JavaScript权威指南（影印版）

第六版
下册

JavaScript

The Definitive Guide

O'REILLY®
東南大學出版社

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

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

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.

PART III

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

Part II. Client-Side JavaScript

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)
```