

JavaScript权威指南 (影印版)

第六版
下册



JavaScript

The Definitive Guide

O'REILLY®

东南大学出版社

David Flanagan 著

(第6版)

JavaScript权威指南 (影印版)

JavaScript: The Definitive Guide

下册



David Flanagan

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目 (CIP) 数据

JavaScript 权威指南: 第 6 版: 英文 / (美) 弗拉纳根 (Flanagan, D.) 著. —影印本. —南京: 东南大学出版社, 2011.10

书名原文: JavaScript: The Definitive Guide, Sixth Edition
ISBN 978-7-5641-2940-8

I. ① J… II. ① 弗… III. ① JAVA 语言—程序设计—英文
IV. ① TP312

中国版本图书馆 CIP 数据核字 (2011) 第 173868 号

江苏省版权局著作权合同登记

图字: 10-2010-447 号

©2011 by O'Reilly Media, Inc.

Reprint of the English Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2011. Authorized reprint of the original English edition, 2011 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2011。

英文影印版由东南大学出版社出版 2011。此影印版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式重制。

JavaScript 权威指南 第六版

出版发行: 东南大学出版社

地 址: 南京四牌楼 2 号 邮编: 210096

出 版 人: 江建中

网 址: <http://www.seupress.com>

电子邮件: press@seupress.com

印 刷: 扬中市印刷有限公司

开 本: 787 毫米 × 960 毫米 16 开本

印 张: 69.5

字 数: 1361 千字

版 次: 2011 年 10 月第 1 版

印 次: 2011 年 10 月第 1 次印刷

书 号: ISBN 978-7-5641-2940-8

定 价: 128.00 元 (上下册)

本社图书若有印装质量问题, 请直接与读者服务部联系。电话 (传真): 025-83792328

*This book is dedicated to all who teach peace
and resist violence.*

Core JavaScript Reference

This part of the book is a reference that documents the classes, methods, and properties defined by the core JavaScript language. This reference is arranged alphabetically by class or object name:

Arguments	EvalError	Number	String
Array	Function	Object	SyntaxError
Boolean	Global	RangeError	TypeError
Date	JSON	ReferenceError	URIError
Error	Math	RegExp	

The reference pages for the methods and properties of classes are alphabetized by their full names, which include the names of the classes that define them. For example, if you want to read about the `replace()` method of the `String` class, you would look under `String.replace()`, not just `replace`.

Core JavaScript defines some global functions and properties, such as `eval()` and `NaN`. Technically, these are properties of the global object. Since the global object has no name, however, they are listed in this reference section under their own unqualified names. For convenience, the full set of global functions and properties in core JavaScript is summarized in a special reference page named “Global” (even though there is no object or class by that name).

Table of Contents

Preface	xiii
----------------------	-------------

1. Introduction to JavaScript	1
1.1 Core JavaScript	4
1.2 Client-Side JavaScript	8

Part I. Core JavaScript

2. Lexical Structure	21
2.1 Character Set	21
2.2 Comments	23
2.3 Literals	23
2.4 Identifiers and Reserved Words	23
2.5 Optional Semicolons	25

3. Types, Values, and Variables	29
3.1 Numbers	31
3.2 Text	36
3.3 Boolean Values	40
3.4 null and undefined	41
3.5 The Global Object	42
3.6 Wrapper Objects	43
3.7 Immutable Primitive Values and Mutable Object References	44
3.8 Type Conversions	45
3.9 Variable Declaration	52
3.10 Variable Scope	53

4. Expressions and Operators	57
4.1 Primary Expressions	57
4.2 Object and Array Initializers	58
4.3 Function Definition Expressions	59

4.4	Property Access Expressions	60
4.5	Invocation Expressions	61
4.6	Object Creation Expressions	61
4.7	Operator Overview	62
4.8	Arithmetic Expressions	66
4.9	Relational Expressions	71
4.10	Logical Expressions	75
4.11	Assignment Expressions	77
4.12	Evaluation Expressions	79
4.13	Miscellaneous Operators	82
5.	Statements	87
5.1	Expression Statements	88
5.2	Compound and Empty Statements	88
5.3	Declaration Statements	89
5.4	Conditionals	92
5.5	Loops	97
5.6	Jumps	102
5.7	Miscellaneous Statements	108
5.8	Summary of JavaScript Statements	112
6.	Objects	115
6.1	Creating Objects	116
6.2	Querying and Setting Properties	120
6.3	Deleting Properties	124
6.4	Testing Properties	125
6.5	Enumerating Properties	126
6.6	Property Getters and Setters	128
6.7	Property Attributes	131
6.8	Object Attributes	135
6.9	Serializing Objects	138
6.10	Object Methods	138
7.	Arrays	141
7.1	Creating Arrays	141
7.2	Reading and Writing Array Elements	142
7.3	Sparse Arrays	144
7.4	Array Length	144
7.5	Adding and Deleting Array Elements	145
7.6	Iterating Arrays	146
7.7	Multidimensional Arrays	148
7.8	Array Methods	148
7.9	ECMAScript 5 Array Methods	153
7.10	Array Type	157

7.11	Array-Like Objects	158
7.12	Strings As Arrays	160
8.	Functions	163
8.1	Defining Functions	164
8.2	Invoking Functions	166
8.3	Function Arguments and Parameters	171
8.4	Functions As Values	176
8.5	Functions As Namespaces	178
8.6	Closures	180
8.7	Function Properties, Methods, and Constructor	186
8.8	Functional Programming	191
9.	Classes and Modules	199
9.1	Classes and Prototypes	200
9.2	Classes and Constructors	201
9.3	Java-Style Classes in JavaScript	205
9.4	Augmenting Classes	208
9.5	Classes and Types	209
9.6	Object-Oriented Techniques in JavaScript	215
9.7	Subclasses	228
9.8	Classes in ECMAScript 5	238
9.9	Modules	246
10.	Pattern Matching with Regular Expressions	251
10.1	Defining Regular Expressions	251
10.2	String Methods for Pattern Matching	259
10.3	The RegExp Object	261
11.	JavaScript Subsets and Extensions	265
11.1	JavaScript Subsets	266
11.2	Constants and Scoped Variables	269
11.3	Destructuring Assignment	271
11.4	Iteration	274
11.5	Shorthand Functions	282
11.6	Multiple Catch Clauses	283
11.7	E4X: ECMAScript for XML	283
12.	Server-Side JavaScript	289
12.1	Scripting Java with Rhino	289
12.2	Asynchronous I/O with Node	296

Part II. Client-Side JavaScript

13. JavaScript in Web Browsers	307
13.1 Client-Side JavaScript	307
13.2 Embedding JavaScript in HTML	311
13.3 Execution of JavaScript Programs	317
13.4 Compatibility and Interoperability	325
13.5 Accessibility	332
13.6 Security	332
13.7 Client-Side Frameworks	338
14. The Window Object	341
14.1 Timers	341
14.2 Browser Location and Navigation	343
14.3 Browsing History	345
14.4 Browser and Screen Information	346
14.5 Dialog Boxes	348
14.6 Error Handling	351
14.7 Document Elements As Window Properties	351
14.8 Multiple Windows and Frames	353
15. Scripting Documents	361
15.1 Overview of the DOM	361
15.2 Selecting Document Elements	364
15.3 Document Structure and Traversal	371
15.4 Attributes	375
15.5 Element Content	378
15.6 Creating, Inserting, and Deleting Nodes	382
15.7 Example: Generating a Table of Contents	387
15.8 Document and Element Geometry and Scrolling	389
15.9 HTML Forms	396
15.10 Other Document Features	405
16. Scripting CSS	413
16.1 Overview of CSS	414
16.2 Important CSS Properties	419
16.3 Scripting Inline Styles	431
16.4 Querying Computed Styles	435
16.5 Scripting CSS Classes	437
16.6 Scripting Stylesheets	440
17. Handling Events	445
17.1 Types of Events	447

17.2	Registering Event Handlers	456
17.3	Event Handler Invocation	460
17.4	Document Load Events	465
17.5	Mouse Events	466
17.6	Mousewheel Events	471
17.7	Drag and Drop Events	474
17.8	Text Events	481
17.9	Keyboard Events	484
18.	Scripted HTTP	491
18.1	Using XMLHttpRequest	494
18.2	HTTP by <script>: JSONP	513
18.3	Comet with Server-Sent Events	515
19.	The jQuery Library	523
19.1	jQuery Basics	524
19.2	jQuery Getters and Setters	530
19.3	Altering Document Structure	537
19.4	Handling Events with jQuery	540
19.5	Animated Effects	551
19.6	Ajax with jQuery	558
19.7	Utility Functions	571
19.8	jQuery Selectors and Selection Methods	574
19.9	Extending jQuery with Plug-ins	582
19.10	The jQuery UI Library	585
20.	Client-Side Storage	587
20.1	localStorage and sessionStorage	589
20.2	Cookies	593
20.3	IE userData Persistence	599
20.4	Application Storage and Offline Webapps	601
21.	Scripted Media and Graphics	613
21.1	Scripting Images	613
21.2	Scripting Audio and Video	615
21.3	SVG: Scalable Vector Graphics	622
21.4	Graphics in a <canvas>	630
22.	HTML5 APIs	667
22.1	Geolocation	668
22.2	History Management	671
22.3	Cross-Origin Messaging	676
22.4	Web Workers	680

22.5	Typed Arrays and ArrayBuffers	687
22.6	Blobs	691
22.7	The Filesystem API	700
22.8	Client-Side Databases	705
22.9	Web Sockets	712

Part III. Core JavaScript Reference

Core JavaScript Reference	719
--	------------

Part IV. Client-Side JavaScript Reference

Client-Side JavaScript Reference	859
Index	1019

Core JavaScript Reference

arguments[]

an array of function arguments

Synopsis

```
arguments
```

Description

The `arguments[]` array is defined only within a function body. Within the body of a function, `arguments` refers to the Arguments object for the function. This object has numbered properties and serves as an array containing all arguments passed to the function. The `arguments` identifier is essentially a local variable automatically declared and initialized within every function. It refers to an Arguments object only within the body of a function and is undefined in global code.

See Also

Arguments; Chapter 8

Arguments

arguments and other properties of a function

Object → Arguments

Synopsis

```
arguments  
arguments[n]
```

Elements

The Arguments object is defined only within a function body. Although it is not technically an array, the Arguments object has numbered properties that function as array elements and a `length` property that specifies the number of array elements. Its elements are the values that are passed as arguments to the function. Element 0 is the first argument, element 1 is the second argument, and so on. All values passed as arguments become array elements of the Arguments object, whether or not those arguments are given names in the function declaration.

Properties

callee

A reference to the function that is currently executing.

length

The number of arguments passed to the function and the number of array elements in the Arguments object.

Description

When a function is invoked, an Arguments object is created for it, and the local variable `arguments` is automatically initialized to refer to that Arguments object. The main purpose of the Arguments object is to provide a way to determine how many arguments are passed to the function and to refer to unnamed arguments. In addition to the array elements and `length` property, however, the `callee` property allows an unnamed function to refer to itself.

For most purposes, the Arguments object can be thought of as an array with the addition of the `callee` property. However, it is not an instance of Array, and the `Arguments.length` property does not have any of the special behaviors of the `Array.length` property and cannot be used to change the size of the array.

In non-strict mode, the Arguments object has one *very* unusual feature. When a function has named arguments, the array elements of the Arguments object are synonyms for the local variables that hold the function arguments. The Arguments object and the argument names provide two different ways of referring to the same variable. Changing the value of an argument with an argument name changes the value that is retrieved through the Arguments object, and changing the value of an argument through the Arguments object changes the value that is retrieved by the argument name.

See Also

Function; Chapter 8

Arguments.callee

not defined in strict mode

the function that is currently running

Synopsis

```
arguments.callee
```

Description

`arguments.callee` refers to the function that is currently running. It provides a way for an unnamed function to refer to itself. This property is defined only within a function body.

Example

```
// An unnamed function literal uses the callee property to refer
// to itself so that it can be recursive
var factorial = function(x) {
    if (x < 2) return 1;
```

```

    else return x * arguments.callee(x-1);
  }
  var y = factorial(5); // Returns 120

```

Arguments.length

the number of arguments passed to a function

Synopsis

```
arguments.length
```

Description

The `length` property of the `Arguments` object specifies the number of arguments passed to the current function. This property is defined only within a function body.

Note that this property specifies the number of arguments actually passed, not the number expected. See `Function.length` for the number of declared arguments. Note also that this property does not have any of the special behavior of the `Array.length` property.

Example

```

// Use an Arguments object to check that correct # of args were passed
function check(args) {
  var actual = args.length;           // The actual number of arguments
  var expected = args.callee.length; // The expected number of arguments
  if (actual !== expected) {         // Throw exception if they don't match
    throw new Error("Wrong number of arguments: expected: " +
      expected + "; actually passed " + actual);
  }
}
// A function that demonstrates how to use the function above
function f(x, y, z) {
  check(arguments); // Check for correct number of arguments
  return x + y + z; // Now do the rest of the function normally
}

```

See Also

`Array.length`, `Function.length`

Array

built-in support for arrays

Object → Array

Constructor

```

new Array()
new Array(size)
new Array(element0, element1, ..., elementn)

```

Arguments

size

The desired number of elements in the array. The returned array has its `length` field set to *size*.

element0, ... elementn

An argument list of two or more arbitrary values. When the `Array()` constructor is invoked with these arguments, the newly created array is initialized with the specified argument values as its elements and its `length` field set to the number of arguments.

Returns

The newly created and initialized array. When `Array()` is invoked with no arguments, the returned array is empty and has a `length` field of 0. When invoked with a single numeric argument, the constructor returns an array with the specified number of undefined elements. When invoked with any other arguments, the constructor initializes the array with the values specified by the arguments. When the `Array()` constructor is called as a function, without the `new` operator, it behaves exactly as it does when called with the `new` operator.

Throws

RangeError

When a single integer *size* argument is passed to the `Array()` constructor, a `RangeError` exception is thrown if *size* is negative or is larger than $2^{32}-1$.

Literal Syntax

ECMAScript v3 specifies an array literal syntax. You may also create and initialize an array by placing a comma-separated list of expressions within square brackets. The values of these expressions become the elements of the array. For example:

```
var a = [1, true, 'abc'];
var b = [a[0], a[0]*2, f(x)];
```

Properties

`length`

A read/write integer specifying the number of elements in the array or, when the array does not have contiguous elements, a number one larger than the index of the last element in the array. Changing the value of this property truncates or extends the array.

Methods

The methods `every()`, `filter()`, `forEach()`, `indexOf()`, `lastIndexOf()`, `map()`, `reduce()`, `reduceRight()`, and `some()` are new in ECMAScript 5 but were implemented by browsers other than IE before ES5 was standardized.

`concat()`

Concatenates elements to an array.

`every()`

Test whether a predicate is true for every array element.

- filter()**
Return array elements that satisfy a predicate function.
- forEach()**
Invoke a function for each element of the array.
- indexOf()**
Search an array for a matching element.
- join()**
Converts all array elements to strings and concatenates them.
- lastIndexOf()**
Search backward through an array.
- map()**
Compute new array elements from the elements of this array.
- pop()**
Removes an item from the end of an array.
- push()**
Pushes an item to the end of an array.
- reduce()**
Compute a value from the elements of this array.
- reduceRight()**
Reduce this array from right-to-left.
- reverse()**
Reverses, in place, the order of the elements of an array.
- shift()**
Shifts an element off the beginning of an array.
- slice()**
Returns a subarray slice of an array.
- some()**
Test whether a predicate is true for at least one element of this array.
- sort()**
Sorts, in place, the elements of an array.
- splice()**
Inserts, deletes, or replaces array elements.
- toLocaleString()**
Converts an array to a localized string.
- toString()**
Converts an array to a string.
- unshift()**
Inserts elements at the beginning of an array.

Description

Arrays are a basic feature of JavaScript and are documented in detail in Chapter 7.

See Also

Chapter 7

Array.concat()

concatenate arrays

Synopsis

```
array.concat(value, ...)
```

Arguments

value, ...

Any number of values to be concatenated with *array*.

Returns

A new array, which is formed by concatenating each of the specified arguments to *array*.

Description

`concat()` creates and returns a new array that is the result of concatenating each of its arguments to *array*. It does not modify *array*. If any of the arguments to `concat()` is itself an array, the elements of that array are concatenated, rather than the array itself.

Example

```
var a = [1,2,3];
a.concat(4, 5)           // Returns [1,2,3,4,5]
a.concat([4,5]);        // Returns [1,2,3,4,5]
a.concat([4,5],[6,7])   // Returns [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]])  // Returns [1,2,3,4,5,[6,7]]
```

See Also

Array.join(), Array.push(), Array.splice()

Array.every()

ECMAScript 5

test whether a predicate is true for every element

Synopsis

```
array.every(predicate)
array.every(predicate, o)
```