Louvando
JavaScript: O Guia Definitivo
, Assim,
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: 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:
Primeiro lançamento
Ver
http://oreilly.com/catalog/errata.csp?isbn=9781491952023
para
Detalhes da liberação.

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

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

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
Suplementar
material (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 exige
permissão.
Agradecemos, mas geralmente não exigem, atribuição.Uma atribuição
Geralmente inclui o título, autor, editor e ISBN.Por exemplo:
“
JavaScript: O Guia Definitivo
, Sétima edição, por 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,
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 .

E-mail

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
Observe -nos no 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 nndo 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 (ag
ora 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 padronizad
o
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.
Esse
O livro trata o ES5 como a linha de base da compatibilidade e não discute mais versões anteriores
do idioma.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 ECMAScri
pt 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-E
S5)
versões.Para manter a compatibilidade com versões anteriores, não é possível remover recursos legado
s, não
importa o quão falho.Mas
no ES5 e posterior, os programas podem optar por JavaScript's
modo rigoroso
em que a
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 frequente
mente implicitamente
Invoca o modo rigoroso.Por exemplo, se você usar o ES6
aula
palavra -chave ou criar um módulo ES6, então
Todo o código dentro da classe ou módulo é automaticamente rigoroso, e os recursos antigos e defeitu
osos não são
disponível nesses contextos.Este livro abordará os recursos legados do JavaScript, mas é cuidadoso p
ara
apontar que eles não estão disponíveis em
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
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
Exemplo 1-1
no final do
capítulo antes de seguir em frente.)

1.1 Explorando JavaScript

Quando
Aprendendo 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
retratado em
Figura 1-1
), o que geralmente é bastante conveniente.

Figura 1-1.
O JavaScript Console nas ferramentas de desenvolvedor do Firefox
Outro
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 começar 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 está 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 a função
console.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
Olá.js
Arquivo que contém esta linha de código:
console.log ("Hello World!");
e executar o arquivo com
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
Olá.html
, e colocar
este texto nele:
<script src = "hello.js"> </script>
Em seguida, carregue
Olá.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
Esse
a 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:
Capítulo 2
explica coisas como JavaScript
Comentários, semicolons e o conjunto de caracteres Unicode.
Capítulo 3
inicia
Para ficar mais interessante: explica as variáveis ■■JavaScript e os valores

Você pode atribuir a essas variáveis.
Aqui está
algum código de amostra 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:
deixar

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"
;

```
// 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.
deixar

livro

=

{

// Os objetos estão fechados em Curly

aparelho ortodôntico.

tópico
:

"JavaScript"
, Assim,

// A propriedade "tópico" tem valor

"JavaScript."

edição
:

7

// A propriedade "edição" tem

Valor 7
};

// A cinta encaracolada marca o fim

do objeto.
// Acesse as propriedades de um objeto com.ou []:
livro
.
tópico

// => "JavaScript"
livro
[[
"edição"
]

// => 7: Outra maneira de acessar

Valores da propriedade.
livro
.
autor

=
```

```
deixar

pontos

=

[[

// Uma matriz com 2 elementos.

{
x
:

0
, Assim,

y
:

0
},

// Cada elemento é um objeto.

{
x
:

1
, Assim,

y
:

1
}
];
deixar

dados

=

{

// um objeto com 2 propriedades

Trial1
:

[[
1
, Assim,
2
],

[[
3
```

usar
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,

Você pode atribuir a essas variáveis.
Aqui está
algum código de amostra 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:
deixar

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"
;

retornar

x

*

x
;

// Calcule o valor da função
};

// semicolon marca o fim do

atribuição.
quadrado
(
Plus1
(
y
))

// => 16: Invoque duas funções em

uma expressão
Em
ES6 e mais tarde, há uma sintaxe abreviada para definir funções.
Esse
Usos conciso de sintaxe
=>
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

distância entre pontos

deixar

P1

=

esse
[[
0
];

// Primeiro elemento da matriz somos

invocou

deixar

P2

=

esse
[[
1
];

// segundo elemento do "isto"

objeto

deixar

um

=

P2
.
x
-
P1
.
x
;

// diferença em coordenadas x

deixar

b

=

P2
.
y
-

```
soma

+=

x
;

// Adicione o valor do elemento ao

soma.

}

// Este é o fim do loop.

retornar

soma
;

// retorna a soma.
}
soma
(
primos
)

// => 28: Soma dos primeiros 5

Prima 2+3+5+7+11
função

fatorial
(
n
)

{

// uma função para calcular

fatoriais

deixar

produto

=

1
;

// Comece com um produto de 1

enquanto
(
n

>
```

Figura 1-1.
O JavaScript Console nas ferramentas de desenvolvedor do Firefox
Outro
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 começar um

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 são
disponível para todos
Programas JavaScript.Isso inclui dados importantes
Estruturas como
Mapas e conjuntos, uma classe de expressão regular para textual
padrão
Combinação, funções para serializar dados de JavaScript
estruturas e
muito mais.
Capítulo 12,
Iteradores e geradores
Explica como o
para/de
Loop funciona e como você pode
fazer o seu
próprios aulas iteráveis ■■com
para/de
.Também abrange
gerador
funções e o
colheita
declaração.
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
o
assíncrono
e
aguarde
palavras -chave.
Embora o Javascript Core
A linguagem não é assíncrona,
APIs assíncronas são o padrão em
navegadores da web e nó,
E este capítulo explica as técnicas
Para trabalhar com aqueles
APIs.
Capítulo 14,
Metaprogramação
Apresenta vários recursos avançados de
JavaScript que pode ser
De interesse para os 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 web
Os navegadores executam o código JavaScript e abrange mais
importante de
As muitas APIs definidas pelos navegadores da Web.Isso 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 mais
produtivo
programador.
1.4 Exemplo: Frequência do personagem
Histogramas
Esse
O 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
aula

DefaultMap

estende -se

Mapa

{

construtor
(
DefaultValue
)

{

super
();

// Invoque a Superclass

construtor

esse
.
DefaultValue

=

DefaultValue
;
```

```
}
}
// Esta classe calcula e exibe histogramas de frequência de carta
aula

Histograma

{

construtor
()

{

esse
.
letterCounts

=

novo

DefaultMap
(
0
);

// mapa de

cartas para contar

esse
.
Totalletters

=

0
;

// Quantos

letras em tudo

}

// Esta função atualiza o histograma com as letras de

texto.

adicionar
(
texto
)

{

// remova o espaço em branco do texto e converta para
```

```
diferir

retornar

b
[[
1
]

-

um
[[
1
];

// classificar por maior

contar.

}

});

// converte as contagens em porcentagens

para
(
deixar

entrada

de

entradas
)

{

entrada
[[
1
]

=

entrada
[[
1
]

/

esse
.
Totalletters
*
100
;
```

```
// Faz um objeto histograma a partir de entrada padrão, depois imprime

o histograma.
histogramfromstdin
().
então
(
histograma

=>

{

console
.
registro
(
histograma
.
ToString
());

});
```

1.5 Resumo

Esse
O 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
Pegue a palavra
definitivo
no título deste livro seriamente, 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
O
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.Esse
significa essa linguagem
palavras -chave, variáveis, nomes de funções e outros
identificadores
deve sempre
ser digitado com uma capitalização consistente das letras.O
enquanto
A palavra -chave, por exemplo, deve ser digitada "while", não "enquanto" ou
"ENQUANTO."De forma similar,
on-line
, Assim,
On-line
, Assim,
On-line
, e
ON-LINE
são
quatro nomes variáveis ■■distintos.

JavaScript
Ignora espaços que aparecem entre os tokens nos programas.Para
a maior parte, 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 do código e
entender.
Além do personagem espacial regular (
\ u0020
), JavaScript também
reconhece
Guias, variadas
Caracteres de controle ASCII 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
JavaScript
Suporta dois estilos de comentários.Qualquer texto entre um
//
e o fim de uma linha é tratado como um comentário e é ignorado por
JavaScript.Qualquer
texto entre os personagens
/*
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 JavaScript legal
Comentários:

```
// 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 estão todos
Literais:

```
12

// o número doze
1.2

// o número número um dois
"Hello World"

// Uma sequência de texto
'Oi'

// Outra string
verdadeiro

// Um ■■valor booleano
falso

// o outro valor booleano
nulo

// ausência de um objeto
```

Detalhes completos sobre literais numéricos e de cordas aparecem em
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
O
As palavras a seguir fazem parte da linguagem JavaScript.Muitos
de
estes (como
se
, Assim,
enquanto
, e
para
) são palavras -chave reservadas que
deve
não ser usado como nomes de constantes, variáveis, funções ou classes
(no entanto
Todos eles podem ser usados ■■como nomes de propriedades dentro de um
objeto).Outros (como
de
, Assim,
de
, Assim,
pegar
, e
definir
) são usados ■■em limitado
contextos sem ambiguidade sintática e são perfeitamente legais como
identificadores.Ainda outras palavras -chave (como
deixar
) não pode ser totalmente reservado
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.(
deixar
pode ser usado como um nome 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
de
, Assim,
definir
, e
alvo
, Assim,
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

público
Por razões históricas,
argumentos
e
aval
não são permitidos como
identificadores em certas circunstâncias e são melhor evitados inteiramente.
2.5 Unicode
JavaScript
Os programas 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 cartas, 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

π

=

3.14
;
const

si

=

verdadeiro
;
2.5.1 Sequências de Escape Unicode
Alguns
hardware e software de computador não podem exibir, entrada 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.Esses
As 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:
deixar

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
.
registro
(
"\ 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 que combina marca
\ 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 normalizado e faz
não
fazer qualquer
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, JavaScript usa o ponto e vírgula (
;
)
para separar declarações (ver
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 encurralada:
}
.) Muitos
Programadores JavaScript (e o código neste livro) usam semicolons para
marque explicitamente os fins das declarações, mesmo onde elas não estão
obrigatório.

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:
um

=

3
;
b

=

4
;
Escrito da seguinte forma, no entanto, é necessário o primeiro ponto de vírgula:
um

=

3
;

b

=

4
;
Observação
Esse 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:
deixar

um
um
=
3
console
.
registro
(
um
)
JavaScript interpreta este código como este:
deixar

um
;

um

=

```
soma

+=

x
;

// Adicione o valor do elemento ao

soma.

}

// Este é o fim do loop.

retornar

soma
;

// retorna a soma.
}
soma
(
primos
)

// => 28: Soma dos primeiros 5

Prima 2+3+5+7+11
função

fatorial
(
n
)

{

// uma função para calcular

fatoriais

deixar

produto

=

1
;

// Comece com um produto de 1

enquanto
(
n

>
```

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
retornar
, Assim,
lançar
, Assim,
colheita
, Assim,
quebrar
, e
continuar
declarações (ver
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

verdadeiro
;
Isso significa que você não deve inserir uma quebra de linha entre
retornar
, Assim,
quebrar
, ou
continuar
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 o
++
e
- -
operadores (
§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.O
Terceira exceção envolve
funções definidas usando sintaxe concisa de "Arrow": o
=>
seta em si
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

Computador
Os programas 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

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 do numérico (
§3.2
) e string (
§3.3
) Tipos em JavaScript.
Booleanos estão cobertos em
§3.4
.
Os valores especiais de JavaScript
nulo
e
indefinido
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 em
§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 tipo
objeto
) 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
, está coberto em
§3.7
, Assim,
mas a cobertura mais geral e mais detalhada dos objetos está em
Capítulo 6
.
Um
O 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 de
Capítulo 7
.
Além de objetos e matrizes básicos, JavaScript define uma série de
Outros tipos de objetos úteis.UM
O objeto SET 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
O tipo regexp representa padrões textuais e permite
Combinação sofisticada, pesquisando e substituindo operações em strings.
O
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 executa automático
Coleção 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.)
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
um
, por exemplo,
Nós não passamos
um
para um
organizar()
função.
Em vez disso, invocamos o
organizar()
método de
um
:
um
.
organizar
();

// A versão orientada ao objeto (a).

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 são
disponível para todos
Programas JavaScript.Isso inclui dados importantes
Estruturas como
Mapas e conjuntos, uma classe de expressão regular para textual
padrão
Combinação, funções para serializar dados de JavaScript
estruturas e
muito mais.
Capítulo 12,
Iteradores e geradores
Explica como o
para/de
Loop funciona e como você pode
fazer o seu
próprios aulas iteráveis ■■com
para/de
.Também abrange
gerador
funções e o
colheita
declaração.
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
o
assíncrono
e
aguarde
palavras -chave.
Embora o Javascript Core
A linguagem não é assíncrona,
APIs assíncronas são o padrão em
navegadores da web e nó,
E este capítulo explica as técnicas
Para trabalhar com aqueles
APIs.
Capítulo 14,
Metaprogramação
Apresenta vários recursos avançados de
JavaScript que pode ser
De interesse para os 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 web
Os navegadores executam o código JavaScript e abrange mais
importante de
As muitas APIs definidas pelos navegadores da Web.Isso é de longe
o mais longo

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
deixar
(ou com
var
no código JavaScript mais antigo).JavaScript
Constantes e variáveis ■■são
sem topo
: 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
JavaScript
Tipo numérico primário, 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
operadores bitwise descritos em
Capítulo 4
) são realizados com 32 bits
Inteiros.Se você precisar representar exatamente números inteiros maiores, veja
§3.2.5
.
Quando
um número aparece diretamente em um programa JavaScript, 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 de uma série 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
Em
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
Literais de ponto flutuante
pode 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.
Ponto flutuante
Os literais 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 × 10²³
1.4738223
E
-
32

// 1.4738223 × 10
■
32
Separadores em literais numéricos
Você
pode usar sublinhados em literais numéricos para quebrar literais longos em pedaços que são mais fác
eis de
ler:
deixar

bilhão

=

1
_000_000_000
;

// ressalta como milhares de separadores.
deixar

bytes

=

0x89
_Ab_cd_ef
;

3.2.3 aritmética em javascript
JavaScript
Os programas funcionam com números usando os operadores aritméticos
.que o idioma fornece.Esses
incluir
+
para adição,
-
para
subtração,
*
para multiplicação,
/
para divisão e
%
para módulo
(restante após a divisão).ES2016
adiciona
**
para exponenciação.Completo
detalhes sobre esses e outros operadores podem ser encontrados em
Capítulo 4
.
Em
Além desses operadores aritméticos básicos, JavaScript suporta
operações matemáticas mais complexas através de um conjunto de funções e
constantes definidas como propriedades do
Matemática
objeto:
Matemática
.
prisioneiro de guerra
(
2
, Assim,
53
)

// => 9007199254740992: 2 para o

potência 53
Matemática
.
redondo
(.
6
)

// => 1,0: rodada para o mais próximo

Inteiro
Matemática
.
CEIL
(.
6
)

// => 1.0: Recurso para um número inteiro

ES6
define mais funções no
Matemática
objeto:
Matemática
.
cbrt
(
27
)

// => 3: raiz de cubo
Matemática
.
Hypot
(
3
, Assim,

4
)

// => 5: raiz quadrada da soma dos quadrados de

todos os argumentos
Matemática
.
log10
(
100
)

// => 2: Logaritmo Base-10
Matemática
.
log2
(
1024
)

// => 10: logaritmo base-2
Matemática
.
log1p
(
x
)

// log natural de (1+x);preciso para muito

pequeno x
Matemática
.
EXPM1
(
x
)

// math.exp (x) -1;o inverso 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 constantes globais
Infinidade
e
Nan
para segurar o
infinito positivo e valor não um número, e esses valores também são
disponível como propriedades do
Número
objeto:
Infinidade

// um número positivo muito grande para

representar
Número
.
Positivo_infinity

// O mesmo valor
1
/
0

// => Infinito
Número
.
Max_value

*

2

// => infinito;transbordamento
-
Infinidade

// um número negativo muito grande para

representar
Número
.

```
Número
.
Min_value
/
2

// => 0: subflow
-
Número
.
Min_value
/
2

// => -0: Zero negativo
-
1
/
Infinidade

// -> -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
.
isfinita
(
x
)

// é x um número e finito?
Número
.
isinteger
```

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
um divisor:
deixar

zero

=

0
;

// zero regular
deixar

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
Lá
são 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

```
deixar

x

=

.
3

-

.
2
;

// trinta centavos menos 20 centavos
deixar

y

=

.
2

-

.
1
;

// vinte centavos menos 10 centavos
x

===

y

// => false: os dois valores não são os

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

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
, Assim,
0o
, e
0x
Prefixos para bigints binários, octais e hexadecimais:
1234
n

// um literal não-so-big bigint
0B11111111
n

// um bigint binário
0o7777
n

// um bigint octal
0x8000000000000000
n

// => 2n ** 63N: um número inteiro de 64 bits
Você pode usar
Bigint ()
como uma função para converter regular
Números de JavaScript ou Strings para valores Bigint:
Bigint
(
Número
.
Max_safe_integer
)

// => 9007199254740991N
deixar

corda

=

"1"

+

"0"
.
repita
(
100
);

Embora o padrão
+
, Assim,
-
, Assim,
*
, Assim,
/
, Assim,
%
, e
**
Os operadores trabalham 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 veja
§3.9.1
Para mais sobre a diferença entre
==
e
===
):
1

<

2
n

// => true
2

>

1
n

// => true
0

==

0
n

// => true
0

===

```
Timestamp
que especifica o número de milissegundos decorridos desde
1 de janeiro de 1970:
deixar

Timestamp

=

Data
.
agora
();

// a hora atual como um

Timestamp (um número).
deixar

agora

=

novo

Data
();

// a hora atual como uma data

objeto.
deixar

EM

=

agora
.
gettime
();

// converter em um milissegundo

Timestamp.
deixar

ISO

=

agora
.
ToisSotring
();

// converter em uma string em

formato padrão.
```

UTF-16 como uma sequência (conhecida como um
"Par de substitutos") 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 caract
ere unicode:
deixar

euro

=

"€"
;
deixar

amor

=

"
❤
"
;
euro
.
comprimento

// => 1: este personagem tem um elemento de 16 bits
amor
.
comprimento

// => 2: UTF-16 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.
Em
ES6, no entanto, as cordas são
iterável
, e se você usar o
para/de
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
Inclua 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
`
).Caracteres duplos e
Backticks podem estar contidos em strings delimitados por uma única citação

em uma única linha, e é comum ver o código JavaScript que cria
longas seqüências de cordas 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 barro (
\
).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 do personagem
\ n
(documentado 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`
```

Observação
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).
Em
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:
<botão

ONCLICK =
"Alert ('obrigado')"
>
Clique em mim
</button>
3.3.2 Sequências de fuga em literais de cordas
O
Personagem de barragem (
\
) tem um propósito especial nas cordas JavaScript.
Combinado com o personagem que o segue, ele representa um personagem
Isso não é representável de outra forma dentro da string.Para
exemplo,
\ n
é
um
sequência de fuga
Isso representa um personagem de nova linha.
Outro
exemplo, mencionado anteriormente, é o
\ '
Escape, 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'
Tabela 3-1
listas
as sequências de escape JavaScript e os personagens que eles
representar.Três
As sequências de escape são genéricas e podem ser usadas para
representar qualquer caractere especificando seu código de caracteres unicode como um
Número hexadecimal.Para
exemplo, a sequência
\ xa9
representa o
símbolo de direitos autorais, que tem o unicode codificação dada pelo

```
Timestamp
que especifica o número de milissegundos decorridos desde
1 de janeiro de 1970:
deixar

Timestamp

=

Data
.
agora
();

// a hora atual como um

Timestamp (um número).
deixar

agora

=

novo

Data
();

// a hora atual como uma data

objeto.
deixar

EM

=

agora
.
gettime
();

// converter em um milissegundo

Timestamp.
deixar

ISO

=

agora
.
ToisSotring
();

// converter em uma string em

formato padrão.
```

Se o
\
o personagem precede qualquer caráter 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, ES5
permite a
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 usa o
+
Operador com números, ele os adiciona.Mas se
Você usa este operador em strings, ele se junta a eles anexando o segundo
para o primeiro.Por exemplo:

```
deixar

msg

=

"Olá, "

+

"mundo"
;

// produz a corda

"Olá, mundo"
deixar

saudações

=

"Bem -vindo ao meu blog,"

+

""

+

nome
;
```

Cordas
pode ser comparado com o padrão
===

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:
deixar

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
.
registro
(
"\ 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 que combina marca
\ 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:

```
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
, Assim,

"*"
)

// => "** x": adicione estrelas à esquerda para

um comprimento de 3
"X"
.
padend
(
3
, Assim,

"-"
)

// => "x--": adicione traços à direita

para um comprimento de 3
// Funções de corte de espaço.Trim () é ES5;Outros ES2019
" teste "
.
aparar
()

// => "teste": remova os espaços no início

e fim
" teste "
.
```

3.3.4 Literais de modelo
Em
ES6 e mais tarde, 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
pode 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 sequência 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.
ES6
tem uma função de tag embutida:
String.raw ()
.Ele retorna o texto
Dentro de backticks sem nenhum processamento de backslash escapas:
`
\
n`
.
comprimento

// => 1: a string tem um único

personagem newline
Corda
.
cru
`
\
n`
.
comprimento

// => 2: um caractere de barra 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.Vamos ver um exemplo em
§14.5
.
3.3.5 correspondência de padrões
JavaScript
define um tipo de dados conhecido como um
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 em
§11.3
.Porque os regexps são
poderoso e comumente usado para processamento de texto, no entanto, esta seção
Fornece uma breve visão geral.
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:
deixar

texto

=

"Teste: 1, 2, 3"
;

// Texto da amostra
deixar

padrão

=

padrão
.
teste
(
texto
)

// => true: existe uma correspondência
texto
.
procurar
(
padrão
)

// => 9: posição do primeiro

corresponder
texto
.
corresponder
(
padrão
)

// => ["1", "2", "3"]: matriz

de todas as partidas
texto
.
substituir
(
padrão
, Assim,

"#"
)

// => "Teste: #, #, #"
texto
.
dividir
(
/\ D+/
)

// => ["", "1", "2", "3"]:

dividido em não "
3.4 valores booleanos
UM
O 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
Avalie para esses dois valores.
Booleano
Os valores geralmente são o resultado de comparações que você faz em

Se o
\
o personagem precede qualquer caráter 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, ES5
permite a
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 usa o
+
Operador com números, ele os adiciona.Mas se
Você usa este operador em strings, ele se junta a eles anexando o segundo
para o primeiro.Por exemplo:

```
deixar

msg

=

"Olá, "

+

"mundo"
;

// produz a corda

"Olá, mundo"
deixar

saudações

=

"Bem -vindo ao meu blog,"

+

""

+

nome
;
```

Cordas
pode ser comparado com o padrão
===

confie no fato de que
nulo
é falsamente e os objetos são verdadeiros:
se

(
o
)

...
No primeiro caso, o corpo do
se
será executado apenas se
o
não é
nulo
.O segundo caso é menos rigoroso: ele executará o corpo do
se
Somente se
o
não é
falso
ou qualquer valor falsamente (como
nulo
ou
indefinido
).Qual
se
A declaração é apropriada para o seu programa
realmente depende de quais valores você espera ser atribuído
o
.Se você
precisa distinguir
nulo
de
0
e
""
, então você deve usar um
comparação explícita.
Booleano
Os valores têm a
ToString ()
método que você pode usar para
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

Completo
Detalhes sobre esses operadores estão em
§4.10
.
3.5 NULL e indefinido
nulo
é
uma palavra -chave do idioma que avalia um valor especial que é
geralmente usado para indicar a ausência de um valor.Usando o
typeof
operador em
nulo
Retorna a sequência "objeto", indicando que
nulo
pode
Seja pensado como um valor de objeto especial que indica "nenhum objeto".Em
prática, no entanto,
nulo
é normalmente 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 JavaScript
nulo
: você pode estar familiarizado com isso como
NULO
, Assim,
nil
, ou
Nenhum
.
O JavaScript também possui um segundo valor que indica ausência de valor.O
indefinido
O valor 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
indefinido
valor 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 idioma como
nulo
, embora isso não seja um
distinção importante na prática) que é inicializada para o
indefinido
valor.Se você aplicar o
typeof
operador para o
indefinido
valor, ele
retorna "indefinido", indicando que esse valor é o único membro de um
Tipo especial.
Apesar dessas diferenças,
nulo
e
indefinido
Ambos 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
falso
Quando um valor booleano é necessário.Nenhum
nulo
nem
indefinido
têm propriedades ou métodos.De fato, usando
.
ou
[]
para
Acesse uma propriedade ou método desses valores causa um TypeError.
Eu considero
indefinido
para representar um nível de sistema, inesperado, ou
ausência de valor semelhante a erros e
nulo
Para representar um nível de programa,
Ausência normal ou esperada de valor.Eu 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 costumo usar
nulo
.Alguns programadores se esforçam para evitar
nulo
inteiramente e uso
indefinido
em seu lugar onde quer que possam.
3.6 Símbolos
Símbolos
foram introduzidos no ES6 para servir como nomes de propriedades que não são de string.
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:
deixar

strname

=

"Nome da string"
;

// uma string para usar como um

Nome da propriedade
deixar

um valor único de símbolo.Se você fornecer um argumento de string, essa string irá
ser incluído na saída do símbolo do símbolo
ToString ()
método.
Observe, no entanto, que chama
Símbolo()
duas vezes com a mesma corda
produz dois valores de símbolo completamente diferentes.
deixar

s

=

Símbolo
(
"Sym_x"
);
s
.
ToString
()

// => "símbolo (sym_x)"
ToString ()
é o único método interessante de instâncias de símbolo.
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
Symbol.iterator
mecanismo
descrito anteriormente.
Para servir este último caso de uso, JavaScript define um símbolo global
registro.O
Symbol.for ()
função 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, o
Symbol.for ()
A função é completamente diferente do
Símbolo()
função:
Símbolo()
nunca retorna o mesmo valor duas vezes,
mas
Symbol.for ()
sempre retorna o mesmo valor quando chamado com
a mesma corda.A corda passou para
Symbol.for ()
aparece no
saída de
ToString ()

recuperado ligando
Symbol.Keyfor ()
no símbolo retornado.
deixar

s

=

Símbolo
.
para
(
"compartilhado"
);
deixar

t

=

Símbolo
.
para
(
"compartilhado"
);
s

===

t

// => true
s
.
ToString
()

// => "Símbolo (compartilhado)"
Símbolo
.
keyfor
(
t
)

// => "compartilhado"
3.7 O objeto global
O
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

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
em programas de nós.
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
Objeto de janela global tem um auto-referencial
janela
propriedade 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
tópicos (
§15.13
) ter um objeto global diferente do que a janela com
que eles estão associados.O código em um trabalhador pode se referir ao seu global
objeto como
auto
.
ES2020
finalmente define
global
como a 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
Lá
é uma diferença fundamental no JavaScript entre primitivo
valores (
indefinido
, Assim,
nulo
, 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:
deixar

s

=

"olá"
;

// Comece com algum texto em minúsculas
s
.
toupppercase
();

// retorna "Hello", mas não altera S
s

// => "Hello": a string original não tem

mudado
Primitivos também são comparados
por 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
mutável
-deles
Os valores podem mudar:
deixar

o

=

{

x
:

1

};

// Comece com um objeto
o

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:
deixar

o

=

{
x
:

1
},

p

=

{
x
:

1
};

// dois objetos com o mesmo

propriedades
o

===

p

// => false: objetos distintos

nunca são iguais
deixar

um

=

[],

b

=

[];

// duas matrizes distintas e vazias
um

===

b

```
copiar em
para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

eu
++
)

{

// Para cada índice de um []

b
[[
eu
]

=

um
[[
eu
];

// copie um elemento de um

em b
}
deixar

c

=

Variedade
.
de
(
b
);

// em ES6, copiar matrizes
```

dê a um número (ou para
Nan
Se não pode executar 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
deixar

n

=

1

-

"X"
;

// n == nan;String "X" não pode se converter para

um número
n

+

"Objetos"

// => "Nan Objects": nan se converte para

string "nan"
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"

-0
"0"
falso
1
(finito, diferente de zero)
"1"
verdadeiro
Infinidade
"Infinidade"
verdadeiro
-Infinidade
"-Infinidade"
verdadeiro
Nan
"Nan"
falso
{}
(qualquer objeto)
ver
§3.9.3
ver
§3.9.3
verdadeiro
[]
(matriz vazia)
""
0
verdadeiro
[9]
(um elemento numérico)
"9"
9
verdadeiro
['um']
(qualquer outra matriz)
Use o método junção ()
Nan
verdadeiro
função(){}
(qualquer função)
ver
§3.9.3
Nan
verdadeiro
As conversões primitivas para primitivas mostradas na tabela são relativamente
direto.Conversão para booleana já foi discutida em
§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 para 1 e
falso

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:
deixar

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
.
registro
(
"\ 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 que combina marca
\ 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:

Às vezes, você pode precisar realizar uma conversão explícita, ou você
Pode preferir tornar as conversões explícitas para manter seu código mais claro.
O
A maneira mais simples de realizar uma conversão de tipo explícita é usar o
Booleano ()
, Assim,
Número()
, e
Corda()
funções:
Número
(
"3"
)

// => 3
Corda
(
falso
)

// => "false": ou use false.toString ()
Booleano
([])

// => true
Qualquer valor que não seja
nulo
ou
indefinido
tem um
ToString ()
método, e o resultado desse método geralmente é o mesmo
devolvido pelo
Corda()
função.
Como um aparte, observe que o
Booleano ()
, Assim,
Número()
, e
Corda()
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.
Certo
Os operadores JavaScript realizam conversões de tipo implícito e são
Às vezes usado explicitamente para fins de conversão de tipo.Se um
operando do
+
O operador é uma string, converte o outro em um
corda.O unário
+
O operador converte seu operando em um número.E o
unário

+
x

// => número (x)
x
-
0

// => número (x)
!!
x

// => booleano (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

ToString ()
método 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:
deixar

n

=

17
;
deixar

binário

=

"0b"

+

n
.
ToString
(
2
);

// binário == "0B10001"
deixar

octal

=

"0o"

```
soma

+=

x
;

// Adicione o valor do elemento ao

soma.

}

// Este é o fim do loop.

retornar

soma
;

// retorna a soma.
}
soma
(
primos
)

// => 28: Soma dos primeiros 5

Prima 2+3+5+7+11
função

fatorial
(
n
)

{

// uma função para calcular

fatoriais

deixar

produto

=

1
;

// Comece com um produto de 1

enquanto
(
n

>
```

livre para pular a frente para
§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 o 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
implementar este algoritmo como
número preferido
.A classe de data
implementa esse algoritmo como
preferência string
.
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
Objeto a booleano
As conversões são triviais: todos os objetos se convertem para
verdadeiro
.
Observe que esta conversão não requer o uso do objeto para-

operadores de casos especiais que não usam o objeto básico para cordas e
Conversões de objeto para número descritas anteriormente.
O
+
operador
no javascript executa adição numérica e string
concatenação.Se qualquer um de seus operandos for um objeto, JavaScript converte
eles para valores primitivos usando o
sem preferência
algoritmo.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.
O

==
e
! =
Os operadores realizam testes de igualdade e desigualdade 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
sem preferência
algoritmo e depois compare os dois
valores primitivos.
Finalmente, o
operadores relacionais
<
, Assim,
<=
, Assim,
>
, e
> =
Compare a ordem de
seus operandos e podem ser usados nnpara comparar números e strings.Se
Qualquer um operando é um objeto, é convertido em um valor primitivo usando o
número preferido
algoritmo.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 a
<
e
>
, mas a representação da string não é.Para data
objetos, o
sem preferência
algoritmo se converte em uma corda, então o fato
Esse javascript usa o
número preferido

algoritmo
Para esses operadores
significa que podemos usá -los para comparar a ordem de dois objetos de data.

Os métodos ToString () e ValueOf ()
Todos
Objetos herdam dois métodos de conversão que são usados nnpor objeto a
conversões primitivas e antes que possamos explicar o
preferência string
, Assim,
número preferido
, e
sem preferência
algoritmos de conversão, temos que
Explique esses dois métodos.
O primeiro método é
ToString ()
, e seu trabalho é devolver uma string
representação do objeto.O padrão
ToString ()
Método faz
não retornar um valor muito interessante (embora o achemos útil em
§14.4.3
):
({
x
:

1
, Assim,

y
:

2
}).
ToString
()

// => "[objeto objeto]"
Muitas classes definem versões mais específicas do
ToString ()
método.O
ToString ()
Método da aula de matriz, por exemplo,
converte cada elemento da matriz em uma string e se junta às strings resultantes
juntamente com vírgulas no meio.O
ToString ()
Método do
A classe de função converte funções definidas pelo usuário em strings de javascript
código -fonte.A classe de data define um
ToString ()
método que
Retorna uma data e hora legíveis por humanos (e JavaScript)
corda.A classe regexp define um
ToString ()
método que
converte objetos regexp em uma string que se parece com um literal regexp:
[[
1
, Assim,
2
, Assim,

A outra função de conversão de objetos é chamada
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, então o padrão
valueof ()
o método simplesmente retorna o próprio objeto, em vez de retornar um
primitivo.Classes de wrapper, como string, número e booleano, definem
valueof ()
Métodos 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
valueof ()
método
que retorna a data em sua representação interna: o número de
milissegundos desde 1º de janeiro de 1970:
deixar

d

=

novo

Data
(
2010
, Assim,

0
, Assim,

1
);

// 1 de janeiro de 2010, (Pacífico

tempo)
d
.
valorof
()

// => 1262332800000
Algoritmos de conversão de objeto a princípios
Com
o
ToString ()
e
valueof ()
Métodos explicados, nós podemos
agora explique aproximadamente como os três objeto a princípio
Algoritmos funcionam (os detalhes completos são adiados até
§14.4.7

TypeError.
O
número preferido
algoritmo funciona como o
preferência string
algoritmo, exceto que tenta
valueof ()
primeiro e
ToString ()
segundo.
O
sem preferência
O algoritmo depende da classe do
objeto sendo convertido.Se o objeto for um objeto de data, então
JavaScript usa o
preferência string
algoritmo.Para qualquer outro
objeto, JavaScript usa o
número preferido
algoritmo.
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
número preferido
Conversão 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
número preferido
algoritmo e depois converte o resultante
valor primitivo para um número.O
número preferido
Algoritmo tenta
valueof ()
Primeiro e depois recai novamente
ToString ()
.Mas a matriz
classe herda o padrão
valueof ()
método, que não retorna um
valor primitivo.Então, quando tentamos converter uma matriz em um número, nós
acaba invocando o
ToString ()
Método 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
Um
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 um
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
Nós atribuímos 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
declarar
isto.Em
ES6 e mais tarde, isso é feito com o
deixar
e
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
Em
JavaScript moderno (ES6 e mais tarde), as variáveis ■■são declaradas com o
deixar
Palavra -chave, assim:
deixar

eu
;
deixar

soma
;
Você também pode declarar várias variáveis ■■em um único
deixar
declaração:

deixar

eu
, Assim,

soma
;
É uma boa prática de programação atribuir um valor inicial ao seu
variáveis ■■quando você as declara, quando isso é possível:
deixar

mensagem

=

"olá"
;
deixar

eu

=

0
, Assim,

j

=

0
, Assim,

k

=

0
;
deixar

x

=

2
, Assim,

y

=

x
*
x
;

// inicializadores podem usar anteriormente

Existem duas escolas de pensamento sobre o uso do
const
palavra -chave.Um
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
deve

não
mudar.No outro,
nós usamos
const
Para qualquer valor que não mude.Eu prefiro o primeiro
abordagem em meu próprio código.
Em
Capítulo 5
, vamos aprender sobre o
para
, Assim,
para/in
, e
para/de
laço
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
deixar
:
para
(
deixar

eu

=

0
, Assim,

Len

=

para
(
const

propriedade

em

objeto
)

console
.
registro
(
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
deixar
e
const
são
Block Scoped
.Isso significa que eles são definidos apenas dentro
o bloco de código em que o
deixar
ou
const
A declaração é exibida.
A classe JavaScript e as definições de função são blocos, assim como os
corpos de
se/else
declarações,
enquanto
loops,
para
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 do
deixar
ou
const
declaração que declara a variável).
Variáveis ■■e constantes declaradas como parte de um
para
, Assim,
para/in
, ou
para/de
Loop tem o corpo do loop como seu escopo, mesmo que eles
tecnicamente aparecem fora dos aparelhos encaracolados.

Declarações repetidas
É um erro de sintaxe usar o mesmo nome com mais de um
deixar
ou
const
Declaração 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
se

(
x

===

1
)

{

deixar

x

=

2
;

// dentro de um bloco x pode se referir a um

valor diferente

console
.
registro
(
x
);

// imprime 2
}
console
.
registro
(
x
);
```

Em
versões do JavaScript antes do ES6, a única maneira de declarar uma variável
está com o
var
palavra -chave, e não há como declarar constantes.O
Sintaxe de
var
é exatamente como a sintaxe de
deixar
:

```
var

x
;
var

dados

=

[],

contar

=

dados
.
comprimento
;
para
(
var

eu

=

0
;

eu

<

contar
;

eu
++
)

console
.
registro
(
dados
[[
eu
```

Loops, é típico para cada um começar
for (var i = 0;
...
.Porque
var
não escova essas variáveis ■■para o loop
corpo, cada um desses loops é (inofensivamente) re-declaração e re-
inicializando a mesma variável.
Um
das características mais incomuns de
var
declarações são
conhecido como
içar
.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.Então variáveis ■■declaradas com
var
pode ser
Usado, sem erro, em qualquer lugar da função de anexo.Se o
O código de inicialização ainda não foi executado, o valor do
variável pode ser
indefinido
, 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
deixar
correto: se você declarar uma variável com
deixar
mas tente
para usá -lo antes do
deixar
Declaração é executada, você receberá um real
erro em vez de apenas ver um
indefinido
valor.)
Usando variáveis ■■não declaradas
Em
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
atribuir um valor a um nome que não tenha sido declarado com
deixar
, Assim,
const
, ou
var
, Assim,
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

3.10.3 Atribuição de destruição
ES6
implementa um tipo de declaração e tarefa 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 um
const
, Assim,
deixar
, ou
var
declaração de declaração, mas também pode ser
feito em expressões regulares de tarefas (com variáveis ■■que têm
já foi declarado).E, como veremos em
§8.3.5
, destruição pode
Também seja usado ao definir os parâmetros para uma função.
Aqui estão tarefas simples de destruição usando matrizes de valores:
deixar

[[
x
, Assim,
y
]

=

[[
1
, Assim,
2
];

// o mesmo que let x = 1, y = 2
[[
x
, Assim,
y
]

=

[[
x
+
1
, Assim,
y
+
1

```
função

Tocartesiano
(
r
, Assim,

Theta
)

{

retornar

[[
r
*
Matemática
.
cos
(
Theta
),

r
*
Matemática
.
pecado
(
Theta
)];
}
deixar

[[
r
, Assim,
Theta
]

=

Topolar
(
1.0
, Assim,

1.0
);

// r == Math.sqrt (2);

Theta == Math.pi/4
deixar

[[
x
, Assim,
```

deixar

[[
x
, Assim,

...
y
]

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
];

// y == [2,3,4]
Veremos três pontos usados ■■dessa maneira novamente em
§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:
deixar

[[
um
, Assim,

[[
b
, Assim,

c
]]

=

[[
1
, Assim,

[[
2
, Assim,
2.5
],

3
];

// O mesmo que const sin = math.sin, cos = math.cos, tan = math.tan
const

{
pecado
, Assim,

cos
, Assim,

bronzeado
}

=

Matemática
;
Observe no código aqui que o
Matemática
objeto 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 atribuída
indefinido
.
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
, Assim,

bronzeado
:

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,

```
deixar

pontos

=

[{
x
:

1
, Assim,

y
:

2
},

{
x
:

3
, Assim,

y
:

4
}];

// uma variedade de

dois objetos de pontos
deixar

[{
x
:

x1
, Assim,

y
:

Y1
},

{
x
:

x2
, Assim,

y
:
```

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, nulo , e indefinido .

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 dobro em Java, C ++ e mais moderno linguagens de programação.

2 Existem extensões de JavaScript, como TypeScript e Flow (§17.8), isso permite que tipos ser especificado como parte de declarações variáveis ■■com sintaxe como Seja x: número = 0; .

Capítulo 4. Expressões e Operadores

Esse Capítulo documenta expressões JavaScript e os operadores com que muitas dessas expressões são construídas.Um

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. O 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

O
expressões mais simples, conhecidas como
expressões primárias
, são aqueles que
Stail sozinho - eles não incluem expressões mais simples.Primário
Expressões em JavaScript são constantes ou
literal
valores, certo
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 em
§3.2
.Corda
Literais foram documentados em
§3.3
.A expressão regular da sintaxe literal
foi introduzido em
§3.3.5
e será documentado em detalhes em
§11.3
.
Alguns
Das palavras reservadas de JavaScript são expressões primárias:

```
verdadeiro

// avalia para o valor verdadeiro booleano
falso

// avalia o valor falso booleano
nulo

// avalia o valor nulo
esse

// avalia o objeto "atual"
```

Aprendemos sobre
verdadeiro
, Assim,
falso
, e
nulo
em
§3.4

as outras palavras -chave,
esse
não é uma constante - é avaliado para diferente
valores em diferentes locais do programa.O
esse
A palavra -chave é usada em
Programação orientada a objetos.Dentro do corpo de um método,
esse
Avalia o objeto no qual o método foi invocado.Ver
§4.5
, Assim,
Capítulo 8
(especialmente
§8.2.2
), e
Capítulo 9
Para mais informações
esse
.
Finalmente, o terceiro tipo de expressão primária é uma referência a um
variável, constante ou propriedade do objeto global:
eu

// Avalia o valor da variável i.
soma

// 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
Objeto
e
Inicializadores de matriz
são
expressões cujo valor é um recém
objeto criado ou matriz.Essas expressões de inicializador às vezes são
chamado
objetos literais
e
matriz literais
.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 do Array 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
, Assim,
3
+
4
]

// Uma matriz de 2 elementos.O primeiro elemento é 3, segundo

é 7
O
Expressões de elementos em um inicializador de matriz podem ser matrizes
Inicializadores, o que significa que essas expressões podem criar aninhadas
Matrizes:
deixar

matriz

=

[[
1
, Assim,
2
, Assim,
3
],

[[
4
, Assim,
5
, Assim,
6
],

[[
7
, Assim,
8
, Assim,
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:

deixar

p

=

{

x
:

2.3
, Assim,

y
:

-
1.2

};

// um objeto com 2 propriedades
deixar

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
§6.10

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 de função estão em Capítulo 8 .

4.4 Expressões de acesso à propriedade

UM

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:

```
deixar o = { x : 1 , Assim, y : { z : 3 }};

// um exemplo de objeto
deixar um = [[ o
```

um
[[
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
existe, então o valor da expressão de acesso à propriedade é
indefinido
.
O
.Identifier
A sintaxe 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 (ver
§6.3.1
para um exemplo).
Objetos e suas propriedades são cobertos em detalhes em
Capítulo 6
, e
matrizes e seus elementos são cobertos em
Capítulo 7
.
4.4.1 Acesso à propriedade condicional
ES2020
Adiciona dois novos tipos de expressões de acesso à propriedade:

expressão
?.
identificador
expressão
?. [
expressão
]
Em JavaScript, os valores
nulo
e
indefinido
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
Esquerda avalia
nulo
ou
indefinido
.Você pode usar
?.
e
?. []
Sintaxe para proteger contra erros desse tipo.
Considere a expressão
A? .B
.Se
um
é
nulo
ou
indefinido
, então o
expressão avalia para
indefinido
sem qualquer tentativa de acessar o
propriedade
b
.Se
um
é algum outro valor, então
A? .B
Avalia para qualquer coisa
A.B
avaliaria para (e se
um
não tem uma propriedade chamada
b
, Assim,
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:
deixar

indefinido
e não lança um erro.Isso é porque propriedade
Acesso com
?.
é "curto-circuito": se a subexpressão à esquerda de
?.
avalia para
nulo
ou
indefinido
, então toda a expressão
imediatamente avalia
indefinido
sem propriedade adicional
Tentativas de acesso.
Claro, se
A.B
é um objeto, e se esse objeto não tiver propriedade denominada
c
, então
A.B? .C.D
novamente jogará um TypeError, e vamos querer
Use outro acesso à propriedade condicional:
deixar

um

=

{

b
:

{}

};
um
.
b
?
.
c
?
.
d

// => indefinido
O acesso à propriedade condicional também é possível usando
?. []
em vez de
[]
.
Na expressão
abc]
, se o valor de
um
é
nulo

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
Um
expressão de invocação
é
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.

```
Matemática
.
máx
(
x
, Assim,
y
, Assim,
z
)

// math.max é a função;x, y e z são
```

os argumentos.

```
um
.
organizar
()

// 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 usa um
retornar
declaração para retornar um valor, então que
O valor se torna o valor da expressão de invocação.Caso contrário, o
O valor da expressão de invocação é
indefinido
.Detalhes completos

Todo
A 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 do
esse
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.Ver
Capítulo 9
Para 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
nulo
ou
indefinido
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
, Assim,
Então toda a expressão de invocação avalia para
indefinido
e não
A exceção é lançada.
Variedade
Objetos têm um
organizar()
método 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
organizar()
Isso exige um argumento de função opcional, você normalmente
use um
se
declaração para verificar se o argumento da função foi definido
antes de invocá -lo no corpo do
se
:
função

função opcional

se

(
registro
)

{

// se a função opcional for

passou

registro
(
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 disso
A invocação só acontecerá se houver um valor a ser chamado:
função

quadrado
(
x
, Assim,

registro
)

{

// O segundo argumento é um

função opcional

registro
?

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
?. ()
trabalhar 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

Um
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
novo
:
novo

Objeto
()
novo

Apontar
(
2
, Assim,
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
novo

Data
O valor de uma expressão de criação de objetos é o objeto recém -criado.
Os construtores são explicados em mais detalhes em
Capítulo 9
.
4.7 Visão geral do operador
Operadores
são usados ■■para as expressões aritméticas de JavaScript, comparação
Expressões, expressões lógicas, expressões de atribuição e muito mais.
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
Instância de
.Os operadores de palavras -chave são operadores regulares,
Assim como os expressos com pontuação;eles simplesmente têm menos
Sintaxe sucinta.
Tabela 4-1
é organizado pela precedência do operador.Os operadores listados

Primeiro, tem maior precedência do que os listados por último.Operadores separados
Por uma linha horizontal tem diferentes níveis de precedência.A coluna
rotulado como dá ao operador associatividade, que pode ser L (da esquerda para a esquerda para
direita) ou r (direita para a esquerda), e a coluna n especifica o número de
operando.Os tipos rotulados pela coluna lista os tipos esperados do
operando e (após o → símbolo) o tipo de resultado para o operador.O
subseções que seguem a tabela explica os conceitos de precedência,
Associatividade e tipo de operando.O
Os próprios operadores são
documentado individualmente após essa discussão.
Tabela 4-1.
Operadores JavaScript
Operador
Operação
UM
N
Tipos
++
Pré ou pós-incremento
R
1
LVAL → NUM
-
Pré ou pós-decisão
R
1
LVAL → NUM
-
Negar número
R
1
NUM → NUM
+
Converter em número
R
1
qualquer → num
~
Inverter bits
R
1
int → int
!
Inverter o valor booleano
R
1
BOOL → BOOL
excluir
Remova uma propriedade
R
1
LVAL → BOOL
typeof
Determine o tipo de operando
R
1
qualquer → str
vazio
Retornar valor indefinido

+
Cordas concatenadas
L
2
str, str → str
<<
Mudança para a esquerda
L
2
int, int → int
>>
Mudar à direita com a extensão do sinal
L
2
int, int → int
>>>
Mudar à direita com extensão zero
L
2
int, int → int
<
, Assim,
<=
, Assim,
>
, Assim,
> =
Compare em ordem numérica
L
2
num, num → bool
<
, Assim,
<=
, Assim,
>
, Assim,
> =
Compare em ordem alfabética
L
2
str, str → bool
Instância de
Teste da classe de objeto
L
2
Obj, func → bool
em
Teste se a propriedade existe
L
2
qualquer, obj → bool
==
Teste para igualdade não rigorosa
L
2
qualquer, qualquer → bool
! =
Teste para desigualdade não esticada

+=
, Assim,
-=
, Assim,
& =
, Assim,
^=
, Assim,

| =
, Assim,
<< =
, Assim,
>> =
, Assim,
>>> =
, Assim,
Descarte o 1º operando, retornar

2º
L
2
qualquer, qualquer → qualquer
4.7.1 Número de operandos
Operadores
pode ser categorizado com base no número de operandos que eles
Espere (deles
ARITY
).Maioria
Operadores de javascript, como o
*
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 uma série de
operadores unários
, que converte um
expressão única em uma expressão única e mais complexa.O
-
operador na expressão
–x
é um operador unário que executa o
operação de negação no operando
x
.Finalmente, JavaScript
suportes
um
operador ternário
, o operador condicional
?:
, que combina
três expressões em uma única expressão.
4.7.2 Operando e tipo de resultado
Alguns
Os 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

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
+
O operador adiciona numérico
operando, mas concatena operandos de string.Da mesma forma, a comparação
operadores como
<
realizar comparação em numérico ou alfabético
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.
Perceber
que os operadores de tarefas e alguns dos outros operadores
listado em
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
Avaliação
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 pode 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
excluir
O operador também tem efeitos colaterais:
Excluir uma propriedade é como (mas não o mesmo que) atribuir
indefinido
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
O
operadores listados em
Tabela 4-1
estão dispostos em ordem da 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:

```
c
=
x
+
y
*
z
;
```

O operador de multiplicação
*
tem uma precedência maior que a adição
operador
+
, então 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:
c

=

(
x

+

y
)
*
z
;
Observe que as expressões de acesso e invocação de propriedades têm maior
precedência do que qualquer um dos operadores listados em
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.
typeof

meu
.
funções
[[
x
] (
y
)
Embora
typeof
é um dos operadores de maior prioridade, o
typeof
a 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 baixa precedência do que
||
e

ou
&&
.Da mesma forma, o novo
**
O operador de exponenciação não tem 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
Em
Tabela 4-1
, o
A coluna rotulada como especifica o
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:
c

=

x

-

y

-

z
;
é o mesmo que:
c

=

((
x

-

y
)

-

z
);
Por outro lado, as seguintes expressões:
y

=

um

c

=

(
x

=

(
y

=

z
));
q

=

um
?
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
Operador
Precedência e associatividade especificam 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
c
é
avaliado primeiro, seguido por
x
, Assim,
y
, e
z
.Então os valores de
y
e
z

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 (veja
§3.2.5
) operando ou com números regulares, como
desde que você não misture os dois tipos.
O
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 o
Nan
valor.Se qualquer um opera é (ou converter para)
Nan
, o resultado do
a operação é (quase sempre)
Nan
.
O
**
O operador tem maior precedência do que
*
, Assim,
/
, e
%
(que por sua vez
ter maior precedência do 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

Operador: foi adicionado ao idioma com ES2016.O
Math.pow ()
A função está disponível desde as primeiras versões
de JavaScript, no entanto, e executa exatamente a mesma operação que
o
**
operador.
O
/
O operador divide seu primeiro operando em seu 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
.A divisão por zero produz positivo ou
Infinidade negativa, enquanto
0/0
avalia para
Nan
: nenhum desses casos
levanta um erro.
O
%
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
binário
+
O operador 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
+
dar
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
+
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 seus

ToString ()
método, e todos os outros objetos são convertidos via
valueof ()
, Assim,
Se esse método retornar um valor primitivo.No entanto, a maioria
objetos não têm um útil
valueof ()
método, então eles 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.

corda
1

+

{}

// => "1 [objeto objeto]": concatenação após

objeto para cordas
verdadeiro

+

verdadeiro

// => 2: adição após boolean para número
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
+
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

+

"Ratos cegos"

// => "3 ratos cegos"
1

+

(
2

+

Os operadores aritméticos unários são os seguintes:
UNARY Plus
(
+
)
O
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

++
incrementos do operador (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
++
O 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 para o
não conceituado
valor de
aquele operando.Considere a diferença entre essas duas linhas de
código:
deixar

eu

=

1
, Assim,

j

=

++

Observe que a expressão
x ++
nem sempre é o mesmo que
x = x+1
.
O
++
O operador nunca executa concatenação de string: sempre
converte seu operando em um número e o incrementa.Se
x
é a string
"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
para
laço
(
§5.4.3
).
Decremento
(
-
)
O

-
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
não -afirmado
valor.Quando usado após seu operando, não
A quebra de linha é permitida entre o operando e o operador.
4.8.3 Operadores bitwise
O
Os operadores bitwise realizam manipulação de baixo nível dos bits no
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
, Assim,
Infinidade
, e
-Infinidade
Todos se convertem para 0
Quando usado como operando desses operadores bit -bit.
Todos esses operadores bit new, exceto
>>>
pode ser usado com regular
operando numéricos ou com bigint (veja
§3.2.5
) operando.
Bit nejudado e
(
&
)
O

&
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 nejudado 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 executa um exclusivo booleano ou operação em cada
um pouco de seus argumentos inteiros.Exclusivo ou significa que também
operando um é
verdadeiro
ou 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

~
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,
aplicando o
~
O operador para um valor é equivalente a mudar seu sinal
e subtrair 1. Por exemplo,
~ 0x0f
avalia para
0xfffffff0
, ou -16.
Mudança para a esquerda
(
<<
)
O

<<
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
um

O

>>
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
a 3, por exemplo, mas observe isso e
-7 >> 1
Avalia como -4.
Mudar à direita com preenchimento zero
(
>>>
)
O

>>>
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
Esse
A 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
, Assim,
enquanto
, e
para

declarações (ver
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,
em
e
Instância de
.

4.9.1 Operadores de igualdade e desigualdade

O

==
e
===
Os operadores 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 operandos são os
mesmo e
falso
Se eles são diferentes.O
===
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
==
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
! ==
Os operadores testam exatamente o oposto do
==
e
===
operadores.O
! =
Retornos do operador de desigualdade
falso
Se dois
Os valores são iguais um ao outro de acordo
==
e retorna
verdadeiro
de outra forma.O
! ==
Retorna do operador
falso
Se dois valores forem estritamente
igual um ao outro e retorna
verdadeiro
de outra forma.Como você verá em
§4.10

A outra função de conversão de objetos é chamada
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, então o padrão
valueof ()
o método simplesmente retorna o próprio objeto, em vez de retornar um
primitivo.Classes de wrapper, como string, número e booleano, definem
valueof ()
Métodos 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
valueof ()
método
que retorna a data em sua representação interna: o número de
milissegundos desde 1º de janeiro de 1970:
deixar

d

=

novo

Data
(
2010
, Assim,

0
, Assim,

1
);

// 1 de janeiro de 2010, (Pacífico

tempo)
d
.
valorof
()

// => 1262332800000
Algoritmos de conversão de objeto a princípios
Com
o
ToString ()
e
valueof ()
Métodos explicados, nós podemos
agora explique aproximadamente como os três objeto a princípio
Algoritmos funcionam (os detalhes completos são adiados até
§14.4.7

```
deixar

x

=

.
3

-

.
2
;

// trinta centavos menos 20 centavos
deixar

y

=

.
2

-

.
1
;

// vinte centavos menos 10 centavos
x

===

y

// => false: os dois valores não são os

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

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
+
dar
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
+
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 seus

ToString ()
método, e todos os outros objetos são convertidos via
valueof ()
, Assim,
Se esse método retornar um valor primitivo.No entanto, a maioria
objetos não têm um útil
valueof ()
método, então eles 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.

Menor ou igual
(
<=
)
O

<=
O operador avalia para
verdadeiro
Se seu primeiro operando for menor que
ou igual ao seu segundo operando;Caso contrário, ele avalia para
falso
.
Maior ou igual
(
> =
)
O

> =
O operador avalia para
verdadeiro
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 é
valueof ()
Método retorna um valor primitivo,
esse valor é usado.Caso contrário, o valor de retorno de seu
ToString ()
o método 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.
Infinidade
é maior do que qualquer número que não seja, e
-
Infinidade
é menor do que qualquer número que não seja ele mesmo.Se
ou operando é (ou convertido para)
Nan
, então 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 leva o local-
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
.
Ambos
+
operador e 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

// => false: 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

1

em

dados

// => true: os números são convertidos

para cordas
3

em

dados

// => false: nenhum elemento 3
4.9.4 A instância do operador
O

Instância de
O 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 para
verdadeiro
Se o objeto do lado esquerdo for uma instância do
classe do lado direito e avalia
falso
de outra forma.
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:
deixar

d

=

novo

Data
();

// Crie um novo objeto com a data ()

construtor
d

Instância de

Data

// => true: D foi criado com Date ()
d

Instância de

mecanismo, e é descrito em
§6.3.2
.Para avaliar a expressão
o
Instância de f
, JavaScript avalia
F.Prototipo
, e então
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
verdadeiro
.
Se
F.Prototipo
não é um dos valores na cadeia de protótipos de
o
, Assim,
então
o
não é uma instância de
f
e
Instância de
retorna
falso
.
4.10 Expressões lógicas
O
operadores lógicos
&&
, Assim,
||
, 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 em
§3.4
.
4.10.1 lógico e (&&)
O

&&
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

Expressões relacionais sempre avaliam para
verdadeiro
ou
falso
, então quando
usado assim, o
&&
O próprio operador retorna
verdadeiro
ou
falso
.
Os operadores relacionais têm maior precedência do 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".(Ver
§3.4
Para detalhes.
Os valores falsamente são
falso
, Assim,
nulo
, Assim,
indefinido
, Assim,
0
, Assim,
-0
, Assim,
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
faz
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
que precisamos descrever
&&
no terceiro e último nível.Este operador

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
+
dar
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
+
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 seus

ToString ()
método, e todos os outros objetos são convertidos via
valueof ()
, Assim,
Se esse método retornar um valor primitivo.No entanto, a maioria
objetos não têm um útil
valueof ()
método, então eles 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.

trabalha sobre verdade e falsidade
valores.
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
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 com o
&&
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.
deixar

máx

=

MaxWidth

||

preferências
.
MaxWidth

||

500
;
Observe que se 0 é um valor legal para
MaxWidth
, então este código não vai
Trabalhe corretamente, pois 0 é um valor falsamente.Veja o
?

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
função

cópia
(
o
, Assim,

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 para
falso
.Se
x
é falsamente, então
! x
é
verdadeiro
.
Ao contrário do
&&
e

álgebra aqui que podemos expressar usando
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:
eu

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:
eu

=

j

=

k

=

0
;

// Inicialize 3 variáveis ■■para 0
4.11.1 Atribuição com operação
Além do mais
o normal
=
Operador de atribuição, 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 poderia esperar, o
+=
O operador trabalha para números ou strings.
Para operandos numéricos, ele executa adição e atribuição;para string
operandos, executa concatenação e atribuição.
Operadores semelhantes incluem

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
op
é um operador, é equivalente à expressão:
a = a op b
Na primeira linha, a expressão
um
é avaliado uma vez.No segundo, é
avaliado duas vezes.Os dois casos serão diferentes apenas se
um
Inclui lado

efeitos como uma chamada de função ou um operador de incremento.A seguir
Duas tarefas, por exemplo, não são
o mesmo:
dados
[[
eu
++
]

*=

2
;
dados
[[
eu
++
]

=

dados
[[
eu
++
]

*

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
avaliar ()
:
aval
(
"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 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 usar com
avaliar ()
.Por causa disso

avaliar ()
é uma função, mas está incluído neste capítulo sobre expressões porque realmente deveria ter
foi um operador.As primeiras versões do idioma definiram um
avaliar ()
função, 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 ligar
avaliar ()
, o intérprete não pode otimizar isso
função.O problema de definir
avaliar ()
Como função é que pode receber outros nomes:

```
deixar f = aval;
deixar g = f;
```

Se isso for permitido, o intérprete não pode ter certeza de quais funções chamam
avaliar ()
, então não pode
otimizar agressivamente.Este problema poderia ter sido evitado se
avaliar ()
era um operador (e um
palavra reservada).Vamos aprender (em
§4.12.2
e
§4.12.3
) sobre restrições colocadas em
avaliar ()
para fazer isso
Mais como operador.

4.12.1 Eval ()

avaliar ()
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 coisa chave sobre
avaliar ()

declara uma nova variável local
y
.Por outro lado, se avaliado
Usos de string
deixar
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:
aval
(
"Função f () {return x+1;}"
);
Se você ligar
avaliar ()
Do código de nível superior, ele opera em variáveis ■■globais
e funções globais, é claro.
Observe que a sequência de código que você passa
avaliar ()
deve fazer sintático
Sentir por conta própria: você não pode usá -lo para colar fragmentos de código em um
função.Não faz sentido escrever
Eval ("Return;")
, para
exemplo, porque
retornar
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 ()
.De outra forma,
avaliar ()
vai jogar um
SyntaxError.
4.12.2 Global Eval ()
Isto
é a capacidade de
avaliar ()
para alterar as variáveis ■■locais que são tão
Problemático para otimizadores de JavaScript.Como solução alternativa, no entanto,
Os intérpretes simplesmente fazem menos otimização em qualquer função que chama
avaliar ()
.Mas o que um intérprete de javascript deve fazer, no entanto, se um
script define um pseudônimo para
avaliar ()
e então chama isso de função por

Outro nome?A especificação JavaScript declara que quando quando
avaliar ()
é invocado por qualquer nome que não seja "avaliar", ele 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 o
avaliar ()
função 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 ()
use 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
Chamadas podem definir novas variáveis ■■apenas com
var
.Usos de
deixar
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

=

aval
;

// usando outro nome faz

Uma avaliação global
deixar

x

=

"global"
, Assim,

y

=

"global"
;

// duas variáveis ■■globais
função

f
()

```
Geval
(
"y += 'alterado';"
);

// Conjuntos de avaliação indireta

variável global

retornar

y
;

// retorna local inalterado local

variável
}
console
.
registro
(
f
(),

x
);

// Variável local alterada: impressões

"LocalChanged Global":
console
.
registro
(
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 começo 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 ()

Estrito modo (veja §5.6.3) impõe restrições adicionais ao comportamento de o

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
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 pareça
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.
Enquanto você pode obter resultados semelhantes usando o
se
declaração (
§5.3.1
),
o
?:
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:
saudações

=

"olá "

+

(
nome de usuário

?

nome de usuário

:

"lá"
);
Isso é equivalente a, mas mais compacto do que o seguinte
se
declaração:
saudações

=

"olá "
;
se

(
nome de usuário
)

{

saudações

+=

nome de usuário
;
}

outro

{

saudações

+=

"lá"
;
}
4.13.2 Primeiro definido (??)
O
operador primeiro definido

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
+
dar
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
+
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 seus

ToString ()
método, e todos os outros objetos são convertidos via
valueof ()
, Assim,
Se esse método retornar um valor primitivo.No entanto, a maioria
objetos não têm um útil
valueof ()
método, então eles 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.

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:

```
(
um

?

b
)

||

c

// ??Primeiro, depois ||
um

?

(
b

||

c
)

// ||Primeiro, então ??
um

?

b

||

c

// SyntaxError: Parênteses são necessários
```

O
?
O operador é definido por
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 do seu único operando,

qualquer bigint
"bigint"
qualquer string
"corda"
qualquer símbolo
"símbolo"
qualquer função
"função"
qualquer objeto de não função
"objeto"
Você pode usar o
typeof
operador em uma expressão como esta:

```
// Se o valor for uma string, embrulhe -o nas citações, caso contrário,

converter
(
typeof

valor

===

"corda"
)

?

""""

+

valor

+

""""

:

valor
.
ToString
()
```

Observe que
typeof
retorna "objeto" se o valor do operando for
nulo
.Se
você quer distinguir
nulo
de objetos, você terá que explicitamente
teste para esse valor de caso especial.
Embora as funções JavaScript sejam um tipo de objeto, o
typeof
O operador considera as funções que são suficientemente diferentes que eles têm
seu próprio valor de retorno.
Porque
typeof

excluir
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,
excluir
é normalmente usado para o seu
Efeito colateral da exclusão da propriedade e não para o valor que ele retorna.Alguns
Exemplos:
deixar

o

=

{

x
:

1
, Assim,

y
:

2
};

// Comece com um objeto
excluir

o
.
x
;

// Exclua uma de suas propriedades
"X"

em

o

// => false: a propriedade não

existe mais
deixar

um

=

[[
1
, Assim,
2
, Assim,
3

de
eliminação.
Em
modo rigoroso,
excluir
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
).Modo rigoroso também especifica que
excluir
levanta a
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
falso
para indicar que o operando poderia
não ser excluído.
Aqui estão alguns exemplos de usos do
excluir
operador:
deixar

o

=

{
x
:

1
, Assim,

y
:

2
};
excluir

o
.
x
;

// excluir uma das propriedades do objeto;retorna

verdadeiro.
typeof

o
.
x
;

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
aguarde
Operador é o valor de atendimento do objeto Promise.Importante,
aguarde
é apenas legal dentro de funções que foram declaradas
assíncrono com o
assíncrono
palavra -chave.Mais uma vez, veja
Capítulo 13
para completo
detalhes.
4.13.6 O operador vazio
vazio
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
indefinido
.
Como o valor do operando é descartado, usando o
vazio
O operador faz
Senta apenas se o operando tiver efeitos colaterais.
O
vazio
O operador 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:
deixar

contador

=

0
;

```
const

incremento

=

()

=>

vazio

contador
++
;
incremento
()

// => indefinido
contador

// => 1
```

4.13.7 O operador de vírgula (,)

O

vírgula
operador 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:

```
eu
=
0
```

, Assim,

```
j
=
1
```

, Assim,

```
k
=
2
;
```

avalia para 2 e é basicamente equivalente a:

```
eu

=

0
;

j

=

1
;

k
```

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
avaliado
para 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 javascript
+
O operador 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
Capítulo 4
descrito
Expressões como frases 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
avaliado
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
maneira é conhecida como
declarações de expressão
.Uma categoria semelhante de
declarações são o
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
isso faz
Apenas isso:
Condicionais
Declarações
como
se
e
trocar
que fazem o javascript
o intérprete executar ou pular outras declarações, dependendo do valor

de uma expressão
Loops
Declarações
como
enquanto
e
para
que executa outras declarações
repetidamente
Saltos
Declarações
como
quebrar
, Assim,
retornar
, e
lançar
isso causa 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, quando você
estão familiarizados com as declarações de JavaScript, você pode começar a escrever
Programas JavaScript.
5.1 declarações de expressão
O
tipos mais simples de declarações em javascript são expressões que
tem efeitos colaterais.Este tipo de afirmação foi mostrado em
Capítulo 4
.
As declarações de atribuição são uma categoria principal de expressão
declarações.Por exemplo:
saudações

=

"Olá "

+

nome
;
eu

*=

3
;
Os operadores de incremento e decréscimo,
++
e
-
, estão relacionados a
declarações de atribuição.Estes têm o efeito colateral de mudar um

Existem duas escolas de pensamento sobre o uso do
const
palavra -chave.Um
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
deve

não
mudar.No outro,
nós usamos
const
Para qualquer valor que não mude.Eu prefiro o primeiro
abordagem em meu próprio código.
Em
Capítulo 5
, vamos aprender sobre o
para
, Assim,
para/in
, e
para/de
laço
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
deixar
:
para
(
deixar

eu

=

0
, Assim,

Len

=

5.2 declarações compostas e vazias
Apenas
Como o operador de vírgula (
§4.13.7
) combina várias expressões
em uma única expressão, um
Bloco de declaração
combina múltiplo
declarações em um único
declaração 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

=

Matemática
.
Pi
;

cx

=

Matemática
.
cos
(
x
);

console
.
registro
(
"cos (π) ="

+

cx
);
}
```

Há algumas coisas a serem observadas sobre este bloco de declaração.Primeiro, sim
não
termine 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, o
enquanto
Sintaxe do loop

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
laço (
para
Loops serão cobertos em
§5.4.3
):
// inicialize uma matriz a
para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

um
[[
eu
++
]

=

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 a
para
laço,
enquanto
loop, ou
se
a declaração pode causar

para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

um
[[
eu
++
]

=

0
)

/* vazio */

;
5.3 Condicionais
Declarações condicionais
executar ou pular 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
se/else
declaração, e também cobre
trocar
, um mais complicado,
Declaração de Multiway Branch.
5.3.1 se
O

se
declaração 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
)

Ou da mesma forma:
// Se o nome de usuário for nulo, indefinido, falso, 0, "" ou nan, dê

é um novo valor
se

(
!
nome de usuário
)

nome de usuário

=

"John Doe"
;
Observe que os parênteses ao redor do
expressão
são uma parte necessária de
a sintaxe para o
se
declaração.
A sintaxe javascript requer uma única declaração após o
se
palavra -chave e
expressão entre parênteses, mas você pode usar um bloco de declaração para
Combine várias declarações em uma.Então o
se
declaração também pode
Parece isso:
se

(
!
endereço
)

{

endereço

=

""
;

mensagem

=

"Especifique um endereço de correspondência."
;
}
A segunda forma do
se
A declaração apresenta um
outro
Cláusula que é

```
console
.
registro
(
`Você tem
$ {
n
}
novas mensagens
);
Quando você tiver aninhado
se
declarações com
outro
cláusulas, alguma cautela
é necessário para garantir que o
outro
Cláusula acompanha o apropriado
se
declaração.Considere as seguintes linhas:
eu

=

j

=

1
;
k

=

2
;
se

(
eu

===

j
)

se

(
j

===

k
)

console
.
registro
```

```
se

(
eu

===

j
)

{

se

(
j

===

k
)

{

console
.
registro
(
"Eu igual a k"
);

}
}

outro

{

// que diferença a localização de uma cinta encaracolada

faz!

console
.
registro
(
"Eu não igual a J"
);
}
```

Muitos programadores têm o hábito de envolver os corpos de
`se`
e
`outro`
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

}
Não há nada de especial nesse código.É apenas uma série de
se
declarações, onde cada um seguindo
se
faz parte do
outro
Cláusula do
declaração anterior.Usando o
caso contrário, se
o idioma é preferível e
mais legível do que, escrevendo essas declarações
Formulário equivalente e totalmente aninhado:
se

(
n

===

1
)

{

// Execute o bloco de código nº 1
}
outro

{

se

(
n

===

2
)

{

// Executar o bloco de código #2

}

outro

{

se

(
n

===

3

+
x

// => número (x)
x
-
0

// => número (x)
!!
x

// => booleano (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

ToString ()
método 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:
deixar

n

=

17
;
deixar

binário

=

"0b"

+

n
.
ToString
(
2
);

// binário == "0B10001"
deixar

octal

=

"0o"

padrão
:

// Se tudo mais falhar ...

// Executar o bloco de código #4.

quebrar
;

// Pare aqui
}
Observe o
quebrar
palavra -chave usada no final de cada
caso
Neste código.
O
quebrar
declaração, descrita mais adiante neste capítulo, causa o
intérprete para pular até o fim (ou "quebrar") do
trocar
declaração e continue com a declaração que a segue.O
caso
cláusulas em a
trocar
Declaração Especifique apenas o
ponto de partida
do
código desejado;Eles não especificam nenhum ponto final.Na ausência de
quebrar
declarações, a
trocar
a declaração começa a executar seu bloco de
código no
caso
rótulo que corresponde ao valor de seu
expressão
e
continua executando declarações até chegar ao final do bloco.Sobre
ocasiões raras, é útil escrever um código como esse que "cai"
de um
caso
rótulo para o próximo, mas 99% do tempo você deve ser
Cuidado para terminar a cada
caso
com um
quebrar
declaração.(Ao usar
trocar
Dentro de uma função, no entanto, você pode usar um
retornar
declaração
em vez de um
quebrar
declaração.Ambos servem para encerrar o
trocar
declaração e impedir a execução de cair para o próximo
caso

retornar

""

+

x

+

""
;

padrão
:

// converte qualquer outro tipo em

da maneira usual

retornar

Corda
(
x
);

}
}
Observe que nos dois exemplos anteriores, o
caso
Palavras -chave são
seguido de literais de número e cordas, respectivamente.É assim que o
trocar
a declaração é mais frequentemente usada na prática, mas observe que o
O padrão Ecmascript permite cada
caso
a ser seguido por um arbitrário
expressão.
O
trocar
A declaração primeiro avalia a expressão que segue o
trocar
palavra -chave e depois avalia o
caso
expressões, 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, para que as expressões devem corresponder sem qualquer tipo
conversão.
Porque nem todos os
caso
expressões são avaliadas cada vez que
trocar
A declaração é executada, você deve evitar usar

Exemplos mostrados, o
padrão:
o rótulo aparece no final do
trocar
corpo,
seguindo
tudo
caso
Rótulos.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
Entenda declarações condicionais, imaginamos o JavaScript
intérprete seguindo um caminho de ramificação através do seu código -fonte.O
declarações de loop
são aqueles que dobram esse caminho de volta para si
Repita partes do seu código.JavaScript tem cinco declarações de loop:
enquanto
, Assim,
faça/while
, Assim,
para
, Assim,
para/de
(e é
para/aguardar
variante),
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

Apenas
Como o
se
declaração é
JavaScript
Condicional básico, o
enquanto
A declaração é o loop básico de JavaScript.Tem a seguinte sintaxe:
enquanto (
expressão
)

declaração
Para executar a
enquanto
declaração, o intérprete avalia primeiro
expressão
.Se o valor da expressão for falsamente, então o intérprete
pula sobre o
declaração
que serve como o corpo do loop e passa para
A próxima declaração no programa.Se, por outro lado, o
expressão

de volta ao topo do loop e avaliar
expressão
de novo.Outro
maneira de dizer isso é que o intérprete é executado
declaração
repetidamente
enquanto
o
expressão
é verdade.Observe que você pode criar um loop infinito
com a sintaxe
enquanto (verdadeiro)
.
Geralmente, você não deseja que o JavaScript execute exatamente o mesmo
operação repetidamente.Em quase todos os loops, um ou mais
variáveis ■■mudam com cada
iteração
do loop.Desde as variáveis
mudança, as ações executadas executando
declaração
pode diferir cada um
tempo através do loop.Além disso, se a variável em mudança ou
variáveis ■■estão envolvidas em
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
enquanto
loop que imprime o
números de 0 a 9:
deixar

contar

=

0
;
enquanto
(
contar

<

10
)

{

console
.
registro
(
contar
);

contar
++

5.4.2 Do/while
O

faça/while
Loop é como um
enquanto
loop, 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
faça/while
loop é menos comumente usado do que o seu
enquanto
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
faça/while
laço:
função

PrintArray
(
um
)

{

deixar

Len

=

um
.
comprimento
, Assim,

eu

=

0
;

se

(
Len

===

5.4.3 para
O

para
a declaração fornece uma construção em loop que geralmente é mais
conveniente que o
enquanto
declaração.O
para
A declaração 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.O
para
declaração 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
, Assim,
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
Loop facilita a compreensão do que um
para
Loop está fazendo e
evita erros como esquecer de inicializar ou incrementar o loop
variável.
A maneira mais simples de explicar como um
para
Loop Works é mostrar o
equivalente
enquanto
laço:
inicializar
;
enquanto(
teste
) {

declaração

incremento
;

Em outras palavras, o
inicializar
A expressão é avaliada uma vez, antes do
Loop começa.Para ser útil, essa expressão deve ter efeitos colaterais
(geralmente uma tarefa).JavaScript também permite
inicializar
ser um
Declaração de declaração variável para que você possa declarar e inicializar um
contador de loop ao mesmo tempo.O
teste
A expressão é avaliada antes
cada iteração e controla se o corpo do loop é executado.Se
teste
avalia um valor verdadeiro, o
declaração
esse é o corpo do
O loop é executado.Finalmente, o
incremento
A expressão 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
para
loop como o
seguindo.Contraste com o equivalente
enquanto
loop mostrado no
Seção anterior:

```
para
(
deixar

contar

=

0
;

contar

<

10
;

contar
++
)

{

console
.
```

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
loop para atravessar 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

cauda
(
o
)

{

// retorna o

cauda de lista vinculada o

para
(;

o
.
próximo
;

o

=

o
.
próximo
)

/* vazio */

;

// atravessar enquanto

O.Next é verdade

retornar

o
;
}
Observe que este código não tem
inicializar
expressão.Qualquer um dos três
expressões podem ser omitidas de um
para
loop, mas os dois semicolons
são necessários.Se você omitir o
teste
expressão, o loop se repete para sempre,

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:

```
deixar

dados

=

[[
1
, Assim,

2
, Assim,

3
, Assim,

4
, Assim,

5
, Assim,

6
, Assim,

7
, Assim,

8
, Assim,

9
],

soma

=

0
;
para
(
deixar

elemento

de

dados
)

{

soma
```

Os objetos não são (por padrão) iterable.Tentando usar
para/de
em um
Objeto regular lança um TypeError em tempo de execução:
deixar

o

=

{

x
:

1
, Assim,

y
:

2
, Assim,

z
:

3

};
para
(
deixar

elemento

de

o
)

{

// joga TypeError porque O não é

iterável

console
.
registro
(
elemento
);
}
Se
Você quer iterar através das propriedades de um objeto, você pode usar
o
para/in
Loop (introduzido em

E
Se você estiver interessado nas chaves e nos valores de um objeto
propriedades, você pode usar
para/de
com
Object.entries ()
e
atribuição de destruição:
deixar

pares

=

""
;
para
(
deixar

[[
k
, Assim,

v
]

de

Objeto
.
entradas
(
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:

Para/de com set e mapa
O
As classes de conjunto e mapa do ES6 embutidas são iteráveis.
Quando você itera um
Definido com
para/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:

```
deixar

texto

=

"Na Na Na Na Na Na Na Batman!"
;
deixar

wordset

=

novo

Definir
(
texto
.
dividir
(
""
));
deixar

exclusivo

=

[];
para
(
deixar

palavra

de

wordset
)

{

exclusivo
.
empurrar
(
palavra
);
```

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

para
(
deixar

p

em

o
)

{

// atribui nomes de propriedades de O a

variável p

console
.
registro
(
o
[[
p
]);

// Imprima o valor de cada propriedade
}
Para executar a
para/in
declaração, o intérprete JavaScript primeiro
avalia o
objeto
expressão.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 o
variável
expressão e atribui o nome
da propriedade (um valor de string) a ela.
Observe que o
variável
no
para/in
Loop 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:
deixar

o

=

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 quer usar
para/de
em vez de
de
para/in
.
O
para/in
O loop 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
enumerável
propriedades (ver
§14.1
).Os vários métodos internos
Definido pelo JavaScript Core não é enumerável.Todos os objetos têm um
ToString ()
método, por exemplo, mas o
para/in
Loop não
enumerar isso
ToString
propriedade.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 em
§14.1
.)
Propriedades herdadas enumeráveis ■■(ver
§6.3.2
) também são enumerados por
o
para/in
laço.Isso significa que se você usar
para/in
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
para/de
loop com
Object.keys ()
em vez de um
para/in
laço.
Se o corpo de um
para/in
loop exclui 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.Ver
§6.6.1

qual
para/in
enumera as propriedades de um objeto.
5,5 saltos
Outro
categoria de declarações de javascript são
Jump declarações
.Como o
O nome indica, eles fazem com que o intérprete JavaScript salte para um novo
Localização no código -fonte.O
quebrar
declaração 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
rotulado
, e
quebrar
e
continuar
pode
Identifique o loop de destino ou outro rótulo de instrução.
O
retornar
declaração faz o intérprete saltar de uma função
invocação de volta ao código que o invocou e também fornece o valor
para a invocação.O
lançar
A declaração é um tipo de retorno interino
de uma função de gerador.O
lançar
A declaração aumenta, ou
joga
, um
exceção e foi projetado para trabalhar com o
tente/capturar/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
rotulado
precedendo 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
quebrar
e
continuar
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 a
continuar
declarar isso
usa o rótulo.
MAINLOOP
:

enquanto
(
token

! ==

nulo
)

{

// Código omitido ...

continuar

MAINLOOP
;

// pule para a próxima iteração do

Nomeado loop

// mais código omitido ...
}
O
identificador
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

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

quebrar
Declaração, usada sozinha, causa o loop interno de fechamento
ou
trocar
declaração para sair imediatamente.Sua sintaxe é simples:
quebrar
;
Porque causa um loop ou
trocar
Para sair, esta forma do
quebrar
A declaração é legal apenas se aparecer dentro de uma dessas declarações.
Você já viu exemplos do
quebrar
declaração dentro de um
trocar
declaração.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
quebrar
declarações
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 um
quebrar
declaração se encontrar o que é
Procurando na matriz:
para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

eu
++
)

{

quebrar
LabelName
;
Quando
quebrar
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
quebrar
Nesta forma, se não houver nenhuma declaração de anexo com o
etiqueta especificada.Com esta forma do
quebrar
declaração, o nome
a declaração não precisa ser um loop ou
trocar
:
quebrar
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 o
quebrar
palavra -chave e o
LabelName
.Isso é resultado da inserção automática de JavaScript de
omitido semicolons: se você colocar um terminador de linha entre o
quebrar
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.(Ver
§2.6
.)
Você precisa da forma rotulada do
quebrar
declaração quando você quiser
sair de uma declaração que não é o loop fechado mais próximo ou um
trocar
.O código a seguir
demonstra:
deixar

matriz

=

getData
();

// Obtenha uma variedade 2D de números de

em algum lugar
// Agora soma todos os números na matriz.
deixar

soma

=

```
para
(
deixar

y

=

0
;

y

<

linha
.
comprimento
;

y
++
)

{

deixar

célula

=

linha
[[
y
];

se

(
Isnan
(
célula
))

quebrar

Computesum
;

soma

+=

célula
;

}
```

ser usado apenas dentro do corpo de um loop.Usá -lo em qualquer outro lugar causa um erro de sintaxe.
Quando o
continuar
declaração 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
enquanto
Loop, o especificado
expressão
no início de
o loop é testado novamente e se for
verdadeiro
, o corpo do loop é
executado a partir do topo.
Em um
faça/while
loop, 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
para
Loop, o
incremento
a expressão é avaliada e o
teste
A expressão é testada novamente para determinar se outra iteração deve ser feito.
Em um
para/de
ou
para/in
Loop, o loop começa com o
Próximo valor iterado ou o próximo nome da propriedade sendo atribuído ao variável especificada.
Observe a diferença no comportamento do
continuar
declaração no
enquanto
e
para
Loops: a
enquanto
Loop retorna diretamente à sua condição,
mas a
para
O loop primeiro avalia seu
incremento
expressão e depois retorna
à sua condição.Anteriormente, consideramos o comportamento do
para
loop in
termos de um "equivalente"
enquanto
laço.Porque o
continuar
A declaração se comporta de maneira diferente para esses dois loops, no entanto, não é realmente possível simular perfeitamente um

O exemplo a seguir mostra um não marcado
continuar
declaração
sendo usado para pular o restante da iteração atual de um loop quando um
Ocorre erro:

```
para
(
deixar

eu

=

0
;

eu

<

dados
.
comprimento
;

eu
++
)

{

se

(
!
dados
[[
eu
]))

continuar
;

// não pode prosseguir com indefinidamente

dados

total

+=

dados
[[
eu
];
}
```

Como o
quebrar
declaração, o

quadrado
(
2
)

// => 4
Com não
retornar
Declaração, 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 avalia para
indefinido
.O
retornar
A declaração geralmente aparece como a última declaração em uma função, mas precisa
não ser o último: uma função retorna ao seu chamador quando um
retornar
declaração é
executado, mesmo que haja outras declarações restantes na função
corpo.
O
retornar
A declaração também pode ser usada sem um
expressão
para
Faça a função retornar
indefinido
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 o
retornar

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

eles também.Um objeto de erro tem um
nome
propriedade que especifica o
tipo de erro e um
mensagem
propriedade que mantém a sequência 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

novo

Erro
(
"X não deve ser negativo"
);

// caso contrário, calcule um valor e retorne normalmente

deixar

f
;

para
(
f

=

1
;

x

>

1

O

tente/capturar/finalmente
Declaração é a exceção de JavaScript
mecanismo de manuseio.O
tentar
Cláusula desta declaração simplesmente define
O bloco de código cujas exceções devem ser tratadas.O
tentar
bloquear
é seguido por um
pegar
Cláusula, que é um bloco de declarações que são
invocado quando uma exceção ocorre em qualquer lugar dentro do
tentar
bloquear.
O
pegar
Cláusula é seguida por um
finalmente
bloco contendo
Código de limpeza que é garantido para ser executado, independentemente do que
acontece no
tentar
bloquear.Ambos
pegar
e
finalmente
Os blocos são
opcional, mas a
tentar
O bloco deve ser acompanhado por pelo menos um desses
blocos.O
tentar
, Assim,
pegar
, e
finalmente
todos 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/capturar/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

}
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 o
pegar
A palavra -chave é geralmente 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 a
pegar
Cláusula tem escopo de bloco - ela é definida apenas
dentro do
pegar
bloquear.
Aqui está um exemplo realista do
tente/capturar
declaração.Ele usa o
fatorial()
método definido na seção anterior e no cliente
Métodos laterais de JavaScript
incitar()
e
alerta()
para entrada e
saída:
tentar

{

// Peça ao usuário para inserir um número

deixar

n

```
pegar
(
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 um
tente/capturar
declaração com não
finalmente
cláusula.
Embora
finalmente
não é usado com tanta frequência quanto
pegar
, pode ser útil.
No entanto, seu comportamento requer explicação adicional.O
finalmente
a cláusula é garantida para ser executada se alguma parte do
tentar
bloco é
executado, independentemente de como o código no
tentar
bloco completa.Isso é
geralmente usado para limpar após o código no
tentar
cláusula.
No caso normal, o intérprete JavaScript chega ao final do
tentar
bloquear e depois prossegue para o
finalmente
bloco, que executa
qualquer limpeza necessária.Se o intérprete deixou o
tentar
bloco por causa de
um
retornar
, Assim,
continuar
, ou
quebrar
declaração, o
finalmente
bloco é
Executado antes que o intérprete salte para seu novo destino.
Se ocorrer uma exceção no
tentar

exceção, essa exceção substitui qualquer exceção que estivesse no processo
de ser jogado.Se a
finalmente
Cláusula emitem a
retornar
declaração, o
o método retorna normalmente, mesmo que uma exceção tenha sido lançada e tenha
ainda não foi tratado.
tentar
e
finalmente
pode ser usado juntos sem um
pegar
cláusula.Em
Este caso, o
finalmente
O bloco é simplesmente o código de limpeza que é
garantido para ser executado, independentemente do que acontece no
tentar
bloquear.Lembre -se de que não podemos simular completamente um
para
loop com um
enquanto
loop porque o
continuar
declaração se comporta de maneira diferente para
os dois loops.Se adicionarmos um
tente/finalmente
declaração, podemos escrever um
enquanto
loop que funciona como um
para
loop e isso lida
continuar
declarações corretamente:

```
// simular para (
inicializar
;
teste
;
incremento
) corpo;
inicializar
;
enquanto(
teste
) {
tentar {
corpo
;}
finalmente {
incremento
;}
}
```

Observe, no entanto, que um
corpo
que contém a
quebrar
declaração se comporta

```
// como json.parse (), mas retorne indefinido em vez de jogar um erro
função

Parsejson
(
s
)

{

tentar

{

retornar

JSON
.
analisar
(
s
);

}

pegar

{

// algo deu errado, mas não nos importamos com o que foi

retornar

indefinido
;

}
}
```

5.6 Declarações diversas

Esse
A seção descreve as três declarações JavaScript restantes
-
com
, Assim,
Depurador
, e
"Use rigoroso"
.

5.6.1 com

O
com
a declaração executa 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
```

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:

```
documento
.
formas
[[
0
].
endereço
.
valor
```

Se você precisar escrever expressões como essa várias vezes, você pode
use o
com
declaração para tratar as propriedades do objeto de forma como
variáveis:

```
com
(
documento
.
formas
[[
0
]))

{

// Acesso a elementos do formulário diretamente aqui.Por exemplo:

nome
.
valor

=

""
;

endereço
.
valor

=

""
;

e-mail
.
valor

=

""
;
```

5.6.2 Depurador
O

Depurador
A declaração 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 chamado
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
)

{

se

(
o

===

indefinido
)

Depurador
;

// linha temporária para

fins de depuração

...

// o restante 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.

"Use rigoroso"
é
um
diretivo
introduzido no ES5.Diretivas não são
declarações (mas estão próximas o suficiente para que
"Use rigoroso"
está documentado
aqui).Existem duas diferenças importantes entre o
"usar
estrito"
Diretivas 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 um
"Use rigoroso"
Diretiva é indicar que o código
que se segue (no script ou função) é
código rigoroso
.O nível superior
(não função) o código de um script é um código rigoroso se o script tiver um
"usar
estrito"
diretivo.Um corpo de função é um código rigoroso se for definido
dentro de código rigoroso ou se tiver um
"Use rigoroso"
diretivo.Código passado
para o
avaliar ()
o método é um código rigoroso se
avaliar ()
é chamado de rigoroso
código ou se a sequência de código incluir um
"Use rigoroso"
diretivo.Em
adição ao código declarado explicitamente como rigoroso, qualquer código em um
aula
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 um explícito
"Use rigoroso"
diretivo.
Código rigoroso é executado em
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):
O
com
A declaração não é permitida 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,
pegar
parâmetro da cláusula, 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
esse
valor de
indefinido
.(Em não rigoroso
modo, as funções invocadas como funções são sempre passadas o
objeto global como seu
esse
valor.) Além disso, no modo rigoroso, quando
uma função é invocada com
chamar()
ou
aplicar()
(
§8.7.4
), o
esse
o valor é exatamente o valor passado como o primeiro argumento para
chamar()
ou
aplicar()
.(No modo não estrito,
nulo
e
indefinido
Os valores 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 passou 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
avaliar ()
.Este escopo é
descartado quando o
avaliar ()
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

quais elementos da matriz e parâmetros de função nomeados
Ambos se referem ao mesmo valor.
No modo rigoroso, um SyntaxError é jogado se o
excluir
O operador é seguido por um identificador não qualificado, como um
Parâmetro variável, função ou função.(No modo não
tal
excluir
a expressão 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
o
excluir
expressão avalia para
falso
.)
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 octal
Literais.)
No modo rigoroso, os identificadores
aval
e
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
pegar
bloquear.
No modo rigoroso, a capacidade de examinar a pilha de chamadas é
restrito.
argumentos.caller
e
argumentos.Callee
Ambos jogam um TypeError dentro de um rigoroso
função de modo.Funções de modo rigoroso também têm
chamador
e
argumentos
Propriedades que jogam TypeError quando lidas.

(Algumas implementações definem essas propriedades fora do padrão
em funções não rigíveis.)
5.7 declarações
O
palavras -chave
const
, Assim,
deixar
, Assim,
var
, Assim,
função
, Assim,
aula
, Assim,
importar
, e
exportar
não são tecnicamente declarações, mas eles 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
em vez de
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
O

const
, Assim,
deixar
, e
var
declarações são cobertas em
§3.10
.
Em ES6
E mais tarde,
const
declara constantes e
deixar
declara variáveis.Anterior
para ES6, o
var
a palavra -chave 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
var
em vez de
deixar
.
const

Tau

=

2
*
Matemática
.
Pi
;
deixar

raio

=

3
;
var

circunferência

=

Tau

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 um
gerador
.
Declarações de gerador usam o
função
palavra -chave, mas siga com
um asterisco.
§13.3
descreve funções assíncronas, que também são
declarado usando o
função
palavra -chave, mas são prefixadas com o
assíncrono
palavra -chave.
5.7.3 Classe
Em
ES6 e mais tarde, o
aula
a declaração 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:
aula

Círculo

{

construtor
(
raio
)

{

esse
.
r

=

raio
;

}

área
()

{

retornar

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
importar
.Módulos são o assunto de
Capítulo 10
, e
importar
e
exportar
são cobertos em detalhe em
§10.3
.
importar
Diretivas são usadas para importar um ou mais valores de outro Arquivo do código JavaScript e dê -lhes nomes no módulo atual.
importar
As diretrizes vêm em algumas formas diferentes.Aqui estão alguns Exemplos:
importar

Círculo

de

'./geometry/circle.js'
;
importar

{

Pi
, Assim,

Tau

}

de

'./geometry/constants.js'
;
importar

{

magnitude

como

hipotenusa

}

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 a forma especial
exportação padrão
:
exportar

const

Tau

=

2

*

Matemática
.
Pi
;
exportar

função

magnitude
(
x
, Assim,
y
)

{

retornar

Matemática
.
sqrt
(
x
*
x

+

y
*
y
);

}
exportar

padrão

aula

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
Dar um nome à declaração para uso com
quebrar
e
continuar
deixar
Declare e inicialize uma ou mais variáveis ■■escassas de blocos (Novo

sintaxe)
retornar
Retornar um valor de uma função
trocar
Filial multiway para
caso
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 que o
caso
Expressões são avaliadas em tempo de execução, faz o JavaScript
trocar
declaração muito diferente de (e menos eficiente que)
trocar
declaração

de C, C ++ e Java.Nessas línguas, o
caso
expressões devem ser tempo de compilação
constantes do mesmo tipo, e
trocar
As declarações geralmente podem se compilar muito
eficiente
saltar mesas
.
2
Quando consideramos o
continuar
declaração em
§5.5.3
, veremos que isso
enquanto
loop é
não é um equivalente exato do
para
laço.

Capítulo 6.
Objetos

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
, Assim,
cada um dos quais tem um nome e um valor.Os nomes de propriedades são geralmente
cordas (embora, como veremos em
§6.10.3
, 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

Erro ao traduzir esta página.

Tenha duas propriedades com o mesmo nome.O valor pode ser qualquer
Valor JavaScript, ou pode ser uma função Getter ou Setter (ou ambos).
Vamos aprender sobre funções getter e setter
§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.
Em
Além de seu nome e valor, cada propriedade tem três
propriedade
atributos
:
O
gravável
atributo
especifica se o valor do
A propriedade pode ser definida.
O
enumerável
atributo
Especifica se o nome da propriedade
é devolvido por um
para/in
laço.
O
configurável
atributo
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-defensor
Valores do atributo de propriedade para seus objetos.
6.2 Criando objetos
Objetos
pode ser criado com literais de objeto, com o
novo
palavra -chave e
com o
Object.create ()
função.As subseções abaixo

Mesma linha da expressão a que se aplicam.O
Terceira exceção envolve
funções definidas usando sintaxe concisa de "Arrow": o
=>
seta em si
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.

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
O operador cria e inicializa um novo objeto.O
novo
A palavra -chave deve ser seguida por uma invocação de função.UM
função usada em
Assim é chamado de
construtor
e serve para inicializar um recém -criado
objeto.O JavaScript inclui construtores para seus tipos internos.Para
exemplo:
deixar

o

=

novo

Objeto
();

// Crie um objeto vazio: o mesmo que {}.
deixar

um

=

novo

Variedade
();

// Crie uma matriz vazia: o mesmo que [].
deixar

d

=

novo

As funções próprias do construtor para inicializar objetos recém -criados.Fazendo isso está coberto em
Capítulo 9
.

6.2.3 Protótipos

Antes
Podemos 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 a
protótipo
, e o primeiro objeto herda as propriedades do
protótipo.
Todos
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 o
novo
palavra -chave e a
Invocação do construtor Use o valor do
protótipo
propriedade de
o construtor funciona como seu protótipo.Então o objeto criado por
novo objeto ()
herda de
Object.prototype
, assim como o
objeto criado por
{}
faz.Da mesma forma, o objeto criado por
novo
Variedade()
usos
Array.prototype
como seu protótipo e o objeto
criado por
nova data ()
usos
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 um
protótipo
propriedade.São esses objetos com
protótipo
propriedades que definem o
protótipos
para todo o outro
objetos.
Object.prototype
é um dos objetos raros que não tem 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 de
Object.prototype
, então um objeto de data criado por
nova data ()
herda as propriedades de ambos
Date.prototype
e
Object.prototype
.Esse
série vinculada de objetos de protótipo é
conhecido como a
Cadeia de protótipo
.
Uma explicação de como a herança da propriedade funciona em
§6.3.2
.
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 seu
protótipo
propriedade 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 em
§14.3
.
6.2.4 Object.Create ()
Object.create ()
cria
um novo objeto, usando seu primeiro argumento como
O protótipo desse objeto:
deixar

O1

=

Objeto
.
criar
({
x
:

1
, Assim,

y
:

2
});

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Expressões têm o mesmo valor:
objeto
.
propriedade
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
Array é conhecido como um
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:

```
deixar

addr

=

""
;
```

```
para
(
deixar

eu

=

0
;

eu

<

4
;

eu
++
)

{

addr

+=

cliente
[[
`Endereço
$ {
eu
}
`
]

+

"\ n"
;
}
```

Este código lê e concatena o
Endereço0
, Assim,
Endereço1
, Assim,
Endereço2
, e
endereço3
propriedades do
cliente
objeto.
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

Erro ao traduzir esta página.

JavaScript
Os objetos 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 o
Object.create ()
função para criar objetos com
Protótipos especificados.Vamos ver
Capítulo 9
, no entanto, que toda vez
você cria uma instância de uma classe com
novo
, 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 tem 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 a propriedade
x
é encontrado
ou até um objeto com um
nulo
O protótipo é pesquisado.Como você pode ver,
o

protótipo
atributo de um objeto cria uma cadeia ou lista vinculada
de onde as propriedades são herdadas:
deixar

o

=

{};

// o herda métodos de objeto de

Object.prototype
o
.
x

=

1
;

Agora suponha que você atribua à propriedade
x
do objeto
o
.Se
o
já
possui uma propriedade própria (não-herdada) nomeada
x
, então 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
herda uma propriedade somente leitura
nomeado
x
, por exemplo, a tarefa 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:

```
deixar

unitcircle

=

{

r
:

1

};

// um objeto para herdar

de
deixar

c

=
```

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, o fará em
o
, e deixará novamente a cadeia de protótipos não modificada.
6.3.3 Erros de acesso à propriedade
Propriedade
Expressões de acesso 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
Expressão de acesso à propriedade
boi
avalia para
indefinido
.Lembre -se disso
Nosso objeto de livro possui uma propriedade "subtítulo", mas não uma propriedade "legenda":
livro
.
Legenda

// => indefinido: a propriedade não existe
É um erro, no entanto, tentar consultar uma propriedade de um objeto que
não existe.O
nulo
e
indefinido
Os valores não têm propriedades,
E é um erro consultar propriedades desses valores.Continuando o
Exemplo anterior:
deixar

Len

=

livro
.
Legenda
.
comprimento
;

//! TypeError: indefinido

não tem comprimento
Expressões de acesso à propriedade falharão se o lado esquerdo do
.

uma propriedade
p
de um objeto
o
falha nessas circunstâncias:
o
tem uma propriedade própria
p
isso é somente leitura: não é possível
Defina somente leitura
propriedades.
o
tem uma propriedade herdada
p
isso é somente leitura: não é
possível esconder uma propriedade somente leitura herdada com um próprio
propriedade de mesmo nome.
o
não tem uma propriedade própria
p
;
o
não herdar um
propriedade
p
com um método de setter e
o
S.
extensível
atributo
(ver
§14.2
) é
falso
.Desde
p
ainda não existe em
o
, e
Se não houver um método Setter para chamar, então
p
deve ser adicionado a
o
.
Mas se
o
não é extensível, então nenhuma nova propriedade pode ser
definido nele.
6.4 Excluindo propriedades
O

excluir
operador (
§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 o valor da propriedade, mas no

UM
excluir
expressão avalia para
verdadeiro
Se a exclusão foi bem -sucedida ou se
O delete não teve efeito (como excluir uma propriedade inexistente).
excluir
também avalia
verdadeiro
Quando usado (sem sentido) com um
expressão que não é uma expressão de acesso à propriedade:
deixar

o

=

{
x
:

1
};

// O tem propriedade própria x e herda

Propriedade ToString
excluir

o
.
x

// => true: exclui a propriedade x
excluir

o
.
x

// => true: nada (x não existe)

Mas é verdade de qualquer maneira
excluir

o
.
ToString

// => true: não faz nada (a tostragem não é

uma propriedade própria)
excluir

1

// => true: absurdo, mas verdade de qualquer maneira
excluir
não remove as propriedades que têm um

```
global
.
x

=

1
;

// Crie um global configurável

propriedade (sem deixar ou var)
excluir

x

// => true: esta propriedade pode ser

excluído
No modo rigoroso, no entanto,
excluir
levanta um síntaxe se o seu operando estiver
um identificador não qualificado como
x
, e você tem que ser explícito sobre o
Acesso à propriedade:
excluir

x
;

// SyntaxError no modo rigoroso
excluir

global
.
x
;

// Isso funciona
```

6.5 Propriedades de teste

JavaScript
Objetos 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
em
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
em
O operador espera um nome de propriedade no lado esquerdo e um objeto

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

quebrar
Declaração, usada sozinha, causa o loop interno de fechamento
ou
trocar
declaração para sair imediatamente.Sua sintaxe é simples:
quebrar
;
Porque causa um loop ou
trocar
Para sair, esta forma do
quebrar
A declaração é legal apenas se aparecer dentro de uma dessas declarações.
Você já viu exemplos do
quebrar
declaração dentro de um
trocar
declaração.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
quebrar
declarações
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 um
quebrar
declaração se encontrar o que é
Procurando na matriz:
para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

eu
++
)

{

propriedade
Há uma coisa que
em
O operador pode fazer isso a propriedade simples
A técnica de acesso mostrada aqui não pode fazer.
em
pode distinguir entre
propriedades que não existem e propriedades que existem, mas foram definidas como
indefinido
.Considere este código:
deixar

o

=

{

x
:

indefinido

};

// a propriedade está explicitamente definida como

indefinido
o
.
x

! ==

indefinido

// => false: a propriedade existe mas

é indefinido
o
.
y

! ==

indefinido

// => false: propriedade nem mesmo

existe
"X"

em

o

// => true: a propriedade existe
"Y"

```
o
.
propriedadesenumerable
(
"ToString"
)

// => false: não

enumerável
para
(
deixar

p

em

o
)

{

// percorrer o

propriedades

console
.
registro
(
p
);

// imprime x, y e z,

mas não tostring
}
```

Para se proteger contra a enumitação de propriedades herdadas com
para/in
, você
pode adicionar uma verificação explícita dentro do corpo do loop:

```
para
(
deixar

p

em

o
)

{

se

(
!
```

Object.getownPropertySymbols ()
retorna
ter
propriedades cujos nomes são símbolos, sejam eles
enumerável.
Reflete.ownskeys ()
retorna
Todos os próprios nomes de propriedades, ambos
enumerável e não enumerável, e símbolo e símbolo.
(Ver
§14.6
.)
Existem exemplos do uso de
Object.keys ()
com um
para/de
loop in
§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 ()
, Assim,
Object.GetownPropertyNames ()
, Assim,
Object.getownPropertySymbols ()
, Assim,
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

Erro ao traduzir esta página.

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 obtidas 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:
Objeto
.
atribuir
(
o
, Assim,

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

=

Objeto
.
atribuir
({},

padrões
, Assim,

o
);

Vamos ver
§6.10.4
que você também pode expressar este objeto copiar e-
substituir a operação usando o
...
espalhe o operador assim:
o

=

{...
padrões
, Assim,

...
o
};
Também poderíamos evitar a sobrecarga da criação extra de objetos e
copiando escrevendo uma versão de
Object.assign ()
que cópias
propriedades apenas se estiverem faltando:
// como object.assign (), mas não substitui

propriedades
// (e também não lida com propriedades de símbolo)
função

mesclar
(
alvo
, Assim,

...
fontes
)

{

para
(
deixar

fonte

de

fontes
)

{

para
(
deixar

chave

de

6.8 Objetos serializados
Objeto

serialização
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:
deixar

o

=

{
x
:

1
, Assim,

y
:

{
z
:

[[
falso
, Assim,

nulo
, Assim,

""
]]};

// Defina um teste

objeto
deixar

s

=

JSON
.
stringify
(
o
);

Processo de Stringification.A documentação completa para essas funções é
em
§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
HASOWNPROPERTY ()
e
PropertyIsEnumerable ()
métodos,
por exemplo.(E também já cobrimos alguns estáticos
funções definidas no
Objeto
construtor, 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 a definir esses métodos de maneira mais geral para
uma classe inteira de objetos.
6.9.1 O método ToString ()
O

ToString ()
O método 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 padrão
ToString ()
o método não é muito informativo (embora seja
Útil para determinar a classe de um objeto, como veremos em
§14.4.3
).
Por exemplo, a seguinte linha de código simplesmente avalia na string
"[Objeto objeto]":
deixar

s

=

{

x
:

1
, Assim,

y
:

1

}.
ToString
();

// s == "[objeto objeto]"
Porque esse método padrão não exibe muita informação útil,
Muitas classes definem suas próprias versões de
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 o seu próprio
ToString ()
Método como este:
deixar

apontar

=

{

x
:

1
, Assim,

y
:

Tolocalestring ()
que tentam formatar números, datas e
vezes de acordo com as convenções locais.Array define um
Tolocalestring ()
método que funciona como
ToString ()
exceto
que formata os elementos da matriz chamando seus
Tolocalestring ()
métodos em vez de seus
ToString ()
Métodos.Você pode fazer o
A mesma coisa com um
apontar
objeto como este:
deixar

apontar

=

{

x
:

1000
, Assim,

y
:

2000
, Assim,

ToString
:

função
()

{

retornar

`(
$ {
esse
.
x
}
, Assim,
$ {
esse
.
y
}
) `
;

valueof ()
método.A classe de data define
valueof ()
para converter
datas para números, e isso permite que os objetos de data sejam cronologicamente
comparado com
<
e
>
.Você poderia fazer algo semelhante com um ponto
objeto, definindo a
valueof ()
método que retorna a distância de
a origem até o ponto:
deixar

apontar

=

{

x
:

3
, Assim,

y
:

4
, Assim,

valorof
:

função
()

{

retornar

Matemática
.
Hypot
(
esse
.
x
, Assim,

esse
.
y
);

}

Tojson
:

função
()

{

retornar

esse
.
ToString
();

}
};
JSON
.
stringify
([
apontar
]))

// => '["(1, 2)"]'
6.10 Sintaxe literal de objeto estendido
Recente
As versões do JavaScript estenderam a sintaxe para objeto
Literais de várias maneiras úteis.As seguintes subseções explicam
essas extensões.
6.10.1 Propriedades abreviadas
Suponha que você tenha valores armazenados em variáveis
x
e
y
e quero criar
um objeto com propriedades nomeadas
x
e
y
que mantêm esses valores.Com
Sintaxe literal de objeto básico, você acabaria repetindo cada identificador duas vezes:
deixar

x

=

1
, Assim,

y

=

2
;
deixar

Erro ao traduzir esta página.

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

O
A sintaxe de 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:

```
const

extensão

=

Símbolo
(
"Meu símbolo de extensão"
);
deixar

o

=

{

[[
extensão
]
:

{

/ * Dados de extensão armazenados neste objeto */

}
};
o
[[
extensão
].
x

=

0
;

// Isso não vai conflitar com outro
```

Propriedades de O.
Conforme explicado

função com que você invoca
novo
.) O valor retornado por
Símbolo()
não é igual a nenhum outro símbolo ou outro valor.Você pode passar uma corda
para
Símbolo()
, 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,
Claro, use
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

Em
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:

```
deixar

posição

=

{

x
:

0
, Assim,

y
:

0

};
deixar

dimensões

=

{

largura
```

Erro ao traduzir esta página.

um
Sobre)
operação.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 ineficiente
Sobre
)
algoritmo isso
não vai escalar bem como
n
fica maior.

6.10.5 Métodos de abreviação

Quando
Uma função é definida como uma propriedade de um objeto, chamamos isso
função a
método
(teremos muito mais a dizer sobre 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:

```
deixar

quadrado

=

{

área
:

função
()

{

retornar

esse
.
lado

*

esse
.
lado
;

},

lado
:
```

função especificada.A sintaxe abreviada deixa mais claro que
área()
é um método e não uma propriedade de dados como
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:
adição a um identificador javascript regular como o nome
área
acima,
Você também pode usar literais de cordas e nomes de propriedades computadas, que
pode incluir nomes de propriedades de símbolo:
const

Method_name

=

"M"
;
const

símbolo

=

Símbolo
();
deixar

Weirdmethods

=

{

"Método com espaços"
(
x
)

{

retornar

x

+

1
;

},

[[
Method_name
] (
x
)

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
indefinido
.
As propriedades do acessador podem ser definidas com uma extensão do objeto
Sintaxe literal (ao contrário das outras extensões ES6, temos
Visto aqui, getters
e os setters foram introduzidos no ES5):
deixar

o

=

{

// Uma propriedade de dados comum

DataProp
:

valor
, Assim,

// Uma propriedade acessadora definida como um par de funções.

pegar

ACESTORPROP
()

{

retornar

esse
.
DataProp
;

},

definir

ACESTORPROP
(
valor
)

Nomes de propriedades calculados ao definir getters e setters.Simplesmente
Substitua o nome da propriedade depois
pegar
ou
definir
com uma expressão em
Suportes quadrados.)
Os métodos de acessórios definidos acima simplesmente obtêm e definem o valor de um
propriedade de dados, e não há razão para preferir a propriedade do acessador
sobre a propriedade de dados.Mas como um exemplo mais interessante, considere o
a seguir o objeto que representa um ponto cartesiano 2D.Tem comum
propriedades de dados para representar o
x
e
y
coordenadas do ponto, e
possui propriedades de acessórios que dão as coordenadas polares equivalentes do
apontar:
deixar

p

=

{

// x e y são propriedades regulares de dados de leitura de leitura.

x
:

1.0
, Assim,

y
:

1.0
, Assim,

// r é uma propriedade de acessor de leitura-write com getter e

setter.

// Não se esqueça de colocar uma vírgula após métodos de acessórios.

pegar

r
()

{

retornar

Matemática
.
Hypot
(

Erro ao traduzir esta página.

```
// Defina um novo valor de n, mas apenas se for maior que

atual

definir

próximo
(
n
)

{

se

(
n

>

esse
.
_n
)

esse
.
_n

=

n
;

outro

lançar

novo

Erro
(
"O número de série só pode ser definido

para um valor maior "
);

}
};
Serialnum
.
próximo

=

10
;

// Defina o número de série inicial
```

e mais tarde.
Como ler, escrever, excluir, enumerar e verificar o
presença das propriedades de um objeto.
Como a herança baseada em protótipo funciona em javascript e como
para criar um objeto que herda de outro objeto com
Object.create ()
.
Como copiar propriedades de um objeto para outro com
Object.assign ()
.
Todos os valores de JavaScript que não são valores primitivos são objetos.Esse
inclui matrizes e funções, que são os tópicos dos próximos dois
capítulos.
1
Lembrar;Quase todos os objetos têm um protótipo, mas a maioria não tem uma propriedade chamada
protótipo
.A herança de JavaScript funciona mesmo que você não possa acessar o objeto de protótipo
diretamente.Mas veja
§14.3
Se você quiser aprender a fazer isso.

Capítulo 7.
Matrizes
Esse
Capítulo documenta as matrizes, um tipo de dados fundamental em javascript
e na maioria das outras linguagens de programação.Um
variedade
é
um ordenado
Coleção de valores.Cada valor é chamado de
elemento
, e cada elemento
tem uma posição numérica na matriz, conhecida como seu
índice
.JavaScript
Matrizes são
sem topo
: 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.
JavaScript
Matrizes são
baseado em zero
e use 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.
JavaScript
Matrizes são
dinâmico
: eles 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.JavaScript
Matrizes podem ser
escasso
: os elementos não precisam ter índices contíguos, e pode haver
ser lacunas.Cada matriz JavaScript tem um
comprimento
propriedade.Para não parse
Matrizes, esta propriedade especifica o número de elementos na matriz.Para
matrizes esparsas,
comprimento
é maior que o maior índice 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
32

mais rápido que o acesso a propriedades regulares de objetos.
Matrizes
herdar propriedades de
Array.prototype
, que define um
Rico conjunto de métodos de manipulação de matrizes, cobertos em
§7.8
.A maioria destes
Métodos são
genérico
, 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 nós discutiremos isso em
§7.10
.
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 bytes a dados binários e são abordados em
§11.2
.
7.1 Criando matrizes
Lá
são várias maneiras de criar matrizes.As subseções a seguir
Explique como criar matrizes com:
Matriz literais
O
...
Espalhe o operador em um objeto iterável
O
Variedade()
construtor
O
Array.of ()
e
Array.From ()
Métodos de fábrica
7.1.1 Literais da matriz
Por
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:

deixar

vazio

=

[];

// Uma matriz sem elementos
deixar

primos

=

[[
2
, Assim,

3
, Assim,

5
, Assim,

7
, Assim,

11
];

// Uma matriz com 5 numérico

elementos
deixar

misc

=

[[

1.1
, Assim,

verdadeiro
, Assim,

"um"
, Assim,

];

// 3 elementos de vários

Tipos + vírgula à direita
Os valores em uma matriz literal não precisam ser constantes;eles podem ser
arbitrário
Expressões:

Erro ao traduzir esta página.

Converta imediatamente o conjunto em uma matriz usando o operador de espalhamento:
deixar

cartas

=

[...
"Hello World"
];
[...
novo

Definir
(
cartas
)]

// => ["h", "e", "l", "o", "

"," W "," R "," D "]
7.1.3 O construtor Array ()
Outro
maneira de criar uma matriz é com o
Variedade()
construtor.Você
Pode invocar este construtor de três maneiras distintas:
Chame sem argumentos:
deixar

um

=

novo

Variedade
();
Este método cria uma matriz vazia sem elementos e é
equivalente à matriz literal
[]
.
Chame com um único argumento numérico, que especifica um
comprimento:
deixar

um

=

novo

Variedade
(
10
);
Esta técnica cria uma matriz com o comprimento especificado.Esse
forma do
Variedade()

"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
Variedade()
construtor.
7.1.4 Array.of ()
Quando
o
Variedade()
A função construtora 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 o
Variedade()
Construtor não pode ser usado para criar uma matriz com um único numérico
elemento.
Em
ES6, o
Array.of ()
função aborda este 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:
Variedade
.
de
()

// => [];retorna uma matriz vazia sem

argumentos
Variedade
.
de
(
10
)

// => [10];pode criar matrizes com um único

argumento numérico
Variedade
.
de
(
1
, Assim,
2
, Assim,
3
)

// => [1, 2, 3]
7.1.5 Array.From ()
Array.From
é
Outro método de fábrica de matriz introduzido no ES6.Isto

[... iterável]
faz.Também é uma maneira simples de fazer uma cópia de um
variedade:
deixar

cópia

=

Variedade
.
de
(
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 s
ão
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:
deixar

Truearray

=

Variedade
.
de
(
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
variedade
mapa()
método que será introduzido mais tarde 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
Acesse 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

Erro ao traduzir esta página.

o
[[
"1"
]

// => "One";nomes numéricos e de propriedades de string

são iguais
Isto
é útil para distinguir claramente um
Índice de Array
de um
objeto
Nome da propriedade
.Todos os índices são nomes de propriedades, mas apenas propriedade
nomes que são 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 seus
comprimento
propriedade 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
Inteiro:
um
[[
-
1.23
]

=

verdadeiro
;

// Isso cria uma propriedade chamada "-1,23"
um
[[
"1000"
]

=

0
;

// Este é o 1001º elemento da matriz
um
[[
1.000
]

um
[[
2
]

// => indefinido;nenhum elemento nisso

índice.
um
[[
-
1
]

// => indefinido;Sem propriedade com isso

nome.
7.3 Matrizes esparsas
UM

escasso
Array é aquele em que os elementos não têm contíguo
índices começando em 0. Normalmente, o
comprimento
propriedade de uma matriz
Especifica o número de elementos na matriz.Se a matriz for escassa, o
valor do
comprimento
A propriedade é maior que o número de elementos.
Matrizes esparsas podem ser criadas com o
Variedade()
construtor ou simplesmente
atribuindo a um índice de matriz maior que a matriz atual
comprimento
.
deixar

um

=

novo

Variedade
(
5
);

// Sem elementos, mas A. o comprimento é 5.
um

=

[];

// Crie uma matriz sem elementos e

comprimento = 0.
um

deixar

A1

=

[,];

// Esta matriz não tem elementos e

comprimento 1
deixar

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
indefinido
elementos.
7.4 Comprimento da matriz
Todo
Array tem um
comprimento
propriedade, e é essa propriedade que faz
Matrizes diferentes dos objetos JavaScript regulares.Para matrizes que são
denso (isto é, não escasso), o
comprimento
propriedade especifica o número de
elementos na matriz.Seu valor é mais do que o maior índice em

Manter esse invariante, as matrizes têm dois comportamentos especiais.O primeiro nós descrito acima: se você atribuir um valor a um elemento de matriz cujo índice
eu
é maior ou igual ao atual da matriz
comprimento
, o valor de
o
comprimento
A propriedade está definida como
I+1
.
O segundo comportamento especial que as matrizes implementam para manter O comprimento invariante é que, se você definir o
comprimento
propriedade para um não
Inteiro negativo
n
menor que seu valor atual, qualquer elementos de matriz cujo índice é maior ou igual a
n
são excluídos da matriz:
um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];

// Comece com uma matriz de 5 elementos.
um
.
comprimento

=

3
;

// a agora é [1,2,3].
um
.
comprimento

=

0
;

// Exclua todos os elementos.a é [].
um

Fim de uma matriz:
deixar

um

=

[];

// Comece com uma matriz vazia
um
.
empurrar
(
"zero"
);

// Adicione um valor no final.a =

["zero"]
um
.
empurrar
(
"um"
, Assim,

"dois"
);

// Adicione mais dois valores.a = ["zero",

"One", "Two"]
Empurrando um valor para uma matriz
um
é o mesmo que atribuir o valor a
A [A.Length]
.Você pode usar o
NIFT ()
Método (descrito em
§7.8
) para inserir um valor no início de uma matriz, mudando o
elementos de matriz para índices mais altos.O
pop ()
Método é o oposto de
empurrar()
: ele remove o último elemento da matriz e o devolve,
reduzindo o comprimento de uma matriz em 1. Da mesma forma, o
mudança()
método
remove e retorna o primeiro elemento da matriz, reduzindo o comprimento
por 1 e mudar todos os elementos para um índice um menor que o seu
Índice atual.Ver
§7.8
Para mais informações sobre esses métodos.
Você pode excluir elementos de matriz com o
excluir
operador, assim como você
pode excluir propriedades do objeto:

torna -se escasso.
Como vimos acima, você também pode remover elementos do final de um
Array simplesmente definindo o
comprimento
propriedade para o novo comprimento desejado.
Finalmente,
emenda ()
é o método de uso geral para inserção,
excluir ou substituir elementos de matriz.Altera o
comprimento
propriedade
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
Como
do ES6, a maneira mais fácil de percorrer cada um dos elementos de um
matriz (ou qualquer objeto iterável) é com o
para/de
Loop, que era
coberto em detalhes em
§5.4.4
:

```
deixar

cartas

=

[...
"Hello World"
];

// Uma variedade de letras
deixar

corda

=

""
;
para
(
deixar

carta

de

cartas
)

{

corda

+=
```

O exemplo a seguir mostra um não marcado
continuar
declaração
sendo usado para pular o restante da iteração atual de um loop quando um
Ocorre erro:

```
para
(
deixar

eu

=

0
;

eu

<

dados
.
comprimento
;

eu
++
)

{

se

(
!
dados
[[
eu
]))

continuar
;

// não pode prosseguir com indefinidamente

dados

total

+=

dados
[[
eu
];
}
```

Como o
quebrar
declaração, o

Você
também pode percorrer os elementos de uma matriz com um bom velho
formado
para
laço (
§5.4.3
):
deixar

vogais

=

""
;
para
(
deixar

eu

=

0
;

eu

<

cartas
.
comprimento
;

eu
++
)

{

// para cada índice

na matriz

deixar

carta

=

cartas
[[
eu
];

// Obtenha o elemento

Nesse índice

Elementos inexistentes, você pode escrever:
para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

eu
++
)

{

se

(
um
[[
eu
]

===

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
variável
matriz
é uma variedade de matrizes de números.Cada elemento em

7.8 Métodos de matriz

O
As seções anteriores se concentraram na sintaxe básica de javascript para
trabalhando com matrizes.Em geral, porém, são os métodos definidos por
A classe de matriz que é a mais poderosa.O próximo documento das seções
esses métodos.Ao ler sobre esses métodos, lembre -se de que
Alguns deles modificam a matriz que são chamados e outros
Deixe a matriz inalterada.Vários métodos retornam uma matriz:
Às vezes, essa é uma nova matriz, e o original permanece inalterado.Outro
tempos, um método modificará a matriz no lugar e também retornará um
referência à matriz modificada.
Cada uma das subseções a seguir cobre um grupo de matriz relacionada
Métodos:
Os métodos do iterador percorrem os elementos de uma matriz, normalmente
invocando uma função que você especifica em cada um deles
elementos.
Métodos de pilha e fila Adicionar e remover elementos de matriz para
e desde o início e o final de uma matriz.
Os métodos de subarray são para extrair, excluir, inserir,
recheio e copiando regiões contíguas de uma matriz maior.
Os métodos de pesquisa e classificação são para localizar elementos
dentro de uma matriz e para classificar os elementos de uma matriz.
As subseções a seguir também cobrem os métodos estáticos da matriz
classe e alguns métodos diversos para concatenar matrizes e
convertendo matrizes em cordas.

Erro ao traduzir esta página.

é definido em outro lugar.Sintaxe da função de seta (ver
§8.1.3
) funciona
particularmente bem com esses métodos, e nós o usaremos nos exemplos
isso a seguir.
Foreach ()
O
foreach ()
método
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:
deixar

dados

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
],

soma

=

0
;
// calcular a soma dos elementos da matriz
dados
.
foreach
(
valor

=>

{

soma

+=

quais elementos da matriz e parâmetros de função nomeados
Ambos se referem ao mesmo valor.
No modo rigoroso, um SyntaxError é jogado se o
excluir
O operador é seguido por um identificador não qualificado, como um
Parâmetro variável, função ou função.(No modo não
tal
excluir
a expressão 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
o
excluir
expressão avalia para
falso
.)
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 octal
Literais.)
No modo rigoroso, os identificadores
aval
e
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
pegar
bloquear.
No modo rigoroso, a capacidade de examinar a pilha de chamadas é
restrito.
argumentos.caller
e
argumentos.Callee
Ambos jogam um TypeError dentro de um rigoroso
função de modo.Funções de modo rigoroso também têm
chamador
e
argumentos
Propriedades que jogam TypeError quando lidas.

E para fechar lacunas e remover elementos indefinidos e nulos, você pode
usar
filtro
, assim:
um

=

um
.
filtro
(
x

=>

x

! ==

indefinido

&&

x

! ==

nulo
);
Encontre () e findIndex ()
O

encontrar()
e
FindIndex ()
Métodos são como
filtro()
nisso
eles iteram através da sua matriz procurando elementos para os quais seu
A função predicada retorna um valor verdadeiro.Diferente
filtro()
, no entanto,
Esses dois métodos param de iterar na primeira vez que o predicado encontra um
elemento.Quando isso acontece,
encontrar()
Retorna o elemento correspondente,
e
FindIndex ()
Retorna o índice do elemento correspondente.Se não
o elemento correspondente é encontrado,
encontrar()
retorna
indefinido
e
FindIndex ()
retorna
-1
:

Todos os elementos da matriz:
deixar

um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
um
.
todo
(
x

=>

x

<

10
)

// => true: Todos os valores são <10.
um
.
todo
(
x

=>

x

%

2

===

0
)

// => false: nem todos os valores são

até.
O
alguns()
O método é como o quantificador matemático "existe"

e também passa pelos nomes "injetar" e "dobrar".Exemplos de ajuda
ilustrar como funciona:
deixar

um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
um
.
reduzir
((
x
, Assim,
y
)

=>

x
+
y
, Assim,

0
)

// => 15;a soma do

valores
um
.
reduzir
((
x
, Assim,
y
)

=>

x
*
y
, Assim,

1
)

Você deve ter notado que a terceira chamada para
reduzir()
Neste exemplo
Tem apenas um argumento: não há valor inicial especificado.Quando
você invoca
reduzir()
Assim 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
reduzir()
Em uma matriz vazia sem argumento de valor inicial
causa a
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.
Reduteright ()
funciona como
reduzir()
, exceto que isso
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
deixar

um

=

[[
2
, Assim,

3
, Assim,

4
];
um
.
Reduceright
((
acc
, Assim,
Val
)

=>

Matemática
.
prisioneiro de guerra

objeto.
Os exemplos mostrados até agora foram numéricos por simplicidade, mas
reduzir()
e
Reduteright ()
não são destinados 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 ()
Em
ES2019, o
plano()
o método 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
, Assim,

[[
2
, Assim,

3
]].
plano
()

// => [1, 2, 3]
[[
1
, Assim,

[[
2
, Assim,

[[
3
]]].
plano
()

// => [1, 2, [3]]
Quando chamado sem argumentos,
plano()
Aplica um nível de ninho.
Elementos da matriz original que são próprios são achatados,
Mas elementos de matriz de
aqueles
Matrizes não são achatadas.Se você quiser
achatar mais níveis, passar um número para
plano()

O
Flatmap ()
o método funciona como o
mapa()
método (ver
"mapa()"
), exceto que a matriz retornada é automaticamente achatada como se
passou para
plano()
.Isto é, chamando
a.flatmap (f)
é o mesmo que
(mas mais eficiente do que)
a.map (f) .flat ()
:
deixar

frases

=

[[
"Hello World"
, Assim,

"O guia definitivo"
];
deixar

palavras

=

frases
.
Flatmap
(
frase

=>

frase
.
dividir
(
""
));
palavras

// => ["Hello", "World", "The", "Definitive", "Guide"];
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 faça

achatado
um
.
Concat
(
4
, Assim,

[[
5
, [
6
, Assim,
7
]])

// => [1,2,3,4,5, [6,7]];mas não

matrizes aninhadas
um

// => [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)
, Assim,
Então você deve pensar em modificar sua matriz no lugar com
empurrar()
ou
emenda ()
Em vez de criar um novo.
7.8.4 pilhas e filas com push (), pop (), shift (),
e não dividido ()
O

empurrar()
e
pop ()
métodos permitem que você trabalhe com matrizes como se
Eles eram pilhas.O
empurrar()
Método anexa um ou mais novos
Elementos até o final de uma matriz e retorna o novo comprimento da matriz.
Diferente
concat ()
, Assim,
empurrar()
não achate os argumentos da matriz.O
pop ()
O método 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
empurrar()
e

O
empurrar()
o método 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:
um
.
empurrar
(...
valores
);
O

NIFT ()
e
mudança()
métodos se comportam muito como
empurrar()
e
pop ()
, exceto que eles inserem e removem elementos do
Início de uma matriz e não do final.
NIFT ()
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.
mudança()
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
NIFT ()
e
mudança()
Para implementar uma pilha, mas seria menos
eficiente do que usar
empurrar()
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
empurrar()
Para adicionar elementos no final de um
Array e
mudança()
Para removê -los desde o início da matriz:
deixar

q

=

[];

// q == []

NIFT ()
, 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:
deixar

um

=

[];

// a == []
um
.
deserto
(
1
)

// a == [1]
um
.
deserto
(
2
)

// a == [2, 1]
um

=

[];

// a == []
um
.
deserto
(
1
, Assim,
2
)

// a == [1, 2]
7.8.5 subarrays with slice (), splice (), preench () e CopyWithin ()
Matrizes
Defina 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

fatiar()
O método retorna a
fatiar
, ou subarray, do especificado

```
deixar

um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
um
.
fatiar
(
0
, Assim,
3
);

// retorna [1,2,3]
um
.
fatiar
(
3
);

// retorna [4,5]
um
.
fatiar
(
1
, Assim,
-
1
);

// retorna [2,3,4]
um
.
fatiar
(
-
3
, Assim,
-
2
);

// retorna [3]
Emenda ()
```

```
um
.
emenda
(
1
, Assim,
2
)

// => [2,3];A agora é [1,4]
um
.
emenda
(
1
, Assim,
1
)

// => [4];a agora [1]
```

Os dois primeiros argumentos para
emenda ()
Especifique quais elementos da matriz
devem ser excluídos.Esses argumentos podem ser seguidos por qualquer número de
argumentos adicionais que especificam elementos a serem inseridos na matriz,
começando na posição especificada pelo primeiro argumento.Por exemplo:
deixar

```
um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
um
.
emenda
(
2
, Assim,
0
, Assim,
"um"
, Assim,
"B"
)

// => [];A agora é [1,2, "A", "B", 3,4,5]
um
.
```

Erro ao traduzir esta página.

regiões de destino.
7.8.6 Métodos de pesquisa e classificação de matrizes
Matrizes
implementar
indexOf ()
, Assim,
LastIndexOf ()
, e
inclui ()
métodos semelhantes aos mesmos métodos de
cordas.Existem também
organizar()
e
reverter()
Métodos para
reordenando os elementos de uma matriz.Esses métodos são descritos no
subseções a seguir.
IndexOf () e LastIndexOf ()
indexOf ()
e

LastIndexOf ()
Pesquise 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 a terminar e
LastIndexOf ()
Pesquisas do fim
começo:
deixar

um

=

[[
0
, Assim,
1
, Assim,
2
, Assim,
1
, Assim,
0
];
um
.
IndexOf
(
1
)

// => 1: a [1] é 1
um

Erro ao traduzir esta página.

Erro ao traduzir esta página.

variedade.Quando
organizar()
é chamado sem argumentos, ele classifica a matriz
elementos em ordem alfabética (convertendo temporariamente em strings
Para executar a comparação, se necessário):
deixar

um

=

[[
"banana"
, Assim,

"cereja"
, Assim,

"maçã"
];
um
.
organizar
();

// 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 para
organizar()
.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:
deixar

um

=

[[
33
, Assim,

4
, Assim,

1111
, Assim,

222
];
um

tipo alfabético insensível em uma variedade de cordas passando por um
função de comparação que converte seus dois argumentos em minúsculas
(com o
tolowerCase ()
método) antes de compará -los:
deixar

um

=

[[
"formiga"
, Assim,

"Erro"
, Assim,

"gato"
, Assim,

"Cachorro"
];
um
.
organizar
();

// a == ["bug", "cachorro", "formiga", "gato"];caso-
tipo sensível
um
.
organizar
(
função
(
s
, Assim,
t
)

{

deixar

um

=

s
.
tolowercase
();

deixar

b

=

O

juntar()
Método 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:
deixar

um

=

[[
1
, Assim,

2
, Assim,

3
];
um
.
juntar
()

// => "1,2,3"
um
.
juntar
(
""
)

// => "1 2 3"
um
.
juntar
(
''
)

// => "123"
deixar

b

=

novo

Variedade
(
10
);

// Uma variedade de comprimento 10 com não

para
(
deixar

eu

=

0
;

eu

<

um
.
comprimento
;

um
[[
eu
++
]

=

0
)

/* vazio */

;
5.3 Condicionais
Declarações condicionais
executar ou pular 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
se/else
declaração, e também cobre
trocar
, um mais complicado,
Declaração de Multiway Branch.
5.3.1 se
O

se
declaração 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
)

Erro ao traduzir esta página.

Documentos html (como
Document.QuerySelectorAll ()
, Assim,
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

isarray
(
o
)

{

se

(
o

&&

// O não é nulo,

indefinido, etc.

typeof

o

===

"objeto"

&&

// O é um objeto

Número
.
isfinita
(
o
.
comprimento
)

&&

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 invocá -los indiretamente usando o
Function.Call
Método, no entanto (ver
§8.7.4
Para detalhes):
deixar

um

=

{
"0"
:

"um"
, Assim,

"1"
:

"B"
, Assim,

"2"
:

"C"
, Assim,

comprimento
:

3
};

// Um

objeto semelhante a uma matriz
Variedade
.
protótipo
.
juntar
.
chamar
(
um
, Assim,

"+"
)

// =>

O
typeof
O operador ainda retorna "string" para strings, é claro, e
o
Array.esArray ()
Retorna de método
falso
Se você passar um
corda.
O principal benefício das seqüências indexíveis é simplesmente que podemos substituir
chamadas para
charat ()
com suportes quadrados, 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:
Variedade
.
protótipo
.
juntar
.
chamar
(
"JavaScript"
, Assim,

""
)

// => "J A V 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
empurrar()
, Assim,
organizar()
, Assim,
reverter()
, e
emenda ()
modificar 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 da praça
Suportes.
Elementos de matriz individuais são acessados ■■especificando o
Índice de matriz desejado dentro de colchetes.

O
para/de
loop e
...
Operador espalhado introduzido no ES6
são maneiras particularmente úteis de iterar matrizes.
A classe da matriz define um rico conjunto de métodos para manipular
matrizes, e você deve se familiarizar com o
API da matriz.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Métodos.Esta sintaxe de definição de função foi coberta em
§6.10.6
.
Observe que as funções também podem ser definidas com o
Função()
construtor, que é o assunto de
§8.7.7
.Além disso, o JavaScript define
Alguns tipos especializados de funções.
função*
define gerador
funções (ver
Capítulo 12
) e
função assíncrona
define
funções assíncronas (veja
Capítulo 13
).
8.1.1 declarações de função
As declarações de função consistem no
função
palavra -chave, 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
(
deixar

p

em

o
)

```
}
}
// calcula a distância entre os pontos cartesianos (x1, y1) e

(x2, y2).
função

distância
(
x1
, Assim,

Y1
, Assim,

x2
, Assim,

y2
)

{

deixar

dx

=

x2

-

x1
;

deixar

dy

=

y2

-

Y1
;

retornar

Matemática
.
sqrt
(
dx
*
dx
```

O valor de retorno da função é
indefinido
.
O

PrintProps ()
A função é diferente: seu trabalho é produzir o
Nomes e valores das propriedades de um objeto.Nenhum valor de retorno é
necessário, e a função não inclui um
retornar
declaração.O
valor de uma invocação do
PrintProps ()
A função é sempre
indefinido
.Se uma função não contém um
retornar
declaração, isso
simplesmente executar cada declaração no corpo da função até atingir o
fim e retorna o
indefinido
valor para o chamador.
Antes do ES6, as declarações de função eram permitidas apenas no nível superior
dentro de um arquivo JavaScript ou dentro de outra função.Enquanto alguns
As implementações dobraram a regra, não era tecnicamente legal definir
funções dentro do corpo de loops, condicionais ou outros blocos.Em
o
Modo rigoroso de ES6, no entanto, as declarações de função são permitidas dentro de
blocos.Uma função definida em um bloco existe apenas dentro desse bloco,
No entanto, e não é visível fora do bloco.
8.1.2 Expressões de função
Expressões de função
Parece muito com declarações de função, mas elas
aparecer no contexto de uma expressão ou afirmação maior, e o
O nome é opcional.Aqui estão algumas expressões de função de exemplo:

```
// Esta expressão de função define uma função que quadrar

seu argumento.
// Observe que o atribuímos a uma variável
const

quadrado

=

função
(
x
)

{

retornar

x
*
x
;
```

```
retornar

x
*
fato
(
x
-
1
);

};
// As expressões de função também podem ser usadas como argumentos para

Outras funções:
[[
3
, Assim,
2
, Assim,
1
].
organizar
(
função
(
um
, Assim,
b
)

{

retornar

um
-
b
;

});
// As expressões de função às vezes são definidas e imediatamente

invocado:
deixar

tensquared

=

(
função
(
x
)

{
retornar
```

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

Em
ES6, você pode definir funções usando uma sintaxe particularmente compacta
conhecido como "funções de seta".Esse
A sintaxe é uma reminiscência de
notação matemática e usa um
=>
"Arrow" para separar a função
parâmetros do corpo da função.O
função
A palavra -chave 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 pelo
=>
Arrow, seguida pelo corpo da função em Curly
aparelho ortodôntico:

```
const

soma

=

(
x
, Assim,

y
)

=>

{

retornar

x

+

y
;

};
```

Mas as funções de seta suportam uma sintaxe ainda mais compacta.Se o corpo
da função é um único
retornar
declaração, você pode omitir o
retornar
palavra -chave, o semicolon que o acompanha, e o encaracolado

Além disso, se uma função de seta tiver exatamente um parâmetro, você pode
Omita os parênteses em torno da lista de parâmetros:
const

polinomial

=

x

=>

x
*
x

+

2
*
x

+

3
;
Observe, no entanto, que uma função de seta sem argumentos deve ser
Escrito com um par de parênteses vazios:
const

ConstantFunc

=

()

=>

42
;
Observe que, ao escrever uma função de seta, você não deve colocar uma nova linha
entre os parâmetros da função e o
=>
seta.Caso contrário, você
poderia acabar com uma linha como
const polinomial = x
, que é um
Declaração de atribuição sintaticamente válida por conta própria.
Além disso, se o corpo da sua função de seta for um único
retornar
declaração
Mas a expressão a ser devolvida é um objeto literal, então você tem que
Coloque o objeto literal interno parênteses para evitar a ambiguidade sintática
entre os aparelhos encaracolados de um corpo de função e os aparelhos encaracolados de um
Objeto literal:
const

f

Declaração, então uma função que retorna indefinida é criada.No quarto
linha, no entanto, o objeto mais complicado literal não é válido
declaração, e esse código ilegal causa um erro de sintaxe.
A sintaxe concisa das funções de seta os torna ideais quando você
precisa passar uma função para outra função, o que é uma coisa comum
para ver com métodos de matriz como
mapa()
, Assim,
filtro()
, e
reduzir()
(ver
§7.8.1
), por exemplo:
// Faça uma cópia de uma matriz com elementos nulos removidos.
deixar

filtrado

=

[[
1
, Assim,
nulo
, Assim,
2
, Assim,
3
].
filtro
(
x

=>

x

! ==

nulo
);

//

filtrado == [1,2,3]
// quadrado alguns números:
deixar

quadrados

=

[[
1
, Assim,
2
, Assim,
3

```
função

quadrado
(
x
)

{

retornar

x
*
x
;

}

retornar

Matemática
.
sqrt
(
quadrado
(
um
)

+

quadrado
(
b
));
}
```

O
coisa interessante sobre as funções aninhadas é o 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
quadrado()
pode ler e escrever os parâmetros
um
e
b
definido pela função externa
hipotenusa()
.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

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
});
deixar

total

=

distância
(
0
, Assim,
0
, Assim,
2
, Assim,
1
)

+

distância
(
2
, Assim,
1
, Assim,
3
, Assim,
5
);
deixar

probabilidade

=

fatorial
(
5
)
/
fatorial
(
13
);
```

Em uma invocação, cada expressão de argumento (aqueles entre os

expressão
f?. (X)
é equivalente (assumindo nenhum efeito colateral) a:
(
f

! ==

nulo

&&

f

! ==

indefinido
)

?

f
(
x
)

:

indefinido
Detalhes completos sobre esta sintaxe de invocação condicional estão em
§4.5.1
.
Para
invocação de funções no modo não rigoroso, o contexto de invocação (o
esse
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
esse
valor que está em vigor onde eles são definidos.
Funções
escrito para ser invocado como funções (e não como métodos)
normalmente não usa o
esse
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

estrito

=

(

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
, Invoque assim:
o
.
m
();
Ou, se
m ()
Espera dois argumentos, você pode invocar assim:
o
.
m
(
x
, Assim,

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,

```
deixar

calculadora

=

{

// um objeto literal

operand1
:

1
, Assim,

operand2
:

1
, Assim,

adicionar
()

{

// Estamos usando a sintaxe abreviada do método para

esta função

// observe o uso da palavra -chave para se referir ao

contendo objeto.

esse
.
resultado

=

esse
.
operand1

+

esse
.
operand2
;

}
};
calculadora
.
adicionar
();
```

Erro ao traduzir esta página.

esse
O 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 seu
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
esse
para
Obtenha o contexto de invocação do método.O código a seguir
demonstra o problema:
deixar

o

=

{

// um objeto o.

m
:

função
()

{

// método m do objeto.

deixar

auto

=

esse
;

// salve o valor "Este" em um

variável.

esse

===

o

// => true: "this" é o objeto o.

f
();

// Agora chame a função auxiliar

Fim de uma matriz:
deixar

um

=

[];

// Comece com uma matriz vazia
um
.
empurrar
(
"zero"
);

// Adicione um valor no final.a =

["zero"]
um
.
empurrar
(
"um"
, Assim,

"dois"
);

// Adicione mais dois valores.a = ["zero",

"One", "Two"]
Empurrando um valor para uma matriz
um
é o mesmo que atribuir o valor a
A [A.Length]
.Você pode usar o
NIFT ()
Método (descrito em
§7.8
) para inserir um valor no início de uma matriz, mudando o
elementos de matriz para índices mais altos.O
pop ()
Método é o oposto de
empurrar()
: ele remove o último elemento da matriz e o devolve,
reduzindo o comprimento de uma matriz em 1. Da mesma forma, o
mudança()
método
remove e retorna o primeiro elemento da matriz, reduzindo o comprimento
por 1 e mudar todos os elementos para um índice um menor que o seu
Índice atual.Ver
§7.8
Para mais informações sobre esses métodos.
Você pode excluir elementos de matriz com o
excluir
operador, assim como você
pode excluir propriedades do objeto:

deixar

vazio

=

[];

// Uma matriz sem elementos
deixar

primos

=

[[
2
, Assim,

3
, Assim,

5
, Assim,

7
, Assim,

11
];

// Uma matriz com 5 numérico

elementos
deixar

misc

=

[[

1.1
, Assim,

verdadeiro
, Assim,

"um"
, Assim,

];

// 3 elementos de vários

Tipos + vírgula à direita
Os valores em uma matriz literal não precisam ser constantes;eles podem ser
arbitrário
Expressões:

8.2.4 Invocação indireta

JavaScript
funções são objetos e, como todos os objetos JavaScript, eles
tem métodos.Dois desses métodos,
chamar()
e
aplicar()
, Invoque
a função indiretamente.Ambos os métodos permitem especificar explicitamente
o
esse
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
chamar()
o método usa sua própria lista de argumentos como
argumentos para a função e o
aplicar()
Método espera uma matriz
de valores a serem usados ■■como argumentos.O
chamar()
e
aplicar()
Os métodos são descritos em detalhes em
§8.7.4
.

8.2.5 Invocação de função implícita

Lá
são 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.
Ver
§6.10.6
Para mais informações.
Quando um objeto é usado em um contexto de string (como quando é

concatenado com uma string), seu
ToString ()
Método é
chamado.Da mesma forma, quando um objeto é usado em um contexto numérico,
isso é
valueof ()
O método é chamado.Ver
§3.9.3
Para detalhes.
Quando você percorre os elementos de um objeto iterável, lá
são uma série de chamadas de método que ocorrem.
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 em
§14.7
) ter 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
Definições de função JavaScript
não especifique 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
, Assim,

um
)

{

se

(
um

===

indefinido
)

um

=

[];

// Se indefinido, use um novo

variedade

para
(
deixar

propriedade

em

o
)

um
.
empurrar
(

Observe que ao projetar funções com argumentos opcionais, você
Deve ter certeza de colocar os opcionais no final da lista de argumentos
para que eles possam ser omitidos.O programador que chama sua função
não pode omitir o primeiro argumento e passar no segundo: eles teriam que
passar explicitamente
indefinido
como o primeiro argumento.
No ES6 e mais tarde, você pode definir um valor padrão para cada um de seus
Parâmetros de função diretamente na lista de parâmetros da sua função.
Basta seguir o nome do parâmetro com um sinal igual e o padrão
valor a ser usado quando nenhum argumento é fornecido para esse parâmetro:
// 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
, Assim,

um

=

[])

{

para
(
deixar

propriedade

em

o
)

um
.
empurrar
(
propriedade
);

retornar

um
;
}
Expressões padrão do parâmetro são avaliadas quando sua função é
chamado, não quando é definido, então cada vez que isso
getPropertyNames ()
A função é invocada com um argumento, um

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 pode aceitar qualquer número de
argumentos são chamados
funções variádicas
, Assim,
funções variáveis ■■de arity
, ou
funções vararg
.Este livro usa o termo mais coloquial,
varargs
, Assim,
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
...
Operador espalhado, descrito em
§8.3.4
, o que pode
ser usado em função
invocações.
8.3.3 O objeto de argumentos
Descansar
Os parâmetros 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 o
max ()
função de anterior,

reescrito para usar o objeto Argumentos em vez de um parâmetro REST:
função

máx
(
x
)

{

deixar

maxvalue

=

-
Infinidade
;

// percorre os argumentos, procurando e

Lembrando, o maior.

para
(
deixar

eu

=

0
;

eu

<

argumentos
.
comprimento
;

eu
++
)

{

se

(
argumentos
[[
eu
]

>

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 obtidas 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:
Objeto
.
atribuir
(
o
, Assim,

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

=

Objeto
.
atribuir
({},

padrões
, Assim,

o
);

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 exige
permissão.
Agradecemos, mas geralmente não exigem, atribuição.Uma atribuição
Geralmente inclui o título, autor, editor e ISBN.Por exemplo:
“
JavaScript: O Guia Definitivo
, Sétima edição, por 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,
O'Reilly Media
forneceu tecnologia e negócios
Treinamento, conhecimento e insight para ajudar as empresas a ter sucesso.

}
Não há nada de especial nesse código.É apenas uma série de
se
declarações, onde cada um seguindo
se
faz parte do
outro
Cláusula do
declaração anterior.Usando o
caso contrário, se
o idioma é preferível e
mais legível do que, escrevendo essas declarações
Formulário equivalente e totalmente aninhado:
se

(
n

===

1
)

{

// Execute o bloco de código nº 1
}
outro

{

se

(
n

===

2
)

{

// Executar o bloco de código #2

}

outro

{

se

(
n

===

3

```
params
)
{

retornar

{

x
:

x1

+

x2
, Assim,

y
:

Y1

+

y2

};
}
VectorAdd
({
x
:

1
, Assim,

y
:

2
},

{
x
:

3
, Assim,

y
:

4
})

// => {x: 4, y: 6}
```

A coisa complicada de destruir a sintaxe como
{x: x1, y: y1}

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 assim:
função

Arraycopy
({
de
, Assim,

para
=
de
, Assim,

n
=
de
.
comprimento
, Assim,

Fromindex
=
0
, Assim,

toIndex
=
0
})

{

deixar

valuestocópia

=

de
.
fatiar
(
Fromindex
, Assim,

Fromindex

+

n
);

para
.

Propriedades que não foram destruídas.Objeto
Os parâmetros de descanso são
muitas vezes útil com o operador de espalhamento de objeto, que também é um novo
Recurso do ES2018:
// multiplique o vetor {x, y} ou {x, y, z} por um valor escalar,

retém outros adereços
função

vectormultiply
({
x
, Assim,

y
, Assim,

z
=
0
, Assim,

...
adereços
},

escalar
)

{

retornar

{

x
:

x
*
escalar
, Assim,

y
:

y
*
escalar
, Assim,

z
:

z
*
escalar
, Assim,

JavaScript
Os parâmetros do método não têm tipos declarados, e nenhum tipo
A verificação é executada nos valores que você passa para uma função.Você pode
ajuda
Faça seu código auto-documentação escolhendo nomes descritivos
para argumentos de função e documentá -los cuidadosamente no
Comentários para cada função.(Alternativamente, veja
§17.8
para um idioma
Extensão que permite que você verifique o tipo de verificação em cima de
Javascript.)
Conforme descrito em
§3.9
, JavaScript realiza 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 têm
ToString ()
métodos (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

soma
(
um
)

{

deixar

total

=

0
;
```

Erro ao traduzir esta página.

```
deixar

s

=

quadrado
;

// agora s refere -se à mesma função que

quadrado faz
quadrado
(
4
)

// => 16
s
(
4
)

// => 16
```

As funções também podem ser atribuídas às propriedades do objeto, em vez de variáveis.Como já discutimos, chamamos as funções de "métodos"
Quando fazemos isso:

```
deixar

o

=

{
quadrado
:

função
(
x
)

{

retornar

x
*
x
;

}};

// um objeto

literal
deixar

y
```

§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

adicionar
(
x
, Assim,
y
)

{

retornar

x

+

y
;

}
função

subtrair
(
x
, Assim,
y
)

{

retornar

x

-

y
;

}
função

multiplicar
(
x
, Assim,
y
)

```
// A sintaxe usada para chamar a função do operador.
função

operar2
(
operação
, Assim,

operand1
, Assim,

operand2
)

{

se

(
typeof

operadores
[[
operação
]

===

"função"
)

{

retornar

operadores
[[
operação
] (
operand1
, Assim,

operand2
);

}

outro

lançar

"Operador desconhecido"
;
}
operar2
(
"adicionar"
, Assim,
```

// ele usa uma propriedade de si mesma para lembrar o próximo valor para

ser devolvido.
função

Único
()

{

retornar

Único
.
contador
++
;

// retornar e incremento

contra -propriedade
}
Único
()

// => 0
Único
()

// => 1
Como outro exemplo, considere o seguinte
fatorial()
função
que usa propriedades de si mesma (tratando -se como uma matriz) para armazenar em cache
Anteriormente calculado
Resultados:
// calcular fatoriais e resultados de cache como propriedades do

função em si.
função

fatorial
(
n
)

{

se

(
Número
.
isinteger
(
n
)

&&

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:
(
função
()

{

// função de chunknamespace () reescrita como um

expressão sem nome.

// pedaço de código vai aqui
} ());

// termina a função literal e invocar agora.

Esse
técnica de definir e invocar uma função em um único
A expressão é usada com frequência o suficiente para se tornar idiomática e
recebeu o nome "Expressão da função imediatamente invocada".
Observe o uso de parênteses no exemplo de código anterior.O aberto
parênteses antes
função
é necessário porque sem ele, o
O intérprete de JavaScript tenta analisar o
função
palavra -chave como a
declaração de declaração de função.Com o parêntese, o intérprete
reconhece corretamente isso como uma expressão de definição de função.O
Os parênteses principais também ajudam os leitores humanos a reconhecer quando um
A função está sendo definida para ser imediatamente invocada em vez de definida
para uso posterior.
Esse uso de funções como espaço para nome se torna realmente útil quando nós
Defina uma ou mais funções dentro da função de namespace usando
variáveis ■■dentro dessa namespce, mas depois as transmitem como o
Valor de retorno da função de espaço para nome.Funções como essa são conhecidas
como
fechamentos
, e eles são o tópico da próxima seção.

8.6 fechamentos

Como
Mais linguagens de programação modernas, JavaScript usa
lexical
escopo
.Isso significa que as funções são executadas usando a variável
escopo que estava em vigor quando eles foram definidos, não o escopo variável
Isso está em vigor quando eles são invocados.Para implementar lexical
Escopo, o estado interno de um objeto de função JavaScript deve incluir
não apenas o código da função, mas também uma referência ao escopo em
que a definição da função aparece.Esse
combinação de uma função
objeto e um escopo (um conjunto de ligações variáveis) nas quais a função da função

As variáveis ■■são resolvidas é chamada de
encerramento
na ciência da computação
literatura.
Tecnicamente, tudo
As funções JavaScript são fechamentos, mas porque a maioria
As funções são invocadas do mesmo escopo em que foram definidas, é
Normalmente não importa realmente que haja um fechamento envolvido.Fechamentos
Torne -se interessante quando eles são invocados de um escopo diferente de
aquele em que foram definidos. Isso acontece mais comumente quando um
O objeto de função aninhado é devolvido da função em que estava
definido.Existem várias técnicas poderosas de programação que
envolva esse tipo de fechamento de funções aninhadas, e seu uso se tornou
relativamente comum na programação JavaScript.Os fechamentos podem parecer
confuso quando você os encontra pela primeira vez, mas é importante que você
entenda -os bem o suficiente para usá -los
confortavelmente.
O
O primeiro passo para entender o fechamento é revisar o escopo lexical
Regras para funções aninhadas.Considere o seguinte código:
deixar

escopo

=

"Escopo global"
;

// uma variável global
função

ChecksCope
()

{

deixar

escopo

=

"Escopo local"
;

// Uma variável local

função

f
()

{

retornar

escopo
;

Código retornará?
deixar

escopo

=

"Escopo global"
;

// uma variável global
função

ChecksCope
()

{

deixar

escopo

=

"Escopo local"
;

// Uma variável local

função

f
()

{

retornar

escopo
;

}

// retorna o valor em

escopo aqui

retornar

f
;
}
deixar

s

=

ChecksCope

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
ÚnicoInteger ()
função 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
ÚnicoInteger ()
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:

```
deixar

Único

=

(
função
()

{

// Defina e invocar

deixar

contador

=

0
;

// estado privado de

função abaixo

retornar

função
()

{

retornar

contador
++
;

};
} ());
Único
()

// => 0
Único
()
```

Variáveis ■■privadas como
contador
não precisa ser exclusivo para 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
()

{

deixar

n

=

0
;

retornar

{

contar
:

função
()

{

retornar

n
++
;

},

reiniciar
:

função
()

{

n

=

0
;

}
```

Isto
vale a pena notar aqui que você pode combinar esta técnica de fechamento
com getters e setters de propriedades.A seguinte versão do
contador()
a função é uma variação no código que apareceu em
§6.10.6
, Assim,
Mas ele usa fechamentos para estado privado, em vez de confiar regularmente
Propriedade do objeto:
função

```
contador
(
n
)

{

// O argumento da função n é o privado

variável

retornar

{

// Método Getter de propriedade retorna e incrementos

Contador privado var.

pegar

contar
()

{

retornar

n
++
;

},

// O Setter Property não permite o valor de n para

diminuir

definir

contar
(
m
)

{

se
```

Técnica de fechamentos que estamos demonstrando aqui.Este exemplo
define um
AddPrivateProperty ()
função que define um privado
variável e duas funções aninhadas para obter e definir o valor disso
variável.Adiciona essas funções aninhadas como métodos do objeto que você
especificar.
Exemplo 8-2.
Métodos de acessadores de propriedade privada usando fechamentos
// Esta função adiciona métodos de acessórios de propriedade para uma propriedade

com
// O nome especificado para o objeto o.Os métodos são nomeados

Get <name>
// e set <name>.Se uma função predicada for fornecida, o

setter
// O método o usa para testar seu argumento quanto à validade antes

armazenando.
// Se o predicado retornar falso, o método do setter lança um

exceção.
//
// o incomum sobre esta função é que a propriedade

valor
// que é manipulado pelos métodos getter e setter

armazenado em
// O objeto o.Em vez disso, o valor é armazenado apenas em um local

variável
// Nesta função.Os métodos getter e setter também são

definido
// localmente para esta função e, portanto, tem acesso a isso

variável local.
// isso significa que o valor é privado para os dois acessadores

métodos, e isso
// não pode ser definido ou modificado, exceto através do método do setter.
função

AddPrivateProperty
(
o
, Assim,

nome
, Assim,

predicado
)

{

```
lançar

novo

TypeError
(
`Set
$ {
nome
}
: valor inválido

$ {
v
}
`
);

}

outro

{

valor

=

v
;

}

};
}
// O código a seguir demonstra o addPrivateProperty ()

método.
deixar

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
, Assim,

"Nome"
, Assim,

x
```

Isso define os fechamentos.Pense no código a seguir, por exemplo:

```
// retorna uma variedade de funções que retornam os valores 0-9
função

constfuncs
()

{

deixar

funções

=

[];

para
(
var

eu

=

0
;

eu

<

10
;

eu
++
)

{

funções
[[
eu
]

=

()

=>

eu
;

}

retornar
```

Para todas as outras iterações, e cada um desses escopos tem seu próprio
ligação independente de
eu
.
Outra coisa a lembrar ao escrever o fechamento é que
esse
é a
Palavra -chave JavaScript, não uma variável.Como discutido anteriormente, Arrow
funções herdam o
esse
valor da função que os contém, mas
funções definidas com o
função
palavra -chave não.Então, se você é
escrevendo um fechamento que precisa usar o
esse
valor de seu contendo
função, você deve usar uma função de seta ou ligar
vincular()
, no
Fechar antes de devolvê -lo ou atribuir a parte externa
esse
valor para uma variável
que seu fechamento vai
herdar:
const

auto

=

esse
;

// Torne este valor disponível para

funções aninhadas
8.7 Propriedades, métodos e métodos da função
Construtor
Nós temos
Visto que as funções são valores nos programas JavaScript.O
typeof
O operador 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 um
Função()
construtor para criar novos objetos de função.As subseções a seguir
documentar o
comprimento
, Assim,
nome
, e
protótipo
propriedades;o
chamar()
, Assim,
aplicar()
, Assim,

O
somente leitura
comprimento
propriedade de uma função especifica o
ARITY
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
propósitos disso
comprimento
propriedade.

8.7.2 A propriedade Nome

O
somente leitura
nome
propriedade de uma função especifica o nome que era
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

Todos
funções, exceto as funções de seta, têm um
protótipo
propriedade
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 o
protótipo
Propriedade eram
discutido em
§6.2.3
e será coberto novamente em
Capítulo 9
.

8.7.4 Os métodos Call () e Apply ()

chamar()
e
aplicar()
permitir
você para invocar indiretamente (
§8.2.4
) a
Funcionar como se fosse um método de algum outro objeto.O primeiro argumento
para ambos
chamar()
e
aplicar()
é o objeto em que a função é
ser invocado;Este argumento é o contexto de invocação e se torna o
valor do
esse
palavra -chave dentro do corpo da função.Para invocar

Erro ao traduzir esta página.

Se uma função é definida para aceitar um número arbitrário de argumentos, o
aplicar()
o método permite que você invoca essa função no conteúdo
de uma variedade de comprimento arbitrário.Em ES6 e mais tarde, podemos apenas usar o
espalhar o operador, mas você pode ver
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
aplicar()
método
para passar os elementos da matriz para o
Math.max ()
função:
deixar

o maior

=

Matemática
.
máx
.
aplicar
(
Matemática
, Assim,

Arrayofnumbers
);
O
traço ()
a função definida a seguir é semelhante ao
cronometrado ()
função definida em
§8.3.4
, mas funciona para métodos em vez disso
de funções.Ele usa o
aplicar()
método 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 como o método do wrapper:
// Substitua o método chamado m do objeto O com uma versão

isso registra
// mensagens antes e depois de invocar o método original.
função

traço
(
o
, Assim,

m
)

8.7.5 O método bind ()
O
propósito primário de
vincular()
é para
vincular
uma função para um objeto.
Quando você invoca o
vincular()
Método em uma função
f
e passar um
objeto
o
, o método retorna uma nova função.Invocando a nova função
(em função) chama 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
)

{

retornar

esse
.
x

+

y
;

}

// essa função precisa

para ser amarrado
deixar

o

=

{

x
:

1

Este recurso de aplicação parcial de
vincular()
trabalha com seta
funções.Aplicação parcial é uma técnica comum em funcional
programação e às vezes é chamada
Currying
.Aqui estão alguns
Exemplos do
vincular()
Método usado para aplicação parcial:
deixar

soma

=

(
x
, Assim,
y
)

=>

x

+

y
;

// retorna a soma de 2 args
deixar

suc

=

soma
.
vincular
(
nulo
, Assim,

1
);

// liga o primeiro argumento a

1
suc
(
2
)

// => 3: x está ligado a 1 e passamos 2 para o y

argumento

que pode ser usado para criar novas funções:
const

f

=

novo

Função
(
"X"
, Assim,

"Y"
, Assim,

"Retornar 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
, Assim,

y
)

{

retornar

x
*
y
;

};
O
Função()
Construtor 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
Isso não leva argumentos, você simplesmente passaria uma única corda
- O corpo da função - para o construtor.
Observe que o
Função()
Construtor não passou por nenhum argumento
Isso especifica um nome para a função que ele cria.Como literais da função,

funções aninhadas e expressões de função que aparecem dentro
Os loops não são recompilados toda vez que são encontrados.
Um último ponto muito importante sobre o
Função()
construtor é que as funções que cria não usam lexical
escopo;Em vez disso, eles são sempre compilados como se estivessem no topo
Funções de nível, como o código a seguir demonstra:

```
deixar

escopo

=

"global"
;
função

Função de construção
()

{

deixar

escopo

=

"local"
;

retornar

novo

Função
(
"Escopo de retorno"
);

// não captura o escopo local!
}
// Esta linha retorna "global" porque

a função retornada pelo
// function () construtor não usa

o escopo local.
Função de construção
() ()

// => "Global"
```

O
Função()
o construtor é melhor pensar como um escopo globalmente
versão de
avaliar ()
(ver
§4.12.2

Todo
A 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 do
esse
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.Ver
Capítulo 9
Para 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
nulo
ou
indefinido
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
, Assim,
Então toda a expressão de invocação avalia para
indefinido
e não
A exceção é lançada.
Variedade
Objetos têm um
organizar()
método 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
organizar()
Isso exige um argumento de função opcional, você normalmente
use um
se
declaração para verificar se o argumento da função foi definido
antes de invocá -lo no corpo do
se
:
função

```
// Primeiro, defina duas funções simples
const

soma

=

(
x
, Assim,
y
)

=>

x
+
y
;
const

quadrado

=

x

=>

x
*
x
;
// então use essas funções com métodos de matriz para calcular

Média e Stddev
deixar

dados

=

[[
1
, Assim,
1
, Assim,
3
, Assim,
5
, Assim,
5
];
deixar

significar

=
```

UM
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

deixar

resultado

=

f
.
aplicar
(
esse
, Assim,

args
);

// que chama f

retornar

!
resultado
;

// e nega seu

retornar

um

=>

mapa
(
um
, Assim,

f
);
}
const

incremento

=

x

=>

x
+
1
;
const

incremental

=

mapeador
(
incremento
);
incremental
([
1
, Assim,
2
, Assim,
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

de argumentos.Dizemos que isso liga a função a um objeto e
aplica parcialmente os argumentos.O
vincular()
O método se aplica parcialmente
argumentos à esquerda - ou seja, os argumentos que você passa
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
, Assim,

...
externos
)

{

retornar

função
(...
Innerargs
)

{

// retorna esta função

deixar

args

=

[...
externos
, Assim,

...
Innerargs
];

// construa o

Lista de argumentos

retornar

f
.
aplicar
(
```

A outra função de conversão de objetos é chamada
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, então o padrão
valueof ()
o método simplesmente retorna o próprio objeto, em vez de retornar um
primitivo.Classes de wrapper, como string, número e booleano, definem
valueof ()
Métodos 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
valueof ()
método
que retorna a data em sua representação interna: o número de
milissegundos desde 1º de janeiro de 1970:
deixar

d

=

novo

Data
(
2010
, Assim,

0
, Assim,

1
);

// 1 de janeiro de 2010, (Pacífico

tempo)
d
.
valorof
()

// => 1262332800000
Algoritmos de conversão de objeto a princípios
Com
o
ToString ()
e
valueof ()
Métodos explicados, nós podemos
agora explique aproximadamente como os três objeto a princípio
Algoritmos funcionam (os detalhes completos são adiados até
§14.4.7

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
, Assim,
y
)

=>

x
*
y
;
const

neg

=

parcial
(
produto
, Assim,

-
1
);
const

sqrt

=

parcial
(
Matemática
.
prisioneiro de guerra
, Assim,

indefinido
, Assim,

.
5
);
const

recíproca

função,
Memoize ()
, isso 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

MECHOIZE
(
f
)

{

const

cache

=

novo

Mapa
();

// cache de valor armazenado no

encerramento.

retornar

função
(...
args
)

{

// Crie uma versão de string dos argumentos para usar como

uma chave de cache.

deixar

chave

=

args
.
comprimento

+

args
.

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
ÚnicoInteger ()
função 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
ÚnicoInteger ()
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:

```
deixar

Único

=

(
função
()

{

// Defina e invocar

deixar

contador

=

0
;

// estado privado de

função abaixo

retornar

função
()

{

retornar

contador
++
;

};
} ());
Único
()

// => 0
Único
()
```

Parâmetros de função opcionais, para reunir vários argumentos
em uma matriz usando um parâmetro de repouso e para destruir o objeto
e argumentos de matriz nos parâmetros de função.
Você pode usar o
...
espalhar o operador para passar os elementos de
uma matriz ou outro objeto iterável como argumentos em uma função
invocação.
Uma função definida dentro e devolvida por um anexo
A função mantém o acesso ao seu escopo lexical e pode, portanto, pode
Leia e escreva as variáveis ■■definidas dentro da função externa.
As funções usadas dessa maneira são chamadas
fechamentos
, e isso é um
técnica que vale a pena entender.
Funções são objetos que podem ser manipulados por JavaScript,
E isso permite um estilo funcional de programação.
1
O termo foi cunhado por Martin Fowler.Ver
http://martinfowler.com/dslcatalog/methodchening.html
.
2
Se você está familiarizado com Python, observe que isso é diferente de Python, no qual todos
A invocação compartilha o mesmo valor padrão.
3
Isso pode não parecer um ponto particularmente interessante, a menos que você esteja familiarizado c
om mais
idiomas estáticos, nos quais as funções fazem parte de um programa, mas não podem ser manipuladas po
r
o programa.

Capítulo 9.
Classes

JavaScript
Objetos foram cobertos em
Capítulo 6
.Aquele capítulo tratou cada
Objeto como um conjunto único de propriedades, diferente de todos os outros objetos.Isto
geralmente é útil, no entanto, para definir um
aula
de objetos que compartilham certos
propriedades.Membros, ou
instâncias
, da classe tem 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
coberto em
§4.6
, Assim,
§6.2.2
, e
§8.2.3
, e este capítulo tem mais em
§9.2
.
O JavaScript sempre permitiu a definição de classes.ES6 introduzido
uma sintaxe totalmente nova (incluindo um
aula
palavra -chave) que faz isso mesmo
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
JavaScript
classes 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
Object.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, 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: é
define um objeto de protótipo para uma classe que representa uma variedade de valores
e também
define a
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.
função

faixa
(
de
, Assim,

para
)

{

// 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.

deixar

r

=

Objeto
.
criar
(
faixa
.
Métodos
);

// Armazene os pontos de partida e final (estado) desta nova linha

objeto.

*
[[
Símbolo
.
iterador
] ()

{

para
(
deixar

x

=

Matemática
.
CEIL
(
esse
.
de
);

x

<=

esse
.
para
;

x
++
)

colheita

x
;

},

// retorna uma representação de string do intervalo

ToString
()

{

retornar

"("

+

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
geradores são cobertos em detalhes em
Capítulo 12
.Por enquanto, o
Upshot é que as instâncias dessa classe de intervalo podem ser usadas com
o
para/de
loop e com o
...
Operador espalhado.
Os métodos compartilhados e herdados definidos em
range.methods
Todos usam o
de
e
para
propriedades que foram inicializadas no
faixa()
função de fábrica.Para se referir a eles, eles usam
o
esse
palavra -chave para se referir ao objeto através do qual eles
foram invocados.Este uso de
esse
é um fundamental
característica dos métodos de qualquer classe.
9.2 Classes e Construtores
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 de construtor 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 construtoras é que o
protótipo
propriedade do
O construtor é usado como o protótipo do novo objeto.
§6.2.3
apresentou protótipos e enfatizou que, embora quase todos os objetos
ter um protótipo, apenas alguns objetos têm um
protótipo
propriedade.
Finalmente, podemos esclarecer isso: são objetos de função que têm um
protótipo
propriedade.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 alcance 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 o ES6
aula
palavra -chave.Embora
aula
está 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 o
aula
palavra -chave.
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.
função

Faixa
(
de
, Assim,

para
)

{

// Armazene os pontos de partida e final (estado) desta nova linha

objeto.

// estas são propriedades não herdadas que são exclusivas para

este objeto.

esse
.
de

=

de
;

esse

Este recurso de aplicação parcial de
vincular()
trabalha com seta
funções.Aplicação parcial é uma técnica comum em funcional
programação e às vezes é chamada
Currying
.Aqui estão alguns
Exemplos do
vincular()
Método usado para aplicação parcial:
deixar

soma

=

(
x
, Assim,
y
)

=>

x

+

y
;

// retorna a soma de 2 args
deixar

suc

=

soma
.
vincular
(
nulo
, Assim,

1
);

// liga o primeiro argumento a

1
suc
(
2
)

// => 3: x está ligado a 1 e passamos 2 para o y

argumento

valor.O
Faixa()
construtor apenas tem que inicializar
esse
.
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 o
novo
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
novo
.
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 o
novo
palavra -chave.Quando discutirmos subclasses em
§9.5
, Assim,
Vamos ver isso
new.target
nem sempre é uma referência ao construtor em que é usado: também pode se referir
à função construtora de uma subclasse.
Se
new.target
é
indefinido
, então a função contendo foi invocada como uma função, sem o
novo
palavra -chave.Os vários construtores de erro de JavaScript podem ser invocados sem
novo
, e se você quiser
Emular esse recurso em seus próprios construtores, você pode escrevê -los assim:
função

C
()

{

se

(
!
novo
.
alvo
)

arbitrário.No segundo exemplo, o protótipo é
Range.prototype
, e esse nome é obrigatório.Uma invocação de
o
Faixa()
o construtor usa automaticamente
Range.prototype
como
o protótipo do novo objeto de intervalo.
Finalmente, observe também as coisas que não mudam entre exemplos
9-1
e
9-2
: Os métodos de intervalo são definidos e invocados da mesma maneira
Para ambas as classes.Porque
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 o
função
palavra -chave.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.Lembrar de
§8.1.3
essas funções definidas dessa maneira não têm um
protótipo
propriedade
e assim não pode ser usado como construtores.Além disso, as funções de seta herdam o
esse
palavra -chave do contexto em que são definidos e não
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
esse
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
aula
palavra -chave 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
protótipo
propriedades 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
Faixa()
construtor cria
Objetos de alcance.Mais fundamentalmente, no entanto, os construtores são usados ■■como
o operando direto do
Instância de
operador ao testar
objetos para associação a uma classe.Se tivermos um objeto
r
e quero
Saiba se é um objeto de intervalo, podemos escrever:
r

Instância de

Faixa

// => true: r herda

Range.prototype
O
Instância de
O operador foi descrito 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 para
verdadeiro
se
o
herda de
C.Prototipo
.A herança não precisa ser direta: se
o
herda de
um objeto que herda de um objeto que herda
C.Prototipo
, a expressão ainda será avaliada para
verdadeiro
.

Tecnicamente falando, no exemplo de código anterior, o
Instância de
O operador não está verificando se
r
era realmente
inicializado pelo
Faixa
construtor.Em vez disso, está verificando se
r
herda de
Range.prototype
.Se definirmos uma função
Estranho()
e defina seu protótipo como o mesmo que
Range.prototype
, então objetos criados com
novo estranho ()
contará como objetos de alcance até onde
Instância de
está preocupado (eles
na verdade não funcionará como objetos de alcance, no entanto, porque seus
de
e
para
As propriedades não foram inicializadas):
função

Estranho
()

{}
Estranho
.
protótipo

=

Faixa
.
protótipo
;
novo

Estranho
()

Instância de

Faixa

// => true
Embora
Instância de
não pode realmente verificar o uso de um
Construtor, ele ainda usa uma função construtora como seu lado direto
Porque os construtores são a identidade pública de uma classe.
Se você deseja testar a cadeia de protótipos de um objeto para um específico
protótipo e não quero usar a função do construtor como um
intermediário, você pode usar o

Em

Exemplo 9-2
, nós 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 um
protótipo
propriedade.Portanto, todo JavaScript regular
função
automaticamente tem um
protótipo
propriedade.O valor disso
A propriedade é um objeto que tem um único, não entusiasmado
construtor
propriedade.O valor do
construtor
A propriedade é a função
objeto:
deixar

F

=

função
()

{};

// Este é um objeto de função.
deixar

p

=

F
.
protótipo
;

// Este é o objeto de protótipo

associado a F.
deixar

c

=

p
.
construtor

construtor e as instâncias criadas com o construtor.
Figura 9-1.
Uma função construtora, seu protótipo e instâncias
Observe isso
Figura 9-1
usa o nosso
Faixa()
construtor como exemplo.
De fato, no entanto, a classe de alcance definida em
Exemplo 9-2
substitui
o predefinido
Range.prototype
objeto com um objeto próprio.
E o novo objeto de protótipo que ele define não tem um
construtor
propriedade.Então, instâncias da classe Range, conforme definido,
não tem um
construtor
propriedade.
Podemos remediar este problema
adicionando explicitamente um construtor ao protótipo:
Faixa
.
protótipo

=

{

construtor
:

Faixa
, Assim,

// Defina explicitamente o construtor

referência de volta

/ * Definições do método vêm aqui */
};
Outra técnica comum que você provavelmente verá em mais velho
O código JavaScript é usar o objeto de protótipo predefinido com seu
construtor
propriedade e adicione métodos a ele um de cada vez com código

assim:
// estende o range predefinido.prototipo objeto, então não

substituir
// O Range.prototype automaticamente criado

propriedade.
Faixa
.
protótipo
.
inclui

=

função
(
x
)

{

retornar

esse
.
de

<=

x

&&

x

<=

esse
.
para
;
};
Faixa
.
protótipo
.
ToString

=

função
()

{

retornar

"("

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.)
A palavra -chave
construtor
é usado para definir o construtor
função para a classe.A função definida não é realmente
nomeado "construtor", no entanto.O
aula
declaração
A declaração define uma nova variável
Faixa
e atribui o valor
deste especial
construtor
função nessa variável.
Se sua classe não precisar fazer nenhuma inicialização, você pode
omita o
construtor
palavra -chave e seu corpo, e um vazio
A função construtora será criada implicitamente para você.
Se você deseja definir uma classe que subclasse - ou
herda de
-
Outra classe, você pode usar o
estende -se
palavra -chave com o
aula
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
aula

Span

estende -se

Faixa

{

construtor
(
começar
, Assim,

comprimento
)

{

se

(

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
O operador cria e inicializa um novo objeto.O
novo
A palavra -chave deve ser seguida por uma invocação de função.UM
função usada em
Assim é chamado de
construtor
e serve para inicializar um recém -criado
objeto.O JavaScript inclui construtores para seus tipos internos.Para
exemplo:
deixar

o

=

novo

Objeto
();

// Crie um objeto vazio: o mesmo que {}.
deixar

um

=

novo

Variedade
();

// Crie uma matriz vazia: o mesmo que [].
deixar

d

=

novo

Ao contrário das declarações de função, as declarações de classe não são
"Içado."Lembrar 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
aula
corpo prefixando o
declaração de método com o
estático
palavra -chave.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 a
Exemplo 9-3
:

```
estático

analisar
(
s
)

{

deixar

partidas

=

s
.
corresponder
(
/^\(\d+)\.\.\.(\d+)\)$/
);

se

(
!
partidas
)

{

lançar

novo
```

r
.
analisar
(
'(1 ... 10)'
);

// TypeError: R.Parse não é

uma função
Você vai
às vezes veja 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 o
regular
métodos de instância
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 o
esse
palavra -chave em um método estático.
Veremos exemplos de métodos estáticos em
Exemplo 9-4
.
9.3.2 Getters, Setters e outros formulários de método
Dentro de
um
aula
corpo, você pode definir métodos getter e setter
(
§6.10.6
) assim como você pode em literais de objetos.A única diferença é que em
Corpos de classe, você não coloca vírgula após o getter ou setter.
Exemplo 9-4
Inclui um exemplo prático de um método getter em uma classe.
Em geral, todo o
Sintaxe de definição de método abreviada permitida 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:
*
[[
Símbolo
.
iterador
] ()

{

para
(
deixar

9.3.3 Campos públicos, privados e estáticos
Em
a discussão aqui de classes definidas com o
aula
Palavra -chave, 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
Cromo
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:
aula

Buffer

{

construtor
()

{

esse
.
tamanho

=

0
;

esse
.
capacidade

=

4096
;

esse
.
buffer

Com a nova sintaxe do campo de instância que provavelmente será padronizada, você
em vez disso, poderia escrever:
aula

Buffer

{

tamanho

=

0
;

capacidade

=

4096
;

buffer

=

novo

Uint8array
(
esse
.
capacidade
);
}
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
esse.
Prefixos que apareceram no lado esquerdo das tarefas desapareceram,
Mas observe que você ainda deve usar
esse.
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.

A mesma proposta que procura padronizar esses campos de instância também
Define campos de instância privada.Se você usar o campo da instância
Sintaxe de inicialização mostrada no exemplo anterior para definir um campo
cujo nome começa com
#
(que normalmente não é um caráter legal em
Identificadores de JavaScript), esse campo será utilizável (com o
#
prefixo)
dentro do corpo da classe, mas será invisível e inacessível (e
portanto imutável) a qualquer código fora do corpo da classe.Se, para o
aula de buffer hipotético anterior, você queria garantir que os usuários de
a classe não poderia modificar inadvertidamente o
tamanho
campo de uma instância,
você poderia usar um particular
#tamanho
em vez disso, defina um getter
função para fornecer acesso somente leitura ao valor:
aula

Buffer

{

#
tamanho

=

0
;

pegar

tamanho
()

{

retornar

esse
.
#
tamanho
;

}
}
Observe que os campos privados devem ser declarados usando esta nova sintaxe de campo
antes que eles possam ser usados.Você não pode simplesmente escrever
isto.#tamanho = 0;
em
o construtor de uma classe, a menos que você inclua uma "declaração" do campo
diretamente no corpo da classe.
Finalmente, uma proposta relacionada procura padronizar o uso do
estático
palavra -chave para campos.Se você adicionar

esse:
estático

IntegerRangePattern

=

/^\(\d+)\.\.\.(\d+)\)$/
;
estático

analisar
(
s
)

{

deixar

partidas

=

s
.
corresponder
(
Faixa
.
IntegerRangePattern
);

se

(
!
partidas
)

{

lançar

novo

TypeError
(
`Não é possível analisar o alcance de

"
$ {
s
}
". '
)

}

número complexo

// aqui, com código como este:

//

// #r = 0;

// #i = 0;

// Esta função do construtor define os campos de instância r

e eu em cada

// instância que cria.Esses campos mantêm o real e

partes imaginárias de

// O número complexo: eles são o estado do objeto.

construtor
(
real
, Assim,

imaginário
)

{

esse
.
r

=

real
;

// este campo mantém a parte real

do número.

esse
.
eu

=

imaginário
;

// Este campo mantém o imaginário

papel.

}

// Aqui estão dois métodos de instância para adição e

```
pegar

imaginário
()

{

retornar

esse
.
eu
;

}

pegar

magnitude
()

{

retornar

Matemática
.
Hypot
(
esse
.
r
, Assim,

esse
.
eu
);

}

// as aulas quase sempre devem ter um método ToString ()

ToString
()

{

retornar

`{
$ {
esse
.
r
}
, Assim,
$ {
esse
```

9.4 Adicionando métodos às classes existentes
JavaScript
O mecanismo de herança baseado em protótipo é dinâmico: um
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 classe complexa de
Exemplo 9-4
:
// retorna um número complexo que é o conjugado complexo de

Este.
Complexo
.
protótipo
.
conj

=

função
()

{

retornar

novo

Complexo
(
esse
.
r
, Assim,

-
esse
.
eu
);

};
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 ...
se

(
!
Corda
.
protótipo

```
// 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
.
protótipo
.
vezes

=

função
(
f
, Assim,

contexto
)

{

deixar

n

=

esse
.
valorof
();

para
(
deixar

eu

=

0
;

eu

<

n
;

eu
++
)

f
.
```

de
eliminação.
Em
modo rigoroso,
excluir
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
).Modo rigoroso também especifica que
excluir
levanta a
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
falso
para indicar que o operando poderia
não ser excluído.
Aqui estão alguns exemplos de usos do
excluir
operador:
deixar

o

=

{
x
:

1
, Assim,

y
:

2
};
excluir

o
.
x
;

// excluir uma das propriedades do objeto;retorna

verdadeiro.
typeof

o
.
x
;

```
// Verifique se o protótipo de extensão herda do intervalo

protótipo
Span
.
protótipo

=

Objeto
.
criar
(
Faixa
.
protótipo
);
// não queremos herdar range.prototype.constructor, então nós
// Defina nossa própria propriedade construtora.
Span
.
protótipo
.
construtor

=

Span
;
// Ao definir seu próprio método tostring (), o span substitui o
// ToString () Método que ele herdaria de

Faixa.
Span
.
protótipo
.
ToString

=

função
()

{

retornar

`(
$ {
esse
.
de
}
... +
$ {
esse
.
para
```

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 o
super
palavra -chave AS
parte do
aula
sintaxe.A próxima seção demonstra como funciona.
9.5.2 subclasses com extensões e super
Em
ES6 e mais tarde, você pode criar uma superclasse simplesmente adicionando um
estende -se
cláusula para 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.
aula

Ezarray

estende -se

Variedade

{

pegar

primeiro
()

{

retornar

esse
[[
0
];

}

pegar

durar
()

{

retornar

esse
[[
esse
.
comprimento

Erro ao traduzir esta página.

O

>>
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
a 3, por exemplo, mas observe isso e
-7 >> 1
Avalia como -4.
Mudar à direita com preenchimento zero
(
>>>
)
O

>>>
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
Esse
A 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
, Assim,
enquanto
, e
para

```
// retorna qualquer que o método da superclasse retorne.

retornar

super
.
definir
(
chave
, Assim,

valor
);

}
}
```

Os dois primeiros argumentos para o
TypeDMap ()
construtor é o
Tipos de chave e valor desejados.Estas devem ser strings, como "número"
e "booleano", que o
typeof
Retorna do operador.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 o
super
palavra -chave como se fosse um nome de função.
O
Mapa()
Construtor requer um argumento opcional: um objeto iterável
de
[Chave, valor]
matrizes.Então o terceiro argumento opcional do
TypeDMap ()
construtor 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
TypeDMap ()
o construtor a seguir inicializa seu próprio estado de subclasse por
contexto
this.KeyType
e
this.valuetype
para o especificado
tipos.Ele precisa definir essas propriedades para que possa usá -las novamente em
o
definir()
método.
Existem algumas regras importantes que você precisará saber sobre o uso
super()
Nos construtores:

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 o
esse
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 indefinido em
funções que são invocadas sem o
novo
palavra -chave.Em
funções construtoras, no entanto,
new.target
é uma referência
ao construtor que foi invocado.Quando uma subclasse
construtor é invocado e usa
super()
para invocar o
construtor de superclasse, esse construtor de superclasse verá o
construtor de 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 registro, por exemplo.
Depois do construtor, a próxima parte de
Exemplo 9-6
é um método chamado
definir()
.A superclass do mapa define um método chamado
definir()
para adicionar um
nova entrada para o mapa.Nós dizemos que isso
definir()
Método no TypeDMap
substituir
o
definir()
Método 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.Esse
definir()
o método não tem
qualquer maneira de acrescentar a chave e valor ao próprio mapa, mas é isso que
Superclass
definir()
método é para.Então nós usamos o

Para invocar a versão do método da superclasse.Nesse contexto,
super
funciona como o
esse
palavra -chave faz: refere -se ao
objeto atual, mas permite o acesso a métodos substituídos definidos no
Superclass.
Nos construtores, você deve invocar o construtor de superclasse
Antes de poder acessar
esse
e inicialize o novo objeto você mesmo.
Não existem regras dessas quando você substitui um método.Um método que
substitui um método de superclasse não é necessário para invocar a superclasse
método.Se usar
super
para invocar o método substituído (ou qualquer
método) na superclasse, pode fazer isso no início ou no meio
ou o final do método de substituição.
Finalmente, antes de deixarmos o exemplo do TypeDMap para trás, vale a pena
Observando que esta classe é um candidato ideal para o uso de campos privados.
Como a classe está escrita agora, um usuário pode alterar o
keytype
ou
ValueType
propriedades para subverter a verificação do tipo.Uma vez privado
Os campos são suportados, poderíamos mudar essas propriedades para
#KeyType
e
#ValueType
para que eles não pudessem ser alterados de fora.
9.5.3 Delegação em vez de herança
O

estende -se
A palavra -chave facilita a criação de subclasses.Mas isso
não significa que você
deve
Crie muitas subclasses.Se você quiser
Escreva uma classe que compartilha o comportamento de outra classe, você pode tentar
herdar esse comportamento criando uma subclasse.Mas muitas vezes é mais fácil e
mais flexível para colocar o comportamento desejado em sua classe tendo
Sua classe cria uma instância da outra classe e simplesmente delegando para
essa instância, conforme necessário.Você cria uma nova classe não por subclassificação, mas

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
JavaScript
Set Class, exceto isso 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 adicionando um
contar()
método.Por outro
mão, uma vez que começamos a pensar em como podemos implementar isso
contar()
método, 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.
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.
*/
aula

Histograma

{
2

```
pegar

imaginário
()

{

retornar

esse
.
eu
;

}

pegar

magnitude
()

{

retornar

Matemática
.
Hypot
(
esse
.
r
, Assim,

esse
.
eu
);

}

// as aulas quase sempre devem ter um método ToString ()

ToString
()

{

retornar

`{
$ {
esse
.
r
}
, Assim,
$ {
esse
```

Tudo
Histograma ()
construtor faz
Exemplo 9-7
é criar a
Objeto de mapa.E a maioria dos métodos são frases que apenas delegam
a um método do mapa, tornando a implementação bastante simples.
Porque usamos a delegação e não a herança, um objeto de histograma
não é uma instância de conjunto ou mapa.Mas o histograma implementa um número
de métodos de conjunto comumente usados, e em um idioma não vindado como
JavaScript, isso geralmente é bom o suficiente: uma relação formal de herança
Às vezes é bom, mas geralmente opcional.
9.5.4 Hierarquias de classe e classes abstratas
Exemplo 9-6
demonstrado
Como podemos subclasse mapa.
Exemplo 9-7
demonstraram como podemos delegar a um objeto de mapa sem
Na verdade, subclassificando qualquer coisa.Usando classes JavaScript para encapsular
dados e modularizar seu código geralmente é uma ótima técnica, e você pode
encontre -se usando o
aula
palavra -chave com frequência.Mas você pode encontrar
que você prefere composição à herança e que raramente precisa
usar
estende -se
(exceto quando você está usando uma biblioteca ou estrutura que
Requer que você estenda suas classes base).
Existem algumas circunstâncias em que vários níveis de subclasse são
apropriado, no entanto, e terminaremos este capítulo com um prolongado
Exemplo que demonstra uma hierarquia de classes representando diferente
tipos de conjuntos.(As classes definidas definidas em
Exemplo 9-8
são semelhantes a,
Mas não é completamente compatível com a classe de conjunto interno do JavaScript.)
Exemplo 9-8
define muitas subclasses, mas também demonstra como
você pode definir
Classes abstratas
—Classes que não incluem um completo
implementação - para servir como uma superclasse comum para um grupo de

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.
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
Classe final em
Exemplo 9-8
muita manipulação de bits com o
&
, Assim,
|
, Assim,
e
~
operadores, que você pode revisar em
§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().
*/
aula

AbstractSet

{

// joga um erro aqui para que as subclasses sejam forçadas

// Para definir sua própria versão de trabalho deste método.

tem
(
x
)

{

lançar

novo

Erro
(
"Método abstrato"
);

}
}
/**
* O Notset é uma subclasse concreta do abstractset.

qual
para/in
enumera as propriedades de um objeto.
5,5 saltos
Outro
categoria de declarações de javascript são
Jump declarações
.Como o
O nome indica, eles fazem com que o intérprete JavaScript salte para um novo
Localização no código -fonte.O
quebrar
declaração 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
rotulado
, e
quebrar
e
continuar
pode
Identifique o loop de destino ou outro rótulo de instrução.
O
retornar
declaração faz o intérprete saltar de uma função
invocação de volta ao código que o invocou e também fornece o valor
para a invocação.O
lançar
A declaração é um tipo de retorno interino
de uma função de gerador.O
lançar
A declaração aumenta, ou
joga
, um
exceção e foi projetado para trabalhar com o
tente/capturar/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

```
aula

AbstractEnumerabableset

estende -se

AbstractSet

{

pegar

tamanho
()

{

lançar

novo

Erro
(
"Método abstrato"
);

}

[[
Símbolo
.
iterador
] ()

{

lançar

novo

Erro
(
"Método abstrato"
);

}

isEmpty
()

{

retornar

esse
.
tamanho

===
```

Erro ao traduzir esta página.

retornar

um

=>

mapa
(
um
, Assim,

f
);
}
const

incremento

=

x

=>

x
+
1
;
const

incremental

=

mapeador
(
incremento
);
incremental
([
1
, Assim,
2
, Assim,
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

e bit

deixar

pedaço

=

x

%

8
;

se

(
!
esse
.
_tem
(
byte
, Assim,

pedaço
))

{

// se esse bit for

ainda não está definido

esse
.
dados
[[
byte
]

| =

Bitset
.
bits
[[
pedaço
];

// então

defina

esse
.
n
++

Este recurso de aplicação parcial de
vincular()
trabalha com seta
funções.Aplicação parcial é uma técnica comum em funcional
programação e às vezes é chamada
Currying
.Aqui estão alguns
Exemplos do
vincular()
Método usado para aplicação parcial:
deixar

soma

=

(
x
, Assim,
y
)

=>

x

+

y
;

// retorna a soma de 2 args
deixar

suc

=

soma
.
vincular
(
nulo
, Assim,

1
);

// liga o primeiro argumento a

1
suc
(
2
)

// => 3: x está ligado a 1 e passamos 2 para o y

argumento

declaração.
As subclasses podem invocar o construtor de sua superclasse ou
Métodos substituídos de sua superclasse com o
super
palavra -chave.
1
Exceto as funções retornadas pelo ES5

Function.bind ()
método.Funções vinculadas
não têm propriedade protótipo, mas eles usam o protótipo do subjacente
função se forem invocados como construtores.
2
Ver
Padrões de design
(Addison-Wesley Professional) de Erich Gamma et al.ou
Eficaz
Java
(Addison-Wesley Professional) de Joshua Bloch, por exemplo.

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 outro módulos.

Até Recentemente, JavaScript não tinha suporte interno 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 um requer () função, que foi adotado pelo nó. requer () - -módulos baseados são um parte fundamental do ambiente de programação do nó, mas foram Nunca adotado como parte oficial da linguagem JavaScript.Em vez de, ES6 define módulos usando importar e exportar palavras -chave.Embora importar e exportar 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 nós usando
requer ()
Módulos ES6 usando
exportar
, Assim,
importar
, e
importar()
10.1 módulos com classes, objetos e
Fechamentos
No entanto
Pode ser óbvio, vale ressaltar que um dos
características importantes das classes é que elas atuam como módulos para seus
Métodos.Pense em voltar
Exemplo 9-8
.Esse exemplo definiu um número
de diferentes classes, todas as quais tinham um método chamado
tem()
.Mas você
não teria nenhum problema em escrever um programa que usasse vários conjuntos
Classes desse exemplo: não há perigo que a implementação
de
tem()
de singletonset substituirá o
tem()
método 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 em
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.Considerar
Exemplo 9-8
de novo.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, a
_válido()
e
_tem()
Os métodos são internos
Utilitários que não devem realmente ser expostos aos usuários da classe.E
Bitset.bits
e
Bitset.Masks
são detalhes da 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

=

(
função
()

{

// defina bitset para o retorno

valor desta função

// Detalhes de implementação privada aqui

função

isValid
(
definir
, Assim,

const

Máscaras

=

novo

Uint8array
([
~
1
, Assim,

~
2
, Assim,

~
4
, Assim,

~
8
, Assim,

~
16
, Assim,

~
32
, Assim,

~
64
, Assim,

~
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

retornar

aula

Bitset

```
objeto

retornar

{

significar
, Assim,

stddev

};
} ());
// e aqui está como podemos usar o módulo
estatísticas
.
significar
([
1
, Assim,

3
, Assim,

5
, Assim,

7
, Assim,

9
]))

// => 5
estatísticas
.
stddev
([
1
, Assim,

3
, Assim,

5
, Assim,

7
, Assim,

9
]))

// => Math.sqrt (10)
```

10.1.1 Automatando a modularidade baseada em fechamento

Observação
que é um processo bastante mecânico transformar um arquivo de
JavaScript Code nesse tipo de módulo inserindo algum texto no
Início e final do arquivo.Tudo o que é necessário é alguma convenção para

```
const

soma

=

(
x
, Assim,

y
)

=>

x

+

y
;

const

quadrado

=

x

=

>

x

*

x
;

exportações
.
significar

=

função
(
dados
)

{

...

};

exportações
```

JavaScript sobre uma conexão de rede relativamente lenta, não há necessidade
ou se beneficie em agrupar um programa de nós em um único arquivo JavaScript.
No nó, cada arquivo é um módulo independente com um espaço de nome privado.
Constantes, variáveis, funções e classes definidas em um arquivo são
privado para esse arquivo, a menos que o arquivo os exportte.E valores exportados por
Um módulo é apenas visível em outro módulo se esse módulo explicitamente
importa -os.
Módulos de nó importam outros módulos com o
requer ()
função
e exportar sua API pública definindo propriedades do objeto de exportação
ou substituindo o
Module.Exports
objeto inteiramente.
10.2.1 Exportações de nós
Nó
define um global
exportações
objeto que é sempre definido.Se você
estão escrevendo um módulo de nó que exporta vários valores, você pode simplesmente
Atribua -os às propriedades deste objeto:
const

soma

=

(
x
, Assim,

y
)

=>

x

+

y
;
const

quadrado

=

x

=>

x

*

x
;
exportações

Erro ao traduzir esta página.

que você instalou em seu sistema através de um gerenciador de pacotes, então
Você simplesmente usa o nome não qualificado do módulo, sem qualquer "/"
Personagens que o transformariam em um caminho do sistema de arquivos:

```
// Esses módulos são incorporados para o nó
const

fs

=

exigir
(
"FS"
);

// o embutido

Módulo de sistema de arquivos
const

http

=

exigir
(
"http"
);

// o http integrado

módulo
// A estrutura Express HTTP Server é um módulo de terceiros.
// não faz parte do nó, mas foi instalado localmente
const

expressar

=

exigir
(
"expressar"
);
```

Quando você deseja importar um módulo de seu próprio código, o módulo
o nome deve ser o caminho para o arquivo que contém esse código, em relação a
o arquivo do módulo atual.É legal usar caminhos absolutos que começam
com um
/
caráter, mas normalmente, ao importar módulos que fazem parte
do seu próprio programa, os nomes dos módulos começarão com
./
ou
às vezes
../
para indicar que eles são relativos ao diretório atual ou
o diretório pai.Por exemplo:

```
const
```

função,
Memoize ()
, isso 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

MECHOIZE
(
f
)

{

const

cache

=

novo

Mapa
();

// cache de valor armazenado no

encerramento.

retornar

função
(...
args
)

{

// Crie uma versão de string dos argumentos para usar como

uma chave de cache.

deixar

chave

=

args
.
comprimento

+

args
.

ES6
adiciona
importar
e
exportar
palavras -chave para 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 o
importar
e
exportar
declarações,
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 ES6
aula
definição) é automaticamente no modo rigoroso
(ver
§5.6.3
).Isso significa que, quando você começa a usar módulos ES6,
Você nunca terá que escrever
"Use rigoroso"
de novo.E isso significa que
O código nos módulos não pode usar o
com
declaração ou o
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,
esse
é
indefinido
.Em módulos,
esse
é
indefinido
Mesmo no topo
código de nível.(Por outro lado, scripts em navegadores da web e conjunto de nós
esse
para
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 navegadore
s da web
Além do Internet Explorer.Quando usados ■■nativamente, os módulos ES6 são adicionados em
Páginas HTML com um especial
<script type = "módulo">
tag, 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
Para
exportar uma constante, variável, função ou classe de um módulo ES6,
Basta adicionar a palavra -chave
exportar
Antes da declaração:

```
exportar

const

Pi

=

Matemática
.
Pi
;
exportar

função

DeGreestoradians
(
d
)

{

retornar

d

*

Pi

/

180
;

}
exportar
```

Erro ao traduzir esta página.

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.
Finalmente, observe que o
exportar
A palavra -chave 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
valores de importação 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'
;
Este é o
importar
palavra -chave, seguida por um identificador, seguido por
o
de
palavra -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 o
const
palavra -chave.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.) Um
Especificador do módulo
string 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
exportação padrão
.Para importar valores de um

Módulo que exporta vários valores, usamos uma sintaxe ligeiramente diferente:
importar

{

significar
, Assim,

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.Um
importar
declaração que faz referência a que o módulo pode
importar qualquer subconjunto desses valores simplesmente listando seus nomes dentro
aparelho encaracolado.Os aparelhos encaracolados fazem esse tipo de
importar
declaração
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
um
importar
declaração como esta:
importar

*

como

estatísticas

de

"./stats.js"
;
Um
importar
declaração como essa cria um objeto e o atribui a um
constante nomeado
estatísticas
.Cada uma das exportações não padrão do módulo
Ser importado se torna uma propriedade disso
estatísticas
objeto.Não-defasa

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.)
A palavra -chave
construtor
é usado para definir o construtor
função para a classe.A função definida não é realmente
nomeado "construtor", no entanto.O
aula
declaração
A declaração define uma nova variável
Faixa
e atribui o valor
deste especial
construtor
função nessa variável.
Se sua classe não precisar fazer nenhuma inicialização, você pode
omita o
construtor
palavra -chave e seu corpo, e um vazio
A função construtora será criada implicitamente para você.
Se você deseja definir uma classe que subclasse - ou
herda de
-
Outra classe, você pode usar o
estende -se
palavra -chave com o
aula
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
aula

Span

estende -se

Faixa

{

construtor
(
começar
, Assim,

comprimento
)

{

se

(

para
importar
Para que ele seja executado como parte do nosso programa.
Observe que você pode usar este nada de importação
importar
sintaxe mesmo com
módulos que têm exportações.Se um módulo define comportamento útil
independente dos valores que exporta e se o seu programa não precisar
Qualquer um desses valores exportados, você ainda pode importar o módulo.apenas para
esse comportamento padrão.
10.3.3 Importações e exportações com renomeação
Se
Dois módulos exportam dois valores diferentes usando o mesmo nome e
Você quer importar esses dois valores, você terá que renomear um
ou ambos os valores quando você o importa.Da mesma forma, se você quiser
importar um valor cujo nome já está em uso no seu módulo, você irá
precisa renomear o valor importado.Você pode usar o
como
palavra -chave com
Nomeado importações para renomeá -las enquanto você as importa:
importar

{

renderizar

como

renderImage

}

de

"./imageutils.js"
;
importar

{

renderizar

como

renderui

}

de

"./ui.js"
;
Essas linhas importam duas funções para o módulo atual.O
funções são nomeadas
renderizar ()
Nos módulos que os definem
mas são importados com os nomes mais descritivos e desambiguadores
renderImage ()
e

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
, Assim,

significar
, Assim,

stddev

}

de

"./HISTOMSTATS.JS"
;
Nesse caso, a palavra -chave JavaScript
padrão
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 do
exportar
declaração.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 o
como
palavra -chave para fazer isso:
exportar

{

layout

como

CalculateLayout
, Assim,

renderizar

como

renderlayout
};
Lembre -se de que, embora os aparelhos encaracolados pareçam algo como objeto

10.3.4 Reexports
Por todo
Este capítulo, discutimos um hipotético "./stats.js"
módulo que exporta
significar()
e
stddev ()
funções.Se fôssemos
Escrevendo um módulo assim e pensamos que muitos usuários do módulo
gostaria de apenas uma função ou outra, então podemos querer
definir
significar()
em um módulo “./stats/mean.js” e defina
stddev ()
em
"./Stats/stddev.js".Dessa forma, os programas só precisam importar exatamente o
funções de que precisam e não são inchadas por importar código que eles não
precisar.
Mesmo se tivéssemos definido essas funções estatísticas em individual
módulos, no entanto, podemos esperar que haveria muitos
programas que desejam as duas funções e apreciariam um conveniente
"./Stats.js", do qual eles poderiam importar ambos em uma linha.
Dado que as implementações estão agora em arquivos separados, definindo isso
“./Stat.js” o módulo é simples:
importar

{

significar

}

de

"./stats/mean.js"
;
importar

{

stddev

}

de

"./stats/stddev.js"
;
exportar

{

significar
, Assim,

stdev

};
Os módulos ES6 antecipam esse caso de uso e fornecem uma sintaxe especial para

Observe que os nomes
significar
e
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:
exportar

*

de

"./stats/mean.js"
;
exportar

*

de

"./stats/stddev.js"
;
Reexportar sintaxe permite renomear com
como
tão regular
importar
e
exportar
declarações fazem.Suponha que queríamos reexportar o
significar()
função, mas também defina
média()
como outro nome para a função.
Poderíamos fazer isso assim:
exportar

{

significar
, Assim,

significar

como

média

}

de

"./stats/mean.js"
;
exportar

{

esse:
// importar a função média () de ./stats.js e torná -lo o
// Exportação padrão deste módulo
exportar

{

significar

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

{

padrão

}

de

"./stats/mean.js"
10.3.5 Módulos JavaScript na Web
O
Seções anteriores descreveram os módulos ES6 e seus
importar
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
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, código
Bundling
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 a otimizar seu módulo ES6
implementações.
Mesmo que as ferramentas de agrupamento ainda possam ser desejáveis nnna produção, elas
não são mais necessários no desenvolvimento, pois todos os navegadores atuais

Forneça suporte nativo para módulos JavaScript.Lembrar
que os módulos usam
modo rigoroso por padrão,
esse
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 nativamente
importar
diretivas em um navegador da web, você deve
Diga ao navegador da web que seu código é um módulo usando um
<script
type = "Módulo">
marcação.
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
importar
A declaração pode ser tratada como um URL relativo.UM
<script
type = "Módulo">
tag marca o ponto de partida de um modular
programa.Nenhum dos módulos que ele importa deve estar em
<Cript>
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 a
<script type = "módulo">
tag para definir o
O ponto de entrada principal para um programa modular JavaScript pode ser tão simples quanto
esse:

```
<script
tipo =
"módulo"
>
importar

"./main.js"
;
</script>
```

Codificar dentro de um embutido
<script type = "módulo">
tag é um ES6
módulo e, como tal, pode usar o
exportar
declaração.Não há nenhum
apontar para fazer isso, no entanto, porque o html
<Cript>
Sintaxe de tag

Isto
vale a pena notar aqui que você pode combinar esta técnica de fechamento
com getters e setters de propriedades.A seguinte versão do
contador()
a função é uma variação no código que apareceu em
§6.10.6
, Assim,
Mas ele usa fechamentos para estado privado, em vez de confiar regularmente
Propriedade do objeto:
função

```
contador
(
n
)

{

// O argumento da função n é o privado

variável

retornar

{

// Método Getter de propriedade retorna e incrementos

Contador privado var.

pegar

contar
()

{

retornar

n
++
;

},

// O Setter Property não permite o valor de n para

diminuir

definir

contar
(
m
)

{

se
```

type = "Módulo">
, você sabe que ele só será carregado por navegadores
isso pode apoiá -lo.E
como um recuo para o IE11 (que, em 2020, é
efetivamente o único navegador restante que não suporta ES6), você
pode usar ferramentas como Babel e Webpack para transformar seu código em não
código ES5 modular e, em seguida, carregue esse código transformado menos eficiente via
<Nomodule <script>
.
Outra diferença importante entre scripts regulares e módulo
Os scripts têm a ver com o carregamento de origem cruzada.Um regular
<Cript>
marcação
Carregará um arquivo de código JavaScript de qualquer servidor na Internet e
A infraestrutura da Internet de publicidade, análise e código de rastreamento
depende desse fato.Mas
<script type = "módulo">
fornece um
oportunidade de apertar isso, e os módulos só podem ser carregados do
mesma origem que o documento HTML contendo ou quando os cors adequados
Os cabeçalhos estão em vigor para permitir cargas de origem cruzada.Um
O infeliz efeito colateral dessa nova restrição de segurança é que ela faz
difícil de testar módulos ES6 no modo de desenvolvimento usando
arquivo:
URLs.Ao usar módulos ES6, você provavelmente precisará configurar uma estática
servidor da web para teste.
Alguns programadores gostam de usar a extensão do nome do arquivo
.mjs
para
distinguir seus arquivos JavaScript modulares de seus regulares, não
arquivos javascript modulares com o tradicional
.js
extensão.Para o
fins de navegadores da web e
<Cript>
tags, a extensão do arquivo é
Na verdade, irrelevante.(O tipo MIME é relevante, no entanto, se você usar
.mjs
Arquivos, pode ser necessário configurar seu servidor da web para servi -los
com o mesmo tipo MIME que
.js
arquivos.) O suporte do Node para ES6 faz
Use a extensão do nome do arquivo como uma dica para distinguir qual módulo

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
o
.mjs
Convenção de Nomeação.
10.3.6 Importações dinâmicas com importação ()
Nós temos
vi que o ES6
importar
e
exportar
Diretivas 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 um novo
<Cript>
Marque no HTML atual
Documento e aplicativos da Web fazem isso há muitos anos.

Erro ao traduzir esta página.

JavaScript sobre uma conexão de rede relativamente lenta, não há necessidade
ou se beneficie em agrupar um programa de nós em um único arquivo JavaScript.
No nó, cada arquivo é um módulo independente com um espaço de nome privado.
Constantes, variáveis, funções e classes definidas em um arquivo são
privado para esse arquivo, a menos que o arquivo os exportte.E valores exportados por
Um módulo é apenas visível em outro módulo se esse módulo explicitamente
importa -os.
Módulos de nó importam outros módulos com o
requer ()
função
e exportar sua API pública definindo propriedades do objeto de exportação
ou substituindo o
Module.Exports
objeto inteiramente.
10.2.1 Exportações de nós
Nó
define um global
exportações
objeto que é sempre definido.Se você
estão escrevendo um módulo de nó que exporta vários valores, você pode simplesmente
Atribua -os às propriedades deste objeto:
const

soma

=

(
x
, Assim,

y
)

=>

x

+

y
;
const

quadrado

=

x

=>

x

*

x
;
exportações

carregado com
requer ()
), a sintaxe especial
Import.Meta
refere -se a
um objeto que contém metadados sobre o módulo atualmente executando.
O
url
propriedade deste objeto é o URL do qual o módulo
foi carregado.(No nó, este será um
arquivo://
Url.)
O caso de uso primário de
Import.Meta.url
é 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
Url ()
construtor 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
L10N/
diretório, que está no
mesmo diretório do próprio módulo.Seu módulo pode carregar suas cordas
Usando um URL criado com uma função, como esta:
função

LocalStringsurl
(
localidade
)

{

retornar

novo

Url
(
`L10N/
$ {
localidade
}
.json`
, Assim,

importar
.
Meta
.
url
);
}
10.4 Resumo
O objetivo da modularidade é permitir que os programadores ocultem o

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 definindo o
Module.Exports
propriedade.
No ES6, o JavaScript finalmente conseguiu seu próprio sistema de módulos com
importar
e
exportar
palavras -chave e ES2020 está adicionando
suporte para importações dinâmicas com
importar()
.
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
Datatipos, 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.
1

Erro ao traduzir esta página.

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
O 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.
Criar
um objeto definido com o
Definir()
construtor:
deixar

s

=

novo

Definir
();

// um novo conjunto vazio
deixar

t

=

novo

Definir
([
1
, Assim,

s
]);

// Um ■■novo conjunto com dois membros
O argumento para o
Definir()
construtor não precisa ser uma matriz: nenhum
Objeto iterável (incluindo outros objetos set) é permitido:
deixar

t

=

novo

Definir
(

Os conjuntos não precisam ser inicializados quando você os cria.Você pode adicionar e remova os elementos a qualquer momento com
adicionar()
, Assim,
excluir()
, 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:
deixar

s

=

novo

Definir
();

// Comece vazio
s
.
tamanho

// => 0
s
.
adicionar
(
1
);

// Adicione um número
s
.
tamanho

// => 1;agora o conjunto tem um membro
s
.
adicionar
(
1
);

// Adicione o mesmo número novamente
s
.
tamanho

// => 1;O tamanho não muda
s
.
adicionar
(
verdadeiro
);

Erro ao traduzir esta página.

número
OneDigitPries
.
tem
(
3
)

// => true: assim é 3
OneDigitPries
.
tem
(
4
)

// => false: 4 não é um primo
OneDigitPries
.
tem
(
"5"
)

// => false: "5" não é nem 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
tem()
O método 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
para/de
laço
Para enumerar todos os elementos de um conjunto:
deixar

soma

=

0
;
para
(
deixar

p

de

OneDigitPries
)

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 a forma especial
exportação padrão
:
exportar

const

Tau

=

2

*

Matemática
.
Pi
;
exportar

função

magnitude
(
x
, Assim,
y
)

{

retornar

Matemática
.
sqrt
(
x
*
x

+

y
*
y
);

}
exportar

padrão

aula

deixar

m

=

novo

Mapa
();

// Crie um novo mapa vazio
deixar

n

=

novo

Mapa
([

// Um ■■novo mapa é inicializado com teclas de string

mapeado para números

[[
"um"
, Assim,

1
],

[[
"dois"
, Assim,

2
]
]);
O argumento opcional para o
Mapa()
construtor 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
Mapa()
construtor para copiar
outros mapas ou para copiar os nomes e valores de propriedades de um existente
objeto:
deixar

cópia

=

Expressões relacionais sempre avaliam para
verdadeiro
ou
falso
, então quando
usado assim, o
&&
O próprio operador retorna
verdadeiro
ou
falso
.
Os operadores relacionais têm maior precedência do 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".(Ver
§3.4
Para detalhes.
Os valores falsamente são
falso
, Assim,
nulo
, Assim,
indefinido
, Assim,
0
, Assim,
-0
, Assim,
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
faz
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
que precisamos descrever
&&
no terceiro e último nível.Este operador

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 o
esse
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 indefinido em
funções que são invocadas sem o
novo
palavra -chave.Em
funções construtoras, no entanto,
new.target
é uma referência
ao construtor que foi invocado.Quando uma subclasse
construtor é invocado e usa
super()
para invocar o
construtor de superclasse, esse construtor de superclasse verá o
construtor de 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 registro, por exemplo.
Depois do construtor, a próxima parte de
Exemplo 9-6
é um método chamado
definir()
.A superclass do mapa define um método chamado
definir()
para adicionar um
nova entrada para o mapa.Nós dizemos que isso
definir()
Método no TypeDMap
substituir
o
definir()
Método 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.Esse
definir()
o método não tem
qualquer maneira de acrescentar a chave e valor ao próprio mapa, mas é isso que
Superclass
definir()
método é para.Então nós usamos o

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 o
chaves ()
e
valores ()
Métodos: estes retornam iterável
Objetos que iteram as chaves e valores, em ordem de inserção.(O
entradas ()
O método retorna um objeto iterável que itera a chave/valor
pares, mas isso é exatamente o mesmo que iterando o mapa diretamente.)
[...
m
.
chaves
()]

// => ["x", "y"]: apenas as chaves
[...
m
.
valores
()]

// => [1, 2]: apenas os valores
[...
m
.
entradas
()]

// => [["x", 1], ["y", 2]]: o mesmo que [... m]
Os objetos de mapa também podem ser iterados usando o
foreach ()
método que
foi implementado pela primeira vez pela classe da matriz.
m
.
foreach
((
valor
, Assim,

chave
)

=>

{

// NOTA VALOR, CHAVE NÃO CHAVE,

valor

// Na primeira invocação, o valor será 1 e a chave será

seja "x"

chave em segundo lugar.
11.1.3 Frawmap e fraco
O
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 "fraco"
Referências
para seus valores -chave para que não sejam
alcançável através do mapa fraco, e sua presença no mapa faz
não impedir que sua memória seja recuperada.
O
FrawMap ()
construtor é exatamente como o
Mapa()
construtor, 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
pegar()
, Assim,
definir()
, Assim,
tem()
, e
excluir()
Métodos.Em particular, o mapa fraco não é iterável
e não define
chaves ()
, Assim,
valores ()
, ou
foreach ()
.Se
Frafmap era iterável, então suas chaves seriam acessíveis e
Não seria fraco.
Da mesma forma, o Frafmap não implementa o
tamanho
propriedade
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.Ver
§6.10.3
.)
Frawset implementa um conjunto de objetos que não impedem aqueles
Objetos de lixo coletados.O
Frawset ()
construtor
funciona como o
Definir()
construtor, 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.
Frawset implementa apenas o
adicionar()
, Assim,
tem()
, e
excluir()
métodos e não é iterável.
Frawset não tem um
tamanho
propriedade.
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

JavaScript regular
Matrizes podem ter elementos de qualquer tipo e podem 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 novos em
ES6,
estão muito mais próximos das matrizes de baixo nível desses idiomas.
Matrizes digitadas não são tecnicamente matrizes (
Array.esArray ()
retorna
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

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

Construtor	Tipo numérico
3	

Erro ao traduzir esta página.

Os operadores de incremento e decréscimos são semelhantes, pois realizam um atribuição implícita.O
excluir
O operador também tem efeitos colaterais:
Excluir uma propriedade é como (mas não o mesmo que) atribuir
indefinido
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
O
operadores listados em
Tabela 4-1
estão dispostos em ordem da 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:

```
c
=
x
+
y
*
z
;
```

O operador de multiplicação
*
tem uma precedência maior que a adição
operador
+
, então 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

Erro ao traduzir esta página.

Número de bytes de memória que você deseja alocar:
deixar

buffer

=

novo

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
Int32Array ()
construtor 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:
deixar

Asbytes

=

novo

Uint8array
(
buffer
);

// Visto como

bytes
deixar

Essas quatro matrizes digitadas oferecem quatro vistas diferentes na memória
Representado pelo ArrayBuffer.É importante entender que todos
Matrizes digitadas têm um arranjo subjacente, mesmo que você não
especificar explicitamente um.Se você ligar para um construtor de matriz digitado sem
Passando um objeto buffer, um buffer do tamanho apropriado será
criado automaticamente.Como descrito mais adiante, o
buffer
propriedade de qualquer
A matriz digitada refere -se ao seu objeto ArrayBuffer subjacente.A razão para
Trabalhe diretamente com objetos ArrayBuffer é que às vezes você pode
Deseja ter várias visualizações de matriz digitadas de um único buffer.
11.2.3 Usando matrizes digitadas
Uma vez
você criou uma matriz digitada, você pode ler e escrever seu
elementos com notação regular de suporte quadrado, exatamente como você faria com
Qualquer outro objeto semelhante a uma matriz:

```
// retorna o maior primo menor que n, usando a peneira

de Eratóstenos
função

peneira
(
n
)

{

deixar

um

=

novo

Uint8array
(
n
+
1
);

// a [x] será 1 se

x é composto

deixar

máx

=

Matemática
.
chão
(
Matemática
.
```

}
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
Variedade()
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:
deixar

ints

=

novo

Int16Array
(
10
);

// 10 números inteiros curtos
ints
.
preencher
(
3
).
mapa
(
x
=>
x
*
x
).
juntar
(
""
)

// => "9999999999"
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
empurrar()
, Assim,
pop ()
, Assim,
NIFT ()
, Assim,
mudança()
, e
emenda ()

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
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 pareça
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.
Enquanto você pode obter resultados semelhantes usando o
se
declaração (
§5.3.1
),
o
?:
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

O fato de que o
Subarray ()
Método retorna uma nova visão de um
A Array existente nos traz de volta ao tópico dos ArrayBuffers.Cada digitado
A Array possui três propriedades relacionadas ao buffer subjacente:
last3
.
buffer

// o objeto ArrayBuffer para um

matriz digitada
last3
.
buffer

===

ints
.
buffer

// => true: ambos são vistas de

o mesmo buffer
last3
.
byteoffset

// => 14: Esta vista começa em

byte 14 do buffer
last3
.
BytELength

// => 6: Esta exibição é de 6 bytes (3

INTS de 16 bits) longo
last3
.
buffer
.
BytELength

// => 20: mas o subjacente

Buffer tem 20 bytes
O
buffer
A propriedade é a matriz da matriz.
byteoffset
é a posição inicial dos dados da matriz no buffer subjacente.
E
BytELength
é a duração dos dados da matriz em bytes.Para qualquer
matriz digitada,
um
, esse invariante deve sempre ser verdadeiro:

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 depois 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:
deixar

bytes

=

novo

Uint8array
(
1024
);

// 1024 bytes
deixar

ints

=

novo

UINT32Array
(
bytes
.
buffer
);

// ou 256

Inteiros
deixar

flutua

[[
0
]

===

1
;
Hoje, as arquiteturas de CPU mais comuns são
Little-Endian.Muitos
protocolos de rede e alguns formatos de arquivo binário exigem
Big-Endian
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
deixar

visualizar

=

novo

DataView
(
bytes
.
buffer
, Assim,

bytes
.
byteoffset
, Assim,

bytes
.
BytELength
);
deixar

int

=

visualizar

getfloat64 ()
.O primeiro argumento é o deslocamento do byte dentro do
ArrayBuffer no qual o valor começa.
Todos esses métodos getter,
diferente de
getint8 ()
e
getUint8 ()
, aceite um opcional
valor booleano como seu segundo argumento.Se o segundo argumento for
omitido ou é
falso
, pedidos de byte big-endian são usados.Se o segundo
argumento é
verdadeiro
, pedidos pouco endianos são usados.
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 for
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
UM

expressão regular
é
um objeto que descreve um padrão textual.O
JavaScript
A classe regexp 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

Manter esse invariante, as matrizes têm dois comportamentos especiais.O primeiro nós descrito acima: se você atribuir um valor a um elemento de matriz cujo índice
eu
é maior ou igual ao atual da matriz
comprimento
, o valor de
o
comprimento
A propriedade está definida como
I+1
.
O segundo comportamento especial que as matrizes implementam para manter O comprimento invariante é que, se você definir o
comprimento
propriedade para um não
Inteiro negativo
n
menor que seu valor atual, qualquer elementos de matriz cujo índice é maior ou igual a
n
são excluídos da matriz:
um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];

// Comece com uma matriz de 5 elementos.
um
.
comprimento

=

3
;

// a agora é [1,2,3].
um
.
comprimento

=

0
;

// Exclua todos os elementos.a é [].
um

foi definido com o
Regexp ()
construtor, como este:
deixar

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
Regexp ()
construtor.Se quiséssemos combinar strings
isso terminou com "s" ou "s", por exemplo, poderíamos usar o
eu
bandeira com o nosso
Expressão regular para indicar que queremos correspondência insensível ao caso:
deixar

padrão

=

/s $/i
;
As seções a seguir descrevem os vários caracteres e meta-
caracteres usados ■■em expressões regulares JavaScript.
Personagens literais
Todos
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.
Tabela 11-1
lista esses personagens.
Tabela 11-1.
Personagens literais de expressão regular

Chara cter Partidas	Alfão Umérico
Charact er Em si	\ 0
O personagem NUL (\ u0000)	\ t
Guia (\ u0009)	\ n
Newline (\ u000a)	\ v
Guia vertical (\ u000b)	\ f
FORME FORME (\ u000c)	\ r
Retorno de carruagem (\ u000d)	\ x nn
O caractere latino especificado pelo número hexadecimal nn ;por exemplo, \ x0a é o mesmo que \ n .	\ u xxxx
O caractere unicode especificado pelo número hexadecimal xxxx	

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 tem 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 barriga:
/\\//
.(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
Individual
caracteres literais 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 um dos personagens 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
/[um-
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 seja
não
Unicode
espaço em branco.
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 caráter 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 o
s
Flag, então um período corresponde a qualquer personagem, 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 personagem de espaço em branco e
\ d
corresponde a qualquer
Digit, então
/[\ s \ d]/
corresponde a qualquer caractere ou dígito em branco.
Observe que há um caso especial.Como você verá mais tarde, o
\ b
escapar
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
Em
ES2018, se uma expressão regular usa o
u
bandeira, então classes de personagens
\ 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 Unic
ode,
e o conjunto de caracteres que eles representam pode mudar à medida que o Unicode evolui.
O
\ d
A classe de caracteres corresponde apenas aos dígitos ASCII.Se você quiser combinar um dígito decima
l 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 é
não
Um dígito decimal em qualquer idioma, você pode capitalizar o
p
e escreva
\ P {decimal_number}
.Se você deseja combinar com qualquer caráter semelhante ao número, incluindo frações e
algarismos 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 ca
racteres
definido pelo padrão Unicode.
O
\c
A classe de caracteres 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 adi
cionar
As categorias também “Connector_Puntuation” e “junção_control”.)
Como exemplo final, o
\ p
A sintaxe também nos permite definir expressões regulares que correspondem aos personagens
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

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.
Tabela 11-3
resume a sintaxe de repetição.
Tabela 11-3.
Personagens regulares de repetição de expressão
Char
ACTER
Significado
{
n
, Assim,
m
}
Combine o item anterior pelo menos
n
vezes mas não mais do 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:
deixar

r

=

/\ d {2,4}/

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
/um*/
Na verdade, corresponde à string "bbbb" porque a string contém
Zero ocorrências da letra A!
Repetição sem graça
O
Personagens de repetição listados em
Tabela 11-3
combinar 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
marca:
?
, Assim,
+?
, Assim,
*?
, 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.

esse:
estático

IntegerRangePattern

=

/^\(\d+)\.\.\.(\d+)\)$/
;
estático

analisar
(
s
)

{

deixar

partidas

=

s
.
corresponder
(
Faixa
.
IntegerRangePattern
);

se

(
!
partidas
)

{

lançar

novo

TypeError
(
`Não é possível analisar o alcance de

"
$ {
s
}
". '
)

}

não
consulte o padrão para essa subexpressão, mas sim para o 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
Combina qualquer que seja 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
)
, comece o grupo com
(?:
e fim
com
)
.Considere o seguinte padrão:

/([Jj] ava (?: [Ss] crip)?) \ Sis \ s (diversão \ w*)/
Neste exemplo, a subexpressão
(?: [Ss] critério)
é usado 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 combinado por
(Fun \ W*)
.
Tabela 11-4
resume a alternância de expressão regular, 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: itens de grupo em uma única unidade que pode ser usada com
*
, Assim,
+
, Assim,
?
, Assim,
|
, e assim

sobre.Lembre -se também dos personagens que correspondem a esse grupo para uso com mais tarde

Referências.
(
?
:
.
.
.
)

\
n
Combine os mesmos personagens que foram correspondidos quando o número do grupo
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
ES2018
padroniza um novo recurso que pode tornar as expressões regulares mais auto-documentação 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 Fire
fox.
Para citar um grupo, use
(? <...>
em vez de
(
e coloque o nome entre os colchetes do ângulo.Para
exemplo, aqui está uma expressão regular que pode ser usada para verificar a formatação da linha fin
al 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.Em
§11.3.2
, quando discutimos a string
substituir()
e
corresponder()
Métodos e o
Regexp
exec ()
Método, 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 f
azer 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 fechada
s
teve que combinar.Poderíamos reescrever este regexp usando um grupo de captura nomeado e um nome nom
eado
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

entre caracteres em vez de caracteres reais.
\ b
, por exemplo,
corresponde a um limite da palavra ascii - o limite entre um
\c
(Personagem da palavra ascii) e um
\C
(personagem não -palavra), ou o limite
entre um personagem de palavra ascii e o início ou o fim de um
corda.
Elementos como
\ b
Não especifique 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 eles ancoram o padrão a um
posição específica na sequência de pesquisa.A âncora mais usada
elementos são
^
, que vincula o padrão ao início da corda,
e
$
, que ancora o padrão no final da corda.
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 tentar 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 personagens de espaço reais com
\ s
, combine (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/
Combina “JavaScript”
e "PostScript", mas não "script" ou
"Scripts".
Você também pode usar expressões regulares arbitrárias como condições de âncora.Se
4

Tolocalestring ()
que tentam formatar números, datas e
vezes de acordo com as convenções locais.Array define um
Tolocalestring ()
método que funciona como
ToString ()
exceto
que formata os elementos da matriz chamando seus
Tolocalestring ()
métodos em vez de seus
ToString ()
Métodos.Você pode fazer o
A mesma coisa com um
apontar
objeto como este:
deixar

apontar

=

{

x
:

1000
, Assim,

y
:

2000
, Assim,

ToString
:

função
()

{

retornar

`(
$ {
esse
.
x
}
, Assim,
$ {
esse
.
y
}
) `
;

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 exige
permissão.
Agradecemos, mas geralmente não exigem, atribuição.Uma atribuição
Geralmente inclui o título, autor, editor e ISBN.Por exemplo:
“
JavaScript: O Guia Definitivo
, Sétima edição, por 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,
O'Reilly Media
forneceu tecnologia e negócios
Treinamento, conhecimento e insight para ajudar as empresas a ter sucesso.

Bandeiras suportadas e seus significados são:
g
O
g
A bandeira 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 da corda
corresponder()
método e o regexp
exec ()
método de maneiras importantes.
eu
O
eu
sinalizador especifica que a correspondência de padrões deve ser de caso
insensível.
m
O
m
Flag especifica que a correspondência deve ser feita em "Multiline"
modo.Diz que o regexp será usado com cordas multilinas
e que o
^
e
$
As âncoras devem corresponder ao começo e
final da corda e também o começo e o fim das linhas individuais
dentro da string.
s
Como o
m
bandeira, o
s
A bandeira 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 o
s
bandeira é
usado, no entanto, "".vai corresponder a qualquer personagem, incluindo linha
Terminadores.O

s
Flag foi adicionado ao JavaScript no ES2018 e, como
do início de 2020, é apoiado em nó, cromo, borda e safari, mas
Não Firefox.
u
O
u
A bandeira significa Unicode, e faz a expressão regular

Combine os pontos de codepates unicode completos em vez de corresponder valores de 16 bits.
Esta bandeira foi introduzida no ES6, e você deve criar o hábito de
Usando -o em todas as expressões regulares, a menos que você tenha algum motivo, não
para.Se você não usar esta bandeira, seus regexps não funcionarão
bem com texto que inclui emoji e outros personagens (incluindo
muitos caracteres chineses) que exigem mais de 16 bits.Sem
o
u
Bandeira, o "".O caractere corresponde a qualquer valor de 1 UTF-16 de 16 bits.
Com a bandeira, no entanto, "".corresponde a um ponto de código unicode,
incluindo aqueles que têm mais de 16 bits.Definindo o
u
bandeira em um
Regexp também permite que você use o novo
\ u {...}
sequência de fuga
para caráter unicode e também permite o
\ p {...}
notação para
Classes de caracteres Unicode.
y
O
y
A bandeira indica que a expressão regular é "pegajosa" e
deve corresponder no início de uma corda ou no primeiro personagem
Após a partida anterior.Quando usado com uma expressão regular
que foi projetado para encontrar uma única correspondência, trata efetivamente que
expressão regular como se comece com
^
para ancorá -lo ao
começo da string.Esta bandeira é mais útil com regular
expressões que são usadas repetidamente para encontrar todas as correspondências dentro de um
corda.Nesse caso, causa comportamento especial da corda
corresponder()
método e o regexp
exec ()
Método para aplicar
que cada partida subsequente está ancorada na posição da string em
que o último terminou.
Esses sinalizadores podem ser especificados em qualquer combinação e em qualquer ordem.Para
por exemplo, se você deseja que sua expressão regular seja consciente de unicode para
Faça uma correspondência insensível ao caso e você pretende usá-lo para encontrar múltiplos
Corresponde dentro de uma string, você especificaria as bandeiras
uig
, Assim,
GUI
, ou qualquer
Outra permutação dessas três letras.

11.3.2 Métodos de string para correspondência de padrões

Até
Agora, estamos descrevendo 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
procurar()
.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"
.
procurar
(
/script/ui) // => 4
"Python"
.
procurar
(
/script/ui) // => -1
```

Se o argumento para
procurar()
não é uma expressão regular, é o primeiro
convertido a um passando para o
Regexp
construtor.
procurar()
não suporta pesquisas globais;Ignora o
g
bandeira de seu regular
argumento de expressão.

SUBSTITUIR()

O

substituir()
O método 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 bandeira, o
substituir()
O método substitui todas as correspondências na string
com a sequência de substituição;Caso contrário, ele substitui apenas a primeira partida
encontra.Se o primeiro argumento para
substituir()
é uma corda 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
procurar()
faz.Como exemplo, você pode usar
substituir()
o seguinte para fornecer capitalização uniforme da palavra
"JavaScript" em uma série de texto:
// Não importa como seja capitalizado, substitua -o pelo

capitalização correta
texto
.
substituir
(
/javascript/gi
, Assim,

"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 a
$
seguido de um dígito aparece em
a corda de substituição,
substituir()
substitui esses dois personagens
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.
deixar

citar

=

/"([^"]*) "/g
;
// Substitua as aspas retas por GuilleMets
// deixando o texto citado (armazenado em US $ 1) inalterado.
'Ele disse "pare"'
.

Se o seu regexp usar grupos de captura nomeados, você poderá se referir ao
Texto correspondente por nome e não por número:
deixar

citar

=

/"(?
;
'Ele disse "pare"'
.
substituir
(
citar
, Assim,

'«$ <TaTtext>»'
)

// => 'ele

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, toda a corda que
substituir()
foi chamado é passado.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:
deixar

s

=

"15 vezes 15 é 225"
;
s
.
substituir
(
/\ d+/gu, n => parseint (n) .ToString (16)) // => "F

vezes f é e1 "
CORRESPONDER()
O

corresponder()
O método é o mais geral da string regular

Erro ao traduzir esta página.

Neste caso não global, a matriz devolvida por
corresponder()
também tem alguns
Propriedades do objeto Além dos elementos de matriz numerados.O
entrada
propriedade refere -se à sequência em que
corresponder()
foi chamado.
O
índice
propriedade é a posição dentro daquela string na qual a
a partida começa.E se a expressão regular contiver captura nomeada
grupos, então a matriz devolvida também tem um
grupos
propriedade cujo
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:

```
deixar

url

=

/(?)
<TACH> \ S*)/
;
deixar

texto

=

"Visite meu blog em http://www.example.com/~david"
;
deixar

corresponder

=

texto
.
corresponder
(
url
);
corresponder
[[
0
]

// => "http://www.example.com/~david"
corresponder
.
entrada

// => texto
corresponder
```

Ao contrário das declarações de função, as declarações de classe não são
"Içado."Lembrar 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
aula
corpo prefixando o
declaração de método com o
estático
palavra -chave.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 a
Exemplo 9-3
:

```
estático

analisar
(
s
)

{

deixar

partidas

=

s
.
corresponder
(
/^\(\d+)\.\.\.(\d+)\)$/
);

se

(
!
partidas
)

{

lançar

novo
```

é implementado pelos modernos navegadores da web e nó.
Matchall ()
espera um regexp com o
g
Conjunto de bandeira.Em vez de devolver uma variedade de
substringas correspondentes como
corresponder()
no entanto, ele retorna um iterador
isso produz o tipo de correspondência de objetos que
corresponder()
retorna quando usado
com um regexp não global.Isso faz
Matchall ()
o mais fácil e
maneira mais geral de percorrer todas as partidas dentro de uma string.
Você pode usar
Matchall ()
para percorrer as palavras em uma série de
texto como este:
// um ou mais caracteres alfabéticos unicode entre a palavra

limites
const

palavras

=

/\ b \ p {alfabético}+\ b/gu;// \ p não é suportado

no Firefox ainda
const

texto

=

"Este é um teste ingênuo do Matchall ()

método."
;
para
(
deixar

palavra

de

texto
.
Matchall
(
palavras
))

{

console

com um argumento de string como este:
"123.456.789"
.
dividir
(
","
)

// => ["123", "456",

"789"]
O
dividir()
o método também pode assumir uma expressão regular como sua
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"
.
dividir
(
/\ s*, \ s*/
)

// => ["1", "2", "3", "4",

"5"]
Surpreendentemente, se você ligar
dividir()
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"
.
dividir
(
htmltag
)

// => ["teste", "br/",

"1,2,3"]
11.3.3 A classe Regexp
Esse
Seção documenta o
Regexp ()

Erro ao traduzir esta página.

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 para leitura que é verdadeira se o
g
sinalizador está definido.
ignorecase
Uma propriedade booleana somente para leitura que é verdadeira se o
eu
sinalizador está definido.
Multilina
Uma propriedade booleana somente para leitura que é verdadeira se o
m
sinalizador está definido.
Dotall
Uma propriedade booleana somente para leitura que é verdadeira se o
s
sinalizador está definido.
unicode
Uma propriedade booleana somente para leitura que é verdadeira se o
u
sinalizador está definido.
pegajoso
Uma propriedade booleana somente para leitura que é verdadeira se o
y
sinalizador está 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 pelo
exec ()
e
teste()
métodos,
descrito nas próximas duas subseções.

TESTE()
O

teste()
O método 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 corresponde ao padrão ou
falso
se não corresponder.
teste()
funciona simplesmente chamando o (muito mais complicado)
exec ()
método descrito na próxima seção e retornando
verdadeiro
se
exec ()
Retorna um valor não nulo.Por causa disso, se você usar
teste()
com um regexp que usa o
g
ou
y
bandeiras, então seu comportamento depende
o valor do
LastIndex
propriedade do objeto regexp, que
pode mudar inesperadamente.Ver
“A propriedade LastIndex e Regexp
Reutilizar ”
Para mais detalhes.
Exec ()
O
Regexp
exec ()
o método 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 retorna
nulo
.Se uma partida
é encontrado, no entanto, ele retorna uma matriz exatamente como a matriz devolvida pelo
corresponder()
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
índice
a propriedade contém a posição do personagem na qual a partida
ocorreu e o
entrada
Propriedade especifica a sequência que foi
pesquisado, e o
grupos
propriedade, se definida, refere -se a um objeto que
mantém as substringas que correspondem aos grupos de captura qualquer nome.

Ao contrário da string
corresponder()
método,
exec ()
retorna o mesmo tipo de
matriz se a expressão regular tem ou não o global
g
bandeira.
Lembre -se disso
corresponder()
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
exec ()
é
chamou uma expressão regular que possui o global
g
bandeira ou o
pegajoso
y
Conjunto de bandeira, ele consulta o
LastIndex
propriedade 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
exec ()
encontra com sucesso um
combinar, ele atualiza o
LastIndex
propriedade para o índice do
Personagem imediatamente após o texto correspondente.
Se
exec ()
não consegue encontrar um
Match, ele é redefinido
LastIndex
para 0. Este comportamento especial permite que você
chamar
exec ()
repetidamente para percorrer todo o regular
A expressão corresponde a uma string.(Embora, como descrevemos, em
ES2020 e mais tarde, o
Matchall ()
O método de 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:
deixar

padrão

=

Como
Você já viu, a API de expressão regular do JavaScript é complicada.O uso do
LastIndex
propriedade com o
g
e
y
Flags é uma parte particularmente estranha desta API.Quando você usa
Essas bandeiras, você precisa ser particularmente cuidadoso ao ligar para o
corresponder()
, Assim,
exec ()
, ou
teste()
métodos porque o comportamento desses métodos depende de
LastIndex
, e o valor de
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 todos
<p>
Tags dentro de uma sequência de texto HTML.
Podemos escrever código como este:
deixar

corresponder
, Assim,

posições

=

[];
enquanto
((
corresponder

=

/<p>/g
.
exec
(
html
))

! ==

nulo
)

{

// possível loop infinito

posições
.
empurrar

Módulos do faça você mesmo com classes, objetos e fechamentos
Módulos de nós usando
requer ()
Módulos ES6 usando
exportar
, Assim,
importar
, e
importar()
10.1 módulos com classes, objetos e
Fechamentos
No entanto
Pode ser óbvio, vale ressaltar que um dos
características importantes das classes é que elas atuam como módulos para seus
Métodos.Pense em voltar
Exemplo 9-8
.Esse exemplo definiu um número
de diferentes classes, todas as quais tinham um método chamado
tem()
.Mas você
não teria nenhum problema em escrever um programa que usasse vários conjuntos
Classes desse exemplo: não há perigo que a implementação
de
tem()
de singletonset substituirá o
tem()
método 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 em
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

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
Data()
construtor, os interpreta no UTC e retorna um
Milissegund Timestamp que você pode passar para o
Data()
construtor:

```
// meia -noite na Inglaterra, 1 de janeiro de 2100
deixar

século

=

novo

Data
(
Data
.
UTC
(
2100
, Assim,

0
, Assim,

1
));
```

Se você imprimir uma data (com
console.log (século)
, por exemplo), isso
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
ToisSoString ()
.
Finalmente, se você passar uma string para o
Data()
construtor, vai tentar
Para analisar essa string como especificação de data e hora.O construtor pode
datas de análise especificadas nos formatos produzidos pelo
ToString ()
, Assim,
toutcString ()
, e
ToisSoString ()

Métodos:

```
deixar

século

=
```

deixar

d

=

novo

Data
();

// Comece com o

data atual
d
.
SetlyEarear
(
d
.
Comborda
()

+

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 opcionalmente permite 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 ()
Retorne o dia da semana (0 para domingo
até 6 para sábado).O dia da semana é somente leitura, então não há um
correspondente
setday ()
método.
11.4.1 Timestamps
JavaScript
representa datas internamente como números inteiros que especificam o

d
.
settime
(
d
.
gettime
()

+

30000
);
Esses valores de milissegundos às vezes são chamados
TIMESTAMPS
, e é
às vezes útil para trabalhar com eles diretamente, e não com a data
objetos.A estática
Date.now ()
o método retorna a hora atual como um
Timestamp e é útil quando você deseja medir quanto tempo
O código leva para executar:
deixar

StartTime

=

Data
.
agora
();
Reticulatesplines
();

// Faça alguma operação demorada
deixar

final

=

Data
.
agora
();
console
.
registro
(
`A reticulação do spline tomou
$ {
final

-

StartTime
}
Sra. '

Objetos de data
pode ser comparado com o padrão do JavaScript
<
, Assim,
<=
, Assim,
>
, 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 h
ora
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
setDate ()
, Assim,
setmonth ()
, e
Setyear ()
.Aqui, por exemplo, é o código que adiciona três meses e
duas semanas até a data atual:
deixar

d

=

novo

Data
();
d
.
setmonth
(
d
.
getmonth
()

+

3
, Assim,

d
.
getDate
()

+

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:

```
deixar

d

=

novo

Data
(
2020
, Assim,

0
, Assim,

1
, Assim,

17
, Assim,

10
, Assim,

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' Local
d
.
tolocaletimestring
()
```

TodATestring ()
Esse
Método formato apenas a parte da data da data e omite
o tempo.Ele usa o fuso horário local e não faz localidade-
formatação apropriada.
tolocaledatestring ()
Esse
Método formato apenas a data.Ele usa o fuso horário local e um
Formato de data apropriado para localidade.
Totimestring ()
Esse
O método formata apenas o tempo e omite a data.Ele usa o
Fuso horário local, mas não formate o tempo de maneira consciente do local.
tolocaletimestring ()
Esse
o método formata o tempo de maneira consciente de localidade 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 uma estática
DATE.PARSE ()
Método 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
Data()
o construtor pode e é garantido
capaz de analisar a saída de
ToisSoString ()
, Assim,
toutcString ()
, Assim,
e
ToString ()
.

11.5 Classes de erro
O
JavaScript
lançar
e
pegar
declarações podem jogar e capturar qualquer
Valor JavaScript, incluindo valores primitivos.Não há tipo de exceção
Isso deve ser usado para sinalizar erros.JavaScript define um erro
classe, no entanto, e é tradicional usar instâncias de erro ou um
subclasse ao sinalizar um erro com
lançar
.Um bom motivo para usar
Um objeto de erro é que, quando você cria um erro, ele captura o estado de
A pilha JavaScript, e se a exceção não for capturada, o rastreamento da pilha
será exibido com a mensagem de erro, o que o ajudará a depurar
a questão.(Observe que o rastreamento da pilha mostra onde estava o objeto de erro
criado, não onde o
lançar
A declaração joga isso.Se você sempre criar
o objeto imediatamente antes de jogá -lo com
lançar novo erro ()
, esse
não causará nenhuma confusão.)
Os objetos de erro têm duas propriedades:
mensagem
e
nome
, e a
ToString ()
método.O valor do
mensagem
propriedade é o
valor que você passou para o
Erro()
construtor, convertido em uma corda se
necessário.Para objetos de erro criados com
Erro()
, o
nome
propriedade
é sempre "erro".O
ToString ()
o método simplesmente retorna o valor
do
nome
propriedade seguida por um cólon e espaço e o valor de
o
mensagem
propriedade.
Embora não faça parte do padrão ECMAScript, nó e tudo
navegadores modernos também definem um
pilha
propriedade em objetos de erro.O
O valor desta propriedade é uma sequência de várias linhas que contém um rastreamento de pilha de
a pilha de chamadas JavaScript no momento em que o objeto de erro foi
criado.Esta pode ser uma informação útil para registrar quando um inesperado

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
Avaliador
, 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
tem um
nome
propriedade 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
mensagem
propriedades.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 Parseerror com
linha
e
coluna
propriedades 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
status
Propriedade que contém o código de status HTTP (como 404 ou 500)
da solicitação fracassada.
Por exemplo:
aula

Httperror

estende -se

Erro

{

construtor
(
status
, Assim,

Statustext
, Assim,

url
)

{

super
(

```
esse
.
url

=

url
;

}

pegar

nome
()

{

retornar

"Httperror"
;

}
}
deixar

erro

=

novo

Httperror
(
404
, Assim,

"Não encontrado"
, Assim,

"http://example.com/"
);
erro
.
status

// => 404
erro
.
mensagem

// => "404 não encontrado:

http://example.com/ "
erro
.
nome
```

e é de uso comum, mesmo com programas não baseados em Javascript.
JavaScript
suporta a serialização e a deserialização do 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 ()
:
deixar

o

=

{
s
:

""
, Assim,

n
:

0
, Assim,

um
:

[[
verdadeiro
, Assim,

falso
, Assim,

nulo
]};
deixar

s

=

O

>>
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
a 3, por exemplo, mas observe isso e
-7 >> 1
Avalia como -4.
Mudar à direita com preenchimento zero
(
>>>
)
O

>>>
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
Esse
A 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
, Assim,
enquanto
, e
para

String de volta em uma estrutura de dados.
11.6.1 Customizações JSON
Se
Json.Stringify ()
é solicitado a serializar um valor que não é
Nativamente apoiado pelo formato JSON, parece ver se esse valor tem
um
Tojson ()
método e, se assim for, chama esse método e depois
Rigifica o valor de retorno no lugar do valor original.Objetos de data
implementar
Tojson ()
: isto
Retorna a mesma string que
ToisSoString ()
Método 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 o
esse
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
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:
deixar

dados

=

JSON
.
analisar
(
texto
, Assim,

função
(
chave
, Assim,

valor
)

{

// Remova todos os valores cujo nome da propriedade começa com um

sublinhado

se

(
chave
[[
0
]

===

"_"
)

retornar

indefinido
;

// Se o valor for uma string no formato ISO 8601 Data

converta -o em uma data.

Se você aprovar uma função, é uma função de substituição - efetivamente o inverso
da função de reviver opcional para você pode passar
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 retorna 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
deixar

texto

=

JSON
.
stringify
(
endereço
, Assim,

[[
"cidade"
, Assim,
"estado"
, Assim,
"país"
]);
// Especifique uma função de substituição que omite o valor regexp

propriedades
deixar

JSON

=

JSON
.
stringify
(
o
, Assim,

(
k
, Assim,

v

Ferramentas e idiomas compatíveis com JSON.

11.7 A API de internacionalização

O
A API de internacionalização JavaScript consiste nas três classes
Intl.numberFormat, intl.dateTimeFormat e intl.collator que permitem
nós para formatar números (incluindo quantidades e porcentagens monetárias),
datas e tempos de maneiras apropriadas para o local e comparar seqüências em
maneiras apropriadas para localidade.Essas classes não fazem parte do ecmascript
padrão, mas são definidos
Como parte do
ECMA402 padrão

e são
bem suportado por navegadores da web.A API INTL
também é apoiado em
Nó, mas no momento da redação deste artigo, binários de nós pré -construídos não enviam
com os dados de localização necessários para fazê -los trabalhar com locais
Além de nós, inglês.Então, para usar essas classes com nó, você
pode precisar baixar um pacote de dados separado ou usar uma construção personalizada de
Nó.
Um
das partes mais importantes da internacionalização estão exibindo
texto que foi traduzido para o idioma do usuário.Existem vários
maneiras de conseguir isso, mas nenhum deles está dentro do escopo do INTL
API descrita aqui.

11.7.1 Números de formatação

Usuários
em todo o mundo espera que os números sejam formatados em diferentes
caminhos.Pontos decimais podem ser períodos ou vírgulas.Milhares de separadores
pode ser vírgulas ou períodos, e eles não são usados ■■a cada três dígitos em todos
lugares.Algumas moedas são divididas em centésimos, outras em
milésimos, e alguns não têm subdivisões.Finalmente, embora o SO-

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.
O
Intl.NumberFormat Class define um
formatar()
método 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
Intl.numberFormat ()
construtor, se
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"
.
Especificar
"por cento"
Para formatar um número
como uma porcentagem ou especificar
"moeda"
para especificar um número como um
quantidade de dinheiro.
moeda

Se
estilo é
"moeda"
, então 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"
, então esta propriedade especifica como o
A moeda é exibida.O valor padrão
"símbolo"
usa a
Símbolo da moeda se a moeda tiver uma.O valor
"código"
usos
o código ISO de três letras e o valor
"nome"
soletra o
nome da moeda em forma longa.
useGrouping
Defina esta propriedade para
falso
Se você não deseja que os números tenham
milhares de separadores (ou seus equivalentes apropriados para localidade).
MinimintegerDigits
O número mínimo de dígitos a serem usados nnpara 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.
MinimumFractionDigits
, Assim,
MaximumFractionDigits
Esses
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.
MinimumsignificantDDigits
, Assim,
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 o seu
formatar()
Método, que retorna uma string adequadamente formatada.Por exemplo:
deixar

Euros

=

Intl
.
NumberFormat
(
"Es"
, Assim,

{
estilo
:

"moeda"
, Assim,

moeda
:

"EUR"
});
Euros
.
formatar
(
10
)

// => "10,00 €": dez euros, espanhol

formatação
deixar

libras

=

Intl
.
NumberFormat
(
"en"
, Assim,

{
estilo

Localidade de EN-US
Alguns idiomas, como o árabe, usam seu próprio script para decimal
dígitos:
deixar

árabe

=

Intl
.
NumberFormat
(
"AR"
, Assim,

{
useGrouping
:

falso
}).
formatar
;
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-
para o 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:
deixar

hindi

=

Intl
.
NumberFormat
(
"Hi-in-u-nu-deva"
).
formatar
;
hindi
(
1234567890
)

A instância intl.dateTimeFormat é chamando seu
formatar()
método para
converter um objeto de data em uma string.
Como
mencionado em
§11.4
, a classe de data define simples
tolocaledatestring ()
e
tolocaletimestring ()
Métodos que produzem saída apropriada para localidade para o local do usuário.
Mas esses métodos não fornecem nenhum controle sobre quais campos do
Data e hora são exibidas.Talvez você queira omitir o ano, mas adicione um
Dia da semana até o formato da data.Você quer que o mês seja representado
numericamente ou soletrado pelo nome?A classe intl.dateTimeFormat
fornece controle de granulação fina sobre o que é emitido com base no
propriedades no objeto de opções que é passado como o segundo argumento para
o construtor.Observe, no entanto, que intl.dateTimeFormat não pode
Sempre exiba exatamente o que você pede.Se você especificar opções para
Formatar horas e segundos, mas omitir minutos, você descobrirá que o
O formatador exibe os minutos de qualquer maneira.A ideia é que você use o
Opções se opõem a especificar em que data e hora os campos você gostaria de apresentar
para o usuário e como você gostaria daqueles formatados (por nome ou por número,
por exemplo), então o formatador procurará um local apropriado
Formato que mais combina com o que você pediu.
As opções disponíveis são as seguintes.Especifique apenas propriedades para
campos de data e hora que você gostaria de aparecer no formato
saída.
ano
Usar
"numérico"
por um ano completo de quatro dígitos ou
"2 dígitos"
para um
dois dígitos
abreviação.

mês
Usar
"numérico"
para um número possivelmente curto como "1", ou
"2-
dígito "
Para uma representação numérica que sempre tem dois dígitos,
como "01".Usar
"longo"
Para um nome completo como "janeiro",
"curto"
para um nome abreviado como "Jan", e
"estreito"
por um altamente
Nome abreviado como "J" que não é garantido para ser único.
dia
Usar
"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
Usar
"longo"
Para um nome completo como "segunda -feira",
"curto"
para um
nome abreviado como "seg" e
"estreito"
por 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
"longo"
, Assim,
"curto"
, e
"estreito"
.
hora
, Assim,
minuto
, Assim,
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
Esse
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.Usar
"longo"
por 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
hora12
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.
Usar
"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"
especificar que a meia -noite é de 24.
Aqui estão alguns exemplos:
deixar

d

=

novo

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"
).
formatar
(
d
)

```
2 de janeiro de 2020 "
Intl
.
DATETIMEFORMAT
(
"Es-es"
, Assim,

Optos
).
formatar
(
d
)

// => "Jueves, 2

de Enero de 2020 "
// O tempo em Nova York, para um canadense de língua francesa
Optos

=

{

hora
:

"numérico"
, Assim,

minuto
:

"2 dígitos"
, Assim,

fuso horário
:

"America/new_york"

};
Intl
.
DATETIMEFORMAT
(
"FR-CA"
, Assim,

Optos
).
formatar
(
d
)

// => "8 h 14"
```

Intl.dateTimeFormat pode exibir datas usando

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
organizar()
Método em uma variedade de cordas.
Mas
Se você criar um objeto intl.collator, poderá passar o
comparar()
método desse objeto para o
organizar()
Método para executar o local-
Classificação apropriada das cordas.Os objetos intl.Collator podem ser
configurado para que o
comparar()
O método executa 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 ()
, Assim,
o
Intl.collator ()
Construtor 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
"organizar"
, mas você também pode especificar
"procurar"
.
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.)
"sotaque"
considera acentuos em comparações, mas ignora
caso.
"caso"
considera o caso e ignora os sotaques.E
"variante"
realiza comparações rigorosas que consideram os dois casos
e sotaques.O valor padrão para esta propriedade é
"variante"
quando
uso
é
"organizar"
.Se
uso
é
"procurar"
, então o
A sensibilidade padrão depende do local.
ignorepuncção
Defina esta propriedade para
verdadeiro
para ignorar espaços e pontuação quando
Comparando strings.Com esta propriedade definida como
verdadeiro
, as cordas “qualquer
Um "e" qualquer pessoa ", por exemplo, serão considerados iguais.
numérico
Defina esta propriedade para
verdadeiro
Se as cordas que você está 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.

é definido em outro lugar.Sintaxe da função de seta (ver
§8.1.3
) funciona
particularmente bem com esses métodos, e nós o usaremos nos exemplos
isso a seguir.
Foreach ()
O
foreach ()
método
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:
deixar

dados

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
],

soma

=

0
;
// calcular a soma dos elementos da matriz
dados
.
foreach
(
valor

=>

{

soma

+=

```
const

Collator

=

novo

Intl
.
Collator
().
comparar
;
[[
"um"
, Assim,

"Z"
, Assim,

"UM"
, Assim,

"Z"
].
organizar
(
Collator
)

// => ["a", "a",

"Z", "Z"]
// Os nomes de arquivos geralmente incluem números, então devemos classificá -los

especialmente
const

FILENAMEORDER

=

novo

Intl
.
Collator
(
indefinido
, Assim,

{

numérico
:

verdadeiro
```

```
const

Tradicionalpanish

=

Intl
.
Collator
(
"Es-es-u-co-
trading "
).
comparar
;
deixar

Palabras

=

[[
"Luz"
, Assim,

"lhama"
, Assim,

"Como"
, Assim,

"Chico"
];
Palabras
.
organizar
(
ModernSpanish
)

// => ["chico", "como",

"Llama", "Luz"]
Palabras
.
organizar
(
Tradicionalpanish
)

// => ["Como", "Chico",

"Luz", "Llama"]
```

11.8 A API do console

Você tem
vi o
console.log ()
função usada ao longo disso
Livro: Em navegadores da web, ele imprime uma string na guia "Console" do

Essas funções são quase idênticas a
console.log ()
.Em
Nó,
console.error ()
envia sua saída para o fluxo Stderr
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 do típico
afirmar()
funções,
console.assert ()
não faz 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 ()
Tentativas de exibir seu
argumento
em 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.Grupend ()
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 isso 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
chamar 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
console.time ()
, então imprime isso
string seguida pelo tempo decorrido desde
console.time ()
chamar.Se houver algum argumento adicional para
console.timelog ()
, Assim,
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
console.time ()
, então imprime isso
argumento e o tempo decorrido.Depois de ligar
console.TimeEnd ()
, não é mais legal ligar

console.timelog ()
Sem primeiro chamado
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
, Assim,
%eu
, Assim,
%d
, Assim,
%f
, Assim,
%o
, Assim,
%O
, ou
%c
, então este primeiro argumento é tratado como
formato string,
e os valores dos argumentos subsequentes são substituídos
na corda no lugar do dois caracteres
%
Sequências.
Os significados das seqüências são os seguintes:
%s
O argumento é convertido em uma string.
%eu
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 exibem detalhes do objeto.O
A variante da maçaneta usa um formato de saída dependente da implementação
Isso é considerado mais útil para desenvolvedores de software.
6

%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, o
%c
Sequência e It
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 ()
, Assim,
isto
é impresso automaticamente junto com seu rastreamento de pilha.
11.9 URL APIs
Desde
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
.
Criar um objeto de URL com o
Url ()
construtor, 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:

```
deixar

url

=

novo

Url
(
"https://example.com:8000/path/name?
q = termo#fragmento "
);
url
.
Href

// => "https://example.com:8000/path/name?
q = termo#fragmento "
url
.
origem

// => "https://example.com:8000"
url
.
protocolo

// => "https:"
url
.
hospedar

// => "exemplo.com:8000"
url
.
nome do host

// => "exemplo.com"
url
.
porta

// => "8000"
url
.
Nome do caminho

// => "/caminho/nome"
url
.
procurar

// => "? Q = termo"
url
.
```

```
servidor
url
.
Nome do caminho

=

"API/Pesquisa"
;

// Adicione um caminho para

um endpoint da API
url
.
procurar

=

"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:

```
deixar

url

=

novo

Url
(
"https://example.com"
);
url
.
Nome do caminho

=

"Caminho com espaços"
;
url
.
procurar

=
```

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 o
SearchParams
A propriedade será mais
útil que o
procurar
propriedade.O
procurar
A propriedade é a
Leia/escreva string que permite obter e defina toda a parte de consulta do
Url.O
SearchParams
A propriedade é uma referência somente leitura a um
URLSearchParams
objeto, que tem uma API para obter, definindo,
Adicionando, excluindo e classificando os parâmetros codificados na consulta
parte do URL:
deixar

url

=

novo

Url
(
"https://example.com/search"
);
url
.
procurar

// => "": sem consulta ainda
url
.
SearchParams
.
acrescentar
(
"Q"
, Assim,

"prazo"
);

// Adicione uma pesquisa

parâmetro
url
.
procurar

// => "? Q = termo"
url
.

Erro ao traduzir esta página.

escapar()
e
UNESCAPE ()
funções, que agora estão preteridas
mas ainda amplamente implementado.Eles não devem ser usados.
Quando
escapar()
e
UNESCAPE ()
foram obsoletos, ecmascript
Introduziu dois pares de funções globais alternativas:
codeuri ()
e
decodeuri ()
codeuri ()
leva
uma string como 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 à codificação de URLs inteiros, não
escapar caracteres separadores de URL, como
/
, Assim,
?
, e
#
.Mas isso
significa isso
codeuri ()
não pode funcionar corretamente para URLs que
Tenha esses personagens em seus vários componentes.
codeuricomponent ()
e
decodeuricomponent ()
Esse
Par de funções funciona como
codeuri ()
e
decodeuri ()
exceto que eles pretendem escapar do indivíduo
componentes de um URI, então eles também escapam de personagens como
/
, Assim,
?
, e
#
que são usados ■■para separar esses componentes.Estes são os mais
Útil das funções do URL legado, mas esteja ciente de que
codeuricomponent ()

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 ()
- isso permite
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 que impressam mensagens de console após um segundo,
dois segundos e três segundos:

```
setTimeout
(()

=>

{

console
.
registro
(
"Preparar..."
);

},

1000
);
setTimeout
(()

=>

{

console
.
registro
(
"definir..."
);

},
```

Se você omitir o segundo argumento para
setTimeout ()
, ele padrão é 0.
Isso não significa, no entanto, que a função que você especifica é invocada
imediatamente.Em vez disso, a função é registrada para ser chamada de “assim que
possível."Se um navegador estiver particularmente ocupado lidando com a entrada do usuário ou outro
eventos, pode levar 10 milissegundos ou mais antes que a função seja
invocado.
setTimeout ()
Registre uma função a ser invocada uma vez.Às vezes,
Essa função será chamada
setTimeout ()
para agendar outro
invocação em um momento futuro.Se você quiser invocar uma função repetidamente,
No entanto, muitas vezes é mais simples de usar
setInterval ()
.
setInterval ()
leva os mesmos dois argumentos que
setTimeout ()
mas chama a função repetidamente sempre que o número especificado de
milissegundos (aproximadamente) foram decorridos.
Ambos
setTimeout ()
e
setInterval ()
retornar um valor.Se você
Salvar esse valor em uma variável, você pode usá -lo posteriormente para cancelar o
execução da função passando para
ClearTimeout ()
ou
ClearInterval ()
.O valor retornado é normalmente um número na web
navegadores e é um objeto no nó.O tipo real não importa e
Você deve tratá -lo como um valor opaco.A única coisa que você pode fazer com
Este valor está passando para
ClearTimeout ()
para cancelar a execução de um
função registrada com
setTimeout ()
(supondo que não tenha sido
invocou ainda) ou para interromper a execução repetida de uma função registrada
com
setInterval ()
.
Aqui está um exemplo que demonstra o uso de
setTimeout ()
, Assim,
setInterval ()
, e
ClearInterval ()
Para exibir um simples

Relógio digital com a API do console:

```
// Uma vez por segundo: limpe o console e imprima a corrente

tempo
deixar

relógio

=

setInterval
(()

=>

{

console
.
claro
();

console
.
registro
(
novo

Data
().
tolocaletimestring
());
},

1000
);
// Após 10 segundos: Pare o código repetido acima.
setTimeout
(()

=>

{

ClearInterval
(
relógio
);

},

10000
);
```

Vamos ver
setTimeout ()
e
setInterval ()
novamente quando nós
cubra a programação assíncrona em

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 encontrar
surpreendente.
3
Matrizes digitadas foram introduzidas pela primeira vez no JavaScript do lado do cliente quando os n
avegadores 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.Ver
§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
Iterável
Objetos 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
para/de
loop, como vimos em
§5.4.4
:
deixar

soma

=

0
;
para
(
deixar

eu

de

[[
1
, Assim,
2
, Assim,
3
]))

{

// loop uma vez para cada um desses valores

soma

+=

eu
;
}
soma

// => 6
Iteradores
também pode ser usado com o
...
operador para expandir ou "espalhar"
um objeto iterável em um inicializador de matriz ou invocação de funções, como nós
viu em
§7.1.2

Objetos de data
pode ser comparado com o padrão do JavaScript
<
, Assim,
<=
, Assim,
>
, 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 h
ora
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
setDate ()
, Assim,
setmonth ()
, e
Setyear ()
.Aqui, por exemplo, é o código que adiciona três meses e
duas semanas até a data atual:
deixar

d

=

novo

Data
();
d
.
setmonth
(
d
.
getmonth
()

+

3
, Assim,

d
.
getDate
()

+

O

para/de
O operador 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
iterável
Objetos: esses são tipos como matriz, conjunto e mapa que podem ser
iterado.Segundo, existe o
iterador
o próprio objeto, que executa o
iteração.E terceiro, existe o
resultado da iteração
objeto que detém o
resultado de cada etapa da iteração.
Um
iterável
objeto 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 da iteração
Objeto é um objeto com propriedades nomeadas
valor
e
feito
.Para iterar
Um objeto iterável, você primeiro chama seu método de iterador para obter um iterador
objeto.Então, você chama o
próximo()
método do objeto iterador
repetidamente até que o valor retornado tenha seu
feito
propriedade definida como
verdadeiro
.
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 poderia ser escrito da maneira mais difícil, como
esse:
deixar

iterável

=

[[
99

(Isto é, tem um método chamado
Symbol.iterator
Isso apenas retorna
por si só.) Isso ocasionalmente é útil em código como o seguinte quando você
Quer iterar embora um iterador "parcialmente usado":

```
deixar

lista

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
deixar

iter

=

lista
[[
Símbolo
.
iterador
] ();
deixar

cabeça

=

iter
.
próximo
().
valor
;

// cabeça == 1
deixar

cauda

=

[...
iter
];

// Tail == [2,3,4,5]
```

```
* 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.
*/
aula

Faixa

{

construtor

(
de
, Assim,

para
)

{

esse
.
de

=

de
;

esse
.
para

=

para
;

}

// Faça um intervalo agir como um conjunto de números

tem
(
x
)

{

retornar

typeof

x
```

que terminamos.

},

// Como conveniência, fazemos o próprio iterador

iterável.

[[
Símbolo
.
iterador
] ()

{

retornar

esse
;

}

};

}
}
para
(
deixar

x

de

novo

Faixa
(
1
, Assim,
10
))

console
.
registro
(
x
);

// Logs números 1

a 10
[...
novo

Faixa
(

```
[...
mapa
(
novo

Faixa
(
1
, Assim,
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
função

filtro
(
iterável
, Assim,

predicado
)

{

deixar

iterador

=

iterável
[[
Símbolo
.
iterador
] ();

retornar

{

// este objeto é tanto iterador quanto iterável
```

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

Isso, em vez de tomar uma sequência de origem como argumento, leva o nome de
um arquivo, abre o arquivo, lê linhas e itera as palavras de
essas linhas.Na maioria dos sistemas operacionais, programas que abrem arquivos para ler
a partir deles precisam lembrar de fechar esses arquivos quando terminarem
lendo, então este iterador hipotético certamente fecharia o arquivo
depois do
próximo()
O método retorna a última palavra nela.
Mas os iteradores nem sempre correm até o fim: um
para/de
laço
pode ser encerrado com um
quebrar
ou
retornar
ou por uma exceção.
Da mesma forma, quando um iterador é usado com a atribuição de destruição, o
próximo()
o método é chamado apenas tempo suficiente para obter valores para cada
das variáveis ■■especificadas.O iterador pode ter muito mais valores
poderia voltar, mas eles nunca serão solicitados.
Se nosso iterador hipotético em um arquivo nunca corre até o caminho até o
Fim, ele ainda precisa fechar o arquivo aberto.Por esse motivo, iterador
Objetos podem implementar um
retornar()
método para acompanhar o
próximo()
método.Se a iteração parar antes
próximo()
devolveu um
resultado da iteração com o
feito
propriedade definida como
verdadeiro
(mais comumente
Porque você deixou um
para/de
loop cedo por meio de um
quebrar
declaração), então
O intérprete verificará se o objeto Iterador tem um
retornar()
método.Se esse método existir, o intérprete o invocará sem
Argumentos, dando ao iterador a chance de fechar arquivos, liberar
memória e limpe -se depois de si.O
retornar()
método
deve retornar um objeto de resultado do iterador.As propriedades do objeto são
ignorado, mas é um erro retornar um valor não-objeto.
O
para/de
Loop e o operador de espalhamento são recursos realmente úteis de

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 poderoso novo ES6
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
Crie um gerador, você deve primeiro definir um
Função do gerador
.UM
A função do gerador é sintaticamente como uma função JavaScript regular, mas
é definido com a palavra -chave
função*
em vez de
função
.
(Tecnicamente, essa não é uma palavra -chave nova, apenas um
*
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.
Chamando seu
próximo()
o método causa o corpo da função do gerador
Para funcionar desde o início (ou qualquer que seja sua posição atual) até chegar
um
colheita
declaração.
colheita
é novo no ES6 e é algo como um
retornar
declaração.O valor do
colheita
declaração se torna o
valor retornado pelo
próximo()
Ligue para o iterador.Um exemplo faz
Este mais claro:

```
// uma função de gerador que gera o conjunto de um dígito
```

(Base-10) Prima.
função
*

```
OneDigitPries
()

{

// invocar esta função faz
```

colheita

2
;

// mas apenas retorna um gerador

objeto.Chamando

colheita

3
;

// O Método Next ()

Gerador é executado

colheita

5
;

// o código até um rendimento

A declaração fornece

colheita

7
;

// o valor de retorno para o

Método seguinte ().
}
// Quando invocamos a função do gerador, temos um gerador
deixar

primos

=

OneDigitPries
();
// um gerador é um objeto de iterador que itera o

valores produzidos
primos
.
próximo
().
valor

// => 2
primos
.
próximo
().

Nas classes e literais de objetos, podemos usar a notação de talha para omitir o
função
Palavra -chave inteiramente quando definimos métodos.Para definir um
gerador neste contexto, simplesmente usamos um asterisco antes do método
nome onde o
função
Palavra -chave teria sido, se tivéssemos usado:
deixar

o

=

{

x
:

1
, Assim,

y
:

2
, Assim,

z
:

3
, Assim,

// um gerador que gera cada uma das chaves deste

objeto

*
g
()

{

para
(
deixar

chave

de

Objeto
.
chaves
(
esse
))

{

Erro ao traduzir esta página.

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
organizar()
Método em uma variedade de cordas.
Mas
Se você criar um objeto intl.collator, poderá passar o
comparar()
método desse objeto para o
organizar()
Método para executar o local-
Classificação apropriada das cordas.Os objetos intl.Collator podem ser
configurado para que o
comparar()
O método executa 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 ()
, Assim,
o
Intl.collator ()
Construtor 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:

```
}

}
}
// interlame três objetos iteráveis
[...
Zip
(
OneDigitPries
(),
"Ab"
, [
0
]]]

// =>

[2, "A", 0,3, "B", 5,7]
```

12.3.2 Rendimento* e geradores recursivos

Em adição ao zip () gerador definido no exemplo anterior, Pode ser útil ter uma função de gerador semelhante que produz o elementos de múltiplos objetos iteráveis nnsequencialmente, em vez de intercalando -os.Poderíamos escrever aquele gerador assim:

```
função
*

Sequência
(...
iterables
)

{

para
(
deixar

iterável

de

iterables
)

{

para
(
deixar

item

de

iterável
)
```

```
colheita
*

iterável
;

}
}
[...
Sequência
(
"abc"
, Assim,
OneDigitPries
())]

// =>

["A", "B", "C", 2,3,5,7]
```

A matriz
foreach ()
o método é frequentemente uma maneira elegante de fazer um loop
os elementos de uma matriz, então você pode ser tentado a escrever o
Sequência ()
função assim:
função

```
*

Sequência
(...
iterables
)

{

iterables
.
foreach
(
iterável

=>

colheita
*

iterável

);

//

Erro
}
```

Isso não funciona, no entanto.
colheita
e
colheita*
só pode ser usado

Erro ao traduzir esta página.

```
// mas está disponível se você ligar explicitamente a seguir ()
deixar

gerador

=

Oneanddone
();
gerador
.
próximo
()

// => {value: 1, feito: false}
gerador
.
próximo
()

// => {value: "done", feito: true

}
// Se o gerador já estiver feito, o valor de retorno não será

voltou novamente
gerador
.
próximo
()

// => {valor: indefinido, feito:

verdadeiro }
```

12.4.2 O valor de uma expressão de rendimento

Em
A discussão anterior, tratamos
colheita
como uma declaração que
leva um valor, mas não tem valor próprio.De fato, no entanto,
colheita
é
uma expressão e pode ter um valor.
Quando o
próximo()
O método de um gerador é invocado, o gerador
a função é executada até atingir um
colheita
expressão.A expressão que
segue o
colheita
a palavra -chave 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
colheita
expressão.

```
console
.
registro
(
"Next () invocou uma segunda vez com argumento"
, Assim,

Y1
);

deixar

y2

=

colheita

2
;

// y2 == "c"

console
.
registro
(
"Next () invocou uma terceira vez com argumento"
, Assim,

y2
);

deixar

y3

=

colheita

3
;

// y3 == "d"

console
.
registro
(
"Next () invocou a quarta vez com argumento"
, Assim,

y3
);

retornar

4
```

Erro ao traduzir esta página.

um gerador.Chamando o
lançar()
o método 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
lançar()
Reiniciaria o contador para zero.
Quando um gerador usa
colheita*
Para produzir valores de outro
objeto iterável, então uma chamada para o
próximo()
Método do gerador
causa uma chamada para o
próximo()
método do objeto iterável.O mesmo é
verdadeiro do
retornar()
e
lançar()
Métodos.Se um gerador usa
colheita*
em um objeto iterável que tem esses métodos definidos, então
chamando
retornar()
ou
lançar()
no gerador causa o iterador
retornar()
ou
lançar()
método a ser chamado por sua vez.Todos os iteradores
deve
tem um
próximo()
método.Iteradores que precisam limpar depois
iteração incompleta
deve
definir a
retornar()
método.E qualquer
iterador
poderia
definir a
lançar()
método, embora eu não conheça nenhum
razão prática para fazê -lo.

12.4.4 Uma nota final sobre geradores

Geradores
são uma estrutura de controle generalizada muito poderosa.Eles dão
nós a capacidade de pausar um cálculo com
colheita
e reiniciá -lo novamente em
Alguns arbitrários posteriormente com um valor de entrada arbitrária.
É possível

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
assíncrono
e
aguarde
palavras -chave
(ver
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
para/de
loop e o
...
espalhar o trabalho do operador 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
próximo()
método que retorna um
objeto de resultado da iteração.
Um objeto de resultado da iteração tem um
valor
propriedade que detém o
Próximo valor iterado, se houver um.Se a iteração tiver
concluído, então o objeto de resultado deve ter um
feito
propriedade
definido como
verdadeiro
.
Você pode implementar seus próprios objetos iteráveis, definindo um
[Symbol.iterator] ()
método que retorna um objeto
com um
próximo()
Método que retorna objetos de resultado da iteração.
Você também pode implementar funções que aceitam o iterador
argumentos e valores de retorno do iter.

Erro ao traduzir esta página.

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
programas, no entanto, são significativamente
assíncrono
.Isso significa isso
Eles geralmente precisam parar de calcular enquanto aguardam a chegada dos dados ou
Para que algum evento ocorra.JavaScript
programas em um navegador da web são
tipicamente
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.O
palavras -chave
assíncrono
e
aguarde
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
para/aguardar
O loop 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,
assíncrono
, Assim,
aguarde
, 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

No
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 assíncronia é quando você deseja executar alguns
O código após um certo período de tempo foi decorrido.Como vimos em
§11.10
, Assim,
você
pode fazer isso com o
setTimeout ()
função:

setTimeout
(
checkForupDates
, Assim,

60000
);
O primeiro argumento para
setTimeout ()
é uma função e a segunda é
Um intervalo de tempo medido em milissegundos.No código anterior, um
hipotético
checkForupDates ()
A função será chamada de 60.000
milissegundos (1 minuto) após o
setTimeout ()
chamar.
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 especificado 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
deixar

UpdateIntervalid

=

setInterval
(
checkForupDates
, Assim,

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

Erro ao traduzir esta página.

Detalhes (como o tempo e o ponteiro do mouse coordenam) sobre o
evento.
13.1.3 Eventos de rede
Outro
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
(
VersionCallback
)

{

// Observação

argumento de retorno de chamada

// Faça uma solicitação HTTP com script para uma API de versão de back -end

deixar

solicitar

=

novo

XmlHttPrequest
();

solicitar
.
abrir
(
"PEGAR"
, Assim,

"http://www.example.com/api/version"
);

solicitar
.
enviar
();

// Registre um retorno de chamada que será invocado quando o

A resposta chega

solicitar
.
ONLOAD

=

função

Lado do cliente
O código JavaScript 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 (nós podemos
imagine que é usado pelo hipotético
checkForupDates ()
função que discutimos em
§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 exemplo de código, atribuindo funções ao
ONLOAD
, Assim,
OnError
, Assim,
e
OnTimeout
propriedades.Por convenção, propriedades do ouvinte de eventos
como esses sempre têm nomes que começam com
sobre
.
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 notar 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
nulo
primeiro argumento e o número da versão como o
segundo argumento.Ou, se ocorrer um erro, então
getCurrentVersionNumber ()
Invoca 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
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

=

exigir
(
"FS"
);

// O módulo "FS" tem sistema de arquivos-
APIs relacionadas
deixar

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"
, Assim,

"UTF-8"
```

Ao contrário das declarações de função, as declarações de classe não são
"Içado."Lembrar 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
aula
corpo prefixando o
declaração de método com o
estático
palavra -chave.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 a
Exemplo 9-3
:

```
estático

analisar
(
s
)

{

deixar

partidas

=

s
.
corresponder
(
/^\(\d+)\.\.\.(\d+)\)$/
);

se

(
!
partidas
)

{

lançar

novo
```

```
// Então, registramos mais manipuladores de eventos para serem chamados

Quando chegar.

resposta
.
SetEncoding
(
"UTF-8"
);

// estamos esperando

Texto unicode

deixar

corpo

=

""
;

// que vamos

acumule aqui.

// Este manipulador de eventos é chamado quando um pedaço do

O corpo está pronto

resposta
.
sobre
(
"dados"
, Assim,

pedaço

=>

{

corpo

+=

pedaço
;

});

// Este manipulador de eventos é chamado quando a resposta é

completo

resposta
```

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
getText ()
função no anterior
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
mais
linear
Cadeia de promessa
Isso tende a ser mais fácil de ler e mais fácil de
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 ao
setTimeout ()
função, por exemplo.Mas
Não podemos usar promessas para substituir
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 capítulo longo
com cuidado.

[[
0
]

===

1
;
Hoje, as arquiteturas de CPU mais comuns são
Little-Endian.Muitos
protocolos de rede e alguns formatos de arquivo binário exigem
Big-Endian
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
deixar

visualizar

=

novo

DataView
(
bytes
.
buffer
, Assim,

bytes
.
byteoffset
, Assim,

bytes
.
BytELength
);
deixar

int

=

visualizar

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()
vai ser
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
então()
.
No
um nível sintático simples, o
então()
Método é o distinto
característica das promessas, e é idiomático anexar
.então()
diretamente para
a invocação de funções que retorna a promessa, sem o
Etapa intermediária de atribuir o objeto de promessa a um
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/usuário/perfil"
).
então
(
DisplayUserProfile
);
Lidar com erros com promessas
Assíncrono
operações, particularmente aquelas que envolvem networking,
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/usuário/perfil"
).
então
(
DisplayUserProfile
, Assim,

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 a
então()
.
Quando algo dá errado em um cálculo síncrono, ele joga
uma exceção que se propaga na pilha de chamadas até que haja um
pegar
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 ()
corre normalmente, 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

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
organizar()
Método em uma variedade de cordas.
Mas
Se você criar um objeto intl.collator, poderá passar o
comparar()
método desse objeto para o
organizar()
Método para executar o local-
Classificação apropriada das cordas.Os objetos intl.Collator podem ser
configurado para que o
comparar()
O método executa 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 ()
, Assim,
o
Intl.collator ()
Construtor 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:

Programação e falamos sobre promessas humanas, dizemos que uma promessa é "mantida" ou "quebrada".Qu
ando
Discutindo promessas de JavaScript, os termos equivalentes são "cumpridos" e "rejeitados".Imagine qu
e você
chamou o
então()
Método de uma promessa e já passou duas funções de retorno de chamada.Nós dizemos
que a promessa foi
cumprido
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 é
pendente
.E uma vez que uma promessa é cumprida ou rejeitada, dizemos que é
assentou
.Observação
que uma promessa nunca pode ser cumprida
e
rejeitado.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 assíncro
no terminar - eles representam o
Resultados desse código assíncrono.Se o código assíncrono executar normalmente (e a promessa é cumpr
ida), então isso
O resultado é essencialmente o valor de retorno do código.E se o código assíncrono não concluir norm
almente
(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 ch
amada registrada
com
pegar()
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 po
dem 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
então()
invocações de método, sem ter que ninho
cada operação dentro do retorno de chamada do anterior.Aqui, para

```
})

.
então
(
renderizado

=>

{

// Quando conseguirmos o

documento renderizado

CacheIndatabase
(
renderizado
);

// cache no

Banco de dados local.

})

.
pegar
(
erro

=>

lidar
(
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 nova e baseada em promessa (
§15.11.1
).Em sua forma mais simples, esta
Nova API HTTP é apenas a função
buscar()
.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:

```
buscar
(
```

```
se

(
eu

===

j
)

{

se

(
j

===

k
)

{

console
.
registro
(
"Eu igual a k"
);

}
}

outro

{

// que diferença a localização de uma cinta encaracolada

faz!

console
.
registro
(
"Eu não igual a J"
);
}
```

Muitos programadores têm o hábito de envolver os corpos de
`se`
e
`outro`
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

Argumentos que são passados ■■para os métodos:
buscar
().
então
().
então
()
Quando
Mais de um método é invocado em uma única expressão como esta,
nós chamamos isso de
cadeia de métodos
.Nós sabemos que o
buscar()
função 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
.então()
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
.então()
invocações, não estamos registrando vários retornos de chamada em um único
Objeto de promessa.Em vez disso, cada invocação do
então()
método
Retorna um novo objeto de promessa.Esse novo objeto de promessa não é cumprido
até que a função passou para
então()
é
completo.
Vamos voltar a uma forma simplificada do original
buscar()
cadeia acima.
Se definirmos as funções passadas para o
então()
invocações
Em outros lugares, podemos refatorar o código para ficar assim:
buscar
(
theurl
)

// Tarefa 1;Retorna a promessa 1

.
então
(
chamada de chamada1
)

// Tarefa 2;Retorna Promise 2

.

1
.
Na primeira linha,
buscar()
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
então()
método de
promessa 1, passando o
chamada de chamada1
função que queremos
ser chamado quando a promessa 1 for cumprida.O
então()
método
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
chamada de chamada1
é
invocado.
3
.
Na terceira linha, invocamos o
então()
método de promessa
2, passando pelo
chamada de chamada2
função queremos invocar quando
A promessa 2 é cumprida.Esse
então()
O método se lembra do nosso
retorno de chamada e retorna mais uma promessa.Vamos dizer essa "tarefa
3 ”começa quando
chamada 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 do
buscar()
Chamada 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

assim:
// estende o range predefinido.prototipo objeto, então não

substituir
// O Range.prototype automaticamente criado

propriedade.
Faixa
.
protótipo
.
inclui

=

função
(
x
)

{

retornar

esse
.
de

<=

x

&&

x

<=

esse
.
para
;
};
Faixa
.
protótipo
.
ToString

=

função
()

{

retornar

"("

```
deixar

P4

=

resposta
.
JSON
();

retornar

P4
;

// Retorna Promise 4
}
função

C2
(
perfil
)

{

// retorno de chamada 2

DisplayUserProfile
(
perfil
);
}
deixar

P1

=

buscar
(
"/API/usuário/perfil"
);

// Promise 1, Tarefa 1
deixar

P2

=

P1
.
então
(
C1
);
```

Essa não é uma promessa, é imediatamente cumprida com esse valor.
Então, se
c
retorna uma não promessa, esse valor de retorno se torna o valor de
p
, Assim,
p
é cumprido e terminamos.Mas se o valor de retorno
v
é em si a
Promessa, então
p
é
resolvido, mas ainda não cumprido
.Nesta fase,
p
não pode
se estabelecer até a promessa
v
se estabelece.Se
v
é cumprido, então
p
vai ser
cumprido com o mesmo valor.Se
v
é rejeitado, então
p
será rejeitado para
a mesma razão.É isso que o estado "resolvido" de uma promessa significa:
A promessa tornou -se associada a ou "trancada", outro
Promessa.Ainda não sabemos se
p
será cumprido ou rejeitado, mas
nosso retorno de chamada
c
não tem mais controle sobre isso.
p
está "resolvido" em
A sensação de que seu destino agora depende inteiramente do que acontece para prometer
v
.
Vamos trazer isso de volta ao nosso exemplo de URL.Quando
C1
retorna
P4
, Assim,
P2
está resolvido.Mas ser resolvido não é o mesmo que ser
cumprido, então a Tarefa 3 ainda não começa.Quando o corpo inteiro do HTTP
Resposta fica disponível, então o
.json ()
Método pode analisá -lo
e use esse valor analisado para cumprir
P4
.Quando
P4
é cumprido,
P2

Figura 13-1.
Buscando um URL com promessas
13.2.4 Mais sobre promessas e erros
Mais cedo
No capítulo, vimos que você pode passar um segundo retorno de chamada
função para o
.então()
método 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
.então()
método.Em vez disso, erros relacionados à promessa são normalmente
manipulado adicionando um
.pegar()
invocação de método para 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
Boa notícia é que o
.pegar()
o método facilita o manuseio
erros ao trabalhar com promessas.
A captura e finalmente métodos
O

.pegar()
O método de promessa é simplesmente uma maneira de abreviação de chamar
.então()
com
nulo
como o primeiro argumento e um manuseio 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
.
então
(
nulo
, Assim,

c
);
p
.
pegar
(
c
);
O
.pegar()
abreviação é preferida porque é mais simples e
Porque o nome corresponde ao
pegar
Cláusula em a
tente/capturar
declaração de manipulação de exceção.Como discutimos, exceções normais
Não trabalhe com código assíncrono.O
.pegar()
método 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
pegar
bloquear.
Com uma cadeia de promessas assíncronas, a metáfora comparável
pode ser um erro "escorrendo pela corrente" até encontrar um
.pegar()
invocação.
Em
ES2018, Promise Objects também definem um
.finalmente()
método
cujo objetivo é semelhante ao
finalmente
Cláusula em a
tente/capturar/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
.finalmente()
O retorno de chamada é a maneira ideal de fazer isso.Como
.então()
e
.pegar()
, Assim,

resolve ou rejeita com.Se a
.finalmente()
O retorno de chamada lança um
exceção, no entanto, então a promessa devolvida por
.finalmente()
vai
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:

```
buscar
(
"/API/usuário/perfil"
)

// Inicie a solicitação HTTP

.
então
(
resposta

=>

{

// Ligue para isso quando o status e

Cabeçalhos estão prontos

se

(
!
resposta
.
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.
```

```
outro

{

// Se recebemos um erro 404 acima e retornamos

nulo nós acabamos aqui

DisplayLoggedOutProfilePage
();

}

})

.
pegar
(
e

=>

{

se

(
e

Instância de

NetworkError
)

{

// fetch () pode falhar dessa maneira se a internet

A conexão está baixa

DisplayErRormessage
(
"Verifique sua internet

conexão."
);

}

outro

se

(
e

Instância de

TypeError
```

Objeto NetworkError.Nós não passamos um retorno de chamada de manipulação de erros
funcionar como o segundo argumento para o
.então()
Ligue para, então
P2
rejeita como
Bem, com o mesmo objeto NetworkError.(Se tivéssemos passado um erro
manipulador para aquele primeiro
.então()
Ligue para o manipulador de erros seria invocado,
e se retornar normalmente,
P2
seria resolvido e/ou cumprido com
o valor de retorno desse manipulador.) Sem um manipulador, no entanto,
P2
é
rejeitado, e então
P3
é rejeitado pelo mesmo motivo.Neste ponto, o
C3
Retorno de chamada de manipulação de erros é chamado e o NetworkError específico
código dentro dele é executado.
Outra maneira de nosso código falhar é se nossa solicitação http retornar um 404
Não encontrado ou outro erro HTTP.Essas são respostas http válidas, então
o
buscar()
A chamada não os considera erros.
buscar()
encapsula um 404 não encontrado em um objeto de resposta e atende
P1
com
esse objeto, causando
C1
para ser invocado.Nosso código in
C1
verifica o
OK
propriedade do objeto de resposta para detectar que não recebeu um
Resposta normal HTTP e lida com esse caso simplesmente retornando
nulo
.Porque esse valor de retorno não é uma promessa, ele cumpre
P2
certo
longe, e
C2
é chamado com esse valor.Nosso código in
C2
explicitamente
verifica e lida com valores falsamente, exibindo um resultado diferente para
o usuário.Este é um caso em que tratamos uma condição anormal como um
Não -erro e lide com isso sem realmente usar um manipulador de erros.
Um erro mais sério ocorre em
C1
Se obtemos um http normal
Código de resposta, mas o cabeçalho do tipo conteúdo não está definido adequadamente.
Nosso código espera uma resposta formatada por JSON; portanto, se o servidor estiver
Enviando -nos html, xml ou texto simples, em vez disso, teremos um
problema.
C1

O cabeçalho está errado, trata isso como um problema não recuperável e joga um TypeError.Quando um retorno de chamada passou para .então() (ou .pegar()) lança 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 levanta um tipo de causa P2 para ser rejeitado com esse objeto TypeError. Já que não especificamos um manipulador de erros para P2 , Assim, P3 será rejeitado como bem. C2 não será chamado, e o TypeError será passado para C3 , Assim, 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 regular e síncrono lançar a declaração acaba sendo tratada de forma assíncrona com um .pegar() 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 .então() , e também por que é tão idiomático terminar as correntes de promessa com um .pegar() chamar. Antes de deixarmos o tópico de manuseio de erros, quero ressaltar que, Embora seja idiomático terminar todas as cadeias de promessas com um .pegar() Para limpar (ou pelo menos registrar) quaisquer erros que ocorreram na cadeia, é Também perfeitamente válido para usar .pegar() 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 correr, então você pode inserir um .pegar() ligue para o cadeia, resultando em código que pode ser assim: startasyncoperation ()

.
então
(
Dostagetwo
)

.

.
então
(
Dostagethree
)

.
então
(
DostageFour
)

.
pegar
(
LOGSTAGETHREEAND e FouRERRORS
);
Lembre -se de que o retorno de chamada que você passa
.pegar()
só será
Invocado se o retorno de chamada em um estágio anterior lançar um erro.Se o
Retorno de chamada retorna normalmente, então o
.pegar()
Retorno de chamada será
pulados, e o valor de retorno do retorno de chamada anterior se tornará o
entrada para o próximo
.então()
ligar de volta.Lembre -se também disso
.pegar()
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
.pegar()
Retorno de chamada, ele para de propagar a cadeia de promessas.UM
.pegar()
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 ()
joga um erro,
então o
recuperar destagetwoerror ()
função será
invocado.Se
recuperar destagetwoerror ()
retorna normalmente,
então seu valor de retorno será passado para
Dostagethree ()
e o
A operação assíncrona continua normalmente.Por outro lado, se
recuperar destagetwoerror ()
não conseguiu se recuperar, vai
Por si só, eleva um erro (ou ele repete o erro que ele foi aprovado).Em
Este caso, também não
Dostagethree ()
nem
DostageFour ()

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
()

.
então
(
displayTable
)

.
pegar
(
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 um
.pegar()
chamar:

```
Querydatabase
()

.
pegar
(
e

=>

espere
(
500
).
então
(
Querydatabase
))

// Sobre

fracasso, espere e tente novamente

.
então
(
displayTable
)

.
pegar
(
DisplayDatabaseError
);
```

Se as falhas hipotéticas forem verdadeiramente aleatórias, adicionando esta linha
do código deve reduzir sua taxa de erro de 1% para 0,01%.

```
.
pegar
(
e

=>

espere
(
500
).
então
(
Querydatabase
))
```

Lembrar de
Capítulo 8
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 expres
são
torna -se o valor de retorno da função.Devido a esses atalhos, o código anterior está correto.
Mas considere essa mudança inócua:

```
.
pegar
(
e

=>

{

espere
(
500
).
então
(
Querydatabase
)

})
```

Ao adicionar os aparelhos encaracolados, não obtemos mais o retorno automático.Esta função agora ret
orna
indefinido
Em vez de devolver uma promessa, o que significa que a próxima etapa nesta cadeia de promessas irá
ser invocado com
indefinido
como sua entrada e não o resultado da consulta refúgio.É um erro sutil
Isso pode não ser fácil de depurar.

13.2.5 promessas em paralelo

Nós temos
Passei 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.O
função
Promete.all ()
pode fazer isso.

```
.
pegar
(
e

=>

console
.
erro
(
e
));
```

Promete.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
Promete.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.
Em
ES2020,
Promise.AllSettled ()
leva uma variedade de informações
Promete e retorna uma promessa, assim como
Promete.all ()
faz.Mas
Promise.AllSettled ()
nunca rejeita a promessa devolvida, e isso
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 um
status
propriedade
definido como "cumprido" ou "rejeitado".Se o status for "cumprido", então o objeto
também terá um
valor
propriedade que fornece o valor de atendimento.E
Se o status for "rejeitado", o objeto também terá um
razão
propriedade que fornece o erro ou o valor de rejeição do correspondente
Promessa:

```
Promessa
.
Allettled
([
Promessa
.
resolver
(
1
),

Promessa
.
```

Erro ao traduzir esta página.

O código é trivial porque o objeto de resposta do
buscar()
API
tem um predefinido
JSON ()
método.O
JSON ()
O método retorna a
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
devolvido por
getjson ()
atende ao mesmo valor.Observe que há
Sem tratamento de erros neste
getjson ()
implementação.Em vez de
verificando
resposta.OK
e o cabeçalho do tipo de conteúdo, em vez disso
Apenas permita o
JSON ()
método para rejeitar 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

Gethighscore
()

{

retornar

getjson
(
"/API/usuário/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 um
Highscore

Promise.Resolve ()
e
Promise.Reject ()
fará o quê
você quer.
Promise.Resolve ()
aceita um valor como seu único
argumento e retorna uma promessa que será imediatamente (mas
assíncrono) ser cumprido a esse valor.De forma similar,
Promise.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.)
Lembrar de
§13.2.3
que uma promessa resolvida não é a mesma coisa que um
promessa cumprida.Quando ligamos
Promise.Resolve ()
, nós normalmente
Passe o valor do cumprimento para criar um objeto de promessa que em breve
cumprir com esse valor.O método não é nomeado
Promise.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
Promise.Resolve ()
e
Promise.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
Promise.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
pegar
cláusula e use um assíncrono
.pegar()
método para manusear
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
ambos
getjson ()
e
Gethighscore ()
, começamos por
chamando uma função existente para obter uma promessa inicial e criada e
devolveu uma nova promessa chamando o
.então()
Método desse 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
Promessa()
construtor para criar um novo
Promise objeto que você tem controle completo.Aqui está como é
Trabalhos: você invoca o
Promessa()
construtor e passar uma função como
seu único argumento.A função que você passa deve ser escrita para esperar
Dois parâmetros, que, por convenção, devem ser nomeados
resolver
e
rejeitar
.O construtor chama de síncrona sua função com
Funções argumentos para o
resolver
e
rejeitar
parâmetros.Depois
chamando sua função, o
Promessa()
construtor 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 o
resolver
função para
resolver ou cumprir a promessa devolvida ou chamar o
rejeitar
função para

rejeitar a promessa devolvida.Sua função não precisa ser
assíncrono: pode ligar
resolver
ou
rejeitar
Síncrono, 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 promessa baseada em promessa
espere()
função que usamos em vários exemplos anteriormente no capítulo:
função

espere
(
duração
)

{

// Crie e retorne uma nova promessa

retornar

novo

Promessa
((
resolver
, Assim,

rejeitar
)

=>

{

// Esses

Controle a promessa

// Se o argumento for inválido, rejeite a promessa

se

(
duração

<

0
)

{

rejeitar

valor, que cumpre a promessa devolvida com esse valor.
Exemplo 13-1
é outro exemplo de usar o
Promessa()
construtor.Este implementa nosso
getjson ()
função para uso em
Nó, onde o
buscar()
API não está 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

=

exigir
(
"http"
);
função

getjson
(
url
)

{

// Crie e retorne uma nova promessa

retornar

novo

Promessa
((
resolver
, Assim,

rejeitar
)

=>

{

// Inicie um http get solicitação para o URL especificado

solicitar

=

```
resposta
.
sobre
(
"dados"
, Assim,

pedaço

=>

{

corpo

+=

pedaço
;

});

resposta
.
sobre
(
"fim"
, Assim,

()

=>

{

// Quando o corpo de resposta estiver completo, tente

para analisá -lo

tentar

{

deixar

analisado

=

JSON
.
analisar
(
corpo
);

// se for divulgado com sucesso, cumpra

a promessa
```

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

```
função

buscar sequesteramente
(
URLs
)

{

// Vamos armazenar os corpos de URL aqui enquanto os buscamos

const

corpos

=

[];

// Aqui está uma função de denúncia de promessa que busca uma

corpo

função

FetchOne
(
url
)

{

retornar

buscar
(
url
)

.
então
(
resposta

=>

resposta
.
texto
())

.
então
(
corpo

=>
```

Com isso
FetchSequencialmente ()
função definida, poderíamos buscar
os URLs um de cada vez com código como o código de busca em paralelo
Usamos anteriormente para demonstrar
Promete.all ()
:

```
buscar sequesteramente
(
URLs
)

.
então
(
corpos

=>

{

/* Faça algo com a matriz de

Strings */

})

.
pegar
(
e

=>

console
.
erro
(
e
));
```

O
FetchSequencialmente ()
A função 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

```
Promisesequence ()
parece se chamar recursivamente, mas porque
A chamada "recursiva" é através de um
então()
método, não há realmente
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
, Assim,

Promesemaker
)

{

// Faça uma cópia privada da matriz que podemos modificar

entradas

=

[...
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
```

```
retornar

Promesemaker
(
NextInput
)

// Calcule o

Próximo valor de saída,

// então crie uma nova matriz de saídas com o

novo valor de saída

.
então
(
saída

=>

saídas
.
Concat
(
saída
))

// então "Recurse", passando o novo, mais tempo,

matriz de saídas

.
então
(
handlenextInput
);

}

}

// Comece com uma promessa que cumpre uma matriz vazia

e uso

// A função acima como seu retorno de chamada.

retornar

Promessa
.
resolver
([]).
então
(
handlenextInput
);
```

Outros eventos assíncronos.Embora ainda seja importante entender
como
Promessas
trabalho, grande parte de sua complexidade (e às vezes até
sua própria presença!) desaparece quando você os usa com
assíncrono
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
.pegar()
método.
assíncrono
e
aguarde
Tome 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
O
aguarde
A palavra -chave 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 estabelece.Se
p
cumpre, então o valor de
aguarde
p
é o valor de cumprimento de
p
.Por outro lado, se
p
é rejeitado, então
o
aguarda p
expressão lança o valor de rejeição de
p
.Nós não
geralmente use
aguarde
com uma variável que mantém uma promessa;Em vez disso, nós
Use -o antes da invocação de uma função que retorna uma promessa:
deixar

resposta

=

É fundamental entender imediatamente que o
aguarde
A palavra -chave 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
aguarde
Simplesmente disfarça esse fato.Isso significa isso
Qualquer código que use

aguarde

é
ele próprio assíncrono
.
13.3.2 Funções assíncronas
Porque qualquer código que use
aguarde
é assíncrono, há um
Regra Crítica:
você só pode usar o
aguarde
palavra -chave dentro de funções que
foram declarados com o
assíncrono
palavra -chave
.Aqui está uma versão do
Gethighscore ()
função de anterior no capítulo, reescrito para
usar
assíncrono
e
aguarde
:

```
assíncrono

função

Gethighscore
()

{

deixar

resposta

=

aguarde

buscar
(
"/API/usuário/perfil"
);

deixar

perfil
```

palavra -chave com ele:
DisplayHighScore
(
aguarde

Gethighscore
());
Mas lembre -se, essa linha de código só funcionará se estiver dentro de outro
assíncrono
função!Você pode nidificar
aguarde
expressões dentro
assíncrono
funciona tão profundamente quanto você quiser.Mas se você estiver no nível superior
ou são
dentro de uma função que não é
assíncrono
Por algum motivo, então você não pode usar
aguarde
e tem que lidar com uma promessa devolvida da maneira regular:
Gethighscore
().
então
(
DisplayHighScore
).
pegar
(
console
.
erro
);
Você pode usar o
assíncrono
palavra -chave com qualquer tipo de função.Funciona
com o
função
palavra -chave como uma declaração ou como expressão.Isto
trabalha com funções de seta e com a forma de atalho de método em
Aulas e literais de objeto.(Ver
Capítulo 8
Para mais sobre os vários
maneiras de escrever funções.)
13.3.3 Aguardando várias promessas
Suponha que escrevemos nosso
getjson ()
função usando
assíncrono
:
assíncrono

função

getjson
(
url
)

{

%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, o
%c
Sequência e It
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 ()
, Assim,
isto
é impresso automaticamente junto com seu rastreamento de pilha.
11.9 URL APIs
Desde
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
.
Criar um objeto de URL com o
Url ()
construtor, 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

```
}

});
}
```

É mais difícil expressar o
aguarde
palavra -chave em termos de sintaxe
transformação como esta.Mas pense no
aguarde
palavra -chave como a
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 do
aguarde
-A promessa marcada que o 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
assíncrono
funções e o
aguarde
declarações para essas coisas.
ES2018
fornece uma solução, no entanto.Iteradores assíncronos são como
os iteradores descritos em
Capítulo 12
, mas eles são baseados em promessa e
devem ser usados ■■com uma nova forma do
para/de
laço:
para/aguardar
.

13.4.1 O loop for/wait

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/aguardar
Loop como este:

```
const

fs

=

exigir
(
"FS"
);
assíncrono

função

Parsefile
(
nome do arquivo
)

{

deixar

fluxo

=

fs
.
Createradstream
(
nome do arquivo
, Assim,

{

codificação
:

"UTF-8"
});

para

aguarde

(
deixar

pedaço

de

fluxo
)
```

buscado.(Claro, a primeira busca pode levar mais tempo do que qualquer um dos outros, então isso não é necessariamente mais rápido do que usar Promete.all ()
.)
Matrizes são iteráveis, para que possamos iterar através da variedade de promessas com um regular
para/de
laço:

```
para
(
const

promessa

de

promessas
)

{

resposta

=

aguarde

promessa
;

lidar
(
resposta
);
}
```

Este código de exemplo usa um regular
para/de
Faça um loop com um iterador regular.
Mas como esse iterador retorna promessas, também podemos usar o novo
para/aguardar
Para um código um pouco mais simples:

```
para

aguarde

(
const

resposta

de

promessas
)

{

lidar
(
```

um que pode ser usado com um
para/de
laço.Define um método com o
nome simbólico
Symbol.iterator
.Este método retorna um
iterador
objeto.O objeto iterador tem um
próximo()
método, que pode ser chamado
repetidamente para obter os valores do objeto iterável.O
próximo()
Método do objeto de iterador retorna
resultado da iteração
objetos.O
O objeto de resultado da iteração tem um
valor
propriedade e/ou a
feito
propriedade.
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
Symbol.iterator
.(Como
Vimos anteriormente,
para/aguardar
é compatível com iterável regular
objetos, mas prefere objetos de maneira assíncrona e tenta o
Symbol.asynciterator
método antes de tentar o
Symbol.iterator
Método.) 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
para/aguardar
regularmente, de maneira síncrona
Matriz de promessas iteráveis, estávamos trabalhando com o resultado do iterador síncrono
objetos em que o
valor
propriedade era um objeto de promessa, mas o
feito
propriedade
era síncrono.Os verdadeiros iteradores assíncronos retornam promessas para o resultado da iteração
objetos e ambos
valor
e o
feito
As propriedades 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
nós vimos em
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
aguarde
Como faria em um regular
função assíncrona, e você pode usar
colheita
Como faria em um regular
gerador.Mas valores que você
colheita
são automaticamente envolvidos em
Promessas.Até a sintaxe para geradores assíncronos é uma combinação:
função assíncrona
e
função *
combinar -se
assíncrono
função *
.Aqui está um exemplo que mostra como você pode usar um
gerador assíncrono e um
para/aguardar
loop para executar o código repetidamente em
intervalos fixos usando sintaxe de 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

Tempo decorrido
(
EM
)

{

retornar

novo

Promessa
(
resolver

=>

setTimeout
(
resolver

```
// uma função de teste que usa o gerador assíncrono com

para/aguardar
assíncrono

função

teste
()

{

// assíncrono então nós

pode usar para/aguardar

para

aguarde

(
deixar

marcação

de

relógio
(
300
, Assim,

100
))

{

// loop 100

vezes a cada 300ms

console
.
registro
(
marcação
);

}
}
```

13.4.4 Implementando iteradores assíncronos

Em vez de
de usar geradores assíncicos para implementar iteradores assíncronos,
Também é possível implementá -los diretamente, definindo um objeto
com um
Symbol.asynciterator ()
método que retorna um objeto
com um
próximo()

assíncrono

próximo
()

{

// O método Next () faz

Este é um iterador

se

(
esse
.
contar

>

máx
)

{

// Terminamos?

retornar

{

feito
:

verdadeiro

};

// resultado de iteração

indicando feito

}

// descobrir quando a próxima iteração deve

começar,

deixar

TargetTime

=

esse
.
StartTime

+

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.O
relógio()
função
discutido anteriormente era bastante simples de escrever porque a fonte de
A assincha foi a
setTimeout ()
ligações 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
ligue para o iterador
próximo()
o método deve retornar uma promessa distinta
objeto, e várias chamadas para
próximo()
pode 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 assíncicos a seguir tem
enquistar ()
e
Dequeue ()
Métodos como você esperaria para uma aula de filas.O
Dequeue ()
o método retorna uma promessa em vez de um valor real,
No entanto, o que significa que não há problema em ligar
Dequeue ()
antes
enquistar ()
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
3

```
preso.(Asyncqueue tem a
fechar()
método.Uma vez chamado, não
Mais valores podem ser envolvidos.Quando uma fila fechada está vazia, o
para/aguardar
Loop vai parar de loop.)
Observe que a implementação do AsyncQueue não usa
assíncrono
ou
aguarde
e, em vez disso, trabalha 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.)
*/
aula

Assíncrono

{

construtor
()

{

// valores que foram na fila, mas ainda não

são armazenados aqui

esse
.
valores

=

[];

// quando as promessas são desquedas antes de seus
```

```
}

enquadre
(
valor
)

{

se

(
esse
.
fechado
)

{

lançar

novo

Erro
(
"Assyncqueue fechadas"
);

}

se

(
esse
.
resolvedores
.
comprimento

>

0
)

{

// Se esse valor já foi prometido,

Resolva essa promessa

const

resolver

=

esse
.
resolvedores
```

```
// resolva as promessas pendentes com o final de
marcador de fluxo

enquanto
(
esse
.
resolvedores
.
comprimento

>

0
)

{

esse
.
resolvedores
.
mudança
() (
Assíncrono
.
EOS
);

}

esse
.
fechado

=

verdadeiro
;

}

// define o método que torna esta classe de forma assíncrona

iterável

[[
Símbolo
.
assíncerador
] ()

{

retornar

esse
;
```

```
elemento do documento
// em um objeto assíncrono e devolva a fila para uso como

um fluxo de eventos
função

EventStream
(
ELT
, Assim,

tipo
)

{

const

q

=

novo

Assíncrono
();

// Crie a

fila

ELT
.
addEventListener
(
tipo
, Assim,

e
=>
q
.
enquadre
(
e
));

// Enqueue

eventos

retornar

q
;
}
assíncrono

função
```

codificar em um único
pegar()
Ligue no final de uma cadeia de
então()
chamadas.
O
assíncrono
e
aguarde
Palavras -chave 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
assíncrono
, retornará implicitamente uma promessa.Dentro de um
assíncrono
função, você pode
aguarde
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/aguardar
laço.Você pode criar itemerável de forma assíncrona
objetos implementando um
[Symbol.asynciterator]
()
método ou invocando um
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 pelo
buscar()
API, que é coberta em
§15.11.1
.O exemplo de código mostrado aqui é o último exemplo baseado em xmlhttprequest
permanecendo neste livro.
2
Você normalmente pode usar
aguarde
no nível superior no console do desenvolvedor de um navegador.E
há uma proposta pendente para permitir o nível superior
aguarde
em uma versão futura do JavaScript.
3
Aprendi sobre essa abordagem da iteração assíncrona do blog do Dr. Axel
Rauschmayer,
https://2ality.com
.

Capítulo 14.
Metaprogramação

Esse
O capítulo abrange vários recursos avançados de JavaScript que são
não é comumente usado na programação do dia-a-dia, mas isso pode ser
valioso para os programadores que escrevem bibliotecas reutilizáveis ■■e de interesse para
Quem quer mexer com os detalhes sobre como JavaScript
Objetos se comportam.
Muitos
Dos recursos descritos aqui, podem ser descritos livremente como
"Metaprograma": se a programação regular estiver escrevendo código para
manipular dados, então a metaprogramação está escrevendo código para manipular
outro código.Em uma linguagem dinâmica como JavaScript, as linhas entre
Programação e metaprogramação estão embaçadas - mesmo o simples
capacidade de iterar sobre as propriedades de um objeto com um
para/in
laço
pode ser considerado "meta" por programadores acostumados a mais estáticos
idiomas.

Os tópicos de metaprogramação abordados neste capítulo incluem:

§14.1
Controlar a enumerabilidade, deleteabilidade e
Configurabilidade das propriedades do objeto

§14.2
Controlar a extensibilidade dos objetos e criar
Objetos "selados" e "congelados"

§14.3
Consulta e definindo os protótipos de objetos

§14.4
Ajuste 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
refletir
Métodos
§14.7
Controlar o comportamento do objeto com proxy
14.1 Atributos da propriedade
O
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
gravável
atributo
especifica se o valor de um
A propriedade pode mudar.
O
enumerável
atributo
Especifica se a propriedade é
enumerado pelo
para/in
loop e o
Object.keys ()
método.
O
configurável
atributo
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.
Lembrar de
§6.10.6
que, enquanto “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
, Assim,
gravável
, Assim,
enumerável
, e
configurável
.Propriedades do acessador não
tem um
valor
atributo ou a
gravável
atributo: a escritura deles é
determinado pela presença ou ausência de um levantador.Então os quatro atributos
de uma propriedade acessadora são
pegar
, Assim,
definir
, Assim,
enumerável
, e
configurável
.
O
Métodos JavaScript para consultar e definir os atributos de um
Propriedade Use um objeto chamado A
Descritor de propriedades
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
, Assim,
gravável
, Assim,
enumerável
, e
configurável
.E o
descritor para uma propriedade de acessórios tem
pegar
e
definir
propriedades em vez disso
de
valor
e

```
// retorna {value: 1, gravável: verdadeiro, enumerável: verdadeiro,

Configurável: True}
Objeto
.
getOwnPropertyDescriptor
({
x
:

1
},

"X"
);
// Aqui está um objeto com uma propriedade de acessador somente leitura
const

aleatório

=

{

pegar

octeto
()

{

retornar

Matemática
.
chão
(
Matemática
.
aleatório
()
*
256
);

},
};
// retorna {get:/*func*/, set: indefinido, enumerável: true,

Configurável: True}
Objeto
.
getOwnPropertyDescriptor
(
aleatório
, Assim,

"Octeto"
);
```

```
configurável
:

verdadeiro
});
// verifique se a propriedade está lá, mas não é entusiasmada
o
.
x

// => 1
Objeto
.
chaves
(
o
)

// => []
// Agora modifique a propriedade x para que seja somente leitura
Objeto
.
DefinirProperty
(
o
, Assim,

"X"
, Assim,

{

gravável
:

falso

});
// 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:
```

isso deve ser modificado.O segundo argumento é um objeto que mapeia o
nomes das propriedades a serem criadas ou modificadas para a propriedade
descritores para essas propriedades.Por exemplo:

```
deixar

p

=

Objeto
.
defineproperties
({},

{

x
:

{

valor
:

1
, Assim,

gravável
:

verdadeiro
, Assim,

enumerável
:

verdadeiro
, Assim,

configurável
:

verdadeiro

},

y
:

{

valor
:

1
, Assim,

gravável
:
```

Object.DefineProperties ()
lançar
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 não extensível (ver
§14.2
) objeto.O outro
razões que esses métodos podem lançar o TypeError têm a ver com o
atributos.O
gravável
atributo governa as tentativas de
mude o
valor
atributo.E o
configurável
atributo 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 ()
Essa tentativa de 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
um acessador
propriedade.
Se uma propriedade de dados não estiver configurável, você não pode alterar seu
gravável
atributo de
falso
para
verdadeiro
, mas você pode mudar isso
de
verdadeiro
para
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 o
Object.assign ()
função que copia a propriedade
valores de um ou mais objetos de origem em um objeto de destino.
Object.assign ()
Somente 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 em vez
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
```

```
* setters.
*
* Object.assignDescriptors () propaga qualquer tipo de TypeErrors lançado

por
* Object.DefineProperty ().Isso pode ocorrer se o objeto de destino

está selado
* ou congelado ou se alguma das propriedades da fonte tentar mudar

um existente
* Propriedade não configurável no objeto de destino.
*
* Observe que a propriedade do atribuído cenário é adicionada ao objeto

com
* Object.DefineProperty () para que a nova função possa ser

criado como
* Uma propriedade não enumerável como object.assign ().
*/
Objeto
.
DefinirProperty
(
Objeto
, Assim,

"atribuiScriptors"
, Assim,

{

// corresponde aos atributos de object.assign ()

gravável
:

verdadeiro
, Assim,

enumerável
:

falso
, Assim,

configurável
:

verdadeiro
, Assim,

// a função que é o valor dos atribuintes

propriedade.

valor
:
```

deixar

o

=

{
c
:

1
, Assim,

pegar

contar
()

{
retornar

esse
.
c
++
;}};

// define

objeto com getter
deixar

p

=

Objeto
.
atribuir
({},

o
);

// copie o

Valores da propriedade
deixar

q

=

Objeto
.
atribuiDescriptores
({},

o

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
extensível
atributo é ser capaz de "travar"
objetos em um estado conhecido e impedem adulteração externa.O
extensível
o atributo de objetos é frequentemente usado em conjunto com o
configurável
e
gravável
atributos 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 ()
Bloqueia 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
Object.Seal ()
e
Object.freeze ()
afetar 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 ()
, Assim,
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
deixar

o

=

Objeto
.
selo
(
Objeto
.
criar
(
Objeto
.
congelar
({
x
:

1
}),

{
y
:

{
valor
:

2
, Assim,

gravável
:

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.

valor, que cumpre a promessa devolvida com esse valor.
Exemplo 13-1
é outro exemplo de usar o
Promessa()
construtor.Este implementa nosso
getjson ()
função para uso em
Nó, onde o
buscar()
API não está 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

=

exigir
(
"http"
);
função

getjson
(
url
)

{

// Crie e retorne uma nova promessa

retornar

novo

Promessa
((
resolver
, Assim,

rejeitar
)

=>

{

// Inicie um http get solicitação para o URL especificado

solicitar

=

Objeto
.
protótipo
.
isPrototypeof
(
o
)

// => true: o também
Observe que
isprototypeof ()
desempenha uma função semelhante ao
Instância de
operador (veja
§4.9.4
).
O
protótipo
atributo 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.SetProtTypeOf ()
:
deixar

o

=

{
x
:

1
};
deixar

p

=

{
y
:

2
};
Objeto
.
setPrototypeof
(
o
, Assim,

p
);

// Defina o protótipo de O para P

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.getPrototypeOf ()
e
Object.SetProtTypeOf ()
.Um uso interessante de
__proto__
, no entanto, é definir o protótipo de um objeto literal:
deixar

p

=

{
z
:

3
};
deixar

o

=

{

x
:

1
, Assim,

y
:

2
, Assim,

__proto__
:

p
};
o
.
z

// => 3: o herda de P
14.4 Símbolos bem conhecidos
O

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
O
Symbol.iterator
e

Symbol.asynciterator
Símbolos
permitir que objetos ou classes se tornem iteráveis nnou assíncronos
iterável.Eles foram cobertos em detalhes em
Capítulo 12
e
§13.4.2
, Assim,
respectivamente, e são mencionados novamente aqui apenas para
integridade.
14.4.2 Symbol.HasInsinStance
Quando
o
Instância de
O operador foi descrito em
§4.9.4
, nós dissemos que
o lado da direita deve ser uma função construtora e que o
expressão
o Instância de f
foi avaliado procurando o valor
F.Prototipo
dentro da cadeia de protótipos de
o
.Isso ainda é verdade, mas
em ES6 e além,
Símbolo.hasinstance
fornece uma alternativa.
Em ES6, se o lado direto de
Instância de
é qualquer 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 de
operador.E, é claro, se o valor no caminho
lado não tem um
[Symbol.hasinstance]
método, mas é um
função, então o
Instância de
O operador se comporta de maneira comum.
Símbolo.hasinstance
significa que podemos usar o
Instância de
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
deixar

uint8

=

{

[[
Símbolo
.
hasinstance
] (
x
)

{

retornar

Número
.
isinteger
(
x
)

&&

x

> =

0

&&

x

<=

255
;

}
};
128

Instância de

uint8

// => true
256

Instância de

uint8
```

Número]"
Objeto
.
protótipo
.
ToString
.
chamar
(
falso
)

// => "[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
Classof ()
a função é indiscutivelmente mais útil que o
typeof
Operador, que não faz distinção entre tipos de objetos:
função

classe de
(
o
)

{

retornar

Objeto
.
protótipo
.
ToString
.
chamar
(
o
).
fatiar
(
8
, Assim,
-
1
);
}
classe de
(
nulo
)

Object.prototype.toString ()
procura uma propriedade com o
nome simbólico
Symbol.ToStringTag
em seu argumento, e se assim
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 ()
:
aula

Faixa

{

pegar

[[
Símbolo
.
ToStringtag
] ()

{

retornar

"Faixa"
;

}

// O resto desta classe é omitido aqui
}
deixar

r

=

novo

Faixa
(
1
, Assim,

10
);
Objeto
.
protótipo
.
ToString
.
chamar
(
r

e
emenda ()
, 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:
Em ES6 e mais tarde, o
Variedade()
Construtor tem uma propriedade
com o nome simbólico
Symbol.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
estende -se
, 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
mapa()
e
fatiar()
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
novo
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:

Variedade
[[
Símbolo
.
espécies
]

=

Variedade
;
Nesse caso, os construtores de subclasse herdariam o
Variedade()
construtor como sua "espécie" e invocando
mapa()
em uma matriz
A subclasse retornaria uma instância da superclasse em vez de um
instância da subclasse.Não é assim que o ES6 realmente se comporta,
no entanto.A razão é que
Array [Symbol.Species]
é uma leitura
Somente propriedade de acessor cuja função getter simplesmente retorna
esse
.
Os construtores de subclasse herdam essa função getter, o que significa que por
Padrão, todo construtor de subclasse é sua própria "espécie".
Às vezes, esse comportamento padrão não é o que você deseja, no entanto.Se você
queria os métodos de retorno de matriz de ezarray para retornar a matriz regular
objetos, você só precisa definir
EZARRAY [SYMBOL.SPECES]
para
Variedade
.Mas como a propriedade herdada é um acessador somente leitura, você
Não pode simplesmente configurá -lo com um operador de atribuição.Você pode usar
DefineProperty ()
, no entanto:
Ezarray
[[
Símbolo
.
espécies
]

=

Variedade
;

// tentam definir uma leitura
Somente a propriedade falha
// em vez disso, podemos usar defineproperty ():
Objeto
.
DefinirProperty
(
Ezarray
, Assim,

Símbolo

```
pegar

durar
()

{

retornar

esse
[[
esse
.
comprimento
-
1
];

}
}
deixar

e

=

novo

Ezarray
(
1
, Assim,
2
, Assim,
3
);
deixar

f

=

e
.
mapa
(
x

=>

x

-

1
);
e
.
durar
```

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

aula

Não é adprendido

estende -se

Variedade

{

pegar

[[
Símbolo
.
ISCONCATSPREATABLE
] ()

{

retornar

falso
;

}
}
deixar

um

=

novo

Não é adprendido
(
1
, Assim,
2
, Assim,
3
);
[].
Concat
(
um
).
comprimento

// => 1;(seria 3

elementos longos se A foi espalhado)
14.4.6 Símbolos de correspondência de padrões
§11.3.2
documentado
os métodos de string que executam 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

Objeto de padrão como este:
corda
.
método
(
padrão
, Assim,

Arg
)
Essa invocação se transforma em uma invocação de um nome simbolicamente nomeado
Método em seu objeto de padrão:
padrão
[[
símbolo
] (
corda
, Assim,

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
Globs
:
aula

Glob

{

construtor
(
glob
)

{

esse
.
glob

=

glob
;

// Implementamos a correspondência global usando regexp

internamente.

//?corresponde a qualquer personagem exceto /, e *

```
[[
Símbolo
.
procurar
] (
s
)

{

retornar

s
.
procurar
(
esse
.
regexp
);

}

[[
Símbolo
.
corresponder
] (
s
)

{

retornar

s
.
corresponder
(
esse
.
regexp
);

}

[[
Símbolo
.
substituir
] (
s
, Assim,

substituição
)

{
```

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
"corda"
, isso 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"
, isso significa que 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
<
ou
>
operador ou com aritmética operadores como
-
e
*
.
Se o argumento for
"padrão"
, isso significa que JavaScript é convertendo seu objeto em um contexto em que um numérico ou O valor da string pode funcionar.Isso acontece com o
+
, Assim,
==
, 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 definir a
[Symbol.Toprimitive]
método.

14.4.8 Symbol.unscopables

O
Símbolo bem conhecido final que abordaremos aqui é obscuro que foi introduzido como uma solução alternativa para questões de compatibilidade causadas por

o depreciado
com
declaração.Lembre -se de que o
com
Declaração toma
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 mais tarde, o
com
A declaração 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:
deixar

NewArrayMethods

=

Objeto
.
chaves
(
Variedade
.
protótipo
[[
Símbolo
.
INSCOLHADOS
]);
14.5 Tags de modelo
Cordas
dentro de backticks são conhecidos 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 ser incorporado no código JavaScript.E a biblioteca de emoção usa um
css``
função de tag 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, então a função da tag
será invocado com
n+1
argumentos.O primeiro argumento será um

Matriz de
n+1
cordas, 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
a seguir
html``
modelo, o que é útil quando você quer
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
(
cordas
, Assim,

...
valores
)

{

// converte cada valor em uma string e escape de HTML especial

caracteres

deixar

escapou

=

valores
.
mapa
(
v

=>

Corda
(
v
)

.
substituir
(
"&"
, Assim,

y "
deixar

tipo

=

"jogo"
, Assim,

nome

=

"D&D"
;
html
`<div class ="
$ {
tipo
}
">
$ {
nome
}
 `

// => '<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 padrões definida em
§14.4.6
.Desde o
Glob ()
Construtor toma um
argumento de string única, podemos definir uma função de tag para criar novos
Objetos Glob:
função

glob
(
cordas
, Assim,

...
valores
)

{

// monta as cordas e valores em uma única string

deixar

s

=

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
glob``
Função de tags 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 nosso globo

Literais.
14.6 A API Refletir
O
O objeto reflete 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 o
Refletir
funções não fornecem novos recursos,
Eles agrupam os recursos em uma API conveniente.E,
importante, o conjunto de
Refletir
funções mapeiam um a um com o
Conjunto de métodos de manipulador de proxy sobre os quais aprenderemos em
§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 invoca isso
como uma função sem
esse
valor se
o
é
nulo
) e passa o
valores no
args
Array como argumentos.É equivalente a
F.Applly (O, args)
.
Reflet.Construct (C, Args, NewTarget)
Esta função chama o construtor
c
Como se o
novo
palavra -chave tinha
foi usado e passa os elementos da matriz
args
como argumentos.
Se opcional
newTarget
O argumento é especificado, é usado como o
valor de
new.target
dentro da invocação do construtor.Se não
especificado, então o
new.target
valor será
c
.
Reflete.DefineProperty (O, nome, descritor)
Esta função define uma propriedade no objeto
o
, usando
nome
(um
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
verdadeiro
no sucesso
e
falso
em falhas.(
Object.DefineProperty ()
retorna

método com um getter e se o opcional
receptor
argumento é
especificado, então a função getter é chamada de método de
receptor
em vez de um método de
o
.Chamar esta função é
Semelhante à avaliação
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 ()
, Assim,
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.getPrototypeOf ()
Exceto isso
Object.getPrototypeOf ()
apenas joga um TypeError para
nulo
e
indefinido
Argumentos e coers outros primitivos
valores para seus objetos de wrapper.
Reflete.Has (o, nome)
Esta função retorna
verdadeiro
se o objeto
o
tem uma propriedade com o
especificado
nome
(que deve ser uma corda ou um símbolo).Chamando isso
A função é semelhante à avaliação
nome em o
.
Reflete.isextensible (O)
Esta função retorna
verdadeiro

Refletir.wowys (O)
Esta função retorna uma matriz dos nomes das propriedades do
objeto
o
ou joga um TypeError se
o
não é 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 o deles
Resultados.
Reflete.PreventExtensions (O)
Esta função define o
extensível
atributo (
§14.2
) do objeto
o
para
falso
e retorna
verdadeiro
para indicar sucesso.Joga um
TypeError se
o
não é 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 especificado
nome
do
objeto
o
para o especificado
valor
.Ele retorna
verdadeiro
no sucesso e
falso
na falha (o que pode acontecer se a propriedade for somente leitura).
Joga TypeError se
o
não é um objeto.Se a propriedade especificada for
uma propriedade acessadora com uma função de setter e se o opcional
receptor
argumento é passado, então o levantador será invocado como um
método de
receptor
em vez de ser invocado como um método de

corrente).Joga um TypeError se
o
não é um objeto ou se
p
não é
um objeto nem
nulo
.
Object.SetProtTypeOf ()
é semelhante,
mas retorna
o
Sobre 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
O
A classe de proxy, disponível no ES6 e posterior, é mais
poderoso recurso de metaprogramação.Nos permite escrever código que
altera o comportamento fundamental dos objetos JavaScript.
A API refletida
descrito em
§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:
deixar

Proxy

=

novo

Proxy
(
alvo
, Assim,

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 isso
p
é um objeto de proxy e você
escrever
Exclua P.X
.O
Reflete.DeleteProperty ()
função
tem o mesmo comportamento que o
excluir
operador.E quando você usa o
excluir
operador para excluir uma propriedade de um objeto de proxy, ele procura um
DeleteProperty ()
Método no objeto Manipuladores.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 os mesmos que os das funções refletidas cobertas em
§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:
deixar

t

=

{

x
:

1
, Assim,

y
:

2

};
deixar

p

=

novo

Proxy
(

Útil, no entanto, quando criado como "proxies revogáveis".Em vez de
criando um proxy com o
Proxy ()
construtor, você pode usar o
Proxy.revocable ()
função de fábrica.Esta função retorna um
objeto que inclui
um proxy
objeto e também um
revogar()
função.
Depois de ligar para o
revogar()
função, o proxy para imediatamente
trabalhando:
função

AccessthedAtabase
()

{

/ * Implementação omitida */

retornar

42
;

}
deixar

{
Proxy
, Assim,

revogar
}

=

Proxy
.
revogável
(
AccessthedAtabase
, Assim,

{});
Proxy
()

// => 42: O proxy dá acesso ao subjacente

função alvo
revogar
();

// mas esse acesso pode ser desligado sempre que nós

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
deixar

identidade

=

novo

Proxy
({},

{

// Cada propriedade tem seu próprio nome como seu valor

pegar
(
o
, Assim,

nome
, Assim,

alvo
)

{

retornar

nome
;

},

// Cada nome de propriedade é definido

tem
(
o
, Assim,

nome
)

{
```

```
// Com efeito, isso significa que o objeto não é

extensível

isExtensible
(
o
)

{

retornar

falso
;

},

// Todas as propriedades já estão definidas neste objeto, então

Não poderia

// herdar qualquer coisa, mesmo que tenha um protótipo

objeto.

getPrototypeof
(
o
)

{

retornar

nulo
;

},

// O objeto não é extensível, então não podemos mudar o

protótipo

setPrototypeof
(
o
, Assim,

proto
)

{

retornar

falso
;
```

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

```
* 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
, Assim,

objName
)

{

// Defina manipuladores para o nosso objeto proxy de registro.

// Cada manipulador registra uma mensagem e depois delega para o

objeto alvo.

const

manipuladores

=

{

// 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.

pegar
(
alvo
, Assim,

propriedade
```

Manter esse invariante, as matrizes têm dois comportamentos especiais.O primeiro nós descrito acima: se você atribuir um valor a um elemento de matriz cujo índice
eu
é maior ou igual ao atual da matriz
comprimento
, o valor de
o
comprimento
A propriedade está definida como
I+1
.
O segundo comportamento especial que as matrizes implementam para manter O comprimento invariante é que, se você definir o
comprimento
propriedade para um não
Inteiro negativo
n
menor que seu valor atual, qualquer elementos de matriz cujo índice é maior ou igual a
n
são excluídos da matriz:
um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];

// Comece com uma matriz de 5 elementos.
um
.
comprimento

=

3
;

// a agora é [1,2,3].
um
.
comprimento

=

0
;

// Exclua todos os elementos.a é [].
um

Erro ao traduzir esta página.

// 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
(
deixar

x

de

Proxydata
)

console
.
registro
(
"Dado"
, Assim,

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
Array.map ()
método verifica explicitamente a existência de cada
elemento da matriz (causando o
tem()
manipulador a ser invocado) antes
na verdade, lendo o valor do elemento (que desencadeia o
pegar()
manipulador).
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
nós 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 log: o
Array.toString ()
método
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])
.)

Operador: foi adicionado ao idioma com ES2016.O
Math.pow ()
A função está disponível desde as primeiras versões
de JavaScript, no entanto, e executa exatamente a mesma operação que
o
**
operador.
O
/
O operador divide seu primeiro operando em seu 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
.A divisão por zero produz positivo ou
Infinidade negativa, enquanto
0/0
avalia para
Nan
: nenhum desses casos
levanta um erro.
O
%
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
binário
+
O operador adiciona operandos numéricos ou concatena a string
operando:

Como exemplo, se você criar um proxy para um objeto não extensível, o
Proxy jogará um TypeError se o
isoxtensible ()
manipulador de todos os tempos
retorna
verdadeiro
:
deixar

alvo

=

Objeto
.
prevenir extensões
({});
deixar

Proxy

=

novo

Proxy
(
alvo
, Assim,

{

isExtensible
()

{

retornar

verdadeiro
;

}});
Refletir
.
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

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
Símbolo
objeto tem 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

O
A linguagem JavaScript foi criada em 1994 com o propósito expresso
de ativar o comportamento dinâmico nos documentos exibidos pela Web
navegadores.O
A linguagem 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
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 desencadeado 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:
Controle de conteúdo do documento (
§15.3
) e estilo (
§15.4
)
Determine a posição na tela dos elementos do documento
(
§15.5
)

Se você aprovar uma função, é uma função de substituição - efetivamente o inverso
da função de reviver opcional para você pode passar
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 retorna 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
deixar

texto

=

JSON
.
stringify
(
endereço
, Assim,

[[
"cidade"
, Assim,
"estado"
, Assim,
"país"
]);
// Especifique uma função de substituição que omite o valor regexp

propriedades
deixar

JSON

=

JSON
.
stringify
(
o
, Assim,

(
k
, Assim,

v

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

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>
Web
Os navegadores 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 o HTML
<Cript>
marcação
faz.
O código JavaScript pode aparecer em linha dentro de um arquivo html entre
<Cript>
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 ->
<título>
Relógio digital
</title>
<estilo>

/* Uma folha de estilo CSS para o

relógio */
#relógio

{

/* Os estilos se aplicam ao elemento

com id = "relógio" */

fonte
:

audacioso

24px

Sans-Serif
;
```

```
fronteira
:

sólido

preto

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
()

{

deixar

relógio

=

documento
```

Um arquivo JavaScript contém javascript puro, sem
<Cript>
tags ou
Qualquer outro html.Por convenção, os arquivos do código JavaScript têm nomes
esse fim com
.js
.
UM
<Cript>
tag com o a
src
atributo se comporta exatamente como se o
O conteúdo do arquivo JavaScript especificado apareceu diretamente entre o
<Cript>
e
</script>
tags.Observe que o fechamento
</script>
A tag é necessária nos documentos HTML, mesmo quando o
src
atributo é
especificado: html não suporta um
<script/>
marcação.
Existem várias vantagens em usar o
src
atributo:
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,
usando o
src
atributo 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
src
atributo 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
documentos
JavaScript módulos e cobre seus
importar
e

exportar
diretivas.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 um
<Cript>
Marque isso
tem um
type = "Módulo"
atributo.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
Em
Nos primeiros dias da web, pensava -se que os navegadores poderiam alguns
Dia implementa idiomas que não sejam JavaScript e programadores
Atributos adicionados como
idioma = "JavaScript"
e
type = "Application/JavaScript"
para o deles
<Cript>
tags.
Isso é completamente desnecessário.JavaScript é o padrão (e somente)
linguagem da web.O
linguagem
o atributo está depreciado e lá
são apenas dois motivos para usar um
tipo
atributo em a
<Cript>
marcação:
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
JavaScript foi adicionado primeiro 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

document.write ()
Método para injetar texto HTML no documento
no local do script.
O uso de
document.write ()
não é mais considerado um bom estilo,
mas o fato de ser possível significa que quando o analisador html
Encontros a
<Cript>
elemento, deve, por padrão, executar o script
Só para ter certeza de que ele não produz nenhum html antes de retomar
analisar e renderizar o documento.Isso pode diminuir drasticamente
Analisar e renderizar a página da web.
Felizmente, esse padrão
síncrono
ou
bloqueando
execução do script
O modo não é a única opção.
O
<Cript>
tag pode ter
adiar
e
assíncrono
Atributos, que fazem com que os scripts sejam executados de maneira diferente.Esses
são atributos booleanos - eles não têm um valor;Eles só precisam ser
presente no
<Cript>
marcação.Observe que esses atributos são apenas
significativo quando usado em conjunto com o
src
atributo:
<script
adiar

src =
"adtered.js"
> </script>
<script
assíncrono

src =
"Async.js"
> </script>
Ambos
adiar
e
assíncrono
atributos são maneiras de dizer ao navegador
que o script vinculado não usa
document.write ()
para gerar
Saída HTML, e que o navegador, portanto, pode continuar a analisar
e renderizar o documento ao baixar o script.
O
adiar
atributo faz com que o navegador adie a execução do script até depois
O documento foi totalmente carregado e analisado e está pronto para estar

assíncrono
atributo 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
type = "Módulo"
atributo são, por padrão, executados
Depois que o documento está carregado, como se eles tivessem um
adiar
atributo.Você
pode substituir esse padrão com o
assíncrono
atributo, que causará
o código a ser executado assim que o módulo e todos os seus dependências foram carregadas.
Uma alternativa simples ao
assíncrono
e
adiar
Atributos - 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
importar()
, como descrito em
§10.3.6
.
Se você não estiver usando módulos, pode carregar um arquivo de javascript em demanda simplesmente adicionando um
<Cript>
Marque no seu documento quando
Você quer que o script carregue:

```
// carregar e executar assíncronos e executar um script de um especificado

Url
// retorna uma promessa que resolve quando o script tem

carregado.
função

ImportScript
(
url
)

{

retornar

novo

Promessa
((
resolver
, Assim,

rejeitar
)

=>

{

deixar

s

=

documento
.
CreateElement
(
"Script"
);

// Crie a

<SCRIPT> elemento

s
.
ONLOAD

=

()

=>

{
```

formando uma árvore.Considere o seguinte documento HTML simples:
<html>

<head>

<título>
Documento de amostra
</title>

</head>

<Body>

<H1>
Um documento HTML
</h1>

<p>
Este é um

simples

documento.

</body>
</html>
O nível superior
<html>
tag contém
<head>
e
<Body>
tags.O
<head>
A tag contém a
<título>
marcação.E o
<Body>
tag contém
<H1>
e
<p>
tags.O
<título>
e
<H1>
Tags contêm seqüências de cordas de
texto, e o
<p>
tag contém duas seqüências de texto com um

marcação
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

Erro ao traduzir esta página.

```
.
pegar
(
e

=>

espere
(
500
).
então
(
Querydatabase
))
```

Lembrar de
Capítulo 8
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 expres
são
torna -se o valor de retorno da função.Devido a esses atalhos, o código anterior está correto.
Mas considere essa mudança inócua:

```
.
pegar
(
e

=>

{

espere
(
500
).
então
(
Querydatabase
)

})
```

Ao adicionar os aparelhos encaracolados, não obtemos mais o retorno automático.Esta função agora ret
orna
indefinido
Em vez de devolver uma promessa, o que significa que a próxima etapa nesta cadeia de promessas irá
ser invocado com
indefinido
como sua entrada e não o resultado da consulta refúgio.É um erro sutil
Isso pode não ser fácil de depurar.

13.2.5 promessas em paralelo

Nós temos
Passei 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.O
função
Promete.all ()
pode fazer isso.

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
jogar()
e
pausa()
para
Controlando a reprodução de arquivos de áudio e vídeo.
15.1.3 O objeto global nos navegadores da Web
Lá
é um objeto global por janela ou guia do navegador (
§3.7
).Todos os
Código JavaScript (exceto código em execução em threads de trabalhadores; veja
§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
parseint ()
função, 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, o
documento
A propriedade representa
o documento atualmente exibido, o
buscar()
O método fabrica http
solicitações de rede e o
Áudio ()
Construtor 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
história
(
§15.10.2
),
que representam a história de navegação da janela e
INNERWIDTH
, Assim,
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 um
janela.
prefixo:
Window.innerWidth
é mais claro do que
INNERWIDTH
, por exemplo.

15.1.4 Os scripts compartilham um espaço para nome

Com
módulos, 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 outro
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
função
declarações no nível superior criam

propriedades no objeto global compartilhado.Se um script definir um nível superior
função
f ()
, então outro script no mesmo documento pode invocar que
função como
f ()
ou como
window.f ()
.Por outro lado, o ES6
declarações
const
, Assim,
deixar
, e
aula
, quando usado no nível superior, faça
não criar propriedades no objeto global.Eles ainda estão definidos em um
Namespace compartilhado, no entanto: se um script definir uma classe
C
, outro
scripts poderão criar instâncias dessa classe com
novo C ()
, mas
não com
New Window.c ()
.
Para resumir: em módulos, as declarações de nível superior são escopo para o
módulo e pode ser exportado explicitamente.Em scripts não -módulos, no entanto,
Declarações de nível superior são escopo para o documento contendo e o
As declarações são compartilhadas por todos os scripts no documento.Mais velho
var
e
função
As declarações são compartilhadas por meio de propriedades do objeto global.
Mais recente
const
, Assim,
deixar
, e
aula
declarações também são compartilhadas e têm
o mesmo escopo do documento, mas eles não existem como propriedades de nenhum
Objeto que o código JavaScript tem acesso.

15.1.5 Execução de programas JavaScript

Lá
não é uma definição formal de um
programa
no JavaScript do lado do cliente,
Mas podemos dizer que um programa JavaScript consiste em todo o JavaScript
codificar ou referenciar um documento.Esses bits de código separados
compartilhar um único objeto de janela global, que lhes dá acesso ao
Mesmo objeto de documento subjacente que representa o documento HTML.
Os scripts que não são módulos compartilham adicionalmente um espaço para nome de nível superior.
Se uma página da web inclui um quadro incorporado (usando o
<frame>
elemento), o código JavaScript no documento incorporado tem um

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
<Cript>
elementos (scripts embutidos e scripts externos) é
correr.Os scripts geralmente são executados na ordem em que aparecem no
documento, embora esse 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
fazer
qualquer coisa 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 em
§15.2
.
Alguns
dos primeiros eventos a ocorrer durante a fase orientada a eventos são os
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

A fase dura enquanto o documento for exibido pela web
navegador.Porque esta fase é assíncrona e orientada a eventos, lá, lá
pode ser longos períodos de inatividade em que nenhum JavaScript é executado,
pontuado por explosões de atividade acionadas por eventos de usuário ou rede.
Abordaremos essas duas fases em mais detalhes a seguir.
Modelo de Threading JavaScript do lado do cliente
JavaScript
é uma linguagem de thread única e execução de thread única
cria uma programação muito mais simples: você pode escrever código com o
Garantia de que dois manipuladores de eventos nunca serão executados ao mesmo tempo.Você
pode manipular o conteúdo do documento, sabendo que nenhum outro tópico é
tentando modificá -lo ao mesmo tempo, e você nunca precisa se preocupar
Sobre bloqueios, impasse ou condições de corrida ao escrever JavaScript
código.
Execução de thread única significa que os navegadores da web param de responder
Entrada do usuário enquanto scripts e manipuladores de eventos estão executando.Isso coloca a
Balca de programadores JavaScript: significa que os scripts de JavaScript e
Os manipuladores de eventos não devem correr por muito tempo.Se um script executar um
tarefa computacionalmente intensiva, ela introduzirá um atraso no documento
Carregando e o usuário não verá o conteúdo do documento até o script
completa.Se um manipulador de eventos executa um intensivo computacionalmente intensivo
Tarefa, o navegador pode se tornar não responsivo, possivelmente causando o usuário
pensar que caiu.
O
A plataforma da web define uma forma controlada de simultaneidade chamada A
"Trabalhador da web."
Um trabalhador da web é um tópico de fundo para executar
Tarefas computacionalmente intensivas sem congelar a interface do usuário.O
O código que é executado em um tópico do trabalhador da web não tem acesso a

Você precisará ler os capítulos
12
e
13
Para entender o
para/aguardar
Loop, mas aqui está como fica no código:
// leia pedaços de um fluxo de maneira assíncrona e

Imprima -os
assíncrono

função

PrintStream
(
fluxo
)

{

para

aguarde

(
deixar

pedaço

de

fluxo
)

{

console
.
registro
(
pedaço
);

}
}
5.4.5 para/in
UM

para/in
Loop se parece muito com um
para/de
loop, com o
de
palavra -chave
alterado para
em
.Enquanto um
para/de

Documente o conteúdo que vem antes dele.
3
.
Quando o analisador encontra um
<Cript>
elemento que tem o
assíncrono
Conjunto de atributos, começa a baixar o texto do script (e
Se o script for um módulo, também baixará recursivamente tudo
as dependências do script) e continua analisando o
documento.O script será executado o mais rápido possível após
baixou, mas o analisador não para e espera por isso
para baixar.Scripts assíncronos não devem usar o
document.write ()
método.Eles podem ver seus próprios
<Cript>
tag e todo o conteúdo do documento que vem antes dele,
e pode ou não ter acesso a documentos adicionais
contente.
4
.
Quando o documento é completamente analisado, o
document.readyState
A propriedade muda para "interativa".
5
.
Quaisquer scripts que tivessem o
adiar
Conjunto de atributos (junto com qualquer
scripts de módulo que não têm um
assíncrono
atributo) são
executado na ordem em que eles apareceram no documento.
Os scripts assíncronos também podem ser executados neste momento.Adiado
Os scripts têm acesso ao documento completo e devem
não use o
document.write ()
método.
6
.
O
O navegador dispara um evento "DomContentLoaded" no
Objeto de documento.Isso marca a transição de síncrono
fase de execução de script para o
assíncrono,
orientado a eventos
fase da execução do programa.Observe, no entanto, que pode haver
ainda ser
assíncrono
scripts que ainda não foram executados neste momento.
7
.
O documento é completamente analisado neste momento, mas o
o navegador ainda pode estar esperando por conteúdo adicional, como
imagens, para carregar.Quando todo esse conteúdo termina de carregar e
Quando tudo
assíncrono
scripts foram carregados e executados, o
document.readyState

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, Programas JavaScript do lado do cliente Processar 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 html
<butter>
elementos, ou texto
entrou em html
<Textarea>
elementos, 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
Url ()
construtor (
§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.Ver
§15.12.2
Para mais detalhes.
O global
navegador
A propriedade 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 o usuário preferido

linguagem e
Navigator.hardwarEcoCurrency
Retorna o número de CPUs lógicas disponíveis para a web
navegador.Da mesma forma, o global
tela
Propriedade fornece
acesso ao tamanho de exibição do usuário via
screen.width
e
Screen.Height
propriedades.Em certo sentido, estes
navegador
e
tela
objetos são para navegadores da web que ambiente
variáveis ■■são para o nó
programas.
O JavaScript do lado do cliente normalmente produz saída, quando precisa, por
manipulando o documento HTML com a API DOM (
§15.3
) ou por
usando um nível superior
estrutura
como reação ou angular para manipular
o documento.O código do lado do cliente também pode usar
console.log ()
e
Métodos relacionados (
§11.8
) para produzir saída.Mas esta saída é apenas
Visível no console do desenvolvedor da web, por isso é útil ao depurar,
mas não para saída visível do usuário.

15.1.7 Erros do programa

Diferente
Aplicativos (como aplicativos de nó) que são executados diretamente no topo
Do sistema operacional, os programas JavaScript em um navegador da Web não podem realmente "travar".
Se ocorrer uma exceção enquanto seu programa JavaScript estiver em execução e se
você não tem um
pegar
Declaração para lidar com isso, uma mensagem de erro irá
ser exibido no console do desenvolvedor, mas quaisquer manipuladores de eventos que tenham
foram registrados, continue correndo e respondendo a eventos.
Se você deseja definir um manipulador de erros de último recurso para ser invocado
Quando esse tipo de exceção não capturada ocorre, defina o
OnError
propriedade do objeto de janela para uma função de manipulador de erros.Quando um
a exceção não capturada se propaga até a pilha de chamadas e um
A mensagem de erro está prestes a ser exibida no console do desenvolvedor, o

Window.onerror
A função será invocada com três cordas
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
OnError
manipulador retorna
verdadeiro
, 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
.pegar()
função 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 um
Window.onUnHandledRejection
função ou
usando
window.addeventListener ()
Para registrar um manipulador para
Eventos de “rejeição não entrega”.O objeto de evento passou para este manipulador
terá um
promessa
propriedade cujo valor é o objeto de promessa que
rejeitado e a
razão
propriedade cujo valor é o que teria sido
passou para um
.pegar()
função.Como nos manipuladores de erros descritos
antes, se você ligar
PreventDefault ()
na rejeição não tratada
objeto de evento, ele será considerado tratado e não causará um erro
mensagem no console do desenvolvedor.
Não é frequentemente necessário definir
OnError
ou
OnUnHandledRejection
manipuladores, mas pode ser bastante útil como um
mecanismo de telemetria se você deseja relatar erros do lado do cliente ao
servidor (usando o
buscar()
função 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 que 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

Web
A primeira linha de defesa dos navegadores contra o 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 comunicar
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
O

Política da mesma origem
é uma restrição de segurança abrangente em que web
O código JavaScript de conteúdo pode interagir.Normalmente entra em jogo
Quando uma página da web inclui
<frame>
elementos.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 venham do
mesmo servidor da web.Navegadores normalmente tratam todos
arquivo:
URL como a
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
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 o
src
propriedade de a
<Cript>
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

buscado.(Claro, a primeira busca pode levar mais tempo do que qualquer um dos outros, então isso não é necessariamente mais rápido do que usar Promete.all ()
.)
Matrizes são iteráveis, para que possamos iterar através da variedade de promessas com um regular
para/de
laço:

```
para
(
const

promessa

de

promessas
)

{

resposta

=

aguarde

promessa
;

lidar
(
resposta
);
}
```

Este código de exemplo usa um regular
para/de
Faça um loop com um iterador regular.
Mas como esse iterador retorna promessas, também podemos usar o novo
para/aguardar
Para um código um pouco mais simples:

```
para

aguarde

(
const

resposta

de

promessas
)

{

lidar
(
```

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
Script de câmara cruzada
, 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>
deixar

nome

=

novo

Url
(
documento
.
Url
).
SearchParams
.
pegar
(
"nome"
);
documento
.
QuerySelector
(
'H1'
).
Innerhtml

=

"Olá "

+

nome
;
</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 no primeiro
<H1>
tag no documento.
Esta página pretende

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
ONLOAD
atributo
é executado.O global
alerta()
Função 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 consertar o
Greet.html
arquivo mostrado anteriormente substituindo
caracteres HTML especiais na sequência de entrada não confiável com seus
Entidades HTML equivalentes:

```
se

(
eu

===

j
)

{

se

(
j

===

k
)

{

console
.
registro
(
"Eu igual a k"
);

}
}

outro

{

// que diferença a localização de uma cinta encaracolada

faz!

console
.
registro
(
"Eu não igual a J"
);
}
```

Muitos programadores têm o hábito de envolver os corpos de
`se`
e
`outro`
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

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 do 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
<butter>
Elemento.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
2

Esse
A 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
invocação são explicadas 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 um
tipo
propriedade que
Especifica o tipo de evento e um
alvo
propriedade 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
tipo
e
alvo
- e não carregue muito outro útil
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
<a>
elemento que define isso
2

link.Então é disparado sobre os elementos contendo: talvez um
<p>
elemento, a
<Section>
elemento 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
, Assim,
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
associado 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
Lado do cliente
JavaScript 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 (ver
“Linha do tempo JavaScript do lado do cliente”
).Navegadores
Fire eventos “online” e “offline” no objeto da janela quando

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
()

.
então
(
displayTable
)

.
pegar
(
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 um
.pegar()
chamar:

```
Querydatabase
()

.
pegar
(
e

=>

espere
(
500
).
então
(
Querydatabase
))

// Sobre

fracasso, espere e tente novamente

.
então
(
displayTable
)

.
pegar
(
DisplayDatabaseError
);
```

Se as falhas hipotéticas forem verdadeiramente aleatórias, adicionando esta linha
do código deve reduzir sua taxa de erro de 1% para 0,01%.

seguido pelo nome do evento:
ONCLICK
, Assim,
OnChange
, Assim,
ONLOAD
, Assim,
OnMouseOver
, e assim por diante.Observe que esses nomes de propriedades são casos
sensível e são escritos em todas as minúsculas,
mesmo quando o tipo de evento
(como "Mousedown") consiste em várias palavras.O código a seguir
Inclui dois registros de manipuladores de eventos desse tipo:
// Defina a propriedade OnLar da Window para um

função.
// a função é o manipulador de eventos: é invocado quando o

Cargas de documentos.
janela
.
ONLOAD

=

função
()

{

// Procure um elemento <morm>

deixar

forma

=

documento
.
QuerySelector
(
"Formulário#frete"
);

// Registre uma função de manipulador de eventos no formulário que

será invocado

// Antes de o formulário ser enviado.Suponha que o isformValid () seja

definido em outro lugar.

forma
.
OnSubmit

=

diretamente no arquivo html como atributos no HTML correspondente
marcação.(Manipuladores que seriam registrados no elemento da janela com
JavaScript pode ser definido com atributos no
<Body>
tag in
Html.) Essa técnica geralmente é desaprovada na web moderna
desenvolvimento, mas é possível, e está documentado aqui porque você
Ainda pode vê -lo no código existente.
Ao definir um manipulador de eventos como um atributo html, o atributo
O valor deve ser uma sequência de código JavaScript.Esse código deve ser o
corpo
da função do manipulador de eventos, não uma declaração completa da função.
Isto é, seu código de manipulador de eventos HTML não deve ser cercado por
aparelho encaracolado e prefixado com o
função
palavra -chave.Por exemplo:
<botão

ONCLICK =
"Console.log ('obrigado');"
>
Por favor

Clique
</button>
Se um atributo de manipulador de eventos HTML contiver vários JavaScript
declarações, você deve se lembrar de separar essas declarações com
semicolons ou quebre o valor do atributo em várias linhas.
Quando você especifica uma sequência de código JavaScript como o valor de um html
Atributo do manipulador de eventos, o navegador converte sua string em uma função
que funciona algo assim:
função
(
evento
)

{

com
(
documento
)

{

com
(
esse
.
forma

||

{})

{

com
(

}
O
evento
argumento significa que seu código de manipulador pode se referir ao
objeto de evento atual como
evento
.O
com
declarações significam que o
O código do seu manipulador pode se referir às propriedades do objeto de destino, o
contendo
<morm>
(se houver) e o objeto de documento que contém
diretamente, como se fossem variáveis ■■no escopo.O

com
declaração é
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 pode 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
<butter>
elemento.Observe as diferenças entre as duas técnicas
usado:

```
<botão

id =
"MyButton"
>
Clique em mim
</button>
<Cript>
deixar

b

=

documento
.
QuerySelector
(
"#mybutton"
);
b
.
ONCLICK

=

função
()

{

console
.
registro
(
"Obrigado por clicar

meu!"
);

};
b
.
addEventListener
(
"clique"
, Assim,

()

=>

{

console
.
registro
(
"Obrigado
```

Você então desrespeitaria esses manipuladores quando o evento "MouseUp"
chega.Em tal situação, o código de remoção do seu manipulador de eventos pode
Parece isso:
documento
.
RemovaeventListener
(
"Mousemove"
, Assim,

HandleMousEmove
);
documento
.
RemovaeventListener
(
"MouseUp"
, Assim,

HandleMouseUp
);
O terceiro argumento opcional para
addEventListener ()
é um booleano
valor ou objeto.Se você passar
verdadeiro
, então sua função de manipulador é
registrado como um
captura
manipulador de eventos e é invocado em um diferente
Fase de Despacho de Eventos.Vamos abordar a captura de eventos em
§15.2.4
.Se você
passar um terceiro argumento de
verdadeiro
Quando você registra um ouvinte de evento,
Então você também deve passar
verdadeiro
Como o terceiro argumento para
RemoneeventListener ()
Se você deseja remover o manipulador.
Registrar um manipulador de eventos de captura é apenas uma das três opções
que
addEventListener ()
suporta e, em vez de passar um
Valor booleano único, você também pode passar um objeto que explicitamente
Especifica as opções que você deseja:
documento
.
addEventListener
(
"clique"
, Assim,

Handleclick
, Assim,

{

O ouvinte será removido automaticamente após o acionado uma vez.Se isso
propriedade é
falso
ou é omitido, então o manipulador nunca é
removido automaticamente.
Se o objeto de opções tiver um
passiva
propriedade definida como
verdadeiro
, indica
que o manipulador de eventos nunca ligará
PreventDefault ()
para cancelar
a ação padrão (veja
§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.Esse
passiva
Propriedade 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 o
passiva
propriedade para
falso
.
Você também pode passar um objeto de opções para
RemoneeventListener ()
, Assim,
mas o
capturar
A propriedade é a única que é relevante.Não há
precisa especificar
uma vez
ou
passiva
Ao remover um ouvinte, e
Essas propriedades são ignoradas.

15.2.3 Invocação do manipulador de eventos

Uma vez
Você registrou 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

detalhes, explicando argumentos de manipulador de eventos, o contexto de invocação (o esse
valor) e o significado do valor de retorno de um manipulador de eventos.
Argumento do manipulador de eventos
Os manipuladores de eventos são invocados com um objeto de evento como seu single argumento.As propriedades do objeto de evento fornecem detalhes sobre o evento:
tipo
O tipo do evento que ocorreu.
alvo
O objeto em que o evento ocorreu.
CurrentTarget
Para eventos que se propagam, esta propriedade é o objeto em que o
O manipulador de eventos atual foi registrado.
Timestamp
Um registro de data e hora (em milissegundos) que representa quando o evento ocorreu, mas isso não representa um tempo absoluto.Você pode determinar o tempo decorrido entre dois eventos subtraindo o Timestamp do primeiro evento do registro de data e hora do segundo.
istrado
Esta propriedade será
verdadeiro
Se o evento foi despachado pela web
o próprio navegador e
falso
Se o evento foi despachado por JavaScript
código.
Tipos específicos de eventos têm propriedades adicionais.Mouse e ponteiro

eventos, por exemplo, têm
ClientX
e
Cliente
propriedades isso
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:
alvo
.
ONCLICK

=

função
()

{

/ * 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, o
esse
palavra -chave refere -se ao objeto em
que o manipulador de eventos foi registrado.
Os manipuladores são invocados com o alvo como seu
esse
valor, mesmo quando
registrado usando
addEventListener ()
.Isso não funciona para
Manipuladores definidos como funções de seta, no entanto: Funções de seta sempre
tem o mesmo
esse
valor como o escopo em que eles 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
ONCLICK
manipulador de a
Enviar o botão em um formulário retorna
falso
, por exemplo, então 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

executar uma ação padrão é ligar para o
PreventDefault ()
método (
§15.2.5
) no objeto de evento.
Ordem de invocação
Um alvo de eventos pode ter mais de um manipulador de eventos registrado para um
tipo particular de evento.Quando ocorre um evento desse tipo, o navegador
Invoca todos os manipuladores na ordem em que foram registrados.
Curiosamente, isso é verdade mesmo se você misturar manipuladores de eventos registrados com
addEventListener ()
com um manipulador de eventos registrado em um
propriedade de objeto como
ONCLICK
.
15.2.4 Propagação de eventos
Quando
O alvo de um evento é o objeto da janela ou de algum outro
objeto independente, o navegador responde a um evento simplesmente invocando
os manipuladores apropriados nesse objeto.Quando o alvo do evento é um
Documento ou documento elemento, no entanto, a situação é mais
complicado.
Depois que os manipuladores de eventos registrados no elemento alvo são invocados,
A maioria dos eventos “borbulham” a árvore Dom.Os manipuladores de eventos do
Os pais da Target são invocados.Então os manipuladores registrados no alvo
Avós são invocados.Isso continua até o objeto do documento,
e depois além do objeto da janela.Evento Bubling fornece um
Alternativa ao registro de manipuladores em muitos documentos individuais
Elementos: em vez disso, você pode registrar um único manipulador em um comum
elemento ancestral e lide com eventos lá.Você pode registrar um
Manipulador de "Mudança" em um
<morm>
elemento, por exemplo, em vez de
Registrando um manipulador de "mudança" para todos os elementos no formulário.

A maioria dos eventos que ocorrem na bolha dos elementos do documento.Notável
Exceções são os eventos "Focus", "Blur" e "Roll".A "carga"
evento no documento elementos bolhas, mas para de borbulhar no
Documentar objeto e não se propaga para o
Janela
objeto.
(Os manipuladores de eventos "Carregar" do objeto da janela são acionados apenas
Quando o documento inteiro é carregado.)
O evento Bubling é a terceira "fase" da propagação de eventos.O
A invocação dos manipuladores de eventos do próprio objeto alvo é o segundo
fase.A primeira fase, que ocorre mesmo antes de os manipuladores de destino serem
Invocado, é chamado de fase de "captura".Lembre -se disso
addEventListener ()
leva um terceiro argumento opcional.Se isso
argumento é
verdadeiro
, ou
{Capture: True}
, então o manipulador de eventos é
registrado como um manipulador de eventos de captura para invocação durante este primeiro
fase da propagação de eventos.A fase de captura da propagação do evento
é como a fase borbulhante ao contrário.Os manipuladores de captura do
Objeto de janela é invocado primeiro, depois os manipuladores de captura do
Documente objeto, então do objeto corporal, e assim por diante no DOM
árvore até os manipuladores de eventos de captura do pai do alvo do evento
são invocados.Capturando os manipuladores de eventos registrados na meta de evento
por si só não são invocados.
Captura de eventos oferece uma oportunidade de espiar os eventos antes de eles
são entregues ao seu alvo.Um manipulador de eventos de captura pode ser usado para
Depuração, ou pode ser usado junto com o cancelamento do evento
técnica descrita na próxima seção para filtrar os eventos para que o alvo
Os manipuladores de eventos nunca são realmente invocados.Um uso comum para o evento
Capturar está lidando com arrastos de mouse, onde os eventos de movimento do mouse precisam
ser tratado pelo objeto que está sendo arrastado, não pelos elementos do documento

sobre o qual é arrastado.

15.2.5 Cancelamento de eventos

Navegadores
Responda a muitos eventos de usuário, mesmo que seu código não:
Quando o usuário clica no mouse em um hiperlink, o navegador segue o
link.Se um elemento de entrada de texto HTML tiver o foco do teclado e o
Tipo de usuário uma chave, o navegador inserirá a entrada do usuário.Se o usuário
Move o dedo através de um dispositivo de tela sensível ao toque, o navegador rola.Se
Você registra um manipulador de eventos para eventos como esses, você pode impedir o
navegador de executar sua ação padrão, invocando o
PreventDefault ()
Método do objeto de evento.(A menos que você
registrou o manipulador com o
passiva
opção, o que faz
PreventDefault ()
ineficaz.)
Cancelar a ação padrão associada a um evento é apenas um tipo
do cancelamento do evento.Também podemos cancelar a propagação de eventos por
chamando o
StopPropagation ()
Método do objeto de evento.Se
Existem outros manipuladores definidos no mesmo objeto, o resto daqueles
Os manipuladores ainda serão invocados, mas nenhum manipulador de eventos em qualquer outro objeto
será invocado depois
StopPropagation ()
é chamado.
StopPropagation ()
funciona durante a fase de captura, no
O próprio alvo de eventos e durante a fase borbulhante.
StopImediatePropagation ()
funciona como
StopPropagation ()
, mas também impede a invocação de qualquer
Os manipuladores de eventos subsequentes registrados no mesmo objeto.

15.2.6 Despacha eventos personalizados

Lado do cliente
A API de evento de JavaScript 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
addEventListener ()
método, então
é um "alvo de eventos", e isso significa que também tem um
DispatchEvent ()
método.Você pode criar seu próprio objeto de evento com o
Customevent ()
construtor e passa para
DispatchEvent ()
.
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 o
detalhe
propriedade disso
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: Verdadeiro
para
o segundo
argumento:
// Despacha um evento personalizado para que a interface do usuário saiba que estamos ocupados
documento
.
DispatchEvent
(
novo

Customevent
(
"ocupado"
, Assim,

{

detalhe
:

verdadeiro

}));
// Execute uma operação de rede

```
buscar
(
url
)

.
então
(
HandleNetworkResponse
)

.
pegar
(
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.

documento
.
DispatchEvent
(
novo

Customevent
(
"ocupado"
, Assim,

{

detalhe
:

falso

}));

});
// 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
```

O DOM foi introduzido em
§15.1.2
.Esta seção explica a API em
detalhe.Isto
capas:
Como consultar ou
Selecione
elementos individuais de um documento.
Como fazer
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
Lado do cliente
Os programas JavaScript geralmente precisam manipular um ou mais
elementos dentro do documento.O global
documento
Propriedade 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
Folhas de estilo CSS
ter uma sintaxe muito poderosa, conhecida como
Seletores
, para
descrevendo elementos ou conjuntos de elementos em um documento.O
Dom
Métodos
QuerySelector ()
e
QuerySelectorAll ()
permitir
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 de CSS podem descrever elementos por nome da tag, o valor de seus
eu ia
atribuir, ou as palavras em seus
aula
atributo:
div

//

Qualquer

<
div
>

elemento
#nav

//

O

elemento

com

eu ia
=
"Nav"
.
aviso

//

Qualquer

elemento

com

"aviso"

em

isso é

aula

atributo
O
#
O personagem é usado para corresponder com base no
eu ia
atributo e o
.
O personagem é usado para corresponder com base no
aula
atributo.Elementos podem
também ser selecionado com base em valores de atributo mais gerais:

Se dois seletores forem separados por vírgula, significa que selecionamos
Elementos que correspondem a um dos seletores:
botão
, Assim,

entrada
[[
tipo
=
"botão"
]

//

Todos

<
botão
>

e

<
entrada

tipo
=
"botão"
>

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
QuerySelector ()
Método leva um seletor CSS
string como seu argumento e retorna o primeiro elemento correspondente no
documentar que ele encontra ou retorna
nulo
Se nenhum corresponde:
// Encontre o elemento de documento para a tag html com atributo

id = "Spinner"
deixar

Spinner

=

documento
.
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

comprimento
Propriedade 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 nncom
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
O 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 correspondeu.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
Hiperlink
<a>
Tag em si.Seu manipulador de eventos pode procurar o
mais próximo contendo hiperlink assim:
// Encontre a etiqueta de gabinete mais próxima que tem um href

atributo.
deixar

Hiperlink

=

evento
.
alvo
.
mais próximo
(
"Um [href]"
);
Aqui está outra maneira de usar
mais próximo ()
:
// retorna true se o elemento e estiver dentro de uma lista HTML

elemento
função

insidelista
(
e
)

{

retornar

e
.
mais próximo
(
"Ul, Ol, DL"
)

! ==

nulo
;
}
O
método relacionado
partidas()
não retorna ancestrais ou
Descendentes: simplesmente testa se um elemento é correspondido por um CSS
seletor e retorno
verdadeiro
Se sim e

```
Document.QuerySelector ("#Sect1")
deixar

Seção1

=

documento
.
getElementById
(
"Sect1"
);
// Procure todos os elementos (como caixas de seleção de formulário) que têm um

nome = "cor"
// atributo.Semelhante ao document.QuerySelectorAll ('*
[name = "color"] ');
deixar

cores

=

documento
.
getElementsByName
(
"cor"
);
// Procure todos os elementos <H1> no documento.
// semelhante ao document.QuerySelectorall ("H1")
deixar

títulos

=

documento
.
getElementsByTagName
(
"H1"
);
// getElementsByTagName () também é definido em elementos.
// Obtenha todos os elementos <H2> dentro do elemento SECT1.
deixar

subtítulos

=

Seção1
.
getElementsByTagName
(
"H2"
);
// Procure todos os elementos que têm a classe "ToolTip".
```

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 o
document.forms
propriedade, para
exemplo, você pode acessar o
<form id = "endereço">
tag como:
documento
.
formas
.
endereço
;
Uma API ainda mais desatualizada para selecionar elementos é o
document.all
propriedade, que é como um htmlcollection para todos
elementos no documento.
document.all
está obsoleto e você
não deve mais usá -lo.
15.3.2 Estrutura de documentos e travessia
Uma vez
Você selecionou um elemento de um documento, às vezes
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

exclui crianças que não são de elementos como nós de texto (e comentários
nós).
ChildElementCount
O número de elementos crianças.Retorna o mesmo valor que
filhos
.
FirstElementChild
, Assim,
LastElementChild
Essas propriedades se referem ao primeiro e último elemento filhos de um
Elemento.Eles são
nulo
Se o elemento não tem elementos filhos.
NextElementsibling
, Assim,
anteriores dolementsibling
Essas propriedades se referem aos elementos irmãos imediatamente antes
ou imediatamente após um elemento, ou
nulo
Se não existe tal
irmão.
Usando essas propriedades do elemento, o segundo elemento filho do primeiro
O elemento filho do documento pode ser referido com qualquer um deles
Expressões:
documento
.
crianças
[[
0
].
crianças
[[
1
]
documento
.
FirstElementChild
.
FirstElementChild
.
NextElementsibli
ng
(Em um documento HTML padrão, ambas as expressões se referem ao
<Body>
tag do documento.)
Aqui estão duas funções que demonstram como você pode usá -las
Propriedades para fazer recursivamente uma travessia de profundidade de um documento
Invocando uma função especificada para cada elemento no documento:
// atravessar recursivamente o documento ou elemento e, invocando

```
a função
// f em e e em cada um de seus descendentes
função

atravessar
(
e
, Assim,

f
)

{

f
(
e
);

// Invocar f () em e

para
(
deixar

criança

de

e
.
crianças
)

{

// itera sobre o

crianças

atravessar
(
criança
, Assim,

f
);

// e recorrente a cada

um

}
}
função

Traverse2
(
e
```

Uma lista de nodel de somente leitura que contém todas as crianças (não apenas
Crianças de elemento) do nó.
FirstChild
, Assim,
LastChild
O primeiro e o último filho dos nós de um nó, ou
nulo
Se o nó não tiver
crianças.
próximo
, Assim,
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:
documento
.
Childnodes
[[
0
].
Childnodes
[[
1
]
documento
.
FirstChild
.
FirstChild
.
próximo
Suponha que o documento em questão seja o seguinte:
<html> <head> <title>
Teste
</ititle> </head> <body>
Olá mundo!
</body> </html>

Então o segundo filho do primeiro filho é o
<Body>
elemento.Tem um
NodeType
de 1 e um
Nodename
de "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 o
<html>
e o
<head>
tag, 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
<head>
elemento em vez do
<Body>
elemento.
Para demonstrar essa API Traversal baseada em nó, aqui está uma função que
Retorna todo o texto dentro de um elemento ou documento:

```
// retorna o conteúdo de texto simples do elemento e, recolocando-se em

elementos filhos.
// Este método funciona como a propriedade TextContent
função

TextContent
(
e
)

{

deixar

s

=

""
;

// Acumula o texto

aqui

para
(
deixar

criança

=

e
.
FirstChild
```

escrever
E.TextContent
Para obter o conteúdo textual do elemento
e
.
15.3.3 Atributos
Html
Os elementos consistem em um nome de tag e um conjunto de pares de nome/valor
conhecido como
atributos
.O
<a>
elemento que define um hiperlink, para
exemplo, usa o valor de seu
Href
atribuir como destino do
link.
A classe de elemento define general
getAttribute ()
, Assim,
setAttribute ()
, Assim,
hasattribute ()
, e
removeattribute ()
métodos para consulta, definição, teste e
removendo os atributos de um elemento.Mas os valores de atributo de
Elementos HTML (para todos os atributos padrão do HTML padrão
elementos) estão disponíveis como propriedades dos objetos HTMLELEMENT que
representar esses elementos, e geralmente é muito mais fácil trabalhar com
eles como propriedades JavaScript do que para ligar
getAttribute ()
e
Métodos relacionados.
Atributos HTML como propriedades do elemento
Os objetos do elemento que representam os elementos de um documento HTML
geralmente definindo propriedades de leitura/gravação que refletem os atributos HTML de
os elementos.Elemento define propriedades para o HTML universal
atributos como
eu ia
, Assim,
título
, Assim,
Lang
, e
dir
e manipulador de eventos
propriedades como
ONCLICK
.Subtipos específicos de elemento definem atributos
específico para esses elementos.Para consultar o URL de uma imagem, por exemplo,
você pode usar o
src
propriedade do htmlelement que representa o
<MIG>
elemento:

```
deixar

imagem

=

documento
.
QuerySelector
(
"#Main_Image"
);
deixar

url

=

imagem
.
src
;

// O atributo SRC é o URL de

a imagem
imagem
.
eu ia

===

"Main_image"

// => true;Nós procuramos a imagem

por id
```

Da mesma forma, você pode definir os atributos de submissão de forma de um <morm> Elemento com código como este:

```
deixar

f

=

documento
.
QuerySelector
(
"forma"
);

// primeiro <morm>

no documento
f
.
Ação
```

Alguns nomes de atributos HTML são palavras reservadas no JavaScript.Para
A regra geral é prefixar o nome da propriedade com "html".O
Html
para
atributo (do
<Boel>
elemento), por exemplo,
torna -se o JavaScript
htmlfor
propriedade."Classe" é uma palavra reservada
em JavaScript e o HTML muito importante
aula
atributo é um
exceção à regra: torna -se
ClassName
no código JavaScript.
As propriedades que representam atributos HTML geralmente têm string
valores.Mas quando o atributo é um valor booleano ou numérico (o
DefaultChecked
e
maxlength
atributos de um
<input>
elemento, por exemplo), as propriedades são booleanos ou números
de cordas.Os atributos do manipulador de eventos sempre têm funções (ou
nulo
) como
seus valores.
Observe que esta API baseada em propriedades para obter e definir atributo
Os valores não definem nenhuma maneira de remover um atributo de um elemento.
Em particular, o
excluir
O operador não pode ser usado para esse fim.Se
você precisa excluir um atributo, usar o
removeattribute ()
método.
O atributo de classe
O
aula
atributo de um elemento html é um particularmente importante
um.Seu valor é uma lista separada por espaço de classes CSS que se aplicam ao
elemento e afeta como é estilizado com CSS.Porque
aula
é a
Palavra reservada em JavaScript, o valor deste atributo está disponível
através do
ClassName
propriedade em objetos de elemento.O
ClassName
a propriedade pode definir e retornar o valor do
aula
atributo como uma string.Mas o
aula
atributo é mal nomeado: seu valor

Erro ao traduzir esta página.

No DOM, os objetos de elemento têm um
conjunto de dados
propriedade que se refere a
um objeto que possui propriedades que correspondem ao
dados-
atributos
com o prefixo removido.Por isso,
DataSet.x
guardaria o valor de
o
Data-x
atributo.Mapas de atributos hifenizados para camelcase
nomes de propriedades: o atributo
número de seção de dados
torna -se o
propriedade
DataSet.SectionNumber
.
Suponha que um documento HTML contenha este texto:
<h2

id =
"título"

Number de seção de dados =
"16.1"
>
Atributos
</h2>
Então você pode escrever JavaScript como este para acessar esse número de seção:
deixar

número

=

documento
.
QuerySelector
(
"#título"
).
conjunto de dados
.
SectionNumber
;
15.3.4 Conteúdo do elemento
Olhe novamente para a árvore de documentos retratada em
Figura 15-1
, e pergunte
você mesmo que o "conteúdo" do
<p>
elemento é.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

Conteúdo do elemento como html
Leitura
o
Innerhtml
propriedade 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:
documento
.
corpo
.
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 configuração
Innerhtml
geralmente é bastante eficiente.Observação,
No entanto, esse texto anexo ao
Innerhtml
propriedade 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.Ver
“Scripts de sites cruzados”
Para detalhes.
O
externo
propriedade de um elemento é como
Innerhtml
exceto
que seu valor inclui o próprio elemento.Quando você consulta
externo
, o valor inclui as tags de abertura e fechamento do
elemento.E quando você define
externo
em um elemento, o novo
O conteúdo substitui o próprio elemento.

Um método de elemento relacionado é
insertadjacenthtml ()
, qual
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 em
Figura 15-2
.
Figura 15-2.
Pontos de inserção para insertadjacenthtml ()
Conteúdo do elemento como texto simples
Às vezes você deseja consultar o conteúdo de um elemento como texto simples ou
para inserir texto simples em um documento (sem ter que escapar do ângulo
Suportes e ampeiros usados ■■na marcação HTML).A maneira padrão de
fazer isso é com o
TextContent

propriedade:
deixar

pára

=

documento
.
QuerySelector
(
"P"
);

// primeiro <p> no

documento
deixar

texto

=

pára
.
TextContent
;

// Obtenha o texto de

o parágrafo
pára
.
TextContent

=

"Olá mundo!"

A classe elemento define um
InnerText
propriedade 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>
Em linha

<Cript>
elementos (ou seja, aqueles que não têm um
src
atributo) tem um
texto
propriedade que você
pode usar para recuperar seu texto.O conteúdo de um
<Cript>
O elemento nunca é exibido pelo navegador,
e o analisador HTML ignora suportes de ângulo e amperantes dentro de um script.Isso faz um
<Cript>
Elemento Um local ideal para incorporar dados textuais arbitrários para uso pelo seu aplicativo.Simp
lesmente
defina o
tipo
atributo do elemento a 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 ignora
r
o script, mas o elemento existirá na árvore de documentos e seu
texto
a propriedade retornará os dados para
você.
15.3.5 Criação, inserção e exclusão de nós
Nós temos
visto 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.
Criar um novo elemento com o
createElement ()
Método do
Documentar a classe e anexar seqüências de texto ou outros elementos a ele com
isso é
acrescentar()
e
Apresença ()

Métodos:
deixar

parágrafo

=

```
deixar

ênfase

=

documento
.
CreateElement
(
"Em"
);

// Crie um

Elemento vazio <em>
ênfase
.
acrescentar
(
"Mundo"
);

// Adicionar texto a

o elemento <em>
parágrafo
.
acrescentar
(
"Olá "
, Assim,

ênfase
, Assim,

"!"
);

// Adicionar texto e

<em> para <p>
parágrafo
.
Preparar
(
"¡"
);

// Adicione mais texto

no início de <p>
parágrafo
.
Innerhtml

// => "¡Hello

<em> mundo </em>! "
```

o documento depois de converter strings em nós de texto.
acrescentar()
e
Apresença ()
são definidos apenas em objetos de elemento, mas
depois()
e
antes()
Trabalhe nos nós de elementos e de texto: você pode usá -los
para inserir o conteúdo em relação a um nó de texto.
Observe que os elementos só podem ser inseridos em um ponto no documento.Se
Um elemento já está no documento e você o insere em algum lugar
caso contrário, ele será movido para o novo local, não copiado:
// inserimos o parágrafo após esse elemento, mas agora nós
// mova -o para que apareça antes do elemento
saudações
.
antes
(
parágrafo
);
Se você deseja fazer uma cópia de um elemento, use o
CLONENODE ()
método, passagem
verdadeiro
Para copiar todo o seu conteúdo:
// Faça uma cópia do parágrafo e insira -a após o

elemento saudações
saudações
.
depois
(
parágrafo
.
CLONENODE
(
verdadeiro
));
Você pode remover um elemento ou nó de texto do documento por
chamando seu
remover()
método, ou você pode substituí -lo ligando
replywith ()
em vez de.
remover()
não leva argumentos e
replywith ()
leva qualquer número de cordas e elementos como
antes()
e
depois()
fazer:
// Remova o elemento Saudações do documento e substitua

com
// o elemento do parágrafo (movendo o parágrafo de seu

Localização atual

parágrafo
.
remover
();
A API DOM também define uma geração mais antiga de métodos para
inserção e remoção de conteúdo.
ApndendChild ()
, Assim,
insertBefore ()
, Assim,
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
shows
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
```

```
* Que o TOC pode vincular a eles.As âncoras geradas têm

nomes
* que começam com "Toc", então você deve evitar esse prefixo em

seu próprio
* Html.
*
* As entradas no TOC gerado podem ser estilizadas com CSS.Todos
* As entradas têm uma aula "ToCentry".As entradas também têm uma aula

que
* corresponde ao nível do cabeçalho da seção.Tags <H1>

gerar
* entradas da classe "TOCLEVEL1", <H2> Tags geram entradas de

aula
* "Toclevel2", e assim por diante.Números de seção inseridos em

títulos têm
* classe "TocSectnum".
*
* Você pode usar este script com uma folha de estilo como esta:
*
* #Toc {borda: preto sólido 1px;margem: 10px;preenchimento:

10px;}
* .Tocentry {margem: 5px 0px;}
* .Tocentry a {decoração de texto: nenhum;}
* .Toclevel1 {font-size: 16pt;Peso da fonte: negrito;}
* .Toclevel2 {font-size: 14pt;margem-esquerda: .25in;}
* .Toclevel3 {font-size: 12pt;margem-esquerda: .5in;}
* .TocSectnum: após {content: ":";}
*
* Para ocultar os números da seção, use isto:
*
* .TocSectNum {Display: Nenhum}
**/
documento
.
addEventListener
(
"DomContentLoaded"
, Assim,

()

=>

{

// Encontre o elemento de contêiner do TOC.

// Se não houver um, crie um no início do

documento.

deixar
```

documento são

// marcado com <H2> a <h6>.

deixar

títulos

=

documento
.
QuerySelectorAll
(
"H2, H3, H4, H5, H6"
);

// Inicialize uma matriz que acompanha os números de seção.

deixar

SectionNumbers

=

[[
0
, Assim,
0
, Assim,
0
, Assim,
0
, Assim,
0
];

// Agora passe através dos elementos do cabeçalho da seção que encontramos.

para
(
deixar

cabeçalho

de

títulos
)

{

// Pule o cabeçalho se estiver dentro do contêiner do TOC.

se

(
cabeçalho
.

```
cabeçalho
.
antes
(
âncora
);

// Insira a âncora antes

cabeçalho

âncora
.
acrescentar
(
cabeçalho
);

// e mova indo para dentro

âncora

// Agora crie um link para esta seção.

deixar

link

=

documento
.
CreateElement
(
"um"
);

link
.
Href

=

`#
$ {
FragmentName
}
`
;

// destino de link

// Copie o texto do cabeçalho no link.Este é um seguro

uso de

// innerhtml porque não estamos inserindo nenhum

cordas.
```

Estilos CSS que geralmente são roteirizados de
JavaScript:
Definindo o
mostrar
estilo para "nenhum" esconde um elemento.Você
Mais tarde pode mostrar o elemento configurando
mostrar
para outro
valor.
Você pode posicionar dinamicamente elementos definindo o
posição
estilo para "absoluto", "parente" ou "fixo" e depois
definindo o
principal
e
esquerda
Estilos para as coordenadas desejadas.
Isso é importante ao usar o JavaScript para exibir dinâmico
Conteúdo como diálogos modais e dicas de ferramentas.
Você pode mudar, escalar e girar elementos com o
transformar
estilo.
Você pode animar mudanças em outros estilos de CSS com o
transição
estilo.Essas animações são tratadas
automaticamente pelo navegador da web e não requer
JavaScript, mas você pode usar o JavaScript para iniciar o
animações.

15.4.1 Classes CSS

O
maneira mais simples de usar o JavaScript para afetar o estilo do documento
O conteúdo é adicionar e remover nomes de classe CSS do
aula
atributo das tags HTML.Isso é fácil de fazer com o
Lista de classe
propriedade de objetos de elemento, como explicado em
“O atributo de classe”
.
Suponha, por exemplo, que a folha de estilo do seu documento inclua um
Definição para uma classe "oculta":

```
.escondido

{

mostrar
:
nenhum
;
}
```

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:
documento
.
QuerySelector
(
"#Tooltip"
).
Lista de classe
.
remover
(
"escondido"
);
// e podemos esconder novamente assim:
documento
.
QuerySelector
(
"#Tooltip"
).
Lista de classe
.
adicionar
(
"escondido"
);
15.4.2 Estilos embutidos
Para
Continue 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
dica de ferramenta, então o
Lista de classe
A propriedade não nos ajudará
posicionamento.
Nesse caso, precisamos escrever o
estilo
atributo da dica de ferramenta
Elemento para definir estilos embutidos específicos para esse elemento.O
Dom define um
estilo
propriedade em todos os objetos de elemento que correspondem
para o
estilo
atributo.Ao contrário da maioria dessas propriedades, no entanto, o
estilo
A propriedade não é uma string.Em vez disso, é um
CSSSTYLEDECLARAÇÃO
Objeto: uma representação analisada dos estilos CSS que aparecem em textual
forma no
estilo
atributo.Para exibir e definir a posição de nosso
Distima de ferramentas hipotética com JavaScript, podemos usar código assim:

dip de ferramenta
.
estilo
.
principal

=

`
$ {
y
}
px`
;
}
Convenções de nomenclatura: Propriedades do CSS em JavaScript
Muitos
Propriedades do estilo CSS, como
tamanho de fonte
, contém 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 i
dentificadores.
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 propriedade CSS
largura de borda-esquerda
é acessado através
o javascript
largura de largura
propriedade, por exemplo, e o CSS
Fonte-família
propriedade 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
Folha de estilo ou
estilo
atributo, você pode escrever:
mostrar
:

bloquear
;

Fonte-família
:

Sans-Serif
;

cor de fundo
:

#ffffff
;

marginleft
Propriedade como esta:
e
.
estilo
.
marginleft

=

300
;

// incorreto: este é um número,

não é uma string
e
.
estilo
.
marginleft

=

"300"
;

// incorreto: as unidades são

ausente
São necessárias unidades ao definir propriedades de estilo em JavaScript, assim como
Eles estão definindo propriedades de estilo nas folhas de estilo.A maneira correta de
Defina o valor do
marginleft
propriedade de um elemento
e
a 300
Pixels é:
e
.
estilo
.
marginleft

=

"300px"
;
Se você deseja definir uma propriedade CSS como um valor calculado, certifique -se de
Anexe as unidades no final do cálculo:
e
.
estilo
.
esquerda

=

`

do objeto CSSSTYLEDECLARATION:
// Copie os estilos embutidos do elemento E para o elemento F:
f
.
setattribute
(
"estilo"
, Assim,

e
.
getAttribute
(
"estilo"
));
// ou faça assim:
f
.
estilo
.
CSSTEXT

=

e
.
estilo
.
CSSTEXT
;
Ao consultar o
estilo
propriedade 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 o
estilo
propriedade
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 quer o
estilo calculado
, Assim,
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":
deixar

título

=

documento
.
QuerySelector
(
"#Section1Title"
);
deixar

estilos

=

janela
.
getcomputedstyle
(
título
);
deixar

beforestyles

=

janela
.
getcomputedstyle
(
título
, Assim,

"::antes"
);
O valor de retorno de
getComputedStyle ()
é a
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
absoluto
: 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

O DOM foi introduzido em
§15.1.2
.Esta seção explica a API em
detalhe.Isto
capas:
Como consultar ou
Selecione
elementos individuais de um documento.
Como fazer
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
Lado do cliente
Os programas JavaScript geralmente precisam manipular um ou mais
elementos dentro do documento.O global
documento
Propriedade 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
Folhas de estilo CSS
ter uma sintaxe muito poderosa, conhecida como
Seletores
, para
descrevendo elementos ou conjuntos de elementos em um documento.O
Dom
Métodos
QuerySelector ()
e
QuerySelectorAll ()
permitir
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.

```
a função
// f em e e em cada um de seus descendentes
função

atravessar
(
e
, Assim,

f
)

{

f
(
e
);

// Invocar f () em e

para
(
deixar

criança

de

e
.
crianças
)

{

// itera sobre o

crianças

atravessar
(
criança
, Assim,

f
);

// e recorrente a cada

um

}
}
função

Traverse2
(
e
```

Então o segundo filho do primeiro filho é o
<Body>
elemento.Tem um
NodeType
de 1 e um
Nodename
de "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 o
<html>
e o
<head>
tag, 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
<head>
elemento em vez do
<Body>
elemento.
Para demonstrar essa API Traversal baseada em nó, aqui está uma função que
Retorna todo o texto dentro de um elemento ou documento:

```
// retorna o conteúdo de texto simples do elemento e, recolocando-se em

elementos filhos.
// Este método funciona como a propriedade TextContent
função

TextContent
(
e
)

{

deixar

s

=

""
;

// Acumula o texto

aqui

para
(
deixar

criança

=

e
.
FirstChild
```

Portanto, invisível.Mas se você aplicar o segundo estilo que diz ao
navegador que quando a opacidade do elemento muda, essa mudança
deve ser animado por um período de 0,5 segundos, "facilidade" especifica
que a animação de mudança de opacidade deve começar devagar e depois
acelerar.
Agora suponha que seu documento HTML contenha um elemento com o
Classe "Fadeable":

```
<div

id =
"Inscreva -se"

class =
"Notificação Fadeable"
>
...
</div>
```

Em JavaScript, você pode adicionar a classe "transparente":

```
documento
.
QuerySelector
(
"#SubScribe"
).
Lista de classe
.
adicionar
(
"Transpare
nt "
);
```

Este elemento é configurado para animar mudanças de opacidade.Adicionando o
Classe "transparente" muda a opacidade e desencadeia um animado: o
O navegador “desaparece” o elemento para que se torne totalmente transparente
durante o período de meio segundo.
Isso também funciona ao contrário: se você remover a classe "transparente" de
um elemento "desbotável", que também é uma mudança de opacidade, e o elemento
desaparece de volta e se torna visível novamente.
JavaScript não precisa fazer nenhum trabalho para fazer essas animações
Acontece: eles são um efeito CSS puro.Mas JavaScript pode ser usado para
acionar -os.
JavaScript também pode ser usado para monitorar o progresso de uma transição CSS
Porque o navegador da web dispara eventos no início e no final de um

transição.O evento "TransitionRun" é despachado quando a transição
é desencadeado pela primeira vez.Isso pode acontecer antes do início de quaisquer mudanças visuais,
Quando o
Dellay de transição
estilo foi especificado.Uma vez o
As mudanças visuais começam um evento de “transição” é despachado e quando
A animação está completa, um evento "transitório" é despachado.O
O alvo de todos esses eventos é o elemento que está sendo animado, é claro.O
O objeto de evento passou para os manipuladores para esses eventos é uma transiçãoEvent
objeto.Tem um
PropertyName
propriedade que especifica o CSS
propriedade sendo animada e um
Tempo decorrido
propriedade que para
Eventos "transiceend" especificam quantos segundos se passaram desde
O evento "Transitionstart".
Além das transições, o CSS também suporta uma forma mais complexa de
Animação conhecida simplesmente como "animações do CSS".Estes usam CSS
propriedades como
nome da animação
e
duração da animação
e um especial
@KeyFrames
regra para definir detalhes de animação.Detalhes de
Como as animações do CSS funcionam além do escopo deste livro, mas uma vez
Novamente, se você definir todas as propriedades de animação em uma classe CSS, então
Você pode usar o JavaScript para acionar a animação simplesmente adicionando o
classe para o elemento que deve ser animado.
E, como as transições de CSS, as animações do CSS também desencadeiam eventos que seu
O código JavaScript pode ouvir o formulário."Animationstart" é despachado quando
a animação começa e o "Animationend" é despachado quando é
completo.Se a animação repete mais de uma vez, então um
O evento “animationiteration” é despachado após cada repetição, exceto o
durar.O alvo do evento é o elemento animado e o objeto de evento
Passado para o Handler Functions é um objeto AnimationEvent.Esses eventos

inclua um
AnimationName
propriedade que especifica o
nome da animação
propriedade que define a animação e um
Tempo decorrido
propriedade que especifica quantos segundos passaram
Desde que a animação começou.
15.5 Geometria de documentos e rolagem
Em
Este 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 abstrato, baseado em árvores
modelo
de um documento e o
geométrico, baseado em coordenadas
visualizar
do documento como é estabelecido em um
Janela do navegador.
15.5.1 Documentar coordenadas e viewport
Coordenadas
O
A posição de um elemento de documento é medida em pixels CSS, com
o
x
coordenar o aumento da direita e o
y
coordenar o aumento

Quando descemos.Existem dois pontos diferentes que podemos usar como o
Origem do sistema de coordenadas, no entanto: o
x
e
y
coordenadas de um
o elemento pode ser relativo ao canto superior esquerdo do documento ou
em relação ao canto superior esquerdo de
o
viewport
em que o documento está
exibido.Nas janelas e guias de nível superior, a "viewport" é a parte
do navegador que realmente exibe o conteúdo do documento: ele exclui
Navegador “Chrome”, como menus, barras de ferramentas e guias.Para documentos
exibido em
<frame>
tags, é o elemento iframe no dom que
Define a viewport para o documento aninhado.Em ambos os casos, quando nós
Fale sobre a posição de um elemento, devemos deixar claro se estamos
usando coordenadas de documentos ou coordenadas de viewport.(Observe que
As coordenadas de viewport às vezes são chamadas de "coordenadas de janelas".)
Se o documento for menor que a visualização, ou se não foi
Rolado, o canto superior esquerdo do documento está no canto superior esquerdo
canto da viewport e o documento e coordenada de viewport
Os sistemas são iguais.Em geral, no entanto, para converter entre os dois
Coordenar sistemas, devemos adicionar ou subtrair o

Role compensações
.
Se um
elemento tem um
y
coordenada de 200 pixels em coordenadas de documentos, para
exemplo, e se o usuário rolou para baixo por 75 pixels, então isso
elemento tem um
y
Coordenada de 125 pixels nas coordenadas de viewport.
Da mesma forma, se um elemento tiver um
x
Coordenada de 400 na viewport
Coordena depois que o usuário rolou os 200 pixels da Viewport 200
horizontalmente, então o elemento
x
coordenar em coordenadas de documentos
é 600.
Se usarmos o modelo mental de documentos de papel impresso, é lógico para
Suponha que todo elemento em um documento deve ter uma posição única
nas coordenadas de documentos, independentemente de quanto o usuário rolou

o documento.Essa é uma propriedade atraente de documentos em papel, e isso
se aplica a documentos da Web simples, mas em geral, documento
As coordenadas realmente não funcionam na web.O problema é que o CSS
transbordamento
a propriedade permite que elementos em um documento conterão
mais conteúdo do que pode ser exibido.Elementos podem ter seus próprios
Barras de rolagem e sirva como viewports para o conteúdo que eles contêm.O fato
que a web permite rolagem de elementos em um documento de rolagem
significa que simplesmente não é possível descrever a posição de um
elemento dentro do documento usando um único ponto (x, y).
Porque
Coordenadas de documentos realmente não funcionam, JavaScript do lado do cliente
tende a usar coordenadas de viewport.O
getBoundingClientRect ()
e
ElementFroMpoint ()
métodos descritos em seguida, use coordenadas de viewport, por exemplo, e o
ClientX
e
Cliente
Propriedades do evento de mouse e ponteiro
Objetos também usam esta coordenada
sistema.
Quando você posiciona explicitamente um elemento usando CSS
Posição: corrigido
, o
principal
e
esquerda
propriedades são interpretadas em
Coordenadas de viewport.Se você usa
Posição: relativa
, o elemento
está posicionado em relação a onde teria sido se não tivesse o
posição
conjunto de propriedades.Se você usa
Posição: Absoluto
, então
principal
e
esquerda
são relativos ao documento ou ao mais próximo que contém
elemento posicionado.Isso significa, por exemplo, que um absolutamente
elemento posicionado dentro de um elemento relativamente posicionado está posicionado
em relação ao elemento do contêiner, não em relação ao documento geral.
Às vezes é muito útil criar um recipiente relativamente posicionado
com
principal
e
esquerda
definido como 0 (assim o contêiner é estabelecido normalmente) em

para estabelecer uma nova origem do sistema de coordenadas para o absolutamente
elementos posicionados que ele contém.Podemos nos referir a esta nova coordenada
sistema como "coordenadas de contêiner" para distingui -lo do documento
coordenadas e coordenadas de viewport.
CSS Pixels
Se, como eu, você
têm idade suficiente para lembrar monitores de computador com resoluções de 1024 × 768 e
Telefones de tela sensível ao toque com resoluções de 320 × 480, então você ainda pode pensar que a
palavra "pixel" refere-se
para um único "elemento de imagem" em
hardware
.Os monitores 4K de hoje e os displays de "retina" têm tão alto
Resolução de que os pixels de software foram dissociados de pixels de hardware.Um pixel CSS - e
Portanto, um pixel JavaScript do lado do cliente-pode de fato consistir em vários pixels de disposit
ivo.O
DevicePixelratio
propriedade do objeto de janela especifica quantos pixels de dispositivo são usados ■■para
cada software pixel.Um "DPR" de 2, por exemplo, significa que cada pixel de software é na verdade um
a grade 2 × 2
de pixels de hardware.O
DevicePixelratio
o valor depende da resolução física de seu
Hardware, nas configurações do seu sistema operacional e no nível de zoom no seu navegador.
DevicePixelratio
não precisa ser um número inteiro.Se você estiver usando um tamanho de fonte CSS de "12px" e
A relação pixel do dispositivo é de 2,5, depois o tamanho da fonte real, em pixels de dispositivo, é
30. Porque os valores de pixels
Usamos no CSS não correspondem mais diretamente a pixels individuais na tela, coordenadas de pixels
não
mais precisam ser inteiros.Se o
DevicePixelratio
é 3, então uma coordenada de 3,33 faz perfeita
senso.E se a proporção for realmente 2, uma coordenada de 3,33 será apenas arredondada para 3,5.
15.5.2 Consultando a geometria de um elemento
Você
pode determinar o tamanho (incluindo a fronteira e o preenchimento do CSS, mas não
a margem) e a posição (nas coordenadas da viewport) de um elemento por
chamando seu
getBoundingClientRect ()
método.É preciso não
Argumentos e retorna um objeto com propriedades
esquerda
, Assim,
certo
, Assim,
principal
, Assim,
fundo
, Assim,
largura
, e
altura
.O
esquerda
e
principal
propriedades dão
o
x

Elementos de bloqueio, como imagens, parágrafos e

elementos são
Sempre retangular quando estabelecido pelo navegador.Elementos embutidos, tal
como
<pan>
, Assim,
<Code>
, e
<b>
elementos, no entanto, podem abranger múltiplos
linhas e, portanto, consistem em múltiplos retângulos.Imagine, para
exemplo, algum texto dentro

e

tags que são
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
getClientRects ()
Método para obter um objeto de matriz somente para leitura
cujos elementos são objetos retangulares como os devolvidos por
getBoundingClientRect ()
.
15.5.3 Determinando o elemento em um ponto
O

getBoundingClientRect ()
o método 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
ElementFroMpoint ()
método do objeto de documento.Chame isso
Método com o
x
e
y
coordenadas de um ponto (usando a viewport
coordenadas, não coordenadas de documentos: o
ClientX
e
Cliente
coordenadas de um trabalho de evento do mouse, por exemplo).
ElementFroMpoint ()
retorna um objeto de elemento que está no
posição especificada.O
acertar a detecção
Algoritmo 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 mais alto (CSS mais alto
Z-

índice
elemento) elemento nesse ponto.
15.5.4 Rolagem
O

scrollto ()
Método do objeto de 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.
deixar

DocumentHeight

=

documento
.
DocumentElement
.
OffsetHeight
;
deixar

ViewPorty

=

janela
.
InnerHeight
;
// e role para que a última "página" seja exibida na viewport
janela
.
Scrollto
(
0
, Assim,

DocumentHeight

-

ViewPorty
);
```

O
scrollby ()
O método da janela é semelhante a
scrollto ()

principal
:

DocumentHeight

-

ViewPorty
, Assim,

comportamento
:

"suave"
});
Freqüentemente, em vez de rolar para um local numérico em um documento, nós apenas
Deseja rolar para que um certo elemento no documento seja visível.Você
pode fazer isso com o
scrollIntoView ()
Método no desejado
Elemento HTML.Este método garante que o elemento em que é
Invocado é visível na visualização.Por padrão, ele tenta colocar o topo
borda do elemento na parte superior ou perto da viewport.Se
falso
é
Passado como o único argumento, ele tenta colocar a borda inferior do
Elemento na parte inferior da viewport.O navegador também rolará o
Viewport horizontalmente conforme necessário para tornar o elemento visível.
Você também pode passar um objeto para
scrollIntoView ()
, definindo o
Comportamento: "Smooth"
Propriedade para rolagem suave.Você pode definir
o
bloquear
propriedade para especificar onde o elemento deve ser posicionado
verticalmente e o
em linha
propriedade para especificar como deve ser
Posicionado horizontalmente se for necessário rolagem horizontal.Valores legais
Para ambas as propriedades são
começar
, Assim,
fim
, Assim,
mais próximo
, e
centro
.
15.5.5 Tamanho, tamanho do conteúdo e rolagem da janela de exibição
Posição
Como
Discutimos, as janelas do navegador e outros elementos html podem
Exibir conteúdo de rolagem.Quando esse é o caso, às vezes precisamos
Saiba o tamanho da viewport, o tamanho do conteúdo e o rolagem
compensações do conteúdo dentro da vieta.Esta seção cobre estes
detalhes.

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 na sua
<head>
Para definir o desejado
largura da viewport para a página.) O tamanho total do documento é o
o mesmo que o tamanho do
<html>
elemento,
document.documentElement
.Você pode ligar
getBoundingClientRect ()
sobre
document.documentElement
para obter a largura e a altura do
documento, ou você pode usar o
OffsetWidth
e
OffsetHeight
propriedades de
document.documentElement
.Os compensações de rolagem de
o documento dentro de sua viewport está disponível como
window.scrollx
e
Window.Scrolly
.Essas são propriedades somente de leitura, então você não pode
Defina -os para rolar o documento: Use
window.scrollto ()
em vez de.
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
O
OffsetWidth
e
OffsetHeight
propriedades 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
Offsettop
Propriedades retornam o
x
e
y
coordenadas do elemento.
Para muitos elementos, esses valores são coordenadas de documentos.Mas para

Como células da tabela, essas propriedades retornam coordenadas que são relativas a um
Elemento ancestral em vez do próprio documento.O
OffsetParent
Propriedade 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
cliente
e
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>
, Assim,
<Code>
, e
<pan>
, todos eles retornam 0.
Lwidth Scroll
e
ScrolHeight
Retorne 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
ClientWidth
e
ClientHeight
.Mas quando lá
está transbordando, eles incluem o conteúdo transbordante e os valores de retorno
maior que
ClientWidth
e
ClientHeight
.
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,
Objetos de elemento também têm
scrollto ()
e

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

do
<input>
elemento, 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 em
Figura 15-3
.
15.6.1 Usando componentes da Web
Web
Os componentes são definidos em JavaScript, portanto, 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
tipo =
"módulo"

src =
"Componentes/pesquisa-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

As funções próprias do construtor para inicializar objetos recém -criados.Fazendo isso está coberto em
Capítulo 9
.

6.2.3 Protótipos

Antes
Podemos 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 a
protótipo
, e o primeiro objeto herda as propriedades do
protótipo.
Todos
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 o
novo
palavra -chave e a
Invocação do construtor Use o valor do
protótipo
propriedade de
o construtor funciona como seu protótipo.Então o objeto criado por
novo objeto ()
herda de
Object.prototype
, assim como o
objeto criado por
{}
faz.Da mesma forma, o objeto criado por
novo
Variedade()
usos
Array.prototype
como seu protótipo e o objeto
criado por
nova data ()
usos
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 um
protótipo
propriedade.São esses objetos com
protótipo
propriedades que definem o
protótipos
para todo o outro
objetos.
Object.prototype
é um dos objetos raros que não tem protótipo:
Não herda nenhuma propriedade.Outros protótipos objetos são normais objetos que têm um protótipo.A maioria dos construtores embutidos (e a maioria

diretamente no arquivo html como atributos no HTML correspondente
marcação.(Manipuladores que seriam registrados no elemento da janela com
JavaScript pode ser definido com atributos no
<Body>
tag in
Html.) Essa técnica geralmente é desaprovada na web moderna
desenvolvimento, mas é possível, e está documentado aqui porque você
Ainda pode vê -lo no código existente.
Ao definir um manipulador de eventos como um atributo html, o atributo
O valor deve ser uma sequência de código JavaScript.Esse código deve ser o
corpo
da função do manipulador de eventos, não uma declaração completa da função.
Isto é, seu código de manipulador de eventos HTML não deve ser cercado por
aparelho encaracolado e prefixado com o
função
palavra -chave.Por exemplo:
<botão

ONCLICK =
"Console.log ('obrigado');"
>
Por favor

Clique
</button>
Se um atributo de manipulador de eventos HTML contiver vários JavaScript
declarações, você deve se lembrar de separar essas declarações com
semicolons ou quebre o valor do atributo em várias linhas.
Quando você especifica uma sequência de código JavaScript como o valor de um html
Atributo do manipulador de eventos, o navegador converte sua string em uma função
que funciona algo assim:
função
(
evento
)

{

com
(
documento
)

{

com
(
esse
.
forma

||

{})

{

com
(

largura
:

300px
;

altura
:

50px
;
}
Como elementos HTML regulares, os componentes da web podem ser usados ■■em
JavaScript.Se você incluir um
<search-box>
Tag em sua página da web,
Então você pode obter uma referência a ele
QuerySelector ()
e um
Seletor CSS apropriado, assim como faria para qualquer outra tag HTML.
Geralmente, só faz sentido fazer isso após o módulo que define
O componente foi executado, então tenha cuidado ao consultar componentes da web
que você não o faz muito cedo.Implementações de componentes da web
normalmente (mas isso não é um requisito) define uma propriedade JavaScript para
Cada atributo html que eles suportam.E, como elementos html, eles
Também pode definir métodos úteis.Mais uma vez, a documentação para o
O componente da web que você está usando deve especificar quais propriedades e
Os métodos estão disponíveis para o seu código JavaScript.
Agora que você sabe usar componentes da web, os próximos três
As seções cobrem os três recursos do navegador da web que nos permitem
implemente -os.
Nós de documentário
Antes
Podemos cobrir as APIs de componentes da web, precisamos retornar brevemente à API DOM para explicar
o que um
DocumentFragment é.A API DOM organiza um documento em uma árvore de objetos de nó, onde um
O nó pode ser um documento, um elemento, um nó de texto ou até mesmo um comentário.Nenhum desses tip
os de nó
permite que você represente um fragmento de um documento que consiste em um conjunto de nós irmãos s
em o seu
pai.É aqui que entra o documento: é outro tipo de nó que serve como um
Pai temporário quando você deseja manipular um grupo de nós irmãos como uma única unidade.Você pode
Crie um nó de documentário com
document.createDocumentFragment ()
.Depois de ter
Um documento de frading, você pode usá -lo como um elemento e
acrescentar()
conteúdo para isso.UM
O DocumentFragment é diferente de um elemento porque não possui um pai.Mas mais
É importante ressaltar que, quando você insere um nó de documentação no documento, o documentFragmen
t
por si só não é inserida.Em vez disso, todos os seus filhos são inseridos.

15.6.2 Modelos HTML
O
Html
<Sodemplate>
A tag está apenas vagamente relacionada à web
componentes, mas permite uma otimização útil para componentes
que aparecem frequentemente em páginas da web.
<Sodemplate>
tags e suas
As crianças nunca são renderizadas por um navegador da web e só são úteis em
Páginas da Web que usam JavaScript.A ideia por trás dessa tag é que quando um
A página da web contém várias repetições do mesmo HTML básico
estrutura (como linhas em uma tabela ou a implementação interna de um
componente da web), então podemos usar um
<Sodemplate>
Para definir isso
estrutura de elementos uma vez e use JavaScript para duplicar a estrutura como
muitas vezes conforme necessário.
Em JavaScript, a
<Sodemplate>
A tag é representada por um
Objeto htmlTemplateElement.Este objeto define um único
contente
Propriedade e o valor desta propriedade é um documentário de todos
os nós filhos do
<Sodemplate>
.Você pode clonar isso
Documentfragment e depois insira a cópia clonada em seu
documento conforme necessário.O próprio fragmento não será inserido, mas seu
as crianças serão.Suponha que você esteja trabalhando com um documento que
inclui a
<tabela>
e
<modelo id = "linha">
tag e que o
O modelo define a estrutura das linhas para essa tabela.Você pode usar o
modelo como este:
deixar

corpo de mesa

=

documento
.
QuerySelector
(
"Tody"
);
deixar

modelo

=

documento
.
QuerySelector
(

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 fazer
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
demonstra 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
CustomElements.Define ()
Método 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 passou para
CustomElements.Define ()
deve se estender
Htmlelement e não um tipo mais específico como
HtmlbuttonElement.
Lembrar de
Capítulo 9
que quando um javascript
Classe estende outra classe, a função do construtor deve chamar
super()
antes de usar o
esse
palavra -chave, então se o elemento personalizado
A classe tem um construtor, deve ligar
super()
(sem argumentos)
antes de fazer qualquer outra coisa.

Erro ao traduzir esta página.

O analisador HTML instancia mais duas bolinhas:

<Círculo em linha

diâmetro =
"1.2em"

cor =
"azul"
> </inline-
círculo>

<Círculo em linha

diâmetro =
".6em"

cor =
"ouro"
> </inline-
círculo>
.
Quantas bolas de gude o documento contém agora?
</p>
Figura 15-4.
Um elemento personalizado em círculo embutido
Podemos implementar isso
<Circle inline>
elemento personalizado com o
código mostrado em
Exemplo 15-2
:
Exemplo 15-2.
O elemento personalizado <circle>
CustomElements
.
definir
(
"Círculo em linha"
, Assim,

aula

Inllinecircle

estende -se

HTMLELEMENT

{

// O navegador chama esse método quando um <circle inline>

elemento

// é inserido no documento.Há também um

desconectedCallback ()

```
// Se ainda não houver um tamanho definido, defina um

tamanho padrão

// que é baseado no tamanho da fonte atual.

se

(
!
esse
.
estilo
.
largura
)

{

esse
.
estilo
.
largura

=

"0.8em"
;

esse
.
estilo
.
altura

=

"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

pegar
```

que atualiza

// Os estilos de elemento.

pegar

diâmetro
()

{

retornar

esse
.
getAttribute
(
"diâmetro"
);

}

definir

diâmetro
(
diâmetro
)

{

esse
.
setattribute
(
"diâmetro"
, Assim,

diâmetro
);

}

pegar

cor
()

{

retornar

esse
.
getAttribute
(
"cor"
);

matriz do elemento hospedeiro e não são visitados por DOM normal
métodos de travessia, 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
<udio>
e
<Video>
Elementos: 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 um
Shadowroot
propriedade que JavaScript pode usar para ganhar

acesso aos elementos da raiz das sombras, se tiver alguns
razão para fazer isso.
Os estilos definidos sob uma raiz das sombras são privados para aquela árvore
e nunca afetará os elementos de DOM leve do lado de fora.
(Uma raiz de sombra pode definir estilos padrão para seu elemento host,
mas estes serão substituídos pelos estilos de Dom Light.) Da mesma forma,
Os estilos de Dom Light que se aplicam ao elemento do host das sombras
Não tenha efeito sobre os descendentes da raiz das sombras.
Elementos na sombra Dom herdarão coisas como o tamanho da fonte
e cor de fundo do DOM leve, e estilos no
Shadow Dom pode optar por usar variáveis ■■CSS definidas no
DOM leve.Na maioria das vezes, no entanto, os estilos da luz
Dom e os estilos da sombra dom estão completamente
independente: o autor de um componente da web e o usuário de um
O componente da web não precisa se preocupar com colisões ou
conflitos entre suas folhas de estilo.Ser capaz de "escopo" CSS
Dessa forma, talvez seja a característica mais importante do
Shadow Dom.
Alguns eventos (como "carga") que ocorrem dentro da sombra DOM
estão confinados à sombra DOM.Outros, incluindo foco,
Eventos de mouse e o teclado borbulham para cima e para fora.Quando um
Evento que se origina na sombra Dom atravessa o limite
e começa a se propagar na luz Dom, seu
alvo
A propriedade é alterada para o elemento de host das sombras, então parece
ter se originado diretamente nisso
elemento.
Slots Dom Shadow e Crianças Luz Dom
Um elemento HTML que é um hospedeiro sombra tem duas árvores de descendentes.
Um é o
crianças[]
Array - os descendentes regulares de luz da luz
do elemento hospedeiro - e o outro é a raiz das sombras e todos os seus
Descendentes, e você pode estar se perguntando como duas árvores de conteúdo distintas
pode ser exibido no mesmo elemento host.Aqui está como funciona:

Os descendentes da raiz das sombras são sempre exibidos
dentro do host das sombras.
Se esses descendentes incluem um
<Slot>
elemento, então o
Crianças de DOM da luz comum do elemento host são exibidas
Como se fossem filhos disso
<Slot>
, substituindo qualquer
Shadow DOM Conteúdo no slot.Se a Shadow Dom não
inclua um
<Slot>
, então qualquer conteúdo de DOM leve do host é
nunca exibido.Se a sombra Dom tiver um
<Slot>
, mas o
SHARGH HOST não tem filhos leves dom, então a sombra
O conteúdo DOM do slot é exibido como padrão.
Quando o conteúdo de DOM da luz é exibido dentro de uma sombra DOM
Slot, dizemos que esses elementos foram "distribuídos", mas
é importante entender que os elementos não realmente
Torne -se parte da sombra dom.Eles ainda podem ser consultados
com
QuerySelector ()
, e eles ainda aparecem na luz
Dom como crianças ou descendentes do elemento hospedeiro.
Se o Shadow Dom define mais de um
<Slot>
e
nomeia aqueles slots com um
nome
atribuir, então filhos do
Host Shadow pode especificar qual slot eles gostariam de aparecer
em especificar um
slot = "SlotName"
atributo.Vimos um
Exemplo desse uso em
§15.6.1
Quando demonstramos como
Para personalizar os ícones exibidos pelo
<search-box>
componente.
Shadow Dom API
Por todo o seu poder, o Shadow Dom não tem muito de um
API JavaScript.Para transformar um elemento DOM leve em um host das sombras, apenas
chame seu
Aplicações ()
método, passagem
{mode: "Open"}
Como o
apenas argumento.Este método retorna um objeto raiz sombra e também define
esse objeto como o valor do host do host
Shadowroot
propriedade.O

O objeto raiz de sombra é um documental e você pode usar o DOM
métodos para adicionar conteúdo a ele ou apenas definir seu
Innerhtml
propriedade para um
string de html.
Se o seu componente da web precisar saber quando o conteúdo da luz DOM de
uma sombra dom
<Slot>
mudou, pode registrar um ouvinte para
Eventos "slotchanged" diretamente no
<Slot>
elemento.
15.6.5 Exemplo: um componente da web <search-box>
Figura 15-3
ilustrado
um
<search-box>
Componente da Web.
Exemplo 15-3
demonstra as três tecnologias de habilitação que definem
componentes da web: ele implementa o
<search-box>
componente como a
elemento personalizado que usa um
<Sodemplate>
tag para eficiência e um
raiz de sombra para encapsulamento.
Este exemplo mostra como usar as APIs de componentes da Web de baixo nível
diretamente.Na prática, muitos componentes da web desenvolvidos hoje criam
eles usando bibliotecas de nível superior, como "elemento iluminado".Um dos
Razões para usar uma biblioteca é que a criação de reutilizáveis ■■e personalizáveis
Componentes é realmente muito difícil de fazer bem, e há muitos detalhes
para acertar.
Exemplo 15-3
demonstra componentes da web e faz
algum manuseio básico de foco do teclado, mas de outra forma ignora
acessibilidade e não faz não tentar usar atributos adequados de aria para
Faça o componente funcionar com leitores de tela e outros assistentes
tecnologia.
Exemplo 15-3.
Implementando um componente da Web

```
/**
* Esta classe define um elemento html <search-box> personalizado que
```

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

```
* Manipuladores para os eventos "Pesquisa" e "Clear" só podem ser

registrado com
* addEventListener ().
*/
aula

Pesquisa de pesquisa

estende -se

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.

esse
.
Aplicações
({
modo
:

"abrir"
});

// clonar o modelo que define os descendentes e

folha de estilo para

// este componente personalizado e anexar esse conteúdo a

a raiz da sombra.

esse
.
Shadowroot
.
acrescentar
(
Pesquisa de pesquisa
.
```

```
uma "pesquisa"

// evento.Também o desencadeia se o campo de entrada disparar um

"mudar"

// evento.(O evento "Mudança" não borbulha

a sombra dom.)

esquerda
.
ONCLICK

=

esse
.
entrada
.
OnChange

=

(
evento
)

=>

{

evento
.
StopPropagation
();

// Prevendo eventos de clique

de borbulhar

se

(
esse
.
desabilitado
)

retornar
;

// não faz nada quando

desabilitado

esse
.
DispatchEvent
```

matriz do elemento hospedeiro e não são visitados por DOM normal
métodos de travessia, 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
<udio>
e
<Video>
Elementos: 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 um
Shadowroot
propriedade que JavaScript pode usar para ganhar

```
}
}
// este campo estático é necessário para o

Método AttributeChangedCallback.
// apenas atributos nomeados nesta matriz desencadearão chamadas para

Esse método.
Pesquisa de pesquisa
.
observoutattributes

=

[[
"desabilitado"
, Assim,

"espaço reservado"
, Assim,

"tamanho"
, Assim,

"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.
Pesquisa de pesquisa
.
modelo

=

documento
.
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.
Pesquisa de pesquisa
.
modelo
.
Innerhtml

=
```

```
}
: host ([focado]) { /* Quando o host tem o foco

atributo ... */
Box-Shadow: 0 0 2px 2px #6AE;/* Exiba este foco falso

anel.*/
}
/* O restante da folha de estilo se aplica apenas a elementos no

Shadow Dom.*/
entrada {
largura de fronteira: 0;/* Esconda a borda do interno

campo de entrada.*/
Esboço: Nenhum;/* Esconda o anel de foco também.*/
Fonte: herdar;/* <sput> elementos não herdam

Fonte por padrão */
Antecedentes: herdar;/* O mesmo para a cor de fundo.*/
}
slot {
Cursor: padrão;/* Um cursor de ponteiro de seta sobre o

botões */
Seleção de usuário: Nenhum;/* Não deixe o usuário selecionar o

texto emoji */
}
</style>
<div>
<slot name = "esquerda">
\
u {1f50d} </slot> <!- u+1f50d é um

lupa ->
<input type = "text" id = "input" /> <!- a entrada real

elemento ->
<slot name = "direita">
\
u {2573} </slot> <!-u+2573 é um x->
</div>
`
;
// Finalmente, chamamos o CUNDARELEMENT.DEFINE () para registrar o

Elemento da caixa de pesquisa
// como a implementação da tag <search-box>.Personalizado

elementos são necessários
// para ter um nome de tag que contém um hífen.
CustomElements
.
definir
(
"Caixa de pesquisa"
, Assim,
```

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
.svg
Arquivos de imagem com HTML regular
<MIG>
tags,
Assim como você usaria um
.png
ou
.jpeg
imagem.
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

Svg
As imagens podem, é claro, ser exibidas usando HTML
<MIG>
tags.
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>
<título>
Relógio analógico
</title>
<estilo>
/* Esses estilos CSS se aplicam aos elementos SVG definidos

abaixo */
#relógio

{

/* Estilos para tudo

No relógio:*/

AVC
:

preto
;

/ * linhas pretas */

AVC-LINECAP
:

redondo
;

/ * Com extremidades arredondadas */

preencher
:

#ffe
;

/* em um esbranquiçado

fundo */
}
#relógio

.face

{

largura de derrame
:
```

<g

class =
"ticks"
>

<!- ■■marcas de escala 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 '

sendo essencialmente usado para definir atributos de tags SVG que aparecem no documento.Observe também que o CSS
fonte
Propriedade abreviada não
Trabalhe para tags SVG e você deve definir explicitamente
Fonte-família
, Assim,
tamanho de fonte
, e
peso-fonte
como propriedades de estilo separadas.

15.7.2 Scripts SVG

Um
Razão para incorporar SVG diretamente nos seus arquivos HTML (em vez de
Apenas usando estático
<MIG>
tags) 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>
marcação (
§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
notei que o arquivo html inclui um
<Cript>
marcação.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 em
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,
então usa
QuerySelector ()
Para procurar os elementos SVG que
exibir essas mãos, então define um
transformar
atribuir neles a
Gire -os ao redor do centro da face do relógio.A função usa
setTimeout ()
Para garantir que funcione uma vez por minuto:
(
função

UpdateClock
()

{

// Atualize o gráfico do relógio SVG para

Mostre a hora atual

deixar

agora

=

novo

Data
();

// Atual

tempo

deixar

Sec

=

agora
.
GetsEconds
();

// segundos

deixar

min

=

Estilos CSS que geralmente são roteirizados de
JavaScript:
Definindo o
mostrar
estilo para "nenhum" esconde um elemento.Você
Mais tarde pode mostrar o elemento configurando
mostrar
para outro
valor.
Você pode posicionar dinamicamente elementos definindo o
posição
estilo para "absoluto", "parente" ou "fixo" e depois
definindo o
principal
e
esquerda
Estilos para as coordenadas desejadas.
Isso é importante ao usar o JavaScript para exibir dinâmico
Conteúdo como diálogos modais e dicas de ferramentas.
Você pode mudar, escalar e girar elementos com o
transformar
estilo.
Você pode animar mudanças em outros estilos de CSS com o
transição
estilo.Essas animações são tratadas
automaticamente pelo navegador da web e não requer
JavaScript, mas você pode usar o JavaScript para iniciar o
animações.

15.4.1 Classes CSS

O
maneira mais simples de usar o JavaScript para afetar o estilo do documento
O conteúdo é adicionar e remover nomes de classe CSS do
aula
atributo das tags HTML.Isso é fácil de fazer com o
Lista de classe
propriedade de objetos de elemento, como explicado em
“O atributo de classe”
.
Suponha, por exemplo, que a folha de estilo do seu documento inclua um
Definição para uma classe "oculta":

```
.escondido

{

mostrar
:
nenhum
;
}
```

Erro ao traduzir esta página.

O
para/de
loop e
...
Operador espalhado introduzido no ES6
são maneiras particularmente úteis de iterar matrizes.
A classe da matriz define um rico conjunto de métodos para manipular
matrizes, e você deve se familiarizar com o
API da matriz.

```
deixar

{
largura
, Assim,

altura
, Assim,

cx
, Assim,

cy
, Assim,

r
, Assim,

LX
, Assim,

ly
, Assim,

dados
}

=

opções
;

// Este é o espaço de nome XML para elementos SVG

deixar

svg

=

"http://www.w3.org/2000/svg"
;

// Crie o elemento <Svg> e especifique o tamanho do pixel e

coordenadas do usuário

deixar

gráfico

=

documento
.
createElementns
(
svg
, Assim,
```

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
jogar()
e
pausa()
para
Controlando a reprodução de arquivos de áudio e vídeo.
15.1.3 O objeto global nos navegadores da Web
Lá
é um objeto global por janela ou guia do navegador (
§3.7
).Todos os
Código JavaScript (exceto código em execução em threads de trabalhadores; veja
§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
parseint ()
função, 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, o
documento
A propriedade representa
o documento atualmente exibido, o
buscar()
O método fabrica http
solicitações de rede e o
Áudio ()
Construtor 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
história
(
§15.10.2
),
que representam a história de navegação da janela e
INNERWIDTH
, Assim,
que mantém a largura da janela em pixels.Uma das propriedades deste

Erro ao traduzir esta página.

matriz do elemento hospedeiro e não são visitados por DOM normal
métodos de travessia, 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
<udio>
e
<Video>
Elementos: 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 um
Shadowroot
propriedade que JavaScript pode usar para ganhar

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
getContext ()
Método da tela.Chamar
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
<Canvas>
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>
deixar

tela

=

documento
.
QuerySelector
(
"#quadrado"
);
```

contexto
.
preencher
();

// preencha o

caminho
</script>
Vimos que o SVG descreve formas complexas como um "caminho" de linhas
e curvas que podem ser desenhadas ou preenchidas.A API de tela também usa o
noção de um caminho.Em vez de descrever um caminho como uma série de letras e
números, um caminho é definido por uma série de chamadas de método, como o
BeginPath ()
e
arco()
invocações no código anterior.Uma vez a
o caminho é definido, outros métodos, como
preencher()
, opere nesse caminho.
Várias propriedades do objeto de contexto, como
Filtyle
, especificar
Como essas operações são
realizado.
As subseções a seguir demonstram os métodos e propriedades de
A API 2D de tela.Grande parte do código de exemplo a seguir opera
em uma variável
c
.Esta variável detém o CanVArnderingContext2D
objeto da tela, mas o código para inicializar essa variável é
Às vezes não é mostrado.Para fazer esses exemplos funcionarem, você faria
precisa adicionar marcação HTML para definir uma tela com apropriado
largura
e
altura
atributos e adicione código como este para inicializar o
variável
c
:
deixar

tela

=

documento
.
QuerySelector
(
"#my_canvas_id"
);
deixar

c

=

tela

caminho com o
BeginPath ()
método.Comece um novo
Subpata
com o
moveto ()
método.Depois de estabelecer o ponto de partida de um
subpatina com
moveto ()
, você pode conectar esse ponto a um novo ponto
com uma linha reta ligando
lineto ()
.O código a seguir define
Um caminho que inclui duas linhas
segmentos:

```
c
.
BeginPath
();

// Inicie um novo caminho
c
.
moveto
(
100
, Assim,

100
);

// Comece um sub -caminho em (100.100)
c
.
lineto
(
200
, Assim,

200
);

// Adicione uma linha de (100.100) a

(200.200)
c
.
lineto
(
100
, Assim,

200
);

// Adicione uma linha de (200.200) a

(100.200)
```

Este código simplesmente define um caminho;não desenha nada no

Figura 15-7.
Um caminho simples, cheio e acariciado
Observe que o subpatina definido em
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
preencher()
O método preenche subpaths abertos agindo como se 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
ligaria para o
ClosePath ()
método para conectar o ponto final do
Subpata ao ponto de partida.(Você também pode ligar
Lineto (100.100)
, Assim,
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
ClosePath ()
.)
Existem outros dois pontos importantes para perceber
AVC()
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
, Assim,
100
);

// Comece um novo sub -caminho em (300.100);
c
.
lineto
(
300
, Assim,
200
);

// Desenhe uma linha vertical para

(300.200);
Se então ligamos
AVC()
, desenharíamos duas bordas conectadas de
um triângulo e uma linha vertical desconectada.
O segundo ponto a ser observado sobre
AVC()
e

Esteja lá quando você ligar
AVC()
.Quando você termina com um caminho
e quero começar outro, você deve se lembrar de ligar
BeginPath ()
.
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 ()
, Assim,
lineto ()
, e
ClosePath ()
para definir subspates e de
preencher()
e
AVC()
para desenhar
Esses caminhos.Produz o desenho mostrado em
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
, Assim,

n
, Assim,

x
, Assim,

y
, Assim,

r
, Assim,

ângulo
=
0

O valor de retorno da função é
indefinido
.
O

PrintProps ()
A função é diferente: seu trabalho é produzir o
Nomes e valores das propriedades de um objeto.Nenhum valor de retorno é
necessário, e a função não inclui um
retornar
declaração.O
valor de uma invocação do
PrintProps ()
A função é sempre
indefinido
.Se uma função não contém um
retornar
declaração, isso
simplesmente executar cada declaração no corpo da função até atingir o
fim e retorna o
indefinido
valor para o chamador.
Antes do ES6, as declarações de função eram permitidas apenas no nível superior
dentro de um arquivo JavaScript ou dentro de outra função.Enquanto alguns
As implementações dobraram a regra, não era tecnicamente legal definir
funções dentro do corpo de loops, condicionais ou outros blocos.Em
o
Modo rigoroso de ES6, no entanto, as declarações de função são permitidas dentro de
blocos.Uma função definida em um bloco existe apenas dentro desse bloco,
No entanto, e não é visível fora do bloco.
8.1.2 Expressões de função
Expressões de função
Parece muito com declarações de função, mas elas
aparecer no contexto de uma expressão ou afirmação maior, e o
O nome é opcional.Aqui estão algumas expressões de função de exemplo:

```
// Esta expressão de função define uma função que quadrar

seu argumento.
// Observe que o atribuímos a uma variável
const

quadrado

=

função
(
x
)

{

retornar

x
*
x
;
```

Erro ao traduzir esta página.

Combine os pontos de codepates unicode completos em vez de corresponder valores de 16 bits.
Esta bandeira foi introduzida no ES6, e você deve criar o hábito de
Usando -o em todas as expressões regulares, a menos que você tenha algum motivo, não
para.Se você não usar esta bandeira, seus regexps não funcionarão
bem com texto que inclui emoji e outros personagens (incluindo
muitos caracteres chineses) que exigem mais de 16 bits.Sem
o
u
Bandeira, o "".O caractere corresponde a qualquer valor de 1 UTF-16 de 16 bits.
Com a bandeira, no entanto, "".corresponde a um ponto de código unicode,
incluindo aqueles que têm mais de 16 bits.Definindo o
u
bandeira em um
Regexp também permite que você use o novo
\ u {...}
sequência de fuga
para caráter unicode e também permite o
\ p {...}
notação para
Classes de caracteres Unicode.
y
O
y
A bandeira indica que a expressão regular é "pegajosa" e
deve corresponder no início de uma corda ou no primeiro personagem
Após a partida anterior.Quando usado com uma expressão regular
que foi projetado para encontrar uma única correspondência, trata efetivamente que
expressão regular como se comece com
^
para ancorá -lo ao
começo da string.Esta bandeira é mais útil com regular
expressões que são usadas repetidamente para encontrar todas as correspondências dentro de um
corda.Nesse caso, causa comportamento especial da corda
corresponder()
método e o regexp
exec ()
Método para aplicar
que cada partida subsequente está ancorada na posição da string em
que o último terminou.
Esses sinalizadores podem ser especificados em qualquer combinação e em qualquer ordem.Para
por exemplo, se você deseja que sua expressão regular seja consciente de unicode para
Faça uma correspondência insensível ao caso e você pretende usá-lo para encontrar múltiplos
Corresponde dentro de uma string, você especificaria as bandeiras
uig
, Assim,
GUI
, ou qualquer
Outra permutação dessas três letras.

Linha de linha
no objeto de contexto da tela.Essas propriedades são
atributos gráficos que especificam a cor a ser usada por
preencher()
e por
AVC()
e a largura das linhas a serem desenhadas por
AVC()
.
Observe que esses parâmetros não são passados ■■para o
preencher()
e
AVC()
métodos, mas fazem parte do general
estado gráfico
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
Filtyle
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 html
documentos.
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
O

Linha de linha
Propriedade especifica quão amplo (em pixels CSS)
linhas desenhadas por
AVC()
vai ser.O valor padrão é 1. É
importante entender que a largura da linha é determinada pelo
Linha de linha
propriedade na época
AVC()
é chamado, não na época
que
lineto ()
e outros métodos de construção de caminho são chamados.Para totalmente
entenda o
Linha de linha
propriedade, é importante visualizar caminhos
como linhas unidimensionais infinitamente finas.As linhas e curvas desenhadas por
o
AVC()
o método 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, chame o salvar() e grampo() Métodos primeiro, depois ligue AVC() e restaurar() .(O salvar() , Assim, restaurar() , e grampo() Os métodos são descritos posteriormente.)

Ao desenhar linhas que têm mais de cerca de dois pixels de largura, o LineCap e Linejoin propriedades 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. 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 "bunda".O valor padrão para

Linejoin
é "miter".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 o
Miterlimit
propriedade, aquele vértice
será desenhado com uma junção chanfrada em vez de uma junção de mitered.O padrão
valor para
Miterlimit
é 10.
O
AVC()
o método 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
o 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 ()
assim:

```
c
.
setLinedash
([
18
, Assim,

3
, Assim,

3
, Assim,

3
]);

// 18px Dash, 3px Espaço, 3px
```

DOT, 3px Espaço
Finalmente, o
LinedasHoffset
Propriedade especifica a que distância do
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
O

Filtyle
e
Strokestyle
Propriedades especificam como os caminhos

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
Filtyle
(ou
Strokestyle
) a um objeto canvas degradado devolvido pelo
createLinearGradient ()
ou
CreaterAdialGradient ()
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
CreaterAdialGradient ()
Especifique os centros e os raios de dois círculos.(Eles não precisam ser Concêntrico, mas o primeiro círculo normalmente está inteiramente dentro do segundo.)
Áreas dentro do círculo menor ou fora do maior serão preenchidas com cores sólidas;Áreas entre os dois serão preenchidas com um gradiente de cores.
Depois de criar o objeto Canvasgradiente que define as regiões do tela que será preenchida, você deve definir as cores do gradiente ligando
o
addColorstop ()
Método 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.

Erro ao traduzir esta página.

inserido no documento para ser usado dessa maneira.) O segundo
argumento para
createpattern ()
é a string "repetir", "repetição-x",
"Repetir-y" ou "sem repetir", que especifica se (e em que
dimensões) as imagens de fundo se repetem.
Estilos de texto
O

fonte
Propriedade especifica a fonte a ser usada pela desenho de texto
Métodos
FILLTEXT ()
e
STROKETEXT ()
(ver
"Texto"
).O valor
do
fonte
A propriedade deve ser uma string na mesma sintaxe que o CSS
fonte
atributo.
O
Textalign
Propriedade especifica como o texto deve ser
alinhado horizontalmente em relação à coordenada X passada para
FILLTEXT ()
ou
STROKETEXT ()
.Os valores legais são "iniciantes", "esquerda"
"Center", "Certo" e "End".O padrão é "start", que, para a esquerda para
Texto certo, tem o mesmo significado que "esquerda".
O
TextBaseline
Propriedade especifica como o texto deve ser
verticalmente alinhado em relação ao
y
coordenada.O valor padrão é
"Alfabético", e é apropriado para scripts latinos e similares.O
valor “ideográfico” destina -se ao uso com scripts como chinês
e japonês.O valor "pendurado" é destinado ao uso com Devanagari
e scripts semelhantes (que são usados ■■para muitas das línguas da Índia).
As linhas de base “top”, “meio” e “inferior” são puramente geométricas
linhas de base, com base no “quadrado Em EM” da fonte.
Sombras
Quatro
propriedades do contexto Controle o desenho de queda

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.
O
ShadowColor
Propriedade 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.
O
Shadowoffsetx
e
Shadowoffsety
Propriedades Especifique 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
coordenada
Transformações (
§15.8.5
):
a direção da sombra e a "altura" permanecem consistentes mesmo quando as formas são girado e escalado.
O
Shadowblur
propriedade especifica o quão borrado 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 quer acariciar ou encher um caminho usando uma cor translúcida, você pode definir
Strokestyle
ou
Filtyle
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 o
Globalalpha
propriedade.Todo
pixel que você desenha terá seu valor alfa multiplicado por
Globalalpha
.
O padrão é 1, o que não adiciona transparência.Se você definir
Globalalpha
para 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

GlobalComposePoperation
propriedade 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
para "destino-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
GlobalComposePoperation
, mas a maioria tem apenas especializada
usos e não são cobertos aqui.
Salvando e restaurando o estado gráfico
Desde
A API de tela define atributos gráficos no objeto de contexto,
você pode ficar tentado a ligar
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
<Canvas>
elemento tem apenas um único objeto de contexto e cada
chamar 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
salvar()
o método empurra o estado gráfico atual para uma pilha de
Estados salvos.O
restaurar()
o método 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
Nós temos
já vi alguns métodos básicos de tela -
BeginPath ()
, Assim,
moveto ()
, Assim,
lineto ()
, Assim,
ClosePath ()
, Assim,
preencher()
, e
AVC()
- 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 a corrente
Strokestyle
e outros atributos de linha.
ClearRect ()
é como
Fillrect ()
, mas ignora o preenchimento atual

matriz do elemento hospedeiro e não são visitados por DOM normal
métodos de travessia, 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
<udio>
e
<Video>
Elementos: 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 um
Shadowroot
propriedade que JavaScript pode usar para ganhar

caminho.
elipse()
Esse
Método é muito parecido
arco()
exceto que adiciona uma elipse ou um
parte de uma elipse para o caminho.Em vez de um raio, ele tem dois:
um
x
-exis raio e um
y
-exis raio.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 ()
Esse
o método desenha uma linha reta e um arco circular como o
arco()
o método faz, mas especifica o arco a ser desenhado usando
parâmetros diferentes.Os argumentos para
arcto ()
Especifique os 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 ()
Esse
o 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

Erro ao traduzir esta página.

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
});
deixar

total

=

distância
(
0
, Assim,
0
, Assim,
2
, Assim,
1
)

+

distância
(
2
, Assim,
1
, Assim,
3
, Assim,
5
);
deixar

probabilidade

=

fatorial
(
5
)
/
fatorial
(
13
);
```

Em uma invocação, cada expressão de argumento (aqueles entre os

```
c
.
Arcto
(
400
, Assim,
150
, Assim,
400
, Assim,
50
, Assim,
10
);

// Adicione a borda inferior e inferior esquerdo

canto.
c
.
Arcto
(
400
, Assim,
50
, Assim,
500
, Assim,
50
, Assim,
0
);

// Adicione a borda esquerda e a parte superior esquerda

canto.
c
.
Closepath
();

// Caminho próximo para adicionar o resto

a borda superior.
// Curva Bezier quadrática: um ponto de controle
c
.
moveto
(
525
, Assim,

125
);

// Comece aqui
c
.
quadraticcurveto
```

Quando desenhado usando o
fonte
propriedade, a tela fará com que
Escalando -o ou usando uma fonte mais estreita ou menor.
Se você precisar medir o texto antes de desenhá -lo, passe para o
medutorExt ()
método.Este método retorna um objeto TextMetrics
que especifica as medições do texto quando desenhado com o
atual
fonte
.No momento da redação deste artigo, a única "métrica" ■■contida
no
TextMetrics
objeto é a largura.Consulte a largura na tela
de uma corda como esta:
deixar

largura

=

c
.
medutorext
(
texto
).
largura
;
Isso é útil se você deseja centralizar uma série de texto em uma tela, para
exemplo.
Imagens
Em
Além dos gráficos vetoriais (caminhos, linhas, etc.), a API de tela também
Suporta imagens de bitmap.O
drawImage ()
Método copia os pixels
de uma imagem de origem (ou de um retângulo dentro da imagem de origem) no
tela, dimensionando e girando os pixels da imagem conforme necessário.
drawImage ()
pode ser invocado com três, cinco ou nove argumentos.Em
Todos os casos, o primeiro argumento é a imagem de origem da qual os pixels são
ser copiado.Este argumento de imagem é frequentemente um
<MIG>
elemento, mas isso
também pode ser outro
<Canvas>
elemento ou mesmo um
<Video>
elemento
(do qual um único quadro será copiado).Se você especificar um
<MIG>
ou
<Video>
elemento que ainda está carregando seus dados, o
drawImage ()
Chamada não fará nada.

Erro ao traduzir esta página.

Ao contrário de todos os outros métodos descritos aqui,
Todataurl ()
é a
Método do próprio elemento da tela, não do objeto de contexto.Você
normalmente invocar
Todataurl ()
sem argumentos, e retorna o
conteúdo da tela como uma imagem PNG, codificada como uma string usando um
dados:
Url.O URL retornado é adequado para uso com um
<MIG>
elemento, e você pode fazer um instantâneo estático de uma tela com código como
esse:
deixar

img

=

documento
.
CreateElement
(
"IMG"
);

// Crie um <MG>

elemento
img
.
src

=

tela
.
Todataurl
();

// Defina seu SRC

atributo
documento
.
corpo
.
AppendChild
(
img
);

// anexá -lo ao

documento
15.8.5 Transformações do sistema de coordenadas
Como
Observamos que o sistema de coordenadas padrão de uma tela coloca o
origem no canto superior esquerdo, tem

O

setTransform ()
o método permite que você defina 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.
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.

Figura 15-11.
Coordenar transformações do sistema
O

traduzir()
Método simplesmente move a origem da coordenada
sistema esquerdo, direita, para cima ou para baixo.O
girar ()
O método 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
escala()
Método se estende ou
contrata distâncias ao longo do
x
ou
y
eixos.
Passando um fator de escala negativa para o
escala()
Método vira esse eixo
através da origem, como se fosse refletido em um espelho.Isso é o que era
feito no canto inferior esquerdo de
Figura 15-11
:
traduzir()
foi usado para
Mova a origem para o canto inferior esquerdo da tela, então
escala()
foi usado para virar o
y
eixo ao redor para que
y
as coordenadas aumentam 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
traduzir()
, Assim,
girar ()
, e
escala()
como transformar os eixos
do sistema de coordenadas como ilustrado em
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

Essas equações:
x
'= x + dx;// Uma coordenada X de 0 no novo sistema é DX

no velho
você

=

y

+

dy
;
As operações de escala têm equações igualmente simples.Uma chamada
C.Scale (SX, SY)
pode ser descrito assim:
x
'= sx * x;
você

=

sy

*

y
;
As rotações são mais complicadas.A chamada
c.rotate (a)
é descrito
Por essas equações trigonométricas:
x
'= x * cos (a) - y * sin (a);
você

=

y

*

cos
(
um
)

+

x

*

pecado
(
um
);

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 suportadas pela tela são conhecidas como
afim
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
A transformação afim arbitrária pode ser descrita pelos seis parâmetros
um
através
f
Nessas equações:
x
'= ax + cy + e
você

=

bx

+

dy

+

```
(x, y)
// Isso também pode ser realizado com uma tradução, gira,

traduzir sequência
função

roteabout
(
c
, Assim,

Theta
, Assim,

x
, Assim,

y
)

{

deixar

Ct

=

Matemática
.
cos
(
Theta
);

deixar

st

=

Matemática
.
pecado
(
Theta
);

c
.
transformar
(
Ct
, Assim,

-
st
, Assim,
```

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
lineto ()
método.Todo
segmento de linha única em
Figura 15-12
é desenhado assim:

```
c
.
lineto
(
Len
, Assim,

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

```
deixar

deg

=

Matemática
.
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

floco de neve
(
c
, Assim,

n
, Assim,

x
, Assim,
```

no sentido anti -horário

perna
(
n
);

// Desenhe a segunda perna

c
.
girar
(
-
120
*
deg
);

// gira novamente

perna
(
n
);

// Desenhe a perna final

c
.
Closepath
();

// Fechar o subspô

c
.
restaurar
();

// 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

```
deixar

c

=

documento
.
QuerySelector
(
"tela"
).
getContext
(
"2d"
);
floco de neve
(
c
, Assim,

0
, Assim,

25
, Assim,

125
, Assim,

125
);

// um floco de neve de nível-0 é um

triângulo
floco de neve
(
c
, Assim,

1
, Assim,

175
, Assim,

125
, Assim,

125
);

// A Nível 1 Snowflake é um 6-
estrela do lado
floco de neve
(
c
, Assim,
```

Figura 15-13.
Movimentos não soltos e preenchimentos cortados
Figura 15-13
foi gerado usando o
polígono()
método de
Exemplo 15-5
e o seguinte código:

```
// Defina alguns atributos de desenho
c
.
fonte

=

"Bold 60pt sem serif"
;

// Big Font
```

```
(x, y)
// Isso também pode ser realizado com uma tradução, gira,

traduzir sequência
função

roteabout
(
c
, Assim,

Theta
, Assim,

x
, Assim,

y
)

{

deixar

Ct

=

Matemática
.
cos
(
Theta
);

deixar

st

=

Matemática
.
pecado
(
Theta
);

c
.
transformar
(
Ct
, Assim,

-
st
, Assim,
```

Erro ao traduzir esta página.

Figura 15-14.
Um efeito desfoque de movimento criado pelo processamento de imagem
O código a seguir demonstra
getImagedata ()
e
PutImagedata ()
e mostra como iterar e modificar o
Valores de pixel em um objeto imagedata.
Exemplo 15-8.
Motion Blur com IMAGEDATA
// mancha os pixels do retângulo à direita, produzindo um
// tipo de movimento borrado como se os objetos estivessem se movendo da direita para

esquerda.
// N deve ser 2 ou maior.Valores maiores produzem manchas maiores.
// O retângulo é especificado no sistema de coordenadas padrão.
função

mancha
(
c
, Assim,

n
, Assim,

x
, Assim,

y
, Assim,

c
, Assim,

h
)

{

// Obtenha o objeto IMagedata que representa o retângulo

de pixels para manchar

deixar

pixels

=

c
.
getImagedata
(
x
, Assim,

y
, Assim,

```
para
(
deixar

linha

=

0
;

linha

<

altura
;

linha
++
)

{

// para cada linha

deixar

eu

=

linha
*
largura
*
4

+

4
;

// O deslocamento do segundo

pixel da linha

para
(
deixar

col

=

1
;

col
```

Você
não precisa incluir um
<udio>
Tag em seu documento HTML
Para incluir efeitos sonoros em suas páginas da web.Você pode
Crie dinamicamente
<udio>
Elementos com o DOM normal
document.createElement ()
método, ou, como um atalho, você pode
Basta usar o
Áudio ()
construtor.Você não precisa adicionar o
Elemento criado para o seu documento para reproduzi -lo.Você pode simplesmente
chame seu
jogar()
método:

```
// carrega o efeito sonoro com antecedência, para que esteja pronto para uso
deixar

SOLEFEFT

=

novo

Áudio
(
"Soundeffff.mp3"
);
// Reproduza o efeito sonoro sempre que o usuário clica no mouse

botão
documento
.
addEventListener
(
"clique"
, Assim,

()

=>

{

SOLEFEFT
.
CLONENODE
().
jogar
();

// Carregar e reproduzir o

som
});
```

Observe o uso de
CLONENODE ()

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.
deixar

Audiocontext

=

novo

(
esse
.
Audiocontext
||
esse
.
webkitaudiocontext
) ();
// Defina o som base como uma combinação de três seno puro

ondas
deixar

Notas

=

[[

293.7
, Assim,

370.0
, Assim,

440.0

];

// D Major acorde: D, F#

e a
// Crie nós do oscilador para cada uma das notas que queremos

jogar
deixar

osciladores
```

```
// Conecte cada uma das notas de origem ao controle de volume
osciladores
.
foreach
(
o

=>

o
.
conectar
(
Volumecontrol
));
// e conecte a saída do controle de volume ao

alto -falantes.
Volumecontrol
.
conectar
(
alto -falantes
);
// Agora comece a tocar os sons e deixe -os correr para 1,25

segundos.
deixar

StartTime

=

Audiocontext
.
CurrentTime
;
deixar

StopTime

=

StartTime

+

1.25
;
osciladores
.
foreach
(
o

=>

{
```

O
Hash
e
procurar
As propriedades do objeto de localização são
interessantes.O
Hash
Propriedade retorna o "identificador de fragmento"
parte do URL, se houver um: uma marca de hash (#) seguida por um
ID do elemento.O
procurar
propriedade 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 um
SearchParams
propriedade que é analisada
Representação do
procurar
propriedade.O objeto de localização não
tem um
SearchParams
propriedade, mas se você quiser analisar
Window.Location.Search
, você pode simplesmente criar um objeto URL
a partir do objeto de localização e depois use o URL's
SearchParams
:

```
deixar

url

=

novo

Url
(
janela
.
localização
);
deixar

consulta

=

url
.
SearchParams
.
pegar
(
"Q"
);
```

Os operadores de incremento e decréscimos são semelhantes, pois realizam um atribuição implícita.O
excluir
O operador também tem efeitos colaterais:
Excluir uma propriedade é como (mas não o mesmo que) atribuir
indefinido
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
O
operadores listados em
Tabela 4-1
estão dispostos em ordem da 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:

```
c
=
x
+
y
*
z
;
```

O operador de multiplicação
*
tem uma precedência maior que a adição
operador
+
, então 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

Você também pode carregar uma nova página passando uma nova string para o
atribuir()
Método do objeto de localização.Isso é o mesmo que
atribuindo a string ao
localização
propriedade, no entanto, então não é
particularmente interessante.
O
substituir()
o método do objeto de localização, por outro lado, é
bastante útil.Quando você passa uma corda para
substituir()
, é interpretado
como um URL e faz com que o navegador carregue uma nova página, assim como
atribuir()
faz.A diferença é que
substituir()
substitui o
Documento atual na história do navegador.Se um script no documento A
Define o
localização
propriedade ou chamadas
atribuir()
Para carregar o documento B
E então o usuário clica no botão Voltar, o navegador voltará a
Documento A. Se você usa
substituir()
Em vez disso, 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, o
substituir()
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.
se

(
!
IsbrowSerSuportded
())
```

Erro ao traduzir esta página.

Se uma janela contém janelas infantis (como
<frame>
elementos),
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

Um
A técnica de gerenciamento de história envolve
location.hash
e
O evento "Hashchange".Aqui estão os principais fatos que você precisa saber para
Entenda esta técnica:
O
location.hash
Propriedade 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 o
Hash
propriedade como

esse.
Definindo o
location.hash
Atualiza a propriedade do URL
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
location.hash
explicitamente, um evento "hashchange"
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
Retorne ao seu URL anterior antes de definir
location.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 "hashcrange"
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.

que você instalou em seu sistema através de um gerenciador de pacotes, então
Você simplesmente usa o nome não qualificado do módulo, sem qualquer "/"
Personagens que o transformariam em um caminho do sistema de arquivos:

```
// Esses módulos são incorporados para o nó
const

fs

=

exigir
(
"FS"
);

// o embutido

Módulo de sistema de arquivos
const

http

=

exigir
(
"http"
);

// o http integrado

módulo
// A estrutura Express HTTP Server é um módulo de terceiros.
// não faz parte do nó, mas foi instalado localmente
const

expressar

=

exigir
(
"expressar"
);
```

Quando você deseja importar um módulo de seu próprio código, o módulo
o nome deve ser o caminho para o arquivo que contém esse código, em relação a
o arquivo do módulo atual.É legal usar caminhos absolutos que começam
com um
/
caráter, mas normalmente, ao importar módulos que fazem parte
do seu próprio programa, os nomes dos módulos começarão com
./
ou
às vezes
../
para indicar que eles são relativos ao diretório atual ou
o diretório pai.Por exemplo:

```
const
```

Quando descemos.Existem dois pontos diferentes que podemos usar como o
Origem do sistema de coordenadas, no entanto: o
x
e
y
coordenadas de um
o elemento pode ser relativo ao canto superior esquerdo do documento ou
em relação ao canto superior esquerdo de
o
viewport
em que o documento está
exibido.Nas janelas e guias de nível superior, a "viewport" é a parte
do navegador que realmente exibe o conteúdo do documento: ele exclui
Navegador “Chrome”, como menus, barras de ferramentas e guias.Para documentos
exibido em
<frame>
tags, é o elemento iframe no dom que
Define a viewport para o documento aninhado.Em ambos os casos, quando nós
Fale sobre a posição de um elemento, devemos deixar claro se estamos
usando coordenadas de documentos ou coordenadas de viewport.(Observe que
As coordenadas de viewport às vezes são chamadas de "coordenadas de janelas".)
Se o documento for menor que a visualização, ou se não foi
Rolado, o canto superior esquerdo do documento está no canto superior esquerdo
canto da viewport e o documento e coordenada de viewport
Os sistemas são iguais.Em geral, no entanto, para converter entre os dois
Coordenar sistemas, devemos adicionar ou subtrair o

Role compensações
.
Se um
elemento tem um
y
coordenada de 200 pixels em coordenadas de documentos, para
exemplo, e se o usuário rolou para baixo por 75 pixels, então isso
elemento tem um
y
Coordenada de 125 pixels nas coordenadas de viewport.
Da mesma forma, se um elemento tiver um
x
Coordenada de 400 na viewport
Coordena depois que o usuário rolou os 200 pixels da Viewport 200
horizontalmente, então o elemento
x
coordenar em coordenadas de documentos
é 600.
Se usarmos o modelo mental de documentos de papel impresso, é lógico para
Suponha que todo elemento em um documento deve ter uma posição única
nas coordenadas de documentos, independentemente de quanto o usuário rolou

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
indefinido
.
As propriedades do acessador podem ser definidas com uma extensão do objeto
Sintaxe literal (ao contrário das outras extensões ES6, temos
Visto aqui, getters
e os setters foram introduzidos no ES5):
deixar

o

=

{

// Uma propriedade de dados comum

DataProp
:

valor
, Assim,

// Uma propriedade acessadora definida como um par de funções.

pegar

ACESTORPROP
()

{

retornar

esse
.
DataProp
;

},

definir

ACESTORPROP
(
valor
)

Erro ao traduzir esta página.

```
a função
// f em e e em cada um de seus descendentes
função

atravessar
(
e
, Assim,

f
)

{

f
(
e
);

// Invocar f () em e

para
(
deixar

criança

de

e
.
crianças
)

{

// itera sobre o

crianças

atravessar
(
criança
, Assim,

f
);

// e recorrente a cada

um

}
}
função

Traverse2
(
e
```

```
// 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

Fromurl
(
url
)

{

deixar

s

=

novo

Gamestate
();

deixar

params

=

novo

Url
(
url
).
SearchParams
;

s
.
baixo

=

parseint
(
params
.
pegar
(
```

```
deixar

entrada

=

documento
.
QuerySelector
(
"#entrada"
);

//

Adivinhe o campo de entrada

deixar

Playagain

=

documento
.
QuerySelector
(
"#playagain"
);

// Atualize o cabeçalho e o título do documento

cabeçalho
.
TextContent

=

documento
.
título

=

`
EU
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 feedback com base no usuário '
s

durar
```

```
UpdateForGuess
(
adivinhar
)

{

// se for um número e está no intervalo certo

se

((
adivinhar

>

esse
.
baixo
)

&&

(
adivinhar

<

esse
.
alto
))

{

// Atualize o objeto de estado com base neste palpite

se

(
adivinhar

<

esse
.
segredo
)

esse
.
baixo

=

adivinhar
;

outro
```

```
// 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
documento
.
QuerySelector
(
"#entrada"
).
OnChange

=

(
evento
)

=>

{

se

(
gamestate
.
UpdateForGuess
(
parseint
(
evento
.
alvo
.
valor
)))

{

história
.
pushState
(
gamestate
, Assim,

""
, Assim,

gamestate
.
Tourl
());

}
```

As funções próprias do construtor para inicializar objetos recém -criados.Fazendo isso está coberto em
Capítulo 9
.

6.2.3 Protótipos

Antes
Podemos 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 a
protótipo
, e o primeiro objeto herda as propriedades do
protótipo.
Todos
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 o
novo
palavra -chave e a
Invocação do construtor Use o valor do
protótipo
propriedade de
o construtor funciona como seu protótipo.Então o objeto criado por
novo objeto ()
herda de
Object.prototype
, assim como o
objeto criado por
{}
faz.Da mesma forma, o objeto criado por
novo
Variedade()
usos
Array.prototype
como seu protótipo e o objeto
criado por
nova data ()
usos
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 um
protótipo
propriedade.São esses objetos com
protótipo
propriedades que definem o
protótipos
para todo o outro
objetos.
Object.prototype
é um dos objetos raros que não tem protótipo:
Não herda nenhuma propriedade.Outros protótipos objetos são normais objetos que têm um protótipo.A maioria dos construtores embutidos (e a maioria

esquecido o que são, você pode querer reler
Capítulo 13
antes
continuando com esta seção.)
Aqui está
que a
buscar()
Solicitação parece se você estiver usando
então()
E espere que a resposta do servidor à sua solicitação seja formatada por JSON:

```
buscar
(
"/API/Usuários/Current"
)

// Faça um http (ou

Https) Obtenha solicitação

.
então
(
resposta

=>

resposta
.
JSON
())

// analisar seu corpo como um

Objeto json

.
então
(
CurrentUser

=>

{

// então processe isso

objeto analisado

DisplayUserinfo
(
CurrentUser
);

});
```

Aqui está uma solicitação semelhante feita usando o
assíncrono
e
aguarde
palavras -chave

Códigos de status HTTP, cabeçalhos de resposta e
Erros de rede
O
três etapas
buscar()
processo mostrado em
§15.11.1
elide todos os erros-
Código de manuseio.Aqui está uma versão mais realista:

```
buscar
(
"/API/Usuários/Current"
)

// Faça um http (ou https)

solicitar.

.
então
(
resposta

=>

{

// Quando obtemos uma resposta,

primeiro verifique

se

(
resposta
.
OK

&&

// para um código de sucesso e o

Tipo esperado.

resposta
.
cabeçalhos
.
pegar
(
"Tipo de conteúdo"
)

===

"Aplicativo/JSON"
)

{
```

A promessa devolvida por
buscar()
resolve para um objeto de resposta.O
status
A propriedade 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, o
OK
propriedade de uma resposta é
verdadeiro
se
status
é 200 ou qualquer código entre 200 e 299 e é
falso
Para qualquer outro código.
buscar()
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.O
cabeçalhos
propriedade de um objeto de resposta
é um objeto de cabeçalhos.Use seu
tem()
método para testar a presença de um
cabeçalho ou usar seu
pegar()
Método para obter o valor de um cabeçalho.Http
Os nomes dos cabeçalhos são insensíveis a maiúsculas, para que você possa passar em minúsculas ou mi
stas
Nomes de cabeçalho de casos para essas funções.
O objeto de cabeçalhos também é iterável se você precisar fazer isso:

```
buscar
(
url
).
então
(
resposta

=>

{

para
(
deixar

[[
nome
, Assim,
valor
]

de
```

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 um
.pegar()
Cláusula sempre que você fizer
um
buscar()
chamar.
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 um
?
.Os URL e URLSearchparams
classes (que foram cobertas em
§11.9
) facilite a construção de URLs
nesta forma, e o
buscar()
função aceita objetos de URL como seus
primeiro argumento, para que você possa incluir parâmetros de solicitação em um
buscar()
Solicitação como este:
assíncrono

função

procurar
(
prazo
)

{

deixar

url

=

novo

Url
(
"/API/Pesquisa"
);

url
.
SearchParams
.
definir
(
"Q"
, Assim,

prazo

objeto que especifica o URL a buscar.O segundo argumento é um
objeto que pode fornecer opções adicionais, incluindo cabeçalhos de solicitação:
deixar

AuthHeaders

=

novo

Cabeçalhos
();
// Não use auth básico, a menos que esteja acima de um https

conexão.
AuthHeaders
.
definir
(
"Autorização"
, Assim,

`BASIC
$ {
BTOA
(
`
$ {
nome de usuário
}
:
$ {
senha
}
`
)
}
`
);
buscar
(
"/API/Usuários/"
, Assim,

{

cabeçalhos
:

AuthHeaders

})

.
então
(
resposta

=>

adição a
JSON ()
e
texto()
, o objeto de resposta também tem esses
Métodos:
ArrayBuffer ()
Esse
O método retorna uma promessa que se 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
) dos quais você pode ler os dados binários.
blob ()
Esse
O método retorna uma promessa que resolve 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
navegadores também definem baseados em promessa
texto()
e
ArrayBuffer ()
métodos que dão uma rota mais direta para
obtendo o conteúdo de um blob.
formData ()
Esse
O 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
Em
Além dos cinco métodos de resposta que retornam de forma assíncrona
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.
O
corpo
A propriedade de um objeto de resposta é um objeto ReadableStream.Se
você já chamou um método de resposta como
texto()
ou
JSON ()
que lê, analisa e devolve o corpo, então
BodyUsed
vai ser
verdadeiro
para indicar que o
corpo
o fluxo já foi lido.Se
BodyUsed
é
falso
, no entanto, o fluxo ainda não foi lido.Nesse caso,
você pode ligar
getReader ()
sobre
resposta.body
Para obter um riacho
objeto de leitor e depois use o
ler()
Método deste objeto de leitor para
Leia de forma assíncrona pedaços de texto do fluxo.O
ler()
o método retorna uma promessa que se resolve a um objeto com
feito
e
valor
propriedades.
feito
vai ser
verdadeiro
Se todo o corpo foi lido
ou se o fluxo foi fechado.E
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
assíncrono
e
aguarde
mas é surpreendentemente complexo se você tentar usá -lo com cru
Promessas.

com o
JSON ()
método do objeto de resposta, mas você pode fazer isso
com o
streambody ()
função, como esta (assumindo que um
updateProgress ()
A função é definida para definir o
valor
atributo
em um html
<Progresso>
elemento):
buscar
(
'big.json'
)

.
então
(
resposta

=>

streambody
(
resposta
, Assim,

UpdateProgress
))

.
então
(
BodyText

=>

JSON
.
analisar
(
BodyText
))

.
então
(
handlebigjsonObject
);
O
streambody ()
a função pode ser implementada como mostrado em
Exemplo 15-10
.
Exemplo 15-10.
Transmitindo o corpo de resposta a partir de uma solicitação de busca ()

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
});
deixar

total

=

distância
(
0
, Assim,
0
, Assim,
2
, Assim,
1
)

+

distância
(
2
, Assim,
1
, Assim,
3
, Assim,
5
);
deixar

probabilidade

=

fatorial
(
5
)
/
fatorial
(
13
);
```

Em uma invocação, cada expressão de argumento (aqueles entre os

bytesread

+=

valor
.
comprimento
;

// passou,

Então chame

RelatórioProgress
(
bytesread
, Assim,

bytesread

/

esperado
);

}

}

se

(
feito
)

{

// se isso for

o último pedaço,

quebrar
;

// Saia do

laço

}

}

retornar

corpo
;

// retorna o texto do corpo que acumulamos
}

"PEGAR"
ou
"CABEÇA"
(que não suportam órgãos de solicitação), você pode
Especifique um corpo de solicitação definindo o
corpo
propriedade das opções
objeto:
buscar
(
url
, Assim,

{

método
:

"PUBLICAR"
, Assim,

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":
buscar
(
url
, Assim,

{

método
:

"PUBLICAR"
, Assim,

cabeçalhos
:

novo

Cabeçalhos
({
"Tipo de conteúdo"
:

"Aplicativo/JSON"
}),

corpo

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 passar o
URLSearchParams objeto como o valor do
corpo
propriedade.
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 a
<morm>
elemento para o
FormData ()
construtor.
Mas você também pode criar órgãos de solicitação "multipart/formulários"
invocando o
FormData ()
construtor sem argumentos
e inicializando o nome do nome/valor que ele representa com o
definir()
e
acrescentar()
Métodos.
Upload de arquivo com fetch ()
Upload
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 um
<entrada
type = "arquivo">
Elemento em sua página da web e ouça "Change"
eventos nesse elemento.Quando um evento de "mudança" ocorre, o
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
Matriz 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.
assíncrono

função

getCanvasblob
(
tela
)

{

retornar

novo

Promessa
((
resolver
, Assim,

rejeitar
)

=>

{

tela
.
Toblob
(
resolver
);

});
}
// Aqui está como enviamos um arquivo PNG de uma tela
assíncrono

função

UPLOPLACHANVASIMAGE
(
tela
)
```

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
buscar()
é usado com uma origem cruzada
URL, o navegador adiciona um cabeçalho de "origem" à solicitação (e não
permita que seja substituído pelo
cabeçalhos
propriedade) 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
Promessa devolvida por
buscar()
é rejeitado.
Abortando um pedido
Às vezes
você pode querer abortar um
buscar()
solicitar que você tenha
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 um
buscar()
solicitar, então
Crie um objeto abortController antes de iniciar a solicitação.O
sinal
A propriedade do objeto do controlador é um objeto aborto.
Passe este objeto de sinal como o valor do
sinal
propriedade do

Você
não precisa incluir um
<udio>
Tag em seu documento HTML
Para incluir efeitos sonoros em suas páginas da web.Você pode
Crie dinamicamente
<udio>
Elementos com o DOM normal
document.createElement ()
método, ou, como um atalho, você pode
Basta usar o
Áudio ()
construtor.Você não precisa adicionar o
Elemento criado para o seu documento para reproduzi -lo.Você pode simplesmente
chame seu
jogar()
método:

```
// carrega o efeito sonoro com antecedência, para que esteja pronto para uso
deixar

SOLEFEFT

=

novo

Áudio
(
"Soundeffff.mp3"
);
// Reproduza o efeito sonoro sempre que o usuário clica no mouse

botão
documento
.
addEventListener
(
"clique"
, Assim,

()

=>

{

SOLEFEFT
.
CLONENODE
().
jogar
();

// Carregar e reproduzir o

som
});
```

Observe o uso de
CLONENODE ()

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 de

cache

:

"padrão"

Este valor especifica o comportamento de cache padrão.Fresco respostas

no cache são servidos diretamente do cache e obsoleto respostas

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.

redirecionar
Esta propriedade controla como o navegador lida com respostas redirecionadas
do servidor.Os três valores legais são:
"seguir"
Este é o valor padrão e faz o navegador seguir
redireciona automaticamente.Se você usar esse padrão, a resposta
objetos que você obtém
buscar()
nunca deveria ter um
status
em
A faixa de 300 a 399.
"erro"
Este valor faz
buscar()
rejeitar sua promessa retornada se o
O servidor retorna uma resposta de redirecionamento.
"manual"
Este valor significa que você deseja lidar com o redirecionamento manualmente
respostas e a promessa devolvida por
buscar()
pode resolver
a um objeto de resposta com um
status
na faixa de 300 a 399.Em
Neste caso, você terá que usar o cabeçalho "localização" do
Resposta a seguir manualmente o redirecionamento.
REFERN
Você pode definir esta propriedade como uma string que contém um URL relativo para
Especifique o valor do cabeçalho HTTP "Referent" (que é
Historicamente, erros com três Rs em vez de quatro).Se você definir isso
Propriedade da string vazia, então o cabeçalho "referente" será
omitido da solicitação.
15.11.2 Eventos enviados pelo servidor
UM
característica fundamental do protocolo HTTP sobre o qual a web é

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
Eventsource ()
construtor.Quando o servidor grava (corretamente
Dados formatados) para a conexão, o objeto Eventsource traduz
Aqueles em eventos que você pode ouvir:
deixar

ticker

=

novo

Eventsource
(
"Stockprices.php"
);
ticker
.
addEventListener
(
"oferta"
, Assim,

(
evento
)

=>

{

DisplayNewbid
(
evento
.
dados
);
}
O
Objeto de evento associado a um evento de mensagem tem um

O objeto de evento também tem um
tipo
propriedade, 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
Eventsource
objeto),
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
buscar()
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

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.
deixar

Audiocontext

=

novo

(
esse
.
Audiocontext
||
esse
.
webkitaudiocontext
) ();
// Defina o som base como uma combinação de três seno puro

ondas
deixar

Notas

=

[[

293.7
, Assim,

370.0
, Assim,

440.0

];

// D Major acorde: D, F#

e a
// Crie nós do oscilador para cada uma das notas que queremos

jogar
deixar

osciladores
```

```
// analisar o URL solicitado

deixar

Nome do caminho

=

url
.
analisar
(
solicitar
.
url
).
Nome do caminho
;

// Se a solicitação foi para "/", envie o chat do lado do cliente

Ui.

se

(
Nome do caminho

===

"/"
)

{

// Um nnpedido de interface do usuário de bate -papo

resposta
.
writehead
(
200
, Assim,

{
"Tipo de conteúdo"
:

"Texto/html"
}).
fim
(
ClientHtml
);

}

// de outra forma, envie um erro 404 para qualquer caminho que não seja
```

```
resposta
.
fim
();

});

// defina cabeçalhos e envie um evento de bate -papo inicial para isso apenas

um cliente

resposta
.
writehead
(
200
, Assim,

{

"Tipo de conteúdo"
:

"Texto/fluxo de eventos"
, Assim,

"Conexão"
:

"Keep-alive"
, Assim,

"Controle de cache"
:

"Sem cache"

});

resposta
.
escrever
(
"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.
assíncrono
```

```
// Agora envie este evento para todos os clientes de escuta

clientes
.
foreach
(
cliente

=>

cliente
.
escrever
(
evento
));
}
```

15.11.3 Websockets

O
Websocket API 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 da webSocket começam com
WSS: //
em vez de
https: //
, no entanto.(Os navegadores normalmente restringem
WebSockets para trabalhar apenas em páginas carregadas em segurança
https: //
conexões).
Para estabelecer uma conexão WebSocket, o navegador primeiro estabelece um
Conexão http e envia ao servidor um
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 usar 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, criar um
Objeto WebSocket, especificando o
WSS: //
URL que identifica o
Servidor e serviço que você deseja usar:
deixar

soquete

=

novo

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.
O
ReadyState
propriedade do soquete especifica em que afirma o
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 o
ONOPEN
propriedade 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
OnError
para
Defina um manipulador ou, alternativamente, use
addEventListener ()
.
Quando terminar com um webSocket, você pode fechar a conexão por
chamando o
fechar()
Método do objeto WebSocket.Quando a
WebSocket muda no estado fechado, ele dispara um evento "próximo" e
você pode definir o
ONCLOSE
propriedade para ouvir este evento.
Enviando mensagens sobre um webSocket
Para
Envie uma mensagem para o servidor do outro lado de um webSocket
conexão, basta invocar o
enviar()
Método do WebSocket
objeto.
enviar()
espera um único argumento de mensagem, que pode ser um
String, Blob, ArrayBuffer, Array digitado ou objeto DataAView.
O
enviar()
Método buffer a mensagem especificada a ser transmitida
e retorna antes que a mensagem seja realmente enviada.O

bufferamount
propriedade 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
Receba 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 da MessageEvent com um
dados
propriedade que contém
a mensagem do servidor.Se o servidor enviou um texto codificado UTF-8, então
Event.Data
será uma string contendo esse texto.
Se o servidor enviar uma mensagem que consiste em dados binários em vez de
texto, então o
dados
Propriedade (por padrão) será um objeto BLOB
representando esses dados.Se você preferir receber mensagens binárias como
Arraybuffers em vez de blobs, defina o
BinaryType
propriedade 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 (ver
“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
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 nisso: 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, o
protocolo
propriedade do
O objeto WebSocket especifica qual versão do protocolo o servidor escolheu.

15.12 Armazenamento Web

Os aplicativos 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.

Lá são várias formas de armazenamento do lado do cliente:

Armazenamento na web

A API de armazenamento da Web consiste no `LocalStorage` e `SessionStorage` objetos, 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
Web
Os navegadores 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 sal vam 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 cont as financeiras,
ou outras informações igualmente sensíveis.

15.12.1 LocalStorage e SessionStorage

O
LocalStorage
e
SessionStorage
propriedades
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:

```
deixar

nome

=

LocalStorage
.
nome de usuário
;

// consulta um armazenado

valor.
se

(
!
nome
)

{

nome
```

pergunta.

LocalStorage
.
nome de usuário

=

nome
;

// armazenar o usuário

resposta.
}
Você pode usar o
excluir
operador para remover propriedades de
LocalStorage
e
SessionStorage
, e você pode usar um
para/in
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,
ligue para o
claro()
método:
LocalStorage
.
claro
();
Objetos de armazenamento também definem
getItem ()
, Assim,
setItem ()
, e
deleteItem ()
métodos que você pode usar em vez de direto
acesso à propriedade e o
excluir
operador, se você quiser.
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

estrutura
LocalStorage
.
dados

=

JSON
.
stringify
(
dados
);

// codifica e

loja
deixar

dados

=

JSON
.
analisar
(
LocalStorage
.
dados
);

// 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.
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
o mesmo
LocalStorage
dados (independentemente da origem dos scripts
Isso realmente acessa
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).

correndo.Quando a janela ou a guia é fechada permanentemente, quaisquer dados
armazenado através
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
associado
SessionStorage
pode ser mais longo do que parece.)
Como
LocalStorage
, Assim,
SessionStorage
é escopo para o
documentar a origem para que documentos com origens diferentes nunca
compartilhar
SessionStorage
.Mas
SessionStorage
também está escopo em
uma base por janela.Se um usuário tiver duas guias do navegador exibindo
Documentos da mesma origem, essas duas guias têm separado
SessionStorage
Dados: 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 em
LocalStorage
mudanças, 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 em
LocalStorage
, a outra guia vai
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
claro()
Método foi chamado, esta propriedade será
nulo
.
NewValue
Mantém o novo valor do item, se houver um.Se
removetem ()
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
LocalStorage
objeto.
url
O URL (como uma string) do documento cujo script fez isso
mudança de armazenamento.
Observe que
LocalStorage
e o evento de "armazenamento" pode 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 em
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
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 o
Cookie
propriedade do documento
objeto.
Por que "cookie"?
O
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 referi
r a um pequeno
pedaço de dados, particularmente um pedaço de dados privilegiados ou secretos, semelhante a uma senh
a, 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
seguro de qualquer forma (embora transmiti -los através de um
https:
a conexão ajuda).
O
A API para manipular cookies é antiga e enigmática.Há
Não há métodos envolvidos: os cookies são consultados, definidos e excluídos pela leitura

e escrevendo o
Cookie
propriedade 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 especialmente formatadas definidas no mesmo
Cookie
propriedade.
As subseções a seguir explicam como consultar e definir valores de cookies
e atributos.
Lendo cookies
Quando
você leu o
document.cookie
Propriedade, 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
document.cookie
Propriedade, você normalmente deve ligar para o
dividir()
Método para dividi -lo em pares de nome/valor individuais.
Depois de extrair o valor de um biscoito do
Cookie
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 então para
Json.parse ()
.
O código a seguir define um
getcookie ()
função que analisa
o
document.cookie
propriedade e retorna um objeto cujo
As propriedades especificam os nomes e valores dos cookies do documento:

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
jogar()
e
pausa()
para
Controlando a reprodução de arquivos de áudio e vídeo.
15.1.3 O objeto global nos navegadores da Web
Lá
é um objeto global por janela ou guia do navegador (
§3.7
).Todos os
Código JavaScript (exceto código em execução em threads de trabalhadores; veja
§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
parseint ()
função, 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, o
documento
A propriedade representa
o documento atualmente exibido, o
buscar()
O método fabrica http
solicitações de rede e o
Áudio ()
Construtor 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
história
(
§15.10.2
),
que representam a história de navegação da janela e
INNERWIDTH
, Assim,
que mantém a largura da janela em pixels.Uma das propriedades deste

Lifetime, o navegador armazenará cookies em um arquivo e os excluirá apenas
Uma vez que eles expirarem.
A visibilidade do biscoito é escopo por origem do documento como
LocalStorage
e
SessionStorage
são, mas também por caminho do documento.Este escopo é
Configurável através de atributos de cookies
caminho
e
domínio
.Por padrão,
Um cookie está associado e acessível à página da web que criou
e qualquer outra página da web no mesmo diretório ou em qualquer subdiretoria
desse diretório.Se a página da web
exemplo.com/catalog/index.html
cria um biscoito, por exemplo, que o cookie também é visível para
exemplo.com/catalog/order.html
e
exemplo.com/catalog/widgets/index.html
, mas não é visível para
exemplo.com/about.html
.
Esse comportamento de visibilidade padrão geralmente é exatamente o que você deseja.
Às vezes, porém, você deseja usar valores de cookies ao longo de um
Site, independentemente de qual página cria o cookie.Por exemplo, se
O usuário insere seu endereço de correspondência em um formulário em uma página, você pode
deseja salvar esse endereço para usar como padrão na próxima vez que eles retornarem
para a página e também como o padrão de uma forma totalmente não relacionada
Outra página em que eles são solicitados a inserir um endereço de cobrança.Para permitir
Este uso, você especifica um
caminho
para o biscoito.Então, qualquer página da web
Do mesmo servidor web cujo URL começa com o prefixo do caminho
Especificado pode compartilhar o cookie.Por exemplo, se um biscoito definido por
exemplo.com/catalog/widgets/index.html
tem seu caminho definido para "/catálogo",
Esse cookie também é visível para
exemplo.com/catalog/order.html
.Ou, se o
O caminho é definido como "/", o cookie é visível para qualquer página no
exemplo.com
domínio, dando ao cookie um escopo como o de
LocalStorage
.

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
ordem.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
caminho
atribuir a "/" e seu
domínio
atributo a ".example.com", esse cookie está disponível para todas as páginas da web
sobre
catalog.example.com
, Assim,
orders.example.com
, e qualquer outro servidor em
o
exemplo.com
domínio.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 nomeado
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
Biscoitos
destinam-se ao armazenamento de pequenas quantidades de dados por scripts do lado do servidor, e ess
es 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 irrestr
ito, mas
não exige que os navegadores retenham mais de 300 cookies no total, 20 cookies por servidor da web o
u 4 kb
de dados por cookie (nome e valor da contagem de valores para esse limite de 4 kb).Na prática, os na
vegadores permitem
Muitos mais de 300 cookies no total, mas o limite de tamanho de 4 kb ainda pode ser aplicado por alg
uns.
Armazenando biscoitos
Para
associar um valor de cookie transitório ao documento atual, simplesmente
defina o
Cookie
propriedade para um
nome = valor
corda.Por exemplo:

```
documento
.
Cookie

=

`versão =
$ {
Encodeuricomponent
(
documento
.
Último modificado
)
}
`
;
```

Na próxima vez que você ler o
Cookie
Propriedade, 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
vida útil (em segundos) com um
MAX-AGE
atributo.Você pode fazer isso por
definindo o
Cookie
propriedade para uma sequência do formulário:
nome = valor;
Max-Aage = segundos
.A função a seguir define um cookie com um
opcional
MAX-AGE
atributo:

```
// 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
```

}
Da mesma forma, você pode definir o
caminho
e
domínio
atributos de um biscoito por
Anexando seqüências de cordas do formulário
; caminho = valor
ou
; domínio = valor
para a string que você definiu no
document.cookie
propriedade.Para definir
o
seguro
Propriedade, basta anexar
;seguro
.
Para alterar o valor de um cookie, defina seu valor novamente usando o mesmo
Nome, caminho e domínio junto com o novo valor.Você pode mudar o
vida de um biscoito quando você muda seu valor especificando um novo
MAX-AGE
atributo.
Para excluir um cookie, defina novamente usando o mesmo nome, caminho e domínio,
especificando um valor arbitrário (ou vazio) e um
MAX-AGE
atributo de
0.
15.12.3 IndexedDB
Web
A arquitetura de aplicativos tradicionalmente apresentava HTML, CSS,
e JavaScript no cliente e um banco de dados no servidor.Você pode encontrar
É surpreendente, portanto, aprender que a plataforma da web inclui um simples
Banco de dados de objeto com uma API JavaScript para armazenar persistentemente
JavaScript objeta no computador do usuário e recuperando -os como
necessário.
IndexedDB é um banco de dados de objetos, não um banco de dados relacional, e é
Muito mais simples que os bancos de dados que suportam consultas SQL.É mais
poderoso, eficiente e robusto que o armazenamento de chave/valor fornecido por
o
LocalStorage
, no entanto.Como o
LocalStorage
, Assim,

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 nomeado
Armazenamento de objetos
.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 (ver
"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
objeto deve ter um
chave
pelo 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
definir qualquer número de
índices
na loja 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

Erro ao traduzir esta página.

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
trabalhar 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
o
ONSUCCEST
e
OnError
propriedades.Dentro de um
ONSUCCEST
manipulador, o resultado da operação está disponível como o
resultado
propriedade
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
comprometer-se()
método.Mas com indexedDB, as transações são automaticamente
comprometido (se você não os abortar explicitamente) quando todo o
ONSUCCEST
Os manipuladores de eventos 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 ()
chamar.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
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
Nem todos,
das características básicas de
Indexeddb.
Exemplo 15-13
é longo, mas
bem
comentou.
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
(
ligar de volta
)

{

deixar

solicitar

=

indexeddb
.
abrir
(
"ZipCodes"
, Assim,

1
);

// Solicitação v1

do banco de dados
```

```
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.

solicitar
.
OnUpGradEneeded

=

()

=>

{

initdb
(
solicitar
.
resultado
, Assim,

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
```

```
// 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

buscar
(
"Zipcodes.json"
)

// Faça um http obter

solicitar

.
então
(
resposta

=>

resposta
.
JSON
())

// analisar o corpo

como JSON

.
então
(
ZipCodes

=>

{

// Obtenha o código postal de 40k

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
```

Erro ao traduzir esta página.

```
deixar

transação

=

dB
.
transação
([
"ZipCodes"
]);

deixar

loja

=

transação
.
ObjectStore
(
"ZipCodes"
);

// Desta vez, também obtemos o índice da cidade do objeto

loja

deixar

índice

=

loja
.
índice
(
"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.

deixar

solicitar
```

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
Trabalhador()
construtor, passando a
URL que especifica o código JavaScript que o trabalhador deve executar:
deixar

DatacRuncher

=

novo

Trabalhador
(
"Utils/Cruncher.js"
);
Se você especificar um URL relativo, ele será resolvido em relação ao URL do
documento que contém o script que chamou
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
PostMessage ()
vai ser
copiado usando o algoritmo de clone estruturado (ver
“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

=

função
(
e
)

{

deixar

estatísticas

=

e
.
dados
;

// A mensagem é a propriedade de dados

do evento

console
.
registro
(
`Média:
$ {
estatísticas
.
significar
}
`
);
}
```

Como todas as metas de eventos, os objetos do trabalhador definem o padrão addEventListener () e RemoneeventListener () métodos, e você pode usá -los no lugar do OnMessage .

Além de

O objeto Workerglobalscope tem um
PostMessage ()
método e
um
OnMessage
Propriedade do manipulador de eventos que são como os dos
Objeto de trabalhador, mas trabalhe na direção oposta: chamando
PostMessage ()
Dentro de um trabalhador, gera um evento de mensagem lá fora
o trabalhador e as mensagens enviadas de fora do trabalhador são transformadas em
eventos e entregue ao
OnMessage
manipulador.Porque o
Workerglobalscope é o objeto global para um trabalhador,
PostMessage ()
e
OnMessage
parece 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 um
nome
propriedade, então o valor de
essa propriedade se torna o valor do
nome
propriedade no trabalhador
objeto global.Um trabalhador pode incluir esse nome em qualquer mensagem
impressões com
console.warn ()
ou
console.error ()
.
O
fechar()
A função permite que um trabalhador se termine, e é
semelhante em vigor ao
finalizar ()
método 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, o
isnan ()
função e o
Data()
construtor.Em
Além disso, no entanto, o workerglobalscope também tem o seguinte
Propriedades do objeto de janela do lado do cliente:
auto
é uma referência ao próprio objeto global.
Workerglobalscope não é um objeto de janela e não
definir a
janela
propriedade.

Os métodos do timer
setTimeout ()
, Assim,
ClearTimeout ()
, Assim,
setInterval ()
, e
ClearInterval ()
.
UM
localização
propriedade que descreve o URL que foi
passou para o
Trabalhador()
construtor.Esta propriedade refere -se a um
Objeto de localização, exatamente como o
localização
propriedade de uma janela
faz.O objeto de localização tem propriedades
Href
, Assim,
protocolo
, Assim,
hospedar
, Assim,
nome do host
, Assim,
porta
, Assim,
Nome do caminho
, Assim,
procurar
, e
Hash
.
Em um trabalhador, essas propriedades são somente leitura, no entanto.
UM
navegador
propriedade que se refere a um objeto com
propriedades como as do objeto de navegador de uma janela.UM
O objeto de navegador do trabalhador tem as propriedades
AppName
, Assim,
AppVersion
, Assim,
plataforma
, Assim,
UserAgent
, e
on-line
.
Os métodos de destino de eventos usuais
addEventListener ()
e
RemoneeventListener ()
.
Finalmente, o objeto WorkerglobalsCope inclui um importante lado ao cliente
APIs de JavaScript, incluindo o objeto de console, o
buscar()

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 forma 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
ImportScripts ()
pode ser chamado
ImportScripts ()
para carregar os arquivos depende.Nota, no entanto,
que
ImportScripts ()
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
ImportScripts ()
Retornos: lá
não é necessário um retorno de chamada, manipulador de eventos,
então()
método ou
aguarde
.
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
Em
Para usar módulos nos trabalhadores, você deve passar um segundo argumento para o
Trabalhador()
construtor.
Este segundo argumento deve ser um objeto com um
tipo
Propriedade definida como a sequência "Módulo".Passando a
Tipo: "Módulo"
opção para o
Trabalhador()
construtor é muito parecido com o uso do
type = "Módulo"
atributo em um html
<Cript>
Tag: isso significa que o código deve ser interpretado como um módulo e

que
importar
Declarações são permitidas.
Quando um trabalhador carrega um módulo em vez de um script tradicional, o workerglobalscope não def
ine
o
ImportScripts ()
função.
Observe que, no início de 2020, o Chrome é o único navegador que suporta módulos verdadeiros e
importar
declarações em trabalhadores.
15.13.4 Modelo de execução do trabalhador
Trabalhador
Os threads 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 próprio chamando o global
fechar()
função.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
uma exceção ocorre em um trabalhador e não é pego por nenhum
pegar
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
PreventDefault ()
é chamado lá, então propagação
termina.Caso contrário, uma mensagem de erro é impressa no console do desenvolvedor
e o manipulador do OnError (
§15.1.7
) do objeto da janela é invocado.
// lida com erros de trabalhador não capturados com um manipulador dentro do

trabalhador.
auto
.
OnError

=

função
(
e
)

{

console
.
registro
(
`Erro no trabalhador em

$ {
e
.
nome do arquivo
}
:
$ {
e
.
Lineno
}
:
$ {
e
.
mensagem
}
`

);

e
.
PreventDefault
();
};
// ou, lide com erros de trabalhador não capturados com um manipulador do lado de fora

o trabalhador.
trabalhador
.

Os conjuntos não precisam ser inicializados quando você os cria.Você pode adicionar e remova os elementos a qualquer momento com
adicionar()
, Assim,
excluir()
, 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:
deixar

s

=

novo

Definir
();

// Comece vazio
s
.
tamanho

// => 0
s
.
adicionar
(
1
);

// Adicione um número
s
.
tamanho

// => 1;agora o conjunto tem um membro
s
.
adicionar
(
1
);

// Adicione o mesmo número novamente
s
.
tamanho

// => 1;O tamanho não muda
s
.
adicionar
(
verdadeiro
);

Erro ao traduzir esta página.

```
urgentport
.
addEventListener
(
"mensagem"
, Assim,

HandleurgentMessage
);
urgentport
.
começar
();

// 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
PostMessage ()
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 para
PostMessage ()
Inclui 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
PostMessage ()
argumento.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 ()

Lá
é outro caso de uso para o
PostMessage ()
Método em Cliente
JavaScript lateral.Envolve janelas em vez de trabalhadores, mas há
semelhanças suficientes entre os dois casos em que descreveremos o
PostMessage ()
Método do objeto de janela aqui.

Quando um documento contém um
<frame>
elemento, esse elemento age
como uma janela incorporada, mas independente.O objeto de elemento que
representa o
<frame>
tem um
ContentWindow
propriedade que é o
Objeto de janela para o documento incorporado.E para scripts em execução
Dentro daquele iframe aninhado, o
window.parent
propriedade refere -se ao
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 outro
janela.
Para
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
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
PostMessage ()
O método de uma janela é um pouco diferente de
o
PostMessage ()
Método 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
PostMessage ()
Método 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.
JavaScript
código executado dentro de uma janela ou
<frame>
pode receber
mensagens postadas nessa janela ou quadro definindo o
OnMessage
propriedade dessa janela ou ligando
addEventListener ()
para
Eventos de "mensagem".Como com os trabalhadores, quando você recebe uma "mensagem"
evento para uma janela, o
dados
propriedade do objeto de evento é o
mensagem que foi enviada.Além disso, no entanto, eventos de "mensagem"
entregue ao Windows também define
fonte
e
origem
propriedades.
O
fonte
Propriedade especifica o objeto de janela que enviou o evento,
e você pode usar
event.source.postMessage ()
Para enviar uma resposta.
O
origem
Propriedade 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

Esse
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 de imagem 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
em
Figura 15-16
.
Figura 15-16.
Uma parte do conjunto de Mandelbrot

seguido pelo nome do evento:
ONCLICK
, Assim,
OnChange
, Assim,
ONLOAD
, Assim,
OnMouseOver
, e assim por diante.Observe que esses nomes de propriedades são casos
sensível e são escritos em todas as minúsculas,
mesmo quando o tipo de evento
(como "Mousedown") consiste em várias palavras.O código a seguir
Inclui dois registros de manipuladores de eventos desse tipo:
// Defina a propriedade OnLar da Window para um

função.
// a função é o manipulador de eventos: é invocado quando o

Cargas de documentos.
janela
.
ONLOAD

=

função
()

{

// Procure um elemento <morm>

deixar

forma

=

documento
.
QuerySelector
(
"Formulário#frete"
);

// Registre uma função de manipulador de eventos no formulário que

será invocado

// Antes de o formulário ser enviado.Suponha que o isformValid () seja

definido em outro lugar.

forma
.
OnSubmit

=

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

=

função
(
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

{
telha
, Assim,

Erro ao traduzir esta página.

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

```
// 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

Fromurl
(
url
)

{

deixar

s

=

novo

Gamestate
();

deixar

params

=

novo

Url
(
url
).
SearchParams
;

s
.
baixo

=

parseint
(
params
.
pegar
(
```

```
/*
* 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.
*/
aula

Trabalhador

{

construtor
(
NumWorkers
, Assim,

Workersource
)

{

esse
.
trabalhadores ociosos

=

[];
```

```
Document.QuerySelector ("#Sect1")
deixar

Seção1

=

documento
.
getElementById
(
"Sect1"
);
// Procure todos os elementos (como caixas de seleção de formulário) que têm um

nome = "cor"
// atributo.Semelhante ao document.QuerySelectorAll ('*
[name = "color"] ');
deixar

cores

=

documento
.
getElementsByName
(
"cor"
);
// Procure todos os elementos <H1> no documento.
// semelhante ao document.QuerySelectorall ("H1")
deixar

títulos

=

documento
.
getElementsByTagName
(
"H1"
);
// getElementsByTagName () também é definido em elementos.
// Obtenha todos os elementos <H2> dentro do elemento SECT1.
deixar

subtítulos

=

Seção1
.
getElementsByTagName
(
"H2"
);
// Procure todos os elementos que têm a classe "ToolTip".
```

```
se

(
esse
.
trabalhadores ociosos
.
comprimento

>

0
)

{

deixar

trabalhador

=

esse
.
trabalhadores ociosos
.
pop
();

esse
.
Workermap
.
definir
(
trabalhador
, Assim,

[[
resolver
, Assim,

rejeitar
]);

trabalhador
.
Postmessage
(
trabalhar
);

}

outro

{

esse
```

```
}

// Este método de fábrica obtém estado de um URL ou retorna

nulo se

// Um ■■estado válido não pôde ser lido no URL.

estático

Fromurl
(
url
)

{

deixar

s

=

novo

Pagestate
();

deixar

u

=

novo

Url
(
url
);

// inicialize o estado do

Params de pesquisa da URL.

s
.
cx

=

parsefloat
(
u
.
SearchParams
.
pegar
(
```

Erro ao traduzir esta página.

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

```
f
(
esse
.
estado
);

}

outro

{

para
(
deixar

propriedade

em

f
)

{

esse
.
estado
[[
propriedade
]

=

f
[[
propriedade
];

}

}

// Em ambos os casos, comece a renderizar o novo estado o mais rápido possível.

esse
.
renderizar
();

// Normalmente, salvamos o novo estado.Exceto quando estamos

chamado com

// um segundo argumento de false que fazemos quando obtemos um

Evento PopState.
```

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.

deixar

{
cx
, Assim,

cy
, Assim,

perpixel
, Assim,

Maxiterations
}

=

esse
.
estado
;

deixar

x0

=

cx

-

perpixel

*

esse
.
largura
/
2
;

deixar

Erro ao traduzir esta página.

que
importar
Declarações são permitidas.
Quando um trabalhador carrega um módulo em vez de um script tradicional, o workerglobalscope não def
ine
o
ImportScripts ()
função.
Observe que, no início de 2020, o Chrome é o único navegador que suporta módulos verdadeiros e
importar
declarações em trabalhadores.
15.13.4 Modelo de execução do trabalhador
Trabalhador
Os threads 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 próprio chamando o global
fechar()
função.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
uma exceção ocorre em um trabalhador e não é pego por nenhum
pegar
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

.
pegar
((
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
.
erro
(
"Promise rejeitada em render ():"
, Assim,

razão
);

})

.
finalmente
(()

=>

{

// Quando terminamos de renderizar, limpe o

Bandeiras pendentes

esse
.
Pendente

=

nulo
;

// e se os pedidos de renderização chegaram enquanto estávamos

Ocupado, reproduzido agora.

se

```
}

// 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
)

{

trocar
(
evento
.
chave
)

{

caso

"Escapar"
:

// tipo de fuga para voltar ao

estado inicial

esse
.
setState
(
Pagestate
.
InitialState
());

quebrar
;

caso

"+"
:

// tipo + para aumentar o número de

iterações
```

```
s
.
perpixel
);

quebrar
;

caso

"Arrowright"
:

// seta direita para rolar a direita

esse
.
setState
(
s

=>

s
.
cx

+=

esse
.
largura
/
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
```

```
`tradução (
$ {
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.

esse
.
tela
.
RemovaeventListener
(
"Pointermove"
, Assim,

PointermoveHandler
);

esse
.
tela
.
RemovaeventListener
(
"Ponterup"
, Assim,

PointerupHandler
);
```

```
esse
.
tela
.
estilo
.
transformar

=

`tradução (
$ {
-
CDX
*
2
}
px,
$ {
-
Cdy
*
2
}
px)

Escala (2) `
;

// Defina as coordenadas complexas do novo

ponto central e

// zoom por um fator de 2.

esse
.
setState
(
s

=>

{

s
.
cx

+=

CDX

*

s
.
perpixel
;
```

15.15 Resumo e sugestões para Leitura adicional

Esse Capítulo longo cobriu os fundamentos do lado do cliente JavaScript programação:

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 html <Canvas> elemento.

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

O
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
Uma vez
Você escreveu um aplicativo da web e o lançou 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.O
desempenho
propriedade de
O objeto da janela é o ponto de entrada principal para esta API.Inclui a
Fonte de tempo de alta resolução
performance.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
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
Esse
Capítulo apresentou 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
Contente-
Política de segurança
Cabeçalho HTTP e compreensão de 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

O
Os objetos de janela e documento têm vários recursos que
não foram cobertos neste capítulo:
O objeto da janela define
alerta()
, Assim,
confirmar()
, e
incitar()
métodos que exibem diálogos modais simples para
o usuário.Esses métodos bloqueiam o encadeamento principal.O

confirmar()
Método retorna de forma síncrona um valor booleano,
e
incitar()
Retorna de síncrona uma sequência de entrada do usuário.
Estes não são adequados para uso da produção, mas podem ser úteis para
Projetos e protótipos simples.
O
navegador
e
tela
propriedades 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
solicitaçãofullcreen ()
Método de qualquer elemento
Objeto solicita que esse elemento (A
<Video>
ou
<Canvas>
elemento, por exemplo) ser exibido no modo de tela cheia.O
ExitfullScreen ()
Método do documento retorna para
Modo de exibição normal.
O
requestanimationframe ()
Método 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 do sistema
Praça de transferência
para
Ativar interações de cópia e colar com aplicativos fora
do navegador.
Um recurso pouco conhecido dos navegadores da web é que html

Erro ao traduzir esta página.

Erro ao traduzir esta página.

Erro ao traduzir esta página.

notificação.Usar esta API é complicada pelo fato de você
deve primeiro solicitar a permissão do usuário para exibir
notificações.
O
Push API 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
Lá
são 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.)
O
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

Navigator.vibrate ()
Método 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.
O
A API de rastreação 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.
O
"Eventos" Devicemotion "e" Deviceroentation "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.)
O
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
Digitado
Matrizes, matrizes e a classe DataView (todos cobertos em
§11.2
) Permitir que o JavaScript trabalhe com dados binários.Conforme descrito anteriormente
Neste capítulo, o
buscar()
API 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 = "arquivo>
elemento de forma) 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,

Erro ao traduzir esta página.

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
estão disponíveis apenas para o código JavaScript em execução
dentro de documentos que foram carregados em uma conexão HTTPS segura.
O
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.
O
API de solicitação de pagamento adiciona suporte ao navegador para fazer cartão de crédito
pagamentos na web.Ele permite que os usuários armazenem seus detalhes de pagamento
com segurança no navegador para que eles não precisem digitar seu cartão de crédito
Número cada vez que eles fazem uma compra.Aplicativos da Web que desejam
solicitar um pagamento criar um objeto de pagamento de pagamento e chamar seu
mostrar()
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.

Erro ao traduzir esta página.

Capítulo 16.
Lado do servidor
JavaScript com nó

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.
O
Definindo recurso do nó é o seu evento único baseado em eventos
simultaneidade ativada por um
API assíncrona por padrão.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
Nó
é software de código aberto.
Visita
https://nodejs.org
Para baixar e instalar o nó para Windows e
Macos.No Linux, você poderá instalar o nó com o seu gerenciador de pacotes normal, ou você pode
visita
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 nest
e
O capítulo usará apenas os pacotes internos do Node e não exigirá NPM ou bibliotecas externas.
Finalmente, 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
Bem
Comece 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 são que
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
.
registro
(
"Olá mundo!"
);
Existem maneiras de escrever em nível mais baixo, mas sem mais sofisticadas
maneira oficial do que simplesmente ligar
console.log ()
.
Em navegadores da web,
console.log ()
, Assim,
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
console.error ()
para
Exibir texto no console onde o usuário o verá, mesmo que o texto
impresso com
console.log ()
está escondido.
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 nnambientais.
O nó segue essas convenções Unix.Um programa de nós pode ler seu
Argumentos da linha de comando da matriz de strings
process.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
.
registro
(
processo
.
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 de
process.argv
vai ser
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 aparece em
process.argv
.(O
-
rastreio
O argumento da linha de comando 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
) isso aparece
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 embora o
process.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
Nó -h
ou
Nó -Help
Para descobrir o que
-p
e
-e
Argumentos da linha de comando fazem.No entanto, como uma dica, observe que você
poderia reescrever a linha acima como
Node --eval 'process.env' -
-imprimir
.

16.1.3 Ciclo de vida do programa

O
nó
comando
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 não
pegar
cláusula
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:
processo
.
setuncaughtexceptionCaptureCallback
(
e

=>

{

console
.
erro
(
"Exceção não capturada:"
, Assim,

e
);
});
Uma situação semelhante surge se uma promessa criada pelo seu programa for rejeitado e não há
.pegar()
invocação 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:
processo
.
sobre
(
"Rejeição não combinada"
, Assim,

(
razão
, Assim,

promessa
)

=>

{

// 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
documentado
Sistemas de módulos JavaScript, 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 o
requer ()
função para importar valores para um
módulo e o
exportações
objeto ou o
Module.Exports
propriedade
para exportar valores de um módulo.Estes são uma parte fundamental do
Modelo de programação do nó, e eles são cobertos em detalhes em
§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 estará usando
requer ()
e
Module.Exports
ou se isso
estará usando
importar
e
exportar
.Quando o nó carrega um arquivo de

Identificadores extras como
exigir
, Assim,
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 vai
Carregue -o como um módulo ES6, esperará que ele use
importar
e
exportar
, Assim,
e não fornecerá um
requer ()
função.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 um
requer ()
função e vontade
Jogue um SyntaxError se você usar
importar
ou
exportar
declarações.
Para arquivos que não têm um explícito
.mjs
ou
.CJS
Extensão, nó parece
Para um arquivo nomeado
package.json
no mesmo diretório que o arquivo e depois
em cada um dos diretórios que contêm.Uma vez mais próximo
package.json
arquivo
é encontrado, verifica o nó para um nível superior
tipo
Propriedade no JSON
objeto.Se o valor do
tipo
A propriedade é "módulo", depois as cargas do nó
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
package.json
Arquivo 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

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 nomeado
package.json
no diretório raiz do seu
projeto.Esse
package.json
Arquivo criado por NPM é onde você adicionaria
"Tipo": "módulo"
Se você quisesse usar módulos ES6 para o seu
projeto.
Este capítulo não cobre o NPM com detalhes (mas veja
§17.4
por 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 inicial
package.json
arquivo com base no seu
respostas.
Agora, para começar a usar o Express, você digitaria
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 local
node_modules/
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 expresso - no
package.json
arquivo.Com
Esta dependência registrada em
package.json
, você poderia dar outro
programador uma cópia do seu código e seu
package.json
, e eles
poderia simplesmente digitar
NPM Instale
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
Nó
Os programas podem executar vários processos do sistema operacional e o nó 10 e o trabalhador de sup
orte posterior
objetos (
§16.11
), que são um tipo de tópico emprestado de 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 intensiv
a
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 típic
a de
Programação multithreaded porque a comunicação de interprocesso e trabalho de trabalho é via mensage
m
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
ainda leia ou já esqueci
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-
Primeiros retornos de chamada
, que geralmente 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
readfile ()
função 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

=

exigir
(
"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
, Assim,

ligar de volta
)

{

fs
.
ReadFile
(
caminho
, Assim,

"UTF8"
, Assim,

(
errar
, Assim,

texto
)

=>

{

se

(
errar
)

{

// algo deu errado lendo o

arquivo

console
.
erro
(
errar
);

ligar de volta
(
```

retornar

JSON
.
analisar
(
texto
);

});
}
Também podemos simplificar a função baseada em promessa anterior usando
assíncrono
e
aguarde
(novamente, se você ainda não leu
Capítulo 13
, este seria um bom momento para fazê -lo):
assíncrono

função

ReadConfigfile
(
caminho
)

{

deixar

texto

=

aguarde

PFS
.
ReadFile
(
caminho
, Assim,

"UTF-8"
);

retornar

JSON
.
analisar
(
texto
);
}
O
util.promisify ()
O invólucro pode produzir uma promessa baseada em promessa

este código e escreva uma versão puramente síncrona do nosso
readConfigfile ()
função.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

=

exigir
(
"FS"
);
função

ReadConfigFilesync
(
caminho
)

{

deixar

texto

=

fs
.
readfilesync
(
caminho
, Assim,

"UTF-8"
);

retornar

JSON
.
analisar
(
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

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 ret
orno 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

Erro ao traduzir esta página.

"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:

```
deixar

b

=

Buffer
.
de
([
0x41
, Assim,

0x42
, Assim,

0x43
]);

// <buffer

41 42 43>
b
.
ToString
()

// =>

"ABC";padrão "utf8"
b
.
ToString
(
"Hex"
)

// =>

"414243"
deixar

computador

=

Buffer
```

deixar

morto

=

Buffer
.
aloc
(
1024
, Assim,

"Deadbeef"
, Assim,

"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.
morto
.
READUINT32BE
(
0
)

// => 0xDeadBeef
morto
.
READUINT32BE
(
1
)

// => 0xadbeefde
morto
.
readbiguint64be
(
6
)

// => 0xbeefdeadbeefdeadn
morto
.
READUINT32LE
(
1020
)

// => 0xefbeadde
Se você escrever um programa de nós que realmente manipula dados binários, você

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 de
EventEmitter:
const

EventEmitter

=

exigir
(
"Eventos"
);

// O nome do módulo faz

NOM DO MAIXO CLASSE NOME
const

líquido

=

exigir
(
"líquido"
);
deixar

servidor

=

novo

líquido
.
Servidor
();

// Crie um servidor

objeto
servidor

Instância de

EventEmitter

// => true: servidores

são os EventEmitters
A principal característica dos observadores de eventos é que eles permitem que você se registre
manipulador de eventos funciona com o

dados

// para o cliente e desconecte.

soquete
.
fim
(
"Hello World"
, Assim,

"UTF8"
);
});
Se você preferir nomes de métodos mais explícitos para registrar o evento
ouvintes, você também pode usar
addListener ()
.E você pode remover um
Lista de eventos registrado anteriormente com
desligado()
ou
Removelistener ()
.
Como um caso especial, você pode registrar um evento
ouvinte que será removido automaticamente após ser acionado para o
Primeira vez ligando
uma vez()
em vez de
sobre()
.
Quando um evento de um determinado tipo ocorre para um determinado evento de eventos
Objeto, o nó invoca todas as funções do manipulador que atualmente são
Registrado nessa ordem de eventos para eventos desse tipo.Eles são
Invocado em ordem desde o primeiro registrado até o último registrado.Se lá
é mais de uma função de manipulador, eles são invocados sequencialmente em um
Tópico único: não há paralelismo no nó, lembre -se.E,
É importante ressaltar que as funções de manuseio de eventos são invocadas de forma síncrona, não
assíncrono.O que isso significa é que o
emitir()
o método não
Os manipuladores de eventos na fila para serem invocados em algum momento posterior.
emitir()
invoca todos os manipuladores registrados, um após o outro, e não
Volte até que o último manipulador de eventos retorne.
O que isso significa, na verdade, é que quando uma das APIs de nós embutidos
Emite um evento, que a API está basicamente bloqueando seus manipuladores de eventos.Se
Você escreve um manipulador de eventos que chama uma função de bloqueio como
fs.readfilesync ()
, nenhum manuseio adicional de eventos acontecerá até
Sua leitura de arquivo síncrono está concluída.Se o seu programa é um - como um
servidor de rede - isso precisa ser responsivo, é importante que

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 ()
(ver
§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
emitir()
método 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.
emitir()
deve ser
Invocado com o nome do tipo de evento como seu primeiro argumento.Qualquer
argumentos adicionais que são passados ■■para
emitir()
tornar -se argumentos para
as funções do manipulador de eventos registradas.As funções do manipulador também são
invocado com o
esse
valor definido para o próprio objeto de EventEmitter,
o que geralmente é conveniente.(Lembre -se, porém, essa flecha funciona
sempre use o
esse
valor do 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
o
emitir()
Ligue 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 e
le
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
emitir()
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 a
pegar
bloco, então esse tipo de erro normalmente causa o seu
programa para sair.
16.5 fluxos
Quando
Implementando 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

=

exigir
(
"FS"
);
// um assíncrono, mas semamaming (e, portanto,

Função ineficiente).
função

CopyFile
(
Sourcefilename
, Assim,

DestinationFilename
, Assim,

ligar de volta
)

{

fs
.
ReadFile
(
Sourcefilename
, Assim,

(
errar
, Assim,
```

Erro ao traduzir esta página.

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 devolvidos 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
gzip
algoritmo) os dados escritos para ele.De maneira semelhante, o
cripto.createcipheriv ()
função 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
SetEncoding ()
Método 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

acomodar os novos dados.Com um modelo de simultaneidade baseado em eventos,
No entanto, as chamadas de bloqueio não fazem sentido, e as APIs de fluxo do Node
são baseados em eventos e retorno de chamada.Ao contrário de outras APIs de nó, não há
Versões “sincronizadas” dos métodos que serão descritos mais adiante neste
capítulo.
A necessidade de coordenar a legibilidade do fluxo (buffer não vazio) e
Writability (buffer não cheio) via eventos torna as APIs de fluxo do Node
um pouco complicado.Isso é agravado pelo fato de que essas APIs
evoluíram e mudaram ao longo dos anos: para fluxos legíveis, lá
são duas APIs completamente distintas que você pode usar.Apesar do
Complexidade, vale a pena entender e dominar o streaming do nó
APIs porque eles permitem a E/S de alto rendimento em seus programas.
As subseções a seguir demonstram como ler e escrever de
Classes de fluxo do nó.

16.5.1 Tubos

Às vezes, você
precisa ler dados de um fluxo simplesmente para mudar
e escreva os mesmos dados para outro fluxo.Imagine, por exemplo, que
Você está escrevendo um servidor HTTP simples que serve um diretório de estática
arquivos.Nesse caso, você precisará ler dados de um fluxo de entrada de arquivo
e escreva em um soquete de rede.Mas em vez de escrever o seu próprio
código para lidar com a leitura e a escrita, você pode simplesmente se conectar
os dois soquetes juntos como um "tubo" e deixam o nó lidar com o
complexidades para você.Basta passar o fluxo gravável para o
cano()
Método do fluxo legível:

```
const

fs

=

exigir
(
"FS"
);
```

```
função

PipeFileToSocket
(
nome do arquivo
, Assim,

soquete
)

{

fs
.
Createradstream
(
nome do arquivo
).
cano
(
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:

```
função

cano
(
legível
, Assim,

gravável
, Assim,

ligar de volta
)

{

// Primeiro, configure o manuseio de erros

função

HandleError
(
errar
)

{

legível
.
fechar
();

gravável
.
fechar
```

Figura 15-14.
Um efeito desfoque de movimento criado pelo processamento de imagem
O código a seguir demonstra
getImagedata ()
e
PutImagedata ()
e mostra como iterar e modificar o
Valores de pixel em um objeto imagedata.
Exemplo 15-8.
Motion Blur com IMAGEDATA
// mancha os pixels do retângulo à direita, produzindo um
// tipo de movimento borrado como se os objetos estivessem se movendo da direita para

esquerda.
// N deve ser 2 ou maior.Valores maiores produzem manchas maiores.
// O retângulo é especificado no sistema de coordenadas padrão.
função

mancha
(
c
, Assim,

n
, Assim,

x
, Assim,

y
, Assim,

c
, Assim,

h
)

{

// Obtenha o objeto IMagedata que representa o retângulo

de pixels para manchar

deixar

pixels

=

c
.
getImagedata
(
x
, Assim,

y
, Assim,

```
o fluxo deve

// só pode ser conectado a fluxos legíveis que tiveram

// setEncoding () chamou neles.

_transformar
(
pedaço
, Assim,

codificação
, Assim,

ligar de volta
)

{

se

(
typeof

pedaço

! ==

"corda"
)

{

ligar de volta
(
novo

Erro
(
"Esperava uma string, mas recebia um

buffer "
));

retornar
;

}

// Adicione o pedaço a qualquer linha anteriormente incompleta

e quebrar

// tudo em linhas

deixar

linhas
```

```
ligar de volta
(
nulo
, Assim,

esse
.
UncompleteLine

+

"\ n"
);

}

}
}
// Agora podemos escrever um programa como 'Grep' com esta classe.
deixar

padrão

=

novo

Regexp
(
processo
.
argv
[[
2
]);

// Obtenha um regexp

da linha de comando.
processo
.
stdin

// comece com

entrada padrão,

.
SetEncoding
(
"UTF8"
)

// leia como

Strings Unicode,

.
cano
```

```
fluxo.
assíncrono

função

grep
(
fonte
, Assim,

destino
, Assim,

padrão
, Assim,

codificação
=
"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)

destino
.
sobre
(
"erro"
, Assim,

errar

=>

processo
.
saída
());

// Os pedaços que lemos provavelmente terminam com uma nova linha,
```

```
grep
(
processo
.
stdin
, Assim,

processo
.
stdout
, Assim,

padrão
)

// Ligue para o

função assíncrona grep ().

.
pegar
(
errar

=>

{

// Lidar

exceções assíncronas.

console
.
erro
(
errar
);

processo
.
saída
();

});
```

16.5.3 Escrevendo para fluxos e manuseio Backpressure

O assíncrono grep () função 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 para o escrever() método.O escrever() 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.)

erros.)
O
escrever()
O método tem um valor de retorno muito importante.Quando você
chamar
escrever()
Em um riacho, ele sempre aceitará e amortece o pedaço
de dados que você passou.Então retorna
verdadeiro
Se o buffer interno for
ainda não está cheio.Ou, se o buffer agora estiver cheio ou muito cheio, ele retorna
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
falso
do
escrever()
Método 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 ligar
escrever()
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

escrever
(
fluxo
, Assim,

pedaço
, Assim,

ligar de volta
)

{

// Escreva o pedaço especificado para o fluxo especificado

deixar

HasMoreroom

=

fluxo
.
escrever

}
}
O fato de às vezes ser bom ligar
escrever()
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
cano()
O método é tão atraente: quando você usa
cano()
, Nó
Lida com a contrapressão para você automaticamente.
Se você está usando
aguarde
e
assíncrono
no seu programa e estão tratando
Fluxos legíveis como iteradores assíncronos, é direto
implementar uma versão baseada em promessa do
escrever()
função de utilidade
acima para manipular corretamente a contrapressão.No assíncrono
grep ()
função
Nós apenas olhamos, não lidamos com a contrapressão.O assíncrono
cópia()
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 chamada
cópia (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

escrever
(
fluxo
, Assim,

pedaço
)

{

// Escreva o pedaço especificado para o fluxo especificado

deixar

HasMoreroom

retornar

novo

Promessa
(
resolver

=>

{

// De outra forma,

devolver uma promessa que

fluxo
.
uma vez
(
"ralo"
, Assim,

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).
assíncrono

função

cópia
(
fonte
, Assim,

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)

Erro ao traduzir esta página.

Manuseie a contrapressão do fluxo gravável.Se o
escrever()
Retorna de método
falso
Para indicar que o buffer de gravação está cheio, você pode
chamar
pausa()
no fluxo legível para parar temporariamente
dados
eventos.Então, quando você obtém um evento de "dreno" do fluxo gravável,
você pode ligar
retomar()
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 prometia uma versão melhor para vir.O
O código a seguir 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 um
cano()
ligue, mas isso
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

=

exigir
(
"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
, Assim,

DestinationFilename
, Assim,

ligar de volta
)

```
entrada
.
sobre
(
"dados"
, Assim,

(
pedaço
)

=>

{

// Quando obtivemos novos

dados,

deixar

Hasroom

=

saída
.
escrever
(
pedaço
);

// Escreva para o

fluxo de saída.

se

(
!
Hasroom
)

{

// se a saída

o fluxo está cheio

entrada
.
pausa
();

// então pausa o

fluxo de entrada.

}
```

```
console
.
registro
(
`Copiando arquivo
$ {
de
}
para
$ {
para
}
... `
);
CopyFile
(
de
, Assim,

para
, Assim,

errar

=>

{

se

(
errar
)

{

console
.
erro
(
errar
);

}

outro

{

console
.
registro
(
"feito."
);

}
});
Modo pausado
```

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
ler()
retorna
nulo
ou
escrever()
retorna
falso
, e então começar a ler ou escrever novamente em um
legível
ou
ralo
evento.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

=

exigir
(
"FS"
);
const

cripto

=

exigir
(
"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
, Assim,

ligar de volta
)

As funções próprias do construtor para inicializar objetos recém -criados.Fazendo isso está coberto em
Capítulo 9
.

6.2.3 Protótipos

Antes
Podemos 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 a
protótipo
, e o primeiro objeto herda as propriedades do
protótipo.
Todos
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 o
novo
palavra -chave e a
Invocação do construtor Use o valor do
protótipo
propriedade de
o construtor funciona como seu protótipo.Então o objeto criado por
novo objeto ()
herda de
Object.prototype
, assim como o
objeto criado por
{}
faz.Da mesma forma, o objeto criado por
novo
Variedade()
usos
Array.prototype
como seu protótipo e o objeto
criado por
nova data ()
usos
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 um
protótipo
propriedade.São esses objetos com
protótipo
propriedades que definem o
protótipos
para todo o outro
objetos.
Object.prototype
é um dos objetos raros que não tem protótipo:
Não herda nenhuma propriedade.Outros protótipos objetos são normais objetos que têm um protótipo.A maioria dos construtores embutidos (e a maioria

exemplo.
processo
.
cwd
()

// retorna o funcionamento atual

diretório.
processo
.
chdir
()

// define o funcionamento atual

diretório.
processo
.
cpuusage
()

// relata o uso da CPU.
processo
.
Env

// um objeto de ambiente

variáveis.
processo
.
EXECPATH

// o caminho absoluto do sistema de arquivos para

o nó executável.
processo
.
saída
()

// encerra o programa.
processo
.
EXITCODE

// um código inteiro a ser relatado

Quando o programa sai.
processo
.
getuid
()

// retorna o ID do usuário do Unix do

usuário atual.
processo

carregado com
requer ()
) fornece acesso a um 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

=

exigir
(
"OS"
);
OS
.
arco
()

// Retorna a arquitetura da CPU."x64" ou

"Arm", por exemplo.
OS
.
constantes

// constantes úteis, como

OS.CONSTANTS.Signals.SIGINT.
OS
.
CPUs
()

// dados sobre núcleos de CPU do sistema,

incluindo tempos de uso.
OS
.
Endianness
()

// A Endianness nativa da CPU "be" ou

"Le".
OS
.
EOL

// O Terminador de linha nativo do sistema operacional: "\ n"

ou "\ r \ n".
OS
.
Freemem
()

OS
.
tempo de atividade
()

// retorna o tempo de atividade do sistema em

segundos.
OS
.
userInfo
()

// retorna uid, nome de usuário, casa e

shell do usuário atual.
16.7 trabalhando com arquivos
Nó
O módulo "FS" é uma API abrangente para trabalhar com arquivos e
diretórios.É complementado pelo módulo "caminho", que define
Funções de utilidade para trabalhar com nomes de arquivos e diretórios.O "fs"
O módulo contém um punhado de funções de alto nível para leitura facilmente,
Escrever e copiar arquivos.Mas a maioria das funções do módulo são
ligações JavaScript de baixo nível para chamadas do sistema UNIX (e seus
equivalentes no Windows).Se você trabalhou com baixo nível
chamadas de sistema de arquivos antes (em c ou outro
idiomas),
Então a API do nó
será familiar para você.Caso contrário, você pode encontrar partes da API "FS" para ser
conciso e não intuitivo.A função para excluir um arquivo, por exemplo, é
chamado
desvincular ()
.
O módulo "FS" define uma API grande, principalmente porque geralmente existem
Variantes múltiplas de cada operação fundamental.Conforme discutido no
início do capítulo, a maioria das funções como
fs.readfile ()
são sem bloqueio, baseados em retorno de chamada e assíncronos.Normalmente, porém,
Cada uma dessas funções tem uma variante de bloqueio síncrona, como
fs.readfilesync ()
.No nó 10 e mais tarde, muitos deles
As funções também têm uma variante assíncrona baseada em promessa, como
fs.Promeses.readfile ()
.A maioria das funções "fs" toma uma string como
seu primeiro argumento, especificando o caminho (nome do arquivo mais opcional
nomes de diretórios) para o arquivo que deve ser operado.Mas um número de
Essas funções também suportam uma variante que leva um número inteiro “arquivo

descritor ”como o primeiro argumento em vez de um caminho.Essas variantes têm
Nomes que começam com a letra "f".Por exemplo,
fs.truncato ()
trunca um arquivo especificado por caminho e
fs.ftruncato ()
truncados a
arquivo especificado pelo descritor de arquivo.Há uma promessa baseada
fs.Promeses.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

Em
para usar o módulo "FS" para trabalhar com arquivos, primeiro você 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, mais o
Hierarquia de diretórios nos quais o arquivo aparece.Se um caminho é
absoluto
, Assim,
Isso significa que os diretórios até a raiz do sistema de arquivos estão
especificado.Caso contrário, o caminho é
parente
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
../
Os 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
processo
.
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

=

exigir
(
"caminho"
);
caminho
.
set

//, "/" ou "\"

Dependendo do seu sistema operacional
// O módulo de caminho tem funções de análise simples
deixar

p

=

"Src/pkg/test.js"
;

// Um nnexemplo de caminho
caminho
.
nome de base
(
p
)
```

caminho
.
resolver
(
"/um"
, Assim,

"/b"
, Assim,

"T.JS"
)

// => "/b/t.js"
Observe que
Path.Normalize ()
é simplesmente uma manipulação de cordas
função que não tem acesso ao sistema de arquivos real.O
fs.realpath ()
e
fs.realpathsync ()
funções executadas
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
SO baseado em UNIX e
Path.Sep
é "/."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
definir as mesmas propriedades e funções que
caminho
em si.
Algumas das funções "fs" que abordaremos nas próximas seções
Espere a
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
fs.opensync ()
) função.Os processos só podem ter um
Número limitado de arquivos abertos ao mesmo tempo, por isso é importante que você ligue
fs.close ()
Nos seus descritores de arquivos quando você terminar com eles.
Você precisa abrir arquivos se quiser usar o nível mais baixo
fs.read ()
e
fs.write ()

Erro ao traduzir esta página.

```
// lide com o erro aqui

}

outro

{

// Os bytes do arquivo estão em buffer

}
});
// Leia assíncrona baseada em promessa
fs
.
promessas

.
ReadFile
(
"Data.csv"
, Assim,

"UTF8"
)

.
então
(
ProcessfileText
)

.
pegar
(
Handlereaderror
);
// ou use a API de promessa com aguardar dentro de uma função assíncrona
assíncrono

função

ProcessText
(
nome do arquivo
, Assim,

codificação
=
"UTF8"
)

{

deixar

texto

=
```

esse.
Definindo o
location.hash
Atualiza a propriedade do URL
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
location.hash
explicitamente, um evento "hashchange"
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
Retorne ao seu URL anterior antes de definir
location.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 "hashcrange"
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.

Erro ao traduzir esta página.

fs.WriteSync ()
funções.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 cordas usando
fs.promises.open ()
e o
Objeto FileHandle que produz.
Strings de modo de arquivo
Nós
vi o
fs.open ()
e
fs.opensync ()
métodos antes quando usam 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 qu
e 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

console
.
registro
(
"Olá mundo!"
);
Existem maneiras de escrever em nível mais baixo, mas sem mais sofisticadas
maneira oficial do que simplesmente ligar
console.log ()
.
Em navegadores da web,
console.log ()
, Assim,
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
console.error ()
para
Exibir texto no console onde o usuário o verá, mesmo que o texto
impresso com
console.log ()
está escondido.
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 nnambientais.
O nó segue essas convenções Unix.Um programa de nós pode ler seu
Argumentos da linha de comando da matriz de strings
process.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

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
.
promessas
.
CopyFile
(
"Dados importantes"
, Assim,

`Dados importantes
$ {
novo

Data
().
ToisSotring
()
}
"
fs.constants.copyfile_excl |

fs.constants.copyfile_ficlone)
.then (() => {
console.log ("backup completo");
});
.catch (err => {
console.error ("backup falhou", err);
});
O
fs.rename ()
função (junto com o síncrono usual 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 co
mo o
Terceiro argumento:
fs
.
renamesync
```

As funções
fs.link ()
e
fs.symlink ()
e suas variantes
tem as mesmas assinaturas que
fs.rename ()
e se comportar algo
como
fs.copyfile ()
exceto que eles criam links rígidos e simbólicos
links, respectivamente, em vez de criar uma cópia.
Finalmente,
fs.unlink ()
, Assim,
fs.unlinksync ()
, e
fs.promises.unlink ()
são as funções do Node 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
.
UnbinkSync
(
"Backups/ch15.bak"
);
16.7.5 Metadados do arquivo
O

fs.stat ()
, Assim,
fs.statsync ()
, e
fs.Promeses.stat ()
As funções permitem obter metadados para um arquivo ou diretório especificado.
Por exemplo:
const

fs

=

exigir
(
"FS"
);
deixar

estatísticas

=

fs
.
STATSYNC
(

```
/*
* 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.
*/
aula

Trabalhador

{

construtor
(
NumWorkers
, Assim,

Workersource
)

{

esse
.
trabalhadores ociosos

=

[];
```

A promessa devolvida por
buscar()
resolve para um objeto de resposta.O
status
A propriedade 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, o
OK
propriedade de uma resposta é
verdadeiro
se
status
é 200 ou qualquer código entre 200 e 299 e é
falso
Para qualquer outro código.
buscar()
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.O
cabeçalhos
propriedade de um objeto de resposta
é um objeto de cabeçalhos.Use seu
tem()
método para testar a presença de um
cabeçalho ou usar seu
pegar()
Método para obter o valor de um cabeçalho.Http
Os nomes dos cabeçalhos são insensíveis a maiúsculas, para que você possa passar em minúsculas ou mi
stas
Nomes de cabeçalho de casos para essas funções.
O objeto de cabeçalhos também é iterável se você precisar fazer isso:

```
buscar
(
url
).
então
(
resposta

=>

{

para
(
deixar

[[
nome
, Assim,
valor
]

de
```

```
// 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 (), Assim, 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: deixar

```
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
.
promessas
.
readdir
(
"/tmp"
, Assim,

{
com filetipos
```

E se você omitir o argumento de retorno de chamada, ele retornará uma promessa.Quando
Não há mais entradas de diretório, você receberá
nulo
em vez de um solo
objeto.
A maneira mais fácil de usar objetos dir é como iteradores assíncronos com um
para/aguardar
laço.Aqui, por exemplo, é uma função que usa o
API de streaming para listar entradas de diretório, chamadas
stat ()
em cada entrada,
e impressões de nomes de arquivos e diretórios
e tamanhos:

```
const

fs

=

exigir
(
"FS"
);
const

caminho

=

exigir
(
"caminho"
);
assíncrono

função

ListDirectory
(
Dirpath
)

{

deixar

dir

=

aguarde

fs
.
promessas
.
opendir
(
Dirpath
```

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.
O
getjson ()
função no final de
§13.2.6
usou o
http.get ()
função como parte de uma demonstração do
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 seguir
postjson ()
função demonstra como fazer
usar
https.request ()
Para fazer um https post solicitar que
Inclui um corpo de solicitação JSON.Como o
getjson ()
função de
Capítulo 13
, espera uma resposta JSON e retorna uma promessa que
atende à versão analisada dessa resposta:

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
ainda leia ou já esqueci
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-
Primeiros retornos de chamada
, que geralmente 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
readfile ()
função 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

=

exigir
(
"FS"
);

// requer o módulo do sistema de arquivos
// Leia um arquivo de configuração, analisou seu conteúdo como JSON e passe
```

```
// agora crie a solicitação com base na configuração

objeto

deixar

solicitar

=

https
.
solicitar
(
requestOptions
);

// Escreva o corpo da solicitação de postagem e termine o

solicitar.

solicitar
.
escrever
(
BodyText
);

solicitar
.
fim
();

// falha nos erros de solicitação (como nenhuma rede

conexão)

solicitar
.
sobre
(
"erro"
, Assim,

e

=>

rejeitar
(
e
));

// lide com a resposta quando começar a chegar.

solicitar
.
sobre
(
```

```
// e agora lida com a resposta quando estiver

completo.

resposta
.
sobre
(
"fim"
, Assim,

()

=>

{

// Quando o

A resposta está feita,

tentar

{

// Tente

analisá -lo como JSON

resolver
(
JSON
.
analisar
(
corpo
));

// e

Resolva o resultado.

}

pegar
(
e
)

{

// ou, se

Tudo dá errado,

rejeitar
(
e
);
```

```
endpoint isto
// ecoa a solicitação de entrada, que pode ser útil quando

depurar clientes.
const

http

=

exigir
(
"http"
);

// Use "https" se você tiver um

Certificado
const

url

=

exigir
(
"Url"
);

// para analisar URLs
const

caminho

=

exigir
(
"caminho"
);

// para manipular

Caminhos do sistema de arquivos
const

fs

=

exigir
(
"FS"
);

// Para leitura de arquivos
// serve arquivos do diretório raiz especificado por meio de um http

servidor isso
```

```
// Expaidam os cabeçalhos de solicitação

deixar

cabeçalhos

=

solicitar
.
RawHeaders
;

para
(
deixar

eu

=

0
;

eu

<

cabeçalhos
.
comprimento
;

eu

+=

2
)

{

resposta
.
escrever
(
`
$ {
cabeçalhos
[[
eu
]
}
:

$ {
cabeçalhos
[[
eu
```

```
deixar

fluxo

=

fs
.
Createradstream
(
nome do arquivo
);

fluxo
.
uma vez
(
"legível"
, Assim,

()

=>

{

// 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.

resposta
.
Setheader
(
"Tipo de conteúdo"
, Assim,

tipo
);

resposta
.
writehead
(
200
);
```

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
Web
Servidores e clientes 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
net.createServer ()
, então ligue para o
ouvir()
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.Chamar
fim()
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 se 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

líquido

=

exigir
(
"líquido"
);
const

ReadLine

=

exigir
(
"ReadLine"
);
// Crie um objeto de servidor e comece a ouvir conexões
deixar

servidor

=

líquido
.
CreateServer
();
servidor
.
ouvir
(
6789
, Assim,

()

=>

console
.
registro
(
"Entregando risadas

Porta 6789 "
));
// Quando um cliente se conectar, diga-lhes uma piada de knock-knock.
servidor
.
sobre

```
// Use o módulo ReadLine para ler a entrada do usuário

linha de cada vez.

deixar

LineReader

=

ReadLine
.
createInterface
({

entrada
:

soquete
, Assim,

saída
:

soquete
, Assim,

incitar
:

">>"

});

// uma função de utilidade para gerar uma linha de texto para o

cliente

// e então (por padrão) exibe um prompt.

função

saída
(
texto
, Assim,

incitar
=
verdadeiro
)

{

soquete
.
escrever
(
`
```

```
}

outro

se

(
estágio

===

1
)

{

se

(
inputline
.
tolowercase
()

===

`
$ {
Quem
.
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
}
`
, Assim,

falso
);

retornar
```

stdin para o soquete
soquete
.
sobre
(
"fechar"
, Assim,

()

=>

processo
.
saída
());

// 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".
O
TLS.Server
e
tls.tlssocket
Aulas 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
Em
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 ()
O
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, então
Execsync ()
joga 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 a
diretório com um comando familiar de shell unix, em vez de usar o
fs.readdirsync ()
função:
const Child_process = requer ("Child_Process");
Seja listing = child_process.execsync ("ls -l web/*. html",

{codificação: "utf8"});
O fato disso
Execsync ()
invoca uma concha completa 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
Execsync ()
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 uma concha 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
muitos dos outros
Child_process
As funções têm um segundo ou terceiro opcional
Argumento que especifica detalhes adicionais sobre como o processo infantil deve ser executado.O
codificação
A propriedade deste objeto foi usada anteriormente para especificar que gostaríamos que a saída do c
omando fosse entregue
como uma string e não como um buffer.Outras propriedades importantes que você pode especificar inclu
em o
A seguir (observe que nem todas as opções estão disponíveis para todas as funções do processo filho)
:
cwd
Especifica o diretório de trabalho para o processo filho.Se você omitir isso, então a criança
processo herda o valor de
process.cwd ()
.
Env
Especifica as variáveis ■■de ambiente às quais o processo filho terá acesso.Por
Padrão, os processos filhos simplesmente herdam
process.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
exec
funções.(Não se aplica a
Spawn ()
e
garfo()
, que usam fluxos.) Se uma criança
O processo produz mais saída do que isso, será morto e sairá com um erro.
concha
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 esgotado
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 ao
exec
funções, mas não para

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

```
// Alguns caminhos importantes
processo
.
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

=

exigir
(
"caminho"
);
caminho
.
set

//, "/" ou "\"

Dependendo do seu sistema operacional
// O módulo de caminho tem funções de análise simples
deixar

p

=

"Src/pkg/test.js"
;

// Um nnexemplo de caminho
caminho
.
nome de base
(
p
)
```

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 passar o
URLSearchParams objeto como o valor do
corpo
propriedade.
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 a
<morm>
elemento para o
FormData ()
construtor.
Mas você também pode criar órgãos de solicitação "multipart/formulários"
invocando o
FormData ()
construtor sem argumentos
e inicializando o nome do nome/valor que ele representa com o
definir()
e
acrescentar()
Métodos.
Upload de arquivo com fetch ()
Upload
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 um
<entrada
type = "arquivo">
Elemento em sua página da web e ouça "Change"
eventos nesse elemento.Quando um evento de "mudança" ocorre, o
arquivos

ID do processo da criança.E define um
matar()
Método 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.
garfo()
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
garfo()
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,
garfo()
Ativa outro canal de comunicação muito mais fácil
entre os processos pais e filhos.
Quando você cria um processo infantil com
garfo()
, você pode usar o
enviar()
Método do objeto de processo 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
process.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
garfo()
Para criar uma criança
Processar, então envia uma mensagem a essa criança e aguarda uma resposta:
const

Child_process

=

exigir
(
"Child_process"
);
// Inicie um novo processo de nó executando o código no Child.js em

```
nosso diretório
deixar

criança

=

Child_process
.
garfo
(
`
$ {
__dirname
}
/Child.js`
);
// Envie uma mensagem para a criança
criança
.
enviar
({
x
:

4
, Assim,

y
:

3
});
// Imprima a resposta da criança quando chegar.
criança
.
sobre
(
"mensagem"
, Assim,

mensagem

=>

{

console
.
registro
(
mensagem
.
hipotenusa
);

// Isso deve imprimir "5"

// Como apenas enviamos uma mensagem, esperamos apenas uma
```

(Embora um tópico - veja
§16.11
- pode ser uma escolha melhor do que uma criança
processo neste cenário.)
O primeiro argumento para
enviar()
será serializado com
Json.Stringify ()
e desaperializado no processo infantil com
Json.parse ()
, então você deve incluir apenas valores que são suportados
pelo formato json.
enviar()
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
garfo()
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
enviar()
ele - 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
Como
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 o
API de trabalhadores da web definida por navegadores da web
(
§15.13
).Programação multithread
tem uma reputação bem 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
PostMessage ()
Método 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 de
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
Nó
aplicativo:
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
O módulo do nó que define os trabalhadores é conhecido como "trabalhador_threads".
Nesta seção, nos referiremos a ela com o identificador
tópicos
:

```
const

tópicos

=

exigir
(
"Worker_threads"
);
```

Este módulo define uma classe de trabalhador para representar um fio de trabalhador e
Você pode criar um novo tópico 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

tópicos

=

exigir
(
"Worker_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.
se

(
tópicos
.
ismainthread
)

{

// 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.

módulo
.
exportações

=

```
reticulador
.
sobre
(
"erro"
, Assim,

rejeitar
);

});

};
}

outro

{

// Se chegarmos aqui, significa que estamos no trabalhador, então

Registre a

// Handler para obter mensagens do thread principal.Esse

O trabalhador foi projetado

// para receber apenas uma única mensagem, então registramos o

manipulador de eventos

// com uma vez () em vez de em ().Isso permite que o trabalhador

Saia naturalmente

// Quando seu trabalho estiver concluído.

tópicos
.
parenteport
.
uma vez
(
"mensagem"
, Assim,

splines

=>

{

// Quando obtemos as splines do tópico pai,

laço

// através deles e reticulam todos eles.

para
```

```
<botão

id =
"MyButton"
>
Clique em mim
</button>
<Cript>
deixar

b

=

documento
.
QuerySelector
(
"#mybutton"
);
b
.
ONCLICK

=

função
()

{

console
.
registro
(
"Obrigado por clicar

meu!"
);

};
b
.
addEventListener
(
"clique"
, Assim,

()

=>

{

console
.
registro
(
"Obrigado
```

implementar usando o iterador-retorno
Matchall ()
método
descrito em
§11.3.2
):
função

palavras
(
s
)

{

var

r

=

/\ s+| $/g
;

// corresponde a um ou

mais espaços ou fim

r
.
LastIndex

=

s
.
corresponder
(
/[^]/
).
índice
;

// Comece a combinar

no primeiro não espaço

retornar

{

// retorna um

objeto iterador iterável

[[
Símbolo
.
iterador

Especifique um conjunto personalizado de variáveis ■■de ambiente definindo o
Env
propriedade do segundo argumento para o
Trabalhador()
construtor.Como um caso especial (e potencialmente perigoso), o
O tópico pai pode definir o
Env
propriedade para
threads.share_env
, que fará com que os dois fios
compartilhe um único conjunto de variáveis ■■de ambiente para que uma mudança em
Um tópico é visível no outro.
Por padrão, o
process.stdin
Transmita em um trabalhador nunca
possui dados legíveis sobre ele.Você pode alterar esse padrão por
passagem
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
torna -se legível
sobre
process.stdin
no trabalhador.
Por padrão, o
process.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 ()
produzir saída
exatamente da mesma maneira em um fio de trabalhador que eles
Tópico principal.Você pode substituir esse padrão passando
stdout: verdadeiro
ou
Stderr: Verdadeiro
no segundo argumento para
o
Trabalhador()
construtor.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 tópico de trabalhador chama

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 () lançará exceções quando invocado de um trabalhador. Sinais de 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 usos de threads 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.Chamando
PostMessage ()
em um dos portos irá
causa a
Evento de “mensagem” a ser gerado por outro com um clone estruturado de
o objeto de mensagem:
const

tópicos

=

exigir
(
"Worker_threads"
);
deixar

canal

=

novo

tópicos
.
Messagechannel
();
canal
.
PORT2
.
sobre
(
"mensagem"
, Assim,

console
.
registro
);

// registrar qualquer um

mensagens que recebemos
canal
.
PORT1
.
Postmessage

O
PostMessage ()
função
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
PostMessage ()
método (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 um ou mais
Messageports
(aninhado
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 mais usado no tópico que chamava
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

tópicos

=

exigir
(
"Worker_threads"
);
deixar

canal

=

novo

tópicos
.
Messagechannel
();
// Use o canal padrão do trabalhador para transferir uma extremidade de

o novo canal.
trabalhador
.
Postmessage
({

comando
:

"Changechannel"
, Assim,

dados
:

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
canal
.
PORT2
.
sobre
(
"mensagem"
, Assim,

HandleMessagesFroworker
);
Os objetos MessagePort não são os únicos que podem ser transferidos.Se
você liga
PostMessage ()
com uma matriz digitada como a mensagem (ou com
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

Isso apóia a matriz no segundo argumento para
PostMessage ()
:
deixar

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
, Assim,

[[

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
Em
Além da transferência de 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 a

o simples
++
O operador não é seguro para fios 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

tópicos

=

exigir
(
"Worker_threads"
);
se

(
tópicos
.
ismainthread
)

{

// 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.

deixar

SharedBuffer

=

novo

SharedArrayBuffer
(
4
);

deixar

SharedArray

=

novo

INT32Array
(

```
});
}

outro

{

// No tópico do trabalhador, obtemos a matriz compartilhada de

WorkerData

// e depois aumentam 10 milhões de vezes.

deixar

SharedArray

=

tópicos
.
WorkerData
;

para
(
deixar

eu

=

0
;

eu

<

10
_000_000
;

eu
++
)

SharedArray
[[
0
]
++
;

// Quando terminarmos o incremento, informe o tópico principal

tópicos
.
parenteport
```

apenas olhou e obtenha o resultado correto de 20 milhões de incrementos de um
Elemento de matriz compartilhada:

```
const

tópicos

=

exigir
(
"Worker_threads"
);
se

(
tópicos
.
ismainthread
)

{

deixar

SharedBuffer

=

novo

SharedArrayBuffer
(
4
);

deixar

SharedArray

=

novo

INT32Array
(
SharedBuffer
);

deixar

trabalhador

=

novo

tópicos
.
Trabalhador
```

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.O nível baixo
Atomics.wait ()
e
Atomics.Notify ()
a função 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 o filho
processos.
Módulo "Worker_threads" do Node para verdadeiro multithreaded
Programação usando passagem de mensagens em vez de compartilhado
memória.
1
O nó define a
fs.copyfile ()
função 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 o Unix
garfo()
Chamada do sistema.E eu acho que vale a pena demonstrar
Esta técnica simplesmente por sua estranha elegância.

Capítulo 17.
Ferramentas JavaScript e extensões

Parabéns

Ao 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

Em
Programação, o termo
fia
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
LING
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 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'
;
exportar

função

fatorial
(
x
)

{

se

(
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
```

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
Um
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 o
para
palavra -chave e os parênteses abertos que se seguem
isto.
Uma alternativa moderna para aplicar regras de formatação de código por meio de um linhador é
para adotar uma ferramenta como
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
)
{

se
(
x
===
1
) {
retornar

1
}

outro
{
retornar

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 o
--escrever
opção, vai simplesmente
reformate o arquivo especificado em vigor em vez de imprimir um reformado
versão.Se você usa
git
Para gerenciar seu código -fonte, você pode invocar
Mais bonito com o
--escrever
opção em um gancho de commit 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
Escrita
Testes é 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
Jove
, Assim,
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

=

exigir
(
"./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.
*/
módulo
.
exportações

=

assíncrono

função

gettemperature
(
cidade
)

{

// Obtenha a temperatura em Celsius do serviço da web

deixar

c
```

fórmula
};
Um bom conjunto de testes para esta função pode verificar que
getTemperature ()
está buscando o URL certo e que é
converter escalas de temperatura corretamente.Podemos fazer isso com uma brincadeira
teste baseado como o seguinte.Este código define uma implementação simulada
de
getjson ()
para que o teste não faça uma rede
solicitar.E porque
getTemperature ()
é uma função assíncrona, o
Os testes também são assíncronos - pode ser complicado testar funções assíncronas,
Mas o jest torna isso relativamente fácil:
// importar a função que vamos testar
const

gettemperature

=

exigir
(
"./gettemperature.js"
);
// e zombar do módulo getjson () que gettemperature ()

depende de
Jove
.
zombar
(
"./getjson"
);
const

getjson

=

exigir
(
"./getjson.js"
);
// Diga à função Mock getjson () para retornar um já

promessa resolvida
// com valor de atendimento 0.
getjson
.
MockResolvedValue
(
0
);
// Nosso conjunto de testes para gettemperature () começa aqui
descrever
(
"gettemperature ()"

```
// Este segundo teste verifica que GetTemperature ()

convertidos

// Celsius para Fahrenheit corretamente

teste
(
"Converte C para F corretamente"
, Assim,

assíncrono

()

=>

{

getjson
.
MockResolvedValue
(
0
);

// Se

getjson retorna 0c

esperar
(
aguarde

gettemperature
(
"X"
)).
ser
(
32
);

// Nós

Espere 32f

// 100c deve se converter para 212f

getjson
.
MockResolvedValue
(
100
);

// Se

getjson retorna 100c
```

(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.
Nosso
getTemperature ()
A implementação 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
-
cobertura
argumento quando invocamos
Jove
, 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 daquele módulo e não fomos
Tentando testá -lo, o que é esperado.

17.4 Gerenciamento de pacotes com NPM

Em
Desenvolvimento de software moderno, 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
arquivar e baixar os pacotes de terceiros que o
necessidades do projeto e salva -as em um
node_modules/
diretório.

Se uma janela contém janelas infantis (como
<frame>
elementos),
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

Um
A técnica de gerenciamento de história envolve
location.hash
e
O evento "Hashchange".Aqui estão os principais fatos que você precisa saber para
Entenda esta técnica:
O
location.hash
Propriedade 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 o
Hash
propriedade como

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 tem 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 barriga:
/\\//
.(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
Individual
caracteres literais podem ser combinados em
classes de personagens
por

Use bibliotecas externas entregues como módulos.Desenvolvedores da web
usa módulos ES6 (
§10.3
) durante anos, pois bem antes do
importar
e
exportar
Palavras -chave foram suportadas na web.Em ordem
Para fazer isso, os programadores usam uma ferramenta de Bundler de código que começa no principal
ponto de entrada (ou pontos de entrada) do programa e segue a árvore de
importar
Diretivas para encontrar todos os módulos dos quais o programa depende.Isto
Em seguida, combina todos esses arquivos de módulo individuais em um único pacote
do código JavaScript e reescreve o
importar
e
exportar
diretivas para
Faça o código funcionar nesse novo formulário.O resultado é um único arquivo de código
Isso pode ser carregado em um navegador da Web que não suporta módulos.
Os módulos ES6 são quase universalmente suportados pelos navegadores da web hoje,
Mas os desenvolvedores da web ainda tendem a usar os pacotes de código, pelo menos quando
liberando código de produção.Os desenvolvedores descobrem que a experiência do usuário é a melhor
Quando um único pacote de código de tamanho médio é carregado quando um usuário primeiro
Visita um site do que quando muitos módulos pequenos são carregados.
OBSERVAÇÃO
O desempenho da web é um tópico notoriamente complicado e há muitas variáveis nnpara
Considere, incluindo melhorias contínuas dos fornecedores do navegador, então a única maneira
para ter certeza da maneira mais rápida de carregar seu código é testando completamente e
medindo cuidadosamente.Lembre -se de que há uma variável que é completamente
Sob seu controle: tamanho do código.Menos código JavaScript sempre carregará e executará
mais rápido que mais código JavaScript!
Existem várias boas ferramentas de Bundler JavaScript disponíveis.
Matraços comumente usados nnincluem
webpack
, Assim,
Rollup
e
Parcela
.O
As características básicas dos pacotes são mais ou menos as mesmas e são

diferenciado com base em quão configuráveis nneles 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.
Programas podem usar
importar()
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
apoiar
importar()
pode ser capaz de produzir vários 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()
no seu programa e eles
carregar
Módulos 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.
Matores geralmente podem produzir um
mapa de origem
arquivo 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
importar
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 um
importar
diretivo.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
funções podem ser transformadas relativamente facilmente em
Math.pow ()
e
função
expressões.Outros recursos de linguagem, como o
aula
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
Babel's
interessante
Predefinições é 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:

```
deixar

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:

```
deixar

linha

=

Reagir
.
CreateElement
(
"HR"
, Assim,

nulo
);
```

A sintaxe JSX é como HTML e, como elementos HTML, elementos de reação
pode ter atributos como estes:

```
deixar

imagem

=

<
img

src
=
"logo.png"

alt
=
"O logotipo JSX"

escondido
```

A promessa devolvida por
buscar()
resolve para um objeto de resposta.O
status
A propriedade 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, o
OK
propriedade de uma resposta é
verdadeiro
se
status
é 200 ou qualquer código entre 200 e 299 e é
falso
Para qualquer outro código.
buscar()
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.O
cabeçalhos
propriedade de um objeto de resposta
é um objeto de cabeçalhos.Use seu
tem()
método para testar a presença de um
cabeçalho ou usar seu
pegar()
Método para obter o valor de um cabeçalho.Http
Os nomes dos cabeçalhos são insensíveis a maiúsculas, para que você possa passar em minúsculas ou mi
stas
Nomes de cabeçalho de casos para essas funções.
O objeto de cabeçalhos também é iterável se você precisar fazer isso:

```
buscar
(
url
).
então
(
resposta

=>

{

para
(
deixar

[[
nome
, Assim,
valor
]

de
```

efeitos como uma chamada de função ou um operador de incremento.A seguir
Duas tarefas, por exemplo, não são
o mesmo:
dados
[[
eu
++
]

*=

2
;
dados
[[
eu
++
]

=

dados
[[
eu
++
]

*

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
avaliar ()
:
aval
(
"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 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 usar com
avaliar ()
.Por causa disso

Isso apóia a matriz no segundo argumento para
PostMessage ()
:
deixar

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
, Assim,

[[

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
Em
Além da transferência de 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 a

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

elementos como filho.
função

lista
(
Unid
, Assim,

ligar de volta
)

{

retornar

(

<
ul

estilo
=
{

{
preenchimento
:
10
, Assim,

fronteira
:
"Red Solid Red 4px"
}

}
>

{
Unid
.
mapa
((
item
, Assim,
índice
)

=>

{

```
retornar

Reagir
.
CreateElement
(

"Ul"
, Assim,

{

estilo
:

{

preenchimento
:

10
, Assim,

fronteira
:

"Red Solid Red 4px"

}

},

Unid
.
mapa
((
item
, Assim,

índice
)

=>

Reagir
.
CreateElement
(

"Li"
, Assim,

{

ONCLICK
:

()
```

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 para
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 ()
Ativa 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
barra lateral ()
função:
função

Barra lateral
(
adereços
)

{

retornar

(

<
div
>

<
H1
>
{
adereços
.

}
Este novo
Barra lateral ()
A função é muito parecida com o anterior
barra lateral ()
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:
deixar

barra lateral

=

<
Barra lateral

título
=
"Algo rápido"

contente
=
"Algo sábio"
/>
;
Esse
<Barra lateral/>
elementos compilam como este:
deixar

barra lateral

=

Reagir
.
CreateElement
(
Barra lateral
, Assim,

{

título
:

"Algo rápido"
, Assim,

contente
:

"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
Barra lateral ()

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
para o código e tudo
Babel precisa fazer é retirar essas anotações para devolver seu código para
JavaScript padrão.)
TypeScript versus Flow
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 anota
ções 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 um
aula
palavra -chave ou a
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 T
ypeScript,
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 do título tem
enum
e
espaço para nome
palavras -chave que simplesmente não existem em
JavaScript.Em 2020, o TypeScript tem melhor integração com IDEs e editores de código (particularment
e
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 por
que
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
acabar gastando tempo lendo a documentação em
https://flow.org
.Sobre
Por outro lado, você não precisa dominar o sistema de tipo de fluxo antes
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
1

NPM Install -g Flow -Bin
ou
NPM Instale-Save-dev
Fluxo-barracão
.Se você instalar a ferramenta globalmente com
-g
, então você pode executá -lo
com
fluxo
.E se você o instalar localmente em seu projeto com
--salvar-
Dev
, então você pode executá -lo com
Fluxo NPX
.Antes de usar o fluxo para fazer
Verificação do tipo, a primeira vez que o executa como
fluxo -init
na raiz
diretório do seu projeto para criar um
.FlowConfig
arquivo de configuração.
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
// @fluxo
Comente no topo 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
// @fluxo
comentário.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
eu
e atribuir um objeto a ele.Então
nós usamos
eu
Novamente como uma variável de loop, substituindo o objeto.Avisos de fluxo
isso e sinaliza um erro quando tentamos usar
eu
Como se ainda mantivesse um objeto.
(Uma solução 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);
Fluxo vê que o
tamanho()
Função leva um único argumento.Não
saber o tipo desse argumento, mas pode ver que o argumento é
Espera -se ter um
comprimento
propriedade.Quando vê isso
tamanho()
a função sendo chamada com um argumento numérico, ele sinaliza corretamente
um erro porque os números não têm
comprimento
propriedades.
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:
deixar

mensagem
:

corda

=

"Hello World"
;
deixar

bandeira
:

booleano

=

falso
;
deixar

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 nntambém são
particularmente útil se você tende a declarar todas as suas variáveis nnno 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 um
tamanho()
função que esperou
uma discussão com um
comprimento

Erro

nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn

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] 2n Tamanho da função (s: string): número {
3n Retornar S. comprimento;
4n}
[1] 5n 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
:

corda
)
:

número

=>

s
.
comprimento
;
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 para passar
nulo
ou passar
indefinido
ou para omitir o argumento
(que é basicamente a mesma coisa que passar
indefinido
):
Erro

Se você quiser 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
?corda
ou
?número
em vez de
corda
ou
número
.Se mudarmos nosso
tamanho()
função para esperar um argumento de
tipo
?corda
, então o fluxo não reclama quando passamos
nulo
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. Length
Porque, neste lugar em nosso código,
s
pode ser
nulo
ou
indefinido
, Assim,
E esses valores não têm
comprimento
propriedades.É aqui que flui
Garanta que não cortamos nenhum canto.Se um valor puder ser
nulo
, Fluxo
Will insistirá que verificamos esse caso antes de fazer qualquer coisa que
depende do valor que não está sendo
nulo
.
Nesse caso, podemos corrigir o problema alterando o corpo da função
do seguinte modo:
função

```
// 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
.
comprimento
;

}
}
```

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 nós usamos
S. Length
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
retornar s?S. Length: -1;
.
A sintaxe do fluxo permite um ponto de interrogação antes de qualquer especificação de tipo indicar 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 do parâmetro
s
de
S:? String
para
s?:
corda
, isso significaria que não há problema em ligar
tamanho()
sem argumentos
(ou com o valor
indefinido
, que é o mesmo que omitindo), mas

suportado por fluxo.
17.8.3 Tipos de classe
Em
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/);
exportar

função

DataMatches
(
d
:

Data
, Assim,

p
:

Regexp
)
:

booleano

{

retornar

p
.
teste
(
d
.
ToisSotring
());
}
```

Se você definir suas próprias aulas com o
aula
Palavra -chave, essas aulas
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
```

// Quaisquer propriedades inicializadas pelo construtor devem

tem tipo de fluxo

// anotações acima.

esse
.
r

=

r
;

esse
.
eu

=

eu
;

}

adicionar
(
que
:

Complexo
)

{

retornar

novo

Complexo
(
esse
.
r

+

que
.
r
, Assim,

esse
.
eu

+

Dentro de um tipo de objeto, você pode seguir qualquer um dos nomes de propriedades com um ponto de interrogação para indicar que essa propriedade é opcional e pode ser omitido.Por exemplo, você pode escrever o tipo para um objeto que representa um ponto 2D ou 3D como este:

```
{
x
:

número
, Assim,

y
:

número
, Assim,

z
?:

número
}
```

Se uma propriedade não estiver marcada como opcional em um tipo de objeto, então é exigido, e o fluxo relatará um erro se uma propriedade apropriada não for presente no valor real.Normalmente, no entanto, o fluxo tolera extra propriedades.Se você passar por um objeto que tinha um
c
propriedade para o
distância()
Função acima, o fluxo não reclamaria.

Se você deseja que o fluxo aplique estritamente que um objeto não possui propriedades diferentes das declaradas explicitamente em seu tipo, você pode declarar um
tipo de objeto exato
Adicionando barras verticais aos aparelhos encaracolados:

```
{
|

x
:

número
, Assim,

y
:

número

|
}
```

Os objetos de JavaScript às vezes são usados nncomo dicionários ou string-to-mapas de valor.Quando usado assim, os nomes de propriedades não são conhecidos em avançar e não pode ser declarado em um tipo de fluxo.Se você usar objetos isso maneira, você ainda pode usar o fluxo para descrever a estrutura de dados.Suponha que isso você tem um objeto onde as propriedades são os nomes do mundo as principais cidades e os valores dessas propriedades são objetos que especificam A localização geográfica dessas cidades.Você pode declarar esses dados estrutura como esta:

"Seattle"
:

{

longitude
:

47.6062
, Assim,

latitude
:

-
122.3321

},

// TODO: Se houver outras cidades importantes, adicione

eles aqui.
};
exportar

padrão

CityLocações
;
17.8.5 Aliases do tipo
Objetos
pode 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 de tipos simples como
número
e
?
corda
, geralmente é útil poder definir nomes para o nosso fluxo
tipos.E de fato, o fluxo usa o
tipo
palavra -chave para fazer exatamente isso.
Siga o
tipo
palavra -chave com um identificador, um sinal igual e um
Tipo de fluxo.Depois de fazer isso, o identificador será um alias para o
tipo.Aqui, por exemplo, é como poderíamos reescrever o
distância()
função da seção anterior com um explicitamente definido
Apontar
tipo:
// @fluxo
exportar

tipo

Apontar

Ponto de './distance.js'
Se eles querem usar esse tipo
definição.Lembre -se, porém, que
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 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
Variedade
seguido pelo tipo de elemento em
suportes de ângulo.Você também pode expressar um tipo de matriz seguindo o

Tipo de elemento com suportes quadrados abertos e próximos.Então neste exemplo
nós poderíamos ter escrito
número[]
em vez de
Array <número>
.EU
prefira a notação do suporte do ângulo porque, como veremos, existem outros
Tipos de fluxo que usam esta sintaxe do colack de ângulo.
A sintaxe do tipo de matriz mostrada
trabalha para matrizes com um arbitrário
Número de elementos, todos os quais têm o mesmo tipo.O fluxo tem um
sintaxe diferente para descrever o tipo de um
tupla
: uma matriz com um fixo
Número de elementos, cada um dos quais pode ter um tipo diferente.Para
expressar o tipo de tupla, basta escrever o tipo de cada um de seus
elementos, separe -os com vírgulas e inclua todos eles em quadrado
Suportes.
Uma função que retorna um código de status e mensagem HTTP pode parecer
assim, por exemplo:
função

getstatus
()
:
[[
número
, Assim,

corda
]

{

retornar

[[
getStatuscode
(),

getStatusMessage
()];
}
Funções que retornam tuplas são estranhas para trabalhar, a menos que você use
atribuição de destruição:
deixar

[[
código
, Assim,

mensagem
]

=

getstatus
();
Atribuição de destruição, além de recursos de aliases do fluxo, fazem

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 tem 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 barriga:
/\\//
.(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
Individual
caracteres literais podem ser combinados em
classes de personagens
por

O
empurrar()
o método 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:
um
.
empurrar
(...
valores
);
O

NIFT ()
e
mudança()
métodos se comportam muito como
empurrar()
e
pop ()
, exceto que eles inserem e removem elementos do
Início de uma matriz e não do final.
NIFT ()
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.
mudança()
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
NIFT ()
e
mudança()
Para implementar uma pilha, mas seria menos
eficiente do que usar
empurrar()
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
empurrar()
Para adicionar elementos no final de um
Array e
mudança()
Para removê -los desde o início da matriz:
deixar

q

=

[];

// q == []

[[
"azul"
, Assim,

[[
0
, Assim,

0
, Assim,

1
, Assim,

1
]]
]);
O fluxo permite definir parâmetros de tipo para suas próprias classes também.O
a seguir o código define uma classe de resultado, mas parametrizam essa classe com
um tipo de erro e um tipo de valor.Usamos espaço reservado
E
e
V
no
Código para representar esses parâmetros de tipo.Quando o usuário desta classe
declara uma variável de resultado do tipo, eles especificarão os tipos reais para
substituto para
E
e
V
.A declaração variável pode ser assim:
deixar

resultado
:

Resultado
<
TypeError
, Assim,

Definir
<
corda
>>
;
E aqui está como a classe parametrizada é definida:
// @fluxo
// Esta classe representa o resultado de uma operação que pode

qualquer
// Lança um erro do tipo E ou um valor do tipo V.
exportar

aula

Resultado
<
E

```
}

outro

{

retornar

esse
.
valor
;

}

}
}
```

E você pode até definir parâmetros de tipo para funções:

```
// @fluxo
// combina os elementos de duas matrizes em uma matriz de pares
função

Zip
<
UM
, Assim,
B
>
(
um
:
Variedade
<
UM
>
, Assim,

b
:
Variedade
<
B
>
)
:

Variedade
<
[[
?
UM
, Assim,
?
B
]
>

{
```

Garanta que um objeto ou matriz não possa ser modificado (ver
Object.freeze ()
em
§14.2
Se você quer 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

Apontar

=

{
x
:
número
, Assim,

y
:
número
};
// Esta função leva um objeto de ponto, mas promete não

modifique -o
função

distância
(
p
:

$ Readonly
<
Apontar
>
)
:

número

{

retornar

Matemática
.
Hypot
(
p
.
x
```

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
exportar

tipo

FetchTextCallback

=

(
?
Erro
, Assim,

?
número
, Assim,

?
corda
)

=>

vazio
;
exportar

padrão

função

FetchText
(
url
:

corda
, Assim,

ligar de volta
:

FetchTextCallback
)

{

Erro ao traduzir esta página.

simplesmente um atalho para adicionar um
| nulo | vazio
sufixo para 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
tamanho()
Exemplo nós apenas
Observamos, precisamos verificar explicitamente se o argumento é uma matriz
antes de tentarmos acessar o
comprimento
propriedade do argumento.Observação
que não precisamos distinguir um argumento definido de um mapa
argumento,
no entanto:
Ambas as classes definem um
tamanho
propriedade, então
o código no
outro
a cláusula é segura, desde que o argumento não seja um
variedade.
17.8.11 tipos enumerados e discriminados
Sindicatos
Fluxo
permite que você use literais primitivos como tipos que consistem nisso
um único valor.Se você escrever
Seja 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:
tipo

Responder

=

"sim"

|

"não"
;
tipo

Digit

=

0
|
1
|
2
|
3
|

número a ser dígito
Quando o fluxo verifica seus tipos, ele realmente não faz os cálculos:
Apenas verifica os tipos de cálculos.O fluxo sabe disso
tolowerCase ()
retorna uma string e que o
+
operador em números
Retorna um número.Mesmo sabendo que ambos os cálculos
Retornar valores que estão dentro do tipo, o fluxo não pode saber disso e sinalizadores
erros em ambas as linhas.
Um tipo de sindicato de tipos literais como
Responder
e
Digit
é um exemplo de
um
Tipo enumerado
, ou
enum
.Um caso de uso canônico para tipos de enum é
Representar os ternos das cartas de jogo:
tipo

Terno

=

"Clubes"

|

"Diamantes"

|

"Corações"

|

"Espadas"
;
Um exemplo mais relevante pode ser os códigos de status HTTP:
tipo

Httpstatus

=

|

200

// OK

|

304

// não modificado

Outro uso importante para tipos literais é a criação de
discriminado
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.
exportar

tipo

ResultMessage

=

{

MessageType
:

"resultado"
, Assim,

resultado
:

Variedade
<
ReticulatedSpline
>
, Assim,

// Suponha que esse tipo seja

definido em outro lugar.
};
// O trabalhador envia uma mensagem desse tipo se seu código falhou

com uma exceção.
```

```
// O trabalhador envia uma mensagem desse tipo para relatar o uso

Estatística.
exportar

tipo

StatisticsMessage

=

{

MessageType
:

"Estatísticas"
, Assim,

splinesReticululs
:

número
, Assim,

splinespersegund
:

número
};
// Quando recebermos uma mensagem do trabalhador, será um

WorkerMessage.
exportar

tipo

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
```

reescrito para usar o objeto Argumentos em vez de um parâmetro REST:
função

máx
(
x
)

{

deixar

maxvalue

=

-
Infinidade
;

// percorre os argumentos, procurando e

Lembrando, o maior.

para
(
deixar

eu

=

0
;

eu

<

argumentos
.
comprimento
;

eu
++
)

{

se

(
argumentos
[[
eu
]

>

+ (mais sinal)
Operador de adição e atribuição (+=)
, Assim,
Atribuição com
Operação
Operador de adição
, Assim,
Aritmética em javascript
, Assim,
O operador +
Concatenação de string
, Assim,
Literais de cordas
, Assim,
Trabalhando com cordas
, Assim,
O +
Operador
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
Operador aritmético unário
, Assim,
Operadores aritméticos unários
++ (operador de incremento)
, Assim,
Operadores aritméticos unários
, (operadora de vírgula)
, Assim,
O operador de vírgula (,)
- (sinal de menos)
Operador de subtração
, Assim,
Aritmética em javascript
, Assim,
Aritmética
Expressões
Operador aritmético unário
, Assim,
Operadores aritméticos unários
- (Operador de decrescente)
, Assim,
Operadores aritméticos unários
.(Operador de pontos)
, Assim,
Um passeio de JavaScript
, Assim,
Consulta e configuração de propriedades
/ (operadora de divisão)
, Assim,
Aritmética em javascript
, Assim,
Expressões aritméticas
/ * */ caracteres
, Assim,
Comentários
// (barras duplas)
, Assim,
Um passeio de JavaScript

```
// uma função de teste que usa o gerador assíncrono com

para/aguardar
assíncrono

função

teste
()

{

// assíncrono então nós

pode usar para/aguardar

para

aguarde

(
deixar

marcação

de

relógio
(
300
, Assim,

100
))

{

// loop 100

vezes a cada 300ms

console
.
registro
(
marcação
);

}
}
```

13.4.4 Implementando iteradores assíncronos

Em vez de
de usar geradores assíncicos para implementar iteradores assíncronos,
Também é possível implementá -los diretamente, definindo um objeto
com um
Symbol.asynciterator ()
método que retorna um objeto
com um
próximo()

Visão geral de
, Assim,
Operadores de comparação
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
> = (maior ou igual ao operador)
Visão geral de
, Assim,
Operadores de comparação
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
>> (desligue bem com o operador de sinal)
, Assim,
Operadores bitwise
>>> (desligue com a direita com o operador de preenchimento zero)
, Assim,
Operadores bitwise
?.(Operador de acesso condicional)
, Assim,
Um passeio de JavaScript
, Assim,
Função
Invocação
?: (Operador condicional)
, Assim,
O operador condicional (? :)
?(operador primeiro definido)
, Assim,
Primeiro definido (??)
[] (suportes quadrados)
, Assim,
Um passeio de JavaScript
, Assim,
Trabalhando com cordas
, Assim,
Inicializadores de objetos e matrizes
, Assim,
Consulta e configuração de propriedades
, Assim,
Leitura
e escrever elementos de matriz
, Assim,
Cordas como matrizes
\ (barragem)
, Assim,
Literais de cordas
-
Sequências de fuga em literais de cordas
\ n (newline)
, Assim,
Literais de cordas

_ (sublinhado)
, Assim,
Identificadores e palavras reservadas
_ (sublinhado, como separadores numéricos)
, Assim,
Literais de ponto flutuante
`(Backtick)
, Assim,
Literais de cordas
, Assim,
Literais de modelo
{} (aparelho encaracolado)
, Assim,
Um passeio de JavaScript
, Assim,
Inicializadores de objetos e matrizes
||(Booleano ou operador)
, Assim,
Valores booleanos
, Assim,
Lógico ou (||)
~ (bitwise não operador)
, Assim,
Operadores bitwise
■ (bit newwise ou operador)
, Assim,
Operadores bitwise
… (Operador espalhado)
, Assim,
Operador espalhado
, Assim,
O operador de propagação
, Assim,
O
Espalhe o operador para chamadas de função
, Assim,
Iteradores e geradores
UM
Classes abstratas
, Assim,
Hierarquias de classe e classes abstratas
-
Resumo
acelerômetros
, Assim,
APIs de dispositivo móvel
Propriedades do acessador
, Assim,
Propriedade Getters and Setters
Método addEventListener ()
, Assim,
addEventListener ()
Operador de adição (+)
, Assim,
Aritmética em javascript
, Assim,
O operador +
Recursos avançados
Extensibilidade do objeto

Tags de modelos
, Assim,
Tags de modelos
-
Tags de modelos
símbolos bem conhecidos
Símbolos de correspondência de padrões
, Assim,
Símbolos de correspondência de padrões
Symbol.asynciterator
, Assim,
Symbol.iterator e
Symbol.asynciterator
Símbolo.hasinstance
, Assim,
Símbolo.hasinstance
Symbol.iscoNcatsPreadable
, Assim,
Symbol.iscoNcatsPreadable
Symbol.iterator
, Assim,
Símbolos bem conhecidos
Symbol.Spécies
, Assim,
Symbol.Spécies
-
Symbol.Spécies
Symbol.ToPrimitive
, Assim,
Symbol.ToPrimitive
Symbol.ToStringTag
, Assim,
Symbol.ToStringTag
Symbol.unscopables
, Assim,
Symbol.unscopables
alfabetização
, Assim,
Comparando strings
caracteres âncora
, Assim,
Especificando a posição da correspondência
Apostróficos
, Assim,
Literais de cordas
Método Aplicar ()
, Assim,
Invocação indireta
, Assim,
Os métodos de chamada () e aplicar ()
Método arc ()
, Assim,
Curvas
Método Arcto ()
, Assim,
Curvas
argumentos
Tipos de argumento
, Assim,

Destructar os argumentos da função em parâmetros
, Assim,
Destruição
Funções argumentos em parâmetros
-
Função de destruição
Argumentos nos parâmetros
listas de argumentos de comprimento variável
, Assim,
Parâmetros de descanso e variável-
Listas de argumentos de comprimento
operadores aritméticos
, Assim,
Um passeio de JavaScript
, Assim,
Aritmética em javascript
-
Aritmética em javascript
, Assim,
Expressões aritméticas
-
Operadores bitwise
Índice de Array
, Assim,
Leitura e escrevendo elementos de matriz
Métodos do iterador da matriz
cada () e alguns ()
, Assim,
cada () e alguns ()
filtro()
, Assim,
filtro()
encontre () e findIndex ()
, Assim,
encontre () e findIndex ()
foreach ()
, Assim,
foreach ()
mapa()
, Assim,
mapa()
Visão geral de
, Assim,
Métodos do iterador da matriz
Reduce () e ReduceRight ()
, Assim,
Reduce () e ReduceRight ()
matriz literais
, Assim,
Inicializadores de objetos e matrizes
, Assim,
Matriz literais
Array () construtor
, Assim,
O construtor da matriz ()
Função Array.From ()
, Assim,
Array.From ()
, Assim,

Método ArrayBuffer ()
, Assim,
Analisar os corpos de resposta
matrizes
adicionando e excluindo
, Assim,
Adicionando e excluindo elementos de matriz
comprimento da matriz
, Assim,
Comprimento da matriz
Métodos de matriz
Adicionando matrizes
, Assim,
Adicionando matrizes com concat ()
matriz para conversões de string
, Assim,
Matriz para conversões de string
Arrays achatados
, Assim,
Arrays achatados com plano () e plangmap ()
aplicação genérica de
, Assim,
Matrizes
Métodos do iterador
, Assim,
Métodos do iterador da matriz
-
reduzir () e
Reduteright ()
Visão geral de
, Assim,
Métodos de matriz
pesquisando e classificando
, Assim,
Métodos de pesquisa e classificação de matrizes
-
reverter()
pilhas e filas
, Assim,
Pilhas e filas com push (), pop (),
shift (), e não dividido ()
Funções de matriz estática
, Assim,
Funções de matriz estática
subarrays
, Assim,
Subarrays with slice (), splice (), preench () e
CopyWithin ()
objetos semelhantes a matrizes
, Assim,
Objetos semelhantes a matrizes
-
Objetos semelhantes a matrizes
Matrizes associativas
, Assim,
Introdução aos objetos
, Assim,
Objetos como associativos
Matrizes

Expressões inicializador
, Assim,
Um passeio de JavaScript
, Assim,
Objeto e matriz
Inicializadores
matrizes de iteração
, Assim,
Matrizes de iteração
Matrizes multidimensionais
, Assim,
Matrizes multidimensionais
aninhado
, Assim,
Inicializadores de objetos e matrizes
Visão geral de
, Assim,
Um passeio de JavaScript
, Assim,
Matrizes
processamento com funções
, Assim,
Processando matrizes com funções
Leitura e escrevendo elementos de matriz
, Assim,
LEITURA E ESCREVER ARRAY
Elementos
matrizes esparsas
, Assim,
Matrizes esparsas
cordas como matrizes
, Assim,
Cordas como matrizes
matrizes digitadas
criando
, Assim,
Criando matrizes digitadas
DataView e Endianness
, Assim,
DataView e Endianness
Métodos e propriedades
, Assim,
Métodos de matriz digitados e
Propriedades
Visão geral de
, Assim,
Matrizes digitadas e dados binários
Tipos de matriz digitados
, Assim,
Tipos de matriz digitados
usando
, Assim,
Usando matrizes digitadas
Funções de seta
, Assim,
Um passeio de JavaScript
, Assim,
Definindo funções
, Assim,

afirmações
, Assim,
Um passeio de JavaScript
Operador de atribuição (=)
, Assim,
Um passeio de JavaScript
, Assim,
Igualdade e desigualdade
Operadores
, Assim,
Expressões de atribuição
Matrizes associativas
, Assim,
Introdução aos objetos
, Assim,
Objetos como associativos
Matrizes
Associatividade
, Assim,
Associatividade do operador
palavra -chave assíncrona
, Assim,
assíncrono e aguardar
-
Detalhes da implementação
programação assíncrona
(
ver
também nó)
As palavras -chave assíncronas e aguardam
, Assim,
assíncrono e aguardar
-
Implementação
Detalhes
iteração assíncrona
geradores assíncronos
, Assim,
Geradores assíncronos
Iteradores assíncronos
, Assim,
Iteradores assíncronos
para/aguarda loops
, Assim,
Iteração assíncrona com para/aguardar
, Assim,
Iteração assíncrona
implementação
, Assim,
Implementando iteradores assíncronos
-
Implementando iteradores assíncronos
retornos de chamada
retornos de chamada e eventos no nó
, Assim,
Retornos de chamada e eventos no nó
Definição de termo
, Assim,
Programação assíncrona com

Definição de termo
, Assim,
JavaScript assíncrono
Javascript Suporte para
, Assim,
JavaScript assíncrono
Promessas
encadear promessas
, Assim,
Encadear promessas
-
Encadear promessas
manuseio de erros com
, Assim,
Lidar com erros com promessas
, Assim,
Mais sobre
Promessas e erros
-
A captura e finalmente métodos
fazendo promessas
, Assim,
Fazendo promessas
-
Promessas em sequência
Visão geral de
, Assim,
Promessas
operações paralelas
, Assim,
Promessas em paralelo
Promessas em sequência
, Assim,
Promessas em sequência
-
Promessas em
Sequência
resolvendo promessas
, Assim,
Resolvendo promessas
-
Mais sobre promessas
e erros
retornando dos retornos de chamada de promessa
, Assim,
A captura e finalmente
Métodos
terminologia
, Assim,
Lidar com erros com promessas
usando
, Assim,
Usando promessas
-
Lidar com erros com promessas
APIs de áudio
AUDIO () construtor
, Assim,
O construtor Audio ()

```
console
.
registro
(
`Copiando arquivo
$ {
de
}
para
$ {
para
}
... `
);
CopyFile
(
de
, Assim,

para
, Assim,

errar

=>

{

se

(
errar
)

{

console
.
erro
(
errar
);

}

outro

{

console
.
registro
(
"feito."
);

}
});
Modo pausado
```

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 o
--escrever
opção, vai simplesmente
reformate o arquivo especificado em vigor em vez de imprimir um reformado
versão.Se você usa
git
Para gerenciar seu código -fonte, você pode invocar
Mais bonito com o
--escrever
opção em um gancho de commit 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

manipulação de pixels
, Assim,
Manipulação de pixels
Sensibilidade ao caso
, Assim,
O texto de um programa JavaScript
Pegue cláusulas
, Assim,
tente/capturar/finalmente
-
Declarações diversas
Catch declarações
, Assim,
Classes de erro
Método .Catch ()
, Assim,
A captura e finalmente métodos
-
A captura e finalmente
Métodos
Classes de personagens (expressões regulares)
, Assim,
Classes de personagens
Histogramas de frequência de caracteres
, Assim,
Exemplo: frequência do personagem
Histogramas
-
Resumo
Método Charat ()
, Assim,
Cordas como matrizes
Função de checkscope ()
, Assim,
Fechamentos
Processos Infantis (Nó)
, Assim,
Trabalhando com processos filhos
-
garfo()
benefícios de
, Assim,
Trabalhando com processos filhos
exec () e execfile ()
, Assim,
exec () e execfile ()
execsync () e execfilesync ()
, Assim,
execsync () e execfilesync ()
garfo()
, Assim,
garfo()
opções
, Assim,
execsync () e execfilesync ()
Spawn ()
, Assim,
Spawn ()
Declaração de classe

Adicionando métodos às classes existentes
, Assim,
Adicionando métodos aos existentes
Classes
classes e construtores
, Assim,
Classes e construtores
-
O
Propriedade do construtor
Propriedade do construtor
, Assim,
A propriedade do construtor
construtores, identidade de classe e instância
, Assim,
Construtores,
Identidade de classe e instanceof
expressão new.Target
, Assim,
Classes e construtores
classes e protótipos
, Assim,
Classes e protótipos
Aulas com palavra -chave de classe
, Assim,
Aulas com a palavra -chave da classe
-
Exemplo: uma classe de números complexos
Exemplo de classe de números complexos
, Assim,
Exemplo: um complexo
Classe de número
-
Exemplo: uma classe de números complexos
getters, setters e outros formulários de método
, Assim,
Getters, setters e
Outros formulários de método
Campos públicos particulares e estáticos
, Assim,
Público, privado e estático
Campos
-
Campos públicos, privados e estáticos
Métodos estáticos
, Assim,
Métodos estáticos
programação modular com
, Assim,
Módulos com classes, objetos e
Fechamentos
nomeação
, Assim,
Palavras reservadas
Visão geral de
, Assim,
Visão geral e definições
, Assim,
Classes

Experimentando o código JavaScript
, Assim,
Explorando JavaScript
Ordem de agrupamento
, Assim,
Comparando strings
cores
, Assim,
Cores, padrões e gradientes
operador de vírgula (,)
, Assim,
O operador de vírgula (,)
Comentários
Sintaxe para
, Assim,
Um passeio de JavaScript
, Assim,
Comentários
Sintaxe em exemplos de código
, Assim,
Um passeio de JavaScript
Método Compare ()
, Assim,
Comparando strings
operadores de comparação
, Assim,
Operadores de comparação
composição
, Assim,
Translucidez e composição
declarações compostas
, Assim,
Declarações compostas e vazias
Propriedades computadas
, Assim,
Nomes de propriedades computadas
Método concat ()
, Assim,
Adicionando matrizes com concat ()
Operador de acesso condicional (?.)
, Assim,
Um passeio de JavaScript
, Assim,
Função
Invocação
Invocação condicional
, Assim,
Invocação condicional
, Assim,
Invocação de funções
operador condicional (? :)
, Assim,
O operador condicional (? :)
Declarações condicionais
, Assim,
Declarações
, Assim,
Condicionais
-

funções definidas por
, Assim,
A API do console
-
A API do console
suporte para
, Assim,
A API do console
função console.log ()
, Assim,
Olá mundo
, Assim,
Saída do console
palavra -chave const
, Assim,
Declarações com Let and Const
constantes
declarando
, Assim,
Visão geral e definições
, Assim,
Declarações com let e
const
, Assim,
const, let e var
Definição de termo
, Assim,
Declaração e atribuição variáveis
nomeação
, Assim,
Palavras reservadas
construtores
Array () construtor
, Assim,
O construtor da matriz ()
AUDIO () construtor
, Assim,
O construtor Audio ()
Aulas e
, Assim,
Classes e construtores
-
A propriedade do construtor
Propriedade do construtor
, Assim,
A propriedade do construtor
-
O
Propriedade do construtor
construtores, identidade de classe e instância
, Assim,
Construtores,
Identidade de classe e instanceof
expressão new.Target
, Assim,
Classes e construtores
Invocação do construtor
, Assim,
Invocação do construtor

Cabeçalho HTTP de políticas de segurança de conteúdo
, Assim,
Segurança
continue declarações
, Assim,
continuar
Estruturas de controle
, Assim,
Um passeio de JavaScript
-
Um passeio de JavaScript
, Assim,
Declarações
biscoitos
API para manipular
, Assim,
Biscoitos
Definição de termo
, Assim,
Biscoitos
Atributos de vida e escopo
, Assim,
Atributos de biscoito: vida e escopo
limitações de
, Assim,
Atributos de biscoito: vida e escopo
Origem do nome
, Assim,
Biscoitos
leitura
, Assim,
Lendo cookies
armazenando
, Assim,
Armazenando biscoitos
Transformações do sistema de coordenadas
, Assim,
Transformações do sistema de coordenadas
-
Exemplo de transformação
símbolo de direitos autorais (\ xa9)
, Assim,
Sequências de fuga em literais de cordas
Método copyWithin ()
, Assim,
CopyWithin ()
API de gerenciamento de credenciais
, Assim,
Criptografia e APIs relacionadas
Compartilhamento de Recursos Cross-Origin (CORS)
, Assim,
A política da mesma origem
, Assim,
Solicitações de origem cruzada
Scripts de sites cruzados (XSS)
, Assim,
Script de câmara cruzada
criptografia
, Assim,

Folhas de estilo CSS
Estilos CSS comuns
, Assim,
Scripts CSS
Estilos computados
, Assim,
Estilos computados
Animações e eventos CSS
, Assim,
Animações e eventos CSS
Classes CSS
, Assim,
Classes CSS
Sintaxe do seletor CSS
, Assim,
Selecionando elementos com seletores CSS
Estilos embutidos
, Assim,
Estilos embutidos
Convenções de nomeação
, Assim,
Estilos embutidos
folhas de estilo de script
, Assim,
Folhas de estilo de script
Objeto CSSStyleleDeclaration
, Assim,
Estilos embutidos
Brace Curly ({})
, Assim,
Um passeio de JavaScript
, Assim,
Inicializadores de objetos e matrizes
moeda
, Assim,
Números de formatação
curvas
, Assim,
Curvas
D
propriedades de dados
, Assim,
Propriedade Getters and Setters
Classe DataView
, Assim,
DataView e Endianness
Tipo de data
, Assim,
Visão geral e definições
, Assim,
Datas e tempos
datas e tempos
data aritmética
, Assim,
Data aritmética
Strings de data de formatação e análise
, Assim,
Data de formatação e análise
Cordas

```
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
, Assim,

ligar de volta
)

{

fs
.
ReadFile
(
caminho
, Assim,

"UTF8"
, Assim,

(
errar
, Assim,

texto
)

=>

{

se

(
errar
)

{

// algo deu errado lendo o

arquivo

console
.
erro
(
errar
);

ligar de volta
(
```

Evento de dispositivo de dispositivo
, Assim,
APIs de dispositivo móvel
DevicePixelratio Propriedade
, Assim,
Documentar coordenadas e viewport
Coordenadas
dicionários
, Assim,
Introdução aos objetos
, Assim,
Objetos como matrizes associativas
diretórios (nó)
, Assim,
Trabalhando com diretórios
função a distância ()
, Assim,
Declarações de função
Operador de divisão (/)
, Assim,
Aritmética em javascript
, Assim,
Expressões aritméticas
fazer/enquanto loops
, Assim,
faça/while
Documentar geometria e rolagem
, Assim,
Documentar geometria e rolagem
-
Tamanho da viewport, tamanho de conteúdo e posição de rolagem
CSS Pixels
, Assim,
Documentar coordenadas e coordenadas de viewport
Determinando elemento em um ponto
, Assim,
Determinando o elemento em um
Apontar
Documentar coordenadas e coordenadas de viewport
, Assim,
Documento
Coordena e coordenadas de viewport
GEOMETRIA DE CONSUMENTO DE ELEMENTOS
, Assim,
Consulta a geometria de um
Elemento
rolando
, Assim,
Rolando
Tamanho da viewport, tamanho de conteúdo e posição de rolagem
, Assim,
Tamanho da viewport,
Tamanho do conteúdo e posição de rolagem
Modelo de objeto de documento (DOM)
, Assim,
O modelo de objeto de documento
-
Exemplo: gerando um índice
estrutura de documentos e travessia

geração dinamicamente tabelas de conteúdo
, Assim,
Exemplo: gerando a
Índice
elementos iframe
, Assim,
Documentar coordenadas e viewport
Coordenadas
Modificando conteúdo
, Assim,
Conteúdo do elemento como html
estrutura modificadora
, Assim,
Criando, inserindo e excluindo nós
Visão geral de
, Assim,
Documentos de script
consulta e definição de atributos
, Assim,
Atributos
selecionando elementos do documento
, Assim,
Selecionando elementos do documento
Shadow Dom
, Assim,
Shadow Dom
-
Shadow Dom API
Nós de documentário
, Assim,
Usando componentes da web
documentos, carregando novos
, Assim,
Carregando novos documentos
sinal de dólar ($)
, Assim,
Identificadores e palavras reservadas
Evento DomContentLoaded
, Assim,
Execução de programas JavaScript
, Assim,
Cliente-
Linha do tempo do JavaScript lateral
Operador de pontos (.)
, Assim,
Um passeio de JavaScript
, Assim,
Consulta e configuração de propriedades
Citações duplas (")
, Assim,
Literais de cordas
barras duplas (//)
, Assim,
Um passeio de JavaScript
, Assim,
Um passeio de JavaScript
, Assim,
Comentários
função drawImage ()

retângulos
, Assim,
Retângulos
texto
, Assim,
Texto
matrizes dinâmicas
, Assim,
Matrizes
E
ECMA402 padrão
, Assim,
A API de internacionalização
ECMAScript (s)
, Assim,
Introdução ao JavaScript
Método elementFromPoint ()
, Assim,
Documentar coordenadas e viewport
Coordenadas
elementos
elementos da matriz
Definição de termo
, Assim,
Matrizes
leitura e escrita
, Assim,
Leitura e escrevendo elementos de matriz
elementos do documento
elementos personalizados
, Assim,
Elementos personalizados
Determinando elemento em um ponto
, Assim,
Determinando o elemento em um
Apontar
iframe
, Assim,
Documentar coordenadas e coordenadas de viewport
GEOMETRIA DE CONSUMENTO DE ELEMENTOS
, Assim,
Consultar a geometria de
um elemento
selecionando
, Assim,
Selecionando elementos do documento
Método Ellipse ()
, Assim,
Curvas
caso contrário, declarações
, Assim,
caso contrário, se
emojis
, Assim,
Unicode
, Assim,
Sequências de fuga em literais de cordas

declarações vazias
, Assim,
Declarações compostas e vazias
cordas vazias
, Assim,
Texto
função codeuri ()
, Assim,
Funções do URL do Legado
função codeuricomponent ()
, Assim,
Funções do URL do Legado
Contrações em inglês
, Assim,
Literais de cordas
atributo enumerável
, Assim,
Introdução aos objetos
, Assim,
Atributos da propriedade
Operador de igualdade (==)
Visão geral de
, Assim,
Operadores de igualdade e desigualdade
Digite conversões
, Assim,
Visão geral e definições
, Assim,
Conversões e
Igualdade
, Assim,
Conversões especiais de operadoras de casos
operadores de igualdade
, Assim,
Um passeio de JavaScript
Classes de erro
, Assim,
Classes de erro
manuseio de erros
usando promessas
, Assim,
Lidar com erros com promessas
, Assim,
Mais sobre promessas
e erros
ambiente de host de navegador da web
, Assim,
Erros de programa
ES2016
Operador de exponenciação (**)
, Assim,
Aritmética em javascript
, Assim,
Aritmética
Expressões
Inclui () método
, Assim,
inclui ()
ES2017, Palavras -chave assíncronas e aguardam

Iterador assíncrono
, Assim,
Iteração assíncrona com para/aguardar
, Assim,
Iteração assíncrona
Destrutura com parâmetros de descanso
, Assim,
Função de destruição
Argumentos nos parâmetros
.Finalmente () método
, Assim,
A captura e finalmente métodos
expressões regulares
ASSERÇÕES LOLHEBEHIND
, Assim,
Especificando a posição da correspondência
Nomeados grupos de captura
, Assim,
Alternância, agrupamento e referências
s Flag
, Assim,
Bandeiras
Classes de caracteres Unicode
, Assim,
Classes de personagens
Operador espalhado (…)
, Assim,
Operador espalhado
, Assim,
Função de destruição
Argumentos nos parâmetros
, Assim,
Iteradores e geradores
ES2019
cláusulas de captura nua
, Assim,
tente/capturar/finalmente
Arrays achatados
, Assim,
Arrays achatados com plano () e plangmap ()
ES2020
?operador
, Assim,
Primeiro definido (??)
Tipo bigint
, Assim,
Inteiros de precisão arbitrária com bigint
Bigint64Array ()
, Assim,
Tipos de matriz digitados
Biguint64Array ()
, Assim,
Tipos de matriz digitados
Operador de acesso condicional (?.)
, Assim,
Um passeio de JavaScript
, Assim,
Propriedade
Erros de acesso

global
, Assim,
O objeto global
importação () função
, Assim,
Importações dinâmicas com importação ()
LastIndex e Regexp API
, Assim,
exec ()
método matchall ()
, Assim,
Matchall ()
, Assim,
exec ()
, Assim,
Implementando iterável
Objetos
precedência do operador
, Assim,
Precedência do operador
Promise.AllSettled ()
, Assim,
Promessas em paralelo
Expressões de acesso à propriedade
, Assim,
Acesso à propriedade condicional
ES5
Método Aplicar ()
, Assim,
Os métodos de chamada () e aplicar ()
Breaking strings em várias linhas
, Assim,
Literais de cordas
, Assim,
Escapar
Sequências em literais de string
Bugs abordados por variáveis nnescopidas em bloco
, Assim,
Fechamentos
linha de base de compatibilidade
, Assim,
Introdução ao JavaScript
Método function.bind ()
, Assim,
A propriedade do construtor
getters e setters
, Assim,
Propriedade Getters and Setters
IE11 Solução alternativa
, Assim,
Módulos JavaScript na web
transpilação com Babel
, Assim,
Transpilação com Babel
ES6
Função de Array.of ()
, Assim,
Array.of ()
Funções de seta

Declaração de classe
, Assim,
aula
Palavra -chave da classe
, Assim,
Aulas com a palavra -chave da classe
-
Exemplo: a
Classe de números complexos
Propriedades computadas
, Assim,
Nomes de propriedades computadas
sintaxe literal de objeto estendido
, Assim,
Propriedades de abreviação
para/de loops
, Assim,
para/de
-
para/in
IE11 Solução alternativa
, Assim,
Módulos JavaScript na web
strings iteráveis ■■em
, Assim,
Texto
matrizes de iteração
, Assim,
Matrizes de iteração
Objeto de matemática
, Assim,
Aritmética em javascript
módulos em
importações dinâmicas com importação ()
, Assim,
Importações dinâmicas com
importar()
exportações
, Assim,
Es6 exportações
Import.Meta.url
, Assim,
Import.Meta.url
importações
, Assim,
ES6 importações
-
ES6 importações
importações e exportações com renomeação
, Assim,
Importações e exportações com
Renomear
Módulos JavaScript na web
, Assim,
Módulos JavaScript no
Web
-
Módulos JavaScript na web
Visão geral de

encadear promessas
, Assim,
Encadear promessas
-
Encadear promessas
manuseio de erros com
, Assim,
Mais sobre promessas e erros
-
A captura
e finalmente métodos
fazendo promessas
, Assim,
Fazendo promessas
-
Promessas em sequência
Visão geral de
, Assim,
Promessas
operações paralelas
, Assim,
Promessas em paralelo
Promessas em sequência
, Assim,
Promessas em sequência
-
Promessas em
Sequência
resolvendo promessas
, Assim,
Resolvendo promessas
-
Mais sobre promessas
e erros
retornando dos retornos de chamada de promessa
, Assim,
A captura e finalmente
Métodos
usando
, Assim,
Usando promessas
-
Lidar com erros com promessas
Ordem de enumeração da propriedade
, Assim,
Ordem de enumeração da propriedade
liberação de
, Assim,
Introdução ao JavaScript
Classes de conjunto e mapa
, Assim,
para/de com set e mapa
Métodos abreviados
, Assim,
Métodos abreviados
Operador espalhado (…)
, Assim,
O operador de propagação
Strings delimitadas com backsticks

Declaração variável em
, Assim,
Declaração e atribuição variáveis
rendimento* palavra -chave
, Assim,
rendimento* e geradores recursivos
Sequências de fuga
Apostróficos
, Assim,
Literais de cordas
em literais de cordas
, Assim,
Sequências de fuga em literais de cordas
Unicode
, Assim,
Sequências de Escape Unicode
função escape ()
, Assim,
Funções do URL do Legado
Eslint
, Assim,
LING COM ESLINT
EVAL () função
, Assim,
Expressões de avaliação
-
EVAL
Global Eval ()
, Assim,
Global Eval ()
EVAL
, Assim,
EVAL
Expressões de avaliação
, Assim,
Expressões de avaliação
-
EVAL
ouvintes de eventos
, Assim,
Eventos
, Assim,
Eventos
Modelo de programação orientado a eventos
, Assim,
JavaScript assíncrono
, Assim,
Eventos
-
Despacha eventos personalizados
, Assim,
Eventos enviados ao servidor
Definição de termo
, Assim,
JavaScript assíncrono
Despacha eventos personalizados
, Assim,
Despacha eventos personalizados
Cancelamento de eventos

Registrando manipuladores de eventos
, Assim,
Registrando manipuladores de eventos
eventos enviados ao servidor
, Assim,
Eventos enviados ao servidor
Recursos da plataforma da web para investigar
, Assim,
Eventos
Classe de EventEmitter
, Assim,
Eventos e EventEmitter
todo () método
, Assim,
cada () e alguns ()
exceções, jogando e pegando
, Assim,
lançar
método EXEC ()
, Assim,
exec ()
notação exponencial
, Assim,
Literais de ponto flutuante
Operador de exponenciação (**)
, Assim,
Aritmética em javascript
, Assim,
Aritmética
Expressões
declaração de exportação
, Assim,
importação e exportação
Exportar palavra -chave
, Assim,
Módulos em ES6
declarações de expressão
, Assim,
Declarações de expressão
expressões
Expressões aritméticas
, Assim,
Expressões aritméticas
-
Operadores bitwise
Expressões de atribuição
, Assim,
Expressões de atribuição
-
Atribuição
com operação
Definição de termo
, Assim,
Expressões e operadores
Incorporação em literais de cordas
, Assim,
Literais de cordas
Expressões de avaliação
, Assim,

expressões de função
, Assim,
Expressões de função
, Assim,
Funções como
Namespaces
expressão inicializadora
, Assim,
Um passeio de JavaScript
, Assim,
Objeto e matriz
Inicializadores
Expressões de invocação
, Assim,
Expressões de invocação
-
Condicional
Invocação
, Assim,
Invocação de funções
-
Invocação do construtor
expressões lógicas
, Assim,
Expressões lógicas
-
Lógico não (!)
expressão new.Target
, Assim,
Classes e construtores
Inicializadores de objetos e matrizes
, Assim,
Inicializadores de objetos e matrizes
Expressões de criação de objetos
, Assim,
Expressões de criação de objetos
expressões primárias
, Assim,
Expressões primárias
Expressões de acesso à propriedade
, Assim,
Expressões de acesso à propriedade
-
Acesso à propriedade condicional
expressões relacionais
, Assim,
Expressões relacionais
-
A instância
Operador
Versus declarações
, Assim,
Um passeio de JavaScript
, Assim,
Declarações
extensibilidade
, Assim,
Extensibilidade do objeto
F

abortando solicitações
, Assim,
Abortando um pedido
solicitações de origem cruzada
, Assim,
Solicitações de origem cruzada
exemplos de
, Assim,
buscar()
upload de arquivo
, Assim,
Upload de arquivo com fetch ()
Códigos de status HTTP, cabeçalhos de resposta e erros de rede
, Assim,
Http
Códigos de status, cabeçalhos de resposta e erros de rede
opções de solicitação diversas
, Assim,
Opções de solicitação diversas
analisar os corpos de resposta
, Assim,
Analisar os corpos de resposta
Definindo cabeçalhos de solicitação
, Assim,
Definindo cabeçalhos de solicitação
Definindo parâmetros de solicitação
, Assim,
Definindo parâmetros de solicitação
Especificando o método de solicitação e o corpo de solicitação
, Assim,
Especificando o
solicitar método e solicitar o corpo
Etapas de
, Assim,
buscar()
corpos de resposta de streaming
, Assim,
Corpos de resposta de streaming
campos, público, privado e estático
, Assim,
Campos públicos, privados e estáticos
Manuseio de arquivos (nó)
, Assim,
Trabalhando com arquivos
-
Trabalhando com diretórios
diretórios
, Assim,
Trabalhando com diretórios
metadados do arquivo
, Assim,
Metadados do arquivo
Strings de modo de arquivo
, Assim,
Escrevendo arquivos
operações de arquivo
, Assim,
Operações de arquivo
Visão geral de

leitura de arquivos
, Assim,
Leitura de arquivos
escrevendo arquivos
, Assim,
Escrevendo arquivos
método de preenchimento ()
, Assim,
preencher()
Método filtro ()
, Assim,
filtro()
.Finalmente () método
, Assim,
A captura e finalmente métodos
-
A captura e finalmente
Métodos
números de conta financeira
, Assim,
Armazenar
Método Find ()
, Assim,
encontre () e findIndex ()
Método FindIndex ()
, Assim,
encontre () e findIndex ()
Ferramentas de desenvolvedor do Firefox
, Assim,
Explorando JavaScript
operador primeiro definido (??)
, Assim,
Primeiro definido (??)
Método plano ()
, Assim,
Arrays achatados com plano () e plangmap ()
Método Flatmap ()
, Assim,
Arrays achatados com plano () e plangmap ()
Literais de ponto flutuante
, Assim,
Literais de ponto flutuante
, Assim,
Ponto flutuante binário
e erros de arredondamento
Extensão da linguagem de fluxo
, Assim,
Verificação de tipo com fluxo
-
Enumerado
Tipos e sindicatos discriminados
Tipos de matriz
, Assim,
Tipos de matriz
Tipos de classe
, Assim,
Tipos de classe
tipos enumerados e sindicatos discriminados
, Assim,

tipos de objetos
, Assim,
Tipos de objetos
Outros tipos parametrizados
, Assim,
Outros tipos parametrizados
Visão geral de
, Assim,
Verificação de tipo com fluxo
Tipos somente leitura
, Assim,
Tipos somente leitura
Tipo Aliases
, Assim,
Tipo Aliases
TypeScript versus Flow
, Assim,
Verificação de tipo com fluxo
Tipos de sindicatos
, Assim,
Tipos de sindicatos
usando anotações de tipo
, Assim,
Usando anotações de tipo
para loops
, Assim,
para
, Assim,
Matrizes de iteração
para/aguarda loops
, Assim,
Iteração assíncrona com para/aguardar
, Assim,
Assíncrono
Iteração
para/em loops
, Assim,
para/in
, Assim,
Propriedades enumeradas
para/de loops
, Assim,
Texto
, Assim,
para/de
-
para/in
, Assim,
Matrizes de iteração
, Assim,
Iteradores e
Geradores
Método foreach ()
, Assim,
Matrizes de iteração
, Assim,
foreach ()
Método Format ()
, Assim,

Function () construtor
, Assim,
O construtor function ()
função* palavra -chave
, Assim,
Geradores
funções
Funções de seta
, Assim,
Um passeio de JavaScript
, Assim,
Definindo funções
, Assim,
Seta
Funções
Sensibilidade ao caso
, Assim,
O texto de um programa JavaScript
fechamentos
, Assim,
Fechamentos
-
Fechamentos
definindo
, Assim,
Definindo funções
-
Funções aninhadas
Definindo suas próprias propriedades de função
, Assim,
Definindo o seu próprio
Propriedades da função
funções de fábrica
, Assim,
Classes e protótipos
Funções argumentos e parâmetros
Tipos de argumento
, Assim,
Tipos de argumento
objeto de argumentos
, Assim,
O objeto de argumentos
Destructar os argumentos da função em parâmetros
, Assim,
Destructar os argumentos da função em parâmetros
-
Destructar os argumentos da função em parâmetros
parâmetros e padrões opcionais
, Assim,
Parâmetros opcionais e
Padrões
Visão geral de
, Assim,
Funções argumentos e parâmetros
Parâmetros de descanso
, Assim,
Parâmetros de descanso e comprimento de variável
Listas de argumentos
Espalhe o operador para chamadas de função

listas de argumentos de comprimento variável
, Assim,
Parâmetros de descanso e variável-
Listas de argumentos de comprimento
Expressões de definição de função
, Assim,
Expressões de definição de função
Invocação de funções
, Assim,
Criando objetos com novo
propriedades de função, métodos e construtor
, Assim,
Função
Propriedades, métodos e construtor
-
O construtor function ()
método bind ()
, Assim,
O método bind ()
Call () e Aplicar () métodos
, Assim,
Os métodos de chamada () e aplicar ()
Function () construtor
, Assim,
O construtor function ()
propriedade de comprimento
, Assim,
A propriedade de comprimento
Propriedade do nome
, Assim,
A propriedade Nome
Propriedade do protótipo
, Assim,
A propriedade do protótipo
Método ToString ()
, Assim,
O método tostring ()
Programação funcional
explorando
, Assim,
Programação funcional
Funções de ordem superior
, Assim,
Funções de ordem superior
memórias
, Assim,
Memórias
Aplicação parcial de funções
, Assim,
Aplicação parcial de
Funções
Processando matrizes com função
, Assim,
Processando matrizes com
Funções
Funciona como namespaces
, Assim,
Funciona como namespaces
funciona como valores

invocando
abordagens para
, Assim,
Invocando funções
Invocação do construtor
, Assim,
Invocação do construtor
exemplos
, Assim,
Um passeio de JavaScript
Invocação de função implícita
, Assim,
Invocação de função implícita
Invocação indireta
, Assim,
Invocação indireta
Expressões de invocação
, Assim,
Invocação de funções
Invocação do método
, Assim,
Invocação do método
-
Invocação do método
nomeação
, Assim,
Palavras reservadas
Visão geral de
, Assim,
Visão geral e definições
, Assim,
Funções
funções recursivas
, Assim,
Invocação de funções
Sintaxe abreviada para
, Assim,
Um passeio de JavaScript
Funções de matriz estática
, Assim,
Funções de matriz estática
G
coleta de lixo
, Assim,
Visão geral e definições
Funções do gerador
, Assim,
Geradores
, Assim,
O valor de retorno de um gerador
Função
(
ver
também iteradores e geradores)
API de geolocalização
, Assim,
APIs de dispositivo móvel
Método GetBoundingClientRect ()
, Assim,

diferenciado com base em quão configuráveis nneles 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.
Programas podem usar
importar()
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
apoiar
importar()
pode ser capaz de produzir vários 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()
no seu programa e eles
carregar
Módulos 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.
Matores geralmente podem produzir um
mapa de origem
arquivo que define um
mapeamento entre as linhas de código no pacote e o

comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
maior ou igual ao operador (> =)
Visão geral de
, Assim,
Operadores de comparação
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
H
HashChange Events
, Assim,
Gerenciamento de história com eventos de hashchange
hashtables
, Assim,
Introdução aos objetos
, Assim,
Objetos como matrizes associativas
Operador HasownProperty
, Assim,
Propriedades de teste
Olá mundo
, Assim,
Olá mundo
, Assim,
Saída do console
Literais hexadecimais
, Assim,
Literais inteiros
, Assim,
Sequências de fuga em string
Literais
Funções de ordem superior
, Assim,
Funções de ordem superior
Histogramas, frequência do caractere
, Assim,
Exemplo: frequência do personagem
Histogramas
-
Resumo
Método History.pushstate ()
, Assim,
Gerenciamento de história com pushState ()
Método history.Replacestate ()
, Assim,
Gerenciamento de história com pushState ()
içar
, Assim,
Declarações variáveis ■■com VAR
Tags html <cript>
, Assim,

número a ser dígito
Quando o fluxo verifica seus tipos, ele realmente não faz os cálculos:
Apenas verifica os tipos de cálculos.O fluxo sabe disso
tolowerCase ()
retorna uma string e que o
+
operador em números
Retorna um número.Mesmo sabendo que ambos os cálculos
Retornar valores que estão dentro do tipo, o fluxo não pode saber disso e sinalizadores
erros em ambas as linhas.
Um tipo de sindicato de tipos literais como
Responder
e
Digit
é um exemplo de
um
Tipo enumerado
, ou
enum
.Um caso de uso canônico para tipos de enum é
Representar os ternos das cartas de jogo:
tipo

Terno

=

"Clubes"

|

"Diamantes"

|

"Corações"

|

"Espadas"
;
Um exemplo mais relevante pode ser os códigos de status HTTP:
tipo

Httpstatus

=

|

200

// OK

|

304

// não modificado

imutabilidade
, Assim,
Trabalhando com cordas
, Assim,
Valores primitivos imutáveis ■■e
Referências de objetos mutáveis
Invocação de função implícita
, Assim,
Invocação de função implícita
declaração de importação
, Assim,
importação e exportação
Importar palavra -chave
, Assim,
Módulos em ES6
importação () função
, Assim,
Importações dinâmicas com importação ()
Import.Meta.url
, Assim,
Import.Meta.url
no operador
, Assim,
O operador in
, Assim,
Propriedades de teste
Inclui () método
, Assim,
inclui ()
Operador de incremento (++)
, Assim,
Operadores aritméticos unários
Posição do índice
, Assim,
Matrizes
, Assim,
Leitura e escrevendo elementos de matriz
Indexeddb
, Assim,
Indexeddb
-
Tópicos de trabalhadores e mensagens
Método Indexof ()
, Assim,
indexOf () e LastIndexOf ()
Invocação indireta
, Assim,
Invocação indireta
Operador de desigualdade (! ==)
valores booleanos
, Assim,
Valores booleanos
Visão geral de
, Assim,
Operadores de igualdade e desigualdade
comparação de string
, Assim,
Trabalhando com cordas
valor infinito

métodos de instância
, Assim,
Métodos estáticos
Instância do operador
, Assim,
A instância do operador
, Assim,
Construtores, classe
Identidade e instanceof
Literais inteiros
, Assim,
Literais inteiros
API de internacionalização
Aulas incluídas em
, Assim,
A API de internacionalização
Comparando strings
, Assim,
Comparando strings
-
Comparando strings
Datas e tempos de formatação
, Assim,
Datas e tempos de formatação
-
Datas e tempos de formatação
Números de formatação
, Assim,
Números de formatação
-
Números de formatação
suporte para no nó
, Assim,
A API de internacionalização
texto traduzido
, Assim,
A API de internacionalização
interpolação
, Assim,
Literais de cordas
Intl.DateTimeFormat Class
, Assim,
Datas e tempos de formatação
-
Formatação
Datas e tempos
Intl.NumberFormat Class
, Assim,
Números de formatação
-
Números de formatação
Expressões de invocação
Invocação condicional
, Assim,
Invocação condicional
, Assim,
Função
Invocação
Invocação do método

iteradores e geradores
(
ver
também métodos de iterador de matriz)
Recursos avançados do gerador
Valor de retorno das funções do gerador
, Assim,
O valor de retorno de um
Função do gerador
Métodos de retorno () e Throw ()
, Assim,
O retorno () e o arremesso ()
Métodos de um gerador
valor das expressões de rendimento
, Assim,
O valor de uma expressão de rendimento
assíncrono
, Assim,
Iteradores assíncronos
-
Implementação
Iteradores assíncronos
fechando iteradores
, Assim,
“Fechando” um iterador: o método de retorno
geradores
benefícios de
, Assim,
Uma nota final sobre geradores
criando
, Assim,
Geradores
Definição de termo
, Assim,
Geradores
exemplos de
, Assim,
Exemplos de gerador
rendimento* e geradores recursivos
, Assim,
rendimento* e recursivo
Geradores
como os iteradores funcionam
, Assim,
Como os iteradores funcionam
implementando objetos iteráveis
, Assim,
Implementando objetos iteráveis
-
Implementando objetos iteráveis
Visão geral de
, Assim,
Iteradores e geradores
J
JavaScript

benefícios de
, Assim,
Introdução ao JavaScript
, Assim,
Resumo
Introdução a
Visão geral do capítulo
, Assim,
Um passeio de JavaScript
, Assim,
Um tour de
JavaScript
Histogramas de frequência de caracteres
, Assim,
Exemplo: Personagem
Histogramas de frequência
-
Resumo
Olá mundo
, Assim,
Olá mundo
história de
, Assim,
JavaScript em navegadores da web
Interpretadores JavaScript
, Assim,
Explorando JavaScript
estrutura lexical
, Assim,
Estrutura lexical
-
Resumo
nomes, versões e modos
, Assim,
Introdução ao JavaScript
Sintaxe e recursos
, Assim,
Um passeio de JavaScript
-
Um tour de
JavaScript
Documentação de referência
, Assim,
Prefácio
Biblioteca padrão JavaScript
API do console
, Assim,
A API do console
-
Saída formatada com console
datas e tempos
, Assim,
Datas e tempos
-
Data de formatação e análise
Cordas
Classes de erro
, Assim,
Classes de erro

O DOM foi introduzido em
§15.1.2
.Esta seção explica a API em
detalhe.Isto
capas:
Como consultar ou
Selecione
elementos individuais de um documento.
Como fazer
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
Lado do cliente
Os programas JavaScript geralmente precisam manipular um ou mais
elementos dentro do documento.O global
documento
Propriedade 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
Folhas de estilo CSS
ter uma sintaxe muito poderosa, conhecida como
Seletores
, para
descrevendo elementos ou conjuntos de elementos em um documento.O
Dom
Métodos
QuerySelector ()
e
QuerySelectorAll ()
permitir
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.

K
palavras -chave
palavra -chave assíncrona
, Assim,
assíncrono e aguardar
-
Detalhes da implementação
aguarde palavras -chave
, Assim,
assíncrono e aguardar
-
Detalhes da implementação
Sensibilidade ao caso
, Assim,
O texto de um programa JavaScript
Palavra -chave da classe
, Assim,
Aulas com a palavra -chave da classe
-
Exemplo: a
Classe de números complexos
palavra -chave const
, Assim,
Declarações com Let and Const
Exportar palavra -chave
, Assim,
Módulos em ES6
Palavra -chave da função
, Assim,
Definindo funções
função* palavra -chave
, Assim,
Geradores
Importar palavra -chave
, Assim,
Módulos em ES6
Deixe a palavra -chave
, Assim,
Um passeio de JavaScript
, Assim,
Declarações com let e
const
, Assim,
const, let e var
nova palavra -chave
, Assim,
Criando objetos com novo
, Assim,
Invocação do construtor
palavras reservadas
, Assim,
Palavras reservadas
, Assim,
Expressões primárias
essa palavra -chave
, Assim,
Um passeio de JavaScript
, Assim,
Expressões primárias

Declarações rotuladas
, Assim,
Declarações rotuladas
Propriedade do LastIndex
, Assim,
exec ()
método lastIndexOf ()
, Assim,
indexOf () e LastIndexOf ()
menos que o operador (<)
Visão geral de
, Assim,
Operadores de comparação
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
menor ou igual ao operador (<=)
Visão geral de
, Assim,
Operadores de comparação
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
Deixe a palavra -chave
, Assim,
Um passeio de JavaScript
, Assim,
Declarações com Let and Const
, Assim,
const, let e var
Escopo lexical
, Assim,
Fechamentos
estrutura lexical
, Assim,
Estrutura lexical
-
Resumo
Sensibilidade ao caso
, Assim,
O texto de um programa JavaScript
Comentários
, Assim,
Comentários
identificadores
, Assim,
O texto de um programa JavaScript
-
Identificadores e
Palavras reservadas
quebras de linha
, Assim,
O texto de um programa JavaScript

espaços
, Assim,
O texto de um programa JavaScript
Conjunto de caracteres unicode
Sequências de fuga
, Assim,
Sequências de Escape Unicode
normalização
, Assim,
Normalização unicode
Visão geral de
, Assim,
Unicode
quebras de linha
, Assim,
O texto de um programa JavaScript
, Assim,
Semicolons opcionais
-
Semicolons opcionais
Estilos de linha
, Assim,
Estilos de linha
Terminadores de linha
, Assim,
O texto de um programa JavaScript
Ferramentas de linha
, Assim,
LING COM ESLINT
Literais
numérico
Literais de ponto flutuante
, Assim,
Literais de ponto flutuante
, Assim,
Binário
Ponto flutuante e erros de arredondamento
Literais inteiros
, Assim,
Literais inteiros
números negativos
, Assim,
Números
separadores em
, Assim,
Literais de ponto flutuante
expressões regulares
, Assim,
Correspondência de padrões
, Assim,
Combinação de padrões com
Expressões regulares
corda
, Assim,
Literais de cordas
Literais de modelo
, Assim,
Literais de modelo
, Assim,

"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:

```
deixar

b

=

Buffer
.
de
([
0x41
, Assim,

0x42
, Assim,

0x43
]);

// <buffer

41 42 43>
b
.
ToString
()

// =>

"ABC";padrão "utf8"
b
.
ToString
(
"Hex"
)

// =>

"414243"
deixar

computador

=

Buffer
```

marechaling
, Assim,
Serialização e análise de json
método match ()
, Assim,
corresponder()
método matchall ()
, Assim,
Matchall ()
método matches ()
, Assim,
Selecionando elementos com seletores CSS
Função math.pow
, Assim,
Expressões aritméticas
operações matemáticas
, Assim,
Aritmética em javascript
-
Aritmética em
JavaScript
Site da MDN
, Assim,
Prefácio
APIs de mídia
, Assim,
APIs de mídia
memórias
, Assim,
Memórias
Gerenciamento de memória
, Assim,
Visão geral e definições
Eventos de mensagem
, Assim,
Modelo de Threading JavaScript do lado do cliente
, Assim,
Eventos
, Assim,
Eventos enviados ao servidor
, Assim,
Objetos de trabalhador
-
O objeto global em trabalhadores
, Assim,
Modelo de execução do trabalhador
-
PostMessage (), Messageports e
Messagechannels
, Assim,
Mensagens de origem cruzada com pós-maquiagem ()
, Assim,
garfo()
-
Tópicos dos trabalhadores
, Assim,
Canais de comunicação e porportes de mensagem
, Assim,
Tipos enumerados e sindicatos discriminados

Tópicos de trabalhadores e mensagens
, Assim,
Tópicos de trabalhadores e mensagens
-
Mensagens de origem cruzada com pós-maquiagem ()
Mensagens de origem cruzada
, Assim,
Mensagens de origem cruzada com
PostMessage ()
, Assim,
Mensagens de origem cruzada com pós-maquiagem ()
Modelo de execução
, Assim,
Modelo de execução do trabalhador
Código de importação
, Assim,
Importar código para um trabalhador
Exemplo de conjunto de Mandelbrot
, Assim,
Exemplo: o conjunto Mandelbrot
-
Resumo e sugestões para leitura adicional
módulos
, Assim,
Importar código para um trabalhador
Visão geral de
, Assim,
Tópicos de trabalhadores e mensagens
PostMessage (), MessagePorts e Messagechannels
, Assim,
PostMessage (), MessagePorts e Messagechannels
Objetos de trabalhador
, Assim,
Objetos de trabalhador
Objeto Workerglobalscope
, Assim,
O objeto global em trabalhadores
metaprogramação
, Assim,
Metaprogramação
Métodos
Adicionando métodos às classes existentes
, Assim,
Adicionando métodos aos existentes
Classes
Métodos de matriz
aplicação genérica de
, Assim,
Matrizes
Visão geral de
, Assim,
Métodos de matriz
Métodos de classe versus instância
, Assim,
Métodos estáticos
criando
, Assim,
Um passeio de JavaScript
Definição de termo

encadeamento de método
, Assim,
Invocação do método
Invocação do método
, Assim,
Expressões de invocação
, Assim,
Invocação do método
-
Invocação do método
Métodos abreviados
, Assim,
Getters, setters e outros formulários de método
Sintaxe abreviada
, Assim,
Métodos abreviados
Métodos estáticos
, Assim,
Métodos estáticos
Métodos de matriz digitados
, Assim,
Métodos e propriedades da matriz digitada
Sign de menos (-)
Operador de subtração
, Assim,
Aritmética em javascript
, Assim,
Aritmética
Expressões
Operador aritmético unário
, Assim,
Operadores aritméticos unários
APIs de dispositivo móvel
, Assim,
APIs de dispositivo móvel
módulos
automatizando a modularidade baseada em fechamento
, Assim,
Automatizando baseado em fechamento
Modularidade
em ES6
importações dinâmicas com importação ()
, Assim,
Importações dinâmicas com
importar()
exportações
, Assim,
Es6 exportações
Import.Meta.url
, Assim,
Import.Meta.url
importações
, Assim,
ES6 importações
-
ES6 importações
importações e exportações com renomeação
, Assim,
Importações e exportações com

Web
-
Módulos JavaScript na web
Visão geral de
, Assim,
Módulos em ES6
reexporta
, Assim,
Reexporta
módulo FS (nó)
, Assim,
Trabalhando com arquivos
-
Trabalhando com diretórios
Diretivas de importação e exportação
, Assim,
Módulos
no nó
, Assim,
Módulos no nó
-
Módulos no estilo de nó na web
, Assim,
Módulos do nó
Exportações de nós
, Assim,
Exportações de nós
Nó importações
, Assim,
Nó importações
Módulos no estilo de nó na web
, Assim,
Módulos no estilo de nó no
Web
Visão geral de
, Assim,
Módulos
propósito de
, Assim,
Módulos
Usando em trabalhadores
, Assim,
Importar código para um trabalhador
com aulas, objetos e fechamentos
, Assim,
Módulos com classes, objetos,
e fechamentos
-
Automatizando a modularidade baseada em fechamento
Operador Modulo (%)
, Assim,
Aritmética em javascript
, Assim,
Expressões aritméticas
Operador de multiplicação (*)
, Assim,
Aritmética em javascript
, Assim,
Expressões e

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 o
--escrever
opção, vai simplesmente
reformate o arquivo especificado em vigor em vez de imprimir um reformado
versão.Se você usa
git
Para gerenciar seu código -fonte, você pode invocar
Mais bonito com o
--escrever
opção em um gancho de commit 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

corpos de resposta de streaming
, Assim,
Corpos de resposta de streaming
Visão geral de
, Assim,
Networking
eventos enviados ao servidor
, Assim,
Eventos enviados ao servidor
WebSocket API
, Assim,
WebSockets
API XMLHTTPREQUEST (XHR)
, Assim,
buscar()
nova palavra -chave
, Assim,
Criando objetos com novo
, Assim,
Invocação do construtor
expressão new.Target
, Assim,
Classes e construtores
newline (\ n)
, Assim,
Literais de cordas
, Assim,
Sequências de fuga em literais de cordas
NEWLINES
, Assim,
Semicolons opcionais
-
Semicolons opcionais
Usando para formatação de código
, Assim,
O texto de um programa JavaScript
Nó
iteração assíncrona em
, Assim,
O loop for/wait
, Assim,
O nó é
Assíncrono por padrão
-
O nó é assíncrono por padrão
benefícios de
, Assim,
Introdução ao JavaScript
, Assim,
JavaScript do lado do servidor
com nó
Tipo bigint
, Assim,
Inteiros de precisão arbitrária com bigint
buffers
, Assim,
Buffers
retornos de chamada e eventos em
, Assim,

metadados do arquivo
, Assim,
Metadados do arquivo
Strings de modo de arquivo
, Assim,
Escrevendo arquivos
operações de arquivo
, Assim,
Operações de arquivo
Visão geral de
, Assim,
Trabalhando com arquivos
Caminhos, descritores de arquivos e trabalhos de arquivo
, Assim,
Caminhos, arquivo
Descritores e trabalhos de arquivo
leitura de arquivos
, Assim,
Leitura de arquivos
escrevendo arquivos
, Assim,
Escrevendo arquivos
Clientes e servidores HTTP
, Assim,
Clientes e servidores HTTP
-
Http
Clientes e servidores
instalação
, Assim,
Explorando JavaScript
, Assim,
JavaScript do lado do servidor com
Nó
API INTL
, Assim,
A API de internacionalização
módulos em
, Assim,
Módulos no nó
-
Módulos no estilo de nó na web
Servidores e clientes de rede não-HTTP
, Assim,
Rede não-http
Servidores e clientes
paralelismo com
, Assim,
O nó é assíncrono por padrão
detalhes do processo
, Assim,
Processo, CPU e detalhes do sistema operacional
Programação básica
, Assim,
Noções básicas de programação do nó
-
O nó
Gerente de pacotes
Argumentos da linha de comando

Variáveis ■■de ambiente
módulos
, Assim,
Módulos do nó
Gerente de pacotes
, Assim,
O gerenciador de pacotes de nó
Ciclo de vida do programa
, Assim,
Ciclo de vida do programa
Documentação de referência
, Assim,
Prefácio
fluxos
, Assim,
Fluxos
-
Modo pausado
iteração assíncrona em
, Assim,
Iteração assíncrona
Visão geral de
, Assim,
Fluxos
tubos
, Assim,
Tubos
lendo com eventos
, Assim,
Lendo fluxos com eventos
tipos de
, Assim,
Fluxos
Escrevendo e lidando com a contrapressão
, Assim,
Escrevendo para fluxos e
Manuseio de contrapressão
Tópicos dos trabalhadores
, Assim,
Tópicos dos trabalhadores
-
Compartilhando matrizes digitadas entre
Tópicos
canais de comunicação e porportes de mensagem
, Assim,
Comunicação
Canais e mensagens de mensagens
criando trabalhadores e passando mensagens
, Assim,
Criando trabalhadores
e mensagens passando
Visão geral de
, Assim,
Tópicos dos trabalhadores
Compartilhando matrizes digitadas entre threads
, Assim,
Compartilhando matrizes digitadas
Entre threads
transferindo portões de mensagem e matrizes digitadas

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
exportar

tipo

FetchTextCallback

=

(
?
Erro
, Assim,

?
número
, Assim,

?
corda
)

=>

vazio
;
exportar

padrão

função

FetchText
(
url
:

corda
, Assim,

ligar de volta
:

FetchTextCallback
)

{

Literais inteiros
, Assim,
Literais inteiros
separadores em literais numéricos
, Assim,
Literais de ponto flutuante
Número () função
, Assim,
Conversões explícitas
, Assim,
Conversões explícitas
Número.isfinite () função
, Assim,
Aritmética em javascript
números, formatação para internacionalização
, Assim,
Números de formatação
-
Números de formatação
Literais numéricos
, Assim,
Números
O
objetos literais
Sintaxe estendida para
, Assim,
Sintaxe literal de objeto estendido
-
Propriedade
Getters e setters
Visão geral de
, Assim,
Inicializadores de objetos e matrizes
forma mais simples de
, Assim,
Objetos literais
Nomes de propriedades do objeto
, Assim,
Leitura e escrevendo elementos de matriz
Programação orientada a objetos
Definição de termo
, Assim,
Visão geral e definições
exemplo de
, Assim,
Um passeio de JavaScript
Object.assign () função
, Assim,
Estendendo objetos
Object.Create () função
, Assim,
Object.create ()
, Assim,
Atributos da propriedade
Object.DefineProperties () Método
, Assim,
Atributos da propriedade
Método object.DefineProperty ()
, Assim,

Object.GetownPropertyNames () função
, Assim,
Propriedades enumeradas
Object.GetownPropertySymbols () Função
, Assim,
Propriedades enumeradas
Método Object.Keys
, Assim,
para/de com objetos
Object.Keys () função
, Assim,
Propriedades enumeradas
Object.prototype
, Assim,
Protótipos
, Assim,
Métodos de objeto
objetos
Objeto de argumentos
, Assim,
O objeto de argumentos
objetos semelhantes a matrizes
, Assim,
Objetos semelhantes a matrizes
-
Objetos semelhantes a matrizes
criando
, Assim,
Criando objetos
-
Object.create ()
Excluindo propriedades
, Assim,
Excluindo propriedades
propriedades enumeradas
, Assim,
Propriedades enumeradas
sintaxe literal de objeto estendido
, Assim,
Sintaxe literal de objeto estendido
-
Propriedade Getters and Setters
estendendo objetos
, Assim,
Estendendo objetos
implementando objetos iteráveis
, Assim,
Implementando objetos iteráveis
-
Implementando objetos iteráveis
Introdução a
, Assim,
Objetos
programação modular com
, Assim,
Módulos com classes, objetos e
Fechamentos
referências de objetos mutáveis
, Assim,

```
um
.
emenda
(
1
, Assim,
2
)

// => [2,3];A agora é [1,4]
um
.
emenda
(
1
, Assim,
1
)

// => [4];a agora [1]
```

Os dois primeiros argumentos para
emenda ()
Especifique quais elementos da matriz
devem ser excluídos.Esses argumentos podem ser seguidos por qualquer número de
argumentos adicionais que especificam elementos a serem inseridos na matriz,
começando na posição especificada pelo primeiro argumento.Por exemplo:
deixar

```
um

=

[[
1
, Assim,
2
, Assim,
3
, Assim,
4
, Assim,
5
];
um
.
emenda
(
2
, Assim,
0
, Assim,
"um"
, Assim,
"B"
)

// => [];A agora é [1,2, "A", "B", 3,4,5]
um
.
```

Personagens de repetição
, Assim,
Repetição
especificando a posição da correspondência
, Assim,
Especificando a posição da correspondência
Classes de caracteres Unicode
, Assim,
Classes de personagens
Visão geral de
, Assim,
Combinação de padrões com expressões regulares
Símbolos de correspondência de padrões
, Assim,
Símbolos de correspondência de padrões
Classe regexp
método EXEC ()
, Assim,
exec ()
LastIndex Property and Regexp Reutil
, Assim,
exec ()
Visão geral de
, Assim,
A classe Regexp
Propriedades regexp
, Assim,
Propriedades regexp
teste () método
, Assim,
teste()
Métodos de string para
corresponder()
, Assim,
corresponder()
Matchall ()
, Assim,
Matchall ()
substituir()
, Assim,
substituir()
procurar()
, Assim,
Métodos de string para correspondência de padrões
dividir()
, Assim,
dividir()
Sintaxe para
, Assim,
Correspondência de padrões
padrões
, Assim,
Cores, padrões e gradientes
API de solicitação de pagamento
, Assim,
Criptografia e APIs relacionadas
APIs de desempenho
, Assim,
Desempenho

pixels
, Assim,
Documentar coordenadas e coordenadas de viewport
, Assim,
Pixel
Manipulação
mais sinal (+)
Operador de adição e atribuição (+=)
, Assim,
Atribuição com
Operação
Operador de adição
, Assim,
Aritmética em javascript
, Assim,
O operador +
Concatenação de string
, Assim,
Literais de cordas
, Assim,
Trabalhando com cordas
, Assim,
O +
Operador
Digite conversões
, Assim,
Conversões especiais de operadoras de casos
Operador aritmético unário
, Assim,
Operadores aritméticos unários
Polígonos
, Assim,
Caminhos e polígonos
-
Caminhos e polígonos
Método pop ()
, Assim,
Pilhas e filas com push (), pop (), shift () e
NIFT ()
Evento PopState
, Assim,
Categorias de eventos
, Assim,
Gerenciamento de história com
pushState ()
-
Networking
zero positivo
, Assim,
Aritmética em javascript
possessivos
, Assim,
Literais de cordas
operadores postfix
, Assim,
Semicolons opcionais
Método PostMessage ()
, Assim,
PostMessage (), Messageports e

Valores primitivos imutáveis
, Assim,
Valores primitivos imutáveis ■■e
Referências de objetos mutáveis
Tipo de número
, Assim,
Números
-
Datas e tempos
Visão geral e definições
, Assim,
Visão geral e definições
Tipo de string
, Assim,
Texto
-
Literais de modelo marcados
PrintProps () função
, Assim,
Declarações de função
Campos privados
, Assim,
Campos públicos, privados e estáticos
procedimentos
, Assim,
Funções
programas
manuseio de erros
, Assim,
Erros de programa
Execução de JavaScript
, Assim,
Execução de programas JavaScript
-
Cliente-
Linha do tempo do JavaScript lateral
Modelo de encadeamento do lado do cliente
, Assim,
Trevamento JavaScript do lado do cliente
modelo
Linha do tempo do lado do cliente
, Assim,
Linha do tempo do JavaScript do lado do cliente
entrada e saída
, Assim,
Entrada e saída do programa
Aplicativos da Web progressivos (PWAs)
, Assim,
Aplicativos e serviços progressivos da Web
Trabalhadores
Promessa cadeias
, Assim,
Promessas
, Assim,
Encadear promessas
-
Encadear promessas
Promise.all () função
, Assim,

fazendo promessas
, Assim,
Fazendo promessas
-
Promessas em sequência
baseado em outras promessas
, Assim,
Promessas baseadas em outras promessas
com base em valores síncronos
, Assim,
Promessas baseadas em
valores síncronos
do zero
, Assim,
Promessas do zero
Visão geral de
, Assim,
Promessas
operações paralelas
, Assim,
Promessas em paralelo
Promessas em sequência
, Assim,
Promessas em sequência
-
Promessas em
Sequência
resolvendo promessas
, Assim,
Resolvendo promessas
-
Mais sobre promessas e
Erros
retornando dos retornos de chamada de promessa
, Assim,
A captura e finalmente métodos
terminologia
, Assim,
Lidar com erros com promessas
usando
, Assim,
Usando promessas
-
Lidar com erros com promessas
propriedades
Nomes de propriedades computadas
, Assim,
Nomes de propriedades computadas
Acesso à propriedade condicional
, Assim,
Acesso à propriedade condicional
copiando de um objeto para outro
, Assim,
Estendendo objetos
Definindo suas próprias propriedades de função
, Assim,
Definindo o seu próprio
Propriedades da função
Definição de termo

herdando
, Assim,
Herança
nomeação
, Assim,
Símbolos
, Assim,
Introdução aos objetos
, Assim,
Símbolos como propriedade
Nomes
propriedades não herdadas
, Assim,
Introdução aos objetos
erros de acesso à propriedade
, Assim,
Erros de acesso à propriedade
Expressões de acesso à propriedade
, Assim,
Expressões de acesso à propriedade
Atributos da propriedade
, Assim,
Introdução aos objetos
, Assim,
Atributos da propriedade
-
Atributos da propriedade
Descritores de propriedades
, Assim,
Atributos da propriedade
Propriedade Getters and Setters
, Assim,
Propriedade Getters and Setters
consulta e configuração
, Assim,
Consulta e configuração de propriedades
-
Propriedade
Erros de acesso
teste
, Assim,
Propriedades de teste
Propriedades da matriz digitada
, Assim,
Métodos e propriedades da matriz digitada
Método PropertyisEnumerable ()
, Assim,
Propriedades de teste
herança prototípica
, Assim,
Introdução aos objetos
, Assim,
Herança
Cadeias de protótipo
, Assim,
Protótipos
protótipos
, Assim,
Protótipos

```
// Este segundo teste verifica que GetTemperature ()

convertidos

// Celsius para Fahrenheit corretamente

teste
(
"Converte C para F corretamente"
, Assim,

assíncrono

()

=>

{

getjson
.
MockResolvedValue
(
0
);

// Se

getjson retorna 0c

esperar
(
aguarde

gettemperature
(
"X"
)).
ser
(
32
);

// Nós

Espere 32f

// 100c deve se converter para 212f

getjson
.
MockResolvedValue
(
100
);

// Se

getjson retorna 100c
```

5.4.2 Do/while
O

faça/while
Loop é como um
enquanto
loop, 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
faça/while
loop é menos comumente usado do que o seu
enquanto
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
faça/while
laço:
função

PrintArray
(
um
)

{

deixar

Len

=

um
.
comprimento
, Assim,

eu

=

0
;

se

(
Len

===

Política da mesma origem
, Assim,
A política da mesma origem
Gráficos vetoriais escaláveis nn(SVG)
, Assim,
SVG: gráficos vetoriais escaláveis
-
Criando imagens SVG com JavaScript
Criando imagens SVG com JavaScript
, Assim,
Criando imagens SVG com
JavaScript
Visão geral de
, Assim,
SVG: gráficos vetoriais escaláveis
script svg
, Assim,
Script svg
SVG em html
, Assim,
SVG em html
API de rastreação
, Assim,
APIs de dispositivo móvel
Role compensações
, Assim,
Documentar coordenadas e coordenadas de viewport
rolando
, Assim,
Rolando
Método ScrollTo ()
, Assim,
Rolando
Método de pesquisa ()
, Assim,
Métodos de string para correspondência de padrões
segurança
armazenamento do lado do cliente
, Assim,
Armazenar
Objetivos concorrentes da programação da web
, Assim,
O modelo de segurança da web
Compartilhamento de Recursos Cross-Origin (CORS)
, Assim,
A política da mesma origem
, Assim,
Solicitações de origem cruzada
Scripts de sites cruzados (XSS)
, Assim,
Script de câmara cruzada
APIs de criptografia
, Assim,
Criptografia e APIs relacionadas
defesa contra o código malicioso
, Assim,
O que JavaScript não pode fazer
ataques de negação de serviço
, Assim,

O DOM foi introduzido em
§15.1.2
.Esta seção explica a API em
detalhe.Isto
capas:
Como consultar ou
Selecione
elementos individuais de um documento.
Como fazer
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
Lado do cliente
Os programas JavaScript geralmente precisam manipular um ou mais
elementos dentro do documento.O global
documento
Propriedade 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
Folhas de estilo CSS
ter uma sintaxe muito poderosa, conhecida como
Seletores
, para
descrevendo elementos ou conjuntos de elementos em um documento.O
Dom
Métodos
QuerySelector ()
e
QuerySelectorAll ()
permitir
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.

Formulários de método
Função setTimeout ()
, Assim,
Timers
, Assim,
Timers
Método setTransform ()
, Assim,
Transformações do sistema de coordenadas
Shadow Dom
, Assim,
Shadow Dom
-
Shadow Dom API
sombras
, Assim,
Sombras
Operador esquerdo do turno (<<)
, Assim,
Operadores bitwise
Mudar à direita com o operador de sinal (>>)
, Assim,
Operadores bitwise
Mudar à direita com o operador de preenchimento zero (>>>)
, Assim,
Operadores bitwise
Método Shift ()
, Assim,
Pilhas e filas com push (), pop (), shift () e
NIFT ()
Métodos abreviados
, Assim,
Métodos abreviados
, Assim,
Getters, setters e outros
Formulários de método
Efeitos colaterais
, Assim,
Efeitos colaterais do operador
Citações únicas (')
, Assim,
Literais de cordas
Método Slice ()
, Assim,
fatiar()
Alguns () método
, Assim,
cada () e alguns ()
Ordem de classificação
, Assim,
Comparando strings
Método Sort ()
, Assim,
Invocação condicional
, Assim,
organizar()
matrizes esparsas
, Assim,
Matrizes

Suportes quadrados ([])
, Assim,
Um passeio de JavaScript
, Assim,
Trabalhando com cordas
, Assim,
Inicializadores de objetos e matrizes
, Assim,
Consulta e configuração de propriedades
, Assim,
Leitura
e escrever elementos de matriz
, Assim,
Cordas como matrizes
Biblioteca padrão
(
ver
Biblioteca padrão JavaScript)
Blocos de declaração
, Assim,
Declarações compostas e vazias
declarações
(
ver
também declarações)
declarações compostas e vazias
, Assim,
Composto e vazio
Declarações
Declarações condicionais
, Assim,
Declarações
, Assim,
Condicionais
-
trocar
Estruturas de controle
, Assim,
Um passeio de JavaScript
-
Um passeio de JavaScript
, Assim,
Declarações
declarações de expressão
, Assim,
Declarações de expressão
versus expressões
, Assim,
Um passeio de JavaScript
Declaração se/else
, Assim,
Valores booleanos
Jump declarações
, Assim,
Declarações
, Assim,
Saltos
-
tente/capturar/finalmente

separando -se com semicolons
, Assim,
Semicolons opcionais
-
Opcional
Semicolons
lançar declarações
, Assim,
lançar
Tente/Catch/Finalmente declarações
, Assim,
tente/capturar/finalmente
-
tente/capturar/finalmente
declarações de rendimento
, Assim,
colheita
, Assim,
O valor de uma expressão de rendimento
Campos estáticos
, Assim,
Campos públicos, privados e estáticos
Métodos estáticos
, Assim,
Métodos estáticos
armazenar
, Assim,
Armazenar
-
Indexeddb
biscoitos
, Assim,
Biscoitos
Indexeddb
, Assim,
Indexeddb
LocalStorage e SessionStorage
, Assim,
LocalStorage e SessionStorage
Visão geral de
, Assim,
Armazenar
segurança e privacidade
, Assim,
Armazenar
fluxos (nó)
, Assim,
Fluxos
-
Modo pausado
iteração assíncrona em
, Assim,
Iteração assíncrona
Visão geral de
, Assim,
Fluxos
tubos
, Assim,
Tubos

Visão geral de
, Assim,
Operadores de igualdade e desigualdade
comparação de string
, Assim,
Trabalhando com cordas
Digite conversões
, Assim,
Visão geral e definições
, Assim,
Conversões e
Igualdade
modo rigoroso
Aplicação padrão de
, Assim,
Aulas com a palavra -chave da classe
, Assim,
Módulos
em ES6
, Assim,
Módulos JavaScript na web
Excluir operador e
, Assim,
O operador de exclusão
Excluindo propriedades
, Assim,
Excluindo propriedades
EVAL () função
, Assim,
EVAL
declarações de função
, Assim,
Declarações de função
Invocação de funções
, Assim,
Invocação de funções
versus modo não rigoroso
, Assim,
“Use rigoroso”
-
“Use rigoroso”
optar em
, Assim,
Introdução ao JavaScript
TypeError
, Assim,
Erros de acesso à propriedade
, Assim,
Extensibilidade do objeto
variáveis ■■não declaradas e
, Assim,
Declarações variáveis ■■com VAR
com declaração e
, Assim,
com
, Assim,
Definindo atributos de manipulador de eventos
Literais de cordas
Sequências de fuga em

matriz para conversões de string
, Assim,
Matriz para conversões de string
caracteres e pontos de código
, Assim,
Texto
Métodos para correspondência de padrões
corresponder()
, Assim,
corresponder()
Matchall ()
, Assim,
Matchall ()
substituir()
, Assim,
substituir()
procurar()
, Assim,
Métodos de string para correspondência de padrões
dividir()
, Assim,
dividir()
Visão geral de
, Assim,
Texto
Literais de cordas
, Assim,
Literais de cordas
cordas como matrizes
, Assim,
Cordas como matrizes
trabalhando com
acessando caracteres individuais
, Assim,
Trabalhando com cordas
API para
, Assim,
Trabalhando com cordas
comparando
, Assim,
Trabalhando com cordas
, Assim,
Comparando strings
concatenação
, Assim,
Trabalhando com cordas
Determinação de comprimento
, Assim,
Trabalhando com cordas
imutabilidade
, Assim,
Trabalhando com cordas
Algoritmo de clone estruturado
, Assim,
Gerenciamento de história com pushState ()
subarrays
, Assim,
Subarrays with slice (), splice (), preench () e copywithin ()
subclasses

Classes abstratas
-
Resumo
delegação versus herança
, Assim,
Delegação em vez de herança
Visão geral de
, Assim,
Subclasses
protótipos e
, Assim,
Subclasses e protótipos
com cláusula estendida
, Assim,
Subclasses com extensões e super
-
Subclasses com extensões e super
sub -rotinas
, Assim,
Funções
Operador de subtração (-)
, Assim,
Aritmética em javascript
pares substitutos
, Assim,
Texto
Svg
(
ver
Gráficos vetoriais escaláveis nn(SVG))
Switch Declarações
, Assim,
trocar
-
trocar
Symbol.asynciterator
, Assim,
Symbol.iterator e Symbol.asynciterator
Símbolo.hasinstance
, Assim,
Símbolo.hasinstance
Symbol.iscoNcatsPreadable
, Assim,
Symbol.iscoNcatsPreadable
Symbol.iterator
, Assim,
Símbolos bem conhecidos
Symbol.Spécies
, Assim,
Symbol.Spécies
-
Symbol.Spécies
Symbol.ToPrimitive
, Assim,
Symbol.ToPrimitive
Symbol.ToStringTag
, Assim,
Symbol.ToStringTag
Symbol.unscopables

iteradores e geradores
(
ver
também métodos de iterador de matriz)
Recursos avançados do gerador
Valor de retorno das funções do gerador
, Assim,
O valor de retorno de um
Função do gerador
Métodos de retorno () e Throw ()
, Assim,
O retorno () e o arremesso ()
Métodos de um gerador
valor das expressões de rendimento
, Assim,
O valor de uma expressão de rendimento
assíncrono
, Assim,
Iteradores assíncronos
-
Implementação
Iteradores assíncronos
fechando iteradores
, Assim,
“Fechando” um iterador: o método de retorno
geradores
benefícios de
, Assim,
Uma nota final sobre geradores
criando
, Assim,
Geradores
Definição de termo
, Assim,
Geradores
exemplos de
, Assim,
Exemplos de gerador
rendimento* e geradores recursivos
, Assim,
rendimento* e recursivo
Geradores
como os iteradores funcionam
, Assim,
Como os iteradores funcionam
implementando objetos iteráveis
, Assim,
Implementando objetos iteráveis
-
Implementando objetos iteráveis
Visão geral de
, Assim,
Iteradores e geradores
J
JavaScript

espaços
, Assim,
O texto de um programa JavaScript
Conjunto de caracteres unicode
, Assim,
Sequências de Escape Unicode
-
Unicode
Normalização
operadores lógicos
, Assim,
Um passeio de JavaScript
Métodos
, Assim,
Um passeio de JavaScript
objetos
Acessando propriedades condicionalmente
, Assim,
Um passeio de JavaScript
declarando
, Assim,
Um passeio de JavaScript
Métodos abreviados
, Assim,
Métodos abreviados
declarações
, Assim,
Um passeio de JavaScript
variáveis, atribuindo valores a
, Assim,
Um passeio de JavaScript
T
guias
, Assim,
O texto de um programa JavaScript
Literais de modelo marcados
, Assim,
Literais de modelo marcados
, Assim,
Tags de modelos
Literais de modelo
, Assim,
Literais de modelo
, Assim,
Tags de modelos
operadores ternários
, Assim,
Número de operandos
teste () método
, Assim,
teste()
texto
desenhando tela
, Assim,
Texto
Sequências de fuga em literais de cordas
, Assim,
Sequências de fuga em string
Literais

Literais de cordas
, Assim,
Literais de cordas
Tipo de string representando
, Assim,
Texto
Literais de modelo
, Assim,
Literais de modelo
Trabalhando com cordas
, Assim,
Trabalhando com cordas
Editores de texto
normalização
, Assim,
Normalização unicode
usando com o nó
, Assim,
Olá mundo
Estilos de texto
, Assim,
Estilos de texto
.then () método
, Assim,
Usando promessas
, Assim,
Encadear promessas
, Assim,
Mais sobre promessas
e erros
essa palavra -chave
, Assim,
Um passeio de JavaScript
, Assim,
Expressões primárias
, Assim,
Função
Invocação
rosqueamento
, Assim,
Tópicos de trabalhadores e mensagens
, Assim,
Aplicativos da Web progressivos e
Trabalhadores de serviço
(
ver
também API do trabalhador)
Gráficos 3D
, Assim,
Gráficos em uma <VAS>
lançar declarações
, Assim,
lançar
, Assim,
Classes de erro
Método de Throw ()
, Assim,
Os métodos de retorno () e arremesso () de um gerador
Fusos horários

Método ToisSotring ()
, Assim,
Strings de data de formatação e análise
, Assim,
JSON
Personalizações
Método Tojson ()
, Assim,
O método Tojson ()
, Assim,
Customizações JSON
Método tolocaledatestring ()
, Assim,
Strings de data de formatação e análise
, Assim,
Datas e tempos de formatação
Método tolocalestring ()
, Assim,
O método tolocalestring ()
, Assim,
Array para string
Conversões
, Assim,
Strings de data de formatação e análise
Método tolocaletimestring ()
, Assim,
Strings de data de formatação e análise
, Assim,
Datas e tempos de formatação
ferramentas e extensões
, Assim,
Ferramentas e extensões JavaScript
-
Enumerado
Tipos e sindicatos discriminados
Bundling de código
, Assim,
Bundling de código
JavaScript Formating com mais bonito
, Assim,
JavaScript Formating com
Mais bonito
Extensão da linguagem JSX
, Assim,
JSX: Expressões de marcação em JavaScript
-
JSX: Expressões de marcação em JavaScript
LING COM ESLINT
, Assim,
LING COM ESLINT
Visão geral de
, Assim,
Ferramentas e extensões JavaScript
Gerenciamento de pacotes com NPM
, Assim,
Gerenciamento de pacotes com NPM
transpilação com Babel
, Assim,
Transpilação com Babel

erros.)
O
escrever()
O método tem um valor de retorno muito importante.Quando você
chamar
escrever()
Em um riacho, ele sempre aceitará e amortece o pedaço
de dados que você passou.Então retorna
verdadeiro
Se o buffer interno for
ainda não está cheio.Ou, se o buffer agora estiver cheio ou muito cheio, ele retorna
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
falso
do
escrever()
Método 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 ligar
escrever()
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

escrever
(
fluxo
, Assim,

pedaço
, Assim,

ligar de volta
)

{

// Escreva o pedaço especificado para o fluxo especificado

deixar

HasMoreroom

=

fluxo
.
escrever

translucidez
, Assim,
Translucidez e composição
transpilação
, Assim,
Transpilação com Babel
valores verdadeiros
, Assim,
Valores booleanos
Tente/Catch/Finalmente declarações
, Assim,
tente/capturar/finalmente
-
tente/capturar/finalmente
Verificação de tipo
, Assim,
Verificação de tipo com fluxo
-
Tipos enumerados e
Sindicatos discriminados
Tipos de matriz
, Assim,
Tipos de matriz
Tipos de classe
, Assim,
Tipos de classe
tipos enumerados e sindicatos discriminados
, Assim,
Tipos enumerados
e sindicatos discriminados
Tipos de funções
, Assim,
Tipos de funções
Instalando e executando o fluxo
, Assim,
Instalando e executando o fluxo
tipos de objetos
, Assim,
Tipos de objetos
Outros tipos parametrizados
, Assim,
Outros tipos parametrizados
Visão geral de
, Assim,
Verificação de tipo com fluxo
Tipos somente leitura
, Assim,
Tipos somente leitura
Tipo Aliases
, Assim,
Tipo Aliases
TypeScript versus Flow
, Assim,
Verificação de tipo com fluxo
Tipos de sindicatos
, Assim,
Tipos de sindicatos
usando anotações de tipo
, Assim,

conversões explícitas
, Assim,
Conversões explícitas
dados financeiros e científicos
, Assim,
Conversões explícitas
conversões implícitas
, Assim,
Conversões explícitas
Objeta -se a conversões primitivas
algoritmos para
, Assim,
Objeta -se a conversões primitivas
, Assim,
Objeto para
Algoritmos de conversão primitiva
objeto a booleano
, Assim,
Conversões objeto para boolean
Objeto em número
, Assim,
Conversões de objeto para número
objeto para cordas
, Assim,
Conversões de objeto para corda
Conversões especiais de operadoras de casos
, Assim,
Operador de caso especial
conversões
Métodos ToString () e ValueOf ()
, Assim,
O tostring () e
Métodos Valueof ()
Visão geral de
, Assim,
Digite conversões
matrizes digitadas
criando
, Assim,
Criando matrizes digitadas
DataView e Endianness
, Assim,
DataView e Endianness
Métodos e propriedades
, Assim,
Métodos e propriedades da matriz digitada
Visão geral de
, Assim,
Matrizes digitadas e dados binários
versus matrizes regulares
, Assim,
Matrizes
compartilhando entre threads
, Assim,
Compartilhando matrizes digitadas entre threads
Tipos de matriz digitados
, Assim,
Tipos de matriz digitados
usando

NPM Install -g Flow -Bin
ou
NPM Instale-Save-dev
Fluxo-barracão
.Se você instalar a ferramenta globalmente com
-g
, então você pode executá -lo
com
fluxo
.E se você o instalar localmente em seu projeto com
--salvar-
Dev
, então você pode executá -lo com
Fluxo NPX
.Antes de usar o fluxo para fazer
Verificação do tipo, a primeira vez que o executa como
fluxo -init
na raiz
diretório do seu projeto para criar um
.FlowConfig
arquivo de configuração.
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
// @fluxo
Comente no topo 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
// @fluxo
comentário.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].

5.4.2 Do/while
O

faça/while
Loop é como um
enquanto
loop, 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
faça/while
loop é menos comumente usado do que o seu
enquanto
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
faça/while
laço:
função

PrintArray
(
um
)

{

deixar

Len

=

um
.
comprimento
, Assim,

eu

=

0
;

se

(
Len

===

Invocação de funções
, Assim,
Invocação de funções
Optar no modo rigoroso
, Assim,
Introdução ao JavaScript
modo rigoroso versus não rigoroso
, Assim,
“Use rigoroso”
-
“Use rigoroso”
TypeError
, Assim,
Erros de acesso à propriedade
, Assim,
Extensibilidade do objeto
variáveis ■■não declaradas e
, Assim,
Declarações variáveis ■■com VAR
com declaração e
, Assim,
com
, Assim,
Definindo atributos de manipulador de eventos
Use modo rigoroso
, Assim,
Introdução ao JavaScript
e variáveis ■■globais
, Assim,
Declarações variáveis ■■com VAR
Excluindo propriedades
, Assim,
Excluindo propriedades
Codificação UTF-16
, Assim,
Texto
V
Método Valueof ()
, Assim,
Os métodos ToString () e ValueOf ()
, Assim,
O
Método Valueof ()
valores
atribuição
, Assim,
Um passeio de JavaScript
valores booleanos
, Assim,
Valores booleanos
-
Valores booleanos
falsidade e verdade
, Assim,
Valores booleanos
funciona como valores
, Assim,
Funciona como valores
-

Definições
tipos de
, Assim,
Um passeio de JavaScript
palavra -chave var
, Assim,
Declarações variáveis ■■com VAR
, Assim,
const, let e var
varargs
, Assim,
Parâmetros de descanso e listas de argumentos de comprimento variável
funções variáveis ■■de arity
, Assim,
Parâmetros de descanso e comprimento de variável
Listas de argumentos
variáveis
Sensibilidade ao caso
, Assim,
O texto de um programa JavaScript
declaração e atribuição
declarações com Let and Const
, Assim,
Declarações com let e
const
-
Declarações e tipos
declarações com var
, Assim,
Declarações variáveis ■■com VAR
atribuição de destruição
, Assim,
Atribuição de destruição
-
Atribuição de destruição
Visão geral de
, Assim,
Um passeio de JavaScript
variáveis ■■não declaradas
, Assim,
Declarações variáveis ■■com VAR
Definição de termo
, Assim,
Declaração e atribuição variáveis
içado
, Assim,
Declarações variáveis ■■com VAR
nomeação
, Assim,
Palavras reservadas
Visão geral de
, Assim,
Tipos, valores e variáveis
-
Visão geral e
Definições
escopo de
, Assim,
Escopo variável e constante

fluxos de vídeo
, Assim,
APIs de mídia
viewport
, Assim,
Documentar coordenadas e coordenadas de viewport
, Assim,
Viewport
Tamanho, tamanho do conteúdo e posição de rolagem
operador vazio
, Assim,
O operador vazio
C
Classe de mapa fraco
, Assim,
Map fraco e conjunto fraco
Classe de conjunto fraco
, Assim,
Map fraco e conjunto fraco
API de autenticação da Web
, Assim,
Criptografia e APIs relacionadas
ambiente de host de navegador da web
APIs assíncronas
, Assim,
Eventos
APIs de áudio
, Assim,
APIs de áudio
-
A API de Webudio
Benefícios do JavaScript
, Assim,
JavaScript em navegadores da web
API de tela
, Assim,
Gráficos em uma <VAS>
-
Manipulação de pixels
Dimensões e coordenadas de tela
, Assim,
Dimensões de tela e
Coordenadas
recorte
, Assim,
Recorte
Transformações do sistema de coordenadas
, Assim,
Sistema de coordenadas
Transforma
-
Exemplo de transformação
Operações de desenho
, Assim,
Operações de desenho de lona
atributos gráficos
, Assim,
Atributos gráficos
, Assim,

manipulação de pixels
, Assim,
Manipulação de pixels
Documentar geometria e rolagem
, Assim,
Documentar geometria e
Rolando
-
Tamanho da viewport, tamanho de conteúdo e posição de rolagem
CSS Pixels
, Assim,
Documentar coordenadas e viewport
Coordenadas
Determinando elemento em um ponto
, Assim,
Determinando o elemento em um
Apontar
Documentar coordenadas e coordenadas de viewport
, Assim,
Documento
Coordena e coordenadas de viewport
GEOMETRIA DE CONSUMENTO DE ELEMENTOS
, Assim,
Consultar a geometria de
um elemento
rolando
, Assim,
Rolando
Tamanho da viewport, tamanho de conteúdo e posição de rolagem
, Assim,
Viewport
Tamanho, tamanho do conteúdo e posição de rolagem
eventos
, Assim,
Eventos
-
Despacha eventos personalizados
Despacha eventos personalizados
, Assim,
Despacha eventos personalizados
Cancelamento de eventos
, Assim,
Cancelamento de eventos
categorias de eventos
, Assim,
Categorias de eventos
Invocação do manipulador de eventos
, Assim,
Invocação do manipulador de eventos
propagação de eventos
, Assim,
Propagação de eventos
Visão geral de
, Assim,
Eventos
Registrando manipuladores de eventos
, Assim,
Registrando manipuladores de eventos
APIs legadas

História
-
Gerenciamento de história com pushState ()
História de navegação
, Assim,
História de navegação
Carregando novos documentos
, Assim,
Carregando novos documentos
Visão geral de
, Assim,
Localização, navegação e história
Exemplo de conjunto de Mandelbrot
, Assim,
Exemplo: o conjunto Mandelbrot
-
Resumo
e sugestões para leitura adicional
navegadores com reconhecimento de módulo
, Assim,
Módulos JavaScript na web
networking
, Assim,
Networking
-
Negociação de protocolo
Método Fetch ()
, Assim,
buscar()
Visão geral de
, Assim,
Networking
eventos enviados ao servidor
, Assim,
Eventos enviados ao servidor
WebSocket API
, Assim,
WebSockets
Visão geral de
, Assim,
JavaScript em navegadores da web
Gráficos vetoriais escaláveis ■■(SVG)
, Assim,
SVG: gráficos vetoriais escaláveis
-
Criando imagens SVG com JavaScript
Criando imagens SVG com JavaScript
, Assim,
Criando imagens SVG
com javascript
Visão geral de
, Assim,
SVG: gráficos vetoriais escaláveis
script svg
, Assim,
Script svg
SVG em html
, Assim,
SVG em html

Animações e eventos CSS
, Assim,
Animações e eventos CSS
Classes CSS
, Assim,
Classes CSS
Estilos embutidos
, Assim,
Estilos embutidos
Convenções de nomeação
, Assim,
Estilos embutidos
folhas de estilo de script
, Assim,
Folhas de estilo de script
documentos de script
, Assim,
Documentos de script
-
Exemplo: gerando a
Índice
estrutura de documentos e travessia
, Assim,
Estrutura de documentos e
Traversal
geração dinamicamente tabelas de conteúdo
, Assim,
Exemplo:
Gerando uma tabela de índice
Modificando conteúdo
, Assim,
Conteúdo do elemento como html
estrutura modificadora
, Assim,
Criando, inserindo e excluindo nós
Visão geral de
, Assim,
Documentos de script
consulta e definição de atributos
, Assim,
Atributos
selecionando elementos do documento
, Assim,
Selecionando elementos do documento
armazenar
, Assim,
Armazenar
-
Indexeddb
biscoitos
, Assim,
Biscoitos
Indexeddb
, Assim,
Indexeddb
LocalStorage e SessionStorage
, Assim,
LocalStorage e
SessionStorage

Componentes da Web
, Assim,
Componentes da Web
-
Exemplo: a <search-box>
Componente da Web
elementos personalizados
, Assim,
Elementos personalizados
Nós de documentário
, Assim,
Usando componentes da web
Modelos HTML
, Assim,
Modelos HTML
Visão geral de
, Assim,
Componentes da Web
Exemplo da caixa de pesquisa
, Assim,
Exemplo: uma web <search-box>
Componente
Shadow Dom
, Assim,
Shadow Dom
usando
, Assim,
Usando componentes da web
Recursos da plataforma da web para investigar
APIs binárias
, Assim,
APIs binárias
APIs de criptografia e segurança
, Assim,
Criptografia e relacionado
APIs
eventos
, Assim,
Eventos
HTML e CSS
, Assim,
HTML e CSS
APIs de mídia
, Assim,
APIs de mídia
APIs de dispositivo móvel
, Assim,
APIs de dispositivo móvel
APIs de desempenho
, Assim,
Desempenho
Aplicativos da Web progressivos e serviços de serviço
, Assim,
Progressivo
Aplicativos da Web e trabalhadores de serviço
segurança
, Assim,
Segurança
WebAssembly

WebSocket API
Criando, conectando e desconectando WebSockets
, Assim,
Criando,
Conectando e desconectando WebSockets
Visão geral de
, Assim,
WebSockets
Negociação de protocolo
, Assim,
Negociação de protocolo
recebendo mensagens
, Assim,
Recebendo mensagens de uma webSocket
enviando mensagens
, Assim,
Enviando mensagens sobre um webSocket
enquanto loops
, Assim,
enquanto
com declarações
, Assim,
Declarações diversas
API do trabalhador
Mensagens de origem cruzada
, Assim,
Mensagens de origem cruzada com
PostMessage ()
erros
, Assim,
Erros em trabalhadores
Modelo de execução
, Assim,
Modelo de execução do trabalhador
Código de importação
, Assim,
Importar código para um trabalhador
Exemplo de conjunto de Mandelbrot
, Assim,
Exemplo: o conjunto Mandelbrot
-
Resumo
e sugestões para leitura adicional
módulos
, Assim,
Importar código para um trabalhador
Visão geral de
, Assim,
Tópicos de trabalhadores e mensagens
PostMessage (), MessagePorts e Messagechannels
, Assim,
PostMessage (), MessagePorts e Messagechannels
Objetos de trabalhador
, Assim,
Objetos de trabalhador
Objeto Workerglobalscope
, Assim,
O objeto global em trabalhadores

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 um
janela.
prefixo:
Window.innerWidth
é mais claro do que
INNERWIDTH
, por exemplo.

15.1.4 Os scripts compartilham um espaço para nome

Com
módulos, 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 outro
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
função
declarações no nível superior criam

Sobre o autor
David Flanagan
tem programado e escrevendo sobre
JavaScript desde 1995. Ele vive com sua esposa e filhos no
Noroeste do Pacífico entre as cidades de Seattle, Washington e
Vancouver, Colúmbia Britânica.David é formado em ciência da computação
e engenharia do Instituto de Tecnologia de Massachusetts e
Funciona como engenheiro de software na VMware.

Colofão
O animal na capa de
JavaScript: O Guia Definitivo
, Sétimo
Edição, é um rinoceronte Javan (
Rinoceronte Sondaicus
).Todas as cinco espécies
de rinoceronte se distingue por seu tamanho grande, como uma armadura grossa
pele, três dedos e chifre de focinho único ou duplo.O javan
Os rinocerontes se assemelham ao rinoceronte indiano relacionado e, como com isso
Espécies, os machos têm um único chifre.No entanto, os rinocerontes Javan são
menor e têm texturas de pele únicas.Embora encontrado hoje apenas em
A Indonésia, Javan Rhinos, já variava em todo o sudeste da Ásia.
Eles vivem em habitats da floresta tropical, onde pastam em folhas abundantes
e gramíneas e esconder-se de pragas de insetos, como moscas sugando de sangue por
Levando -se aos focinhos em água ou lama.
O rinoceronte Javan tem uma média de cerca de 6 pés de altura e pode ter até 10
pés de comprimento, com adultos pesando até 3.000 libras.Como o indiano
Rinoceronte Sua pele cinza parece estar separada em "placas", alguns dos
eles texturizam.A vida útil natural de um rinoceronte Javan é estimada em 45 -
50 anos.As fêmeas dão à luz a cada 3 a 5 anos, após um período de gestação de
16 meses.Os bezerros pesam cerca de 100 libras quando nasceram e fique com
suas mães protetoras por até 2 anos.
Rinocerontes são geralmente um animal um tanto abundante, sendo adaptável
a uma variedade de habitats e na idade adulta não possui predadores naturais.
No entanto, os humanos os caçaram quase à extinção.Folclore
sustenta que o chifre dos rinocerontes possui mágico e afrodisíaco
Poderes e, por causa disso, os rinocerontes são um alvo principal para caçadores caçadores.O
A população de rinoceronte Javan é a mais precária: a partir de 2020, os 70 ou mais
Os animais restantes desta espécie vivem, sob guarda, em Ujung Kulon

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.