
pseudoelemento, 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 um
Objeto cssstyleleDeclaration que representa todos os estilos que se aplicam a
o elemento especificado (ou pseudoelemento).Existem vários
diferenças importantes entre um objeto CSSStyleleDeclaration que
representa 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 como
Porcentagens e pontos são convertidos em valores absolutos.Qualquer
propriedade que especifica um tamanho (como um tamanho de margem ou uma fonte
tamanho) terá um valor medido em pixels.Este valor será um
string com um sufixo "px", então você ainda precisará analisá -lo, mas
você não precisará se preocupar em analisar ou converter outros
unidades.Propriedades cujos valores são cores serão devolvidos em
Formato "rgb ()" ou "rgba ()".
As propriedades de atalho não são calculadas - apenas o fundamental
Propriedades em que elas são baseadas são.Não consulte a margem
propriedade, por exemplo, mas use marginleft, margintop,
e assim por diante.Da mesma forma, não consulte a fronteira ou mesmo
largura de fronteira.Em vez
BordertopWidth, e assim por diante.

A propriedade CSSTEXT do estilo computado é indefinida.
Um objeto CSSStyleleDeclaration retornado pelo getComputedStyle ()
geralmente contém muito mais informações sobre um elemento do que o
CssstyleleDeclaration obtido da propriedade em estilo em linha
elemento.Mas os estilos computados podem ser complicados, e consultá -los
Nem sempre fornece as informações que você pode esperar.Considere o
Atributo da família de fontes: aceita uma lista separada por vírgula
Famílias de fontes para portabilidade entre plataformas.Quando você consulta o
Propriedade da Fontfamília de um estilo computado, você está simplesmente recebendo o
valor do estilo mais específico da família de fontes que se aplica ao
elemento.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, se
Um elemento não está absolutamente posicionado, tentando consultar sua posição
e tamanho através das propriedades superior e esquerda de seu estilo calculado
frequentemente retorna o valor automático.Este é um valor de CSS perfeitamente legal, mas
Provavelmente não é o que você estava procurando.
Embora CSS possa ser usado para especificar com precisão a posição e o tamanho de
elementos de documentos, consultar o estilo calculado de um elemento não é
A 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 script
Além de atributos de classe de script e estilos embutidos, o JavaScript pode
também manipular as folhas de estilo.As folhas de estilo estão associadas a
Um documento HTML com uma tag <yoy> ou com um <link
rel = "STILELES SHEET"> TAG.Ambos são tags HTML regulares, então

você pode dar a eles os dois atributos de identificação e depois procurá -los com
Document.QuerySelector ().
Os objetos do elemento para tags <yoy> e <link> têm um
Propriedade 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"
temas
função TogGleTheme () {
Deixe LightTheme = Document.QuerySelector ("#Light-Them");
Let DarkTheme = Document.QuerySelector ("#Dark-Them");
if (DarkTheme.Disabled) {// atualmente leve,
Mude para o escuro
LightTheme.disabled = true;
DarkTheme.Disabled = false;
} else {// atualmente escuro,
Mude para a luz
LightTheme.disabled = false;
DarkTheme.Disabled = true;
}
}
```

Outra maneira simples de folhas de estilo de script é inserir novas no
Documente 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 nomeada
Let 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 tema
folha 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 erend",
"<yoy> corpo {transform: girate (180deg)} </style>"
);
```

Os navegadores definem uma API que permite que o JavaScript olhe para dentro folhas de estilo para consultar, modificar, inserir e excluir regras de estilo nisso folha 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 CSS

Suponha que você tenha as duas classes CSS a seguir definidas em um Folha 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 o
propriedade de nome de animação que define a animação e um
Propriedade 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, web
As aplicações podem tratar documentos como árvores de elementos e nunca precisam
Pense 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 de
Esse elemento.

As seguintes subseções explicam como você pode ir e voltar
entre o modelo abstrato e baseado em árvores de um documento e o
visão geométrica, baseada em coordenadas do documento, como é apresentada em um
Janela do navegador.

15.5.1 Documentar coordenadas e viewport

Coordenadas

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 são
Sempre retangular quando estabelecido pelo navegador.Elementos embutidos, tal
como , <code> e <b> elementos, no entanto, podem abranger múltiplos
linhas e, portanto, consistem em múltiplos retângulos.Imagine, para
exemplo, algum texto dentro das tags e que acontecem
exibido para que ele envolva duas linhas.Seus retângulos consistem no
Fim da primeira linha e início da segunda linha.Se você ligar
getBoundingClientRect () neste elemento, o limite
O 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
Método com as coordenadas X e Y de um ponto (usando a Viewport
Coordenadas, não coordenadas de documentos: o clientx e a cliente
coordenadas de um trabalho de evento do mouse, por exemplo).
ElementFromPoint () retorna um objeto de elemento que está no
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 Rolagem

O método scrollTo () do objeto da janela leva o X e Y
coordenadas de um ponto (em coordenadas de documentos) e as definem como o
Offsets de barra de rolagem.Isto é, ele rola a janela para que o especificado
O ponto está no canto superior esquerdo da viewport.Se você especificar um ponto
isso é muito próximo do fundo ou muito próximo da borda direita do
documento, o navegador o moverá o mais próximo possível do superior
Esquerca esquerda, mas não conseguirá chegar até lá.A seguir
O código rola o navegador para que a página mais inferior do documento
é 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 viewport
Window.ScrollTo (0, DocumentHeight - ViewPorthEight);
```

O método scrollby () da janela é semelhante a
scrollto (), mas seus argumentos são relativos e são adicionados ao
Posição atual de rolagem:

```
// role 50 pixels para baixo a cada 500 ms.Observe que não há como
Para 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 desejado
largura da viewport para a página.) O tamanho total do documento é o
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
Documente, ou você pode usar a largura de deslocamento e o peso
Propriedades do document.documentElement.Os compensações de rolagem de
O documento em sua viewport está disponível como janela.scrollx
e Window.Scrolly.Essas são propriedades somente de leitura, então você não pode
Defina -os para rolar o documento: use window.scrollto ().
As coisas são um pouco mais complicadas para elementos.Cada elemento
Objeto define os três grupos a seguir de propriedades:
OffsetWidth ClientWidth Scrollwidth
OffSetHeight ClientHeight ScrolHeight
OffsetLeft ClientLeft ScrollLeft
OFFSETTOP CLIENTTOP SCROLLTOP
OffsetParent
As propriedades de largura de offset e ofsetset de um elemento
Retorne seu tamanho na tela nos pixels CSS.Os tamanhos retornados incluem o
Border 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
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 <pan>, todos retornam 0.
scrollwidth e scrollheight retornam o tamanho de um elemento
Área de conteúdo mais seu preenchimento mais qualquer conteúdo transbordante.Quando o
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 lá
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
Defina -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
suportado.)

15.6 Componentes da Web

HTML é um idioma para marcação de documentos e define um rico conjunto de
Tags para esse fim.Nas últimas três décadas, tornou -se um
linguagem usada para descrever as interfaces do usuário da web
Aplicativos, mas tags básicas HTML, como <input> e <butto>
são inadequados para designs modernos da interface do usuário.Desenvolvedores da Web são capazes de
fazer funcionar, mas apenas usando CSS e JavaScript para aumentar o
Aparência e comportamento das tags HTML básicas.Considere um usuário típico
Componente de interface, como a caixa de pesquisa mostrada na Figura 15-3.

Figura 15-3.Um componente de interface do usuário da caixa de pesquisa

O elemento html <input> pode ser usado para aceitar uma única linha de
entrada do usuário, mas não tem como exibir ícones como
A lupa à esquerda e o cancelamento X à direita.Em ordem
Para implementar um elemento moderno de interface do usuário como este para a web, nós
precisa usar pelo menos quatro elementos html: um elemento <input> para
Aceite e exiba a entrada do usuário, dois elementos <mg> (ou neste
caso, dois elementos <Span> exibindo glifos unicode) e um
Container 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 usar
JavaScript para fazer com que todos os elementos HTML funcionem juntos.Quando o
O usuário clica no ícone X, precisamos de um manipulador de eventos para limpar a entrada

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 apóie 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
Estenda HTML com novas tags que funcionam como interface de usuário reutilizável e independente
componentes.
As subseções a seguir explicam como usar componentes da web
definido por outros desenvolvedores em suas próprias páginas da web e explique cada
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 web
Componente 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 = "módulo" src = "componentes/search-box.js">
```

Os componentes da web definem seus próprios nomes de tags HTML, com o
Restrição importante de que esses nomes de tags devem incluir um hífen.(Esse

significa que as versões futuras do HTML podem introduzir novas tags sem
hífens, e não há chance de que as tags entrem em conflito
componente da web de qualquer pessoa.) Para usar um componente da web, basta usar sua tag em
Seu arquivo html:
<Search-Box Paceholder = "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 dizer
quais atributos são suportados.Componentes da Web não podem ser definidos
com tags de fechamento automático.Você não pode escrever <Search-box/>, para
exemplo.Seu arquivo html deve incluir a tag de abertura e o
Tag de fechamento.
Como elementos HTML regulares, alguns componentes da web são escritos para
Espere que crianças e outras sejam escritas de tal maneira que elas não
Espere (e não exibirá) crianças.Alguns componentes da web são
escrito para que eles possam aceitar opcionalmente crianças rotuladas especialmente que
aparecerá 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 especificar
quais crianças devem ir para onde.Os nomes de slot - "esquerda" e "direita" em
Este exemplo - é definido pelo componente da web.Se o componente

Você está usando slots de suporte, esse fato deve ser incluído em seu
documentação.
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 diferida.Então isso significa que um
O 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 web
Os 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
exibido 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 acima
Innerhtml.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 um
Nome da tag html para que essas tags no documento sejam
Transformado automaticamente em instâncias da classe na árvore Dom.O
O 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 um
hífen) e uma subclasse de htmlelement como seu segundo argumento.Qualquer
Os elementos existentes no documento com esse nome de tag são "atualizados" para
Instâncias recém -criadas da classe.E se o navegador analisar qualquer
HTML No futuro, ele criará automaticamente uma instância da classe
para cada uma das tags encontra.
A classe passada para alparafements.define () deve estender
Htmlelement e não um tipo mais específico como
HtmlbuttonElement.Lembre -se do capítulo 9 que quando um javascript
Classe estende outra classe, a função do construtor deve chamar
super () antes de usar essa palavra -chave, por isso se o elemento personalizado
A classe tem um construtor, deve chamar super () (sem argumentos)
antes de fazer qualquer outra coisa.

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 embutido
Podemos 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 InLineCircle
estende Htmlelement {
// O navegador chama esse método quando um <circle inline>
elemento
// é inserido no documento.Há também um
desconectedCallback ()
// que não precisamos neste exemplo.
conectadoCallback () {
// Defina os estilos necessários para criar círculos
this.style.display = "inline-block";
this.style.borderradius = "50%";
this.style.border = "Solid Black 1px";
this.style.Transform = "Tradlatey (10%)";

```
// 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 observadattributes Property especifica qual
atributos
// queremos ser notificados sobre as alterações para.(Usamos um
getter aqui desde então
// Só podemos usar "estático" com métodos.)
estático ser observadottributes () {return ["diâmetro",
"cor"];}
// Este retorno de chamada é invocado quando um dos atributos
listado 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 o
Estilos de tamanho
this.style.width = newValue;
this.style.Height = newValue;
quebrar;
caso "cor":
// Se o atributo de cor mudar, atualize a cor
estilos
this.style.backgroundColor = newValue;
quebrar;
}
}
// define propriedades JavaScript que correspondem ao
elemento
// 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 atributeChangedCallback ()
```

Erro ao traduzir esta página.

matriz do elemento hospedeiro e não são visitados por DOM normal
Métodos de Traversal, como QuerySelector ().Por contraste, o
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 o html
Elementos <Audio> e <dife>: eles exibem um usuário não trivial
Interface para controlar a reprodução da mídia, mas a peça e a pausa
botões e outros elementos da interface do usuário não fazem parte da árvore dom e não podem
ser manipulado por JavaScript.Dado que os navegadores da web são projetados para
exibir html, é natural que os fornecedores do navegador gostariam
Exiba UIs internas como essas usando HTML.De fato, a maioria dos navegadores
tenho feito algo assim há muito tempo, e a sombra
Dom o torna uma parte padrão da plataforma da web.
Encapsulamento de sombra DOM
A principal característica do Shadow DOM é o encapsulamento que ele fornece.O
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
escondido de métodos DOM regulares como
QuerySelectorAll ().Quando uma raiz das sombras é criada
e anexado ao seu host das sombras, ele pode ser criado em "aberto" ou
Modo "fechado".Uma raiz de sombra fechada está completamente selada
longe e inacessível.Mais comumente, porém, raízes de sombra
são criados 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.Por
padrão, ele exibe um
* Emoji de copo de lupa (indicando pesquisa) à esquerda de
o campo de texto
* e um X emoji (indicando cancelamento) à direita do texto
campo.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
entrada
* 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
<img> crianças
* de <search-box> com slot = "esquerda" e slot = "direita"
atributos.
*
* <search-box> suporta o HTML normal desativado e oculto
atributos e
* também atributos de tamanho e espaço reservado, que têm o mesmo
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 detalhada
definido 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 e
os tipos de usuário
* Retornar ou guia).
*
* O elemento dispara um evento "claro" quando o usuário clica
o emoji certo
* (o x).Se nenhuma chamada de manipulador prevenirdefault () no evento
então o elemento
* Limpa a entrada do usuário quando a expedição de eventos estiver concluída.
*
* Observe que não há propriedades Onsearch e OnClear ou
Atributos:

```
* 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 ser
primeiro.
// Crie uma árvore de sombra Dom e anexe -a a isso
elemento, configuração
// o valor deste.shadowroot.
this.attachshadow ({mode: "aberto"});
// clonar o modelo que define os descendentes e
folha de estilo para
// este componente personalizado e anexar esse conteúdo a
a raiz da sombra.

this.shadowroot.append (searchbox.template.content.cloneNode (true
));
// recebe referências aos elementos importantes no
Shadow Dom
this.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 nosso
folha de estilo interna
// para exibir ou ocultar um anel de foco falso em todo
componente.Observação
// que os eventos "Blur" e "Focus" borbulham e aparecem
para 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 borbulha
a 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 prevenirdefault () 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.DefaultPreveded) {// se o evento não foi
"cancelado"
this.input.value = "";// então limpe a entrada
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 observadattributes
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 para
propriedades isso
// corresponde aos atributos HTML que apoiamos.O
getters simplesmente retornam
// o valor (ou a presença) do atributo.E o
Setters acabou de definir
// o valor (ou a presença) do atributo.Quando a
Método Setter
// Altera um atributo, o navegador irá automaticamente
invocar o
// atributeChangedCallback acima.
Obtenha espaço reservado () {retornar
this.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 o
Método AttributeChangedCallback.
// apenas atributos nomeados nesta matriz desencadearão chamadas para
Esse método.
SearchBox.ObservedAttributes = ["desativado", "espaço reservado",
"tamanho", "valor"];
// Crie um elemento <Sodemplate> para segurar a folha de estilo e o
árvore de
// elementos que usaremos para cada instância da caixa de pesquisa
elemento.
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 apenas
clonar 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> no
luz
* Dom.Esses estilos são padrões e podem ser substituídos pelo
usuário 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 host
escondeu ... */
Exibir: Nenhum;/* ... Conjunto de atributos não exibi -lo
*/
}
: host ([desativado]) { /* Quando o host tem o desativado
atributo ... */
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 precisa
Descrição independente (daí "escalável") das etapas necessárias para
Desenhe o gráfico desejado.As imagens SVG são descritas por arquivos de texto usando
A 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 <img>,
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 html
documentos.Se você fizer isso, o analisador HTML do navegador permite
você omita namespaces xml e tratar tags SVG como se elas
eram tags html.
3. Você pode usar a API DOM para criar dinamicamente SVG
elementos para gerar imagens sob demanda.
As subseções a seguir demonstram o segundo e o terceiro usos de
Svg.Observe, no entanto, que o SVG tem um grande e moderadamente complexo
gramática.Além das primitivas simples de desenho de formas, inclui
Suporte a curvas arbitrárias, texto e animação.Os gráficos SVG podem
até incorporar scripts JavaScript e folhas de estilo CSS para adicionar
Informações de comportamento e apresentação.Uma descrição completa do SVG é
Muito além do escopo deste livro.O objetivo desta seção é apenas para
Mostre como você pode usar o SVG em seus documentos e script HTML
com 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 para
Exiba uma face do relógio analógico:

```
<html>
<head>
<title> relógio analógico </title>
<estilo>
/* Esses estilos CSS se aplicam aos elementos SVG definidos
abaixo */
#clock { /* estilos para tudo
No relógio:*/
AVC: preto;/ * linhas pretas */
AVC-LINECAP: redondo;/ * Com extremidades arredondadas */
preenchimento: #ffe;/* em um esbranquiçado
fundo */
}
#clock .face {width: 3;} / * Contorno do relógio * /
#clock .Ticks {Width: 2;} /* Linhas que marcam cada
hora */
#clock .hands {stroke-width: 3;} /* Como desenhar o relógio
mãos */
#clock .numbers { /* como desenhar o
números */
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 tela
do gráfico ->
<!- ??O atributo ViewBox fornece a coordenada interna
sistema ->
<círculo class = "face" cx = "50" cy = "50" r = "45"/> <!- o
Face do relógio ->
```

```
<g class = "ticks"> <!- Marcas de ticks para cada um dos 12
horas ->
<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 normais
Tags html.<circ>, <line> e <sext> tags têm óbvio
Os propósitos, porém, e deve ficar claro como esse gráfico SVG funciona.
Existem muitas outras tags SVG, no entanto, e você precisará consultar
Uma referência SVG para saber mais.Você também pode notar que o
A folha de estilo é estranha.Estilos como preenchimento, largura de derrame e texto
A 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 Trabalhe 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 <mg>) 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 paira o ponteiro sobre ele, por exemplo - você pode alcançar Isso com CSS.

Também é possível manipular dinamicamente os gráficos SVG que são diretamente incorporado em html.O exemplo do relógio de rosto no anterior A seção exibe um relógio estático com mãos de hora e minuto voltadas para Exibindo o meio -dia ou meia -noite.Mas você pode ter Percebi que o arquivo HTML inclui uma tag <cript>.Esse script é executado uma função periodicamente para verificar o tempo e transformar a hora e mãos minuciosas girando -lhes o número apropriado de graus para que o relógio realmente exibe o horário atual, como mostrado na Figura 15-5.

Figura 15-5.Um relógio analógico SVG com script

O 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 eles
Gire -os ao redor do centro da face do relógio.A função usa
setTimeout () para garantir que ele funcione uma vez por minuto:

```
(função updateClock () {// Atualize o gráfico do relógio SVG para
Mostre a hora atual
deixe agora = new Date ();// Atual
tempo
deixe sec = agora.getSeconds ();// segundos
deixe min = agora.getminutes () + seg/60;// fracionário
minutos
deixe hora = (agora.gethours () % 12) + min/60;// fracionário
horas
Seja Minangle = min * 6;// 6 graus
por minuto
deixe hourrangle = hora * 30;// 30 graus
por 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 pouco

Matemática para converter os dados que estão sendo traçados em ângulos de picada.O volume

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 real
fatias de torta.O elemento usado para exibir cada fatia é <TACH>.Esse
O 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 letra
Códigos 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 o seguinte
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
cujo
* Os valores das propriedades são os valores associados a cada
rótulo
*
* A função retorna um elemento <Svg>.O chamador deve
insira -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 SVG
Seja svg = "http://www.w3.org/2000/svg";
// Crie o elemento <Svg> e especifique o tamanho do pixel e
coordenadas do usuário
Let Chart = Document.CreateElementns (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 nós
Deixe 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 para
nós 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 começo
em ângulos [i]
// e termina em ângulos [i+1].Os ângulos são medidos em
radianos.
deixe ângulos = [0];
valores.foreach ((x, i) => ângulos.push (ângulos [i] + x/total *
2 * math.pi));
// agora percorre as fatias da torta
valores.foreach ((valor, i) => {
// Calcule os dois pontos em que nossa fatia cruza
o círculo
// essas fórmulas são escolhidas para que um ângulo de 0 seja
Às 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 * math.cos (ângulos [i+1]);
// esta é uma bandeira para ângulos maiores que um meio círculo
// é necessário pelo componente de desenho de arco SVG
Deixe grande = (ângulos [i+1] - ângulos [i]> math.pi)?1: 0;
// Esta string descreve como desenhar uma fatia da torta
gráfico:
Deixe o caminho = `m $ {cx}, $ {cy}` + // mova -se para o círculo
centro.
`L $ {x1}, $ {y1}` + // Desenhe linha para
(x1, y1).
`A $ {r}, $ {r} 0 $ {big} 1` + // desenhar um arco de
raio 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.setAtattribute ("d", caminho);// Defina o
caminho para esta fatia
slice.setAtattribute ("preenchimento", cor);// Defina a fatia
cor
slice.setAtAttribute ("Stroke", "Black");// Contorno
Corte em preto
slice.setattribute ("largura de derrame", "1");// 1 pixel CSS
espesso
Chart.Append (Slice);// Adicione a fatia
para o gráfico
// agora desenha um pequeno quadrado correspondente para a chave
Deixe icon = document.createElementns (svg, "rect");
icon.setAtattribute ("x", lx);// posição
o 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 com
o método getContext () da tela.Ligue para getContext ()
com o argumento "2d" para obter um objeto de venernderingcontext2d
que você pode usar para desenhar gráficos bidimensionais na tela.
Como um exemplo simples da API de tela, o seguinte HTML
Usos do documento <lodvas> elementos e algum JavaScript para exibir
duas formas simples:

```
<p> Este é um quadrado vermelho: <canvas id = "quadrado" largura = 10
Altura = 10> </lvas>.
<p> Este é um círculo azul: <canvas id = "círculo" largura = 10
Altura = 10> </lvas>.
<Cript>
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 a
novo "caminho"
context.arc (5, 5, 5, 0, 2*math.pi, verdadeiro);// Adicione a
círculo 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 acariciado

Observe 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.O
o método preenchimento () preenche subpaths abertos agindo como se fosse uma linha reta
conectou 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 visual
Os 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 memória
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 (em
Pixels 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
ampliado 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
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 CSS
Atributos do estilo de largura e altura.Então, antes de começar a desenhar
Em seu código JavaScript, defina as propriedades de largura e altura do
A tela se opõe ao número de CSS Pixels Times
Window.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
Veja gráficos difusos ou pixelados, se o fizerem.Isso contrasta com o SVG
Gráficos, que permanecem nítidos, independentemente do tamanho ou zoom na tela
nível.)

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 preench () e por
Stroke () 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
detalhado abaixo.
Estilos de linha
A propriedade de largura de linha especifica o quão amplo (em pixels CSS) o
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 do
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 fechado Caminho, ligue para os métodos salvadores () e clip () primeiro e depois ligue Stroke () e Restore ().(O salvamento (), restaure () e Os métodos clip () são descritos posteriormente.)

Ao desenhar linhas que têm mais de cerca de dois pixels de largura, o As propriedades LineCap e Linejoin podem ter um impacto significativo em a aparência visual das extremidades de um caminho e os vértices nos quais Dois segmentos de caminho se encontram.A Figura 15-9 ilustra os valores e resultantes Aparência gráfica de LineCap e Linejoin.

Figura 15-9.Os atributos LineCap e Linejoin

O valor padrão para LineCap é "Butt".O valor padrão para

Linejoin é "Mitre".Observe, no entanto, que se duas linhas se encontrarem em um muito
ângulo 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 traço
padrão é definido e consultado com os métodos setLinedash () e
getLinedash () em vez de com uma propriedade.Para especificar uma corrida pontilhada
padrão, 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 CreaterRadialGradient ()
Métodos do contexto.Os argumentos para
createLinearGradient () são as coordenadas de dois pontos que
Defina uma linha (ela não precisa ser horizontal ou vertical) ao longo da qual
As cores variam.Os argumentos para o CreaterRadialGradient ()
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 Canvasgradiente que define as regiões do
tela que será preenchida, você deve definir as cores do gradiente ligando
O método addColorStop () do canvasgradiente.O primeiro
O argumento deste método é um número entre 0,0 e 1,0.O segundo
O argumento é uma especificação de cores CSS.Você deve chamar esse método em
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,
Eles 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 in
o meio
// Desbotando a cinza translúcido e depois voltou a transparente.
Seja DONUT = C.CreamRadialGradient (300.300.100, 300.300.300);
donut.addcolorstop (0,0, "transparente");//
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
Independente 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
CanvAspattern 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.

sombras.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
o padrão é totalmente transparente preto e as sombras nunca aparecerão, a menos que
Você define esta propriedade para uma cor translúcida ou opaca.Esta propriedade pode
apenas ser definido como uma corda de cor: padrões e gradientes não são permitidos
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 ShadofOffsetx e ShadofOffsety especificam o
X e Y compensações 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 do
tela do computador.Compensações maiores produzem sombras maiores e fazem
Os objetos desenhados parecem 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
girado e escalado.
A propriedade Shadowblur especifica o quão embaçado as bordas do
Shadow são.O valor padrão é 0, que produz nítido e não -ilícito
sombras.Valores maiores produzem mais borrão, até uma implementação-
limite superior definido.
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 alfa canal para cada cor, ou se você quiser adicionar translucidez ao opaco Imagens ou padrões, você pode definir a propriedade GlobalAlpha.Todo O pixel que você desenha terá seu valor alfa multiplicado pela GlobalAlpha. O padrão é 1, o que não adiciona transparência.Se você definir GlobalAlpha 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, eles Basta substituir os pixels que já estão lá.Se você está desenhando com Pixels translúcidos, o novo pixel ("" fonte ") é combinado com o antigo (?Destino?) Pixel para que o pixel antigo apareça através do novo pixel Com base em quão transparente é esse pixel.

Este processo de combinar novos pixels de origem (possivelmente translúcidos) Com os pixels de destino existentes (possivelmente translúcidos) são chamados composição, 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 o
A tela combinará pixels como se os novos pixels de origem fossem desenhados
sob os pixels de destino existentes.Se o destino for translúcido
ou transparente, parte ou toda a cor da fonte de pixels é visível no
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 especializada
usos 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
Ligue para getContext () retorna o mesmo canVAsRenderingContext2D
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
O método salva () empurra o estado gráfico atual para uma pilha de
Estados salvos.O método Restore () aparece a pilha e restaura o
Mais recentemente, estado salvo.Todas as propriedades que foram
descrito nesta seção fazem parte do estado salvo, assim como o atual
Região 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 tela
A API também inclui outros métodos de desenho.

Retângulos

CanvasRanderingContext2D 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 um
largura positiva e altura positiva, mas você também pode especificar outros
cantos e passam dimensões negativas.
FILLRECT () preenche o retângulo especificado com a corrente
Filtyle.strokerect () acaricia o contorno do especificado
retângulo usando os atributos atuais do Strokestyle e outros 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, a
o caminho.Como outros métodos de definição de caminho, ele não preenche ou golpe
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
conectado com segmentos de linha reta, mas que nem sempre precisam ser os
caso.O objeto CanvasRanderingContext2D define uma série de
métodos que adicionam um novo ponto ao subspato e conectam a corrente
aponte 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 atual
aponte para o início do arco com uma linha reta (que é útil
ao desenhar cunhas ou fatias de torta), então conecta o início de
o 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
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 incomum
O método de especificar arcos é realmente bastante útil para desenhar
formas com cantos arredondados.Se você especificar um raio de 0, este
O 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 círculo
Até que esteja indo na direção de P2.
beziercurveto ()
Este método adiciona um novo ponto P ao subspato e o conecta a
O ponto atual com uma curva de bezier cúbica.A forma da curva
é especificado por dois "pontos de controle", C1 e C2.No início do
curva (no ponto atual), a curva cabeças na direção de C1.
No final da curva (no ponto P), a curva chega do
direção de C2.Entre esses pontos, a direção da curva
varia suavemente.O ponto P se torna o novo ponto atual para o

Subpata.
quadraticcurveto ()
Este método é como BezierCurveto (), mas usa um quadrático
Curva 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.O
Os métodos demonstrados neste código são alguns dos mais complicados em
a API de tela;consulte uma referência on -line para obter detalhes completos sobre o
Métodos e seus argumentos.
Exemplo 15-6.Adicionando curvas a um caminho

```
// Uma função de utilidade para converter ângulos de graus em radianos
função 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 sem
direto
// linha do ponto atual para o início do círculo.
C.BeginPath ();
c.arc (75,100,50, // centro em (75.100), raio 50
0, 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 maneira
C.BeginPath ();// iniciar um novo caminho não conectado a
o círculo
C.Ellipse (200, 100, 50, 35, Rads (15), // Center, Radii e
rotação
0, rads (360), falso);// Iniciar ângulo, fim
ângulo, direção
// Desenhe uma cunha.Ângulos são medidos no sentido horário a partir do
eixo x positivo.
// Observe que o arc () adiciona uma linha do ponto atual ao
ARC START.
C.Moveto (325, 100);// Comece no centro do círculo.
c.arc (325, 100, 50, // Circle Center e Radius
rads (-60), rads (0), // comece no ângulo -60 e vá para o ângulo
0
verdadeiro);// no sentido anti -horário
C.ClosePath ();// Adicione o raio de volta ao centro de
o círculo
// cunha semelhante, compensar um pouco e na direção oposta
C.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 topo
borda.
C.Arcto (500.50.500.150,30);// Adicione parte da borda superior e superior
Canto direito.
C.Arcto (500.150.400.150,20);// Adicione a borda direita e inferior direito
canto.
```

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Na versão de três argumentos de drawimage (), o segundo e o terceiro
Argumentos especificam as coordenadas X e Y nas quais a parte superior esquerda
canto da imagem deve ser desenhado.Nesta versão do método, o
A imagem de origem inteira é copiada para a tela.As coordenadas X e Y são
interpretado no sistema de coordenadas atual e a imagem é escalada
e girado, se necessário, dependendo da transformação da tela atualmente
com efeito.
A versão de cinco argumentos de drawimage () adiciona largura e
Argumentos de altura para os argumentos X e Y descritos anteriormente.Esses
Quatro argumentos definem um retângulo de destino dentro da tela.O
canto superior esquerdo da imagem de origem vai em (x, y), e o inferior
O canto direito vai em (x+largura, y+altura).Novamente, o todo
A imagem de origem é copiada.Com esta versão do método, a fonte
A imagem será dimensionada para caber no retângulo de destino.
A versão de nove argumentos de drawimage () especifica uma fonte
retângulo e um retângulo de destino e copia apenas os pixels dentro
o retângulo de origem.Argumentos dois a cinco especificam a fonte
retângulo.Eles são medidos em pixels CSS.Se a imagem de origem for
Outra tela, o retângulo de origem usa o sistema de coordenadas padrão
para aquela tela e ignora qualquer transformação que tenha sido
especificado.Argumentos de seis a nove especificam o retângulo de destino
no qual a imagem é desenhada e está no sistema de coordenadas atual
da tela, não no sistema de coordenadas padrão.
Além de desenhar imagens para uma tela, também podemos extrair o
Conteúdo de uma tela como uma imagem usando o método Todataurl ().

Erro ao traduzir esta página.

O método setTransform () permite definir uma tela
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 seus
Efeito no sistema de coordenadas de tela.O programa que produziu o
A figura desenhou o mesmo conjunto de eixos sete vezes seguidos.A única coisa
Isso mudou cada vez foi a transformação atual.Observe que o
As transformações afetam o texto e as linhas desenhadas.

Erro ao traduzir esta página.

Figura 15-11.Coordenar transformações do sistema

O método TRANSTE () simplesmente move a origem da coordenada
sistema esquerdo, direita, para cima ou para baixo.O método girate () gira o
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
Mova 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 como nós
Suba a página.Um sistema de coordenadas invertidas como esse é familiar de
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 transforma geometricamente, pensando sobre
tradução (), girt () e escala () como transformando os eixos
do sistema de coordenadas, conforme ilustrado na Figura 15-11.É também
possível entender transforma algebricamente 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 com
o sistema de coordenadas padrão de uma tela, depois traduzi -lo e depois
escalar.Para mapear o ponto (x, y) na coordenada atual
sistema de volta ao 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
sistema e use as equações de tradução para mapear deste
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 do
Última (mais recente) transformação para a primeira.Quando pensar
Geometricamente sobre eixos transformados, no entanto, você trabalha adiante
Da primeira transformação para o último.
As transformações apoiadas pela tela são conhecidas como afins
transforma.As transformações afine podem modificar as distâncias entre
Pontos e os ângulos entre as linhas, mas as linhas paralelas sempre permanecem
paralelo após uma transformação afim - não é possível, por exemplo,
para especificar uma distorção da lente olho de peixe com uma transformação afim.Um
Transformação afim arbitrária pode ser descrita pelos seis parâmetros a
Através de F nessas equações:
x '= ax + cy + e
y '= bx + dy + f
Você pode aplicar uma transformação arbitrária à coordenada atual
sistema passando esses seis parâmetros para o método transform ().
A Figura 15-11 ilustra dois tipos de transformações-tenhas e
rotações sobre um ponto especificado - que você pode implementar com o
Método 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 Snowflakes

O código que produz esses números é elegante, mas o uso de recursivo
As transformações do sistema de coordenadas tornam um pouco difícil
entender.Mesmo se você não seguir todas as nuances, observe que o código
Inclui apenas uma única invocação do método lineto ().Todo
O 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 de
o programa, assim a posição, orientação e duração de cada uma das linhas
segmentos são determinados por traduções, rotações e escala
operações.

Exemplo 15-7.Um floco de neve Koch com transformações

```
Deixe deg = math.pi/180;// para converter graus em radianos
// Desenhe um Fractal de Floco de Neve Koch Level-N no contexto da tela
c,
// com canto inferior à esquerda em (x, y) e comprimento lateral len.
função snowflake (c, n, x, y, len) {
C.Save ();// Salvar a transformação atual
c.Translate (x, y);// traduz a origem para o ponto de partida
c.Moveto (0,0);// Comece um novo subspato no novo
origem
perna (n);// Desenhe a primeira etapa do floco de neve
c.rotate (-120*gra);// agora gira 120 graus
```

```
no sentido anti -horário
perna (n);// Desenhe a segunda perna
c.rotate (-120*gra);// gira novamente
perna (n);// Desenhe a perna final
C.ClosePath ();// Fechar o subspô
C.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 do
perna tem
// desenhado e traduz o sistema de coordenadas para que o
O ponto atual é (0,0).
// isso significa que você pode chamar facilmente girtate () depois de desenhar um
perna.
Função perna (n) {
C.Save ();// salvar a corrente
transformação
if (n === 0) {// caso não recursivo:
c.lineto (Len, 0);// Apenas desenhe uma horizontal
linha
} //
_ _
else {// caso recursivo: desenhe 4 sub-
pernas como: \//
c.Scale (1/3,1/3);// Sub-pernas são 1/3 do tamanho de
esta perna
perna (n-1);// recorrente para o primeiro sub-
perna
c.rotate (60*graus);// gira 60 graus no sentido horário
perna (n-1);// Segunda sub-perna
c.rotate (-120*gra);// gira 120 graus de volta
perna (n-1);// Terceira sub-perna
c.rotate (60*graus);// Gire de volta para o nosso original
cabeçalho
perna (n-1);// Sub-perl
}
C.Restore ();// restaurar a transformação
c.Translate (Len, 0);// mas traduza para fazer o fim de
perna (0,0)
}
}
```

Seja c = document.QuerySelector ("Canvas"). GetContext ("2D");
Snowflake (C, 0, 25, 125, 125);// um floco de neve de nível-0 é um
triângulo
Snowflake (c, 1, 175, 125, 125);// A Nível 1 Snowflake é um 6-
estrela do lado
Snowflake (C, 2, 325, 125, 125);// etc.
Snowflake (C, 3, 475, 125, 125);
Snowflake (C, 4, 625, 125, 125);// ANEL
Como um floco de neve!
c.stroke ();// golpe muito complicado
caminho
15.8.6 recorte
Depois de definir um caminho, você geralmente chama Stroke () ou Fill () (ou
ambos).Você também pode chamar o método clip () para definir um recorte
região.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 recorte
região e depois preenchida após a definição da região de recorte triangular.

Figura 15-13.Movimentos não soltos e preenchimentos cortados

A Figura 15-13 foi gerada usando o método polygon () de Exemplo 15-5 e o seguinte código:

```
// Defina alguns atributos de desenho
C.Font = "Bold 60pt sem serrif";// Big Font
```

C.LineWidth = 2;// linhas estreitas
c.strokestyle = "#000";// linhas pretas
// descreva um retângulo e algum texto
C.STRAISTERECT (175, 25, 50, 325);// Uma faixa vertical para baixo
o meio
c.strokeText ("<Canvas>", 15, 330);// Nota StroKetext ()
Em vez de FillText ()
// Defina um caminho complexo com um interior que está fora.
polígono (c, 3.200.225.200);// Triângulo grande
polígono (c, 3.200.225,100,0, verdadeiro);// reverso menor
Triângulo dentro
// Faça esse caminho a região de recorte.
c.clip ();
// acaricia o caminho com uma linha de 5 pixels, inteiramente dentro do
região 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 () o
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 um
região retangular da sua tela.Você pode criar imagens vazias
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 é
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 linha
deixe i = linha*largura*4 + 4;// O deslocamento do segundo
pixel da linha
for (let col = 1; col <width; col ++, i+= 4) {// para
cada coluna
dados [i] = (dados [i] + dados [i-4]*m)/n;// Vermelho
componente pixel
dados [i+1] = (dados [i+1]+dados [i-3]*m)/n;// Verde
dados [i+2] = (dados [i+2]+dados [i-2]*m)/n;// Azul
dados [i+3] = (dados [i+3]+dados [i-1]*m)/n;// alfa
componente
}
}
// agora copie os dados de imagem manchados de volta para o mesmo
posição na tela
c.putImagedata (pixels, x, y);
}
```

15.9 APIs de áudio

As tags html <udio> e <dide> permitem que você inclua facilmente som 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 o Propriedades de volume e reprodução para controlar o volume de áudio e velocidade de reprodução.E você pode pular para um determinado momento dentro do mídia configurando a propriedade CurrentTime.

Não abordaremos tags <udio> 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 <udio> no seu documento HTML
Para incluir efeitos sonoros em suas páginas da web.Você pode
Crie dinamicamente elementos <udio> com o DOM normal
Document.CreateElement () Método, ou, como um atalho, você pode
Basta usar o construtor Audio ().Você não precisa adicionar o
Elemento criado para o seu documento para reproduzi -lo.Você pode simplesmente
Chame seu método play ():

```
// carrega o efeito sonoro com antecedência, para que esteja pronto para uso
Let Soundeffft = new Audio ("Soundeffft.mp3");
// Reproduza o efeito sonoro sempre que o usuário clica no mouse
botão
document.addeventListener ("clique", () => {
Soundeffft.CloneNode (). Play ();// Carregar e reproduzir o
som
});
```

Observe o uso de CLONENODE () aqui.Se o usuário clicar no mouse
Rapidamente, queremos poder ter várias cópias sobrepostas do
Efeito sonoro tocando ao mesmo tempo.Para fazer isso, precisamos de múltiplos
Elementos de áudio.Porque os elementos de áudio não são adicionados ao
Documento, eles serão coletados de lixo quando terminarem.

15.9.2 A API Webaudio

Além da reprodução de sons gravados com elementos de áudio, Web
Os navegadores também permitem a geração e a reprodução dos sons sintetizados
com a API Webaudio.Usar a API Webaudio é como conectar
Um 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 um
acordes 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
requer
// 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 seno puro
ondas
Deixe as notas = [293.7, 370.0, 440.0];// D Major acorde: D, F#
e a
// Crie nós do oscilador 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 volume
oscillators.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 1,25
segundos.
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 evento
manipuladores
osciladores [0] .AddeventListener ("Ended", () => {
// Este manipulador de eventos é chamado quando a nota parar
jogando
});
```

15.10 Localização, navegação e história

A propriedade de localização da janela e dos objetos do documento refere -se ao objeto de localização, que representa o URL atual do documento exibido na janela e que também fornece uma API para Carregando novos documentos na janela.

O objeto de localização é muito parecido com um objeto URL (§11.9) e você pode usar propriedades como protocolo, nome de host, porta e caminho para Acesse as várias partes do URL do documento atual.O

A propriedade Href retorna todo o URL como uma string, assim como o Método 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 de
O URL que começa com um ponto de interrogação: muitas vezes algum tipo de consulta
corda.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 propriedade SearchParams que é analisada
Representação da propriedade de pesquisa.O objeto de localização não
Tenha uma propriedade SearchParams, mas se você quiser analisar
Window.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 detém 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 o
Atribuir () Método do objeto de localização.Isso é o mesmo que
atribuir a string à propriedade Location, no entanto, não é
particularmente interessante.
O método substituir () do objeto de localização, por outro lado, é
bastante útil.Quando você passa uma string para substituir (), ela é interpretada
como um URL e faz com que o navegador carregue uma nova página, assim como
atribuir () faz.A diferença é que substitui () substitui o
Documento atual na história do navegador.Se um script no documento A
Define a propriedade Location ou as chamadas atribuídas () para carregar o documento B
E então o usuário clica no botão Voltar, o navegador voltará a
Documento A. Se você usa substituir (), então o documento A é
apagado da história do navegador e quando o usuário clica na parte traseira
Botão, o navegador retorna a qualquer documento exibido antes
Documento A.
Quando um script carrega incondicionalmente um novo documento, a substituição ()
O método é uma escolha melhor do que atribuir ().Caso contrário, o botão traseiro
levaria o navegador de volta ao documento original e o mesmo
O script carregaria novamente o novo documento.Suponha que você tenha um
Versão aprimorada de JavaScript da sua página e uma versão estática que faz
não use JavaScript.Se você determinar que o navegador do usuário não
Suporte as APIs da plataforma da web que você deseja usar, você pode usar
location.Replace () para carregar a versão estática:

```
// Se o navegador não suportar as APIs de JavaScript, nós
precisar,
// redireciona para uma página estática que não usa JavaScript.
if (! IsbrowsherSupported ())
```

location.Replace ("staticpage.html");
Observe que o URL passou para substituir () é relativo.Parente
URLs são interpretados em relação à página em que aparecem, assim como
Eles seriam se fossem usados ??em um hiperlink.
Além dos métodos Atribuir () e substituir (), o local
Objeto também define Reload (), que simplesmente faz o navegador
Recarregue o documento.

15.10.2 História de navegação

A propriedade da história do objeto de janela refere -se à história
objeto para a janela.O objeto de história modela o histórico de navegação
de uma janela como uma lista de documentos e documentos estados.O comprimento
propriedade do objeto de história especifica o número de elementos no
Lista de histórico de navegação, mas por razões de segurança, os scripts não podem
Acesse os URLs armazenados.(Se pudessem, qualquer script poderia bisbilhotar
sua história de navegação.)
O objeto de história tem métodos de back () e forward () que
se comportar como os botões de trás e para frente do navegador: eles fazem o
O navegador vai para trás ou para frente um passo em sua história de navegação.UM
terceiro método, go (), pega um argumento inteiro e pode pular qualquer
número de páginas adiante (para argumentos positivos) ou para trás (para
argumentos negativos) na lista de histórico:

```
history.go (-2);// Volte 2, como clicar no botão Voltar
duas vezes
HISTÓRIA.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
intercalado com a história da janela principal.Isso significa isso
Chamando 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 conteúdo
dinamicamente e exibir novos estados de aplicativos sem realmente
Carregando novos documentos.Aplicações como esses devem executar seus
Gerenciamento de histórico próprio se eles querem que o usuário possa usar o
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
Eventos

Uma técnica de gerenciamento de história envolve localização.hash e
O evento "Hashchange".Aqui estão os principais fatos que você precisa saber para
Entenda esta técnica:
A propriedade Location.Hash define o identificador de fragmento
do URL e é tradicionalmente usado para especificar o id de um
Seção de documentos para rolar.Mas location.hash não
tem que ser um ID do elemento: você pode defini -lo como qualquer string.Desde que
Como nenhum elemento tem essa string como seu id, o
O navegador não rola quando você define a propriedade Hash como

esse.
Definindo o local.
exibido na barra de localização e, muito importante, adiciona um
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.
é demitido.E, como mencionamos, essa mudança no local
O objeto cria uma nova entrada no histórico de navegação do navegador.
Então, se o usuário agora clicar no botão Voltar, o navegador irá
Volte 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 demitido neste caso.Isso significa
que, desde que você possa criar um identificador de fragmento único para
Cada estado possível de seu aplicativo, eventos "hashchange"
notificá -lo se o usuário 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
codificar as informações de estado necessárias para renderizar uma "página" do seu
aplicação em uma série relativamente curta de texto adequado para uso como
um identificador de fragmento.E você precisará escrever uma função para converter
Page declare em uma string e outra função para analisar a string e re-
Crie o estado da página que ele representa.
Depois de escrever essas funções, o resto é fácil.Definir a
Window.onhashchange função (ou registre uma ?hashchange?
ouvinte 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, mas
A maioria dos navegadores não o apóia, e você deve simplesmente passar um vazio
corda.O terceiro argumento é um URL opcional que será exibido em
a barra de localização imediatamente e também se o usuário retornar a este estado via
Botões para trás e para frente.URLs relativos são resolvidos contra o
Localização atual do documento.Associando um URL a cada estado
Permite que o usuário marque os estados internos do seu aplicativo.
Lembre -se, porém, que se o usuário salvar um marcador e depois visitar
Um dia depois, você não receberá um evento "popstate" sobre essa visita: você vai
tem que restaurar seu estado de aplicação analisando o URL.
O algoritmo de clone estruturado
O método history.pushstate () não usa json.stringify () (§11.6) para serializar o estado
dados.Em vez
Técnica 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, ele
Permite a serialização da maioria dos outros tipos de JavaScript, incluindo mapa, conjunto, data, regexp e digitado
Matrizes e pode lidar com estruturas de dados que incluem referências circulares.O clone estruturado
No entanto, o algoritmo não pode serializar funções ou classes.Quando clonar objetos, ele não copia o
Objeto de protótipo, getters e setters, ou propriedades não enumeráveis.Enquanto o clone estruturado
O algoritmo pode clonar a maioria dos tipos de javascript embutidos, não pode copiar os tipos definidos pelo host
ambiente, como objetos de elemento do documento.
Isso significa que o objeto de estado que você passa para a história.pushstate () não precisa se limitar ao
Objetos, matrizes e valores primitivos que JSON.Stringify () suporta.Observe, no entanto, que se você
Passe uma instância de uma classe que você definiu, essa instância será serializada como um comum
Javascript Object e perderá seu protótipo.
Além do método pushState (), o objeto de história também
define replacestate (), que leva os mesmos argumentos, mas
substitui o estado da história atual em vez de adicionar um novo estado ao
História de navegação.Quando um aplicativo que usa pushState () é
Primeiro 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 janela
objeto.O objeto de evento associado ao evento tem uma propriedade
nomeado estado, que contém uma cópia (outro clone estruturado) do
Objeto de estado que você passou para PushState ().
Exemplo 15-9 é um aplicativo da web simples-o jogo de adulsão de números
Na 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 número
Exemplo 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" oculto
OnClick = "Location.Search = '';"> reproduza novamente </botão>
<Cript>
/**
* Uma instância desta classe gamestate representa o interno
estado de
* Nosso jogo de adivinhação de número.A classe define fábrica estática
Métodos 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
modificando 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 <
100
s.low = 0;// suposições devem ser
maior que isso
```

```
s.High = 100;// suposições devem ser
Menos do que isso
s.numguesses = 0;// Quantas suposições
foram feitos
S.Guess = NULL;// que último palpite
era
retorno s;
}
// Quando salvamos o estado do jogo com
history.pushstate (), é apenas
// um objeto JavaScript simples que é salvo, não um
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 de
codificar 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 no
url ou isso
// vai doar o segredo.Se o usuário marcar o
página com
// esses parâmetros e depois retorna a ele, nós iremos
Basta escolher um
// novo número aleatório entre baixo e alto.
retornar url.href;
}
```

```
// Esta é uma função de fábrica que cria um novo GameState
objeto e
// inicializa -o a partir do URL especificado.Se o URL fizer
não conter o
// parâmetros esperados ou se forem malformados apenas
retorna 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 precisamos
ou 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 cada
tempo nós
// restaura um jogo de um URL.
S.Secret = S.Randomint (S.Low, S.High);
retorno s;
}
// retorna um número inteiro n, min <n <max
randomint (min, max) {
Retornar min + math.ceil (math.random () * (máx - min -
1));
}
// modifica o documento para exibir o estado atual do
jogo.
render () {
Let Heading = Document.QuerySelector ("#Heading");//
O <H1> no topo
Let Range = Document.QuerySelector ("#Range");//
Exibir intervalo de adivinhação
```

```
Deixe input = document.QuerySelector ("#input");//
Adivinhe o campo de entrada
Deixe PlayAgain = document.QuerySelector ("#PlayAgain");
// Atualize o cabeçalho e o título do documento
heading.TextContent = document.title =
`Estou pensando em um número entre $ {this.low} e
$ {this.high} .`;
// Atualize a faixa visual de números
range.style.marginleft = `$ {this.low}%`;
range.style.width = `$ {(this.high-tis.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.O
entrada
// espaço reservado será exibido porque fizemos a entrada
campo vazio.
if (this.guess === null) {
input.placeholder = "Digite seu palpite e pressione
Digitar";
} 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 usuário
adivinhou.
// 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
estado 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
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 o
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 no navegador
história, 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 estado
Document.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 um
Evento PopState
// no objeto da janela com uma cópia do objeto de estado nós
salvo com
// pushState.Quando isso acontecer, renderize o novo estado.
window.onpopstate = (event) => {
gamestate = gamestate.FromStateObject (event.state);//
Restaurar o estado
gamestate.render ();// e
exibi -lo
};
</script>
</body> </html>
```

15.11 Rede de rede

Toda 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 como As imagens, fontes, scripts e folhas de estilo das quais o arquivo depende.Mas Além de poder fazer solicitações de rede em resposta ao usuário Ações, os navegadores da web também expõem as APIs de JavaScript para redes como bem.

Esta seção abrange três APIs de rede:

O método Fetch () define uma API baseada em promessa para
Fazendo solicitações HTTP e HTTPS.A API Fetch ()
simplifica os pedidos básicos, mas tem um abrangente
Conjunto de recursos que também suporta praticamente qualquer possível uso HTTP
caso.
A API de eventos enviados pelo servidor (ou SSE) é uma conveniente, eventos
interface baseada nas técnicas de ?pesquisa longa? http
O servidor da web mantém a conexão de rede aberta para que possa
Envie dados para o cliente sempre que quiser.
WebSockets é um protocolo de rede que não é HTTP, mas é
Projetado para interoperar com HTTP.Define um
API assíncrona que passa por mensagens onde clientes e servidores
pode enviar e receber mensagens um do outro de uma maneira que
é 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 deseja
recuperar.
2. Obtenha o objeto de resposta que é devolvido de forma assíncrona por
Etapa 1 Quando a resposta HTTP começa a chegar e ligar para um
método deste objeto de resposta para pedir o corpo do
resposta.
3. Obtenha o objeto corporal que é devolvido de forma assíncrona pela etapa 2
e processá -lo como quiser.
A API Fetch () é completamente baseada em promessa e há dois
Passos assíncronos aqui, então você normalmente espera duas chamadas () ou
Duas 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 (ou
Https) Obtenha solicitação
.Then (Response => Response.json ()) // Analisar seu corpo como um
Objeto json
.then (currentUser => {// então processe que
objeto analisado
DisplayUserinfo (CurrentUser);
});
```

Aqui está uma solicitação semelhante feita usando as palavras -chave assíncronas e aguardas
a uma API que retorna uma corda simples em vez de um objeto JSON:

```
função assíncrona isrederviceReady () {
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% de
O que você precisa saber para usar a API Fetch ().As subseções que
Siga demonstrará como fazer solicitações e receber respostas
que são um pouco mais complicados do que os mostrados aqui.

Adeus xmlHttPrequest

A API Fetch () substitui a API barroca e enganosamente nomeada xmlhttprequest (que tem
nada a ver com XML).Você ainda pode ver XHR (como é frequentemente abreviado) no código existente, mas lá
Hoje não é motivo para usá -lo em novo código e não está documentado neste capítulo.Há um
Exemplo de xmlhttprequest neste livro, no entanto, e você pode se referir ao §13.1.3 se quiser ver um
Exemplo de rede JavaScript de estilo antigo.

Códigos de status HTTP, cabeçalhos de resposta e
Erros de rede
O 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) obter
solicitar.
.Then (resposta => {// quando obtemos uma resposta,
primeiro verifique
if (Response.ok && // para um código de sucesso e o
Tipo 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 resolve
DisplayUserinfo (CurrentUser);// Faça algo com
o corpo analisado.
})
.catch (error => {// ou se alguma coisa deu errado,
Basta registrar o erro.
// Se o navegador do usuário estiver offline, buscar () próprio
irá 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.O
A propriedade de status deste objeto é o código de status HTTP, como 200
Para solicitações bem -sucedidas ou 404 para respostas "não encontradas".
(Statustext fornece o texto em inglês padrão que acompanha o
Código 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 for
falso para qualquer outro código.
fetch () resolve sua promessa quando a resposta do servidor começa a
Chegue, assim que o status HTTP e os cabeçalhos de resposta estiverem disponíveis,
Mas normalmente antes que o corpo de resposta completo chegasse.Mesmo que o
O corpo ainda não está disponível, você pode examinar os cabeçalhos neste segundo
Etapa do processo de busca.A propriedade dos cabeçalhos de um objeto de resposta
é um objeto de cabeçalhos.Use seu método Has () para testar a presença de um
Cabeçalho ou use seu método get () para obter o valor de um cabeçalho.Http
Os nomes dos cabeçalhos são insensíveis a maiúsculas, para que você possa passar em minúsculas ou mistas
Nomes 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 promessa
que foi devolvido será cumprido com um objeto de resposta, mesmo que o
A resposta do servidor foi um erro 404 não encontrado ou um servidor interno 500
Erro.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 faz
não existe.Porque essas coisas podem acontecer em qualquer solicitação de rede, é
Sempre uma boa ideia incluir uma cláusula .catch () sempre que você fizer
uma 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/valor
Pares 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 seus
primeiro 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 https
conexão.
AuthHeaders.Set ("Autorização",
`Basic $ {btoa (` $ {nome de usuário}: $ {senha} `)}`);
fetch ("/api/users/", {cabeçalhos: authheaders})
.Then (Response => Response.json ()) // Erro
manuseio 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 a
passar dois argumentos para buscar () é passar os mesmos dois
Argumentos para o construtor de solicitação () e depois passam o resultante
Objeto de solicitação para buscar ():
deixe solicitação = nova solicitação (url, {cabeçalhos});
busca (solicitação) .then (resposta => ...);
Analisar os corpos de resposta
No processo de busca de três etapas () que demonstramos, o
O segundo passo termina chamando os métodos json () ou text () do
Objeto de resposta e retornando o objeto de promessa de que esses métodos
retornar.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 são
As 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 esses
Métodos:
ArrayBuffer ()
Este método retorna uma promessa que resolve a um ArrayBuffer.
Isso é útil quando a resposta contém dados binários.Você pode usar
o ArrayBuffer para criar uma matriz digitada (§11.2) ou um dataView
Objeto (§11.2.5) do qual você pode ler os dados binários.
blob ()
Este método retorna uma promessa que resolve a um objeto BLOB.Blobs
não estão cobertos com detalhes neste livro, mas o nome significa
"Objeto grande binário" e eles são úteis quando você espera grande
quantidades de dados binários.Se você pedir o corpo da resposta como um
BLOB, a implementação do navegador pode transmitir os dados de resposta para
um arquivo temporário e depois retorne um objeto BLOB que representa que
arquivo temporário.Objetos de blob, portanto, não permitem acesso aleatório
para o corpo de resposta, a maneira como um matriz faz.Uma vez que você
ter uma bolha, você pode criar um URL que se refere a ele
Url.createObjecturl (), ou você pode usar o evento baseado em evento
API FileReader para obter assíncrono o conteúdo do Blob como
uma string ou uma matriz.No momento da redação deste artigo, alguns
Os navegadores também definem o texto baseado em promessa () e
Métodos ArrayBuffer () que fornecem uma rota mais direta para
obtendo 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 resposta
a ser codificado no formato "Multipart/Form-Data".Este formato é
comum em solicitações de postagem feitas a um servidor, mas incomum em
Respostas do servidor, portanto, esse método não é usado com frequência.

Corpos de resposta de streaming
Além dos cinco métodos de resposta que retornam assíncronos
Alguma forma do corpo de resposta completo para você, há também um
opção para transmitir o corpo de resposta, o que é útil se houver alguns
tipo de processamento que você pode fazer nos pedaços do corpo de resposta como
Eles chegam pela rede.Mas transmitir a resposta também é útil
Se você quiser exibir uma barra de progresso para que o usuário possa ver o
Progresso do download.
A propriedade corporal de um objeto de resposta é um objeto ReadableStream.Se
Você já chamou um método de resposta como text () ou json ()
Isso lê, analisa e devolve o corpo, então BodyUsed será verdadeiro
para indicar que o fluxo corporal já foi lido.Se Bodysed usou
é falso, no entanto, o fluxo ainda não foi lido.Nesse caso,
você pode ligar para getReader () em resposta.body para obter um fluxo
Objeto do leitor e use o método read () deste objeto de leitor para
Leia de forma assíncrona pedaços de texto do fluxo.O read ()
o método retorna uma promessa que resolve a um objeto com feito e
Propriedades do valor.feito será verdadeiro se todo o corpo for lido
ou 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 e
aguarde, mas é surpreendentemente complexo se você tentar usá -lo com cru
Promessas.Exemplo 15-10 demonstra a API definindo um
função streambody ().Suponha que você quisesse baixar um grande
JSON 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 um
A função UpdateProgress () é definida para definir o atributo de valor
em 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 em
Exemplo 15-10.

Exemplo 15-10.Transmitindo o corpo de resposta a partir de uma solicitação de busca ()

```
/**
* Uma função assíncrona para transmitir o corpo de um
Objeto 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, que
RelatórioProgress
* 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
* devolvido por esse retorno de chamada.Caso contrário, a string é a
concatenação de
* Os valores de bloco convertidos em strings UTF-8.
*/
Função assíncreada streambody (resposta, reportprogress,
ProcessChunk) {
// Quantos bytes esperamos, ou nan se não houver cabeçalho
Deixe o esperado
Comprimento"));
deixe bytesRead = 0;// Quantos bytes
recebido até agora
Let Reader = Response.body.getReader ();// leia bytes com
esta função
Deixe decodificador = novo textDecoder ("UTF-8");// para converter
bytes para texto
Deixe Body = "";// Texto lido So
distante
while (true) {// loop até
Saímos abaixo
Seja {feito, value} = aguardar leitor.read ();// Leia a
pedaço
se (valor) {// se conseguirmos
Uma matriz de bytes:
if (ProcessChunk) {// Processo
os bytes se
deixe processado = processChunk (valor);// a
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 chame
RelatórioProgress (BYTESRAD, BYTESRAD /
esperado);
}
}
se (feito) {// se isso for
o último pedaço,
quebrar;// Saia do
laço
}
}
corpo de retorno;// retorna o texto do corpo que acumulamos
}
Esta API de streaming é nova no momento da redação deste artigo e deve
evoluir.Em particular, há planos para fazer objetos readableStream
itemerável de forma assíncrona para que eles possam ser usados ??para/aguardam
Loops (§13.4.1).
Especificando o método de solicitação e solicitação
CORPO
Em cada um dos exemplos de busca () mostrados até agora, fizemos um
Http (ou https) obtenha solicitação.Se você quiser usar um pedido diferente
Método (como post, put ou exclusão), basta usar os dois-
Versão de argumento de fetch (), passando um objeto de opções com um
parâmetro do método:
Fetch (url, {método: "post"}). Então (r =>
r.json ()). Então (HandleResponse);
Publicar e colocar solicitações normalmente têm um corpo de solicitação contendo dados
a 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
Especifique 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 "texto/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
Especifique 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 um
digite 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, e
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 parece
a 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 formDados 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
e inicializando o nome do nome/valor que ele representa com o
Métodos set () e append ().
Upload de arquivo com fetch ()
O upload de arquivos do computador de um usuário para um servidor da web é comum
tarefa e pode ser realizado usando um objeto FormData como solicitação
corpo.Uma maneira comum de obter um objeto de arquivo é exibir uma <entrada
type = "arquivo"> ??elemento na sua página da web e ouça "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.Arquivo
Os objetos também estão disponíveis na API de arrastar e soltar HTML.Que
API não está abordada neste livro, mas você pode obter arquivos do
DataTransfer.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 web
Aplicativo que permite ao usuário criar desenhos em um <Canvas>
elemento.Você pode fazer upload dos desenhos do usuário como arquivos PNG com código
Como 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 tela
função 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 cruzada
Na 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 o
Script 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 seguro
solicitações de origem cruzada.Quando Fetch () é usado com uma origem cruzada
URL, o navegador adiciona um cabeçalho de "origem" à solicitação (e não
Permitir que ele seja substituído pela propriedade dos cabeçalhos) para notificar a web
servidor que a solicitação vem de um documento com um diferente
origem.Se o servidor responder à solicitação com um apropriado
Cabeçalho ?Access-Control-Alow-Origin? e a solicitação prossegue.
Caso contrário, se o servidor não permitir explicitamente a solicitação, então o
A promessa devolvida por fetch () é rejeitada.
Abortando um pedido
Às vezes, você pode querer abortar um pedido de busca ()
já emitido, talvez porque o usuário clique em um botão de cancelamento ou o
O pedido está demorando muito.A API busca permite que os pedidos sejam abortados
usando as classes abortcontroller e abortSignal.(Essas classes
definir um mecanismo de aborto genérico adequado para uso por outras APIs como
bem.)
Se você quiser ter a opção de abortar uma solicitação buscada (), então
Crie um objeto abortController antes de iniciar a solicitação.O
A 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
Chame o método abort () do objeto do controlador para abortar a solicitação,
que causará quaisquer objetos de promessa relacionados à solicitação de busca para
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 completo
// dentro do número de milissegundos especificados por isso
propriedade.
função fetchwithtimeout (url, opções = {}) {
if (options.timeout) {// se houver propriedade de tempo limite
e é diferente de zero
Let Controller = novo abortController ();// Crie a
controlador
options.signal = controller.signal;// Defina o
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 cancele
// Este temporizador.Chamar abort () após a busca for
completo has
// sem efeito.
setTimeout (() => {controller.abort ();},
options.timeout);
}
// agora apenas faça uma busca normal
retornar buscar (URL, opções);
}
```

Opções de solicitação diversas
Vimos 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:

cache

Use 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 verifiquei 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 eles
solicitações.Alguns aplicativos da web acham útil, no entanto, ter seu servidor
Envie -lhes notificações quando ocorrerem eventos.Isso não vem
naturalmente para HTTP, mas a técnica que foi criada é para o
cliente para fazer uma solicitação ao servidor, e então nem o cliente nem o
Servidor Fechar a conexão.Quando o servidor tem algo para dizer o
Cliente sobre, ele grava dados na conexão, mas o mantém aberto.O
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
aberto para sempre, mas se o cliente detectar que a conexão foi fechada, ela
pode simplesmente fazer outra solicitação para reabrir a conexão.
Esta técnica para permitir que os servidores enviem mensagens para os clientes é
surpreendentemente eficaz (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 solicitação de longa duração para um servidor da web, basta passar um URL para
o construtor Eventsource ().Quando o servidor grava (corretamente
Dados formatados) 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
Isso 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 de
os eventos que são gerados.Se o servidor omitar um nome de evento no
Dados 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 um
conexão com o servidor (quando cria o objeto Eventsource),
e o servidor mantém essa conexão aberta.Quando ocorre um evento, o
O servidor grava linhas de texto na conexão.Um evento passando por cima do
Wire pode ficar assim, se os comentários foram omitidos:

```
Evento: BID // define o tipo de objeto de evento
Dados: Goog // Define a propriedade de dados
Dados: 999 // anexa uma nova linha e mais dados
// Uma linha em branco marca o final do evento
```

Existem alguns detalhes adicionais para o protocolo que permitem que os eventos sejam
dados IDs e permitir que um cliente de reconexão para dizer ao servidor qual é o ID
do último evento que recebeu foi, para que um servidor possa reenviar quaisquer eventos
perdeu.Esses detalhes são invisíveis no lado do cliente, no entanto, e
não são discutidos aqui.
Um aplicativo óbvio para eventos enviados ao servidor é para multiuser
Colaborações como bate -papo online.Um cliente de bate -papo pode usar o fetch () para
poste mensagens na sala de bate -papo e assine o fluxo de conversas
com um objeto Eventsource.Exemplo 15-11 demonstra como é fácil
é 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 </ititle> </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: sólido
Black 2px "/>
<Cript>
// Cuide de alguns detalhes da interface do usuário
Deixe Nick = Prompt ("Digite seu apelido");// Obtenha o usuário
apelido
deixe input = document.getElementById ("input");// Encontre a entrada
campo
input.focus ();// Defina o teclado
foco
// Registre -se para notificação de novas mensagens usando o EventSource
Let Chat = New Eventsource ("/Chat");
Chat.addeventListener ("Chat", evento => {// Quando um bate -papo
Mensagem chega
deixe div = document.createElement ("div");// Crie um <div>
div.append (event.data);// Adicionar texto de
a mensagem
input.be antes (div);// e adicione div
antes da entrada
input.ScrollIntoView ();// Garanta a entrada
ELT é visível
});
// Publique as mensagens do usuário no servidor usando busca
input.addeventListener ("alteração", () => {// Quando o usuário
greves retornam
busca ("/chat", {// iniciar um http
solicitação a este URL.
Método: "Post", // Faça uma postagem
solicitação com o corpo
corpo: nick + ":" + input.value // definido como o usuário
Nick e entrada.
})
.catch (e => console.error);// ignora a resposta, mas
registre 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 simples
Servidor HTTP.Quando um cliente solicita o URL root "/", ele envia o chat
Código do cliente mostrado no Exemplo 15-11.Quando um cliente faz um Get
Solicitação para o URL "/bate -papo", ele salva o objeto de resposta e mantém isso
conexão 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 o
Formato "Text/Event-stream" para cada um dos objetos de resposta salvos.O
O código do servidor escuta na porta 8080, então depois de executá -lo com nó, ponto
Seu navegador para http: // localhost: 8080 para se conectar e começar
conversando 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 com
Nodejs.
// 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 de
mensagens
// do mesmo URL.Fazer uma solicitação para / retorna um
arquivo 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 enviar
eventos para
deixe 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 função
Server.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 cliente
Ui.
if (pathname === "/") {// uma solicitação para a interface do usuário de bate -papo
Response.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
gerado quando
// o cliente cria um novo objeto de ourcepção (ou quando o
Eventsource
// se reconecta automaticamente).
Função AcceptNewClient (solicitação, resposta) {
// Lembre -se do objeto de resposta para que possamos enviar futuro
mensagens para isso
clientes.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 apenas
um cliente
Response.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 enviado
Eventos 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 assíncroada 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 e
feche a conexão
Response.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 o 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 escuta
clientes.ForEach (client => client.write (evento));
}
15.11.3 Websockets
A API do WebSocket é uma interface simples para um complexo e poderoso
Protocolo de rede.Os websockets permitem o código JavaScript no navegador
Troca facilmente mensagens de texto e binário com um servidor.Como em
Eventos enviados ao servidor, o cliente deve estabelecer a conexão, mas uma vez
A conexão é estabelecida, o servidor pode enviar de forma assíncrona
mensagens para o cliente.Ao contrário da SSE, as mensagens binárias são suportadas e
As 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 nível
soquetes de rede, terminais de conexão não são identificados por endereço IP
e porta.Em vez disso, quando você deseja se conectar a um serviço usando o
Protocolo WebSocket, você especifica o serviço com um URL, assim como você
faria para um serviço da web.URLs WebSocket começam com WSS: //
em vez de https: //, no entanto.(Os navegadores normalmente restringem
WebSockets para trabalhar apenas em páginas carregadas em https seguros: //
conexões).
Para estabelecer uma conexão WebSocket, o navegador primeiro estabelece um
Conexão HTTP e envia ao servidor uma atualização: WebSocket
Cabeçalho solicitando que a conexão seja alterada do HTTP
protocolo para o protocolo Websocket.O que isso significa é que, a fim de
Use websockets em seu JavaScript do lado do cliente, você precisará ser

Trabalhando com um servidor da web que também fala o protocolo WebSocket,
E 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 isso
A seção explicará tudo o que você precisa saber para lidar com o cliente-
extremidade lateral da conexão.Se o seu servidor não suportar o
Protocolo da WebSocket, considere o uso de eventos enviados pelo servidor (§15.11.2)
em vez de.

Criando, conectando e desconectando
WebSockets

Se você deseja se comunicar com um servidor habilitado para WebSocket, crie um
Objeto WebSocket, especificando o wss: // url que identifica o
Servidor e serviço que você deseja usar:

```
Deixe soquete = new websocket ("wss: //example.com/stockticker");
```

Quando você cria um webSocket, o processo de conexão começa
automaticamente.Mas um WebSocket recém -criado não será conectado
quando é devolvido pela primeira vez.

A propriedade ReadyState do soquete especifica em que afirma
A conexão está dentro. Esta propriedade pode ter os seguintes valores:

WebSocket.Connecting
Este WebSocket está conectando.
WebSocket.open
Este 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 inicial
A tentativa falha.
Quando um WebSocket transita da conexão para a abertura
Estado, ele dispara um evento "aberto" e você pode ouvir este evento por
Definindo a propriedade ONOPEN do WebSocket ou ligando
addEventListener () nesse objeto.
Se um protocolo ou outro erro ocorrer para uma conexão WebSocket, o
O objeto WebSocket dispara um evento de "erro".Você pode definir o OnError para
Defina um manipulador ou, alternativamente, use addEventListener ().
Quando terminar com um webSocket, você pode fechar a conexão por
Chamando o método Close () do objeto WebSocket.Quando a
WebSocket muda no estado fechado, ele dispara um evento "próximo" e
Você pode definir a propriedade OCLOSE para ouvir este evento.
Enviando mensagens sobre um webSocket
Para enviar uma mensagem para o servidor do outro lado de um webSocket
conexão, basta invocar o método send () do websocket
objeto.send () espera um único argumento de mensagem, que pode ser um
String, Blob, ArrayBuffer, Array digitado ou objeto DataAView.
O método send () buffer a mensagem especificada a ser transmitida
e retorna antes que a mensagem seja realmente enviada.O

propriedade bufferedamount do objeto websocket especifica o
Número 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 webSocket
Para receber mensagens de um servidor em um WebSocket, registre um evento
Manipulador para eventos de "mensagem", definindo o OnMessage
propriedade do objeto WebSocket, ou ligando
addEventListener ().O objeto associado a uma "mensagem"
Evento é uma instância do MessageEvent com uma propriedade de dados que contém
a mensagem do servidor.Se o servidor enviou um texto codificado UTF-8, então
Event.Data será uma string que contém esse texto.
Se o servidor enviar uma mensagem que consiste em dados binários em vez de
texto, então a propriedade de dados (por padrão) será um objeto BLOB
representando esses dados.Se você preferir receber mensagens binárias como
Arraybuffers em vez de blobs, defina a propriedade Binarype do
WebSocket objeto para a string "ArrayBuffer".
Existem várias APIs da Web que usam objetos de MessageEvent para
trocando mensagens.Algumas dessas APIs usam o clone estruturado
Algoritmo (consulte "O algoritmo de clone estruturado") para permitir o complexo
Estruturas de dados como carga útil da mensagem.Websockets não é um desses
APIs: as mensagens trocadas por um websocket são uma única string
de caracteres unicode ou uma única sequência de bytes (representada como uma bolha
ou 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 seus
Protocolo de comunicação próprio 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 a
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
Faça isso, você está especificando uma lista de protocolos de aplicativos que você conhece
Como lidar e pedir ao servidor para escolher um.Durante a conexão
processo, o servidor escolherá um dos protocolos (ou falhará com um
erro se não suportar nenhuma das opções do cliente).Uma vez o
A conexão foi estabelecida, a propriedade do protocolo do
O objeto WebSocket especifica qual versão do protocolo o servidor escolheu.

15.12 Armazenamento

Aplicativos 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 use -o como mecanismo de comunicação.Entrada de dados em um formulário Em 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 pode ser armazenado 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 web

A API de armazenamento da Web consiste na localização e objetos de sessão, que são essencialmente persistentes Objetos 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.

Biscoitos

Os 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 Escritsable no lado do cliente, mas eles são difíceis de usar e Adequado 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
IndexedDB é uma API assíncrona para um banco de dados de objetos que
suporta indexação.
Armazenamento, segurança e privacidade
Os navegadores da web geralmente se oferecem para lembrar senhas da web para você, e eles as armazenam com segurança em
formulário criptografado no dispositivo.Mas nenhuma das formas de armazenamento de dados do lado do cliente descrito neste
Capítulo Envolver Criptografia: Você deve assumir que qualquer coisa que seus aplicativos da Web salvam residem em
o 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 um
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 usuário
resposta.
}
Você pode usar o operador de exclusão para remover as propriedades de
LocalStorage e SessionStorage, e você pode usar um
para/em loop ou object.Keys () para enumerar as propriedades de um
Objeto 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 () e
Métodos DeleteItem (), que você pode usar em vez de direto
Acesso à propriedade e o operador Excluir, se desejar.
Lembre -se de que as propriedades dos objetos de armazenamento só podem armazenar
cordas.Se você deseja armazenar e recuperar outros tipos de dados, você vai
tem que codificar e decodificá -lo sozinho.
Por exemplo:
// Se você armazenar um número, ele é automaticamente convertido em um
corda.
// 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 quando
recebendo
LocalStorage.Lastread = (new Date ()). ToutcString ();
Seja Lastread = new Date (DATE.PARSE (LocalStorage.Lastread));
// json faz uma codificação conveniente para qualquer primitivo ou dados

estrutura
LocalStorage.data = json.Stringify (dados);// codifica e
loja
deixe dados = json.parse (localStorage.data);// recuperar e
decodificar.
Vida útil e escopo de armazenamento
A diferença entre LocalStorage e SessionStorage
Envolve a vida e o escopo do armazenamento.Dados armazenados
LocalStorage é permanente: não expira e permanece armazenado
no dispositivo do usuário até que um aplicativo da web o exclua ou o usuário pergunta o
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 seu
Protocolo, nome do host e porta.Todos os documentos com o mesmo compartilhamento de origem
os mesmos dados de armazenamento local (independentemente da origem dos scripts
que realmente acessam o localStorage).Eles podem ler um do outro
dados, e eles podem substituir os dados um do outro.Mas documentos com
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
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 é excluído.(Observe, no entanto, isso
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 o
documentar a origem para que documentos com origens diferentes nunca
Compartilhe sessionStorage.Mas sessionStorage também está escopo
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
substitua os dados escritos por scripts na outra guia, mesmo que ambas as guias
estão visitando exatamente a mesma página e estão executando exatamente o mesmo
scripts.
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 isso
Os dados são visíveis (mas não na janela que fizeram a alteração).Se a
navegador tem duas guias abertas para páginas com a mesma origem e uma de
Essas páginas armazenam um valor no localStorage, a outra guia irá
Receba 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 Remofitem () 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.

Storageea

O objeto de armazenamento que mudou.Este é geralmente o Objeto LocalStorage.

url

O URL (como uma string) do documento cujo script fez isso 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 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 -lo em visitas futuras.E armazenando a preferência, gera um evento que

permite que outras janelas exibam o mesmo site para honrar a solicitação que
bem.
Como outro exemplo, imagine um aplicativo de edição de imagem baseado na Web
Isso permite ao usuário exibir paletas de ferramentas em janelas separadas.Quando
O usuário seleciona uma ferramenta, o aplicativo usa o LocalStorage para salvar
o estado atual e para gerar uma notificação para outras janelas que um
Nova ferramenta foi selecionada.

15.12.2 Cookies

Um cookie é uma pequena quantidade de dados nomeados armazenados pelo navegador da web
e associado a uma página ou site da Web específico.Os biscoitos eram
Projetado para programação do lado do servidor e, no nível mais baixo, eles são
implementado como uma extensão ao protocolo HTTP.Dados de cookies são
transmitido automaticamente entre o navegador da web e o servidor da web, então
Os 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 podem
manipular cookies usando a propriedade de cookies do documento
objeto.

Por que "cookie"?

O nome "Cookie" não tem muito significado, mas não é usado sem precedentes.No
Anais da história da computação, o termo "cookie" ou "biscoito mágico" tem sido usado para se referir a um pequeno
pedaço de dados, particularmente um pedaço de dados privilegiados ou secretos, semelhante a uma senha, que prova a identidade
ou permite acesso.No JavaScript, os cookies são usados ??para salvar o estado e podem estabelecer um tipo de identidade
Para um navegador da web.Os cookies em JavaScript não usam nenhum tipo de criptografia, no entanto, e não são
Segure de qualquer maneira (embora transmiti -los em um HTTPS: a conexão ajuda).
A API para manipular biscoitos é antiga e enigmática.Há
Não 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 especificado individualmente com atributos de cookie.Esses atributos são
também especificado com seqüências de cordas especialmente formatadas no mesmo cookie
propriedade.
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.O
String 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 isso
Cookie.(Vamos falar sobre atributos a seguir.) Para fazer uso do
Propriedade Document.Cookie, normalmente você deve chamar a divisão ()
Método para dividi -lo em pares de nome/valor individuais.
Depois de extrair o valor de um biscoito do biscoito
propriedade, você deve interpretar esse valor com base em qualquer formato ou
A codificação foi usada pelo criador do cookie.Você pode, por exemplo,
Passe 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 cujo
As 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 com
codeuricomponent ().
function getcookies () {
deixe cookies = novo map ();// o objeto que retornaremos
deixe tudo = document.cookie;// Obtenha todos os cookies em um grande
corda
deixe list = all.split (";");// dividido em individual
pares de nome/valor
para (vamos cookie da lista) {// para cada cookie naquele
lista
if (! Cookie.includes ("=")) continuar;// Pule se lá
não é = sinal
Seja p = cookie.indexof ("=");// Encontre o
primeiro = sinal
deixe o nome = cookie.substring (0, p);// Obtenha biscoito
nome
deixe 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 escopo

Além de um nome e um valor, cada cookie tem atributos opcionais que controlam sua vida útil e escopo.Antes que possamos descrever como definir Cookies com JavaScript, precisamos explicar os atributos de cookies.

Os cookies são transitórios por padrão;Os valores que eles armazenam duram para o 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 escoposos por origem do documento.Grandes sites
pode querer que os cookies sejam compartilhados entre os subdomínios, no entanto.Para
Exemplo, o servidor em order.example.com pode precisar ler cookie
Valores 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 um
Cookie 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 são
conectado 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
transferido 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 você
O armazenado está incluído na lista de cookies para o documento.Valores de biscoito
Não pode incluir semicolons, vírgulas ou espaço em branco.Por esse motivo,
Você pode querer usar a função global JavaScript central
codeuricomponent () para codificar o valor antes de armazená -lo em
o biscoito.Se você fizer isso, terá que usar o correspondente
Decodeuricomponent () função quando você lê o cookie
valor.
Um cookie escrito com um nome simples de nome/valor dura para a corrente
Sessão de navegação na Web, mas é perdida quando o usuário sai do navegador.Para
Crie um cookie que possa durar nas sessões do navegador, especifique seu
Lifetime (em segundos) com um atributo em idade máxima.Você pode fazer isso por
Definir a propriedade Cookie como uma string do formulário: name = value;
Max-Aage = segundos.A função a seguir define um cookie com um
Atributo opcional Max-Age:
// armazenar o par de nome/valor como um cookie, codificando o valor
com
// codeuricomponent () para escapar de semicolons,
vírgulas e espaços.
// Se Daystolive for um número, defina o atributo da era máxima para
que o biscoito
// expira após o número especificado de dias.Passe 0 para
Exclua 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 escopos
Documento: Duas páginas da web com a mesma origem podem acessar
dados, mas páginas da web de diferentes origens não podem.
Cada origem pode ter qualquer número de bancos de dados indexedDB.Cada um
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
O nome implica, um loja de objetos armazena objetos.Objetos são serializados em
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
pode ter propriedades cujos valores são mapas, conjuntos ou matrizes digitadas.Cada
o objeto deve ter uma chave pela qual pode ser classificado e recuperado do
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
pode ser classificado.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
Você insere em um armazenamento de objetos já terá uma propriedade que é
Adequado 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
Defina qualquer número de índices no armazenamento de objetos.(A capacidade de indexar
Um armazenamento de objetos explica o nome "indexedDB".) Cada índice define um
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
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 -los 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
Procure um objeto chamando o método get () do armazenamento de objetos ou
Armazene um novo objeto chamando put () (ou chamando add (), se você quiser
para evitar a substituição de objetos existentes).
Se você quiser procurar os objetos para uma variedade de chaves, você cria um
Objeto Idbrange que especifica os limites superior e inferior do
alcance 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 um único
chave ou um objeto Idbrange.
Essa simplicidade conceitual da API indexedDB é complicada,
No entanto, pelo fato de a API ser assíncrona (para que os aplicativos da web
pode 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
baseado 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 loja e 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, e você pode definir manipuladores com
as propriedades Onsuccess e OnError.Dentro de um ãccess
Manipulador, 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
Execute consultas e atualizações.Em uma API síncrona, você esperaria
Para marcar explicitamente o fim da transação chamando um commit ()
método.Mas com indexedDB, as transações são automaticamente
comprometido (se você não os abortar explicitamente) quando todo o
Os manipuladores de eventos Onsuccess foram 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 um
Evento "Atualizada" no objeto de solicitação retornado pelo
Indexeddb.open () CALL.O trabalho do manipulador de eventos para
Eventos "Atualizados" é definir ou atualizar o esquema para o novo
banco de dados (ou a nova versão do banco de dados existente).Para indexedDB
bancos de dados, isso significa criar armazenamentos de objetos e definir índices em
Esses objetos armazenam.E, de fato, a única vez que a API indexedDB
permite que você crie um armazenamento de objetos ou um índice é uma resposta a um
Evento "Upgradeneeded".
Com esta visão geral de alto nível do indexedDB em mente, você deve agora
ser capaz de entender o exemplo 15-13.Esse exemplo usa indexedDB
Para criar e consultar um banco de dados que nos mapeia códigos postais (códigos postais) para
Cidades dos EUA.Demonstra muitos, mas não todos, das características básicas de
Indexeddb.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 assíncrono
objeto (criando
// e inicializando o db, se necessário) e passa para o
ligar de volta.
função withdb (retorno de chamada) {
let request = indexeddb.open ("zipcodes", 1);// Solicitação v1
do banco de dados
request.onerror = console.error;// registre quaisquer erros
request.Onsuccess = () => {// ou ligue para isso quando feito
Seja db = request.Result;// o resultado do pedido
é o banco de dados
retorno de chamada (dB);// Invoque o retorno de chamada com
o banco de dados
};
// Se a versão 1 do banco de dados ainda não existir, então
este 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.
//
// Nosso banco 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.
função initdb (db, retorno de chamada) {
// Crie o armazenamento de objetos, especificando um nome para a loja
e
// 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
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.zip
busca ("zipcodes.json") // faça um http obter
solicitar
.Then (Response => Response.json ()) // Analisar o corpo
como JSON
.THEN (ZIPCODES => {// Get 40K CEP
registros
// para inserir dados de código postal no
Banco de dados que precisamos de um
// Objeto de transação.Para criar nossa transação
objeto, precisamos
// Para especificar quais armazenamentos de objetos usaremos
(Nós só temos
// um) e precisamos dizer que estaremos fazendo
escreve para o
// banco de dados, não apenas lê:
Deixe transação = db.transaction (["ZipCodes"],
"readwrite");
transaction.onerror = console.error;
// Obtenha nossa loja de objetos na transação
deixe armazenar = transação.ObjectStore ("ZipCodes");
// A melhor parte da API indexedDB é que
Armazenamento de objetos
// são * realmente * simples.Veja como adicionamos (ou
atualização) nossos registros:
para (deixe o registro de zipcodes) {store.put (registro);}
// Quando a transação é concluída com êxito, o
banco de dados
// está inicializado e pronto para uso, para que possamos ligar
o
// função de retorno de chamada que foi originalmente passado para
withdb ()
transaction.oncomplete = () => {retorno de chamada (db);};
});
}
// Dado um código postal, use a API indexedDB para assíncrona
Procure 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 isso
consulta.O
// O argumento é uma variedade de lojas de objetos que precisaremos
para usar.
Deixe transação = db.transaction (["ZIPCODES"]);
// Obtenha a loja de objetos da transação
Seja zipcodes = transaction.ObjectStore ("ZipCodes");
// agora solicita o objeto que corresponda ao especificado
Chave ZIPCODE.
// As linhas acima eram síncronas, mas esta é
assíncrono.
deixe solicitação = zipcodes.get (zip);
request.onerror = console.error;// erros de log
request.Onsuccess = () => {// ou chame isso
função no sucesso
deixe registrar = request.Result;// Esta é a consulta
resultado
se (registro) {// se encontrarmos uma correspondência, passe para
o retorno de chamada
retorno de chamada (`$ {Record.city}, $ {Record.state}`);
} else {// caso contrário, diga ao retorno de chamada que
nós falhamos
retorno de chamada (nulo);
}
};
});
}
// Dado o nome de uma cidade, use a API indexedDB para
assíncrono
// 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 objeto
loja
```

```
Deixe transação = db.transaction (["ZIPCODES"]);
deixe armazenar = transação.ObjectStore ("ZipCodes");
// Desta vez, também obtemos o índice da cidade do objeto
loja
deixe index = store.index ("cidades");
// Peça todos os registros correspondentes no índice com o
especificado
// nome da cidade, e quando os pegamos, passamos para o
ligar de volta.
// Se esperávamos mais resultados, poderíamos usar
OpenCursor () 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 mensagens

Uma das características fundamentais do JavaScript é que ele é único
Trebo: 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 documento
simplesmente não são possíveis, e os programadores do lado do cliente não precisam
Pense em programação simultânea ou mesmo entender.Um corolário
É 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 irá
não 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 com
A 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 Mensagem assíncrona 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 acho útil criar dezenas de trabalhadores, mas é improvável que um A aplicação com centenas ou milhares de trabalhadores seria prática. Os trabalhadores são úteis quando seu aplicativo precisa executar Tarefas computacionalmente intensivas, como processamento de imagens.Usando a 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 moderadamente intensivos 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 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 rosqueamento, existem duas partes na API do trabalhador.O

Primeiro é o objeto de trabalhador: é assim que um trabalhador se parece do
lá fora, para o tópico que o cria.O segundo é o
Workerglobalscope: este é o objeto global para um novo trabalhador, e ele
é como é um tópico de trabalhador, por dentro, para si mesmo.
As seções a seguir cobrem o trabalhador e o workerglobalscope e também
Explique a API que passa de mensagem que permite que os trabalhadores se comuniquem
com o fio principal e um ao outro.A mesma API de comunicação é
usado para trocar mensagens entre um documento e <frame>
elementos contidos no documento, e isso é coberto no
Seções a seguir também.

15.13.1 Objetos trabalhadores

Para criar um novo trabalhador, ligue para o construtor do trabalhador (), passando um
URL 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 do
documento que contém o script que chamou de trabalhador ()
construtor.Se você especificar um URL absoluto, deve ter o mesmo
Origem (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 () será
copiado usando o algoritmo de clone estruturado (consulte ?O clone estruturado
Algoritmo ?), e a cópia resultante será entregue ao trabalhador via
Um evento de mensagem:

datacruncher.postMessage ("/api/data/to/crunch");
Aqui estamos apenas passando uma única mensagem de string, mas você também pode usar
Objetos, matrizes, matrizes digitadas, mapas, conjuntos e assim por diante.Você pode receber
mensagens de um trabalhador ouvindo eventos de "mensagem" no
Objeto de trabalhador:
datacruncher.onmessage = function (e) {
deixe estatísticas = e.data;// A mensagem é a propriedade de dados
do evento
console.log (`média: $ {stats.mean}`);
}
Como todas as metas de eventos, os objetos do trabalhador definem o padrão
addEventListener () e RemowEventListener ()
Métodos, e você pode usá -los no lugar da OnMessage.
Além de PostMessage (), os objetos trabalhadores têm apenas um outro
método, terminine (), que força um fio de trabalhador a parar
correndo.
15.13.2 O objeto global em trabalhadores
Quando você cria um novo trabalhador com o construtor trabalhador (), você
Especifique o URL de um arquivo de código JavaScript.Esse código é executado em um
Novo ambiente de execução de JavaScript, isolado do
Script que criou o trabalhador.O objeto global para essa nova execução
O ambiente é um objeto WorkerglobalsCope.Um Workerglobalscope é
algo mais do que o objeto global Javascript central, mas menos de um
Objeto 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 dos
Objeto de trabalhador, mas trabalhe na direção oposta: chamando
PostMessage () dentro de um trabalhador gera um evento de mensagem fora
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 na
objeto global.Um trabalhador pode incluir esse nome em qualquer mensagem
Imprima 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 tudo
As propriedades do objeto global Javascript central, como o JSON
objeto, a função isnan () e o construtor date ().Em
Além disso, no entanto, o workerglobalscope também tem o seguinte
Propriedades 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 não
Defina uma propriedade de janela.

Os métodos do timer setTimeout (), clearTimeout (),
setInterval () e ClearInterval ().
Uma propriedade de localização que descreva o URL que foi
Passado para o construtor Worker ().Esta propriedade refere -se a um
Objeto de localização, exatamente como a propriedade de localização de uma janela
faz.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 com
propriedades como as do objeto de navegador de uma janela.UM
O objeto Navigator do trabalhador tem o nome de propriedades AppName,
AppVersion, Platform, UserAgent e Online.
Os métodos de destino de evento usuais addEventListener ()
e removaeventListener ().
Finalmente, o objeto WorkerglobalsCope inclui um importante lado ao cliente
APIs JavaScript, incluindo o objeto do console, a função fetch (),
e a API indexedDB.Workerglobalscope também inclui o
Trabalhador () construtor, o que significa que os threads de trabalhadores podem criar
seus próprios trabalhadores.

15.13.3 Importar código para um trabalhador

Os trabalhadores foram definidos em navegadores da web antes do JavaScript ter um módulo
Sistema, portanto, os trabalhadores têm um sistema exclusivo para incluir código adicional.
Workerglobalscope define o importaScripts () como um global
função que todos os trabalhadores têm acesso a:

```
// Antes de começarmos a trabalhar, carregue as aulas e utilitários
Vamos precisar
ImportScripts ("utils/histogram.js", "utils/bitset.js");
```

Importscripts () leva um ou mais argumentos de URL, cada um dos
que deve se referir a um arquivo de código JavaScript.URLs relativos são
resolvido em relação ao URL que foi passado para o trabalhador ()
construtor (não em relação ao documento que contém).
ImportScripts () Carrega e executa de maneira síncrona
um após o outro, na ordem em que foram especificados.Se carregar
Um script causa um erro de rede, ou se a execução lança um erro de qualquer
Classifique, nenhum dos scripts subsequentes é carregado ou executado.Um script
Carregado com importação () pode ser chamado
ImportScripts () para carregar os arquivos depende.Nota, no entanto,
que o importaScripts () não tenta acompanhar quais scripts
já carregou e não faz nada para evitar ciclos de dependência.
ImportScripts () é uma função síncrona: não retorna
até que todos os scripts tenham carregado e executado.Você pode começar a usar
os scripts que você carregou assim que o importScripts () retorna: lá
Não é necessário um retorno de chamada, manipulador de eventos, então () método ou aguarda.
Depois de internalizar a natureza assíncrona do lado do cliente
JavaScript, é estranho voltar a simples, síncrono
programação novamente.Mas essa é a beleza dos tópicos: você pode usar um
Bloqueio de chamadas de função em um trabalhador sem bloquear o loop de evento em
o fio principal, e sem bloquear os cálculos sendo
realizada simultaneamente em outros trabalhadores.
Módulos em trabalhadores
Para 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 sequência
"Module".Passando a
TIPO: Opção "Módulo" para o construtor Worker () é muito parecido com o uso do tipo = "módulo"
atributo em uma tag html <cript>: 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 declarações em trabalhadores.
15.13.4 Modelo de execução do trabalhador
Os threads trabalhadores executam seu código (e todos os scripts ou módulos importados)
Sincrono de cima para baixo e depois entra em um assíncrono
fase em que eles respondem a eventos e temporizadores.Se um trabalhador registrar
um manipulador de eventos de "mensagem", ele nunca sairá enquanto houver um
possibilidade de que os eventos de mensagem ainda cheguem.Mas se um trabalhador não
Ouça 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, lá
não é de maneira a 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 Global Close ().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 pego por nenhuma captura
Cláusula, então um evento de "erro" é acionado no objeto global do
trabalhador.Se este evento for manuseado e o manipulador chama o
PreventDefault () Método do objeto de evento, o erro
A propagação termina.Caso contrário, o evento "erro" é demitido no trabalhador

objeto.Se prevenirDefault () for chamado lá, então a propagação
termina.Caso contrário, uma mensagem de erro é impressa no console do desenvolvedor
e 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.linenono}: $ {e.message} `);
E.PreventDefault ();
};
// ou, lide com erros de trabalhador não capturados com um manipulador do lado de fora
o trabalhador.
trabalhador.onerror = function (e) {
console.log (`Erro no Trabalhador em
$ {e.filename}: $ {e.linenono}: $ {e.message} `);
E.PreventDefault ();
};
Como o Windows, os trabalhadores podem registrar um manipulador para ser invocado quando um
A promessa é rejeitada e não há função .catch () para lidar com isso.
Dentro de um trabalhador, você pode detectar isso definindo um
Função de Self.OnUnHandledRejection ou usando
addEventListener () para registrar um manipulador global para
Eventos de ?rejeição não entrega?.O objeto de evento passou para este manipulador
terá uma propriedade de promessa cujo valor é o objeto de promessa que
rejeitado e uma propriedade pela qual a propriedade cujo valor é o que teria sido
Passado para uma função .catch ().
15.13.5 PostMessage (), Messageports e
Messagechannels

O método PostMessage () do objeto de trabalhador e o global
PostMesage () função definida dentro de um trabalhador que trabalha por
Invocando os métodos de PostMessage () de um par de Messageport
objetos que são criados automaticamente junto com o trabalhador.Cliente-
JavaScript lateral não pode acessar diretamente esses criados automaticamente
MessagePort Objects, mas pode criar novos pares de portas conectadas
com o construtor messagechannel ():

```
deixe canal = new Messagechannel;// Crie a
novo canal.
deixe myport = canal.port1;// tem
duas portas
deixe yourport = canal.port2;// conectado
um com o outro.
myport.postMessage ("Você pode me ouvir?");// uma mensagem
Postado em um Will
yourport.onMessage = (e) => console.log (e.data);// ser
recebido no outro.
```

Um Messagechannel é um objeto com propriedades Port1 e Port2
que se referem a um par de objetos de Messageport conectados.Um messageport é
Um objeto com um método PostMessage () e um evento onMessage
Propriedade do manipulador.Quando PostMessage () é chamado em um porto de um
Par conectado, um evento de "mensagem" é disparado na outra porta no par.
Você pode receber esses eventos de "mensagem" definindo o OnMessage
propriedade ou usando addEventListener () para registrar um ouvinte
para 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.Esse
impede que as mensagens enviadas por uma extremidade do canal sejam perdidas

pela outra extremidade.Se você usar addEventListener () com um
Messageport, não se esqueça de ligar para Start () ou você pode nunca ver um
mensagem entregue.
Todas as chamadas PostMessage () que vimos até agora fizeram um único
argumento da mensagem.Mas o método também aceita um segundo opcional
argumento.Este segundo argumento é uma variedade de itens que devem ser
transferido para o outro extremo do canal em vez de enviar uma cópia
através do canal.Os valores que podem ser transferidos em vez de copiados são
Messageports e ArrayBuffers.(Alguns navegadores também implementam outros
Tipos transferíveis, como ImageBitmap e OffScreencanvas.Esses
não são universalmente apoiados, no entanto, e não são cobertos neste
Livro.) Se o primeiro argumento a PostMessage () incluir um
Messageport (aninhado em qualquer lugar dentro do objeto de mensagem), então que
Messageport também deve aparecer no segundo argumento.Se você fizer isso,
então o Messageport estará disponível para o outro extremo do
canal e se tornará imediatamente não funcional do seu lado.
Suponha que você criou um trabalhador e queira ter dois canais para
Comunicação com ele: um canal para troca de dados comum e
Um canal para mensagens de alta prioridade.No tópico principal, você pode
Crie um Messagechannel e ligue para PostMessage () no trabalhador
Para 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 como
esse
```

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
Transfira 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 original
Thread para que não haja possibilidade de acesso simultâneo ao seu conteúdo.
Se o primeiro argumento a PostMessage () incluir um ArrayBuffer, ou
Qualquer 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 Cliente
JavaScript 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
em 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 o mesmo-
a política de origem impede que seu script veja o conteúdo de outro
Janela, você ainda pode ligar para PostMessage () naquela janela e
Fazer isso fará 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 de
O método PostMessage () de um trabalhador, no entanto.O primeiro
O argumento ainda é uma mensagem arbitrária que será copiada pelo
Algoritmo de clone estruturado.Mas a listagem opcional do segundo argumento
Objetos a serem transferidos em vez de copiados se tornam um terceiro opcional
argumento.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

Espere estar recebendo a mensagem.Se você passar pela string
"Https://good.example.com" como o segundo argumento, mas a janela
você está postando a mensagem para realmente contém conteúdo de
"Https://malware.example.com", então a mensagem que você postou não será
ser entregue.Se você estiver disposto a enviar sua mensagem para o conteúdo de
Qualquer origem, você pode passar o curinga ?*? como o segundo argumento.
Código JavaScript em execução dentro de uma janela ou <frame> pode receber
Mensagens postadas nessa janela ou quadro definindo o OnMessage
propriedade dessa janela ou ligando para addEventListener () para
Eventos de "mensagem".Como com os trabalhadores, quando você recebe uma "mensagem"
Evento para uma janela, a propriedade de dados do objeto de evento é o
mensagem 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 fonte
janela.Isso não é algo que o remetente da mensagem pode forjar,
E quando você recebe um evento de "mensagem", você normalmente deseja
Verifique se é de uma origem que você espera.

15.14 Exemplo: o conjunto Mandelbrot

Este capítulo sobre JavaScript do lado do cliente culmina com um longo exemplo
que demonstra o uso de trabalhadores e mensagens para paralelizar
tarefas computacionalmente intensivas.Mas está escrito para ser um envolvente,
Aplicativo da Web do mundo real e também demonstra um número do
Outras APIs demonstradas neste capítulo, incluindo história

gerenciamento;uso da classe IMAGEDATA com um <Canvas>;e o
Uso de teclado, ponteiro e redimensionamento eventos.Também demonstra
Recursos importantes de JavaScript, incluindo geradores e um
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 complexo Multiplicação e adição, produzem um valor cuja magnitude permanece limitado.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 do Conjunto de Mandelbrot, você deve calcular individualmente a associação de Cada um dos 250.000 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 um valioso 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 o grade retangular de pixels para os quais está computando Mandelbrot Conjunto de membros.Mas em vez de armazenar real Valores de pixel na imagem, ele usa uma matriz de tinta personalizada para Trate 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 ao
Imaginoutata 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 Mandelbrot
definir
// Este é um trabalhador simples que recebe uma mensagem de seu
tópico pai,
// executa o cálculo descrito por essa mensagem e depois
Publica 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 o
Array retangular
// de pixels, obtenha seu ArrayBuffer interno e crie um
Visualização 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
transformado em
// cores de pixel reais no thread pai.)
const imagagedata = new IMAGEDATA (largura, altura);

```
const iterations = new UINT32Array (imagedata.data.buffer);
// Agora começamos o cálculo.Existem três aninhados para
loops 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 o
pixel 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
o
// 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 imaginário
parte de z (n+1).
r = rr - ii + x;// e a parte real de
z (n+1).
}
iterações [index ++] = n;// Lembrar #
iterações para cada pixel.
if (n> max) max = n;// rastrear o máximo
número que vimos.
if (n <min) min = n;// e o mínimo como
bem.
}
}
// Quando o cálculo estiver concluído, envie os resultados de volta
para o pai
// fio.O objeto de imagem será copiado, mas o
Arraybuffer gigante
// ele contém será transferido para uma boa performance
impulsionar.
PostMessage ({Tile, IMAGEDATA, MIN, MAX},
[imagedata.data.buffer]);
};
```

O aplicativo Mandelbrot Set Viewer que usa esse código do trabalhador é mostrado no Exemplo 15-15.Agora que você quase chegou ao fim de Este capítulo, este longo exemplo é uma experiência de Capstone Isso reúne um número importante e do lado do cliente Recursos de JavaScript e APIs.O código é completamente comentado, e Encorajo você a lê -lo com cuidado.

Exemplo 15-15.Um aplicativo da web para exibir e explorar o Mandelbrot Conjunto

```
/*
* Esta classe representa um sub -rangão de uma tela ou imagem.
Usamos ladrilhos para
* Divida uma tela em regiões que podem ser processadas
independentemente pelos trabalhadores.
*/
```

```
Classe Tile {
construtor (x, y, largura, altura) {
this.x = x;// as propriedades de um
Objeto de ladrilho
this.y = y;// representa o
posição e tamanho
this.width = width;// do ladrilho dentro de um
maior
this.Height = altura;// retângulo.
}
// Este método estático é um gerador que divide um
retângulo do
// largura e altura especificadas no número especificado de
linhas e
// colunas e produz NumRows*Numcols Tile Objects para cobrir
o 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 // Altura
da maioria das linhas
: altura - altura da linha * (numrows -1);// altura
da última linha
para (let col = 0; col <numcols; col ++) {
Deixe TileWidth = (col <numcols-1)
?ColumnWidth // Largura
da maioria das colunas
: largura - largura de coluna * (numcols -1);// e
Última coluna
Rendimento 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
recebe por
* executar algum tipo de computação e depois postar um
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
* Objeto que está inativo atualmente, a mensagem será postada para
aquele trabalhador
* imediatamente.Se não houver objetos de trabalhador ocioso, o
mensagem será
* na fila e será postado para um trabalhador quando alguém se tornar
disponível.
*
* addwork () retorna uma promessa, que resolverá com o
mensagem recebida
* do trabalho, ou rejeitará se o trabalhador jogar um
erro não atendido.
*/
classe Workerpool {
construtor (NumWorkers, Workersource) {
this.idleworkers = [];// trabalhadores que não são
atualmente trabalhando
this.workQueue = [];// trabalho não atualmente
sendo processado
this.WorkerMap = new Map ();// mapeia os trabalhadores para resolver
e rejeitar funcs
// Crie o número especificado de trabalhadores, adicione mensagem
e erro
// Os manipuladores e 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 termina
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
a 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 um
Prometa isso
// será resolvido com a resposta de um trabalhador quando o trabalho for
feito.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 um
Mandelbrot Conjunto.
* As propriedades CX e Cy dão o ponto no plano complexo
esse é o
* centro da imagem.A propriedade Perpixel especifica como
muito o real e
* partes imaginárias desse número complexo mudam para cada
pixel da imagem.
* A propriedade Maxiterations especifica o quanto trabalhamos para
Calcule o conjunto.
* Números maiores requerem mais computação, mas produzem mais nítidos
imagens.
* Observe que o tamanho da tela não faz parte do estado.
Dado CX, CY, e
* Perpixel, simplesmente renderizamos qualquer parte do Mandelbrot
o conjunto se encaixa
* A tela em seu tamanho atual.
*
* Objetos deste tipo são usados ??com histórico.pushstate () e
são usados ??para ler
* O estado desejado de um URL marcado ou compartilhado.
*/
classe pagestate {
// Este método de fábrica retorna um estado inicial para exibir
o conjunto inteiro.
estático inicialState () {
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 retorna
nulo 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 do
Params 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 no
procurar
// parâmetros do local atual do navegador.
Tourl () {
deixe u = novo url (window.location);
U.SearchParams.Set ("CX", this.cx);
U.SearchParams.set ("cy", this.cy);
U.SearchParams.Set ("pp", this.perpixel);
U.SearchParams.Set ("it", this.maxiterations);
retornar u.href;
}
}
// Essas constantes controlam o paralelismo do Mandelbrot
Defina a computação.
// Pode ser necessário ajustá -los para obter o melhor desempenho em
seu computador.
const linhas = 3, cols = 4, NumWorkers =
Navigator.hardwarEcoCurrency ||2;
// Esta é a classe principal do nosso programa Mandelbrot Set.Simplesmente
```

```
invocar o
// Função construtora com o elemento <latavas> para renderizar
em.O programa
// pressupõe que este elemento <Canvas> seja estilizado para que seja
sempre tão grande
// como a janela do navegador.
classe MandelbrotCanvas {
construtor (tela) {
// Armazene a tela, obtenha seu objeto de contexto e
Inicialize um Workerpool
this.Canvas = Canvas;
this.Context = Canvas.getContext ("2D");
this.workerpool = new Workerpool (NumWorkers,
"Mandelbrotworker.js");
// define algumas propriedades que usaremos mais tarde
this.tiles = null;// sub -regiões da tela
this.pendingRender = null;// não estamos atualmente
renderização
this.wantsRerender = false;// Nenhuma renderização está atualmente
solicitado
this.resizetimer = null;// nos impede de
redimensionar com muita frequência
this.colortable = null;// para converter dados brutos
para valores de pixel.
// Configure nossos manipuladores de eventos
this.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.initialState ();
// 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 que
cubra.
this.SetSize ();
// e renderize o Mandelbrot definido na tela.
this.render ();
}
// Defina o tamanho da tela e inicialize uma matriz de ladrilhos
objetos.Esse
// o método é chamado do construtor e também pelo
HandleResize ()
// 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 depois
renderiza o
// Mandelbrot conjunto usando esse novo estado e também salva o
novo estado com
// history.pushstate ().Se o primeiro argumento é uma função
essa função
// será chamado com o objeto de estado como seu argumento e
deve fazer
// muda para o estado.Se o primeiro argumento é um
objeto, então nós simplesmente
// copie as propriedades desse objeto no estado
objeto.Se opcional
// O segundo argumento é falso, então o novo estado não será
salvo.(Nós
// Faça isso ao chamar SetState em resposta a um PopState
evento.)
SetState (f, salvar = true) {
// Se o argumento for uma função, chame para atualizar o
estado.
// caso contrário, copie suas propriedades na corrente
estado.
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 um
Evento 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
mudanças.
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;// anota
Reerender mais tarde
```

```
retornar;// e não faça
qualquer coisa mais agora.
}
// Obtenha nossas variáveis ??de estado e calcule o complexo
Número 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 o
promessa 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 de
a 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 a
os pixels.
Seja min = maxiterations, max = 0;
```

Erro ao traduzir esta página.

```
}
} outro {
// no caso normal em que Min e Max estão
Diferente, use a
// Escala logarítica para atribuir cada um possível
Iteração conta um
// opacidade entre 0 e 255 e depois use o
mudança 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 cada
Respostas
// fotografata para as cores do colortável.
para (deixe r de respostas) {
Deixe iterações = novo
Uint32Array (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 em
deles
// ladrilhos correspondentes da tela usando
putImagedata ().
// (primeiro, porém, remova qualquer transformação de CSS no
tela 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);
}
})
```

```
.CACH ((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 o
Bandeiras pendentes
this.pendingRender = null;
// e se os pedidos de renderização chegaram enquanto estávamos
Ocupado, 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 mandlebrot é
um caro
// operação que não podemos fazer várias vezes por segundo, então
Usamos um cronômetro
// para adiar o manuseio do redimensionamento até que 200 ms tenham decorrido
Desde 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
foi tratado
this.SetSize ();// redimensione tela e
azulejos
this.render ();// reender 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 inicial
this.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 direita
this.setState (s => s.cx += this.width/10 *
s.perpixel);
quebrar;
}
}
// Este método é chamado quando obtemos um evento de ponteiro em
a tela.
// O evento Pointerdown pode ser o início de um zoom
gesto (um clique ou
// toque) ou um gesto de pan (um arrasto).Este manipulador registra
manipuladores para
// Os eventos de ponteiro e ponteiro para responder
para o resto
// do gesto.(Esses dois manipuladores extras são removidos
Quando o gesto
// termina com um ponteiro.)
handlepointer (evento) {
// as coordenadas do pixel e o tempo da inicial
ponteiro para baixo.
// porque a tela é tão grande quanto a janela, estes
Coordenadas 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 tem
passou?
Deixe dx = event.clientx-x0, dy = event.clienty-y0,
dt = date.now ()-t0;
// se o ponteiro se moveu o suficiente ou o suficiente
já passou isso
// Este não é um clique regular e, em seguida, use CSS para pan
a tela.
// (nós o renderia de verdade quando conseguirmos o
Evento 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 ponteira
const 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 indivíduo
constantes.
const {cx, cy, perpixel} = this.state;
// se o ponteiro se moveu o suficiente ou se o suficiente
O tempo passou, então
// Este foi um gesto de pan e precisamos mudar
estado para mudar
// O ponto central.Caso contrário, o usuário clicou ou
bateu em um
// apontar e precisamos centralizar e aumentar o zoom
apontar.
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 pixels
O 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)
escala (2) `;
// Defina as coordenadas complexas do novo
ponto 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 manipuladores
para 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 isso
Arquivo javascript
// é auto-suficiente.O arquivo html precisa incluir isso apenas
um <cript>.
Let Canvas = Document.CreateElement ("Canvas");// Crie a
elemento de tela
document.body.append (Canvas);// insira -o
no corpo
document.body.style = "margem: 0";// Sem margem para
o <body>
canvas.style.width = "100%";// faz tela
Tão largo quanto o corpo
canvas.style.Height = "100%";// e tão alto
como o corpo.
New MandelbrotCanvas (Canvas);// e inicie
Renderizando nele!
```

15.15 Resumo e sugestões para Leitura adicional

Este capítulo longo cobriu os fundamentos do lado do cliente 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 modelo de programação.

O Modelo de Objeto do Documento (DOM) que permite JavaScript código para inspecionar e modificar o conteúdo HTML do documentar 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 aplicado ao conteúdo dentro do documento.

Como o código JavaScript pode obter as coordenadas do documento elementos 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 um
forma segura de simultaneidade.
Este tem sido o capítulo mais longo do livro, de longe.Mas não pode
Chegue perto de cobrir todas as APIs disponíveis para os navegadores da Web.O
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
e use 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
termine 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 web
Kit 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 de
O objeto da janela é o ponto de entrada principal para esta API.Inclui a
desempenho de fonte de tempo de alta resolução.now () e métodos
performance.mark () e performance.measure () para
marcando pontos críticos em seu código e medindo o tempo decorrido
entre eles.Chamar esses métodos cria objetos de desempenho de desempenho
que você pode acessar com performance.getentries ().Navegadores
Adicione seus próprios objetos de desempenho sempre que o navegador carrega um
nova página ou busca um arquivo sobre a rede e estes automaticamente
Os objetos de desempenho criados incluem detalhes de tempo granular de
O desempenho da rede do seu aplicativo.O relacionado
A classe de performanceObServer permite que você especifique uma função para ser
Invocado quando novos objetos de desempenho de desempenho são criados.

15.15.3 Segurança

Este capítulo introduziu a idéia 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
formato 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 promotos () 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 para
Projetos e protótipos simples.
As propriedades do navegador e da tela da janela
o objeto foi mencionado na passagem no início deste capítulo,
Mas os objetos de navegador e tela que eles fazem referência têm
Alguns recursos que não foram descritos aqui que você pode encontrar
útil.
O método solicitfullscreen () de qualquer elemento
Objeto solicita que esse elemento (a <dife> ou <lVAs>
elemento, por exemplo) ser exibido no modo de tela cheia.O
o método de saída do documento retorna para
Modo de exibição normal.
O método requestanimationframe () da janela
Objeto assume uma função como seu argumento e executará que
função quando o navegador está se preparando para renderizar o próximo
quadro.Quando você está fazendo mudanças visuais (especialmente
repetidos ou animados), envolvendo seu código em uma chamada
para requestanimationframe () pode ajudar a garantir que
as mudanças são renderizadas suavemente e de uma maneira que é
otimizado pelo navegador.
Se o usuário selecionar o texto em seu documento, você poderá obter
Detalhes dessa seleção com o método da janela
getSelection () e obtenha o texto selecionado com
getSelection (). ToString ().Em alguns navegadores,
Navigator.clipboard é um objeto com uma API assíncrona
para ler e definir o conteúdo da área de transferência do sistema para
Ativar interações de cópia e colar com aplicativos fora
do navegador.
Um recurso pouco conhecido dos navegadores da web é que html

elementos com um atributo contentedável = "true"
Permita que seu conteúdo seja editado.O
Document.ExecCommand () Método permite o texto rico
Recursos de edição para conteúdo editável.
Um mutationObserver permite que o JavaScript monitore as alterações,
ou abaixo, um elemento especificado no documento.Crie a
MutationObServer com o mutationObServer ()
construtor, passando a função de retorno de chamada que deve ser chamado
Quando as alterações são feitas.Em seguida, ligue para o método Observe ()
do mutationObserver para especificar quais partes das quais
elemento deve ser monitorado.
Um intersectionObserver permite que o JavaScript determine qual
Os elementos do documento estão na tela e que estão próximos
estar na tela.É particularmente útil para aplicações
que desejam carregar dinamicamente o conteúdo sob demanda como o usuário
Rolls.
15.15.6 Eventos
O grande número e a diversidade de eventos apoiados pela web
A plataforma pode ser assustadora.Este capítulo discutiu uma variedade de eventos
Tipos, mas aqui estão alguns mais que você pode achar útil:
Os navegadores disparam eventos "online" e "offline" na janela
Objeto quando o navegador ganha ou perde uma conexão com a Internet.
Os navegadores disparam um evento "VisiblityChange" no documento
objeto quando um documento se torna visível ou invisível (geralmente
porque um usuário alterou as guias).JavaScript pode verificar
document.VisibilityState para determinar se o seu
O 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 mouse
ponteiro e obtenha eventos de mouse bruto como movimento relativo
valores 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 Web Aplicações 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://serviceworke.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 é 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 o scripts e outros ativos do aplicativo da web que ele serve e também podem 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 web aplicação incluindo um nome, um URL e links para ícones em vários tamanhos.Se o seu aplicativo da web usar um trabalhador de serviço e 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
Somente trabalhe em dispositivos Android e não em dispositivos iOS.)
A API de geolocalização permite JavaScript (com o usuário
permissão) 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
Registre um retorno de chamada para ser chamado quando a posição do usuário
mudanças.
O método Navigator.vibrate () causa um celular
dispositivo (mas não iOS) para vibrar.Muitas vezes isso só é permitido em
resposta a um gesto de usuário, mas chamar esse método permitirá
seu aplicativo para fornecer 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
Treque -se à paisagem ou orientação de retrato.
Os eventos de "Devicemoção" e "Devicereentation" 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
Os 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
Os sensores permitem que o JavaScript determine em qual direção um 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
Basta ler arquivos locais.Mas se o usuário selecionar um arquivo para fazer upload (usando um
<input type = "FILE> elemento do formulário) ou usa arrastar e soltar para
Solte um arquivo no seu aplicativo da web, o JavaScript pode acessar que
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 promessa
métodos 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.O
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 <dide> (definindo o
Propriedade srcObject para o fluxo).Os quadros do vídeo ainda podem ser
capturado em uma tela fora de tela <VAS> com a tela
função drawImage () resultando em uma resolução relativamente baixa
fotografia.Fluxos de áudio e vídeo devolvidos por getUsermedia ()
pode ser gravado e codificado 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 seguro
Números pseudorandomos.Outros métodos para criptografia, descriptografia, chave

geração, assinaturas digitais e assim por diante estão disponíveis através
Crypto.subtle.O nome desta propriedade é um aviso para
Todo mundo que usa esses métodos que usam adequadamente o criptográfico
algoritmos são difíceis e que você não deve usar esses métodos, a menos que
Você realmente sabe o que está fazendo.Além disso, os métodos de
Crypto.subtle está 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
permitir que o JavaScript gere, armazenasse 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 infraestrutura substancial
é necessário no lado do servidor para fazer esses métodos funcionarem.Essas APIs
ainda não são universalmente apoiados, mas têm o potencial de
Revolucione 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
Solicite um pagamento Crie um objeto PaymentRequest e ligue para o show ()
Método para exibir a solicitação ao usuário.
1
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
tornou -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.

2
Algumas 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.
3
Se 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.
4
A 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 SERVIDO

JavaScript com nó

O nó é JavaScript com ligações ao sistema operacional subjacente,
possibilitando escrever programas JavaScript que leem e escrevem
Arquivos, execute processos filhos e se comunique pela rede.Esse
Torna o nó útil como um:

Alternativa moderna aos scripts de concha que não sofrem de
A sintaxe arcana de Bash e outras conchas Unix.

Linguagem de programação de uso geral para executar confiança
programas, não sujeitos às restrições de segurança impostas por
Navegadores da Web em código não confiável.

Ambiente popular para escrever eficiente e altamente
Servidores da Web simultâneos.

A característica definidora do nó é o seu evento único baseado em eventos
Concorrência ativada por uma API assíncrona por defesa.Se você tem
programado em outras línguas, mas não fiz muito JavaScript
Codificação, ou se você é um programador JavaScript do lado do cliente experiente
acostumado a escrever código para navegação na web, usar o nó será um pouco de
Ajuste, assim como qualquer nova linguagem ou ambiente de programação.Esse
o capítulo começa explicando o modelo de programação do nó, com um
ênfase na simultaneidade, a API do Node para trabalhar com streaming
Dados e o tipo de buffer do Node para trabalhar com dados binários.Esses
As 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
A esperança é que este capítulo explique o suficiente dos fundamentos para
Torne você produtivo com o nó e confiante de que você pode dominar qualquer
Novas APIs 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
Macos.No Linux, você poderá instalar o nó com o seu gerenciador de pacotes normal, ou você pode
Visite https://nodejs.org/en/download para baixar os binários diretamente.Se você trabalha em contêiner
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
O capítulo usará apenas os pacotes internos do Node e não exigirá 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
estruturado 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 pequenas 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 stdout
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
invocado 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 será
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 Unix
variáveis.Nó os disponibiliza pelo processo.env
objeto.Os nomes de propriedades deste objeto são variáveis ??de ambiente
nomes e os valores da propriedade (sempre strings) são os valores daqueles
variáveis.
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 você
poderia reescrever a linha acima como nó --eval 'process.env' -
-imprimir.

16.1.3 Ciclo de vida do programa

O comando do nó espera um argumento da linha de comando que especifica
o arquivo do código JavaScript a ser executado.Este arquivo inicial normalmente importa
outros módulos do código JavaScript e também podem definir suas próprias classes
e funções.Fundamentalmente, no entanto, o Node executa o JavaScript
Código no arquivo especificado de cima para baixo.Alguns programas de nós saem
quando 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 inicial
foi executado.Como discutiremos nas seções a seguir, nó

Os programas geralmente são assíncronos e baseados em retornos de chamada e evento manipuladores.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 há mais eventos pendentes.Um programa de servidor baseado em nó que ouve
As conexões de rede recebidas terão teoricamente para sempre porque
Sempre 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 no Janela do terminal onde o programa está em execução.Um programa pode ignorar Ctrl-C registrando uma função de manipulador de sinal com
process.on ("sigint", () => {}).
Se o código em seu programa lançar uma exceção e nenhuma cláusula de captura Pega, o programa imprimirá um rastreamento e saída de pilha.Por causa de
A natureza assíncrona do nó, exceções que ocorrem em retornos de chamada ou Os manipuladores de eventos devem ser tratados localmente ou não tratados, que significa que lidar com exceções que ocorrem nas partes assíncronas de
Seu programa pode ser um problema difícil.Se você não quer isso
exceções para fazer com que seu programa trava completamente, registre um global Função 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 for rejeitado e não há invocação .catch () para lidar com isso.A partir do nó 13, este não é um erro fatal que faz com que seu programa sai, mas
Imprima 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
erros.Se você não deseja rejeições não tratadas, imprimir mensagens de erro
ou encerrar seu programa, registre uma função de manipulador global:
Process.on ("UNHLELLEDRIJECEJAÇÃO", (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.Nó
O 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
Os sistemas de módulos não são totalmente compatíveis, então isso é um tanto complicado
fazer.O nó precisa saber - antes de carregar um módulo - seja isso
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
outra mão, quando o nó carrega um arquivo de código como um módulo ES6, ele deve
ative 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
codifique essas informações na extensão do arquivo.Se você salvar seu
Código JavaScript em um arquivo que termina com .mjs, então o nó sempre
Carregue -o como um módulo ES6, espera 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 o tratará como um
Módulo Commonjs, fornecerá uma função requer () e irá
Jogue um SyntaxError se usar declarações de importação ou exportação.
Para arquivos que não possuem uma extensão .mjs ou .cjs explícito, olha de nó
Para 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",
Em seguida, 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 tal
o arquivo é encontrado (ou quando o arquivo é encontrado, mas não tem um tipo
Propriedade), os padrões do nó usando os módulos Commonjs.Esse
Package.json truque só se torna necessário se você quiser usar ES6
Módulos com nó e não desejam 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 como 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 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 respostas.

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 3.058s
encontrou 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 eles
poderia simplesmente digitar o NPM instalar para baixar automaticamente e
Instale 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 para 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.
Paralelismo verdadeiro com nó
Os programas de nó podem executar vários processos de sistema operacional e o nó 10 e o trabalhador de suporte posterior
Objetos (§16.11), que são um tipo de tópico emprestado dos navegadores da Web.Se você usar múltiplos processa 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 a 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 local
FileSystem.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
Esperando o disco girando antes que uma operação de arquivo possa começar.
E em datacenters modernos, o sistema de arquivos "local" pode realmente ser
em toda a rede em algum lugar com latências de rede em cima da unidade
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, para
Exemplo, também são não bloqueadores.
Algumas funções na API do Node são síncronas, mas não bloqueando: elas
Corra 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 promessa
Classe, as APIs de nó assíncronas são baseadas em retorno de chamada.(Se você não tem
No entanto, leia ou já esqueci o capítulo 13, isso seria um bom
Hora de pular de volta para esse capítulo.) Geralmente, o último argumento que você
Passar 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
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 ou
A resposta foi produzida pela função assíncrona original que você
chamado.O motivo para colocar o argumento de erro primeiro é fazê -lo
impossível para você omiti-lo, e você sempre deve verificar um não
valor nulo neste argumento.Se for um objeto de erro, ou mesmo um número inteiro
Código de erro ou mensagem de erro da string, então algo deu errado.Nesta
Caso, o segundo argumento para sua função de retorno de chamada provavelmente será
nulo.
O código a seguir demonstra como usar o não bloqueio
Função 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 nulo
função readconfigfile (caminho, retorno de chamada) {
fs.readfile (caminho, "utf8", (err, text) => {
se (err) {// algo deu errado lendo o
arquivo
console.error (err);
retorno de chamada (nulo);
retornar;
}
deixe dados = nulo;
tentar {
dados = json.parse (texto);
} catch (e) {// algo deu errado analisando o
Conteúdo do arquivo
console.error (e);
}
retorno de chamada (dados);
});
}
```

O nó antecede as promessas padronizadas, mas porque é bastante consistente
Sobre seus retornos de chamada em primeiro lugar, é fácil criar promessas baseadas em promessas
variantes 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 arquivos
const pfs = {// variantes baseadas em promessas de alguns
funções fs
ReadFile: 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 para
manusear dados de streaming.Cobriremos os eventos do nó com mais detalhes mais tarde.
Agora que discutimos a API agressivamente sem bloqueio de Node, vamos
Volte 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 leva
ação para iniciar a operação e depois registra algum tipo de evento
manipulador com o sistema operacional para que seja notificado quando o
Operação 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 o
O sistema operacional envia o evento apropriado para o nó.
Esse tipo de simultaneidade é frequentemente chamado de concorrência baseada em eventos.No
Seu núcleo, nó tem um único thread que executa um "loop de eventos".Quando a
O programa de nó é iniciado, ele executa o código que você disse para executar.Esse
Código presumivelmente chama pelo menos uma função não bloqueadora, causando um
Retorno de chamada ou manipulador de eventos a ser registrado no sistema operacional.(Se

Não, então você escreveu um programa de nós síncronos e um nó
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.Nó mapeia o evento do sistema operacional para o JavaScript
O retorno de chamada que você se registrou e chama essa função.Seu retorno de chamada
a 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
Feito correndo, 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
A simultaneidade é eficiente e eficaz.Um servidor da web pode simultaneamente
lidar com solicitações de 50 clientes diferentes sem precisar de 50 diferentes
tópicos, desde que usem APIs não bloqueadores e existe algum tipo de
Mapeamento interno de soquetes de rede para funções de JavaScript para
Invoque quando a atividade 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 o
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
Core JavaScript suportado matrizes digitadas (ver §11.2) e não havia
UINT8Array para representar uma variedade de bytes não assinados.Nó definiu o
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 personagem
cordas.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 um
codificação de caracteres, podemos decodificar esses bytes em uma sequência de
caracteres.A classe de buffer do nó tem métodos que executam os dois
codificação e decodificação, e você pode reconhecer esses métodos porque
Eles esperam um argumento codificante que especifica que a codificação é
usado.
As codificações no nó são especificadas pelo nome, como strings.O suportado
As 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 hexadecimal
dígitos.
"Base64"
Esta codificação converte cada sequência de três bytes em um
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]);// <buffer
41 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 ++) {// Use
Buffer 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-bytes
valores
// de e para um buffer em qualquer deslocamento especificado.
Dead.readUint32be (0) // => 0xdeadbeef
Dead.readUint32be (1) // => 0xadbeefde
Dead.readbiguint64be (6) // => 0xbeefdeadbeefdeadn
Dead.readUint32LE (1020) // => 0xEfbeadde
Se você escrever um programa de nós que realmente manipula dados binários, você
Pode se encontrar usando a classe buffer extensivamente.Por outro
mão, se você está apenas trabalhando com texto que é lido ou escrito para um
arquivo ou rede, então você pode encontrar apenas buffer como um
Representação intermediária de seus dados.Uma série de APIs de nó podem
Pegue 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 desses
APIs, você precisará especificar o nome da codificação de texto que deseja
usar.E se você fizer isso, talvez não precise usar um objeto buffer em
todos.

16.4 Eventos e EventEmitter

Como descrito, todas as APIs do Node são assíncronas por padrão.Para
Muitos deles, essa assincronia assume a forma de erro de dois argumentos-
Primeiros retornos de chamada que são invocados quando a operação solicitada é
completo.Mas algumas das APIs mais complicadas são baseadas em eventos
em vez de.Este é normalmente o caso quando a API é projetada em torno de um
objeto em vez de uma função, ou quando uma função de retorno de chamada precisa ser
invocado várias vezes, ou quando houver vários tipos de retorno de chamada
funções 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
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 é
ouvindo mais.
No nó, objetos que emitem eventos são casos de EventEmitter ou um
Subclasse do EventEmitter:
const EventEmitter = requer ("eventos");// O nome do módulo faz
NOM DO MAIXO CLASSE NOME
const net = requer ("net");
Deixe servidor = new net.Server ();// Crie um servidor
objeto
Instância do servidor de EventEmitter // => true: servidores
são os EventEmitters
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 de
o 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
Argumentos que você deve ser aprovado:
const net = requer ("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
Use o manipulador para agendar esse cálculo de forma assíncrona usando
setTimeout () (consulte §11.10).O nó também define
SetImediate (), 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 o
Funções de manipulador de eventos registradas a serem invocadas.Isso é útil se você
estão 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
invocado 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
a 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 ele
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 outros
formas de streaming de E/S, elas são vulneráveis ??a imprevisíveis
Erros 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 evento
API, você deve ter o hábito de registrar um manipulador para eventos de "erro".
Os eventos de "erro" recebem tratamento especial da classe EventEmitter.Se
emit () é chamado para emitir um evento de "erro" e se não houver manipuladores
Registrado para esse tipo de evento, uma exceção será lançada.Desde
Isso ocorre de forma assíncrona, não há como você lidar com o
Exceção em um bloco de captura, então esse tipo de erro normalmente causa o seu
programa para sair.
16.5 fluxos
Ao implementar um algoritmo para processar dados, é quase sempre
mais fácil de ler todos os dados na memória, fazer o processamento e depois
Escreva os dados.Por exemplo, você pode escrever uma função de nó para
Copie 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);
```

```
}
});
}
```

Esta função copyfile () usa funções assíncronas e
retornos de chamada, para que não bloqueie e seja adequado para uso em simultâneo
programas como servidores.Mas observe que deve alocar memória suficiente
Para 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 disso
A implementação copyfile () é que ele não pode começar a escrever o novo
Arquivo até terminar de ler o arquivo antigo.
A solução para esses problemas é usar algoritmos de streaming onde
Os dados ?fluem? para o seu programa são processados ??e depois fluem para fora de
seu programa.A idéia é que seu algoritmo processe os dados
Pequenos 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 e
O 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 dos
Programas de nós que você escreve.Veremos uma versão de streaming do
CopyFile () Função em "Modo de fluxo".
O nó suporta quatro tipos básicos de fluxo:
Legível
Os fluxos legíveis são fontes de dados.O fluxo retornou por

fs.cretereadstream (), 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.
Gravável
Fluxos graváveis ??são sumidouros ou destinos para dados.O valor de retorno
de fs.createwritestream (), por exemplo, é uma gravidade
Stream: 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
O fluxo de transformação se torna legível - geralmente em alguns transformados
forma - do mesmo fluxo.O zlib.creategzip ()
função, por exemplo, retorna um fluxo de transformação que comprime
(com o algoritmo GZIP) os dados gravados para ele.De maneira semelhante, o
Função Crypto.createcipheriv () retorna uma transformação
Transmita que criptografa ou descriptograça 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 o
A codificação padrão do buffer ou qualquer codificação que você especificar.Nó
A API de stream também suporta um "modo de objeto" em que os fluxos leem e
Escreva objetos mais complexos do que buffers e cordas.Nenhum dos nó
As APIs principais usam este modo de objeto, mas você pode encontrá -lo em outros
Bibliotecas.
Fluxos legíveis precisam ler seus dados de algum lugar e
Fluxos graváveis ??precisam escrever seus dados em algum lugar, então todo
o fluxo tem duas extremidades: uma entrada e uma saída ou uma fonte e um
destino.O complicado das APIs baseadas em fluxo é que os dois
As extremidades do fluxo quase sempre fluem em velocidades diferentes.Talvez
O código que lê de um fluxo deseja ler e processar dados mais
Rapidamente do que os dados está realmente sendo escrito no fluxo.Ou o
reverso: talvez os dados sejam gravados para um fluxo mais rapidamente do que pode ser
Leia e saiu do riacho da outra extremidade.Fluxo
As implementações quase sempre incluem um buffer interno para manter dados
Isso foi escrito, mas ainda não lido.Buffers ajuda a garantir que
há dados disponíveis para ler quando solicitados e que há
espaço para manter dados quando estiver escrito.Mas nenhuma dessas coisas pode
sempre garantido, e é a natureza da programação baseada em fluxo
que os leitores às vezes terão que esperar que os dados sejam escritos (porque
o buffer de fluxo está vazio), e os escritores às vezes precisam esperar por
dados 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 não
Retorne até que os dados cheguem no fluxo e uma chamada para escrever blocos de dados
até 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 ();
writitable.close ();
retorno de chamada (err);
}
// Definir o tubo e lidar com a terminação normal
caso
legível
.On ("Erro", HandleArror)
.pipe (gravável)
.On ("Erro", HandleArror)
.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
deixe -se fonte = fs.createArtStream (nome do arquivo);
Let Destination = Fs.CreatewritSTream (nome do arquivo + ".gz");
deixe gzipper = zlib.creategzip ();
```

```
// Configure o pipeline
fonte
.on ("Erro", retorno de chamada) // Ligue para o retorno de chamada em leitura
erro
.pipe (gzipper)
.pipe (destino)
.on ("Erro", retorno de chamada) // Ligue para o retorno de chamada na gravação
erro
.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 o
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 buffers
this.pattern = padrão;// o regular
expressão que queremos combinar
this.incompleTeline = "";// 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 um
buffer "));
retornar;
}
// Adicione o pedaço a qualquer linha anteriormente incompleta
e quebrar
// tudo em linhas
deixe linhas = (this.incompleteLeline +
chunk) .split ("\ n");
// O último elemento da matriz é o novo
linha incompleta
this.incompleTeline = lines.pop ();
// Encontre todas as linhas correspondentes
Deixe a saída = linhas // comece com
Todas as linhas completas,
.filter (l => this.pattern.test (l)) // filtre -os
para partidas,
.Join ("\ n");// e junte -se
eles de volta.
// Se alguma coisa corresponde, adicione uma nova linha final
if (output) {
saída += "\ n";
}
// sempre ligue para o retorno de chamada, mesmo que não haja
saída
retorno 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 ela
partidas
// Passe para o retorno de chamada
if (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 com
entrada padrão,
.setEncoding ("utf8") // leia -o como
Strings Unicode,
.pipe (novo Grepstream (padrão)) // Pipa para o nosso
Grepstream,
.pipe (process.stdout) // e pico que
para padrão.
.on ("erro", () => process.exit ());// Saia graciosamente
Se o stdout fechar.
```

16.5.2 iteração assíncrona

No nó 12 e posterior, fluxos legíveis são iteradores assíncronos, o que significa que dentro de uma função assíncrona você pode usar um para/aguarda loop para ler string ou buffer pedaços de um fluxo usando Código estruturado como código síncrono seria.(Veja §13.4 para mais 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 pedaço Você lê de alguma forma.Veja como poderíamos reescrever o programa Grep na seção anterior usando uma função assíncrona e um para/aguardar laço:

```
// leia linhas de texto do fluxo de origem e escreva qualquer
linhas
// que corresponde ao padrão especificado com o destino
```

```
fluxo.
Função assíncreada grep (fonte, destino, padrão,
coding = "utf8") {
// Configure o fluxo de origem para ler strings, não
Buffers
fonte.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
deixe 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 => {// manipula
exceções assíncronas.
console.error (err);
process.Exit ();
});
16.5.3 Escrevendo para fluxos e manuseio
Backpressure
A 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 gravador
fluxo simplesmente passando -o para o método write ().O write ()
O método leva um buffer ou string como o primeiro argumento.(Fluxos de objetos
Espere outros tipos de objetos, mas estão além do escopo deste capítulo.)
Se você passar por um buffer, os bytes desse buffer serão escritos diretamente.Se
você passa uma corda, ela será codificada para um buffer de bytes antes de ser
escrito.Fluxos graváveis ??têm uma codificação padrão que é usada quando
Você passa uma string como o único argumento a escrever ().O padrão
A codificação é tipicamente "UTF8", mas você pode defini -lo explicitamente ligando
setDefaultEncoding () no fluxo gravável.Alternativamente,
Quando você passa uma string como o primeiro argumento a escrever (), você pode passar
um 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 for
mais tempo no buffer interno do fluxo gravável.(Este retorno de chamada também pode
ser chamado se ocorrer um erro, mas isso não for garantido.Você deve
Registre 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
O primeiro lugar 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
deixe hasmoreroom = stream.write (chunk);
// Verifique o valor de retorno do método write ():
if (hasMoreroom) {// se ele retornar
É verdade que
SetImediate (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 um
linha e às vezes você tem que esperar por um evento entre as gravações
cria algoritmos desajeitados.Esta é uma das razões que usam
O método Pipe () é tão atraente: quando você usa Pipe (), Nó
Lida com a contrapressão para você automaticamente.
Se você está usando aguardar e assíncrono em seu programa, e está tratando
Fluxos legíveis como iteradores assíncronos, é direto
Implementar 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 fonte
transmita para um fluxo de destino e cópia de chamada (fonte,
destino) é como ligar
fonte.pipe (destino):
// Esta função escreve o pedaço especificado para o especificado
stream e
// retorna uma promessa que será cumprida quando estiver bom
Escreva novamente.
// Como retorna uma promessa, pode ser usado com aguardar.
função write (stream, chunk) {
// Escreva o pedaço especificado para o fluxo especificado
deixe hasmoreroom = stream.write (chunk);
if (hasMoreroom) {// se o buffer for
Não está cheio, retorne
Return Promise.Resolve (NULL);// já
Objeto 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, caso
padrão
// 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 gravidade O fluxo 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 lidar Backpressure 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
ataque 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
empurre pedaços de dados (buffers ou cordas) para você assim que eles se tornarem
disponível.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
Ligue para pausa () no fluxo legível para interromper temporariamente os dados
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
o 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 uns não transportado
CopyFile () função e prometeu uma versão melhor que está por vir.O
A 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 = requer ("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.createArtStream (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
o fluxo está cheio
input.Pause ();// então pausa o
fluxo de entrada.
}
});
input.on ("end", () => {// quando chegarmos
o fim da entrada,
output.end ();// Diga a 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 cheio,
input.Resume ();// retomar dados
Eventos 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 escrito
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 pausado

O outro modo para fluxos legíveis é "modo pausado".Este é o
Modo 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 em
modo pausado.No modo pausado, o fluxo não empurra dados para você em
a forma de eventos de "dados".Em vez disso, você puxa dados do fluxo por
chamando explicitamente o método read ().Esta não é uma chamada de bloqueio e se
Não há dados disponíveis para leitura no fluxo, eles retornarão NULL.
Como não há uma API síncrona para esperar pelos dados, o modo pausado
A API também é baseada em eventos.Um fluxo legível no modo pausado emite
Eventos "legíveis" quando os dados ficam disponíveis para ler no fluxo.
Em resposta, seu código deve chamar o método read () para ler que
dados.Você deve fazer isso em um loop, chamando read () repetidamente até
retorna nulo.É necessário drenar completamente o buffer do fluxo
Assim para acionar um novo evento "legível" no futuro.Se você
pare de chamar read () Embora ainda haja dados legíveis, você não receberá
Outro evento ?legível? e seu programa provavelmente pendurarão.
Fluxos no modo pausado emitem eventos "end" e "error", assim como
fluxos de modo de fluxo fazem.Se você está escrevendo um programa que lê dados
De 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
falso, e depois começando a ler ou escrever novamente em um legível ou
Evento de drenagem.Isso é deselegante, 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 sha256
hash 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 nó 12 e
mais tarde, seria mais simples escrever esta função usando um para/aguardar
laço.)

```
const fs = requer ("fs");
const cripto = requer ("cripto");
// Calcule um hash sha256 do conteúdo do arquivo nomeado
e passar o
// hash (como uma string) para o retorno de chamada do primeiro erro especificado
função.
função sha256 (nome do arquivo, retorno de chamada) {
Deixe input = fs.createArtStream (nome do arquivo);// os dados
fluxo.
deixe hasher = cripto.createhash ("sha256");// Para
calculando o hash.
input.on ("legível", () => {// quando houver
dados prontos para ler
Deixe Chunk;
while (chunk = input.read ()) {// leia um pedaço, e
se não nulo,
hashher.update (pedaço);// passa para o
Hasher,
```

```
} // e continue em loop
até 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
console.error (err.toString ());// imprima -o como um
erro.
} else {// caso contrário,
console.log (hash);// Imprima o hash
corda.
}
});
```

16.6 Processo, CPU e sistema operacional

Detalhes

O objeto de processo global possui várias propriedades úteis e funções que geralmente se relacionam com o estado do nó atualmente em execução processo.Consulte a documentação do nó para obter detalhes completos, mas aqui são 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 atual
diretório.
process.chdir () // define o funcionamento atual
diretório.
process.cpuusage () // relata o uso da CPU.
process.env // um objeto de ambiente
variáveis.
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 relatado
Quando o programa sai.
Process.GeTuid () // Retornar o ID do usuário do UNIX do
usuário atual.
process.hrtime.bigint () // retorna uma "alta resolução"
Nanossegund Timestamp.
process.kill () // Envie um sinal para outro processo.
process.memoryusage () // retorna um objeto com uso de memória
detalhes.
process.NextTick () // como setImediate (), invocar um
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 recurso
Detalhes de uso.
process.setUid () // define o usuário atual, por id ou
nome.
process.Título // o nome do processo que aparece em
`PS` listagens.
process.umask () // defina ou retorne o padrão
Permissões 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 bibliotecas
O nó depende.
O módulo "OS" (que, diferentemente do processo, precisa ser explicitamente

Carregado com requer ()) fornece acesso a nível semelhante
Detalhes sobre o computador e o sistema operacional que o nó está em execução
sobre.Você pode nunca precisar usar nenhum desses recursos, mas vale a pena
Saber que o nó os disponibiliza:
const os = requer ("os");
os.arch () // retorna arquitetura da CPU."x64" ou
"Arm", por exemplo.
OS.CONSTANTS // Constantes úteis, como
OS.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 em
bytes.
OS.GetPriority () // Retorna a prioridade de agendamento do sistema operacional
de um processo.
os.homedir () // retorna a casa do usuário atual
diretório.
os.hostname () // retorna o nome do host do
computador.
os.loadavg () // retorna os 1, 5 e 15 minutos
Médias de carga.
OS.NetworkInterfaces () // Retorna detalhes sobre o disponível
rede.conexões.
os.platform () // retorna os: "linux", "darwin" ou
"Win32", por exemplo.
os.release () // retorna o número da versão do
OS.
os.setPriority () // tenta definir o agendamento
prioridade para um processo.
os.tmpdir () // retorna o padrão temporário
diretório.
OS.Totalmem () // retorna a quantidade total de RAM em
bytes.
os.type () // retorna os: "linux", "darwin" ou
"Windows_nt", por exemplo

Erro ao traduzir esta página.

descritor ?como o primeiro argumento em vez de um caminho.Essas variantes têm
Nomes que começam com a letra "f".Por exemplo, fs.truncato ()
truques um arquivo especificado por caminho, e fs.ftruncate () truncam um
arquivo especificado pelo descritor de arquivo.Há uma promessa baseada
fs.promesis.truncate () que espera um caminho e outro
Versão baseada em promessa que é implementada como um método de um
Objeto FileHandle.(A classe FileHandle é o equivalente a um arquivo
descritor na API baseada em promessa.) Finalmente, há um punhado de
funções no módulo "FS" que têm variantes cujos nomes são
prefixado com a letra "l".Essas variantes "L" são como a função base
mas não siga links simbólicos no sistema de arquivos e opere
diretamente nos próprios ligações simbólicas.

16.7.1 Caminhos, descritores de arquivos e trabalhos de arquivo

Para usar o módulo "FS" para trabalhar com arquivos, você primeiro precisa ser
capaz de nomear o arquivo com o qual você deseja trabalhar.Os arquivos são mais frequentemente
especificado por caminho, o que significa o nome do próprio arquivo, além do
Hierarquia de diretórios nos quais o arquivo aparece.Se um caminho é absoluto,
Isso significa que os diretórios até a raiz do sistema de arquivos estão
especificado.Caso contrário, o caminho é relativo e só é significativo em
relação com algum outro caminho, geralmente o diretório de trabalho atual.
Trabalhar com caminhos pode ser um pouco complicado, porque a operação diferente
Os sistemas usam caracteres diferentes para separar nomes de diretórios, é fácil
para dobrar acidentalmente esses caracteres separadores ao concatenar
caminhos e porque ../ segmentos de caminho do diretório pai precisam de especial
manuseio.Módulo "Path" do Node e alguns outros nó importantes
Ajuda dos recursos:

```
// Alguns caminhos importantes
process.cwd () // caminho absoluto do trabalho atual
diretório.
__filename // caminho absoluto do arquivo que mantém
o código atual.
__dirname // caminho absoluto do diretório que
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": tiras "./"
segmentos
path.Normalize ("// a // b //") // => "/a/b/": remove
duplicado /
// junção () combina segmentos de caminho, adicionando separadores e depois
normaliza
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 string
função que não tem acesso ao sistema de arquivos real.O
Funções Fs.RealPath () e F.RealPathSync () executam
Canonicalização consciente do sistema de arquivos: eles resolvem links simbólicos e
Interprete os nomes relativos de caminho em relação ao diretório de trabalho atual.
Nos exemplos anteriores, assumimos que o código está em execução em um
OS 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.Posix
em vez de caminho.E inversamente, se você quiser trabalhar com o Windows
Caminhos mesmo quando em um sistema Unix, Path.Win32.Path.Posix e
Path.win32 Defina as mesmas propriedades e funções que o próprio caminho.
Algumas das funções "fs" que abordaremos nas próximas seções
Espere um descritor de arquivo em vez de um nome de arquivo.Os descritores de arquivos são
Os números inteiros usados ??como referências no nível do SO aos arquivos "abertos".Você obtém um
descritor para um determinado nome chamando o fs.open () (ou
função fs.opensync ()).Os processos só podem ter um
Número limitado de arquivos abertos ao mesmo tempo, por isso é importante que você ligue
fs.close () nos descritores de arquivos quando você terminar com eles.
Você precisa abrir arquivos se quiser usar o nível mais baixo
Funções fs.read () e fs.write () que permitem que você pule
Em 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 sentido
Para 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 (), que
Retorna uma promessa que resolve para um objeto FileHandle.Este arquivo de arquivo
O 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 se
Você cria um arquivo de arquivo, lembre -se de ligar para o seu fechamento ()
método depois de terminar com isso.
16.7.2 Leitura de arquivos
O nó permite ler o conteúdo do arquivo de uma só vez, através de um fluxo ou com
A API de baixo nível.
Se seus arquivos forem pequenos ou se o uso e o desempenho da memória não forem o
maior prioridade, então é muitas vezes mais fácil ler todo o conteúdo de um
Arquivo 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 um
Buffer, mas se você especificar uma codificação, receberá uma corda decodificada
em vez de.

```
const fs = requer ("fs");
deixe buffer = fs.readfilesync ("test.data");//
Síncrono, retorna buffer
deixe texto = fs.readfilesync ("data.csv", "utf8");//
Síncrono, Retorna String
// Leia os bytes do arquivo de forma assíncrona
fs.readfile ("test.data", (err, buffer) => {
if (err) {
```

// lide com o erro aqui
} outro {
// Os bytes do arquivo estão em buffer
}
});
// Leia assíncrona baseada em promessa
fs.promises
.readfile ("data.csv", "utf8")
.TENHEN (ProcessfileText)
.catch (Handlereaderror);
// ou use a API de promessa com aguardar dentro de uma função assíncrona
Função 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 um
stream e o método Pipe () para escrever o conteúdo de um arquivo para
saída padrão:
função printfile (nome do arquivo, coding = "utf8") {
fs.cretereadstream (nome do arquivo,
codificação) .pipe (process.stdout);
}
Finalmente, se você precisar de controle de baixo nível sobre exatamente quais bytes você lê
De um arquivo e quando você os lê, você pode abrir um arquivo para obter um arquivo
descritor e depois use fs.read (), fs.readsync () ou
fs.promeses.read () para ler um número especificado de bytes de um

Localização da fonte especificada do arquivo em um buffer especificado no
Posição de destino especificada:

```
const fs = requer ("fs");
// lendo uma parte específica de um arquivo de dados
fs.open ("dados", (err, fd) => {
if (err) {
// Relatório Erro de alguma forma
retornar;
}
tentar {
// leia bytes 20 a 420 em um recém -alocado
buffer.
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 lidos
em.
});
}
finalmente {// use uma cláusula finalmente
fs.close (FD);// Fechar o descritor de arquivo aberto
}
});
```

A API read () baseada em retorno de chamada é estranha de usar se você precisar ler
Mais de um pedaço de dados de um arquivo.Se você pode usar o
API síncrona (ou a API baseada em promessa com aguardar), torna-se
Fácil 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! == 0xdadafeed) {
lançar um novo erro ("o arquivo é do tipo errado");
}
// Agora obtenha o deslocamento e o comprimento dos dados do
cabeçalho
soltar offset = header.readInT32LE (4);
deixe comprimento = cabeçalho.readInT32LE (8);
// e leia esses bytes do arquivo
deixe 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 um
codificação diferente de "utf8", passe a codificação como um terceiro opcional
argumento.
As funções relacionadas fs.appendfile (),
fs.appendfilesync () e fs.promises.appendfile ()
são semelhantes, mas quando o arquivo especificado já existe, eles anexam seus
dados 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 for
Nem todos na memória ao mesmo tempo, então usar um fluxo gravável é um
boa abordagem, assumindo que você planeja escrever os dados de
Começando a terminar sem pular no arquivo:
const fs = requer ("fs");
deixe output = fs.createwritStream ("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 você
deseja poder controlar a posição exata dentro do arquivo em que
Cada pedaço é escrito, então você pode abrir o arquivo com fs.open (),
fs.opensync () ou fs.promises.open () e depois use o
Descritor de arquivo resultante com o fs.write () ou

funções fs.WriteSync ().Essas funções vêm em diferentes
formas para cordas e buffers.A variante da string leva um descritor de arquivo,
uma string e a posição do arquivo para escrever essa string (com um
codificação como um quarto argumento opcional).A variante de buffer leva um
Descritor 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
Escreva, você pode fazer isso com um único fs.writev () ou
fs.WriteVsync ().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
Para escrever um arquivo, no entanto, você também deve especificar um segundo argumento de string que
especifica como você pretende
Para usar o descritor 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 leitura
Se você não passa uma dessas seqüências de bandeira para fs.open () ou fs.opensync (), eles usam o padrão
Sinalizador ?R?, tornando o descritor de arquivo somente leitura.Observe que também pode ser útil passar essas bandeiras para outro
Métodos de redação de arquivos:

```
// escreva em um arquivo em uma chamada, mas anexar qualquer coisa que já esteja
lá.
// 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 acidentalmente 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 ().Esses
As funções seguem um caminho como seu primeiro argumento e um comprimento como seu segundo e modifique o arquivo para que ele tenha o comprimento especificado.Se você
Omita 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 você
Especifique um comprimento maior que o tamanho atual do arquivo, o arquivo é
estendido com zero bytes ao novo tamanho.Se você já abriu
O 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 bobes
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 do
função copyfile.
// Dois sinalizadores são combinados com o bit a bit ou o OPEARTOR |.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 $ {novo
Date (). ToisSotring ()} "
fs.constants.copyfile_excl |
fs.constants.copyfile_ficlone)
.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 comportar
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:
fs.unlinksync ("backups/ch15.bak");
16.7.5 Metadados do arquivo
O fs.stat (), fs.statsync () e fs.promises.stat ()
As funções permitem obter metadados para um arquivo ou diretório especificado.
Por exemplo:
const fs = requer ("fs");
Let Stats = fs.statsync ("livro/CH15.md");
stats.isfile () // => true: este é um arquivo comum
estats.isdirectory () // => false: não é um diretório
STATS.Size // Tamanho do arquivo em bytes
stats.ATIME // Tempo de acesso: data quando foi o último
ler
stats.mtime // Tempo de modificação: data quando foi
último escrito
stats.uid // O ID do usuário do proprietário do arquivo
stats.gid // O ID do grupo do proprietário do arquivo
estats.mode.toString (8) // As permissões do arquivo, como um octal
corda

O objeto de estatísticas retornadas contém outras propriedades mais obscuras e
métodos, mas este código demonstra aqueles que você provavelmente deve
usar.
fs.lstat () e suas variantes funcionam como fs.stat (), exceto que
Se o arquivo especificado for um link simbólico, o nó retornará metadados para o
Link em si em vez de seguir o link.
Se você abriu um arquivo para produzir um descritor de arquivo ou um arquivo de arquivo
Objeto, então você pode usar fs.fstat () ou suas variantes para obter metadados
informações para o arquivo aberto sem ter que especificar o nome do arquivo
de novo.
Além de consultar metadados com fs.stat () e todos os seus
Variantes, também existem funções para a mudança de metadados.
fs.chmod (), fs.lchmod () e fs.fchmod () (junto com
versões síncronas e baseadas em promessas) defina o "modo" ou
permissões de um arquivo ou diretório.Os valores de modo são inteiros nos quais
Cada bit tem um significado específico e é mais fácil de pensar em octal
notação.Por exemplo, para fazer um arquivo somente leitura para seu proprietário e
Inacessível a todos os outros, use 0o400:
fs.chmodsync ("CH15.MD", 0O400);// Não exclua
acidentalmente!
fs.chown (), fs.lchown () e fs.fchown () (junto com
versões síncronas e baseadas em promessas) defina o proprietário e o grupo (como
IDs) para um arquivo ou diretório.(Estes são importantes porque eles interagem com o
Permissões 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.mkdirsync (), ou fs.promises.mkdir ().O primeiro
O argumento é o caminho do diretório a ser criado.O segundo opcional
O argumento 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
A função criará qualquer diretório no caminho que ainda não existe:

```
// Verifique se existem dist/ e dist/ lib/ ambos.
fs.mkdirsync ("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 terminar
fs.rmdirsync (tempdirpath);
}
O módulo "FS" fornece duas APIs distintas para listar o conteúdo de um
diretório.Primeiro, fs.readdir (), fs.readdirsync () e
fs.promises.readdir () Leia o diretório inteiro de uma só vez e
Dê a você uma variedade de cordas ou uma variedade de objetos diretos que especificam
os nomes e tipos (arquivo ou diretório) de cada item.Nomes de arquivos retornados
Por 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 subdiretos
fs.promises.readdir ("/tmp", {withfiletypes: true})
.then (entradas => {
entres.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 de
fs.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
subdiretos
}
Deixe estatísticas = aguarda fs.promises.stat (path.join (dirpath,
nome));
deixe tamanho = estatismo.size;
console.log (string (tamanho) .padstart (10), nome);
}
}
```

16.8 clientes e servidores HTTP
Os módulos "http", https "e" http2 "são completos, mas
implementações de nível relativamente baixo dos protocolos HTTP.Eles
Defina 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
Demonstre 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
começou a chegar.Quando o retorno de chamada é chamado, o status HTTP e
Os cabeçalhos estão disponíveis, mas o corpo ainda não está pronto.O
O objeto de entrada de Message é um fluxo legível e você pode usar o
técnicas demonstradas anteriormente neste capítulo para ler a resposta
corpo dele.
A função getjson () no final do §13.2.6 usou o
http.get () função como parte de uma demonstração da promessa ()
construtor.Agora que você sabe sobre fluxos de nó e o nó
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
o mais geral http.request () e https.request ()
funções.A função Postjson () a seguir demonstra como fazer
use https.request () para fazer uma solicitação de post https que
Inclui um corpo de solicitação JSON.Como a função getjson ()
Capítulo 13, espera uma resposta JSON e retorna uma promessa de que
atende à versão analisada dessa resposta:

```
const https = requer ("https");
/*
* Converta o objeto corporal em uma corda JSON e depois https post
para o
* Ponto de extremidade da API especificado no host especificado.Quando o
A resposta chega,
* analise o corpo de resposta como JSON e resolva o retorno
Promessa 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 resolução
ou rejeitar
// Quando a solicitação HTTPS é bem -sucedida ou falha.
retornar nova promessa ((resolver, rejeitar) => {
// converte o objeto do corpo em uma string
Deixe BodyText = JSON.Stringify (Body);
// Configure a solicitação HTTPS
Deixe requestOptions = {
Método: "post", // ou "get", "put",
"Delete", etc.
host: host, // o host para se conectar
caminho: endpoint, // o caminho da URL
cabeçalhos: {// cabeçalhos http para o
solicitar
"Tipo de conteúdo": "Aplicativo/JSON",
"Length-thength": buffer.bytelength (BodyText)
}
};
se (porta) {// se uma porta for
especificado,
requestOptions.port = porta;// use -o para o
solicitar.
}
// Se as credenciais forem especificadas, adicione uma autorização
cabeçalho.
if (nome de usuário && senha) {
requestOptions.auth = `$ {nome de usuário}: $ {senha}`;
}
```

```
// agora crie a solicitação com base na configuração
objeto
deixe solicitação = https.request (requestOptions);
// Escreva o corpo da solicitação de postagem e termine o
solicitar.
request.Write (BodyText);
request.end ();
// falha nos erros de solicitação (como nenhuma rede
conexão)
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 um
Buffer 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
o
// 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 o
A resposta está feita,
tente {// tente
analisá -lo como JSON
resolve (json.parse (corpo));// e
Resolva o resultado.
} catch (e) {// ou, se
Tudo dá errado,
rejeitar (e);// rejeitar
com o erro
}
});
});
});
}
```

Além de fazer solicitações HTTP e HTTPS, o "http" e
Os módulos "https" também permitem escrever servidores que respondem a eles
solicitações.A abordagem básica é a seguinte:

Crie um novo objeto de servidor.
Ligue para o método Listen () para começar a ouvir solicitações em
uma porta especificada.
Registre um manipulador de eventos para eventos de "solicitação", use que
manipulador para ler a solicitação do cliente (particularmente o
Propriedade do request.url) e escreva sua resposta.

O código a seguir cria um servidor HTTP simples que serve estático
arquivos do sistema de arquivos local e também implementa uma depuração
endpoint que responde à solicitação de um cliente ecoando essa solicitação.

```
// Este é um servidor http estático simples que serve arquivos de
um especificado
// diretório.Ele também implementa um especial /teste /espelho
```

```
endpoint isto
// ecoa a solicitação de entrada, que pode ser útil quando
depurar clientes.
const http = requer ("http");// Use "https" se você tiver um
Certificado
const url = requer ("url");// para analisar URLs
const caminho = requer ("caminho");// para manipular
Caminhos do sistema de arquivos
const fs = requer ("fs");// Para leitura de arquivos
// serve arquivos do diretório raiz especificado por meio de um http
servidor isso
// escuta na porta especificada.
função servir (rootdirectory, porta) {
Deixe servidor = novo http.server ();// Crie um novo http
servidor
Server.Listen (porta);// Ouça no
porta especificada
console.log ("escuta na porta", porta);
// Quando os pedidos chegarem, lide com esta função
Server.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/espelho", envie de volta
o pedido
// literalmente.Útil quando você precisa ver o pedido
Cabeçalhos e corpo.
if (endpoint === "/teste/espelho") {
// Definir cabeçalho de resposta
Response.setheader ("Content-Type", "Text/Plain;
charset = utf-8 ");
// Especifique o código de status da resposta
Response.Writehead (200);// 200 ok
// Comece o corpo de resposta com o pedido
Response.Write (`$ {request.method} $ {request.url}
Http/$ {
request.httpversion
} \ r \ n`);
```

Erro ao traduzir esta página.

```
deixe Stream = fs.cretereadstream (nome do arquivo);
stream.once ("readável", () => {
// Se o fluxo ficar legível, então defina
o
// Content-Type Cabeçalho e um status de 200 OK.
Em seguida, pague o
// Stream do leitor de arquivos para a resposta.O
Will de tubo
// Ligue automaticamente a resposta.end () quando o
Terminos 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 abrir
o 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 ()
função
servir (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 trivial Os servidores são implementados usando bibliotecas externas - como o expresso

estrutura-que fornecem ?middleware? e outros utilitários de nível superior
Os desenvolvedores da Web de back -end esperavam.

16.9 Servidores de rede não-HTTP e Clientes

Servidores e clientes da web se tornaram tão onipresentes que é fácil
esqueça que é possível escrever clientes e servidores que não usam
Http.Mesmo que o nó tenha uma reputação como um bom ambiente para
Escrevendo 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 duplex
fluxo.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 evento
O 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-knock
na 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 conexões
Deixe o servidor = net.createServer ();
Server.Listen (6789, () => console.log ("Entregando risadas em
porta 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 que
ocorrer,
soquete.end ();// mas ainda fechar o
Socket!
});
});
// 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 poderia
Yodel! "
};
// realiza interativamente uma piada de knock sobre este soquete,
sem bloquear.
função assíncrona Telljoke (soquete) {
// Escolha uma das piadas aleatoriamente
Let 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.Write (`$ {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 diferente
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 e
vá 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 estágio
1, então
// entregar a linha de soco e retornar desde
A 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 personalizado cliente.Se o utilitário NC ("NetCat") estiver instalado no seu sistema, você pode Use -o para se comunicar com este servidor da seguinte forma:

```
$ nc localhost 6789
Knock!
>> 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 em Nó.Acabamos de nos conectar ao servidor e, em seguida, transmitir a saída do servidor para Stdout e Pipe Stdin para a entrada do servidor:

```
// Conecte -se à porta de piada (6789) no servidor nomeado no
linha de comando
Deixe soquete = requer ("net"). CreateConnection (6789,
process.argv [2]);
Socket.pipe (process.stdout);// Dados de tubulação de
o soquete para stdout
process.stdin.pipe (soquete);// Dados de tubulação de
```

stdin para o soquete
Socket.on ("Close", () => process.exit ());// desistir quando o
O soquete fecha.
Além de suportar servidores baseados em TCP, o módulo "Net" do Node também
suporta a comunicação interprocessante sobre "soquetes de domínio unix" que
são identificados por um caminho do sistema de arquivos e não por um número da porta.Nós somos
não vai cobrir esse tipo de soquete neste capítulo, mas o nó
A documentação tem detalhes.Outros recursos do nó que não temos
Espaço a cobrir aqui inclui o módulo "dgram" para clientes baseados em UDP
e servidores e o módulo "TLS" que é "net" como "https" é "http".
As classes TLS.Server e Tls.tlSSocket permitem a criação
de servidores TCP (como o servidor de piada de knock-knock) que usam SSL-
Conexões criptografadas como os servidores HTTPS.
16.10 Trabalhando com processos filhos
Além de escrever servidores altamente simultâneos, o Node também funciona bem
para escrever scripts que executam outros programas.No nó
O módulo ?Child_Process? define uma série de funções para executar
Outros programas como processos infantis.Esta seção demonstra alguns dos
essas funções, começando com o mais simples e se movendo para o mais
complicado.
16.10.1 Execsync () e ExecFilesync ()
A maneira mais fácil de executar outro programa é com
Child_process.execsync ().Esta função leva o comando
para correr como seu primeiro argumento.Ele cria um processo infantil, executa uma concha nessa
Processar 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
Especifique uma codificação em um segundo argumento opcional para obter uma string
em vez de.Se o comando gravar qualquer saída para Stderr, essa saída apenas
é passado para o fluxo Stderr do processo pai.
Por exemplo, se você está escrevendo um script e o desempenho não é um
preocupação, você pode usar child_process.execsync () para listar um
diretório com um comando familiar de shell unix, em vez de usar o
função fs.readdirsync ():
const Child_process = requer ("Child_Process");
Seja listing = child_process.execsync ("ls -l web/*. html",
{codificação: "utf8"});
O fato de o Execsync () invocar um shell completo do Unix significa que o
String que você passa para ela pode incluir vários semicolon-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
Tenha cuidado para nunca passar um comando para execsSync () se alguma parte de
Esse comando é entrada do usuário ou vem de uma fonte não confiável semelhante.
A sintaxe complexa dos comandos do shell pode ser facilmente subvertida a
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 de
Argumentos da linha de comando como o segundo argumento:
Seja listing = Child_process.execfilesync ("LS", ["-l",
"Web/"],
{codificação: "utf8"});
Opções de processo da criança
Execsync () e muitas das outras funções Child_process têm um segundo ou terceiro opcional
Argumento que especifica detalhes adicionais sobre como o processo infantil deve ser executado.A codificação
A propriedade deste objeto foi usada anteriormente para especificar que gostaríamos que a saída do comando fosse entregue
como uma string e não como um buffer.Outras propriedades importantes que você pode especificar incluem o
A 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 criança
Processo herda o valor do processo.cwd ().
ENV especifica as variáveis ??de ambiente às quais o processo filho terá acesso.Por
Padrão, 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 para o processo infantil.Esta opção está disponível apenas para as funções síncronas que não
devolver um objeto de processo infantil.
MaxBuffer especifica o número máximo de bytes de saída que serão coletados pelo
funções executivas.(Não se aplica a Spawn () e Fork (), que usam fluxos.) Se uma criança
O 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 que
Normalmente executa um comando Shell, esta opção permite especificar qual shell usar.Para
Funções que normalmente não usam um shell, essa opção permite especificar que um shell
deve 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 ser
permitido correr.Se não tivesse saído antes desse tempo, será morto, será morto e sairá com
um 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 pai
O processo está em execução em uma conta privilegiada, ele pode usar esta opção para executar a criança com redução
privilégios.
16.10.2 EXEC () e EXECFILE ()
As funções Execsync () e ExecFileSync () são, como seus

Nomes indicam, síncrono: eles bloqueiam e não retornam até
Saídas do processo infantil.Usar essas funções é como digitar Unix
comandos em uma janela de terminal: eles permitem que você execute uma sequência de
comanda um de cada vez.Mas se você está escrevendo um programa que precisa
realizar várias tarefas, e essas tarefas não dependem de cada
Outro de qualquer maneira, então você pode paralizá -los e correr
Vários comandos ao mesmo tempo.Você pode fazer isso com o
Funções assíncronas child_process.exec () e
Child_process.execfile ().
exec () e execfile () são como suas variantes síncronas, exceto
que eles retornam imediatamente com um objeto de processo infantil que representa
o processo infantil em execução e eles recebem um retorno de chamada de erro como seu
argumento 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, se
qualquer;Será nulo se o processo terminou normalmente.O segundo
O argumento é a saída coletada que foi enviada para o padrão da criança
fluxo de saída.E o terceiro argumento é qualquer saída que foi enviada para o
Fluxo 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 (que
então ele pode ler a partir de sua entrada padrão).Vamos abordar o processo infantil em
Mais detalhes quando discutirmos o Child_process.spawn ()
função.
Se você planeja executar vários processos infantis ao mesmo tempo, então
pode 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 seu
Entrar e retornar uma promessa que resolve ao resultado de todos aqueles
comandos:

```
const Child_process = requer ("Child_Process");
const util = requer ("util");
const Execp = util.promisify (Child_process.exec);
função 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 do
cumprimento
// 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
assíncrono - são projetados 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 processo
Saída 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 (que
verá esses dados como entrada em seu fluxo de entrada padrão): isso significa que é
possível interagir dinamicamente com um processo infantil, enviando entrada
com base na saída que ele gera.
Spawn () não usa uma concha por padrão, então você deve invocar como
execfile () com o executável a ser executado e uma variedade separada de
Argumentos da linha de comando para passar para ele.Spawn () retorna a
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
e Stderr são fluxos legíveis: quando o processo infantil grava para o seu
stdout e seus fluxos Stderr, essa saída se torna legível através do
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 legível
fluxo 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
Use 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 = requer ("Child_Process");
// Inicie um novo processo de nó executando o código no Child.js em

nosso diretório
Deixe 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 recebê -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 precisa
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 desaperiado no processo infantil com Json.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 soquete e servidor (a partir da ?rede? módulo) para um processo infantil.Os servidores de rede tendem a ser ligados a IO, em vez do que 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 filho para lidar com solicitações.No Processo pai, você pode ouvir eventos de "conexão" em seu servidor objeto, então obtenha o objeto de soquete 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 filho 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
Sincronize 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
Comunique -se pela passagem de mensagens.O tópico principal pode enviar uma mensagem
para um tópico de trabalhador chamando o método pós -message () do
Objeto de trabalhador que representa esse tópico.O tópico do trabalhador pode
Receba 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
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
trabalhar nos vários núcleos, que se tornaram
Comum em computadores hoje.Se você está fazendo científico
computação ou aprendizado de máquina ou processamento de gráficos em
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 um
CPU, você ainda pode usar threads para manter o
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 ao
segundo pedido até que o cálculo da primeira resposta seja
completo.Em vez
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
estão escrevendo um programa que depende do código legado que é
inevitavelmente síncrono, você poderá usar os trabalhadores para
Evite 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 trabalhador
tópicos.
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.ismhrethread) {
// 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
O trabalhador está feito.
Module.Exports = Função Reticulatesplines (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);
```

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 opcional
Configuração 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 compartilhá -lo diretamente com o tópico do trabalhador.Isso impede o
Tópico do trabalhador e o thread principal do compartilhamento de memória.Você pode
Espere que essa cópia seja feita com JSON.Stringify ()
e json.parse () (§11.6).Mas, de fato, o nó empresta mais
Técnica robusta conhecida como algoritmo de clone estruturado da web
navegadores.
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 nó
ambiente, como soquetes e fluxos.Observe, no entanto, esse buffer
Os objetos são parcialmente suportados: se você passar um buffer para

PostMessage () será recebido como um Uint8Array e pode ser
convertido de volta em um buffer com buffer.from ().Leia mais
sobre o algoritmo de clone estruturado em ?O clone estruturado
Algoritmo".

16.11.2 O ambiente de execução do trabalhador

Na maioria das vezes, o código JavaScript em um thread de trabalhador de nós é executado apenas
Como faria no tópico principal do Node.Existem algumas diferenças que
você deve estar ciente, e algumas dessas diferenças envolvem
Propriedades do segundo argumento opcional para o trabalhador ()
construtor:

Como vimos, threads.ismhhread é verdadeiro no
Tópico principal, mas é sempre falso em qualquer tópico de trabalhador.
Em um tópico de trabalhador, você pode usar
threads.parentport.postMessage () para enviar um
mensagem para o tópico pai e
threads.parentport.on para registrar manipuladores de eventos para
mensagens do thread pai.No tópico principal,
threads.parentport é sempre nulo.

Em um tópico de trabalhador, threads.workerdata está definido como uma cópia
da propriedade Workerdata do segundo argumento para o
Trabalhador () construtor.No tópico principal, esta propriedade é
sempre nulo.Você pode usar esta propriedade WorkerData para
passe uma mensagem inicial para o trabalhador que estará disponível como
assim que começa para que o trabalhador não precise esperar por um
Evento de ?mensagem? antes de começar a fazer o trabalho.

Por padrão, process.env em um tópico de trabalhador é uma cópia de
process.env no thread pai.Mas o fio pai pode

Especifique um conjunto personalizado de variáveis ??de ambiente definindo o
Propriedade Env do segundo argumento para o trabalhador ()
construtor.Como um caso especial (e potencialmente perigoso), o
O tópico pai pode definir a propriedade Env para
threads.share_env, que fará com que os dois tópicos
compartilhe um único conjunto de variáveis ??de ambiente para que uma mudança em
Um tópico é visível no outro.
Por padrão, o processo do processo.stdin em um trabalhador nunca
possui dados legíveis sobre ele.Você pode alterar esse padrão por
Passando stdin: verdadeiro no segundo argumento para o
Trabalhador () construtor.Se você fizer isso, então o stdin
A propriedade do objeto trabalhador é um fluxo gravável.Quaisquer dados
que o pai escreve para trabalhador.stdin se torna legível
em process.stdin no trabalhador.
Por padrão, o processo.stdout e process.stderr
fluxos no trabalhador são simplesmente canalizados para o correspondente
fluxos no thread pai.Isso significa, por exemplo, que
console.log () e console.error () produzem saída
exatamente da mesma maneira em um fio de trabalhador que eles
Tópico principal.Você pode substituir esse padrão passando
stdout: verdadeiro ou stderr: true no segundo argumento para
O construtor trabalhador ().Se você fizer isso, então qualquer saída do
O trabalhador escreve para esses fluxos se torna legível pelo pai
Thread no trabalhador.stdout e trabalhador.stderr
tópicos.(Há uma inversão potencialmente confusa do fluxo
Instruções aqui, e vimos a mesma coisa com o filho
processos no início do capítulo: os fluxos de saída de um trabalhador
Os threads são fluxos de entrada para o fio pai e a entrada
O fluxo de um trabalhador é um fluxo de saída para o pai.)
Se um thread trabalhador chama process.exit (), apenas o tópico
sai, 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 invocado de um trabalhador.

Os sinais do sistema operacional (como SIGINT e SIGTERM) são entregue apenas no fio principal;Eles não podem ser recebidos ou tratado 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 a partir do fio pai, e o tópico pai usa o trabalhador Objeta -se a enviar e receber mensagens para e do tópico do trabalhador.

A API do Thread Worker também permite a criação de costumes 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 Os módulos poderiam compartilhar o canal padrão e enviar mensagens com trabalhador.PostMessage (), mas seria mais limpo se cada módulo tem seu próprio canal privado para enviar mensagens ao trabalhador.Ou Considere o caso em que o tópico principal cria dois independentes trabalhadores.Um canal de comunicação personalizado pode permitir que os dois trabalhadores

para se comunicar diretamente entre si, em vez de ter que enviar tudo
suas mensagens através do pai.
Crie um novo canal de mensagem com o Messagechannel ()
construtor.Um objeto Messagechannel tem duas propriedades, nomeado
PORT1 e PORT2.Essas propriedades se referem a um par de mensagens
objetos.Chamar PostMessage () em um dos portos causará um
Evento de ?mensagem? a ser gerado por outro com um clone estruturado de
o objeto de mensagem:

```
const threads = requer ("trabalhador_threads");
deixe canal = new threads.MessageChannel ();
canal.port2.on ("mensagem", console.log);// registrar qualquer um
mensagens que recebemos
canal.port1.postMessage ("hello");// vai causar
"Olá" para ser impresso
```

Você também pode ligar para o fechamento () em qualquer porta para quebrar a conexão
entre as duas portas e sinalizar que não serão mais mensagens
troca.Quando Close () é chamado em qualquer porta, um evento "fechado" é
entregue em ambas as portas.
Observe que o exemplo de código acima cria um par de objetos de Messageport
e então usa esses objetos para transmitir uma mensagem na principal
fio.Para usar canais de comunicação personalizados com trabalhadores,
Devemos transferir uma das duas portas do tópico em que está
criado para o tópico em que será usado.A próxima seção explica
Como fazer isso.

16.11.4 Transferindo Messageports e digitado Matrizes

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 um
Técnica especial.O método PostMessage () (de um objeto de trabalhador,
de threads.parentport, ou de qualquer objeto Messageport) leva um
Segundo argumento opcional.Este argumento (chamado transferlist) é
uma variedade de objetos que devem ser transferidos entre fios e não
sendo copiado.
Um objeto Messageport não pode ser copiado pelo clone estruturado
Algoritmo, mas pode ser transferido.Se o primeiro argumento para
PostMessage () incluiu uma ou mais porteiras (aninhadas
arbitrariamente profundamente dentro do objeto de mensagem), então aqueles Messageport
Objetos também devem aparecer como membros da matriz passados ??como o segundo
argumento.Fazer isso diz ao Node que não precisa fazer uma cópia de
o Messageport, e pode apenas dar o objeto existente ao
outro tópico.A coisa principal a entender, no entanto, sobre a transferência
Os valores entre os threads são que, uma vez transferido um valor, ele não pode
ser usado mais no thread que chamado PostMessage ().
Aqui está como você pode criar um novo MessageChannel e transferir um
de suas porteiras para um trabalhador:

```
// Crie um canal de comunicação personalizado
const threads = requer ("trabalhador_threads");
deixe canal = new threads.MessageChannel ();
// Use o canal padrão do trabalhador para transferir uma extremidade de
o novo
// canal para o trabalhador.Suponha que quando o trabalhador
recebe 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 do
canal 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.Se
você liga
Uma 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 imagem
Processando em milhões de pixels.Então, para eficiência, Postmessage ()
Também nos dá a opção de transferir matrizes digitadas em vez de copiar
eles.(Os threads compartilham a memória por padrão. Trentos do trabalhador em JavaScript
geralmente evite 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, ele
torna -se inutilizável no tópico que o transferiu.Na imagem-
cenário de processamento, o tópico principal pode transferir os pixels de um
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.O
A 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 de
memória
// Suponha que lemos alguns dados nesta matriz digitada e depois
Transfira 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 tentar
Use um Messageport ou uma matriz digitada que foi transferida;esses
Os objetos simplesmente param de fazer qualquer coisa quando você interage com eles.
16.11.5 Compartilhando matrizes digitadas entre threads
Além de transferir matrizes digitadas entre threads, é na verdade
possível compartilhar uma matriz digitada entre os threads.Simplesmente crie um
SharedArrayBuffer do tamanho desejado e depois use esse buffer para criar
uma matriz digitada.Quando uma matriz digitada é apoiada por um
SharedArrayBuffer é passado via PostMessage (), o subjacente
A memória será compartilhada entre os threads.Você não deve incluir
o buffer compartilhado no segundo argumento para PostMessage () neste
caso.
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 escreva de volta.Se dois tópicos estão incrementando um valor ao mesmo tempo, geralmente será incrementado uma vez, como o
O código a seguir demonstra:

```
const threads = requer ("trabalhador_threads");
if (threads.ismhrethread) {
// 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
se 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
incremento 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 de
WorkerData
// e depois aumentam 10 milhões de vezes.
Let SharedArray = Threads.WorkerData;
para (vamos i = 0; i <10_000_000; i ++) sharedArray [0] ++;
// Quando terminarmos o incremento, informe o tópico principal
threads.parentport.postMessage ("feito");
}
```

Um cenário em que pode ser razoável usar um SharedArrayBuffer é quando os dois threads operam totalmente separados seções da memória compartilhada.Você pode aplicar isso criando dois matrizes digitadas que servem como visualizações de regiões não sobrepostas do Buffer compartilhado e, em seguida, seus dois threads usem esses dois separados matrizes digitadas.Uma classificação de mesclagem paralela pode ser feita assim: um tópico classifica 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 são Adequado para esta abordagem: vários tópicos que trabalham em regiões desconts da imagem.

Se você realmente deve permitir que vários tópicos acessem a mesma região de um matriz compartilhada, você pode dar um passo em direção à segurança do tópico com o funções definidas pelo objeto Atomics.Atomics foi adicionado a JavaScript quando SharedArrayBuffer deveria definir operações atômicas Sobre os elementos de uma matriz compartilhada.Por exemplo, o atomics.add () A função lê o elemento especificado de uma matriz compartilhada, adiciona um especificado valor para ele e escreve a soma de volta à matriz.Isso faz isso atomicamente como se fosse uma única operação, e garante que nenhum outro O 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.ismhrethread) {
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);// ThreadSafe
incremento atômico
}
trabalhador.on ("mensagem", (mensagem) => {
// Quando ambos os threads estiverem concluídos, use um threadSafe
função
// para ler a matriz compartilhada e confirmar que
tem 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);// ThreadSafe
incremento 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 profundo
ligações ao sistema operacional significam que também é uma boa alternativa
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
e servidores.

Módulo "Net" do Node para escrever clientes que não http e
servidores.

Módulo "Child_process" do Node para criar e
comunicação com processos infantis.
Módulo "Worker_threads" do Node para verdadeiro multithreaded
Programação usando passagem de mensagens em vez de compartilhado
memória.
1
O nó define uma função fs.copyfile () que você realmente usaria na prática.
2
Muitas vezes, é mais limpo e mais simples definir o código do trabalhador em um arquivo separado.Mas esse truque
de ter dois tópicos executando seções diferentes do mesmo arquivo me impressionaram quando eu primeiro encontrou -o para a chamada do sistema unix fork ().E eu acho que vale a pena demonstrar
Esta técnica simplesmente por sua estranha elegância.

Capítulo 17. Ferramentas JavaScript e extensões

Parabéns por chegar ao capítulo final deste livro.Se você tem Leia tudo o que vem antes, agora você tem um detalhado compreensão da linguagem JavaScript e sabe como usá -la em Nó e nos navegadores da web.Este capítulo é um tipo de graduação presente: introduz um punhado de ferramentas importantes de programação que Muitos programadores JavaScript acham útil e também descrevem dois Extensões amplamente usadas para a linguagem JavaScript central.Seja ou não Você escolhe usar essas ferramentas e extensões para seus próprios projetos, você tem quase certeza de vê -los usados ??em outros projetos, então é importante pelo 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 padronizado caminho.

Jogue como uma solução completa para escrever testes de unidade JavaScript.

NPM para gerenciar e instalar as bibliotecas de software que Seu programa depende.

Ferramentas de Bundling de Código-como Webpack, Rollup e Parcel-que Converta seus módulos de código JavaScript em um único pacote para uso na web.

Babel para traduzir o código JavaScript que usa novidades
Recursos de linguagem (ou que usam extensões de linguagem) em
Código 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 JavaScript
Expressões que se parecem com a marcação HTML.
A extensão da linguagem de fluxo (ou o tipo de texto semelhante
extensão) que permite anotar seu código JavaScript
com 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 suficiente
profundidade que você pode entender por que eles são úteis e quando você pode
quero usá -los.Tudo coberto neste capítulo é amplamente usado em
O mundo da programação JavaScript, e se você decidir adotar uma ferramenta
Ou extensão, você encontrará muita documentação e tutoriais online.

17.1 LING COM ESLINT

Na programação, o termo fiapos refere -se a codificar isso, enquanto tecnicamente
Correto, é desagradável, ou um possível bug, ou abaixo do ideal de alguma forma.UM
Linter 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 você
Execute -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.js
1: 1 Erro Var inesperado, use Let ou const
não-var
1: 5 Erro 'x' é atribuído um valor, mas nunca usado
não-used-vars
1: 9 Strings de aviso devem usar o DoubleQuote
citações
4:11 Erro esperado '===' e, em vez disso, viu '=='
eqeqeq
5: 1 Erro recuo esperado de 8 espaços, mas encontrou 6
recuar
7:28 Erro faltando semicolon
semi
? 6 problemas (5 erros, 1 aviso)
3 erros e 1 aviso potencialmente fixável com o `--fix`
opção.
```

Às vezes, os lintes podem parecer nitpicky.Realmente importa se nós
usou citações duplas ou citações únicas para nossas cordas?Por outro lado,
Obter o recuo correto é importante para a legibilidade e o uso ====
e vamos, em vez de == e Var o protege de bugs sutis.E
Variáveis ??não utilizadas têm peso morto em seu código - não há razão para
Mantenha 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
Defina um arquivo de configuração que tira Eslint para aplicar exatamente as regras
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 que
tipo de aspas são preferidas e se deve haver um
espaço entre a palavra -chave e os parênteses abertos que seguem
isto.

Uma alternativa moderna para aplicar regras de formatação de código por meio de um linhador é
adotar uma ferramenta como o mais bonito para analisar e reformar automaticamente tudo
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 falta
Semicolons, adiciona espaços em torno de operadores binários e insere quebras de linha

depois {e antes}, resultando em muito mais convencional
código:

```
$ Prettier fatorial.js
função fatorial (x) {
if (x === 1) {
retornar 1;
} outro {
retornar x * fatorial (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 executar
Ele 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 e
Então nunca mais preciso pensar em formatar o código novamente.
Pessoalmente, eu realmente gosto de usar projetos mais bonitos em JavaScript.Eu não tenho
usei para o código neste livro, no entanto, porque em grande parte do meu código
Eu confio em uma formatação cuidadosa para alinhar meus comentários verticalmente, e

Mais bonito os estraga.

17.3 Teste de unidade com JEST

Os testes de escrita são uma parte importante de qualquer programação não trivial projeto.Línguas dinâmicas como JavaScript Suport Testing Frameworks Isso reduz drasticamente o esforço necessário para escrever testes e quase Torne a escrita de teste divertida!Existem muitas ferramentas de teste e bibliotecas para JavaScript, 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 = requer ("./ 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 web
que retorna
* Temperaturas mundiais em graus Celsius.
*/
Module.Exports = Função Async GetTemperature (City) {
// Obtenha a temperatura em Celsius do serviço da web
Seja c = aguarda getjson (

`https://globaltemps.example.com/api/city/$ {City.tolowerCase ()
} `
);
// converte em Fahrenheit e retorne esse valor.
retornar (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 0c
Espere (aguarda gettemperature ("x")). Tobe (32);// Nós
Espere 32f
// 100c deve se converter para 212f
getjson.mockResolVedValue (100);// Se
getjson retorna 100c
Espere (aguarda gettemperature ("x")). Tobe (212);// Nós
Espere 212f
});
});
```

Com o teste escrito, podemos usar o comando jest para executá -lo, e nós
Descubra que um de nossos testes falha:

```
$ jest gettemperature
Falha ch17/gettemperature.test.js
getTemperature ()
? Invoca a API correta (4ms)
? converte C para F corretamente (3ms)
? gettemperature () ?converte c para f corretamente
Espere (recebido) .ToBe (esperado) // object.is igualdade
Esperado: 212
Recebido: 87.55555555555556
29 |// 100c deve se converter para 212f
30 |getjson.mockResolVedValue (100);// Se
getjson retorna 100c
> 31 |Espere (aguarde
gettemperature ("x")). Tobe (212);// Espere 212f
|^
32 |});
33 |});
34 |
no objeto. <Anonymous>
```

(CH17/gettemperature.test.js: 31: 43)
Suítes de teste: 1 falha, 1 total
Testes: 1 falhou, 1 passou, 2 total
Instantâneos: 0 Total
Tempo: 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
Exiba a cobertura do código para nossos testes:
$ jest -Cobertura gettemperature
Passe 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 total
Testes: 2 passados, 2 total
Instantâneos: 0 Total
Tempo: 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 litelement 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 depende
e salva essas informações em um arquivo para que quando alguém quiser
Para experimentar o seu programa, eles podem baixar seu código e sua lista de
dependências 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 arquive 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 Express
Além de instalar o pacote nomeado, o NPM também faz um registro
da dependência no arquivo package.json para o projeto.Gravação
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 por
desenvolvedores que querem trabalhar em seu projeto, mas na verdade não são
precisava 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 bonito
Às vezes, você pode querer instalar ferramentas de desenvolvedor globalmente para que
Eles 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@6.7.2
Adicionado 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
O ecossistema de plug-ins, é altamente configurável e pode suportar mais antigo
Bibliotecas 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
Forneça alguns recursos adicionais:
Alguns programas têm mais de um ponto de entrada.Uma web
Aplicação com várias páginas, por exemplo, poderia ser escrita
com um ponto de entrada diferente para cada página.Aparecedores em geral
Permita 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
carregado dinamicamente quando necessário.Isso pode funcionar bem se lá
são apenas algumas chamadas para importar () em seu programa e elas
Módulos de carga com conjuntos de dependências relativamente disjuntos.Se
Os módulos carregados dinamicamente compartilham dependências e depois
torna -se complicado descobrir quantos pacotes produzirem,
E é provável que você tenha que configurar manualmente seu empuxo
para resolver isso.
Muscantes 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
omitido dos feixes.Este recurso passa pelo capricho
Nome de "Troca de árvores".
Matores de pacotes normalmente têm uma arquitetura baseada em plug-in e
Suporte plug-ins que permitam importar e agrupar ?módulos?
que na verdade não são arquivos do código JavaScript.Suponha que isso
Seu programa inclui um grande dados compatível com JSON
estrutura.Os pacotes de código podem ser configurados para permitir que você
Mova 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
Os programas JavaScript podem usar plug-ins de Bundler que lhes permitem
Para importar arquivos CSS com uma diretiva de importação.Nota, no entanto,
que se você importar algo que não seja um arquivo JavaScript, você
estão 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
edita 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 um
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
Recursos em JavaScript que não usa aquela linguagem moderna
características.Porque compila JavaScript ao JavaScript, Babel é
Às vezes chamado de "transpiler".Babel foi criado para que a web
Os desenvolvedores podem usar os novos recursos de idioma do ES6 e mais tarde enquanto
Ainda segmentando navegadores da Web que suportavam apenas o ES5.
Recursos de linguagem, como o ** operador de exponenciação e seta
As funções podem ser transformadas relativamente facilmente em math.pow () e
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 e
Isso ajuda dramaticamente ao trabalhar com código transformado.
Os fornecedores do navegador estão fazendo um trabalho melhor em acompanhar o
Evolução da linguagem JavaScript, e há muito menos necessidade hoje
Para compilar funções de seta e declarações de classe.Babel ainda pode
Ajude 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
Configure para executar o Babel em cada módulo JavaScript, pois é empacotado.
Mesmo que haja menos necessidade de transformar o JavaScript central
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
Defina uma árvore de elementos.JSX está mais intimamente associado ao reagir
estrutura 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.Esse
A seção explica o que você precisa saber para entender dela.(Esse
a seção é sobre a extensão da linguagem JSX, não sobre reação, e
Explica 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 JavaScript
sintaxe.Javascript String literais são delimitados com aspas,
e a expressão regular literais é delimitada com barras.No mesmo
Maneira, a expressão de JSX literais é delimitada com colchetes de ângulo.Aqui está
muito simples:
deixe a linha = <hr/>;
Se você usar o JSX, precisará usar Babel (ou uma ferramenta semelhante) a
Compilar expressões JSX em JavaScript regular.A transformação é
simples o suficiente para que alguns desenvolvedores optem por usar o React sem usar
JSX.Babel transforma a expressão JSX nesta declaração de atribuição
em uma chamada de função simples:
deixe a linha = react.createElement ("hr", nulo);
A sintaxe JSX é como HTML e, como elementos HTML, elementos de reação
pode ter atributos como estes:
Deixe imagem = ;
Quando um elemento tem um ou mais atributos, eles se tornam propriedades de
Um 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 crianças.Assim como os operadores aritméticos de JavaScript podem ser
usado para escrever expressões aritméticas de complexidade arbitrária, JSX
elementos também podem ser aninhados arbitrariamente profundamente para criar árvores de
Elementos:
Deixe a barra lateral = (

<H1> Título </h1>
<hr/>
 Este é o conteúdo da barra lateral

);
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 crianças (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
React.createElement ("h1", null, // este é o primeiro
filho do
"Título"), // e seu próprio primeiro
criança.
React.createElement ("hr", null), // o segundo filho de
o .
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 função
As invocações também podem ser escritas com expressões arbitrárias.Esse
O 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 e
O código resultante passa os parâmetros de função que recebem para
React.createElement () de maneira natural.Observe o truque interessante
que fizemos aqui com o parâmetro da linha drawl e o curto
Circuiting && Operator.Se você ligar para a barra lateral () com apenas três
Argumentos, então drawline padrões para true, e o quarto argumento
para a chamada CreateElement () externo é o elemento <hr/>.Mas se
você passa falsa como o quarto argumento para a barra lateral (), então o
Quarto 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 condicionalmente
um elemento filho, dependendo do valor de alguma outra expressão.(Esse
O idioma trabalha com reação porque o react simplesmente ignora crianças que são
falso ou nulo e não produz nenhuma saída para eles.)
Quando você usa expressões JavaScript nas expressões JSX, você é
não 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 <ul> com uma matriz de <li>
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} </li>
})}
</ul>
);
}
```

Esta função usa um objeto literal como o valor do atributo de estilo
no elemento <ul>.(Observe que os aparelhos duplos encaracolados são necessários
aqui.) O elemento <ul> tem uma única criança, mas o valor dessa criança
é uma matriz.A matriz infantil é a matriz criada usando o mapa ()
função na matriz de entrada para criar uma matriz de elementos <li>.(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
O mesmo que esse elemento com cada um desses elementos da matriz quando crianças.)
Finalmente, observe que cada um dos elementos aninhados <li> tem um OnClick
Atributo do manipulador de eventos cujo valor é uma função de seta.O código JSX
compila com o seguinte código JavaScript puro (que eu tenho
formatado 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 SPLPINC Operador (§6.10.4) para especificar vários atributos de uma só vez.Suponha que isso Você se vê escrevendo muitas expressões JSX que repetem um conjunto comum de atributos.Você pode simplificar suas expressões por definindo os atributos como propriedades de um objeto e ?espalhando -os em "seus elementos JSX:

```
Seja hebraico = {lang: "He", dir: "rtl"};// Especifique a linguagem
e direção
Seja Shalom = <span className = "ênfase"
{... hebraico}> ???? </span>;
```

Babel compila isso para usar uma função _extends () (omitida aqui) que combina esse atributo de nome de classe com os atributos contidos no 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 identificador
Imediatamente após o suporte do ângulo de abertura.Se a primeira letra disso
Identificador é minúsculo (como tem sido em todos os exemplos aqui), então
O identificador é passado para CreateElement () como uma string.Mas se o
A primeira letra do identificador é a maçaneta, então é tratada como um real
identificar, e é o valor JavaScript desse identificador que é passado
como o primeiro argumento a CreateElement ().Isso significa que o JSX
Expressão <Math /> compila com o código JavaScript que passa o
Objeto de matemática global para react.createElement ().
Para reagir, essa capacidade de passar valores de não coragem como o primeiro argumento a
CreateElement () permite a criação de componentes.UM
O componente é uma maneira de escrever uma expressão JSX simples (com um
Nome do componente em maiúsculas) que representa um mais complexo
expressão (usando nomes de tags HTML em minúsculas).
A maneira mais simples de definir um novo componente no React é escrever um
função que pega um "objeto de adereços" como seu argumento e retorna um JSX
expressão.Um objeto de adereços é simplesmente um objeto JavaScript que representa
valores de atributo, como os objetos que são aprovados como o segundo argumento
para createElement ().Aqui, por exemplo, é outra opinião sobre o nosso
Função 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 e
leva um único argumento de objeto em vez de argumentos separados.Esse
o torna um componente de reação e significa que ele pode ser usado no lugar de
Um 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 (o
Função da barra lateral ()) e usará a expressão JSX retornada por
essa função no lugar da expressão <apara>.
17.8 Verificação do tipo com fluxo
O fluxo é uma extensão de linguagem que permite anotar seu
Código JavaScript com informações de tipo e uma ferramenta para verificar seu
Código JavaScript (anotado e não anotado) para erros de tipo.Para
Use o fluxo, você começa a escrever código usando a extensão da linguagem de fluxo para
Adicione 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 código
processo 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 lá
não é uma nova sintaxe que o fluxo precisa compilar ou transformar.Você usa
a extensão da linguagem de fluxo para adicionar anotações ao código e a todos
Babel precisa fazer é retirar essas anotações para devolver seu código para
JavaScript padrão.)

TypeScript versus Flow

O TypeScript é uma alternativa muito popular ao fluxo.TypeScript é uma extensão do JavaScript que adiciona
Tipos e outros recursos de linguagem.O compilador TypeScript "TSC" compila TypeScript
programas em programas JavaScript e, no processo
da 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 no fluxo.
Para uma digitação mais avançada, a sintaxe das duas extensões diverge, mas a intenção e o valor do
Duas extensões são as mesmas.Meu objetivo nesta seção é explicar os benefícios das anotações do tipo e
Análise de código estático.Eu farei isso com exemplos com base no fluxo, mas tudo demonstrado aqui
Também 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 ou um
para/de loop ou módulos ou promessas.O fluxo é uma extensão de linguagem estreita que adiciona tipo
anotações para JavaScript e nada mais.TypeScript, por outro lado, foi muito projetado como um
nova 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 adiciona
JavaScript: a linguagem TypeScript possui palavras -chave enum e namespace que simplesmente não existem em
JavaScript.Em 2020, o TypeScript tem melhor integração com IDEs e editores de código (particularmente
O 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 porque
Não quero tirar o foco do JavaScript.Mas tudo o que você aprende aqui sobre adicionar tipos a
O JavaScript será útil para você se você decidir adotar o TypeScript para seus projetos.
Usar o fluxo requer compromisso, mas eu descobri isso para o meio
E grandes projetos, o esforço extra vale a pena.Leva tempo extra para adicionar
Digite 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
Sentir 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 alguns
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
Sinta -se 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
acaba gastando tempo lendo a documentação em https://flow.org.Sobre
Por outro lado, você não precisa dominar o sistema de tipo de fluxo antes
Você pode 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-dev
Fluxo-barracão.Se você instalar a ferramenta globalmente com -g, poderá executá -la
com 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 fazer
Verificação de tipo, a primeira vez que o executa como fluxo -init na raiz
Diretório do seu projeto para criar um arquivo de configuração .flowconfig.
Você pode nunca precisar adicionar nada a este arquivo, mas o fluxo precisa
Saiba onde está o seu projeto.
Quando você executa o fluxo, ele encontrará todo o código -fonte JavaScript em seu
projeto, mas ele relatará apenas erros de tipo para os arquivos que ?optaram
em ?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 pode
Adote fluxo para projetos existentes e depois comece a converter seu código
um arquivo de cada vez, sem ser incomodado por erros e avisos em
arquivos que ainda não foram convertidos.
O fluxo pode encontrar erros em seu código, mesmo que tudo o que você faça seja optar
com um comentário // @flow.Mesmo se você não usar a linguagem de fluxo
Extensão e não adicione nenhuma anotações de tipo ao seu código, o tipo de fluxo
A ferramenta de verificador ainda pode fazer inferências sobre os valores em seu programa
e alertá -lo quando você os usa inconsistentemente.
Considere a seguinte mensagem de erro de fluxo:
Erro ??????????????????????????????????????
VariBlebreASSignment.js: 6: 3
Não 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 loop
substitui i
4? console.log (i);
5?}
6? I.R = 1;// O fluxo detecta o erro
aqui
```

Nesse caso, declaramos a variável i e atribuímos um objeto a ele.Então
Usamos i novamente como uma variável de loop, substituindo o objeto.Avisos de fluxo
Isso 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 loop
variável local para o loop.)
Aqui está outro erro que o fluxo detecta mesmo sem anotações de tipo:
Erro

```
????????????????????????????????????????????????????
size.js: 3: 14
Não pode obter x.length porque o comprimento da propriedade está ausente em
Número [1].
1? // @flow
2? 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 é
Espera -se ter uma propriedade de comprimento.Quando vê esse tamanho ()
a função sendo chamada com um argumento numérico, ele sinaliza corretamente
um 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 fluxo
anotação a ele seguindo o nome da variável com um cólon 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, flua
conhece o tipo de variável e que você expressou o
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 cólon e
o nome do tipo.Ao anotar uma função, você normalmente também adiciona um
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 especificar explicitamente que espera um argumento de string e
Retorna um número.Observe que o Flow agora sinaliza um erro se passarmos uma matriz para
A função, mesmo que a função funcione nesse caso:

Erro
??
size2.js: 5: 18
Não pode chamar o tamanho com a matriz literal ligada a s porque a matriz
literal [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, embora
pode 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 javascript
valor nulo tem o tipo de fluxo nulo e o valor JavaScript
indefinido tem o tipo de fluxo vazio.Mas nenhum desses valores é um
Membro de qualquer outro tipo (a menos que você o adicione explicitamente).Se você declarar
um parâmetro de função para ser uma string, então deve ser uma string e é um
erro de passar nulo ou passar indefinido ou omitir o argumento
(que é basicamente a mesma coisa que a passagem indefinida):
Erro
??
size3.js: 3: 18
Não pode chamar tamanho com nulo ligado a s porque nulo [1] é
incompatível
com 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 pergunta
marca.Por exemplo, use? String ou? Número em vez de string ou
número.Se mudarmos nossa função de tamanho () para esperar um argumento de
digite? string, então o fluxo não se queixa quando passamos nulos para
a função.Mas agora tem outra coisa para reclamar:
Erro
??
size4.js: 3: 14
Não pode obter o comprimento S., porque o comprimento da propriedade está ausente em
nulo 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. comprimento
Porque, neste lugar em nosso código, S pode ser nulo ou indefinido,
e esses valores não têm propriedades de comprimento.É aqui que flui
Garanta que não cortamos nenhum canto.Se um valor puder ser nulo, flua
Will insistirá que verificamos esse caso antes de fazer qualquer coisa que
depende 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 nula
ou 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.Observação
Esse 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 tipo
indique que, além do tipo especificado, nulo e indefinido
são permitidos também.Pontos de interrogação também podem aparecer após um parâmetro
nome para indicar que o próprio parâmetro é opcional.Então, se mudarmos
A 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 isso
O 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 função
valores.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 classe
nome 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.toisSotring ());
}
```

Se você definir suas próprias aulas com a palavra -chave da classe, essas classes
Torne -se automaticamente tipos de fluxo válidos.Para fazer isso funcionar,
No entanto, o fluxo exige que você use anotações de tipo na classe.Em
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 houvesse
não a
// Tipo anotação para i dentro da classe.
Complex.i = novo complexo (0,1);

17.8.4 Tipos de objetos

O 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 numérico
propriedades:

// @fluxo
// recebeu um objeto com propriedades X e Y numéricas, retorne
o
// 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 pergunta
Marque 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
eles aqui.
};
exportar locações de cidades padrão;
17.8.5 Aliases do tipo
Objetos podem ter muitas propriedades e o tipo de fluxo que descreve
Esse objeto será longo e difícil de digitar.E mesmo relativamente
Tipos 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 fluxo
tipos.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 igual
Tipo 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:
// @fluxo
Ponto de tipo de exportação = {
x: número,
Y: Número
};
// Dado um objeto de ponto, retorne sua distância da origem
Exportar 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
definição.Lembre -se, porém, esse tipo de importação é um fluxo
Extensão do idioma e não uma diretiva de importação JavaScript real.Tipo
importações e exportações são usadas pelo verificador do tipo fluxo, mas como todas
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
Isso espera uma variedade de números e o erro que o fluxo relata se você
Tente chamar a função com uma matriz que possui elementos não numéricos:
Erro ??
média.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 a
a função size () de anteriores e modificá -lo para esperar uma matriz
argumento em vez de um argumento de string.Queremos que a função seja capaz
Para aceitar uma matriz de qualquer comprimento, um tipo de tupla não é apropriado.Mas
Não queremos restringir nossa função a trabalhar apenas para matrizes onde
Todos os elementos têm o mesmo tipo.A solução é o tipo
Array <fixed>:
// @fluxo
Tamanho 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 tentou
Para usar qualquer um desses elementos, o fluxo insistiria que usamos o tipo de
verificações ou outros testes para determinar o tipo do elemento antes
executando qualquer operação insegura nela.(Se você estiver disposto a desistir
Verificação de tipo, você também pode usar qualquer um em vez de misto: permite que você
fazer o que quiser com os valores da matriz sem garantir
que os valores são do tipo que você espera.)

17.8.7 Outros tipos parametrizados

Vimos isso quando você anota um valor como uma matriz, flua
exige que você também especifique o tipo de elementos de matriz dentro do ângulo
Suportes.Este tipo adicional é conhecido como parâmetro de tipo e matriz
não é a única classe JavaScript que é parametrizada.

A classe set de JavaScript é uma coleção de elementos, como uma matriz é, e
Você não pode usar o conjunto como um tipo por si só, mas precisa incluir um tipo
parâmetro entre colchetes de ângulo para especificar o tipo de valores
contido no conjunto.(Embora você possa usar misto ou qualquer um, se o conjunto
pode 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 tipos
parâmetros que devem ser especificados;o tipo de chaves e os tipos de
os valores:

```
// @fluxo
Importar 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 (ver
Object.freeze () em §14.2 se você deseja objetos verdadeiros somente leitura), mas
Porque permite que você pegue bugs causados ??por não intencional
modificações.Se você escrever uma função que leva um objeto ou matriz
argumento e não muda nenhuma das propriedades do objeto ou o
elementos da matriz, então você pode anotar o parâmetro de função com
Um dos tipos somente leitura do Flow.Se você fizer isso, o fluxo relatará um
Erro se você esquecer e modificar acidentalmente o valor de entrada.Aqui estão
Dois exemplos:

```
// @fluxo
tipo de ponto = {x: número, y: número};
// Esta função leva um objeto de ponto, mas promete não
modifique -o
Distância 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 vai
modificar
Média 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 <Mumber> = [1,2,3,4,5];
média (dados) // => 3
```

17.8.9 Tipos de função

Vimos como adicionar anotações de tipo para especificar os tipos de um

Os parâmetros da função e seu tipo de retorno.Mas quando um dos
Parâmetros de uma função é uma função, precisamos ser capazes de
Especifique 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 ()
abaixo
Tipo 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 REPORTIÇÃO.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
faça 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
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 vertical
Personagens de bar:

```
// @fluxo
Tamanho da função (coleção:
Array <fixed> | set <fixed> | mapa <mixed, mixed>): número {
if (Array.esArray (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 um
Mapa ? - o fato de essa sintaxe de fluxo usa a mesma barra vertical
O personagem como JavaScript ou operadores é intencional.
Vimos anteriormente que colocar um ponto de interrogação antes de um tipo permite nulo
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
Permita 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
Sindicatos
O fluxo permite que você use literais primitivos como tipos que consistem nisso
um único valor.Se você escrever, deixe x: 3;, então o fluxo não permitirá
você atribui 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 discriminados
sindicatos.Quando você trabalha com tipos de sindicatos (composto de realmente
tipos diferentes, não de literais), você normalmente precisa escrever código para
discriminar entre os tipos possíveis.Na seção anterior, nós
escreveu uma função que poderia levar uma matriz ou um conjunto ou um mapa como seu
argumento e tive que escrever código para discriminar a entrada da matriz de set ou
Entrada de mapa.Se você deseja criar uma união de tipos de objetos, você pode fazer
Esses tipos fáceis de discriminar usando um tipo literal em cada um dos
os tipos de objetos individuais.
Um exemplo deixará isso claro.Suponha que você esteja usando um tópico de trabalhador
no nó (§16.11) e estão usando PostMessage () e "Mensagem"
eventos para enviar mensagens baseadas em objetos entre o tópico principal e
o tópico do trabalhador.Existem vários tipos de mensagens que o
O trabalhador pode querer enviar para o tópico principal, mas gostaríamos de escrever um
Tipo de união de fluxo que descreve todas as mensagens possíveis.Considere isso
código:

```
// @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 é
definido em outro lugar.
};
// O trabalhador envia uma mensagem desse tipo se seu código falhou
com uma exceção.
Tipo de exportação errorMessage = {
MessageType: "Erro",
Erro: erro,
};
```

```
// O trabalhador envia uma mensagem desse tipo para relatar o uso
Estatística.
Tipo de exportação StatisticsMessage = {
messageType: "estatísticas",
splinesReticulules: Number,
splinespersegund: número
};
// Quando recebermos uma mensagem do trabalhador, será um
WorkerMessage.
Tipo de exportação workermessage = resultMessage |ErrorMessage |
StatisticsMessage;
// O encadeamento principal terá uma função de manipulador de eventos que
é passado
// Um ??Workermessage.Mas porque definimos cuidadosamente cada um
do
// Tipos de mensagem para ter uma propriedade MessageType com um
tipo literal,
// O manipulador de eventos pode facilmente discriminar entre os
Mensagens possíveis:
função handleMessageFromReticulator (Mensagem: WorkerMessage)
{
if (message.MessageType === "resultado") {
// apenas o resultado
este valor
// Então Flow sabe que é seguro usar
message.Result aqui.
// e fluxo irá reclamar se você tentar usar qualquer outro
propriedade.
console.log (message.result);
} else if (message.MessageType === "Error") {
// Somente errorMessage tem uma propriedade Messagetype com
valor "erro"
// SONTE SABE QUE É SEGURO USAR MENSAGEM.Error
aqui.
lança message.error;
} else if (message.MessageType === "Stats") {
// Somente StatisticsMessage possui uma propriedade Messagetype
com valor "estatísticas"
// então sabe que é seguro usar
message.splinespersegund aqui.
console.info (message.splinespersecond);
}
```

Erro ao traduzir esta página.

Índice

Símbolos

!(Boolean não operador), lógico não (!)

! = (Operador de desigualdade não estrito)

Expressões relacionais, igualdade e operadores de desigualdade

Digite conversões, conversões especiais de operadoras de casos

! == (operador de desigualdade)

valores booleanos, valores booleanos

Visão geral de operadores de igualdade e desigualdade

Comparação 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 e

Operadores, expressões aritméticas

** (operador de exponenciação), aritmética em JavaScript, aritmética

Expressões

+ (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
++ (operador de incremento), operadores aritméticos unários
, (operadora de vírgula), o operador de vírgula (,)
- (sinal de menos)
Operador de subtração, aritmética em JavaScript, aritmética
Expressões
Operador 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,
Comentários
Gráficos 3D, gráficos em uma <VAS>
;(Semicolon), semicolons opcionais semicolons-opcionais
<(menos que o operador)

Visão geral dos operadores de comparação
Comparação de strings, trabalhando com strings
Digite conversões, conversões especiais de operadoras de casos
<< (Operador esquerdo de mudança), operadores bitwise
<= (menor ou igual ao operador)
Visão geral dos operadores de comparação
Comparação de strings, trabalhando com strings
Digite conversões, conversões especiais de operadoras de casos
= (operador de atribuição), um tour de JavaScript, igualdade e desigualdade
Operadores, expressões de atribuição
== (operador de igualdade)
Visão geral de operadores de igualdade e desigualdade
Digite conversões, visão geral e definições, conversões e
Igualdade, conversões especiais de operadoras de casos
=== (operador de igualdade rigoroso)
valores booleanos, valores booleanos
Visão geral de operadores de igualdade e desigualdade
Comparação de strings, trabalhando com strings
Digite conversões, visão geral e definições, conversões e
Igualdade
=> (setas), um tour de JavaScript, semicolons opcionais, seta
Funções
> (maior que o operador)

Visão geral dos operadores de comparação
Comparação de strings, trabalhando com strings
Digite conversões, conversões especiais de operadoras de casos
> = (maior ou igual ao operador)
Visão geral dos operadores de comparação
Comparação de strings, trabalhando com strings
Digite 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, função
Invocação
?: (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, leitura
e 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 SEQUÊNCIÇÕES, ESCAPE
Sequências 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, o
Espalhe o operador para chamadas de função, iteradores e geradores
UM
Classes abstratas, hierarquias de classe e classes abstratas-Summary
acelerômetros, APIs de dispositivo móvel
Propriedades do acessador, getters de propriedades e setters
Método addEventListener (), addEventListener ()
Operador de adição (+), aritmético em JavaScript, o operador +
Recursos avançados
extensibilidade do objeto, extensibilidade do objeto
Visão geral da metaprogramação
Atributos da propriedade, atributos de propriedades atributos
Atributo do protótipo, o atributo do protótipo
Objetos de proxy, objetos de proxy-proxy invariantes
Reflita API, a API refletida-refletir API

Tags de modelos, tags de modelo de modelo tags
símbolos bem conhecidos
Símbolos de correspondência de padrões, símbolos de correspondência de padrões
Symbol.asynciterator, símbolo.iterator e
Symbol.asynciterator
Symbol.hasinsance, símbolo.hasinstance
Symbol.iscoNcatsPreadable, symbol.iscoNcatsPreadable
Symbol.iterator, símbolos conhecidos
Symbol.spécies, símbolo.species-symbol.species
Symbol.Toprimitive, símbolo.Toprimitivo
Symbol.ToStringTag, symbol.ToStringTag
Symbol.UNSCOPABLES, SYMBOL.UNSCOPABLES
alfabetização, comparando strings
caracteres âncora, especificando a posição de correspondência
Apostróficos, literais de cordas
Método Aplicar (), Invocação Indireta, Métodos Call () e Aplicar ()
Método arc (), curvas
Método arcto (), curvas
argumentos
tipos de argumento, tipos de argumento
Objeto de argumentos, o objeto de argumentos
Definição de termo, funções

Destructar a função argumentos nos parâmetros, destruindo
Função Argumentos na função de destruturação de parâmetros
Argumentos nos parâmetros
listas de argumentos de comprimento variável, parâmetros de descanso e variável-
Listas de argumentos de comprimento
Operadores aritméticos, um passeio de JavaScript, aritmético em JavaScript-
Aritmética em javascript, operadoras de expressões aritméticas-bitwise
índice de matriz, leitura e escrita de elementos de matriz
Métodos do iterador da matriz
cada () e alguns (), cada () e outros ()
filtro (), filtro ()
Find () e FindIndex (), Find () e FindIndex ()
foreach (), foreach ()
map (), map ()
Visão geral dos métodos do iterador de matriz
Reduce () e ReduceRight (), Reduce () e ReduceRight ()
matriz literais, objetos e matrizes inicializadores, matrizes literais
Array () construtor, o construtor da matriz ()
Array.from () função, Array.from (), funções estáticas da matriz
Função de Array.ISArray (), Funções de Array estático
Array.of () função, array.of (), funções estáticas de matriz
Array.prototype, matrizes, objetos semelhantes a matrizes
Método Array.sort (), funciona como valores

Método ArrayBuffer (), analisando os corpos de resposta
matrizes
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 estática
subarrays, subarrays with slice (), splice (), preench () 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 matriz
Inicializadores
matrizes de iteração, matrizes de iteração
Matrizes multidimensionais, matrizes multidimensionais
Inicializadores aninhados, objetos e matrizes
Visão geral de, um tour de JavaScript, matrizes
processamento com funções, processamento de matrizes com funções
LEITURA E ESCREVER ELEMENTOS DE ARAY, LEITURA E ESCREVER ARRAY
Elementos
Matrizes esparsas, matrizes esparsas
cordas como matrizes, cordas como matrizes
matrizes digitadas
Criando, criando matrizes digitadas
DataView e Endianness, DataView e Endianness
métodos e propriedades, métodos de matriz digitados e
Propriedades
Visão geral de matrizes digitadas e dados binários
Tipos de matriz digitados, tipos de matriz digitados
Usando, usando matrizes digitadas
Funções de seta, um tour de JavaScript, funções definidoras, flechas
Funções
setas (=>), um tour de javascript, semicolons opcionais, seta
Funções
Caracteres de controle ASCII, o texto de um programa JavaScript

Afirações, um passeio de JavaScript
Operador de atribuição (=), um tour de JavaScript, igualdade e desigualdade
Operadores, expressões de atribuição
Matrizes associativas, introdução a objetos, objetos como associativa
Matrizes
Associatividade, Associatividade do Operador
Palavra-chave assíncrona, ASYNC e AWAIT-IMPLEMENTATION Detalhes
Programação assíncrona (ver também nó)
assíncrono e aguardar palavras-chave, assíncronas e implementação
Detalhes
iteração assíncrona
geradores assíncronos, geradores assíncronos
iteradores assíncronos, iteradores assíncronos
Para/aguarda loops, iteração assíncrona com/aguardar,
Iteração assíncrona
implementação, implementando iteradores assíncronos-
Implementando iteradores assíncronos
retornos de chamada
retornos de chamada e eventos no nó, retornos de chamada e eventos no nó
Definição de termo, programação assíncrona com
Retornos de chamada
eventos, eventos
Eventos de rede, eventos de rede
Timers, temporizadores

Definição de termo, javascript assíncrono
Javascript Suporte para JavaScript assíncrono
Promessas
encadear promessas, encadear promessas de promessas
lidar com erros com, lidar com erros com promessas, mais sobre
Promessas e erros-a captura e finalmente métodos
fazendo promessas, fazendo promessas-promessas em sequência
Visão geral de, promessas
operações paralelas, promessas em paralelo
Promessas em sequência, promessas em promoções de sequência em
Sequência
Resolvendo promessas, resolvendo promessas-mais nas promessas
e erros
retornando dos retornos de chamada de promessa, o problema e finalmente
Métodos
terminologia, lidando com erros com promessas
Usando, usando erros de manipulação de promessas com promessas
APIs de áudio
Construtor de áudio (), o construtor Audio ()
Visão geral de APIs de áudio
API Webaudio, a API Webaudio
Aguarde detalhes-chave, assíncronos e detalhes de implementação
aguardar operador, o operador aguardar

Erro ao traduzir esta página.

Booleano ou operador (||), valores booleanos, lógicos ou (||)
valores booleanos, valores booleanos-booleanos
Função booleana (), conversões explícitas
quebrar declarações, quebre
Ferramentas de desenvolvimento do navegador, explorando JavaScript
História de navegação
Gerenciando com eventos de hashchange, gerenciamento de história com HashChange Events
Gerenciando com PushState (), Gerenciamento de História com PushState ()
Visão geral da história de navegação
Algoritmo de clone estruturado, gerenciamento de história com pushState ()
Classe de buffer (nó), buffers
C
API de cache, aplicativos da Web progressivos e trabalhadores de serviço
calendários, datas de formatação e horários
Call () Método, Invocação Indireta, Métodos Call () e Aplicação ()
retornos de chamada
retornos de chamada e eventos no nó, retornos de chamada e eventos no nó
Definição de termo, programação assíncrona com retornos de chamada
eventos, eventos
Eventos de rede, eventos de rede
Timers, temporizadores

Erro ao traduzir esta página.

manipulação de pixels, manipulação de pixels
Sensibilidade ao caso, o texto de um programa JavaScript
Pegue cláusulas, tentativas/finalmente declarações de Catch/Finalmente
Pegue declarações, aulas de erro
Método .Catch (), a captura e finalmente métodos-a captura e finalmente
Métodos
Classes de personagens (expressões regulares), classes de personagens
Histogramas de frequência do caractere, exemplo: frequência do personagem
Histogramas-Summary
Método Charat (), Strings como matrizes
função checkScope (), fechamentos
Processos Infantis (Nó), Trabalhando com Processos Infantis-Fork ()
benefícios de trabalhar com processos infantis
Exec () e Execfile (), Exec () e Execfile ()
execsync () e execfilesync (), execsync () e execfilesync ()
fork (), garfo ()
Opções, Execsync () e ExecFileSync ()
Spawn (), Spawn ()
Declaração de classe, classe
Palavra-chave da classe, aulas com a palavra-chave-exemplo de classe: um complexo
Classe de número
Métodos de classe, métodos estáticos
classes

Adicionando métodos às classes existentes, adicionando métodos aos existentes
Classes
classes e construtores, classes e construtores-
Propriedade do construtor
propriedade construtora, a propriedade do construtor
construtores, identidade de classe e instância, construtores,
Identidade de classe e instanceof
Expressão, classes e construtores do New.Target
classes e protótipos, classes e protótipos
Aulas com palavra-chave de classe, classes com a palavra-chave de classe-
Exemplo: uma classe de números complexos
Exemplo de classe de número complexo, exemplo: um complexo
Exemplo de classe de número: uma classe de números complexos
getters, setters e outros formulários de método, getters, setters e
Outros formulários de método
Campos públicos privados e estáticos, públicos, privados e estáticos
Campos-campos, áreas públicas, privadas e estáticas
Métodos estáticos, métodos estáticos
programação modular com, módulos com classes, objetos e
Fechamentos
Nomeação, palavras reservadas
Visão geral de, visão geral e definições, classes
subclasses
hierarquias de classe e classes abstratas, hierarquias de classe e
Classes abstratas-Summary

Erro ao traduzir esta página.

Funções definidas por, a API do console-a API do console
suporte para, a API do console
Função Console.log (), Hello World, Saída do Console
palavra -chave const, declarações com let e const
constantes
declarando, visão geral e definições, declarações com let e
const, const, let e var
Definição de termo, declaração e atribuição variáveis
Nomeação, palavras reservadas
construtores
Array () construtor, o construtor da matriz ()
Construtor de áudio (), o construtor Audio ()
Aulas e, aulas e construtores-a propriedade do construtor
Propriedade do construtor, a propriedade do construtor-
Propriedade do construtor
construtores, identidade de classe e instância, construtores,
Identidade de classe e instanceof
Expressão, classes e construtores do New.Target
Invocação construtora, invocação construtora
Definição de termo, funções
exemplos de, criando objetos com novo
Function () construtor, o construtor function ()
SET () construtor, a classe Set

Erro ao traduzir esta página.

Folhas de estilo CSS
Estilos CSS comuns, Scripts CSS
Estilos computados, estilos computados
Animações e eventos CSS, animações e eventos CSS
Classes CSS, aulas CSS
Sintaxe do seletor CSS, selecionando elementos com seletores CSS
Estilos embutidos, estilos embutidos
Convenções de nomeação, estilos em linha
folhas de estilo de script, folhas de estilo de script
Objeto CSSStyleleDeclaration, estilos em linha
Brace Curly ({}), um passeio de JavaScript, Objeto e Array Inicializadores
moeda, números de formatação
curvas, curvas
D
Propriedades de dados, getters de propriedades e setters
Classe DataView, DataView e Endianness
Tipo de data, visão geral e definições, datas e horários
datas e tempos
data aritmética, data aritmética
Formatação e análise Data de cadeias, formatação e análise de análise
Cordas
formatação para internacionalização, formatação e datas

Datas e tempos de formatação
Timestamps de alta resolução, registro de data e hora
Visão geral de, datas e horários, datas e horários
TIMESTAMPS, TIMESTAMPS
Declarações Debugger, Debugger
declarações
classe, classe
const, let e var, const, let e var
função, função
importar e exportar, importar e exportar
Visão geral de, declarações
função decodeuri (), funções de URL legado
função decodeuricomponent (), funções de URL legado
Operador de decrementos (-), operadores aritméticos unários
delegação, delegação em vez de herança
Excluir operador, o operador de exclusão, excluindo propriedades
ataques de negação de serviço, escrevendo para fluxos e manuseio
Backpressure
atribuição de destruição, destruição de tarefas
Atribuição, destruição da função argumentos em parâmetros-
Destructar os argumentos da função em parâmetros
Ferramentas de desenvolvimento, explorando JavaScript
Evento de Devicemotion, APIs de dispositivos móveis

Erro ao traduzir esta página.

geração dinamicamente tabelas de conteúdo, exemplo: gerando um
Índice
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, retângulos
texto, texto
Matrizes dinâmicas, matrizes
E
Padrão ECMA402, a API de internacionalização
ECMAScript (s), Introdução ao JavaScript
Método elementFromPoint (), coordenadas de documentos e viewport
Coordenadas
elementos
elementos da matriz
Definição de termo, matrizes
Leitura e escrita, leitura e escrita de elementos de matriz
elementos do documento
elementos personalizados, elementos personalizados
determinando elemento em um ponto, determinando o elemento em um
Apontar
iframe, documentar coordenadas e coordenadas de viewport
consulta geometria de elementos, consultando a geometria de
um elemento
Selecionando, selecionando elementos do documento
Método Ellipse (), curvas
caso contrário, se declarações, senão se
emojis, unicode, sequências de fuga em literais de string

declarações vazias, declarações compostas e vazias
Strings vazios, texto
função codeuri (), funções do URL do legado
função codeuricomponent (), funções de URL legado
Contrações em inglês, literais de cordas
atributo enumerável, introdução a objetos, atributos de propriedade
Operador de igualdade (==)
Visão geral de operadores de igualdade e desigualdade
Digite conversões, visão geral e definições, conversões e
Igualdade, conversões especiais de operadoras de casos
Operadores de igualdade, um tour pelo JavaScript
Classes de erro, classes de erro
manuseio de erros
Usando promessas, lidando com erros com promessas, mais sobre promessas
e erros
Ambiente de host do navegador da web, erros de programa
ES2016
Operador de exponenciação (**), aritmética em JavaScript, aritmética
Expressões
Inclui () método, inclui ()
ES2017, Palavras -chave assíncronas e aguardam, o operador aguardando,
Detalhes assíncronos de JavaScript, Async e Agud-Implementation
ES2018

Iterador assíncrono, iteração assíncrona com para/aguardar,
Iteração assíncrona
Destruição com parâmetros de descanso, função de destruição
Argumentos nos parâmetros
.Finalmente () Método, a captura e finalmente métodos
expressões regulares
ASSERÇÕES LOLHEBEHIND, especificando a posição de correspondência
nomeados grupos de captura, alternância, agrupamento e referências
s 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, tente/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 promessas
Manipulação de erros com, mais sobre promessas e erros-a captura
e finalmente métodos
fazendo promessas, fazendo promessas-promessas em sequência
Visão geral de, promessas
operações paralelas, promessas em paralelo
Promessas em sequência, promessas em promoções de sequência em
Sequência
Resolvendo promessas, resolvendo promessas-mais nas promessas
e erros
retornando dos retornos de chamada de promessa, o problema e finalmente
Métodos
Usando, usando erros de manipulação de promessas com promessas
Ordem de enumeração da propriedade, ordem de enumeração de propriedade
liberação de, introdução ao javascript
Definir e mapas a classes, para/de com set e mapa
Métodos abreviados, métodos abreviados
Operador espalhado (?), o operador de propagação
Strings delimitadas com backsticks, literais de cordas, literais de modelo
Subclasses com cláusula estends, subclasses com estendências e
Super-subclasses com extensões e super
Tipo de símbolo, visão geral e definições
Símbolos como nomes de propriedades, símbolos como nomes de propriedades
Matrizes digitadas, matrizes

Erro ao traduzir esta página.

Registrando manipuladores de eventos, registrando manipuladores de eventos
Eventos enviados ao servidor, eventos enviados ao servidor
Recursos da plataforma da web para investigar, eventos
EventEmitter Class, Events and EventEmitter
todo () método, todo () e alguns ()
exceções, jogando e pegando, jogue
método EXEC (), EXEC ()
notação exponencial, literais de ponto flutuante
Operador de exponenciação (**), aritmética em JavaScript, aritmética
Expressões
Exportar declaração, importação e exportação
Exportar palavras -chave, módulos em ES6
declarações de expressão, declarações de expressão
expressões
Expressões aritméticas, expressões aritméticas-Bitwise Operadores
Expressões de atribuição, assinatura de expressões de atribuição
com operação
Definição de termo, expressões e operadores
Incorporação em literais de cordas, literais de cordas
Expressões de avaliação, Expressões de avaliação Strict Eval ()
formando -se com operadores, um tour por JavaScript, expressões e
Operadores
Expressões de definição de função, expressões de definição de função

expressões de função, expressões de função, funções como
Namespaces
Expressão inicializadora, um passeio de JavaScript, objeto e matriz
Inicializadores
Expressões de invocação, expressões de invocação condicionais
Invocação, invocação de invocação Invocação
Expressões lógicas, expressões lógicas não (!)
Expressão, classes e construtores do New.Target
Inicializadores de objetos e matrizes, Inicializadores de objetos e matrizes
Expressões de criação de objetos, expressões de criação de objetos
Expressões primárias, expressões primárias
Expressões de acesso à propriedade, expressões de acesso à propriedade-
Acesso à propriedade condicional
expressões relacionais, expressões relacionais-a instância
Operador
versus declarações, um tour de javascript, declarações
extensibilidade, extensibilidade do objeto
F
Função () Função, declarações de função, invocação de funções
Funções de fábrica, aulas e protótipos
valores falsamente, valores booleanos
função fetch (), eventos de rede
Método Fetch ()

Erro ao traduzir esta página.

Lendo arquivos, leitura de arquivos
Escrevendo arquivos, escrevendo arquivos
método de preenchimento (), preenchimento ()
Método filtro (), filtro ()
.Finalmente () método, a captura e finalmente métodos-a captura e finalmente
Métodos
Números de contas financeiras, armazenamento
Método Find (), Find () e FindIndex ()
Método FindIndex (), Find () e FindIndex ()
Ferramentas de desenvolvedor do Firefox, explorando JavaScript
operador 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 binário
e erros de arredondamento
Extensão da linguagem de fluxo, verificação de tipo com fluxo de fluxo
Tipos e sindicatos discriminados
Tipos de matriz, tipos de matriz
Tipos de classe, tipos de classe
tipos enumerados e sindicatos discriminados, tipos enumerados
e sindicatos discriminados
Tipos de funções, tipos de função
Instalando e executando, instalando e executando 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
para loops, para, matrizes de iteração
para/aguarda loops, iteração assíncrona com/aguardar, assíncrona
Iteração
para/em loops, para/in, enumerando propriedades
Para/de loops, texto, para/de OF/in, matrizes de iteração, iteradores e
Geradores
Método foreach (), matrizes de iteração, foreach ()
Método formato (), números de formatação
frações, números de formatação
Método FromData (), analisando os corpos de resposta
JavaScript de front-end, JavaScript em navegadores da web
Módulo FS (nó), trabalhando com arquivos que trabalham com diretórios
Declaração da função, função
Expressões de funções, expressões de função, funções como espaço para nome
Função Palavra -chave, Definindo funções

Function () construtor, o construtor function ()
função* palavra -chave, geradores
funções
Funções de seta, um tour de JavaScript, funções definidoras, flechas
Funções
Sensibilidade ao caso, o texto de um programa JavaScript
fechamentos, fechamentos-closões
Definindo, definindo funções aninhadas funções
Definindo suas próprias propriedades de função, definindo a sua própria
Propriedades da função
Funções de fábrica, aulas e protótipos
Funções argumentos e parâmetros
tipos de argumento, tipos de argumento
Objeto de argumentos, o objeto de argumentos
Destructar os argumentos da função em parâmetros,
Destructuring Function argumentos em parâmetros-
Destructar os argumentos da função em parâmetros
parâmetros e padrões opcionais, parâmetros opcionais e
Padrões
Visão geral dos argumentos e parâmetros de função
parâmetros de descanso, parâmetros de descanso e comprimento de variável
Listas de argumentos
espalhe o operador para chamadas de função, o operador de spread para
Chamadas de função

listas de argumentos de comprimento variável, parâmetros de descanso e variável-
Listas de argumentos de comprimento
Expressões de definição de função, expressões de definição de função
invocação de funções, criando objetos com novo
Propriedades da função, métodos e construtor, função
Propriedades, 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 comprimento
Propriedade do nome, a propriedade Nome
Propriedade do protótipo, a propriedade do protótipo
Método ToString (), o método ToString ()
Programação funcional
Explorando, programação funcional
Funções de ordem superior, funções de ordem superior
Memomando, memórias
Aplicação parcial de funções, aplicação parcial de
Funções
processamento de matrizes com função, processando matrizes com
Funções
Funciona como namespaces, funciona como espaços de nome
funciona como valores, funciona como valores que definem o seu próprio
Propriedades da função

Global Eval (), Global Eval ()
Objeto global, o objeto global, o objeto global nos navegadores da web
variáveis ??globais, variável e escopo constante
gradientes, cores, padrões e gradientes
Gráficos
3D, gráficos em um <lVAs>
API de tela, gráficos em uma manipulação de <Canvas> -pixel
dimensões e coordenadas de tela, dimensões de tela e
Coordenadas
recorte, recorte
Transformações do sistema de coordenadas, sistema de coordenadas
Exemplo de transformação de transformação
Operações de desenho, operações de desenho de lona
Atributos gráficos, atributos gráficos
Visão geral de, gráficos em um <lVAs>
Caminhos e polígonos, caminhos e polígonos
manipulação de pixels, manipulação de pixels
salvar e restaurar o estado de gráficos, salvar e restaurar
estado gráfico
Gráficos vetoriais escaláveis ??(SVG), SVG: Graphics de vetor escalável-
Criando imagens SVG com JavaScript
maior 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 estáticos
Instância do operador, a instância do operador, construtores, classe
Identidade e instanceof
Literais inteiros, literais inteiros
API de internacionalização
Classes incluídas na API de internacionalização
Comparando cordas, comparando seqüências de cordas
datas e horários de formatação, datas de formatação e tempos
Datas e tempos de formatação
Números de formatação, formatação de números de formatação
Apoio ao Node, a API de internacionalização
texto traduzido, a API de internacionalização
Interpolação, literais de cordas
Intl.DateTimeFormat Classe, Datas de formatação e Formatação de Tempos
Datas e tempos
Classe Intl.NumberFormat, Números de formatação de números de formatação
Expressões de invocação
invocação condicional, invocação condicional, função
Invocação
Invocação de método, expressões de invocação, invocação de método
Visão geral de, invocação de funções
função isfinite (), aritmética em javascript
função 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.

K
palavras -chave
Palavra-chave assíncrona, ASYNC e AWAIT-IMPLEMENTATION Detalhes
Aguarde detalhes-chave, assíncronos e detalhes de implementação
Sensibilidade ao caso, o texto de um programa JavaScript
Palavra-chave da classe, aulas com o exemplo de palavra-chave da classe: um
Classe de números complexos
palavra -chave const, declarações com let e const
Exportar palavras -chave, módulos em ES6
Função Palavra -chave, Definindo funções
função* palavra -chave, geradores
Importar palavra -chave, módulos em ES6
Deixe a palavra -chave, um passeio de JavaScript, declarações com LET e
const, const, let e var
nova palavra -chave, criando objetos com nova invocação construtora
palavras reservadas, palavras reservadas, expressões primárias
Esta palavra -chave, um passeio de JavaScript, expressões primárias,
Invocação de funções
Palavra -chave var, declarações variáveis ??com var, const, let e var
rendimento* palavra -chave, rendimento* e geradores recursivos
Koch Snowflakes, Exemplo de transformação
L

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
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
enquanto loops, enquanto
lvalue, operando e tipo de resultado
M
Magnetômetros, 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 análise
Método Match (), Match ()
Método matchall (), matchall ()
Método Matches (), selecionando elementos com seletores CSS
Função Math.Pow, expressões aritméticas
operações matemáticas, aritmética em JavaScript-aritmético em
JavaScript
Site da MDN, Prefácio
APIs de mídia, APIs de mídia
Memomando, memórias
Gerenciamento de memória, visão geral e definições
Eventos 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 e
Messagechannels, mensagens de origem cruzada com PostMessage (),
Fork ()-tópicos de trabalhadores, canais de comunicação e mensagens de mensagens,
Tipos enumerados e sindicatos discriminados
Messagechannels, PostMessage (), Messageports e
Messagechannels
MessagePort Objects, PostMessage (), MessagePorts e
Messagechannels, canais de comunicação e mensagens de mensagens
mensagens
WebSocket API
recebendo mensagens, recebendo mensagens de um WebSocket
Enviando 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 com
PostMessage (), mensagens de origem cruzada com Postmessage ()
Modelo de execução, modelo de execução do trabalhador
importar código, importar código para um trabalhador
Exemplo de conjunto de Mandelbrot, exemplo: o conjunto de Mandelbrot-
Resumo e sugestões para leitura adicional
módulos, importar código para um trabalhador
Visão geral de fios de trabalhador e mensagens
PostMessage (), MessagePorts e Messagechannels,
PostMessage (), MessagePorts e Messagechannels
Objetos de trabalhador, objetos de trabalhador
Objeto Workerglobalscope, o objeto global em trabalhadores
metaprogramação, metaprogramação
Métodos
Adicionando métodos às classes existentes, adicionando métodos aos existentes
Classes
Métodos de matriz
Aplicação genérica de matrizes
Visão geral dos métodos de matriz
Métodos de classe versus instância, métodos estáticos
Criando, um tour de JavaScript
Definição de termo, funções, invocação de método

encadeamento de método, invocação de método
invocação de método, expressões de invocação, invocação de método-
Invocação do método
Métodos de abreviação, getters, setters e outros formulários de método
Sintaxe abreviada, métodos abreviados
Métodos estáticos, métodos estáticos
Métodos de matriz digitados, métodos de matriz digitados e propriedades
Sign de menos (-)
Operador de subtração, aritmética em JavaScript, aritmética
Expressões
Operador aritmético unário, operadores aritméticos unários
APIs de dispositivo móvel, APIs de dispositivo móvel
módulos
automatizando a modularidade baseada em fechamento, automatizando baseado em fechamento baseado em fechamento
Modularidade
em ES6
importações dinâmicas com importação (), importações dinâmicas com importar()
exportações, es6 exportações
Import.Meta.url, import.meta.url
importações, ES6 importações-ES6 importações
importações e exportações com renomeação, importações e exportações com Renomear
Módulos 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 streaming
Visão geral de, networking
Eventos enviados ao servidor, eventos enviados ao servidor
Websocket API, WebSockets
API xmlHttPrequest (xhr), busca ()
nova palavra -chave, criando objetos com nova invocação construtora
Expressão, classes e construtores do New.Target
newline (\ n), literais de string, sequências de fuga em literais de string
Newlines, semicolons opcionais semicolons opcionais
Usando para formatação de código, o texto de um programa JavaScript
Nó
iteração assíncrona, o loop for/aguardar, o nó é
O nó assíncrono por padrão é assíncrono por padrão
Benefícios da Introdução ao JavaScript, JavaScript do lado do servidor
com nó
Bigint Type, números inteiros de precisão arbitrária com bigint
buffers, buffers
retornos de chamada e eventos, retornos de chamada e eventos no nó
Processos infantis, trabalhando com processos infantis-Fork ()
Definindo recurso de JavaScript do lado do servidor com nó
Eventos e EventEmitter, Eventos e EventEmitter
Manuseio de arquivos, trabalhando com arquivos que trabalham com diretórios
diretórios, trabalhando com diretórios

metadados do arquivo, metadados do arquivo
Strings de modo de arquivo, escrevendo arquivos
Operações de arquivo, operações de arquivo
Visão geral de, trabalhando com arquivos
caminhos, descritores de arquivos e trabalhos de arquivo, caminhos, arquivo
Descritores e trabalhos de arquivo
Lendo arquivos, leitura de arquivos
Escrevendo arquivos, escrevendo arquivos
Clientes e servidores HTTP, clientes HTTP e servidores-http
Clientes e servidores
Instalando, explorando JavaScript, JavaScript do lado do servidor com
Nó
API INTL, a API de internacionalização
Módulos em, módulos em módulos de nó no nó na web
Servidores e clientes de rede não http, rede não http
Servidores e clientes
Paralelismo com, Node é assíncrono por padrão
Detalhes do processo, processo, CPU e detalhes do sistema operacional
Programação básica, básico de programação do nó-o nó
Gerente de pacotes
Argumentos da linha de comando, argumentos da linha de comando e
Variáveis ??de ambiente
Saída do console, saída do console
variáveis ??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 fluxos
iteração 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 porportes 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 objeto
Métodos de objeto, métodos de objeto-método tojson ()
Visão geral de, um tour de javascript, visão geral e definições
Visão geral e definições
consulta e configuração de propriedades, consultas e configurações de propriedade-
Erros de acesso à propriedade
objetos serializando, objetos serializando
Propriedades de teste, propriedades de teste
Evento OnMessage, recebendo mensagens de um WebSocket, trabalhador
Objetos-objeto global em trabalhadores, PostMessage (), MessagePorts,
e Messagechannels, mensagens de origem cruzada com Postmessage ()
operadores
operadores aritméticos, um tour de JavaScript, aritmético em
JavaScript-aritmético em JavaScript, Expressões aritméticas-
Operadores bitwise
operadores de tarefas, designação de expressões de atribuição com
Operação
operadores binários, número de operando
Operadores de comparação, operadores de comparação
Operadores de igualdade e desigualdade, igualdade e desigualdade
Operadores
Operadores de igualdade, um tour pelo JavaScript
formando expressões com, um tour de JavaScript, expressões e
Operadores
Operadores lógicos, um passeio de JavaScript, expressões lógicas-

Lógico não (!)
operadores diversos
aguardar operador, o operador aguardar
operador de vírgula (,), o operador de vírgula (,)
operador condicional (? :), o operador condicional (? :)
Excluir operador, o operador de exclusão
operador primeiro definido (??), primeiro definido (??)
TIPEOF OPERADOR, O TIPEOF OPERADOR
operador vazio, o operador vazio
Número de operandos, número de operandos
operando e tipo de resultado, operando e tipo de resultado
Associatividade do Operador, Associatividade do Operador
precedência do operador, precedência do operador
Efeitos colaterais do operador, efeitos colaterais do operador
Ordem de avaliação, ordem de avaliação
Visão geral da visão geral do operador
Operadores de pós -fix, semicolons opcionais
Operadores relacionais, um tour de JavaScript, expressões relacionais-
A instância do operador
Tabela de, Visão geral do operador
operadores ternários, número de operando
Semicolons opcionais, semicolons opcionais semicolons opcionais

Erro ao traduzir esta página.

Erro ao traduzir esta página.

pixels, coordenadas de documentos e coordenadas de viewport, pixel
Manipulação
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 com
pushState ()-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 ??e
Referências de objetos mutáveis
Tipo de número, números e horários
Visão geral e definições, visão geral e definições
Tipo de string, literais de modelo marcado com texto
PrintProps () função, declarações de função
Campos privados, campos públicos, privados e estáticos
procedimentos, funções
programas
Manuseio de erros, erros de programa
Execução de JavaScript, execução de programas JavaScript- Client-
Linha do tempo do JavaScript lateral
Modelo de encadeamento do lado do cliente, encadeamento JavaScript do lado do cliente
modelo
Linha do tempo do lado do cliente, linha do tempo JavaScript do lado do cliente
Entrada e saída, entrada e saída de programas
Aplicativos da Web progressivos (PWAs), aplicativos da Web progressivos e serviço
Trabalhadores
Promessa cadeias, promessas, encadeamento de promessas promessas
Função promey.all (), promessas em paralelo
Promessas
encadear promessas, encadear promessas de promessas
lidar com erros com, lidar com erros com promessas, mais sobre
Promessas e erros-a captura e finalmente métodos

Erro ao traduzir esta página.

herdando, herança
nomeação, símbolos, introdução a objetos, símbolos como propriedade
Nomes
propriedades não herdadas, introdução aos objetos
erros de acesso à propriedade, erros de acesso à propriedade
Expressões de acesso à propriedade, expressões de acesso à propriedade
Atributos da propriedade, introdução a objetos, atributos de propriedade-
Atributos da propriedade
Descritores de propriedades, atributos de propriedade
Propriedade Getters e Setters, Property Getters e Setters
Consulta e cenário, consulta e configuração de propriedades-Property
Erros de acesso
Testes, propriedades de teste
Propriedades da matriz digitada, métodos de matriz digitados e propriedades
Método PropertyInumerable (), Propriedades de teste
herança prototípica, introdução a objetos, herança
Cadeias de protótipo, protótipos
protótipos, protótipos, herança, propriedade do protótipo, classes
e protótipos, o atributo do protótipo
invariantes de procuração, invariantes de procuração
Objetos de proxy, objetos de proxy-proxy invariantes
números de pseudorandom, criptografia e APIs relacionadas
Campos públicos, campos públicos, privados e estáticos

Empurre API, aplicativos da Web progressivos e trabalhadores de serviço
método push (), um tour de javascript, pilhas e filas com push (),
pop (), shift () e não dividido ()
Q
Método quadraticcurveto (), curvas
Método querySelector (), selecionando elementos com seletores CSS
Método querySelectorAll (), selecionando elementos com seletores CSS
Marcas de citação
Citações duplas ("), literais de cordas
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 Regexp
Propriedades regexp, propriedades regexp
método test (), teste ()
Tipo de regexp, visão geral e definições, correspondência de padrões, padrão
Combinando 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 operador
Operadores relacionais, um tour pelo JavaScript
Método Substituir (), trabalhando com Strings
requer () função, importações de nó
palavras reservadas, palavras reservadas, expressões primárias
Parâmetros de descanso, parâmetros de descanso e listas de argumentos de comprimento variável
Retornar declarações, retornar
Retornar valores, funções
Método de retorno (), ?fechando? um iterador: o método de retorno, o
Métodos de retorno () e Throw () de um gerador
Método reverso (), um tour de javascript, reverse ()
Erros de arredondamento, ponto flutuante binário e erros de arredondamento
S

Erro ao traduzir esta página.

Política da mesma origem, a política da mesma origem
Recursos da plataforma da web para investigar, segurança
Semicolon (;), semicolons opcionais semicolons opcionais
Informações sensíveis, armazenamento
API do sensor, APIs de dispositivo móvel
serialização, serializando objetos, serialização e análise JSON,
Gerenciamento de história com pushState ()
Eventos enviados ao servidor, eventos de eventos de servidor-servidor-servidores
JavaScript do lado do servidor, JavaScript em navegadores da Web, servidor
JavaScript com nó
Trabalhadores de serviço, aplicativos da Web progressivos e trabalhadores de serviço
Propriedade da sessão, LocalStorage e SessionStorage
Set Class, para/de com set e mapa, a classe Set-a classe Set
Defina objetos, visão geral e definições
SET () construtor, a classe Set
função setInterval (), temporizadores
conjuntos e mapas
Definição de conjuntos, a classe set
Classe de mapa, a classe de mapa-a classe de mapa
Visão geral de, sets e mapas
Classe definida, a classe Set-The Set Class
Classes fracas e fracas, fraco e fraco e fraco
Métodos 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
declarações condicionais, declarações, comutação condicionais
Estruturas de controle, um passeio de JavaScript-um passeio de JavaScript,
Declarações
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-opcionais
Semicolons
Jogue declarações, jogue
Experimente/Finalmente as instruções, tente/capture/finalmente-try/Catch/Finalmente
declarações de rendimento, rendimento, o valor de uma expressão de rendimento
Campos estáticos, campos públicos, privados e estáticos
Métodos estáticos, métodos estáticos
Armazenamento, IndexedDB de armazenamento
Cookies, biscoitos
IndexedDB, indexedDB
LocalStorage e SessionStorage, LocalStorage e SessionStorage
Visão geral de, armazenamento
Segurança e privacidade, armazenamento
fluxos (nó), modo de fluxos de fluxos
iteração 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
operador estrito de igualdade (===)
valores booleanos, valores booleanos

Erro ao traduzir esta página.

Matriz para conversões de string, matriz para conversões de string
caracteres e pontos de código, texto
Métodos para correspondência de padrões
Match (), Match ()
matchall (), matchall ()
substituir (), substituir ()
pesquisa (), métodos de string para correspondência de padrões
Split (), Split ()
Visão geral de, texto
Literais de cordas, literais de cordas
cordas como matrizes, cordas como matrizes
trabalhando com
Acessando caracteres individuais, trabalhando com strings
API para, trabalhando com strings
Comparando, trabalhando com cordas, comparando strings
concatenação, trabalhando com strings
Determinando o comprimento, trabalhando com strings
imutabilidade, trabalhando com cordas
Algoritmo de clone estruturado, gerenciamento de história com pushState ()
subarrays, subarrays with slice (), splice (), preench () e copywithin ()
subclasses
hierarquias 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 super
sub -rotinas, funções
Operador de subtração (-), aritmético em JavaScript
pares substitutos, texto
SVG (ver Graphics Scalable Vector (SVG))
Switch Declarações, switch-switch
Symbol.asynciterator, símbolo.iterator e símbolo.asynciterator
Symbol.hasinsance, símbolo.hasinstance
Symbol.iscoNcatsPreadable, symbol.iscoNcatsPreadable
Symbol.iterator, símbolos conhecidos
Symbol.spécies, símbolo.species-symbol.species
Symbol.Toprimitive, símbolo.Toprimitivo
Symbol.ToStringTag, symbol.ToStringTag
Symbol.UNSCOPABLES, SYMBOL.UNSCOPABLES
Símbolos
Definição de extensões de linguagem, visão geral e definições
nomes de propriedades, símbolos, símbolos como nomes de propriedades

símbolos conhecidos, símbolos conhecidos
Execução de scripts síncronos, quando os scripts são executados: assíncronos e adiados
sintaxe
Estruturas de controle, um passeio de JavaScript-um passeio de JavaScript
Declarando variáveis, um tour pelo JavaScript
Comentários em inglês, um passeio de JavaScript, um tour de
JavaScript
Operadores de igualdade e relacional, um tour pelo JavaScript
expressões
formando com operadores, um tour pelo JavaScript
Expressão inicializadora, um tour de JavaScript
estendido para literais de objetos, sintaxe literal de objeto estendido-
Propriedade Getters and Setters
funções, um tour de javascript
Estrutura lexical, estrutura lexical-sumário
Sensibilidade ao caso, o texto de um programa JavaScript
Comentários, comentários
Identificadores, o texto de um Programa JavaScript-Identificadores e
Palavras reservadas
quebras de linha, o texto de um programa JavaScript
Literais, literais
palavras reservadas, palavras reservadas, expressões primárias
Semicolons, 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 texto
Normalização, normalização unicode
Usando com o nó, olá mundial
Estilos de texto, estilos de texto
.then () método, usando promessas, encadeamento de promessas, mais sobre promessas e 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 ToisSotring (), Formatação e Parsing Data Strings, JSON
Personalizações
Método Tojson (), o método Tojson (), JSON Customizations
TOLOCALEDATESTRING () Método, formatação e análise de datas de análise,
Datas e tempos de formatação
Método tolocalestring (), o método tolocalestring (), matriz para string
Conversões, formatação e análise de datas de análise
TOLOCALETIMESTRING () Método, formatação e análise de datas de análise,
Datas e tempos de formatação
Ferramentas e extensões, ferramentas de JavaScript e extensões-imensas
Tipos e sindicatos discriminados
Bundling de código, agrupamento de código
Formatação de javascript com formatação mais bonita e javascript com
Mais bonito
Extensão da linguagem JSX, JSX: Expressões de marcação em JavaScript-
JSX: Expressões de marcação em JavaScript
linhando com Eslint, linhando com Eslint
Visão geral de ferramentas e extensões JavaScript
Gerenciamento de pacotes com NPM, gerenciamento de pacotes com NPM
Transpilação com Babel, transpilação com Babel
Tipo de verificação com fluxo, verificação de tipo com o fluxo, enumerado
Tipos e sindicatos discriminados
Tipos de matriz, tipos de matriz
Tipos de classe, tipos de classe
tipos enumerados e sindicatos discriminados, enumerados

Tipos e sindicatos discriminados
Tipos de funções, tipos de função
Instalando e executando, instalando e executando 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
Teste de unidade com JEST, teste de unidade com JEST
Método toprecision (), conversões explícitas
método tostring (), valores booleanos, conversões explícitas, o
métodos tostring () e valueof (), o operador +, igualdade com tipo
conversão, o método tostring (), o método tostring (), formatação
e strings de data de análise
TOTIMEstring () Método, formatação e análise de datas de análise
Método touppercase (), trabalhando com strings
Método toutcString (), formatação e análise de datas de análise
Transformações, o sistema de coordenadas transforma a transformação
exemplo
Método 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
e 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 explícitas
Dados financeiros e científicos, conversões explícitas
conversões implícitas, conversões explícitas
Objeta -se a conversões primitivas
algoritmos para, objeta-se a conversões primitivas, objeto a
Algoritmos de conversão primitiva
Conversões objeto a boolean e objeto a booleano
Conversões de objeto para número, objeto a número
Conversões de objetos para cordas, objeto para cordão
Conversões especiais de operador de caso, operador de caso especial
conversões
métodos tostring () e valueof (), o tostring () e
Métodos Valueof ()
Visão geral das conversões de tipo
matrizes digitadas
Criando, criando matrizes digitadas
DataView e Endianness, DataView e Endianness
Métodos e propriedades, métodos de matriz digitados e propriedades
Visão geral de matrizes digitadas e dados binários
versus matrizes regulares, matrizes
Compartilhando entre threads, compartilhando matrizes digitadas entre threads
Tipos de matriz digitados, tipos de matriz digitados
Usando, usando matrizes digitadas

Erro ao traduzir esta página.

subfluxo, aritmética em javascript
sublinhado (_), identificadores e palavras reservadas
sublinhou, como separadores numéricos (_), literais de ponto flutuante
UNESCAPE () FUNÇÃO, FUNÇÕES DE URL LEGADO
Evento de rejeição não entrega, erros de programa
Conjunto de caracteres unicode
Sequências de fuga, sequências de escape unicode, sequências de escape
em literais de cordas
Strings JavaScript, texto
Normalização, normalização unicode
Visão geral de, unicode
Combinação de padrões, classes de personagens
caracteres espaciais, o texto de um programa JavaScript
Teste de unidade, teste de unidade com JEST
Método de Netfift (), pilhas e filas com push (), pop (), shift () e
NIFT ()
URL APIs, URL APIS-LEGACY URL Functions
Use diretiva estrita
Aplicação padrão do modo rigoroso, classes com a palavra -chave de classe,
Módulos em ES6, JavaScript Módulos na Web
Excluir operador e, o operador de exclusão
EVALL () Função, EVAL
declarações de função, declarações de função

Erro ao traduzir esta página.

Definições
tipos de, um tour de javascript
Palavra -chave var, declarações variáveis ??com var, const, let e var
varargs, parâmetros de descanso e listas de argumentos de comprimento variável
funções variáveis ??de arity, parâmetros de descanso e comprimento de variável
Listas de argumentos
variáveis
Sensibilidade ao caso, o texto de um programa JavaScript
declaração e atribuição
declarações com let and const, declarações com let e
const-declarações e tipos
declarações com VAR, declarações variáveis ??com VAR
atribuição de destruição, destruição
Atribuição de destruição
Visão geral de, um tour de JavaScript
variáveis ??não declaradas, declarações variáveis ??com VAR
Definição de termo, declaração e atribuição variáveis
declarações variáveis ??e içadas com VAR
Nomeação, palavras reservadas
Visão geral de, tipos, valores e variáveis-overview e
Definições
escopo de escopo variável e constante, funções aninhadas
Funções variádicas, parâmetros de descanso e argumento de comprimento variável
Listas

fluxos de vídeo, APIs de mídia
Viewport, coordenadas de documentos e coordenadas de viewport, viewport
Tamanho, tamanho do conteúdo e posição de rolagem
operador vazio, o operador vazio
C
Classe de mapa fraco, mapa fraco e conjunto fraco
Classe fraca classe, mapa fraco e conjunto fraco
API de autenticação na web, criptografia e APIs relacionadas
ambiente de host de navegador da web
APIs assíncronas, eventos
APIs de áudio, APIs de áudio-a API Webaudio
Benefícios do JavaScript, JavaScript em navegadores da Web
API de tela, gráficos em uma manipulação de <Canvas> -pixel
dimensões e coordenadas de tela, dimensões de tela e
Coordenadas
recorte, recorte
Transformações do sistema de coordenadas, sistema de coordenadas
Exemplo de transformação de transformação
Operações de desenho, operações de desenho de lona
atributos gráficos, atributos gráficos, salvando e restaurando
estado gráfico
Visão geral de, gráficos em um <lVAs>
Caminhos e polígonos, caminhos e polígonos

manipulação de pixels, manipulação de pixels
Documentar geometria e rolagem, geometria de documentos e
Tamanho da Scrolling-ViewPort, tamanho de conteúdo e posição de rolagem
Pixels CSS, coordenadas de documentos e viewport
Coordenadas
determinando elemento em um ponto, determinando o elemento em um
Apontar
Documentar coordenadas e coordenadas de viewport, documento
Coordena e coordenadas de viewport
consulta geometria de elementos, consultando a geometria de
um elemento
rolando, rolando
Tamanho da viewport, tamanho do conteúdo e posição de rolagem, viewport
Tamanho, tamanho do conteúdo e posição de rolagem
Eventos, eventos para eventos personalizados
Despacha eventos personalizados, despachando eventos personalizados
Cancelamento de eventos, cancelamento de eventos
categorias de eventos, categorias de eventos
Invocação de manipulador de eventos, invocação de manipulador de eventos
propagação de eventos, propagação de eventos
Visão geral dos eventos
Registrando manipuladores de eventos, registrando manipuladores de eventos
APIs legadas, JavaScript em navegadores da web
localização, navegação e história, localização, navegação e

Gerenciamento de história da história com pushState ()
História de navegação, história de navegação
Carregando novos documentos, carregando novos documentos
Visão geral de localização, navegação e história
Exemplo de conjunto de Mandelbrot, exemplo: o Set-Summary Mandelbrot
e sugestões para leitura adicional
navegadores com reconhecimento de módulos, módulos JavaScript na web
Networking, Networking-Protocol Negociação
Método Fetch (), Fetch ()
Visão geral de, networking
Eventos enviados ao servidor, eventos enviados ao servidor
Websocket API, WebSockets
Visão geral de, JavaScript em navegadores da web
Gráficos vetoriais escaláveis ??(SVG), SVG: Graphics de vetor escalável-
Criando imagens SVG com JavaScript
Criando imagens SVG com JavaScript, criando imagens SVG
com javascript
Visão geral de, SVG: gráficos vetoriais escaláveis
Scripts SVG, Scripts SVG
SVG em HTML, SVG em HTML
Scripts CSS, Scripts CSS-CSS Animações e eventos
Estilos CSS comuns, Scripts CSS
Estilos computados, estilos computados

Animações e eventos CSS, animações e eventos CSS
Classes CSS, aulas CSS
Estilos embutidos, estilos embutidos
Convenções de nomeação, estilos em linha
folhas de estilo de script, folhas de estilo de script
Documentos de script, documentos de script-Exemplo: gerando um
Índice
estrutura de documentos e travessia, estrutura de documentos e
Traversal
Geração dinamicamente tabelas de conteúdo, exemplo:
Gerando uma tabela de índice
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
Armazenamento, IndexedDB de armazenamento
Cookies, biscoitos
IndexedDB, indexedDB
LocalStorage e SessionStorage, LocalStorage e
SessionStorage
Visão geral de, armazenamento
Segurança e privacidade, armazenamento

Componentes da Web, componentes da Web-Exemplo: A <search-box>
Componente da Web
elementos personalizados, elementos personalizados
Nós de documentário, usando componentes da web
Modelos HTML, modelos HTML
Visão geral de, componentes da web
Exemplo de caixa de pesquisa, exemplo: uma web <search-box>
Componente
Shadow Dom, Shadow Dom
Usando, usando componentes da web
Recursos da plataforma da web para investigar
APIs binárias, APIs binárias
APIs de criptografia e segurança, criptografia e relacionadas
APIs
eventos, eventos
HTML e CSS, HTML e CSS
APIs de mídia, APIs de mídia
APIs de dispositivo móvel, APIs de dispositivo móvel
APIs de desempenho, desempenho
Aplicativos da Web progressivos e trabalhadores de serviço, progressivos
Aplicativos da Web e trabalhadores de serviço
Segurança, segurança
WebAssembly, WebAssembly

Erro ao traduzir esta página.

WebSocket API
criando, conectando e desconectando websockets, criação,
Conectando e desconectando WebSockets
Visão geral de, websockets
Negociação de protocolo, negociação de protocolo
recebendo mensagens, recebendo mensagens de um WebSocket
Enviando mensagens, enviando mensagens sobre um WebSocket
enquanto loops, enquanto
com declarações, declarações diversas
API do trabalhador
Mensagens de origem cruzada, mensagens de origem cruzada com
PostMessage ()
erros, erros em trabalhadores
Modelo de execução, modelo de execução do trabalhador
importar código, importar código para um trabalhador
Exemplo de conjunto de Mandelbrot, exemplo: o Set-Summary Mandelbrot
e sugestões para leitura adicional
módulos, importar código para um trabalhador
Visão geral de fios de trabalhador e mensagens
PostMessage (), MessagePorts e Messagechannels,
PostMessage (), MessagePorts e Messagechannels
Objetos de trabalhador, objetos de trabalhador
Objeto Workerglobalscope, o objeto global em trabalhadores

atributo gravável, introdução a objetos, atributos de propriedade
X
API xmlHttPrequest (xhr), busca ()
XSS (scripts cross-sites), a política da mesma origem
Y
declarações de rendimento, rendimento, o valor de uma expressão de rendimento
rendimento* palavra -chave, rendimento* e geradores recursivos
Z
zero
zero negativo, aritmética em javascript
zero positivo, aritmética em javascript
Matrizes baseadas em zero, matrizes

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Parque Nacional, em Java, Indonésia.Esta estratégia parece estar ajudando
Garanta 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 um
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;o
A fonte do cabeçalho é um miríade de Adobe condensado;e a fonte do código é Dalton
Ubuntu Mono de Maag.