This page was translated from English by the community, but it's not maintained and may be out-of-date. To help maintain it, learn how to activate locales.

Array

Sumário

O objeto `Array` do JavaScript é um objeto global usado na construção de 'arrays': objetos de alto nível semelhantes a listas.

Criando um Array

```
var frutas = ['Maçã', 'Banana'];

console.log(frutas.length);
// 2
```

Acessar um item (*index*) do Array

```
var primeiro = frutas[0];
// Maçã

var ultimo = frutas[frutas.length - 1];
// Banana
```

Iterar um Array

```
frutas.forEach(function (item, indice, array) {
  console.log(item, indice);
});
// Maçã 0
// Banana 1
```

Adicionar um item ao final do Array

```
var adicionar = frutas.push('Laranja');
// ['Maçã', 'Banana', 'Laranja']
```

```
// ['Maçã', 'Banana', 'Laranja']
```

Remover um item do final do Array

```
var ultimo = frutas.pop(); // remove Laranja (do final)
// ['Maçã', 'Banana'];
```

Remover do início do Array

```
var primeiro = frutas.shift(); // remove Maçã do início
// ['Banana'];
```

Adicionar ao início do Array

```
var adicionar = frutas.unshift('Morango') // adiciona ao início
// ['Morango', 'Banana'];
```

Procurar o índice de um item na Array

```
frutas.push('Manga');
// ['Morango', 'Banana', 'Manga']

var pos = frutas.indexOf('Banana');
// 1
```

Remover um item pela posição do índice

```
var removedItem = frutas.splice(pos, 1); // é assim que se remove um i
// ['Morango', 'Manga']
```

Remover itens de uma posição de índice

```
var vegetais = ['Repolho', 'Nabo', 'Rabanete', 'Cenoura'];
console.log(vegetais);
// ['Repolho', 'Nabo', 'Rabanete', 'Cenoura']

var pos = 1, n = 2;

var itensRemovidos = vegetais.splice(pos, n);
// Isso é como se faz para remover itens, n define o número de itens a
// a partir da posição (pos) em direção ao fim da array.

console.log(vegetais);
```

```
// ['Repolho', 'Cenoura'] (o array original é alterado)

console.log(itensRemovidos);
// ['Nabo', 'Rabanete']
```

Copiar um Array

```
var copiar = frutas.slice(); // é assim que se copia
// ['Morango', 'Manga']
```

Sintaxe

```
[element0, element1, ..., elementN]
new Array(element0, element1, ..., elementN)
new Array(arrayLength)
```

element0, element1, ..., elementN

Um array JavaScript é inicializado com os elementos contém, exceto no caso onde um único argumento é passado para o construtor do `Array` e esse argumento é um número (veja o parâmetro arrayLength abaixo). Esse caso especial só se aplica para os arrays JavaScript criados com o construtor `Array` , e não para literais de array criados com a sintaxe de colchetes [].

arrayLength

Se o único argumento passado para o construtor do `Array` for um número inteiro entre 0 and 2^{32}-1 (inclusive), um novo array com o tamanho desse número é retornado. Se o argumento for qualquer outro número, uma exceção `RangeError` é lançada.

Descrição

Arrays são objetos semelhantes a listas que vêm com uma série de métodos embutidos para realizar operações de travessia e mutação. Nem o tamanho de um array JavaScript nem os tipos de elementos são fixos. Já que o tamanho de um array pode ser alterado a qualquer momento e os dados podem ser armazenados em posições não contíguas, arrays JavaScript não tem a garantia de serem densos; isso depende de como o programador escolhe usá-los. De uma maneira geral, essas são características convenientes, mas, se esses recursos não são desejáveis para o seu caso em particular, você pode considerar usar arrays tipados.

Arrays não podem usar strings como índices (como em um array associativo), devem ser usados números inteiros. Definir ou acessar não-inteiros usando notação de colchetes (ou notação de ponto) não vai definir ou recuperar um elemento do array em si, mas sim definir ou acessar uma variável associada com a coleção de propriedades de objeto daquele array. As propriedades de objeto do array e a lista de elementos do array são separados, e as operações de travessia e mutação não podem ser aplicadas a essas propriedades nomeadas.

Accessando elementos de um array

Arrays JavaScript começam com índice zero: o primeiro elemento de um array está na posição `0` e o último elemento está na posição equivalente ao valor da propriedade `length` (tamanho) menos 1.

```
var arr = ['este é o primeiro elemento', 'este é o segundo elemento']
console.log(arr[0]);              // exibe 'este é o primeiro elemento
console.log(arr[1]);              // exibe 'este é o segundo elemento
console.log(arr[arr.length - 1]); // exibe 'este é o segundo elemento
```

Elementos de um array são somente propriedades de objetos, da forma que `toString` é uma propriedade. Contudo, note que tentando acessar o primeiro elemento de um array da seguinte forma causará um erro de sintaxe, pois o nome da propriedade é inválido:

```
console.log(arr.0); // um erro de sintaxe
```

Não há nada de especial a respeito de arrays JavaScript e suas propriedades que causam isso. As propriedades JavaScript que começam com um dígito não podem ser referenciadas com notação de ponto. Elas necessitam usar notação de colchetes para poderem ser acessadas. Por exemplo, se você tivesse um objeto com a propriedade "3d", também teria que ser referenciá-la usando notação de colchetes. Por exemplo:

```
var anos = [1950, 1960, 1970, 1980, 1990, 2000, 2010];
console.log(anos.0); // um erro de sintaxe
console.log(anos[0]); // funciona corretamente
```

```
renderer.3d.setTexture(model, 'personagem.png'); // um erro de sintaxe
renderer['3d'].setTexture(model, 'personagem.png'); //funciona corretamente
```

Note que no exemplo `3d`, `'3d'` teve de ser colocado entre aspas. É possivel também colocar entre aspas os índices de arrays JavaScript (ou seja, `years['2']` ao invés

de `years[2]`), contudo isto não é necessário. O valor 2 em `years[2]` eventualmente será convertido a uma string pela engine do JavaScript através de uma conversão explicita com o método `toString`. E é por esta razão que `'2'` e `'02'` irão referenciar dois slots diferentes no objeto `anos` e o seguinte exemplo pode ser *`true`*:

```
console.log(anos['2'] != anos['02']);
```

De forma similar, propriedades de objeto que sejam palavras reservadas(!) só podem ser acessadas como strings em notação de colchetes:

```
var promessa = {
  'var': 'texto',
  'array': [1, 2, 3, 4]
};

console.log(promessa['var']);
```

Relação entre *`length`* e propriedades numéricas

As propriedades `length` e numéricas de um array Javascript são conectadas. Varios dos métodos javascript pré-definidos (por exemplo, `join`, `slice`, `indexOf` etc.) levam em conta o valor da propriedade `length` de um array quando eles são chamados. Outros métodos (por exemplo, `push`, `splice` etc.) também resultam em uma atualização na propriedade `length` do array.

```
var frutas = [];
frutas.push('banana', 'maça', 'pêssego');

console.log(frutas.length); // 3
```

Quando configurar uma propriedade num array Javascript em que a propriedade é um índice valido do array e este índice está fora do atual limite do array, o array irá crescer para um tamanho grande o suficiente para acomodar o elemento neste índice, e a engine irá atualizar a propriedade *length* do array de acordo com isto:

```
frutas[5] = 'manga';
console.log(frutas[5]); // 'manga'
console.log(Object.keys(frutas)); // ['0', '1', '2', '5']
console.log(frutas.length); // 6
```

Configurar a propriedade `length` diretamente, também resulta em um comportamento especial:

```
frutas.length = 10;
console.log(Object.keys(frutas)); // ['0', '1', '2', '5']
console.log(frutas.length); // 10
```

Diminuir o valor de `length`, entretanto, apaga elementos:

```
frutas.length = 2;
console.log(Object.keys(frutas)); // ['0', '1']
console.log(frutas.length); // 2
```

Criando um array usando o resultado de uma comparação

O resultado de uma comparação entre uma *expressão regular* e uma string pode criar um array Javascript. Este array tem propriedades e elementos que disponibilizam informações sobre a comparação. Esse array é o valor de retorno dos métodos `RegExp.exec`, `String.match`, e `String.replace`. Para explicar melhor sobre estas propriedades e elementos, veja o seguinte exemplo e então consulte a tabela abaixo:

```
// Encontra um d seguido por um ou mais b's seguido por um d
// Salva os b's encontrados e o d seguinte
// Ignora caixa (maiúscula/minúscula)

var minhaRegex = /d(b+)(d)/i;
var meuArray = minhaRegex.exec('cdbBdbsbz');
```

As propriedades e elementos retornados desta comparação são os seguintes:

Propriedade/Elemento	Descrição	Exemplo
`input`	Uma propriedade somente-leitura que reflete a string original a qual a expressão regular foi comparada.	cdbBdbsbz
`index`	Uma propriedade somente-leitura que é o índice baseado em zero da comparação na string.	1
`[0]`	Um elemento somente-leitura que especifica os ultimos caracteres que foram encontrados.	dbBd

`[1], ...[n]`	Elementos somente-leitura que especificam as *substrings* de comparações entre parênteses encontradas, se incluidas na expressão regular. O número de possíveis *substrings* entre parenteses é ilimitado.	[1]: bB [2]: d

Propriedades

`Array.length`

Propriedade comprimento do construtor `Array` , cujo valor é 1.

`get Array[@@species]` (en-US)

A função de construtor que é utilizada para criar objetos derivados.

`Array.prototype`

Permite a adição de propriedades para todos os objetos array.

Métodos

`Array.from()`

Cria uma nova instância de `Array` a partir de um objeto semelhante ou iterável.

`Array.isArray()`

Retorna `true` se a variável é um array e `false` caso contrário.

`Array.of()`

Cria uma nova instância de `Array` com um número variável de argumentos, independentemente do número ou tipo dos argumentos.

Instâncias de `Array`

Todas as instâncias de `Array` herdam de `Array.prototype` . O protótipo do construtor `Array` pode ser modificado de forma a afetar todas as instâncias de `Array` .

Propriedades

{{ page('/pt-BR/docs/JavaScript/Reference/Global_Objects/Array/prototype', 'Properties') }}

Métodos

Métodos modificadores

{{ page('/pt-BR/docs/JavaScript/Reference/Global_Objects/Array/prototype', 'Mutator_methods') }}

Métodos de acesso

{{ page('/pt-BR/docs/JavaScript/Reference/Global_Objects/Array/prototype', 'Accessor_methods') }}

Métodos de iteração

{{ page('/pt-BR/docs/JavaScript/Reference/Global_Objects/Array/prototype', 'Iteration_methods') }}

Métodos genéricos de `Array`

> **Métodos genéricos de arrays não seguem o padrão, são obsoletos e serão removidos em breve.**

Algumas vezes você poderá querer aplicar métodos de arrays para strings ou outros objetos parecidos com arrays (como em `argumentos` de funções). Ao fazer isto, você trata uma string como um array de caracteres (ou em outros casos onde trata-se não-arrays como um array). Por exemplo, para checar se cada caractere em uma varivável *str* é uma letra, você poderia escrever:

```
function isLetter(character) {
  return (character >= "a" && character <= "z");
}

if (Array.prototype.every.call(str, isLetter))
  alert("A string '" + str + "' contém somente letras!");
```

Esta notação é um pouco despendiosa e o JavaScript 1.6 introduziu a seguinte abreviação genérica:

```
if (Array.every(isLetter, str))
  alert("A string '" + str + "' contém somente letras!");
```

Generics também estão disponíveis em `String`.

Estes não são atualmente parte dos padrões ECMAScript (através do ES2015 `Array.from()` pode se conseguir isto). A seguir segue uma adaptação para permitir o uso em todos os navegadores:

```
/*globals define*/
// Assumes Array extras already present (one may use shims for these a
(function () {
    'use strict';

    var i,
        // We could also build the array of methods with the following
        //   getOwnPropertyNames() method is non-shimable:
        // Object.getOwnPropertyNames(Array).filter(function (methodNa
        methods = [
            'join', 'reverse', 'sort', 'push', 'pop', 'shift', 'unshi
            'splice', 'concat', 'slice', 'indexOf', 'lastIndexOf',
            'forEach', 'map', 'reduce', 'reduceRight', 'filter',
            'some', 'every', 'isArray'
        ],
        methodCount = methods.length,
        assignArrayGeneric = function (methodName) {
            var method = Array.prototype[methodName];
            Array[methodName] = function (arg1) {
                return method.apply(arg1, Array.prototype.slice.call(a
            };
        };

    for (i = 0; i < methodCount; i++) {
        assignArrayGeneric(methods[i]);
    }
}());
```

Exemplos

Exemplo: Criando um array

O exemplo a seguir cria um array, `msgArray` , com *length* 0, então atribui valores para `msgArray[0]` e `msgArray[99]` , trocando o *length* do array para 100.

```
var msgArray = new Array();
msgArray[0] = "Hello";
msgArray[99] = "world";

if (msgArray.length == 100)
   print("O length é 100.");
```

Exemplo: Criando um array bi-dimensional

O exemplo a seguir cria um tabuleiro de xadrez usando dois arrays bi-dimensionais de string. A primeira jogada é feita copiando o 'p' em 6,4 para 4,4. A posição antiga de 6,4 é colocada em branco.

```
var board =
[ ['R','N','B','Q','K','B','N','R'],
  ['P','P','P','P','P','P','P','P'],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  ['p','p','p','p','p','p','p','p'],
  ['r','n','b','q','k','b','n','r']];
print(board.join('\n') + '\n\n');

// Fazendo o King's Pawn avançar 2
board[4][4] = board[6][4];
board[6][4] = ' ';
print(board.join('\n'));
```

Saída:

```
R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
p,p,p,p,p,p,p,p
r,n,b,q,k,b,n,r

R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , ,p, , ,
 , , , , , , ,
p,p,p,p, ,p,p,p
r,n,b,q,k,b,n,r
```

Utilizando um array para tabular um conjunto de valores

```
values = [];
for (var x = 0; x < 10; x++){
```

```
 values.push([
  2 ** x,
  2 * x ** 2
 ])
};
console.table(values)
```

Saída:

```
0       1       0
1       2       2
2       4       8
3       8       18
4       16      32
5       32      50
6       64      72
7       128     98
8       256     128
9       512     162
```

Especificações

Especificação	Status	Comentário
ECMAScript 1st Edition (ECMA-262)	Padrão	Definição inicial
ECMAScript 5.1 (ECMA-262) The definition of 'Array' in that specification.	Padrão	Novos metodos adicionados: `Array.isArray`, `indexOf`, `lastIndexOf`, `every`, `some`, `forEach`, `map`, `filter`, `reduce`, `reduceRight`
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array' in that specification.	Padrão	Novos metodos adicionados: `Array.from`, `Array.of`, `find`, `findIndex`, `fill`, `copyWithin`
ECMAScript (ECMA-262)	Padrão em	

The definition of 'Array' in that specification.	tempo real	Novo metodo adicionado: `Array.prototype.includes()`

Compatibilidade com os navegadores

> **Estamos convertendo nossos dados de compatibilidade para o formato JSON** .
> Esta tabela de compatibilidade ainda usa o formato antigo, pois ainda não convertemos os dados que ela contém. **Descubra como você pode ajudar!**

- Desktop
- Dispositivo móvel

Configuração	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Suporte básico	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

Configuração	Android	Firefox Mobile (Gecko)	IE Phone	Opera Mobile	Safari Mobile
Suporte básico	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

Ver também

- "Indexing object properties" in JavaScript Guide: "Working with objects"
- New in JavaScript 1.7: Array comprehensions
- New in JavaScript 1.6: Array extras
- Draft: Typed Arrays

Last modified: 24 de mai. de 2021, by MDN contributors

Last modified: 24 de mar. de 2021, by MDN contributors

Change your language

Português (do Brasil) Change language