

核心 JavaScript 参考手册

这部分是 JavaScript 核心 API 中的所有类、属性、函数、方法和事件处理程序的完整参考。这一部分前几页先解释如何使用本参考手册，值得重点阅读。

这部分文档所包括的类和对象有：

Arguments	Global	Number
Array	JSONArray	Object
Boolean	JavaClass	RegExp
Date	JavaObject	String
Error	JavaPackage	Function
Math		

核心 JavaScript 参考手册

这部分是一个参考手册，记载了 JavaScript 核心语言定义的类、方法和属性。这个引言和示例参考用来解释如何使用这个参考手册，并从中获得最大的收益。仔细阅读这部分内容，你会更容易找到和使用你需要的信息。

这份参考手册中的条目是按照字母顺序排列的。类的属性和方法都以它们全名的字母顺序排列，这个全名包含定义它们的类的名字。例如，如果想使用 `String` 类的方法 `replace()` 方法，应该查询 `String.replace()`，而不只是 `replace`。

JavaScript 核心语言定义了一些全局函数和属性，如 `eval()` 和 `NaN`。从技术上说，它们是全局对象的属性。但因为全局对象没有名字，所以它们列在自己的非限定名的参考页中。为了方便起见，JavaScript 核心语言中的全局函数和属性的完整集合总结在一个特殊的参考页中，该页的名称为“Global”（尽管没有以 `Global` 命名的对象或类）。

一旦找到了要查询的参考页，那么找到你需要的信息就应该没有太大的困难了。不过，如果知道参考页是如何编写和组织的，就能更好地使用参考手册部分。接下来是一个示例参考页，标题为“示例条目”，它示范了每个参考页的结构，告诉你在哪里可以找到什么类型的信息。在研究参考手册的其余部分之前，应该花时间阅读该参考页。

示例条目

可用性

如何阅读 JavaScript 核心参考手册

从……继承

标题和简述

每个参考条目的标题由四部分组成，如上所示。各个条目按照标题的字母顺序排列。标题行下的简短描述说明了该条目中记叙的项目，它可以帮助你快速地判断是否对该参考页余下的部分感兴趣。

可用性

标题右上角是可用性信息。在本书以前的版本中，这些信息告诉你该项目得到了哪些 Web 浏览器的什么版本的支持。今天，大多数浏览器都支持本书在这里列出的项目，这个可用性部分会告诉你什么标准为该项目提供了正式规范。例如，你可能会在这里看到“ECMAScript v1”或“2级 DOM HTML”。如果该项目已经不再使用了，也会在这里提示。浏览器的名称和版本号有时候也会出现在这里，通常所介绍的项目还是较新兴的功能，并没有被广泛采用，或者该项目是特定于 IE 的功能。

如果一个项目还没有标准化但是在支持 JavaScript 的一个特定版本的浏览器中得到了较好的支持，那么，这部分可能列出 JavaScript 的版本号。例如，History 对象，其可访问性为“JavaScript 1.0”。

如果一个方法的条目没有包含任何可用性信息，这意味着它和定义该方法的类有着同样的可访问性。

从……继承 / 覆盖

如果一个类继承了超类，或一个方法覆盖了超类中的方法，该信息将显示在标题的右下角。第9章讲过，JavaScript 的类可以继承其他类的属性和方法。例如，String 类继承 Object 类，HTMLDocument 类继承 Document 类，Document 类继承 Node 类。String 的条目就把这一继承概括为“Object → String”，而 HTMLDocument 的条目则表示为“Node → Document → HTMLDocument”。在看到这些继承信息时，便可以查询列出的超类的信息。

当一个方法与超类中的方法同名时，该方法覆盖超类的方法。例如，参阅 Array.toString()。

构造函数

如果参考页介绍一个类并且这个类有构造函数，通常用“构造函数”部分来说明如何使用构造方法创建类的实例。由于构造函数是一种方法，因此“构造函数”部分与方法参考页的“摘要”部分看起来很像。

摘要

函数、方法和属性的参考页都有“摘要”部分，该部分展示了如何在代码中使用这些函数、方法和属性。本书中的参考条目使用两种不同风格的摘要。核心 JavaScript 参考中的条目和客户端条目使用不带类型的摘要来介绍和 DOM 不相关的方法（如 Window 方法）。例如，Array.concat() 方法的摘要如下：

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

斜体字说明要替换的文本。array 应该用变量或存放数组及计算数组值的表达式来替换。value 只是要传递给方法的一个参数的名称。这个指定的参数稍后会在摘要中描述，并且这个描述包含了有关参数的类型和作用的信息。省略号 (...) 说明该方法可以有任意多个 value 参数。因为 concat 和开括号及闭括号不是以斜体显示的，所以在 JavaScript 代码中，必须完全采用它们。

客户端JavaScript部分介绍的很多方法都是被W3C标准化了的,它们的规范包含了方法参数和方法返回值的明确的类型信息。这些条目把这些类型信息包含在摘要里。例如, `Document.getElementById()` 方法的摘要如下:

```
Element getElementById(String elementId);
```

这个摘要使用Java式的语法来表明`getElementById()`返回一个`Element`对象并且期待一个名为`elementId`的字符串参数。由于这是`Document`的一个方法,它隐式地在一个文档上调用,但并不包括`document`前缀。

参数

如果参考页描述的函数、方法或类具有构造函数方法,那么“构造函数”或“摘要”后还有“参数”部分,说明这个函数、方法或构造函数的参数。如果没有参数,这个部分将被省略。

arg1

参数部分中的参数用一个列表描述。例如,这就是对参数 *arg1* 的描述。

arg2

这是对参数 *arg2* 的描述。

返回值

如果函数或方法具有返回值,就用这一部分来说明它们的返回值。

抛出

如果构造函数、函数或方法可以抛出异常,这一部分列出了可能被抛出的异常类型,并解释了在什么环境下会抛出异常。

常量

有些类定义了一组常量,作为一个属性的值或者作为一个方法的参数。例如, `Node` 接口定义了重要的常量来充当其 `nodeType` 属性的一组合法值。当一个接口定义了常量,本部分会列出并介绍它们。注意,这些常量是类自身的静态属性,而不是类的实例的属性。

属性

如果参考页说明的是一个类,“属性”部分就会列出这个类定义的属性并且对每个属性进行简短的解释。在本书第三部分,每个属性还具有自己的参考页。在第四部分大多数属性都在属性列表中完整地描述了。最重要的或复杂的客户端属性也有它们自己的参考页,并且这也是值得注意的。在第四部分,和DOM相关的类的属性包含了类型信息。其他的属性和第三部分的所有属性都是没有类型的。

属性列表如下:

`prop1`

这是无类型的属性 `prop1` 的概述。上面只是列出了属性名，但这个描述会告诉你属性的类型、作用以及它是只读的还是可读可写的。

`readonly integer prop2`

这是一个有类型的属性 `prop2`。属性名和它的类型一起出现。这个描述性的段落说明了该属性的作用。

方法

如果一个类定义了方法，那么它的参考页就具有“方法”部分。这一部分与“属性”部分相似，只不过它说明的是方法而不是属性。所有方法还具有自己的参考页。这个概括部分只是提供方法名的列表。参数类型和返回值类型信息可以在方法的参考页中找到。

描述

大多数参考页都具有“描述”部分，该部分对要说明的类、方法、函数或属性进行了描述。它是参考页的核心。如果你是第一次学习类、方法或属性，可以直接跳到这个部分，然后再返回前面查看“参数”、“属性”和“方法”部分。如果你已经熟悉了类、方法或属性，可以不必再阅读这一部分，只需要快速查看某些特定的信息（例如“参数”部分或“属性”部分）即可。

在某些参考页中，这一部分只是一小段。但是在某些参考页中，这一部分可能会占用一页甚至更多的篇幅。对于那些非常简单的方法，“参数”和“返回值”部分已经对它进行了充分的说明，所以它的“描述”部分就被省略了。

例子

有些参考页还包括一个例子，该例子展示了该项目的习惯用法。但大多数参考页没有例子部分，可以在本书的前半部分找到它们的例子。

Bugs

当一个项目不能正常运行时，这一部分描述了导致这种现象的 bug。但要注意，本书并不打算列出所有 JavaScript 版本和实现中的每一个 bug。

参阅

许多参考页的结尾都包含相关的参考信息。有时它们还会引用本书中的某个主要章节。

`arguments[]`

ECMAScript v1

函数参数的数组

摘要

`arguments`

描述

`arguments[]` 数组只在函数体内定义。在函数体内，`arguments` 引用该函数的 `Arguments` 对象。该对象有带编号的属性，并作为一个存放传递给函数的所有参数的数组。标识符 `arguments` 本质上是一个局部变量，在每个函数中都会被自动声明并初始化。它只在函数体中才能引用 `Arguments` 对象，在全局代码中没有定义。

参阅

`Arguments`、第 8 章

Arguments

ECMAScript v1

一个函数的参数和其他属性

Object → Arguments

摘要

`arguments`
`arguments[n]`

元素

`Arguments` 对象只在函数体中定义。虽然技术上说来，它不是数组，但 `Arguments` 对象有带编号的属性，这些属性可以作为数组元素，而且它有 `length` 属性，该属性声明了数组元素的个数。它的元素是作为参数传递给函数的值。元素 0 是第一个参数，元素 1 是第二个参数，以此类推。所有作为参数传递的值都会成为 `Arguments` 对象的数组元素，无论函数声明中是否有这些参数的名称。

属性

`callee`

对当前正在执行的函数的引用。

`length`

传递给函数的参数个数，同时也是 `Arguments` 对象中的数组元素个数。

描述

当一个函数被调用时，会为该函数创建一个 `Arguments` 对象，局部变量 `arguments` 也会自动地初始化以便引用那个 `Arguments` 对象。`Arguments` 对象的主要用途是提供一种方法，用来确定传递给函数的参数个数并且引用未命名的参数。除了数组元素和属性 `length` 之外，属性 `callee` 可以使未命名的函数引用自身。

大多数情况下，可以将 `Arguments` 对象看做是具有 `callee` 属性的数组。但它不是 `Array` 类的实例，`Arguments.length` 属性没有 `Array.length` 属性的专有行为，所以不能用它来改变数组的大小。

`Arguments` 对象有一个非常特殊的特性。当函数具有命名的参数时，`Arguments` 对象的数

组元素是存放函数参数的局部变量的同义词。Arguments 对象和参数名提供了引用同一个变量的两种不同方法。用参数名改变参数值，会改变用 Arguments 对象得到的值，改变用 Arguments 对象得到的参数值，也会改变用参数名得到的值。

参阅

Function 和第 8 章

Arguments.callee

ECMAScript v1

当前正在运行的函数

摘要

arguments.callee

描述

属性 arguments.callee 引用当前正在运行的函数。它给未命名的函数提供了一种自我引用的方式。该属性只在函数体内被定义。

例子

```
// 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

ECMAScript v1

传递给函数的参数个数

摘要

arguments.length

描述

Arguments 对象的属性 length 声明了传递给当前函数的参数个数。该属性只在函数体内被定义。

注意，这个属性声明的是实际传递给函数的参数个数，而不是期望传递的参数个数。有关声明的参数个数，请参阅属性“Function.length”。还要注意，该属性不具备 Array.length 属性的专有行为。

例子

```
// 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
}
```

参阅

Array.length, Function.length

Array

ECMAScript v1

对数组的内部支持

Object → Array

构造函数

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

参数

size

期望的数组元素个数。返回的数组，length 字段将被设为 *size* 的值。

element0, ...elementn

两个或多个值的参数列表。当使用这些参数来调用构造函数 Array() 时，新创建的数组的元素就会被初始化为这些值，它的 length 字段也会被设置为参数的个数。

返回值

新创建并被初始化了的数组。如果调用构造函数 Array() 时没有使用参数，那么返回的数组为空，length 字段为 0。当调用构造函数时只传递给它一个数字参数，该构造函数将返回具有指定个数、元素为 undefined 的数组。当用其他参数调用 Array() 时，该构造函数将用参数指定的值初始化数组。当把构造函数作为函数调用，不使用 new 运算符时，它的行为与使用 new 运算符调用它时的行为完全一样。

抛出

RangeError

当只传递给 Array() 构造函数一个整数参数 *size* 时，如果 *size* 是负数，或者大于 $2^{32} - 1$ ，将抛出 RangeError 异常。

直接量语法

ECMAScript v3 规定了数组直接量语法。可以把一个用逗号分隔的表达式列表放在方括号中，创建并初始化一个数组。这些表达式的值将成为数组元素。例如：

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

属性

length

一个可读可写的整数，声明了数组中的元素个数。如果数组中的元素不连续，它就是比数组中的最后一个元素的下标大 1 的整数。改变这个属性的值将截断或扩展数组。

方法

concat()

给数组添加元素。

join()

将数组中的所有元素都转换成字符串，然后连接起来。

pop()

从数组尾部删除一个项目。

push()

把一个项目添加到数组的尾部。

reverse()

在原数组上颠倒数组中元素的顺序。

shift()

将数组头部的元素移出数组头部。

slice()

返回数组的一个子数组。

sort()

在原数组上对数组元素进行排序。

splice()

插入、删除或替换一个数组元素。

toLocaleString()

把数组转换成一个局部字符串。

toString()

把数组转换成一个字符串。

unshift()

在数组的头部插入一个元素。

描述

数组是 JavaScript 的基本句法特性。第 7 章详细说明过它们。

参阅

第 7 章

Array.concat()

ECMAScript v3

连接数组

摘要

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

参数

`value, ...`

要添加到 `array` 中的值，可以是任意多个。

返回值

一个新数组，是把指定的所有参数添加到 `array` 中构成的。

描述

方法 `concat()` 将创建并返回一个新数组，这个数组是将所有参数都添加到 `array` 中生成的。它并不修改 `array`。如果要进行 `concat()` 操作的参数是一个数组，那么添加的是数组中的元素，而不是数组。

例子

```
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]]
```

参阅

`Array.join()`、`Array.push()`、`Array.splice()`

Array.join()

ECMAScript v1

将数组元素连接起来以构建一个字符串

摘要

`array.join()`

`array.join(separator)`

参数

separator

在返回的字符串中用于分隔数组元素的字符或字符串，它是选用的。如果省略了这个参数，用逗号作为分隔符。

返回值

一个字符串，通过把 *array* 的每个元素转换成字符串，然后把这些字符串连接起来，在两个元素之间插入 *separator* 字符串而生成。

描述

方法 `join()` 把每个数组元素转换成一个字符串，然后把这些字符串连接起来，在两个元素之间插入指定的 *separator* 字符串。返回生成的字符串。

可以用 `String` 对象的 `split()` 方法执行相反的操作，即把一个字符串分割成数组元素。详情参见“`String.split()`”参考页。

例子

```
a = new Array(1, 2, 3, "testing");
s = a.join("+"); // s is the string "1+2+3+testing"
```

参阅

`String.split()`

Array.length

ECMAScript v1

数组的大小

摘要

`array.length`

描述

数组的 `length` 属性总是比数组中定义的最后一个元素的下标大一。对于那些具有连续元素，而且以元素 0 开始的常规数组来说，属性 `length` 声明了数组中的元素个数。

数组的 `length` 属性在用构造函数 `Array()` 创建数组时初始化。给数组添加新元素时，如果必要，将更新 `length` 的值：

```
a = new Array();           // a.length initialized to 0
b = new Array(10);         // b.length initialized to 10
c = new Array("one", "two", "three"); // c.length initialized to 3
c[3] = "four";             // c.length updated to 4
c[10] = "blastoff";        // c.length becomes 11
```

设置属性 `length` 的值可以改变数组的大小。如果设置的值比它的当前值小，数组将被截

断，其尾部的元素将丢失。如果设置的值比它的当前值大，数组将增大，新元素被添加到数组尾部，它们的值为 `undefined`。

Array.pop()

ECMAScript v3

删除并返回数组的最后一个元素

摘要

```
array.pop()
```

返回值

array 的最后一个元素

描述

方法 `pop()` 将删除 `array` 的最后一个元素，把数组长度减 1，并且返回它删除的元素的值。如果数组已经为空，则 `pop()` 不改变数组，返回 `undefined` 值。

例子

方法 `pop()` 和它的伴随方法 `push()` 可以提供先进后出 (FILO) 栈的功能。例如：

```
var stack = [];           // stack: []
stack.push(1, 2);         // stack: [1,2]      Returns 2
stack.pop();              // stack: [1]        Returns 2
stack.push([4,5]);        // stack: [1,[4,5]]  Returns 2
stack.pop();              // stack: [1]        Returns [4,5]
stack.pop();              // stack: []         Returns 1
```

参阅

`Array.push()`

Array.push()

ECMAScript v3

给数组添加元素

摘要

```
array.push(value, ...)
```

参数

`value, ...`

要添加到 `array` 尾部的值，可以是一个或多个。

返回值

把指定的值添加到数组后的新长度。

描述

方法 `push()` 将它的参数顺次添加到 `array` 的尾部。它直接修改 `array`，而不是创建一个新的数组。方法 `push()` 和方法 `pop()` 用数组提供先进后出栈的功能。参阅“`Array.pop()`”中的示例。

参阅

`Array.pop()`

`Array.reverse()`

ECMAScript v1

颠倒数组中的元素顺序

摘要

`array.reverse()`

描述

`Array` 对象的方法 `reverse()` 将颠倒数组中元素的顺序。它在原数组上实现这一操作作为替代：重排指定的 `array` 的元素，但并不创建新数组。如果对 `array` 有多个引用，那么通过所有引用都可以看到数组元素的新顺序。

例子

```
a = new Array(1, 2, 3);    // a[0] == 1, a[2] == 3;
a.reverse();              // Now a[0] == 3, a[2] == 1;
```

`Array.shift()`

ECMAScript v3

将元素移出数组

摘要

`array.shift()`

返回值

数组原来的第一个元素。

描述

方法 `shift()` 将把 `array` 的第一个元素移出数组，返回那个元素的值，并且将余下的所有元素前移一位，以填补数组头部的空缺。如果数组是空的，`shift()` 将不进行任何操作，返回 `undefined` 值。注意，该方法不创建新数组，而是直接修改原有的 `array`。

方法 `shift()` 和方法 `Array.pop()` 相似，只不过它在数组头部操作，而不是在尾部操作。该方法常常和 `unshift()` 一起使用。

例子

```
var a = [1, [2,3], 4];
a.shift(); // Returns 1; a = [[2,3], 4]
a.shift(); // Returns [2,3]; a = [4]
```

参阅

`Array.pop()` 和 `Array.unshift()`

Array.slice()

ECMAScript v3

返回数组的一部分

摘要

`array.slice(start, end)`

参数

start

数组片段开始处的数组下标。如果是负数，它声明从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。

end

数组片段结束处的后一个元素的数组下标。如果没有指定这个参数，切分的数组包含从 *start* 开始到数组结束的所有元素。如果这个参数是负数，它声明的是从数组尾部开始算起的元素。

返回值

一个新数组，包含从 *start* 到 *end*（不包括该元素）指定的 *array* 元素。

描述

方法 `slice()` 将返回 *array* 的一部分，或者说是一个子数组。返回的数组包含从 *start* 开始到 *end* 之间的所有元素，但是不包括 *end* 所指的元素。如果没有指定 *end*，返回的数组包含从 *start* 开始到原数组结尾的所有元素。

注意，该方法并不修改数组。如果想删除数组中的一段元素，应该使用方法 `Array.splice()`。

例子

```
var a = [1,2,3,4,5];
a.slice(0,3); // Returns [1,2,3]
a.slice(3);   // Returns [4,5]
a.slice(1,-1); // Returns [2,3,4]
a.slice(-3,-2); // Returns [3]; buggy in IE 4: returns [1,2,3]
```

Bug

在 Internet Explorer 4 中，参数 *start* 不能为负数。这在 IE 的最新版本中已经更正。

参阅

`Array.splice()`

Array.sort()

ECMAScript v1

对数组元素排序

摘要

`array.sort()`

`array.sort(orderfunc)`

参数

orderfunc

用来指定按什么顺序进行排序的函数，可选。

返回值

对数组的引用。注意，数组在原数组上进行排序，不制作副本。

描述

方法 `sort()` 将在原数组上对数组元素进行排序，即排序时不创建新的数组副本。如果调用方法 `sort()` 时没有使用参数，将按字母顺序（更为精确地说，是按照字符编码的顺序）对数组中的元素进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如果有必要的话），以便进行比较。

如果想按照别的顺序进行排序，就必须提供比较函数，该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 *a* 和 *b*，其返回值如下：

- 如果根据你的评判标准，*a* 小于 *b*，在排序后的数组中 *a* 应该出现在 *b* 之前，就返回一个小于 0 的值。
- 如果 *a* 等于 *b*，就返回 0。
- 如果 *a* 大于 *b*，就返回一个大于 0 的值。

注意，数组中 `undefined` 的元素都排列在数组末尾。即使提供了自定义的排序函数，也是这样，因为 `undefined` 值不会被传递给你提供的 *orderfunc*。

例子

下面的代码展示了如何编写按数字顺序，而不是按字母顺序对数组进行排序的比较函数：

```
// An ordering function for a numerical sort
function numberorder(a, b) { return a - b; }
a = new Array(33, 4, 1111, 222);
a.sort(); // Alphabetical sort: 1111, 222, 33, 4
a.sort(numberorder); // Numerical sort: 4, 33, 222, 1111
```

Array.splice()

ECMAScript v3

插入、删除或替换数组的元素

摘要

`array.splice(start, deleteCount, value, ...)`

参数

start

开始插入和（或）删除的数组元素的下标。

deleteCount

从 *start* 开始，包括 *start* 所指的元素在内要删除的元素个数。这个参数是选用的，如果没有指定它，`splice()` 将删除从 *start* 开始到原数组结尾的所有元素。

value, ...

要插入数组的零个或多个值，从 *start* 所指的下标处开始插入。

返回值

如果从 `array` 中删除了元素，则返回的是含有被删除的元素的数组。

描述

方法 `splice()` 将删除从 *start* 开始（包括 *start* 所指的元素在内）的零个或多个元素，并且用参数列表中声明的一个或多个值来替换那些被删除的元素。位于插入或删除的元素之后的数组元素都会被移动，以保持它们与数组其他元素的连续性。注意，虽然 `splice()` 方法与 `slice()` 方法名称相似，但作用不同，方法 `splice()` 直接修改数组。

例子

读了下面的例子，就很容易理解 `splice()` 的操作了：

```
var a = [1,2,3,4,5,6,7,8]
a.splice(4); // Returns [5,6,7,8]; a is [1,2,3,4]
a.splice(1,2); // Returns [2,3]; a is [1,4]
a.splice(1,1); // Returns [4]; a is [1]
a.splice(1,0,2,3); // Returns []; a is [1 2 3]
```

参阅

`Array.slice()`

Array.toLocaleString()

ECMAScript v1

把数组转换成局部字符串

覆盖 object.toLocaleString()

摘要`array.toLocaleString()`**返回值**数组 `array` 的局部字符串表示。**抛出**

TypeError

调用该方法时，若对象不是 Array，则抛出该异常。

描述

数组的方法 `toLocaleString()` 将返回数组的局部字符串表示。它首先调用每个数组元素的 `toLocaleString()` 方法，然后使用地区特定的分隔符把生成的字符串连接起来，形成一个字符串。

参阅`Array.toString()`, `Object.toLocaleString()`

Array.toString()

ECMAScript v1

将数组转换成一个字符串

覆盖 Object.toString()

摘要`array.toString()`**返回值**`array` 的字符串表示。**抛出**

TypeError

调用该方法时，若对象不是 Array，则抛出该异常。

描述

数组的 `toString()` 方法将把数组转换成一个字符串，首先是把每个数组元素转换为字符串并且返回这个字符串。当数组用于字符串环境中，JavaScript 会调用这一方法将数组自动转换成一个字符串。但在某些情况下，需要显式地调用这个方法。

`toString()` 在把数组转换成字符串时，首先要将数组的每个元素都转换成字符串（通过调用这些元素的 `toString()` 方法）。当每个元素都被转换成字符串时，它就以列表的形式输出这些字符串，字符串之间用逗号分隔。返回值与没有参数的 `join()` 方法返回的字符串相同。

参阅

`Array.toLocaleString()`、`Object.toString()`

Array.unshift()

ECMAScript v3

在数组头部插入元素

摘要

`array.unshift(value, ...)`

参数

`value, ...`

要插入数组头部的一个或多个值。

返回值

数组的新长度。

描述

方法 `unshift()` 将把它的参数插入 `array` 的头部，并将已经存在的元素顺次地移到较高的下标处，以便留出空间。该方法的第一个参数将成为数组新的元素 0，如果还有第二个参数，它将成为新的元素 1，以此类推。注意，`unshift()` 不创建新数组，而是直接修改原有的数组。

例子

方法 `unshift()` 通常和方法 `shift()` 一起使用。例如：

```
var a = [];           // a:[]
a.unshift(1);          // a:[1]           Returns: 1
a.unshift(22);         // a:[22,1]        Returns: 2
a.shift();             // a:[1]           Returns: 22
a.unshift(33,[4,5]);   // a:[33,[4,5],1] Returns: 3
```

参阅

`Array.shift()`

Boolean

ECMAScript v1

对布尔值的支持

Object → Boolean

构造函数

```
new Boolean(value) // Constructor function
Boolean(value)      // Conversion function
```

参数

value

由布尔对象存放的值或者要转换成布尔值的值。

返回值

当作为一个构造函数（带有运算符 `new`）调用时，`Boolean()` 将把它的参数转换成一个布尔值，并且返回一个包含该值的 `Boolean` 对象。如果作为一个函数（不带有运算符 `new`）调用的，`Boolean()` 只将它的参数转换成一个原始的布尔值，并且返回这个值。

0、NaN、null、空字符串 "" 和 undefined 值都将转换成 false。其他的原始值，除了 false（但包括字符串 "false"），以及其他的对象和数组都会被转换成 true。

方法`toString()`

根据 Boolean 对象代表的布尔值返回 "true" 或 "false"。

`valueOf()`

返回 Boolean 对象中存放的原始布尔值。

描述

在 JavaScript 中，布尔值是一种基本的数据类型。Boolean 对象是一个将布尔值打包的布尔对象。Boolean 对象主要用于提供将布尔值转换成字符串的 `toString()` 方法。当调用 `toString()` 方法将布尔值转换成字符串时（通常是由 JavaScript 隐式地调用），JavaScript 会内在地将这个布尔值转换成一个临时的 Boolean 对象，然后调用这个对象的 `toString()` 方法。

参阅

Object

Boolean.toString()

ECMAScript v1

将布尔值转换成字符串

覆盖 Object.toString() 方法

摘要`b.toString()`

返回值

根据原始布尔值或者 Boolean 对象 *b* 的值返回字符串 “true” 或 “false”。

抛出

TypeError

如果调用该方法时，对象不是 Boolean，则抛出该异常。

Boolean.valueOf()

ECMAScript v1

Boolean 对象的布尔值

覆盖 Object.valueOf() 方法

摘要

b.valueOf()

返回值

Boolean 对象 *b* 存放的原始布尔值。

抛出

TypeError

如果调用该方法时，对象不是 Boolean，则抛出该异常。

Date

ECMAScript v1

操作日期和时间的对象

Object → Date

构造函数

```
new Date()  
new Date(milliseconds)  
new Date(datestring)  
new Date(year, month, day, hours, minutes, seconds, ms)
```

没有参数的构造函数 `Date()` 将把创建的 Date 对象设置为当前的日期和时间。如果传递给它的参数是一个数字，那么这个数字将被作为日期的内部数字表示，其单位是 *ms*，就像方法 `getTime()` 的返回值一样。如果传递给它的参数是一个字符串，它就是日期的字符串表示，其格式就是方法 `Date.parse()` 接受的格式。否则，传递给该构造函数的参数是 2~7 个数字，它们分别指定了日期和时间的各个字段。除了前两个字段（年和月字段）外，其他所有字段都是选用的。注意，声明这些日期和时间的字段使用的都是本地时间，而不是 UTC 时间（类似于 GMT 时间）。参阅静态方法 `Date.UTC()`。

`Date()` 还可以作为普通函数被调用，而不带有运算符 `new`。以这种方式调用时，`Date()` 将忽略传递给它的所有参数，返回当前日期和时间的字符串表示。

参数

milliseconds

期望的日期距1970年1月1日午夜(UTC)的`ms`数。例如,假定传递的参数值为5000,那么创建的Date对象代表日期的就是1970年1月1日午夜过5s。

datestring

一个字符串,声明了日期,也可以同时声明时间。这个字符串的格式应该是方法Date.parse()能接受的。

year

年份,一个四位数。例如,2001指的是2001年。为了与早期的JavaScript实现兼容,如果它的值在0~99之间,则给它加上1900。

month

月份,0(代表一月)到11(代表十二月)之间的一个整数。

day

一个月的某一天,1~31之间的一个整数。注意,这个参数将1作为它的最小值,而其他参数则以0为最小值。该参数是选用的。

hours

小时,0(午夜)到23(晚上11点)之间的一个整数。该参数是选用的。

minutes

分钟,0~59之间的一个整数。该参数是选用的。

seconds

秒,0~59之间的一个整数。该参数是选用的。

ms

毫秒,0~999之间的一个整数。该参数是选用的。

方法

Date对象没有可以直接读写的属性,所有对日期和时间值的访问都是通过方法执行的。Date对象的大多数方法采用两种形式,一种是使用本地时间进行操作,另一种是使用世界(UTC或GMT)时进行操作。如果方法的名称中有“UTC”,它将使用世界时进行操作。下面将这些方法对列在一起。例如,get[UTC]Day()指方法getDay()和getUTCDay()。

只有Date对象才能调用Date方法,如果用其他类型的对象调用这些方法,将抛出异常TypeError。

get[UTC]Date()

返回Date对象所代表的月中的某一天,采用本地时间或世界时。

get[UTC]Day()

返回Date对象所代表的一周中的某一天,采用本地时间或世界时。

get[UTC]FullYear()

返回日期中的年份,用四位数表示,采用本地时间或世界时。

`get[UTC]Hours()`

返回 `Date` 对象的小时字段，采用本地时间或世界时。

`get[UTC]Milliseconds()`

返回 `Date` 对象的毫秒字段，采用本地时间或世界时。

`get[UTC]Minutes()`

返回 `Date` 对象的分钟字段，采用本地时间或世界时。

`get[UTC]Month()`

返回 `Date` 对象的月份字段，采用本地时间或世界时。

`get[UTC]Seconds()`

返回 `Date` 对象的秒字段，采用本地时间或世界时。

`getTime()`

返回 `Date` 对象的内部毫秒表示。注意，该值独立于时区，所以没有单独的 `getUTCtime()` 方法。

`getTimezoneOffset()`

返回这个日期的本地时间和 UTC 表示之间的时差，以分钟为单位。注意，是否是夏令时或在指定的日期中夏令时是否有效，将决定该方法的返回值。

`getYear()`

返回 `Date` 对象的年份。一般不使用这种方法，推荐使用的方法是 `getFullYear()`。

`set[UTC]Date()`

设置 `Date` 对象的月中的某一天，采用本地时间或世界时。

`set[UTC]FullYear()`

设置 `Date` 对象的年份字段（月份和天数字段可选），采用本地时间或世界时。

`set[UTC]Hours()`

设置 `Date` 对象的小时字段（分钟、秒和毫秒字段选用），采用本地时间或世界时。

`set[UTC]Milliseconds()`

设置 `Date` 对象的毫秒字段，采用本地时间或世界时。

`set[UTC]Minutes()`

设置 `Date` 对象的分钟字段（秒和毫秒字段选用），采用本地时间或世界时。

`set[UTC]Month()`

设置 `Date` 对象的月份字段（一个月的天数字段选用），采用本地时间或世界时。

`set[UTC]Seconds()`

设置 `Date` 对象的秒字段（毫秒字段选用），采用本地时间或世界时。

`setTime()`

使用毫秒的形式设置 `Date` 对象的各个字段。

`setYear()`

设置 `Date` 对象的年份字段。不赞成使用该方法，推荐使用的方法是 `setFullYear()`。

`toDateString()`

返回日期的日期部分的字符串表示，采用本地时间。

`toGMTString()`

将 `Date` 对象转换成一个字符串，采用 GMT 时间区。该方法被反对使用，推荐使用的方法是 `toUTCString()`。

`toLocaleDateString()`

返回表示日期的日期部分的字符串，采用地方日期，使用地方日期的格式化规约。

`toLocaleString()`

将 `Date` 对象转换成一个字符串，采用本地时间和地方日期的格式化规约。

`toLocaleTimeString()`

返回日期的时间部分的字符串表示，采用本地时间，使用本地时间的格式化规约。

`toString()`

将 `Date` 对象转换成一个字符串，采用本地时间。

`getTimeString()`

返回日期的时间部分的字符串表示，采用本地时间。

`toUTCString()`

将 `Date` 对象转换成一个字符串，采用世界时。

`valueOf()`

将 `Date` 对象转换成它的内部毫秒格式。

静态方法

除了上面列出的实例方法之外，`Date` 对象还定义了两个静态方法。这两个方法由构造函数 `Date()` 自身调用，而不是由 `Date` 对象调用：

`Date.parse()`

解析日期和时间的字符串表示，返回它的内部毫秒表示。

`Date.UTC()`

返回指定的 UTC 日期和时间的毫秒表示。

描述

`Date` 对象是 JavaScript 语言的一种内部数据类型。它由语法 `new Date()` 语法创建，我们在前面已经说明了这种语法。

创建了 `Date` 对象后，就可以使用多种方法来操作它。大多数方法只能用来设置或者获取对象的年份字段、月份字段、天数字段、小时字段、分钟字段以及秒字段，采用本地时间或 UTC（世界时或 GMT）时间。方法 `toString()` 以及它的变种可以把日期转换成人们能够读懂的字符串。所谓 `Date` 对象的内部表示就是距 1970 年 1 月 1 日午夜（GMT 时间）的毫秒数，方法 `getTime()` 可以把 `Date` 对象转换为内部表示，方法 `setTime()` 可以把它从内部表示转换成其他形式。采用标准的毫秒格式时，日期和时间由一个整数表示，这使得

日期算术变得格外简单。ECMAScript 标准要求 Date 对象能够把 1970 年 1 月 1 日前后 10 亿天中的任意日期和时间表示为毫秒。这个范围在正负 273~785 年之间, 所以 JavaScript 的时钟不会超过 275755 年。

例子

一旦创建了 Date 对象, 就可以用各种方法操作它:

```
d = new Date(); // Get the current date and time
document.write('Today is: ' + d.toLocaleDateString() + '. '); // Display date
document.write('The time is: ' + d.toLocaleTimeString()); // Display time
var dayOfWeek = d.getDay(); // What weekday is it?
var weekend = (dayOfWeek == 0) || (dayOfWeek == 6); // Is it a weekend?
```

Date 对象的另一种常见用法是用某个时间的毫秒表示减去当前时间的毫秒表示来判断两个时间的时差。下面的客户端的例子说明了这种用法:

```
<script language="JavaScript">
today = new Date(); // Make a note of today's date
christmas = new Date(); // Get a date with the current year
christmas.setMonth(11); // Set the month to December...
christmas.setDate(25); // and the day to the 25th
// If Christmas hasn't already passed, compute the number of
// milliseconds between now and Christmas, convert this
// to a number of days and print a message
if (today.getTime() < christmas.getTime()) {
    difference = christmas.getTime() - today.getTime();
    difference = Math.floor(difference / (1000 * 60 * 60 * 24));
    document.write('Only ' + difference + ' days until Christmas!<p>');
}
</script>
// ... rest of HTML document here ...
<script language="JavaScript">
// Here we use Date objects for timing
// We divide by 1000 to convert milliseconds to seconds
now = new Date();
document.write('<p>It took ' +
    (now.getTime() - today.getTime()) / 1000 +
    'seconds to load this page.');
```

参见

Date.parse(), Date.UTC()

Date.getDate

ECMAScript v1

返回日期的一个月中的某一天的字段

摘要

date.getDate()

返回值

指定 Date 对象 *date* 所指的月份中的某一天，使用本地时间。返回值是 1~31 之间的一个整数。

Date.getDay()

ECMAScript v1

返回日期的一周中的某一天字段

摘要

```
date.getDay()
```

返回值

指定 Date 对象 *date* 所指的一个星期中的某一天，使用本地时间。返回值是 0（周日）到 6（周六）之间的一个整数。

Date.getFullYear

ECMAScript v1

返回年份

摘要

```
date.getFullYear()
```

返回值

当 *date* 用本地时间表示时返回的年份。返回值是一个四位数，表示包括世纪值在内的完整年份，而不是两位数的缩写形式。

Date.getHours()

ECMAScript v1

返回 Date 对象的 hour 字段

摘要

```
date.getHours()
```

返回值

指定 Date 对象 *date* 的 hour 字段，以本地时间表示。返回值是 0（午夜）到 23（晚上 11 点）之间的一个整数。

Date.getMilliseconds()

ECMAScript v1

返回 Date 对象的 millisecond 字段

摘要

```
date.getMilliseconds()
```

返回值

date 的毫秒字段，用本地时间表示。

Date.getMinutes()

ECMAScript v1

返回 **Date** 对象的分钟字段

摘要

date.getMinutes()

返回值

指定的 **Date** 对象 *date* 的分钟字段，以本地时间表示。返回值在 0~59 之间。

Date.getMonth()

ECMAScript v1

返回 **Date** 对象的月份字段

摘要

date.getMonth()

返回值

指定的 **Date** 对象 *date* 的月份字段，以本地时间表示。返回值在 0（一月）到 11（十二月）之间。

Date.getSeconds()

ECMAScript v1

返回 **Date** 对象的秒字段

摘要

date.getSeconds()

返回值

指定的 **Date** 对象 *date* 的秒字段，以本地时间表示。返回值在 0~59 之间。

Date.getTime()

ECMAScript v1

返回 **Date** 对象的毫秒表示

摘要

date.getTime()

返回值

指定的 **Date** 对象 *date* 的毫秒表示，也就是 *date* 指定的日期和时间距 1970 年 1 月 1 日午夜（GMT 时间）之间的毫秒数。

描述

方法 `getTime()` 可以将日期和时间转换成一个整数。这在比较两个 `Date` 对象或者要判断两个日期之间的时差时非常有用。注意，日期的毫秒表示独立于时区，所以除了这个方法外，没有 `getUTCTime()` 方法。不要混淆 `getTime()` 方法和 `getDay()` 及 `getDate()` 方法，`getDay()` 及 `getDate()` 返回的分别是一周的第几天和一个月的第几天。

可以使用 `Date.parse()` 或 `Date.UTC()` 将日期和时间转换成它们的毫秒表示，在此之前无须先创建一个 `Date` 对象。

参阅

`Date`、`Date.parse()`、`Date.setTime()` 和 `Date.UTC()`

`Date.getTimezoneOffset()`

ECMAScript v1

判断与 GMT 的时间差

摘要

`date.getTimezoneOffset()`

返回值

本地时间与 GMT 时间之间的时差，以分钟为单位。

描述

`getTimezoneOffset()` 返回的是本地时间和 GMT 时间或 UTC 时间之间相差的分钟数。实际上，该函数告诉了你运行 JavaScript 代码的时区，以及指定的时间是否是夏令时。

返回值以分钟计，而不是以小时计，原因是某些国家所占有的时区甚至不到一个小时的间隔。

`Date.getUTCDate()`

ECMAScript v1

返回该天是一个月的哪一天（世界时）

摘要

`date.getUTCDate()`

返回值

当 `date` 对象用世界时表示时，返回值是该月中的哪一天（是 1~31 的一个值）。

`Date.getUTCDay()`

ECMAScript v1

返回该天是星期几（世界时）

摘要

`date.getUTCDay()`

返回值

当 *date* 对象用世界时表示时，返回值是该星期中的哪一天。该值在 0（星期天）到 6（星期六）之间。

Date.getUTCFullYear()

ECMAScript v1

返回年份（世界时）

摘要

```
date.getUTCFullYear()
```

返回值

当 *date* 对象是用世界时表示时所代表的年份。该值是四位数，而不是两位数的缩写。

Date.getUTCHours()

ECMAScript v1

返回 *Date* 对象的 *小时* 字段（世界时）

摘要

```
date.getUTCHours()
```

返回值

当 *date* 对象用世界时表示时的小时字段，该值在 0（午夜）到 23（晚上 11 点）之间。

Date.getUTCMilliseconds()

ECMAScript v1

返回 *Date* 对象的 *毫秒* 字段（世界时）

摘要

```
date.getUTCMilliseconds()
```

返回值

当 *date* 对象是用世界时表示时的毫秒字段。

Date.getUTCMinutes()

ECMAScript v1

返回 *Date* 对象的 *分钟* 字段（世界时）

摘要

```
date.getUTCMinutes()
```

返回值

当 *date* 对象用世界时表示时的分钟字段，该值是 0~59 之间的整数。

Date.getUTCMonth()ECMAScript v1

返回 Date 对象的月份（世界时）

摘要

`date.getUTCMonth()`

返回值

当 `date` 对象用世界时表示时的月份，该值是 0（一月）到 11（十二月）之间的一个整数。注意，Date 对象以 1 代表某个月的第一天，而不是像月份字段那样使用 0 代表一年的第一个月。

Date.getUTCSeconds()ECMAScript v1

返回 Date 对象的秒字段（世界时）

摘要

`date.getUTCSeconds()`

返回值

当 `date` 对象用世界时表示时的秒字段，该值是 0~59 之间的一个整数。

Date.getYear()ECMAScript v3 反对使用

返回 Date 对象的年份字段（世界时）

摘要

`date.getYear()`

返回值

指定 Date 对象 `date` 的年份字段减去 1900。

描述

方法 `getYear()` 返回的是指定的 Date 对象的年份字段减去 1900 后的值。从 ECMAScript v3 起，JavaScript 的实现就不再要求使用该函数，而使用 `getFullYear()` 函数代替它。

Date.parse()ECMAScript v1

解析日期/时间字符串

摘要

`Date.parse(date)`

参数

date

含有要解析的日期和时间的字符串。

返回值

指定的日期和时间距 1970 年 1 月 1 日午夜（GMT 时间）之间的毫秒数。

描述

`Date.parse()` 是 `Date` 对象的静态方法。一般通过 `Date` 构造函数，采用 `Date.parse()` 的形式调用它，而不是通过 `date` 对象，采用 `date.parse()` 调用该方法。`Date.parse()` 只有一个字符串型的参数。它将解析这个字符串中的日期，然后返回它的毫秒形式，这种形式可以直接使用，也可以用于创建一个新的 `Date` 对象，还可以用 `Date.setTime()` 方法来设置一个已经存在的日期。

ECMAScript 标准没有规定 `Date.parse()` 方法解析的字符串的格式，只是说该方法能解析 `Date.toString()` 方法和 `Date.toUTCString()` 方法返回的字符串。但是，这些函数根据实现来格式化日期，所以通常以某种方式编写所有 JavaScript 实现都能理解的日期是不可能的。

参阅

`Date`、`Date.setTime()`、`Date.toGMTString()` 和 `Date.UTC()`

`Date.setDate()`

ECMAScript v1

设置一个月的某一天

摘要

`date.setDate(day_of_month)`

参数

day_of_month

1~31 之间的整数，作为 `date` 的月中某一天字段的新值（以本地时间计）。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化之前，该方法什么都不返回。

`Date.setFullYear()`

ECMAScript v1

设置年份，也可以设置月份和天

摘要

```
date.setFullYear(year)
date.setFullYear(year, month)
date.setFullYear(year, month, day)
```

参数

year

在 *date* 中设置的年份, 用本地时间表示。该参数应该是一个包含世纪值的完整年份, 如 1999, 而不仅是年份的缩写, 如 99。

month

可选的整数, 在 0~11 之间, 用作 *date* 的月份字段的新值 (以本地时间计)。

day

可选的整数, 在 1~31 之间, 用作 *date* 的天数字段的新值 (以本地时间计)。

返回值

调整过的日期的毫秒表示。

Date.setHours()

ECMAScript v1

设置 *Date* 对象的小时字段、分钟字段、秒字段和毫秒字段

摘要

```
date.setHours(hours)
date.setHours(hours, minutes)
date.setHours(hours, minutes, seconds)
date.setHours(hours, minutes, seconds, millis)
```

参数

hours

0 (午夜) 到 23 (晚上 11 点) 之间的整数, 用作 *date* 的小时字段的新值 (以本地时间计)。

minutes

可选的整数, 在 0~59 之间, 用作 *date* 的分钟字段的新值 (以本地时间计)。ECMAScript 标准化前, 不支持该参数。

seconds

可选的整数, 在 0~59 之间, 用作 *date* 的秒字段的新值 (以本地时间计)。ECMAScript 标准化前, 不支持该参数。

millis

可选的整数，在 0~999 之间，用作 *date* 的毫秒字段的新值（以本地时间计）。ECMAScript 标准化前，不支持该参数。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化前，该方法不返回值。

Date.setMilliseconds()

ECMAScript v1

设置 Date 对象的毫秒字段

摘要

`date.setMilliseconds(millis)`

参数

millis

用于设置 *date* 的毫秒字段，用本地时间表示。该参数是 0~999 之间的整数。

返回值

调整过的日期的毫秒表示。

Date.setMinutes()

ECMAScript v1

设置 Date 对象的分钟字段和秒字段

摘要

`date.setMinutes(minutes)`

`date.setMinutes(minutes, seconds)`

`date.setMinutes(minutes, seconds, millis)`

参数

minutes

0~59 之间的整数，用于设置 Date 对象 *date* 的分钟字段（以本地时间计）。

seconds

可选的整数，在 0~59 之间，用做 *date* 的秒字段的新值（以本地时间计）。ECMAScript 标准化前，不支持该参数。

millis

可选的整数，在 0 到 999 之间，用作 *date* 的毫秒字段的新值（以本地时间计）。ECMAScript 标准化前，不支持该参数。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化前，该方法不返回值。

Date.setMonth()ECMAScript v1

设置 **Date** 对象的月份字段和天字段

摘要

```
date.setMonth(month)
date.setMonth(month, day)
```

参数

month

0 (一月) 到 11 (十二月) 之间的整数, 用于设置 **Date** 对象 *date* 的月份字段。注意, 月份从 0 开始编号, 而月中的某一天则从 1 开始编号。

day

可选的整数, 在 1~31 之间, 用做 *date* 的天数字段的新值 (以本地时间计)。ECMAScript 标准化前, 不支持该参数。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化前, 该方法不返回值。

Date.setSeconds()ECMAScript v1

设置 **Date** 对象的秒字段和毫秒字段

摘要

```
date.setSeconds(seconds)
date.setSeconds(seconds, millis)
```

参数

seconds

0~59 之间的一个整数, 用于设置 **Date** 对象 *date* 的秒字段。

millis

可选的整数, 在 0~999 之间, 用做 *date* 的毫秒字段的新值 (以本地时间计)。ECMAScript 标准化前, 不支持该参数。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化前, 该方法不返回值。

Date.setTime()ECMAScript v1

以毫秒设置 **Date** 对象

摘要

```
date.setTime(milliseconds)
```

参数

milliseconds

要设置的日期和时间距 GMT 时间 1970 年 1 月 1 日午夜之间的毫秒数。这种类型的毫秒值可以传递给 `Date()` 构造函数，可以通过调用 `Date.UTC()` 和 `Date.parse()` 方法获得该值。以毫秒形式表示日期可以使它独立于时区。

返回值

参数 *milliseconds*。在 ECMAScript 标准化前，该方法不返回值。

Date.setUTCDate()

ECMAScript v1

设置一个月中的某一天（世界时）

摘要

`date.setUTCDate(day_of_month)`

参数

day_of_month

要给 *date* 设置的一个月中的某一天，用世界时表示。该参数是 1~31 之间的整数。

返回值

调整过的日期的毫秒表示。

Date.setUTCFullYear()

ECMAScript v1

设置年份、月份和天（世界时）

摘要

`date.setUTCFullYear(year)`

`date.setUTCFullYear(year, month)`

`date.setUTCFullYear(year, month, day)`

参数

year

要给 *date* 设置的年份值，用世界时表示。该参数应该是含有世纪值的完整年份，如 1999，而不只是缩写的年份值，如 99。

month

可选的整数，在 0~11 之间，用作 *date* 的月份字段的新值（以世界時計）。注意，月份是以 0 开始计算的，而该月中的日期是从 1 开始计数的。

day

可选的整数，在 1~31 之间，用作 *date* 的天字段的新值（以世界時計）。

返回值

调整过的日期的毫秒表示。

Date.setUTCHours()

ECMAScript v1

设置 Date 对象的小时字段、分钟字段、秒字段和毫秒字段（世界时）

摘要

```
date.setUTCHours(hours)
date.setUTCHours(hours, minutes)
date.setUTCHours(hours, minutes, seconds)
date.setUTCHours(hours, minutes, seconds, millis)
```

参数

hours

要设置的 *date* 的小时字段的值，用世界时表示。该参数应该是 0（午夜）到 23（晚上 11 点）之间的一个整数。

minutes

可选的整数，在 0~59 之间，用作 *date* 的分钟字段的新值（以世界時計）。

seconds

可选的整数，在 0~59 之间，用作 *date* 的秒字段的新值（以世界時計）。

millis

可选的整数，在 0~999 之间，用作 *date* 的毫秒字段的新值（以世界時計）。

返回值

调整过的日期的毫秒表示。

Date.setUTCMilliseconds()

ECMAScript v1

设置 Date 对象的毫秒字段（世界时）

摘要

```
date.setUTCMilliseconds(millis)
```

参数

millis

要设置的 *date* 的毫秒字段的值，用世界时表示。该参数是 0~999 之间的整数。

返回值

调整过的日期的毫秒表示。

Date.setUTCMinutes()ECMAScript v1

设置 `Date` 对象的分钟字段和秒字段（世界时）

摘要

```
date.setUTCMinutes(minutes)
date.setUTCMinutes(minutes, seconds)
date.setUTCMinutes(minutes, seconds, millis)
```

参数

minutes

要设置的 *date* 的分钟字段的值，用世界时表示。该参数应该是 0~59 之间的一个整数。

seconds

可选的整数，在 0~59 之间，用作 *date* 的秒字段的新值（以世界時計）。

millis

可选的整数，在 0~999 之间，用作 *date* 的毫秒字段的新值（以世界時計）。

返回值

调整过的日期的毫秒表示。

Date.setUTCMonth()ECMAScript v1

设置 `Date` 对象的月份字段和天数字段（世界时）

摘要

```
date.setUTCMonth(month)
date.setUTCMonth(month, day)
```

参数

month

要设置的 *date* 的月份字段的值，用世界时表示。该参数是 0（一月）到 11（十二月）之间的整数。注意，月份从 0 开始编码，而月中的某一天则从 1 开始编码。

day

可选的整数，在 1~31 之间，用作 *date* 的天字段的新值（以世界時計）。

返回值

调整过的日期的毫秒表示。

Date.setUTCSeconds()ECMAScript v1

设置 `Date` 对象的秒字段和毫秒字段（世界时）

摘要

```
date.setUTCSeconds(seconds)  
date.setUTCSeconds(seconds, millis)
```

参数

seconds

要设置的 *date* 的秒字段的值，用世界时表示。该参数应该是 0~59 之间的一个整数。

millis

可选的整数，在 0~999 之间，用作 *date* 的毫秒字段的新值（以世界時計）。

返回值

调整过的日期的毫秒表示。

Date.setYear()

ECMAScript v1;ECMAScript v3 反对使用

设置 Date 对象的年份字段

摘要

```
date.setYear(year)
```

参数

year

要设置的 Date 对象 *date* 的年份字段的值，是一个整数。如果这个值在 0~99 之间，包括 0 和 99，将给它加上 1900，作为 1900~1999 间的值处理。

返回值

调整过的日期的毫秒表示。在 ECMAScript 标准化前，该方法不返回值。

描述

方法 `setYear()` 可以设置指定的 Date 对象的年份字段，对于 1900~1999 之间的年份，带有特殊的行为。

从 ECMAScript v3 起，JavaScript 实现不再要求使用该函数，而使用 `setFullYear()` 函数代替它。

Date.toString()

ECMAScript v3

返回 Date 对象日期部分作为字符串

摘要

```
date.toString()
```

返回值

`date` 的日期部分的字符串表示，由实现决定、人们可以读懂，以本地时间表示。

参阅

`Date.toLocaleDateString()`

`Date.toLocaleString()`

`Date.toLocaleTimeString()`

`Date.toString()`

`Date.toTimeString()`

`Date.toGMTString()`

ECMAScript v1;ECMAScript v3 反对使用

将 `Date` 转换为世界时字符串

摘要

`date.toGMTString()`

返回值

`Date` 对象 `date` 所指定的日期和时间的字符串表示。这个日期在转换成字符串之前由本地时区转换成了 GMT 时区。

描述

不赞成使用方法 `toGMTString()`，而赞同使用 `Date.toUTCString()`。

从 ECMAScript v3 起，JavaScript 的实现不再要求使用该函数，而用 `toUTCString()` 代替它。

参阅

`Date.toUTCString()`

`Date.toLocaleDateString()`

ECMAScript v3

返回 `Date` 对象的日期部分作为本地已格式化的字符串

摘要

`date.toLocaleDateString()`

返回值

`date` 的日期部分的字符串表示，由实现决定、人们可以读懂，以本地时间表示，根据本地规则格式化。

参阅

`Date.toDateString()`、`Date.toLocaleString()`、`Date.toLocaleTimeString()`、`Date.toString()`、`Date.toTimeString()`

`Date.toLocaleString()`

ECMAScript v1

将 `Date` 转换为本地已格式化的字符串

摘要

`date.toLocaleString()`

返回值

`Date` 对象 `date` 指定的日期和时间的字符串表示。该日期和时间用本地时间区表示，并根据本地规则格式化。

用法

方法 `toLocaleString()` 可以将日期转换成用本地时间区表示的字符串。该方法的日期和时间格式还使用地方规则，所以在不同平台上以及不同国家之间，日期和时间的格式都有所不同。它返回的字符串格式通常都是用户想要的日期和时间格式。

参阅

`Date.toLocaleDateString()`、`Date.toLocaleTimeString()`、`Date.toString()`、`Date.toUTCString()`

`Date.toLocaleTimeString()`

ECMAScript v3

返回 `Date` 对象的时间部分作为本地已格式化的字符串

摘要

`date.toLocaleTimeString()`

返回值

`date` 的时间部分的字符串表示，由实现决定、人们可以读懂的，以本地时区表示，并根据本地规则格式化。

参阅

`Date.toDateString()`、`Date.toLocaleDateString()`、`Date.toLocaleString()`、`Date.toString()`、`Date.toTimeString()`

Date.toString()

ECMAScript v1

将 Date 转换为字符串

覆盖 Object.toString()

摘要

`date.toString()`

返回值

`date` 的字符串表示，是人们可以读懂的，用本地时间表示。

描述

方法 `toString()` 返回一个人们可以读懂的、由实现决定的日期的字符串表示。它与 `toUTCString()` 的不同，`toString()` 以本地时间表示日期。而它与 `toLocaleString()` 的不同之处在于，它不使用地方规则采用的形式表示日期和时间。

参阅

`Date.parse()`
`Date.toDateString()`
`Date.toLocaleString()`
`Date.toString()`
`Date.toUTCString()`

Date.toTimeString()

ECMAScript v3

返回 Date 对象日期部分作为字符串

摘要

`date.toTimeString()`

返回值

`date` 的时间部分的字符串表示，由实现决定，人们可以读懂，以本地时间表示。

参阅

`Date.toString()`、`Date.toDateString()`、`Date.toLocaleDateString()`、`Date.toLocaleString()`、`Date.toLocaleTimeString()`

Date.toUTCString()

ECMAScript v1

将 Date 转换为字符串（世界时）

摘要

`date.toUTCString()`

返回值

date 的字符串表示，人们可以读取，用世界时表示。

描述

toUTCString() 返回一个特定于实现的字符串，这个字符串用世界时表示日期。

参阅

Date.toLocaleString()、*Date.toString()*

Date.UTC()

ECMAScript v1

将 *Date* 规范转换成毫秒数

摘要

Date.UTC(year, month, day, hours, minutes, seconds, ms)

参数

year

四位数表示的年份值。如果该参数在 0~99 之间（包括 0 和 99），它将加上 1900，作为 1900~1999 之间的年份处理。

month

月份值，是 0（一月）到 11（十二月）之间的整数。

day

一个月中的某一天，是 1~31 之间的整数。注意，该参数的最小值是 1，而其他参数的最小值则是 0。该参数是选用的。

hours

小时值，是 0（午夜）到 23（晚上 11 点）之间的整数。该参数是选用的。

minutes

分钟值，是 0~59 之间的整数。该参数是选用的。

seconds

秒值，是 0~59 之间的整数。该参数是选用的。

ms

毫秒数，是 0~999 之间的整数。该参数是选用的。在 ECMAScript 标准化前，忽略该参数。

返回值

指定的世界时的毫秒表示。简而言之，该方法返回指定的时间距 GMT 时间 1970 年 1 月 1 日午夜的毫秒数。

描述

`Date.UTC()` 是一种静态方法，它通过构造函数 `Date()` 调用，而不是通过某个 `Date` 对象调用。

`Date.UTC()` 方法的参数指定日期和时间，它们都是 UTC 时间，处于 GMT 时区。指定的 UTC 时间将转换成毫秒的形式，这样构造函数 `Date()` 和方法 `Date.setTime()` 就可以使用它了。

`Date.UTC()` 能接受的日期和时间格式，构造函数 `Date()` 也可以接受。区别在于构造函数 `Date()` 假定这些参数是本地时间，而 `Date.UTC()` 却假定它们是世界时 (GMT 时间)。要创建使用 UTC 时间规则的 `Date` 对象，可以使用如下的代码：

```
d = new Date(Date.UTC(1996, 4, 8, 16, 30));
```

参阅

`Date`、`Date.parse()` 和 `Date.setTime()`

`Date.valueOf()`

ECMAScript v1

将 `Date` 转换成毫秒表示

覆盖 `Object.valueOf()`

摘要

`date.valueOf()`

返回值

`date` 的毫秒表示。返回值和方法 `Date.getTime()` 返回的值相等。

`decodeURI()`

ECMAScript v3

URI 中未转义的字符

摘要

`decodeURI(uri)`

参数

`uri`

一个字符串，含有编码的 URI 或要其他要解码的文本。

返回值

`uri` 的副本，其中十六进制的转义序列被它们表示的字符替换了。

抛出

`URIError`

说明 `uri` 中的一个或多个转义序列被错误地格式化，不能被正确解码。

描述

`decodeURI()` 是一个全局函数，它返回参数 *uri* 解码后的副本。它将保留 `encodeURIComponent()` 方法执行的编码操作，详见该函数的参考页。

参阅

`decodeURIComponent()`、`encodeURIComponent()`、`escape()`、`unescape()`

`decodeURIComponent()`

ECMAScript v3

URI 组件中的未转义字符

摘要

`decodeURI(s)`

参数

s 一个字符串，含有编码 URI 组件或其他要解码的文本。

返回值

s 的副本，其中十六进制的转义序列被它们所表示的字符替换。

抛出

`URIError`

说明 *s* 中的一个或多个转义序列被错误地格式化，不能被正确解码。

描述

`decodeURIComponent()` 是一个全局函数，它返回参数 *s* 解码后的副本。它将保留 `encodeURIComponent()` 方法执行的编码操作，详见该函数的参考页。

参阅

`decodeURI()`、`encodeURIComponent()`、`escape()`、`unescape()`

`encodeURIComponent()`

ECMAScript v3

URI 中的转义字符

摘要

`encodeURIComponent(uri)`

参数

uri

一个字符串，含有 URI 或其他要编码的文本。

返回值

`uri` 的副本，其中某些字符被十六进制的转义序列替换了。

抛出

`URIError`

说明 `uri` 中含有格式化错误的 Unicode 替代对，不能被编码。

描述

`encodeURIComponent()` 是全局函数，返回参数 `uri` 的编码副本。ASCII 的字母和数字不编码，此外下面的 ASCII 标点符号也不编码：

`- _ . ! ~ * ' ()`

因为 `encodeURIComponent()` 的目的是给 URI 进行完整的编码，所以以下在 URI 中具有特殊含义的 ASCII 标点符号也不转义：

`; / ? : @ & = + $, #`

`uri` 中的其他字符都将转换成它的 UTF-8 编码字符，然后用十六进制的转义序列（形式为 `%xx`）对生成的一个、两个或三个字节的字符编码，用它们替换 `uri` 中原有的字符。在这种编码模式中，ASCII 字符由一个 `%xx` 转义字符替换，在 `\u0080` 到 `\u07ff` 之间编码的字符由两个转义序列替换，其他的 16 位 Unicode 字符由三个转义序列替换。

如果使用该方法编码 URI，应该确保 URI 组件（如查询字符串）中不含有 URI 分隔符，如 `?` 和 `#`。如果组件中含有这些符号，应该用 `encodeURIComponent()` 方法分别对各个组件编码。

用方法 `decodeURI()` 可以对该方法进行解码操作。在 ECMAScript v3 之前，可以用 `escape()` 和 `unescape()` 方法（反对使用）执行相似的编码解码操作。

例子

```
// Returns http://www.isp.com/app.cgi?arg1=1&arg2=hello%20world
encodeURIComponent("http://www.isp.com/app.cgi?arg1=1&arg2=hello world");
encodeURIComponent("\u00a9"); // The copyright character encodes to %C2%A9
```

参阅

`decodeURI()`、`decodeURIComponent()`、`encodeURIComponent()`、`escape()`、`unescape()`

`encodeURIComponent()`

ECMAScript v3

转义 URI 组件中的字符

摘要

`encodeURIComponent(s)`

参数

s 一个字符串，含有 URI 的一部分或其他要编码的文本。

返回值

s 的副本，其中某些字符被十六进制的转义序列替换了。

抛出

URIError

说明 *s* 中含有格式化错误的 Unicode 替代对，不能被编码。

描述

`encodeURIComponent()` 是全局函数，返回参数 *s* 的编码副本。ASCII 的字母和数字不编码，此外下面的 ASCII 标点符号也不编码：

`- _ . ! ~ * ' ()`

其他字符（像 `/`、`:`、`#` 这样用于分隔 URI 各种组件的标点符号），都由一个或多个十六进制的转义序列替换。关于使用的编码模式，请参阅“`encodeURIComponent()`”的描述。

注意 `encodeURIComponent()` 和 `encodeURIComponent()` 之间的差别，前者假定它的参数是 URI 的一部分（如协议、主机名、路径或查询字符串）。因此，它将转义用于分隔 URI 各个部分的标点符号。

例子

```
encodeURIComponent("hello world?"); // Returns hello%20world%3F
```

参阅

`decodeURI()`、`decodeURIComponent()`、`encodeURIComponent()`、`escape()`、`unescape()`

Error

ECMAScript v3

普通异常

Object → Error

构造函数

```
new Error()
```

```
new Error(message)
```

参数

message

提供异常的详细信息的出错消息，选用。

返回值

新构造的 Error 对象。如果指定了参数 *message*，该 Error 对象将它作为 *message* 属性的

值；否则，它将用实现定义的默认字符串作为该属性的值。如果把 `Error()` 构造函数当作函数调用时不使用 `new` 运算符，它的行为与使用 `new` 运算符调用时一样。

属性

`message`

提供异常详细信息的出错消息。该属性存放传递给构造函数的字符串，或实现定义的默认字符串。

`name`

声明异常类型的字符串。对于 `Error` 类的实例和所有子类来说，该属性声明了用于创建实例的构造函数名。

方法

`toString()`

返回一个表示 `Error` 对象的字符串，该字符串由实现定义。

描述

`Error` 类的实例表示错误或异常，通常与 `throw` 语句和 `try/catch` 语句一起使用。属性 `name` 声明了异常的类型，`message` 属性可提供人们能够读懂的异常的详细信息。

JavaScript 解释器从不直接抛出 `Error` 对象，而是抛出 `Error` 子类（如 `SyntaxError` 或 `RangeError`）的实例。在代码中，你会发现抛出 `Error` 对象指示异常非常方便，或者也可以用原始字符串或数字的形式抛出出错消息或出错代码。

注意，ECMAScript 标准为 `Error` 类定义了 `toString()` 方法（`Error` 的所有子类都继承了该方法），但并不要求该方法返回含有 `message` 属性的字符串。因此，不能期望 `toString()` 方法可以把 `Error` 对象转换成人们可以读懂的字符串。要把出错消息显示给用户，应该明确地使用 `Error` 对象的 `name` 属性和 `message` 属性。

例子

可以用下列代码指示一个异常：

```
function factorial(x) {  
    if (x < 0) throw new Error("factorial: x must be >= 0");  
    if (x <= 1) return 1; else return x * factorial(x-1);  
}
```

如果捕获到一个异常，可以用下列代码把它显示给用户（这段代码使用了客户端 `Window.alert()` 方法）：

```
try { /*(&*/ an error is thrown here */ }  
catch(e) {  
    if (e instanceof Error) { // Is it an instance of Error or a subclass?  
        alert(e.name + ": " + e.message);  
    }  
}
```

参阅

`EvalError`、`RangeError`、`ReferenceError`、`SyntaxError`、`TypeError`、`URIError`

Error.message

ECMAScript v3

人们可以读懂的出错消息

摘要

`error.message`

描述

`Error` 对象（或 `Error` 子类的实例）的 `message` 属性用于存放包含发生的错误或异常的详细情况的字符串，该字符串是人们可以读懂的。如果传递给 `Error()` 构造函数一个消息参数，该消息将成为 `message` 属性的值。如果没有消息参数传递给 `Error()` 参数，`Error` 对象将继承实现为该属性定义的默认值（可能是空串）。

Error.name

ECMAScript v3

错误的类型

摘要

`error.name`

描述

`Error` 对象（或 `Error` 子类的实例）的 `name` 属性声明了发生的错误或异常的类型。所有 `Error` 对象都从它们的构造函数中继承这一属性。该属性的值与构造函数名相同。因此，`SyntaxError` 对象的 `name` 属性值为 “`SyntaxError`”，`EvalError` 对象的 `name` 属性为 “`EvalError`”。

Error.toString()

ECMAScript v3

把 `Error` 对象转换成字符串

覆盖 `Object.toString()`

摘要

`error.toString()`

返回值

实现定义的字符串。ECMAScript 标准除了规定该方法的返回值是字符串外，没有再做其他规定。尤其是，它不要求返回的字符串包含错误名或出错消息。

escape()

ECMAScript v3 反对使用

对字符串编码

摘要

`escape(s)`

参数

s 要被转义或编码的字符串。

返回值

编码了的 *s* 的副本，其中某些字符被替换成了十六进制的转义序列。

描述

`escape()` 是全局函数。它返回一个含有 *s* 的编码版本的新字符串。*s* 自身并没有被修改。在 `escape()` 返回的字符串中，除了 ASCII 字母、数字和标点符号 `@`、`*`、`_`、`+`、`-`、`.` 和 `\` 之外，所有字符都由形为 `%xx` 或 `%uxxxx` (*x* 表示十六进制的数字) 的转义序列替代。从 `\u0000` 到 `\u00ff` 的 Unicode 字符由转义序列 `%xx` 替代，其他所有 Unicode 字符由 `%uxxxx` 序列替代。

使用函数 `unescape()` 可以对 `escape()` 编码的字符串进行解码。

虽然 ECMAScript 的第一个版本标准化了 `escape()` 函数，但是 ECMAScript v3 反对使用该方法，并从标准中删除了它。ECMAScript 的实现可能实现了该函数，但它不是必需的。应该用 `encodeURIComponent()` 和 `encodeURIComponent()` 代替 `escape()`。

例子

```
escape("Hello World!"); // Returns "Hello%20World%21"
```

参阅

`encodeURIComponent()`、`encodeURIComponent()`

eval()

ECMAScript v1

执行字符串中的 JavaScript 代码

摘要

`eval(code)`

参数

code

字符串，含有要计算的 JavaScript 表达式或要执行的语句。

返回值

计算 *code* 得到的值（如果存在的话）。

抛出

SyntaxError

说明 *code* 中没有合法的 JavaScript 表达式或语句。

EvalError

说明非法调用了 `eval()`，例如使用的标识符不是“eval”。参阅下面描述的对该函数的限制。

其他异常

如果传递给 `eval()` 的 JavaScript 代码生成了一个异常，`eval()` 将把那个异常传递给调用者。

描述

`eval()` 是全局方法，它将执行一个 JavaScript 代码的字符串。如果 *code* 含有一个表达式，`eval()` 将计算这个表达式，并返回它的值。如果 *code* 含有一个或多个 JavaScript 语句，`eval()` 将执行这些语句，如果最后一个语句有返回值，它还会返回这个值。如果 *code* 没有返回任何值，`eval()` 将返回 `undefined`。最后，如果 *code* 抛出了一个异常，`eval()` 将把这个异常传递给调用者。

虽然 `eval()` 给 JavaScript 语言提供了非常强大的功能，但在实际程序中极少用它。常见的用法是编写作为递归的 JavaScript 解释器的程序，或者编写动态生成并计算 JavaScript 代码的程序。

大部分使用字符串参数的 JavaScript 函数和方法都会接受其他类型的参数，在继续操作之前把这些参数值转换成字符串。但 `eval()` 的行为不是这样。如果 *code* 参数不是原始的字符串，它将不作任何改变地返回。所以，要注意，当打算传递给 `eval()` 原始字符串值时，不要粗心地给它传递 String 对象。

考虑到实现的效率，ECMAScript v3 标准给 `eval()` 方法的使用加上了一条与众不同的限制。如果试图覆盖 `eval` 属性或把 `eval()` 方法赋予另一个属性，并通过该属性调用它，则 ECMAScript 实现允许抛出一个 `EvalError` 异常。

例子

```
eval("1+2");           // Returns 3
// This code uses client-side JavaScript methods to prompt the user to
// enter an expression and to display the results of evaluating it.
// See the client-side methods Window.alert() and Window.prompt() for
// details.
try {
    alert("Result: " + eval(prompt("Enter an expression:", "")));
}
catch(exception) {
    alert(exception);
}
```

```
}  
var myeval = eval; // May throw an EvalError  
myeval("1+2");     // May throw an EvalError
```

EvalError

ECMAScript v3

在不正确使用 `eval()` 时抛出

Object → Error → EvalError

构造函数

```
new EvalError()  
new EvalError(message)
```

参数

message

提供异常的详细信息的出错消息，选用。如果设置了该参数，它将用作 EvalError 对象的 *message* 属性的值。

返回值

新构造的 EvalError 对象。如果指定了参数 *message*，Error 对象将用它作为 *message* 属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果把 EvalError() 构造函数当作函数调用且不带有 new 运算符，它的行为与使用 new 运算符调用时一样。

属性

message

提供异常的详细信息的出错消息。该属性存放传递给构造函数的字符串，或存放实现定义的默认字符串。详见“Error.message”。

name

声明异常类型的字符串。所有 EvalError 对象的 *name* 属性都继承值“EvalError”。

描述

当在其他名称下调用全局函数 `eval()` 时，EvalError 类的一个实例就会被抛出。关于调用 `eval()` 函数的限制，请参阅“eval()”。关于抛出和捕捉异常的细节，请参阅“Error”。

参阅

Error、Error.message、Error.name

Function

ECMAScript v1

JavaScript 的函数

Object → Function

摘要

```
function functionname(argument_name_list) // Function definition statement  
{
```

```
    body
}
function (argument_name_list) {body}    // Unnamed function literal
functionname(argument_value_list)      // Function invocation
```

构造函数

`new Function(argument_names..., body)`

参数

argument_names...

任意多个字符串参数，每个字符串命名一个或多个要创建的 Function 对象的参数。

body

一个字符串，指定函数的主体，可以含有任意多条 JavaScript 语句，这些语句之间用分号隔开，可以给该构造函数引用前面的参数设置的任何参数名。

返回值

新创建的 Function 对象。调用该函数，将执行 *body* 指定的 JavaScript 代码。

抛出

SyntaxError

说明在参数 *body* 或某个 *argument_names* 参数中存在 JavaScript 语法错误。

属性

`arguments[]`

一个参数数组，元素是传递给函数的参数。反对使用该属性。

`caller`

对调用当前函数的 Function 对象的引用，如果当前函数由顶层代码调用，这个属性的值为 `null`。反对使用该属性。

`length`

在声明函数时指定的命名参数的个数。

`prototype`

一个对象，用于构造函数，这个对象定义的属性和方法由构造函数创建的所有对象共享。

方法

`apply()`

将函数作为指定对象的方法来调用，传递给它的是指定的参数数组。

`call()`

将函数作为指定对象的方法来调用，传递给它的是指定的参数。

`toString()`

返回函数的字符串表示。

描述

函数是 JavaScript 的一种基本数据类型。本书的第 8 章解释了如何定义和使用函数，第 9 章介绍了函数方法、构造函数以及 `prototype` 属性等相关主题。要了解详细情况，请参看这两章。注意，虽然可以用这里介绍的 `Function()` 构造函数创建函数对象，但这样做效率不高，在大多数情况下，建议使用函数定义语句或函数直接量来定义函数。

在 JavaScript 1.1 及以后版本中，函数主体会被自动地给予一个局部变量 `arguments`，它引用一个 `Arguments` 对象。该对象是一个数组，元素是传递给函数的参数值。不要将这一属性和上面介绍的反对使用的属性 `arguments[]` 相混淆。详见“`Arguments`”的参考页。

参阅

`Arguments`、第 8 章和第 9 章

`Function.apply()`

ECMAScript v3

将函数作为一个对象的方法调用

摘要

`function.apply(thisobj, args)`

参数

thisobj

调用 *function* 的对象。在函数主体中，*thisobj* 是关键字 `this` 的值。如果这个参数为 `null`，就使用全局对象。

args

一个数组，它的元素是要传递给函数 *function* 的参数值。

返回值

调用函数 *function* 的返回值。

抛出

`TypeError`

如果调用该函数的对象不是函数，或参数 *args* 不是数组和 `Arguments` 对象，则抛出该异常。

描述

`apply()` 将指定的函数 *function* 作为对象 *thisobj* 的方法来调用，传递给它的是存放在

数组 `args` 中的参数，返回的是调用 `function` 的返回值。在函数体内，关键字 `this` 引用 `thisobj` 对象。

参数 `args` 必须是数组或 `Arguments` 对象。如果想单独指定传递给函数的参数，而不是指定数组元素，请使用 `Function.call()` 方法。

例子

```
// Apply the default Object.toString() method to an object that
// overrides it with its own version of the method. Note no arguments.
Object.prototype.toString.apply(o);
// Invoke the Math.max() method with apply to find the largest
// element in an array. Note that first argument doesn't matter
// in this case.
var data = [1,2,3,4,5,6,7,8];
Math.max.apply(null, data);
```

参阅

`Function.call()`

`Function.arguments[]`

ECMAScript v1;ECMAScript v3 反对使用

传递给函数的参数

摘要

```
function.arguments[i]
function.arguments.length
```

描述

`Function` 对象的 `arguments` 属性是一个参数数组，它的元素是传递给函数的参数。只在执行函数时，它才被定义。`arguments.length` 声明的是数组中的元素个数。

反对使用该属性，赞成使用 `Arguments` 对象。虽然 ECMAScript v1 支持 `Function.arguments` 属性，但 ECMAScript v3 删除了它，遵守该标准的实现不再支持该属性。因此，在新的 JavaScript 代码中，不再应该使用它。

参阅

`Arguments`

`Function.call()`

ECMAScript v3

将函数作为对象的方法调用

摘要

```
function.call(thisobj, args...)
```

参数

thisobj

调用 *function* 的对象。在函数主体中，*thisobj* 是关键字 *this* 的值。如果这个参数为 *null*，就使用全局对象。

args...

任意多个参数，这些参数将传递给函数 *function*。

返回值

调用函数 *function* 的返回值。

抛出

TypeError

如果调用该函数的对象不是函数，则抛出该异常。

描述

call() 将指定的函数 *function* 作为对象 *thisobj* 的方法来调用，把参数列表中 *thisobj* 后的参数传递给它，返回值是调用函数后的返回值。在函数体内，关键字 *this* 引用 *thisobj* 对象，或者如果 *thisobj* 为 *null*，就使用全局对象。

如果指定数组中传递给函数的参数，请使用 *Function.apply()* 方法。

例子

```
// Call the default Object.toString() method on an object that  
// overrides it with its own version of the method. Note no arguments.  
Object.prototype.toString.call(o);
```

参阅

Function.apply()

Function.caller

JavaScript 1.0; ECMAScript 反对使用

调用当前函数的函数

摘要

function.caller

描述

在 JavaScript 的早期版本中，*Function* 对象的 *caller* 属性是对调用当前函数的函数的引用。如果该函数是从 JavaScript 程序的顶层调用的，*caller* 的值就为 *null*。该属性只能在函数内部使用（例如，*caller* 属性只有在函数执行时才会有定义）。

`Function.caller` 属性不属于 ECMAScript 标准, 在遵守该标准的实现中, 不需要该属性。不应该再使用它。

Function.length

ECMAScript v1

已声明的参数的个数

摘要

function.length

描述

函数的 `length` 属性在定义函数时声明已命名的参数的个数。实际调用函数时, 传递给它的参数个数既可以比它多, 也可以比它少。不要混淆了 `Function` 对象和 `Arguments` 对象的 `length` 属性, 后者声明的是实际传递给函数的参数个数。参阅 “`Arguments.length`” 中的示例。

参阅

`Arguments.length`

Function.prototype

ECMAScript v1

对象类的原型

摘要

function.prototype

描述

属性 `prototype` 是在函数作为构造函数时使用的。它引用的是作为整个对象类的原型的对象。由这个构造函数创建的任何对象都会继承属性 `prototype` 引用的对象的所有属性。

要了解 JavaScript 中的构造函数、`prototype` 属性和类定义的完整讨论, 请参阅第 9 章。

参阅

第 9 章

Function.toString()

ECMAScript v1

把函数转换成字符串

摘要

function.toString()

返回值

表示函数的字符串。

抛出

`TypeError`

如果调用该函数的对象不是 `Function`，则抛出该异常。

描述

`Function`对象的方法 `toString()` 可以以一种与实现相关的方式将函数转换成字符串。在大多数实现中，如在 Firefox 和 IE 这样的实现中，该方法返回一个含有有效 JavaScript 代码的字符串，即包括关键字 `function`、参数列表和函数的完整主体的代码。在这些实现中，这个 `toString()` 方法的输出是全局 `eval()` 函数的合法输入。规范并没有要求这一行为，因此，不应该依赖于此。

`getClass()`

LiveConnect

返回一个 `JavaObject` 的 `JavaClass`

摘要

`getClass(javaobj)`

参数

`javaobj`

一个 `JavaObject` 对象。

返回值

`javaobj` 的 `JavaClass` 对象。

描述

`getClass()` 函数接受一个 `JavaObject` 对象 (`javaobj`) 为参数。它返回 `JavaObject` 对象的 `JavaClass`。即它返回 `JavaClass` 对象，该 `JavaClass` 对象表示 Java 对象的 Java 类；而这个 Java 对象所表示的 Java 类是由 `JavaObject` 指定的。

习惯用法

不要把 JavaScript 的 `getClass()` 函数和所有 Java 对象的 `getClass` 方法混淆了。同样，不要把 JavaScript `JavaClass` 对象和 Java `java.lang.Class` 类混淆了。

考虑如下代码行所创建的 `Java Rectangle` 对象：

```
var r = new java.awt.Rectangle();
```

`r` 是保存了一个 `JavaObject` 的 JavaScript 变量。调用 JavaScript 函数 `getClass()` 返回一个 `JavaClass` 对象，该 `JavaClass` 对象表示 `java.awt.Rectangle` 类：

```
var c = getClass(r);
```

可以通过比较这个 `JavaClass` 对象和 `java.awt.Rectangle` 来看到这一点：

```
if (c == java.awt.Rectangle) ...
```

Java `getClass()` 方法的调用方式不同，并且执行完全不同的功能：

```
c = r.getClass();
```

在执行了这行代码之后，`c` 是表示 `java.lang.Class` 对象的一个 Java Object。这个 `java.lang.Class` 对象是一个 Java 对象，它是 `java.awt.Rectangle` 类的一个 Java 表示。参阅 Java 文档详细了解有关可以用 `java.lang.Class` 类做些什么。

最后，对于任何的 `JavaObject o` 你都会看见如下的表达式总是为 `true`。

```
(getClass(o.getClass())) == java.lang.Class)
```

参阅

`JavaArray`、`JavaClass`、`JavaObject`、`JavaPackage`，第 12 章，第 23 章

Global

ECMAScript v1

全局对象

Object → Global

摘要

`this`

全局属性

全局对象不是一个类，所以下面的全局属性在自己名称下有单独的参考条目。也就是说，在“`undefined`”名下可以找到 `undefined` 属性的详细信息，而不是在“`Global.undefined`”下寻找。注意，所有顶层变量也都是全局对象的属性。

`Infinity`

表示正无穷大的数值。

`java`

表示 `java.*` 包层级的一个 `JavaPackage`。

`NaN`

非数字值。

`Packages`

根 `JavaPackage` 对象。

`undefined`

未定义的值。

全局函数

全局对象是一个对象，而不是类。下面列出的全局函数不是任何对象的方法，它们的参考

条目出现在函数名下。例如，在“`parseInt()`”下可以找到`parseInt()`函数的详细信息，在“`Global.parseInt()`”下就无法找到该函数的详细信息。

`decodeURI()`

对 `encodeURIComponent()` 转义的字符串解码。

`decodeURIComponent()`

对 `encodeURIComponent()` 转义的字符串解码。

`encodeURIComponent()`

通过转义某些字符对 URI 编码。

`encodeURIComponent()`

通过转义某些字符对 URI 的组件编码。

`escape()`

用转义序列替换某些字符来对字符串编码。

`eval()`

计算 JavaScript 代码串，返回结果。

`getClass()`

返回一个 `JavaObject` 的 `JavaClass`。

`isFinite()`

检测一个值是否是无穷大的数字。

`isNaN()`

检测一个值是否是非数字的值。

`parseFloat()`

从字符串解析一个数字。

`parseInt()`

从字符串解析一个整数。

`unescape()`

对用 `escape()` 编码的字符串解码。

全局对象

除了上面列出的全局属性和全局函数以外，全局对象还定义了引用 JavaScript 所有预定义对象的属性。除了 `Math` 外，这些属性都是定义类的构造函数，`Math` 引用的对象不是构造函数。

`Array`

构造函数 `Array()`。

`Boolean`

构造函数 `Boolean()`。

`Date`

构造函数 `Date()`。

Error

构造函数 Error()。

EvalError

构造函数 EvalError()。

Function

构造函数 Function()。

Math

对定义算术函数的对象的引用。

Number

构造函数 Number()。

Object

构造函数 Object()。

RangeError

构造函数 RangeError()。

ReferenceError

构造函数 ReferenceError()。

RegExp

构造函数 RegExp()。

String

构造函数 String()。

SyntaxError

构造函数 SyntaxError()。

TypeError

构造函数 TypeError()。

URIError

构造函数 URIError()。

描述

全局对象是预定义的对象，作为 JavaScript 的全局属性和全局函数的占位符。通过使用全局对象，可以访问其他所有预定义的对象、函数和属性。全局对象不是任何对象的属性，所以它没有名字（之所以选择 Global 作为这个参考页的标题，只是为了方便组织，并不是说全局对象名为“Global”）。在顶层 JavaScript 代码中，可以用关键字 `this` 引用全局对象。但通常不必用这种方式引用全局对象，因为全局对象是作用域链的头，这意味着所有非限定性的变量和函数名都会作为该对象的属性来查询。例如，当 JavaScript 代码引用 `parseInt()` 函数时，它引用的是全局对象的 `parstInt` 属性。全局对象是作用域链的头，还意味着在顶层 JavaScript 代码中声明的所有变量都将成为全局对象的属性。

全局对象只是一个对象，而不是类。既没有 `Global()` 构造函数，也无法实例化一个新的全局对象。

当 JavaScript 代码嵌入一个特殊环境中时，全局对象通常具有环境特定的属性。实际上，ECMAScript 标准没有规定全局对象的类型，JavaScript 的实现或嵌入的 JavaScript 都可以把任意类型的对象作为全局对象，只要该对象定义了这里列出的基本属性和函数。例如，在允许通过 LiveConnect 或相关的技术来脚本化 Java 的 JavaScript 实现中，全局对象被赋予了这里列出的 `java` 和 `Package` 属性以及 `getClass()` 方法。而在客户端 JavaScript 中，全局对象是 `Window` 对象，表示运行 JavaScript 代码的 Web 浏览器窗口。

例子

在 JavaScript 核心语言中，全局对象的预定义属性都是不可枚举的，所以可以用 `for/in` 循环列出所有隐式或显式声明的全局变量，如下所示：

```
var variables = ""
for(var name in this)
  variables += name + "\n";
```

参阅

第四部分中的 `Window`; 第 4 章

Infinity

ECMAScript v1

表示无穷大的数字属性

摘要

`Infinity`

描述

`Infinity` 是一个全局属性，它用来存放表示正无穷大的特殊的数值。用 `for/in` 循环不能枚举 `Infinity` 属性，用 `delete` 运算符不能删除它。注意，`Infinity` 不是常量，它可以设为其他值。（但 `Number.POSITIVE_INFINITY` 是常量。）

参阅

`isFinite()`、`NaN`、`Number.POSITIVE_INFINITY`

isFinite()

ECMAScript v1

判断一个数字是否是有限的

摘要

`isFinite(n)`

参数

n 要检测的数字。

返回值

如果 *n* 是有限数字（或者可以转换为有限数字），那么返回值就是 `true`。否则，如果 *n* 是 `NaN`（非数字），或者是正、负无穷大的数，则返回值就是 `false`。

参阅

`Infinity`、`isNaN()`、`NaN`、`Number.NaN`、`Number.NEGATIVE_INFINITY`、`Number.POSITIVE_INFINITY`

isNaN()

ECMAScript v1

检测非数字值

摘要

`isNaN(x)`

参数

x 要检测的值。

返回值

如果 *x* 是特殊的非数字值 `NaN`（或者能被转换为这样的值），返回值就是 `true`。如果 *x* 是其他值，返回值是 `false`。

描述

`isNaN()` 可以通过检测参数来判断值是否是 `NaN`，该值表示一个非法的数字（如被 0 除后得到结果）。这个函数是必需的，因为把 `NaN` 与任何值（包括它自身）进行比较得到的结果都是 `false`，所以要检测一个值是否是 `NaN`，不能使用 `==` 或 `===` 运算符。

`isNaN()` 通常用于检测方法 `parseFloat()` 和 `parseInt()` 的结果，以判断它们表示的是否是合法数字。也可以用 `isNaN()` 来检测算术错误，如用 0 作除数。

例子

```
isNaN(0);           // Returns false
isNaN(0/0);          // Returns true
isNaN(parseInt("3")); // Returns false
isNaN(parseInt("hello")); // Returns true
isNaN("3");          // Returns false
isNaN("hello");       // Returns true
isNaN(true);          // Returns false
isNaN(undefined);    // Returns true
```

参阅

`isFinite()`、`NaN`、`Number.NaN`、`parseFloat()`、`parseInt()`

java

LiveConnect

表示 `java.*` 包层级的 `JavaPackage`

摘要

java

描述

在包含了 LiveConnect 或其他用于脚本化 Java 的技术的视线中，全局 `java` 属性就是一个 `JavaPackage` 对象，它表示 `java.*` 包层级。这个属性的存在意味着像 `java.util` 这样的 JavaScript 表达式引用的是 `java.util` 包。对于不符合 `java.*` 层级的 Java 包，请参阅全局 `Packages` 属性。

参阅

`JavaPackage`、`Packages`，第 12 章

JavaArray

LiveConnect

Java 数组的 JavaScript 表示

摘要

```
javaarray.length // The length of the array
javaarray[index] // Read or write an array element
```

属性

`length`

一个只读的整数，声明了 `JavaArray` 对象表示的 Java 数组中的元素数。

描述

`JavaArray` 对象是 Java 数组的 JavaScript 表示，它使 JavaScript 代码能够使用你所熟悉的 JavaScript 数组语法对数组元素进行读写操作。而且，`JavaArray` 对象的 `length` 字段还声明了 Java 数组中的元素个数。

在对数组元素进行读写操作时，系统会自动执行 JavaScript 表示和 Java 表示之间的数据转换。参阅第 12 章的全面介绍。

习惯用法

注意，Java 数组和 JavaScript 数组有两方面不同之处。第一，Java 数组具有固定的长度，这个长度是在创建它时指定的。因此，`JavaArray` 对象的 `length` 字段是只读的。第二点不

同之处在于, Java 数组的元素是有类型的(如它们的元素都必须具有相同的数据类型)。如果给数组元素设置了错误类型的值, 就会引发 JavaScript 错误或异常。

例子

`java.awt.Polygon` 是一个 `JavaClass` 对象。我们可以使用如下的代码创建一个 `JavaObject` 对象来表示这个类的实例:

```
p = new java.awt.Polygon();
```

对象 `p` 具有属性 `xpoints` 和 `ypoints`, 它们都是 `JavaArray` 对象, 代表整型的 Java 数组。我们可以用如下的 JavaScript 代码初始化这两个数组的内容:

```
for(int i = 0; i < p.xpoints.length; i++)
    p.xpoints[i] = Math.round(Math.random()*100);
for(int i = 0; i < p.ypoints.length; i++)
    p.ypoints[i] = Math.round(Math.random()*100);
```

参阅

`getClass()`、`JavaClass`、`JavaObject`、`JavaPackage`, 第 12 章

JavaClass

LiveConnect

Java 类的 JavaScript 表示

摘要

```
javaclass.static_member    // Read or write a static Java field or method
new javaclass(...)         // Create a new Java object
```

属性

每个 `JavaClass` 对象含有的都是与它表示的 Java 类的公有静态字段和方法同名的属性。用这些属性可以读写那个类的静态字段并调用它的静态方法。每个 `JavaClass` 对象的属性都不同, 可以使用 `for/in` 循环来枚举一个给定的 `JavaClass` 对象的属性。

描述

`JavaClass` 对象是 Java 类的 JavaScript 表示。它的属性表示的是它所代表的类的公有静态字段和方法(有时称它们为类字段和类方法)。注意, `JavaClass` 对象没有表示 Java 类的 *instance* 字段的属性, 因为 Java 类的实例都是由 `JavaObject` 对象表示的。

`JavaClass` 对象实现了 `LiveConnect` 的功能, 这使得 JavaScript 程序可以使用常规的 JavaScript 语法对 Java 类的静态变量进行读写操作。它还提供了让 JavaScript 调用 Java 类的静态方法的功能。

除了允许 JavaScript 代码读写 Java 变量以及方法的值外, `JavaClass` 对象还允许 JavaScript 程序创建 Java 对象(由 `JavaObject` 对象表示), 只需要使用关键字 `new` 并调用 `JavaClass` 对象的构造函数即可。

JavaScript 和 Java 之间通过 `JavaClass` 对象通信必须进行数据转换，`LiveConnect` 会自动处理这一转换。参阅第 12 章的全面介绍。

习惯用法

要记住，Java 是一种强类型的语言。这就是说，对象的每个字段都要具有特定的数据类型，只能把它们设置成属于那种类型的值。如果设置字段时采用的数据类型不对，就会引发 JavaScript 错误或异常。调用方法时传递的参数类型不正确也会引发错误或异常。

例子

`java.lang.System` 是一个 `JavaClass` 对象，表示 Java 中的 `java.lang.System` 类。可以使用如下代码对这个类的静态字段进行读操作：

```
var java_console = java.lang.System.out;
```

也可以使用下面的代码调用这个类的静态方法：

```
var version = java.lang.System.getProperty("java.version");
```

最后，用 `JavaClass` 对象还能创建新的 Java 对象，代码如下：

```
var java_date = new java.lang.Date();
```

参阅

`getClass()`、`JavaArray`、`JavaObject`、`JavaPackage`，第 12 章

JavaObject

LiveConnect

Java 对象的 JavaScript 表示

摘要

`javaobject.member` // Read or write an instance field or method

属性

每个 `JavaObject` 对象的属性和方法都与它表示的 Java 对象的公有实例字段和方法（而不是静态的属性和方法，或者说类方法、类属性）同名。用这些属性可以读写那个类的公有字段并调用它的公有方法。一个给定的 `JavaObject` 对象的属性显然是由它所表示的 Java 对象的类型决定的。可以使用 `for/in` 循环枚举一个给定的 `JavaObject` 对象的属性。

描述

`JavaObject` 对象是 Java 对象的 JavaScript 表示。它的属性表示它所定义的对象公有实例字段和公有实例方法（对象的类或静态字段和对象方法由 `JavaClass` 的对象表示。）

`JavaObject` 对象实现了 `LiveConnect` 的功能，这使得 JavaScript 程序可以使用常规的 JavaScript 语法对 Java 对象的公有实例字段进行读写操作。此外，它还提供了让 JavaScript

调用 Java 对象的方法的功能。LiveConnect 会自动处理 JavaScript 表示和 Java 表示之间的数据转换。参阅第 12 章的全面介绍。

习惯用法

要记住, Java 是一种强类型的语言。这就是说, 对象的每个字段都要具有特定的数据类型, 只能把它们设置成属于那种类型的值。例如, `java.awt.Rectangle` 对象的 `width` 字段是一个整型字段, 如果把它设置成一个字符串就会引发 JavaScript 错误或异常。

例子

`java.awt.Rectangle` 是一个 `JavaClass` 对象, 它表示 `java.awt.Rectangle` 类。我们可以创建一个 `JavaObject` 对象来表示这个类的实例, 代码如下:

```
var r = new java.awt.Rectangle(0,0,4,5);
```

然后就可以使用如下代码对 `JavaObject` 对象 `r` 的公有实例变量进行读操作:

```
var perimeter = 2*r.width + 2*r.height;
```

还可以使用 JavaScript 语法来设置 `r` 的公有实例变量:

```
r.width = perimeter/4;  
r.height = perimeter/4;
```

参阅

`getClass()`、`JavaArray`、`JavaClass`、`JavaPackage`, 第 12 章

JavaPackage

LiveConnect

Java 包的 JavaScript 表示

摘要

```
package.package_name // Refers to another JavaPackage  
package.class_name   // Refers to a JavaClass object
```

属性

`JavaPackage` 对象的属性是它含有的 `JavaPackage` 对象和 `JavaClass` 对象的名称。每个 `JavaPackage` 对象的属性都不同。注意, 不能使用 JavaScript 的 `for/in` 循环遍历 `Package` 对象的属性名列表。要了解一个给定的包中含哪些包和类, 请参阅 Java 的参考手册。

描述

`JavaPackage` 对象是 Java 包的 JavaScript 表示。在 Java 中, 所谓包就是一组相关的类的集合。不过, 在 JavaScript 中, 一个 `JavaPackage` 对象除了可以含有类 (由 `JavaClass` 对象表示) 之外, 还可以含有其他的 `JavaPackage` 对象。

全局对象具有一个名称为 `java` 的 `JavaPackage` 属性, 表示 `java.*` 包层次。这个 `JavaPackage`

对象定义了引用其他 `JavaPackage` 对象的属性。例如，`java.lang` 和 `java.net` 引用 `java.lang` 和 `java.net` 包。`JavaPackage` 对象 `java.awt` 包含了 `Frame` 和 `Button` 属性，它们引用 `JavaClass` 对象，分别表示类 `java.awt.Frame` 和 `java.awt.Button`。

全局对象还定义了 `Packages` 属性，该属性是根 `JavaPackage` 对象，它的属性引用了所有已知包层次的根。例如，表达式 `Packages.javax.swing` 等价于 `Package javax.swing`。

用 `for/in` 循环不能确定一个 `JavaPackage` 对象含有的包名和类的名称。必须预先获得这些信息。在任何一本 Java 参考手册中都会有这些信息，你也可以自己查看 Java 的类层次。

第 12 章对 Java 包、类和使用进行了详细的介绍。

参阅

`java`、`JavaArray`、`JavaClass`、`JavaObject`、`Packages`，第 12 章

JSObject

参阅本书第四部分的 `JSObject`

Math

ECMAScript v1

算术函数和常量

摘要

`Math.constant`

`Math.function()`

常量

`Math.E`

常量 e ，自然对数的底数。

`Math.LN10`

10 的自然对数。

`Math.LN2`

2 的自然对数。

`Math.LOG10E`

以 10 为底的 e 的对数。

`Math.LOG2E`

以 2 为底的 e 的对数

`Math.PI`

常量 π 。

`Math.SQRT1_2`

2 的平方根除 1。

`Math.SQRT_2`
2 的平方根。

静态函数

`Math.abs()`
计算绝对值。

`Math.acos()`
计算反余弦值。

`Math.asin()`
计算反正弦值。

`Math.atan()`
计算反正切值。

`Math.atan2()`
计算从 X 轴到一个点的角度。

`Math.ceil()`
对一个数上舍入。

`Math.cos()`
计算余弦值。

`Math.exp()`
计算 e 的指数。

`Math.floor()`
对一个数下舍入。

`Math.log()`
计算自然对数。

`Math.max()`
返回两个数中较大的一个。

`Math.min()`
返回两个数中较小的一个。

`Math.pow()`
计算 x^y 。

`Math.random()`
计算一个随机数。

`Math.round()`
舍入为最接近的整数。

`Math.sin()`
计算正弦值。

`Math.sqrt()`

计算平方根。

`Math.tan()`

计算正切值。

描述

`Math`是一个对象，定义了引用有用的算术函数和常量的属性。使用如下的语法就可以调用它们：

```
y = Math.sin(x);  
area = radius * radius * Math.PI;
```

`Math`并不像 `Date` 和 `String` 那样是对象的类。没有 `Math()` 构造函数，像 `Math.sin()` 这样的函数只是函数，不是对象的方法。

参阅

`Number`

`Math.abs()`

ECMAScript v1

计算绝对值

摘要

`Math.abs(x)`

参数

`x` 任意数。

返回值

`x` 的绝对值。

`Math.acos()`

ECMAScript v1

计算反余弦值

摘要

`Math.acos(x)`

参数

`x` -1.0 和 1.0 之间的数。

返回值

指定的值 `x` 的反余弦值。返回的是 0 到 π 之间的弧度值。

Math.asin()ECMAScript v1

计算反正弦值

摘要`Math.asin(x)`**参数**

x -1.0 和 1.0 之间的数。

返回值

指定的值 *x* 的反正弦值。返回的是 $-\pi/2$ 到 $\pi/2$ 之间的弧度值。

Math.atan()ECMAScript v1

计算反正切值

摘要`Math.atan(x)`**参数**

x 任意数。

返回值

指定的值 *x* 的反正切值。返回的是 $-\pi/2$ 到 $\pi/2$ 之间的弧度值。

Math.atan2()ECMAScript v1

计算从 X 轴到一个点之间的角度

摘要`Math.atan2(y, x)`**参数**

y 指定点的 Y 坐标。

x 指定点的 X 坐标。

返回值

$-\pi$ 到 π 之间的值，是从 X 轴正向逆时针旋转到点 (*x*, *y*) 时经过的角度。

描述

函数 `Math.atan2()` 计算的是 y/x 的反正切值。参数 *y* 可以看作一个点的 Y 坐标，*x* 可以看作该点的 X 坐标。注意这个函数的参数顺序，Y 坐标在 X 坐标之前传递。

Math.ceil()ECMAScript v1

对一个数上舍入

摘要

`Math.ceil(x)`

参数

x 任意数或表达式。

返回值

大于等于 *x*，并且与它最接近的整数。

描述

`Math.ceil()` 执行的是向上取整计算，它返回的是大于或等于函数参数，并且与之最接近的整数。`Math.ceil()` 执行的操作不同于 `Math.round()`，`Math.ceil()` 总是上舍入，而 `Math.round()` 可以上舍入或下舍入到最接近的整数。还要注意，`Math.ceil()` 不会将负数舍入为更小的负数，而是向 0 舍入。

例子

```
a = Math.ceil(1.99);    // Result is 2.0
b = Math.ceil(1.01);    // Result is 2.0
c = Math.ceil(1.0);     // Result is 1.0
d = Math.ceil(-1.99);   // Result is -1.0
```

Math.cos()ECMAScript v1

计算余弦值

摘要

`Math.cos(x)`

参数

x 一个角的弧度值。要把角度转换成弧度，只需把角度值乘以 0.017453293 ($2\pi/360$) 即可。

返回值

指定的值 *x* 的余弦值。返回的是 -1.0 到 1.0 之间的数

Math.EECMAScript v1

算术常量 *e*

摘要

`Math.E`

描述

`Math.E` 代表算术常量 e ，即自然对数的底数，其值近似于 2.71828。

`Math.exp()`

ECMAScript v1

计算 e^x

摘要

`Math.exp(x)`

参数

x 数值或表达式，被用作指数。

返回值

e^x ，即 e 的 x 次幂。这里 e 代表自然对数的底数，其值近似为 2.71828。

`Math.floor()`

ECMAScript v1

对一个数下舍入

摘要

`Math.floor(x)`

参数

x 任意的数值或表达式。

返回值

小于等于 x ，并且与它最接近的整数。

描述

`Math.floor()` 执行的是向下取整计算，它返回的是小于等于函数参数，并且与之最接近的整数。

`Math.floor()` 将一个浮点值下舍入为最接近的整数。`Math.floor()` 执行的操作不同于 `Math.round()`，它总是进行下舍入，而不是上舍入或下舍入到最接近的整数。还要注意，`Math.floor()` 将负数舍入为更小的负数，而不是向 0 进行舍入。

例子

```
a = Math.floor(1.99);    // Result is 1.0
```

```
b = Math.floor(1.01);    // Result is 1.0
c = Math.floor(1.0);      // Result is 1.0
d = Math.floor(-1.01);    // Result is -2.0
```

Math.LN10

ECMAScript v1

算术常量 $\log_e 10$

摘要

Math.LN10

描述

Math.LN10 就是常量 $\log_e 10$ ，即 10 的自然对数，其值近似为 2.3025850929940459011。

Math.LN2

ECMAScript v1

算术常量 $\log_e 2$

摘要

Math.LN2

描述

Math.LN2 就是常量 $\log_e 2$ ，即 2 的自然对数，其值近似为 0.69314718055994528623。

Math.log()

ECMAScript v1

计算一个数的自然对数

摘要

Math.log(x)

参数

x 任何大于 0 的数值或表达式。

返回值

x 的自然对数。

描述

Math.log() 计算 $\log_e x$ 的值，即它的参数的自然对数。参数值必须大于 0。

可以使用下面的公式，计算一个以 10 为底的对数值和以 2 为底的对数值：

$$\begin{aligned}\log_{10} x &= \log_{10} e \cdot \log_e x \\ \log_2 x &= \log_2 e \cdot \log_e x\end{aligned}$$

这些公式可以转化成如下的 JavaScript 函数：

```
function log10(x) { return Math.LOG10E * Math.log(x); }  
function log2(x) { return Math.LOG2E * Math.log(x); }
```

Math.LOG10E

ECMAScript v1

算术常量 $\log_{10}e$

摘要

Math.LOG10E

描述

Math.LOG10E 表示常量 $\log_{10}e$ ，即以 10 为底 e 的对数。其值近似为 0.43429448190325181667。

Math.LOG2E

ECMAScript v1

算术常量 \log_2e

摘要

Math.LOG2E

描述

Math.LOG2E 表示常量 \log_2e ，即以 2 为底 e 的对数。其值近似为 1.442695040888963387。

Math.max()

ECMAScript v1; 在 ECMAScript v3 中增强

返回最大的参数

摘要

Math.max(*args...*)

参数

args...

0 个或多个值。在 ECMAScript v3 之前，该方法只有两个参数。

返回值

参数中最大的值。如果没有参数，返回 `-Infinity`。如果有一个参数为 NaN，或是不能转换成数字的非数字值，则返回 NaN。

Math.min()

ECMAScript v1; 在 ECMAScript v3 中增强

返回最小的参数

摘要

Math.min(*args...*)

参数

args...

0 个或多个值。在 ECMAScript v3 之前，该函数只有两个参数。

返回值

参数中最小的值。如果没有参数，返回 Infinity。如果有一个参数为 NaN，或是不能转换成数字的非数字值，则返回 NaN。

Math.PI

ECMAScript v1

算术常量 π

摘要

Math.PI

描述

Math.PI 表示的是常量 π ，即圆的周长和它的直径之比。这个值近似为 3.14159265358979。

Math.pow()

ECMAScript v1

计算 x^y

摘要

Math.pow(*x*, *y*)

参数

x 底数。

y 幂数。

返回值

x 的 *y* 次幂，即 x^y 。

描述

Math.pow() 计算 *x* 的 *y* 次幂。*x* 和 *y* 可以是任意值。但如果结果是虚数或复数，Math.pow() 将返回 NaN。在实际应用中，这就意味着如果 *x* 是一个负数，那么 *y* 就应该是一个正整数或是一个负整数。还要记住，指数过大会引起浮点溢出，此时该方法将返回 Infinity。

Math.random()

ECMAScript v1

返回一个伪随机数

摘要

`Math.random()`

返回值

0.0 到 1.0 之间的一个伪随机数。

Math.round()

ECMAScript v1

舍入到最接近的整数

摘要

`Math.round(x)`

参数

x 任意数。

返回值

与 *x* 最接近的整数。

描述

`Math.round()` 将把它的参数上舍入或下舍入到与它最接近的整数。对于 0.5, 它将上舍入。

例如, 2.5 将被舍入为 3, -2.5 将被舍入为 -2。

Math.sin()

ECMAScript v1

计算正弦值

摘要

`Math.sin(x)`

参数

x 一个以弧度表示的角。将角度乘以 0.017453293 ($2\pi/360$) 即可转换成弧度。

返回值

x 的正弦值。返回的值在 -1.0 到 1.0 之间。

Math.sqrt()

ECMAScript v1

计算平方根

摘要

`Math.sqrt(x)`

参数

x 大于等于 0 的数。

返回值

x 的平方根。如果 *x* 小于 0，返回 NaN。

描述

Math.sqrt() 计算数字的平方根。注意，用 Math.pow() 可以计算一个数的任意次根。例如：

```
Math.cuberoot = function(x){ return Math.pow(x,1/3); }  
Math.cuberoot(8); // Returns 2
```

Math.SQRT1_2

ECMAScript v1

算术常量 $1/\sqrt{2}$

摘要

Math.SQRT1_2

描述

Math.SQRT1_2 就是 $1/\sqrt{2}$ ，即 2 的平方根的倒数。它的值近似于 0.7071067811865476。

Math.SQRT2

ECMAScript v1

算术常量 $\sqrt{2}$

摘要

Math.SQRT2

描述

Math.SQRT2 就是常量 $\sqrt{2}$ ，即 2 的平方根。它的值近似于 1.414213562373095。

Math.tan()

ECMAScript v1

计算正切值

摘要

Math.tan(*x*)

参数

x 一个以弧度表示的角。将角度乘以 0.017453293 ($2\pi/360$) 即可转换成弧度。

返回值

指定的角 x 的正切值。

NaN

ECMAScript v1

非数字属性

摘要

NaN

描述

NaN 是全局属性，引用特殊的非数字值。用 `for/in` 循环不能枚举出 NaN 属性，用 `delete` 运算符不能删除它。注意，NaN 不是常量，可以用任意值设置它。

要判断一个值是否是数字，只能用 `isNaN()` 函数，因为 NaN 与所有值都不相等，包括它自己。

参阅

Infinity、`isNaN()`、`Number.NaN`

Number

ECMAScript v1

对数字的支持

Object → Number

构造函数

`new Number(value)`

`Number(value)`

参数

value

要创建的 Number 对象的数值，或是要转换成数字的值。

返回值

当 `Number()` 和运算符 `new` 一起作为构造函数使用时，它返回一个新创建的 Number 对象。如果不用 `new` 运算符，把 `Number()` 作为一个函数来调用，它将把自己的参数转换成一个原始的数值，并且返回这个值（如果转换失败，返回 NaN）。

常量

`Number.MAX_VALUE`

可表示的最大的数。

`Number.MIN_VALUE`

可表示的最小的数。

`Number.NaN`

非数字值。

`Number.NEGATIVE_INFINITY`

负无穷大；溢出时返回该值。

`Number.POSITIVE_INFINITY`

正无穷大；溢出时返回该值。

方法

`toString()`

把数字转换成字符串，使用指定的基数。

`toLocaleString()`

把数字转换成字符串，使用本地数字格式规则。

`toFixed()`

把数字转换成字符串，结果的小数点后有指定位数的数字。

`toExponential()`

把数字转换成字符串，结果采用指数计数法，小数点后有指定位数的数字。

`toPrecision()`

把数字转换成字符串，结果中包含指定位数的有效数字。采用指数计数法或定点计数法，由数字的大小和指定的有效数字位数决定采用哪种方法。

`valueOf()`

返回一个 `Number` 对象的基本数字值。

描述

在 JavaScript 中，数字是一种基本的基本数据类型。JavaScript 还支持 `Number` 对象，该对象是原始数值的包装对象。JavaScript 在必要时会自动地进行原始数据和对象之间的转换。在 JavaScript 1.1 中，可以用构造函数 `Number()` 明确地创建一个 `Number` 对象，尽管这样做并没有什么必要。

构造函数 `Number()` 还可以不与运算符 `new` 一起使用，而直接作为转换函数来使用。以这种方式调用 `Number()` 时，它会把自己的参数转换成一个数字，然后返回转换后的原始数值（或 `NaN`）。

构造函数 `Number()` 通常还用作 5 个有用的数字常量的占位符，这 5 个有用的数字常量分别是可表示的最大的数、可表示的最小的数、正无穷大、负无穷大和特殊的 `NaN` 值。注意，这些值都是构造函数 `Number()` 自身的属性，而不是单独的 `Number` 对象的属性。例如，可以采用如下的形式使用属性 `MAX_VALUE`：

```
var biggest = Number.MAX_VALUE
```

但是却不能使用：

```
var n = new Number(2);  
var biggest = n.MAX_VALUE
```

作为比较，我们看一下 `toString()` 和 `Number` 对象的其他方法，它们是每个 `Number` 对象的方法，而不是 `Number()` 构造函数的方法。前面提到过，在必要时，JavaScript 会自动地把原始数值转换成 `Number` 对象。这就是说，调用 `Number` 方法的既可以是 `Number` 对象，也可以是原始数字值。

```
var value = 1234;  
var binary_value = n.toString(2);
```

参阅

`Infinity`、`Math` 和 `NaN`

Number.MAX_VALUE

ECMAScript v1

最大数值

摘要

`Number.MAX_VALUE`

描述

`Number.MAX_VALUE` 是 JavaScript 中可表示的最大的数。它的值近似为 $1.79E+308$ 。

Number.MIN_VALUE

ECMAScript v1

最小数值

摘要

`Number.MIN_VALUE`

描述

`Number.MIN_VALUE` 是 JavaScript 中可表示的最小的数（接近 0，但不是负数）。它的值近似为 $5E - 324$ 。

Number.NaN

ECMAScript v1

特殊的非数字值

摘要

`Number.NaN`

描述

`Number.NaN` 是一个特殊值，说明某些算术运算（如求负数的平方根）的结果不是数字。方法 `parseInt()` 和 `parseFloat()` 在不能解析指定的字符串时就返回这个值。对于一些常规情况下返回有效数字的函数，也可以采用这种方法，用 `Number.NaN` 说明它的错误情况。

- JavaScript以NaN的形式输出`Number.NaN`。注意，NaN与其他数值进行比较的结果总是不相等的，包括它自身在内。因此，不能通过与`Number.NaN`比较来检测一个值是不是数字，而只能调用函数`isNaN()`来比较。在ECMAScript v1和其后的版本中，还可以用预定义的全局属性NaN代替`Number.NaN`。

参阅

`isNaN()`、NaN

Number.NEGATIVE_INFINITY

ECMAScript v1

负无穷大

摘要

`Number.NEGATIVE_INFINITY`

描述

`Number.NEGATIVE_INFINITY`是一个特殊值，它在算术运算或函数生成了一个比JavaScript能表示的最小负数还小的数（也就是比`-Number.MAX_VALUE`还小的数）时返回。

JavaScript显示`NEGATIVE_INFINITY`时使用的是`-Infinity`。这个值的算术行为和无穷大非常相似。例如，任何数乘无穷大结果仍为无穷大，任何数被无穷大除的结果为0。在ECMAScript v1和其后的版本中，还可以用`-Infinity`代替`Number.NEGATIVE_INFINITY`。

参阅

`Infinity`、`isFinite()`

Number.POSITIVE_INFINITY

ECMAScript v1

正无穷大

摘要

`Number.POSITIVE_INFINITY`

描述

属性`Number.POSITIVE_INFINITY`是一个特殊值，它在算术运算或函数生成了一个比JavaScript能表示的最大的数还大的数（也就是比`Number.MAX_VALUE`还大的数）时返回。注意，当数字向下溢出或比`Number.MIN_VALUE`还小时，JavaScript将它们转换成零。

JavaScript显示`POSITIVE_INFINITY`时使用的是`Infinity`。这个值的算术行为和无穷大非常相似。例如，任何数乘无穷大结果仍为无穷大，任何数被无穷大除结果为0。在ECMAScript v1和其后的版本中，还可以用`Infinity`代替`Number.POSITIVE_INFINITY`。

参阅

Infinity 和 isFinite()

Number.toExponential()

ECMAScript v3

用指数计数法格式化数字

摘要

number.toExponential(digits)

参数

digits

小数点后的数字位数，值在 0~20 之间，包括 0 和 20，有些实现可能支持更大的数值范围。如果省略了该参数，将使用尽可能多的数字。

返回值

number 的字符串表示，采用指数计数法，即小数点之前有一位数字，小数点后有 *digits* 位数字。该数字的小数部分将被舍入，必要时用 0 补足，以便它达到指定的长度。

抛出

RangeError

digits 太小或太大时抛出的异常。0~20 之间（包括 0 和 20）的值不会引发 RangeError。有些实现允许支持更大范围或更小范围内的值。

TypeError

调用该方法的对象不是 Number 时抛出的异常。

例子

```
var n = 12345.6789;
n.toExponential(1);    // Returns 1.2e+4
n.toExponential(5);    // Returns 1.23457e+4
n.toExponential(10);   // Returns 1.2345678900e+4
n.toExponential();     // Returns 1.23456789e+4
```

参阅

Number.toFixed(), Number.toLocaleString(), Number.toPrecision(), Number.toString()

Number.toFixed()

ECMAScript v3

采用定点计数法格式化数字

摘要

number.toFixed(digits)

参数

digits

小数点后的数字位数，是 0~20 之间的值，包括 0 和 20，有些实现可以支持更大的数值范围。如果省略了该参数，将用 0 代理。

返回值

number 的字符串表示，不采用指数计数法，小数点后有固定的 *digits* 位数字。如果需要，该数字会被舍入，也可以用 0 补足，以便它达到指定的长度。如果 *number* 大于 $1e+21$ ，该方法只调用 `Number.toString()`，返回采用指数计数法表示的字符串。

抛出

`RangeError`

digits 太小或太大时抛出的异常。0~20 之间（包括 0 和 20）的值不会引发 `RangeError`。有些实现支持更大范围或更小范围内的值。

`TypeError`

调用该方法的对象不是 `Number` 时抛出的异常。

例子

```
var n = 12345.6789;
n.toFixed();           // Returns 12346: note rounding, no fractional part
n.toFixed(1);          // Returns 12345.7: note rounding
n.toFixed(6);          // Returns 12345.678900: note added zeros
(1.23e+20).toFixed(2); // Returns 123000000000000000000.00
(1.23e-10).toFixed(2)  // Returns 0.00
```

参阅

`Number.toExponential()`、`Number.toLocaleString()`、`Number.toPrecision()`、`Number.toString()`

`Number.toLocaleString()`

ECMAScript v3

把数字转换成本地格式的字符串

摘要

`number.toLocaleString()`

返回值

数字的字符串表示，由实现决定，根据本地规范进行格式化，可能影响到小数点或千分位分隔符采用的标点符号。

抛出

TypeError

调用该方法的对象不是 `Number` 时抛出的异常。

参阅

`Number.toExponential()`、`Number.toFixed()`、`Number.toPrecision()`、`Number.toString()`

`Number.toPrecision()`

ECMAScript v3

格式化数字的有效位

摘要

`number.toPrecision(precision)`

参数

precision

返回的字符串中的有效位数，是 1~21 之间（包括 1 和 21）的值。有些实现允许有选择地支持更大或更小的 *precision*。如果省略了该参数，将调用方法 `toString()`，而不是把数字转换成十进制的值。

返回值

number 的字符串表示，包含 *precision* 个有效数字。如果 *precision* 足够大，能够包括 *number* 整数部分的所有数字，那么返回的字符串将采用定点计数法。否则，采用指数计数法，即小数点前有一位数字，小数点后有 *precision* - 1 位数字。必要时，该数字会被舍入或用 0 补足。

抛出

RangeError

digits 太小或太大时抛出的异常。1~21 之间（包括 1 和 21）的值不会引发 `RangeError`。有些实现支持更大范围或更小范围内的值。

TypeError

调用该方法的对象不是 `Number` 时抛出的异常。

例子

```
var n = 12345.6789;
n.toPrecision(1);    // Returns 1e+4
n.toPrecision(3);    // Returns 1.23e+4
n.toPrecision(5);    // Returns 12346: note rounding
n.toPrecision(10);   // Returns 12345.67890: note added zero
```


参阅

`Number.toExponential()`、`Number.toFixed()`、`Number.toLocaleString()`、`Number.toString()`

Number.toString()

ECMAScript v1

将一个数字转换成字符串

覆盖 `Object.toString()`

摘要

`number.toString(radix)`

参数

radix

选用的参数，指定表示数字的基数，是 2~36 之间的整数。如果省略了该参数，使用基数 10。但要注意，如果该参数是 10 以外的其他值，则 ECMAScript 标准允许实现返回任意值。

返回值

数字的字符串表示。

抛出

`TypeError`

调用该方法的对象不是 `Number` 时抛出的异常。

描述

`Number` 对象的方法 `toString()` 可以将数字换成字符串。当省略了 *radix* 参数或指定它的值为 10 时，该数字将被转换成基数为 10 的字符串。尽管 ECMAScript 规范并不要求实现为 *radix* 保证任何其他值，但是所有的实现通常都接受 2 到 36 之间的值。

参阅

`Number.toExponential()`、`Number.toFixed()`、`Number.toLocaleString()`、`Number.toPrecision()`

Number.valueOf()

ECMAScript v1

返回原始数值

覆盖 `Object.valueOf()`

摘要

`number.valueOf()`

返回值

`Number` 对象的原始数值。几乎没有必要明确调用该方法。

抛出

TypeError

调用该方法的对象不是 Number 时抛出的异常。

参阅

`Object.valueOf()`

Object

ECMAScript v1

含有所有 JavaScript 对象的特性的超类

构造函数

`new Object()`

`new Object(value)`

参数

`value`

选用的参数，声明了要转换成 Number 对象、Boolean 对象或 String 对象的原始值（即数字、布尔值或字符串）。

返回值

如果没有给构造函数传递 `value` 参数，那么它将返回一个新创建的 Object 实例。如果指定了原始的 `value` 参数，构造函数将创建并返回原始值的包装对象，即 Number 对象、Boolean 对象或 String 对象。当不使用 `new` 运算符，而将 `Object()` 构造函数作为函数调用时，它的行为与使用 `new` 运算符时一样。

属性

`constructor`

对一个 JavaScript 函数的引用，该函数是对象的构造函数。

方法

`hasOwnProperty()`

检查对象是否有局部定义的（非继承的）、具有特定名称的属性。

`isPrototypeOf()`

检查对象是否是指定对象的原型。

`propertyIsEnumerable()`

检查指定的属性是否存在，以及是否能用 `for/in` 循环枚举。

`toLocaleString()`

返回对象地方化的字符串表示。该方法的默认实现只调用 `toString()` 方法，但子类可以覆盖它，提供本地化。

toString()

返回对象的字符串表示。Object 类提供的该方法的实现相当普通，并且没有提供更多有用的信息。Object 的子类通过定义自己的 toString() 方法覆盖了这一方法 (toString() 方法能够生成更有用的结果)。

valueOf()

返回对象的原始值 (如果存在)。对于类型为 Object 的对象，该方法只返回对象自身。Object 的子类 (如 Number 和 Boolean) 覆盖了该方法，返回的是与对象相关的原始数值。

描述

Object 类是 JavaScript 语言的内部数据类型。它是其他 JavaScript 对象的超类，因此其他对象都继承了 Object 类的方法和行为。第 7 章对 JavaScript 中的对象的基本行为进行了解释。除了用上面所示的 Object() 构造函数，还可以用第 7 章介绍的 Object 直接量语法创建并初始化对象。

参阅

Array、Boolean、Function、Function.prototype、Number、String，第 7 章

Object.constructor

ECMAScript v1

对象的构造函数

摘要

object.constructor

描述

对象的 constructor 属性引用了该对象的构造函数。例如，如果用 Array() 构造函数创建一个数组，那么 a.constructor 引用的就是 Array：

```
a = new Array(1,2,3);    // Create an object
a.constructor == Array   // Evaluates to true
```

constructor 属性常用于判断未知对象的类型。给定了一个未知的值，就可以使用 typeof 运算符来判断它是原始的值还是对象。如果它是对象，就可以使用 constructor 属性来判断对象的类型。例如，下面的函数用来判断一个给定的值是否是数组：

```
function isArray(x) {
    return ((typeof x == "object") && (x.constructor == Array));
}
```

但是要注意，虽然这种方法适用于 JavaScript 核心语言的内部对象，但对于“主对象”，如 Window 这样的客户端 JavaScript 对象，这种方法就不一定适用了。Object.toString() 方法的默认实现提供了另一种判断未知对象类型的方法。

参阅

`Object.toString()`

`Object.hasOwnProperty()`

ECMAScript v3

检查属性是否被继承

摘要

`object.hasOwnProperty(propname)`

参数

propname

一个字符串，包含 *object* 的属性名。

返回值

如果 *object* 有 *propname* 指定的非继承属性，则返回 `true`。如果 *object* 没有名为 *propname* 指定的属性，或者它从原型对象继承了这一属性，则返回 `false`。

描述

在第 9 章解释过，JavaScript 对象既可以有自己的属性，又可以从原型对象继承属性。`hasOwnProperty()` 方法提供了区分继承属性和非继承的局部属性的方法。

例子

```
var o = new Object();           // Create an object
o.x = 3.14;                     // Define a noninherited local property
o.hasOwnProperty("x");          // Returns true: x is a local property of o
o.hasOwnProperty("y");          // Returns false: o doesn't have a property y
o.hasOwnProperty("toString");   // Returns false: toString property is inherited
```

参阅

`Function.prototype`、`Object.propertyIsEnumerable()`，第 9 章

`Object.isPrototypeOf()`

ECMAScript v3

一个对象是否是另一个对象的原型

摘要

`object.isPrototypeOf(o)`

参数

o 任意对象。

返回值

如果 *object* 是 *o* 的原型，则返回 `true`。如果 *o* 不是对象，或者 *object* 不是 *o* 的原型，则返回 `false`。

描述

在第9章解释过，JavaScript对象继承了原型对象的属性。一个对象的原型是通过用于创建并初始化该对象的构造函数的 `prototype` 属性引用的。`isPrototypeOf()` 方法提供了判断一个对象是否是另一个对象原型的方法。该方法可以用于确定对象的类。

例子

```
var o = new Object(); // Create an object
Object.prototype.isPrototypeOf(o) // true: o is an object
Function.prototype.isPrototypeOf(o.toString); // true: toString is a function
Array.prototype.isPrototypeOf([1,2,3]); // true: [1,2,3] is an array
// Here is a way to perform a similar test
(o.constructor == Object); // true: o was created with Object() constructor
(o.toString.constructor == Function); // true: o.toString is a function
// Prototype objects themselves have prototypes. The following call
// returns true, showing that function objects inherit properties
// from Function.prototype and also from Object.prototype.
Object.prototype.isPrototypeOf(Function.prototype);
```

参阅

`Function.prototype`、`Object.constructor`，第9章

`Object.prototype.isEnumerable()`

ECMAScript v3

是否可以通过 `for/in` 循环看到属性

摘要

`object.propertyIsEnumerable(propname)`

参数

propname

一个字符串，包含 *object* 属性的名称。

返回值

如果 *object* 具有名为 *propname* 的非继承属性，而且该属性是可枚举的（即用 `for/in` 循环可以枚举出它），则返回 `true`。

描述

用 `for/in` 语句可以遍历一个对象“可枚举”的属性。但并非一个对象的所有属性都是可枚举的，通过JavaScript代码添加到对象的属性是可枚举的，而内部对象的预定义属性（如

方法) 通常是不可枚举的。propertyIsEnumerable() 方法提供了区分可枚举属性和不可枚举属性的方法。但要注意, ECMAScript 标准规定, propertyIsEnumerable() 方法不检测原型链, 这意味着它只适用于对象的局部属性, 不能检测继承属性的可枚举性。

例子

```
var o = new Object();           // Create an object
o.x = 3.14;                     // Define a property
o.propertyIsEnumerable("x");    // true: property x is local and enumerable
o.propertyIsEnumerable("y");    // false: o doesn't have a property y
o.propertyIsEnumerable("toString"); // false: toString property is inherited
Object.prototype.propertyIsEnumerable("toString"); // false: nonenumerable
```

参阅

Function.prototype、Object.hasOwnProperty(), 第 7 章

Object.toLocaleString()

ECMAScript v3

返回对象的本地字符串表示

摘要

object.toString()

返回值

表示对象的字符串。

描述

该方法用于返回对象的字符串表示, 本地化为适合本地的形式。Object 类提供的默认的 toLocaleString() 方法只调用 toString() 方法, 返回非本地化的字符串。但其他类 (包括 Array、Date 和 Number) 定义了自己的 toLocaleString() 版本, 指定本地化字符串的转换。在定义自己的类时, 也可以覆盖该方法。

参阅

Array.toLocaleString()、Date.toLocaleString()、Number.toLocaleString()、Object.toString()

Object.toString()

ECMAScript v1

定义一个对象的字符串表示

摘要

object.toString()

返回值

表示对象的字符串。

描述

这里的方法 `toString()` 并不是你在 JavaScript 程序中经常显式调用的那个 `toString()` 方法。它是你在对象中定义的一个方法，当系统需要把对象转换成字符串时就会调用它。当在字符串环境中使用对象时，JavaScript 系统就会调用 `toString()` 方法把那个对象转换成字符串。例如，假定一个函数期望得到的参数是字符串，那么把对象传递给它时，系统就会将这个对象转换成字符串：

```
alert(my_object);
```

同样，在用运算符 “+” 连接字符串时，对象也会被转换成字符串：

```
var msg = 'My object is: ' + my_object;
```

调用方法 `toString()` 时不给它传递任何参数，它返回的应该是一个字符串。要使这个字符串有用，它的值就必须以调用 `toString()` 方法的对象的值为基础。

当用 JavaScript 自定义一个类时，为这个类定义一个 `toString()` 方法很有用。如果没有给它定义 `toString()` 方法，那么这个对象将继承 `Object` 类的默认 `toString()` 方法。这个方法返回的字符串形式如下：

```
[object class]
```

这里，`class` 是一个对象类，其值可以是 “Object”、“String”、“Number”、“Function”、“Window”、“Document”，等等。这种行为有时对确定未知对象的类型或类有用。但由于大多数对象都有自定义的 `toString()` 版本，所以必须明确地对对象 `o` 调用 `Object.toString()` 方法，代码如下所示：

```
Object.prototype.toString.apply(o);
```

注意，这种识别未知对象的方法只适用于内部对象。如果你定义了自己的对象类，那么它的类是 “Object”。在这种情况下，可以用 `Object.constructor` 属性获取更多有关对象的信息。

在调试 JavaScript 程序时，`toString()` 方法非常有用，使用它可以输出对象，查看它们的值。因此，为你创建的每个对象类都定义 `toString()` 方法很有用。

虽然 `toString()` 方法通常由系统自动调用，但你也可以自己调用它。例如，在 JavaScript 不能自动把对象转换成字符串的环境中，可以显式地调用 `toString()` 方法来实现这一点：

```
y = Math.sqrt(x);          // Compute a number
ystr = y.toString();        // Convert it to a string
```

注意，在这个例子中，数字具有内部的 `toString()` 方法，可以用该方法进行强制性的转换。在其他的环境中，即使 JavaScript 可以自动地进行转换，你也可以调用 `toString()` 方法，因为对 `toString()` 的显式调用可以使代码更加清晰：

```
alert(my_obj.toString());
```

参阅

`Object.constructor`、`Object.toLocaleString()`、`Object.valueOf()`

Object.valueOf()

ECMAScript v1

指定对象的原始值

摘要

`object.valueOf()`

返回值

与对象 *object* 相关的原始值（如果存在）。如果没有值与 *object* 相关，则返回对象自身。

描述

对象的 `valueOf()` 方法返回的是与那个对象相关的原始值（如果这样的值存在）。对于类型为 `Object` 的对象来说，由于它们没有原始值，因此该方法返回的是这些对象自身。

对于类型为 `Number` 的对象，`valueOf()` 返回该对象表示的原始数值。同样，对于 `Boolean` 对象来说，该方法返回与对象相关的布尔值。对于 `String` 对象来说，返回与对象相关的字符串。

其实，几乎没有必要自己调用 `valueOf()` 方法。在期望使用原始值的地方，JavaScript 会自动地执行转换。事实上，由于方法 `valueOf()` 是被自动调用的，因此要分辨究竟是原始值还是与之相应的对象非常困难。虽然使用 `typeof` 运算符可以显示字符串和 `String` 对象之间的区别，但在实际应用中，它们在 JavaScript 代码中的作用是一样的。

`Number` 对象、`Boolean` 对象和 `String` 对象的 `valueOf()` 方法可以将这些包装对象转换成它们表示的原始值。在调用构造函数 `Object()` 时，如果把数字、布尔值或字符串作为参数传递给它，它将执行相反的操作，即将原始值打包成相应的对象。几乎在所有的环境中，JavaScript 都可以自动地实现原始值和对象之间的转换，所以一般说来，没有必要用这种方法调用构造函数 `Object()`。

在某些环境中，你可以为自己的对象定制一个 `valueOf()` 方法。例如，你可以定义一个 JavaScript 对象来表示复数（即一个实数加一个虚数）。作为这个对象的一部分，要给它定义执行复数的加法、乘法等其他运算的方法。不过，还有一种功能是你想要的，即像处理常规实数一样处理复数，舍弃它的虚数部分。可以使用下面的代码实现这一点：

```
Complex.prototype.valueOf = new Function("return this.real");
```

有了这个为 `Complex` 对象定义的 `valueOf()` 方法，就可以把复数对象传递给方法 `Math.sqrt()`，它将计算复数的实数部分的平方根。

参阅

`Object.toString()`

Packages

LiveConnect

根 JavaPackage

摘要

Packages

描述

在包含了 LiveConnect 或其他脚本化 Java 的技术的 JavaScript 实现中，全局 Packages 属性是一个 JavaPackage 对象，其属性是 Java 解释器所知道的所有包的根。例如，Packages.javax.swing 引用的是 Java 包 *javax.swing*。全局属性 *java* 是的 Packages.java 简写。

参阅

java、JavaPackage，第 12 章

parseFloat()

ECMAScript v1

把字符串转换成数字

摘要

parseFloat(s)

参数

s 要被解析并转换成数字的字符串。

返回值

解析后的数字，如果字符串 *s* 没有以一个有效的数字开头，则返回 NaN。在 JavaScript 1.0 中，当 *s* 不能被解析成数字时，parseFloat() 返回的是 0 而不是 NaN。

描述

方法 parseFloat() 将对字符串 *s* 进行解析，返回出现在 *s* 中的第一个数字。当 parseFloat() 在 *s* 中遇到了一个不是有效数字的字符时，解析过程就停止了，解析的结果也将在此时返回。如果 *s* 的开头是一个 parseFloat() 不能解析的数字，该函数将返回 NaN。可以用函数 isNaN() 来检测这个值。如果只想解析数字的整数部分，则使用 parseInt() 方法而不是 parseFloat() 方法。

参阅

isNaN()、parseInt()

parseInt()

ECMAScript v1

把字符串转换成整数

摘要

```
parseInt(s)
parseInt(s, radix)
```

参数

s 要被解析的字符串。

radix

选用的整数参数，表示要解析的数字的基数。如果省略了该参数或者它的值为 0，数字将以 10 为基数来解析。如果它以 “0x” 或 “0X” 开头，则以 16 为基数。如果该参数小于 2 或大于 36，则 `parseInt()` 返回 NaN。

返回值

解析后的数字，如果字符串 *s* 不是以一个有效的整数开头，则返回 NaN。在 JavaScript 1.0 中，当 `parseInt()` 不能解析 *s* 时，它返回的是 0 而不是 NaN。

描述

方法 `parseInt()` 将对字符串 *s* 进行解析，并且返回出现在 *s* 中的第一个数字（可以具有减号）。当 `parseInt()` 在 *s* 中遇到的字符不是指定的基数 *radix* 可以使用的有效数字时，解析过程就停止，解析的结果也将在此时返回。如果 *s* 的开头是 `parseInt()` 不能解析的数字，该函数将返回 NaN。可以用函数 `isNaN()` 来检测这个值。

参数 *radix* 指定的是要解析成的数字的基数。如果将它设置为 10，`parseInt()` 就会将字符串解析成十进制的数。将它设置为 8，那么解析的结果就是八进制（使用 0~7 八个数字）的数。将它设置为 16，解析的结果就是十六进制（使用数字 0~9 和字母 A~F 表示）的值。*radix* 的值可以是 2~36 之间的任意一个整数。

如果 *radix* 的值为 0，或者没有设置 *radix* 的值，那么 `parseInt()` 将根据字符串 *s* 来判断数字的基数。如果 *s*（在可选的减号后）以 0x 开头，那么 `parseInt()` 将把 *s* 的其余部分解析成十六进制的整数。如果 *s* 以 0 开头，那么 ECMAScript v3 标准允许 `parseInt()` 的一个实现把其后的字符解析成八进制的数字或十进制的数字。如果 *s* 以 1~9 之间的数字开头，`parseInt()` 将把它解析成十进制的整数。

例子

```
parseInt("19", 10); // Returns 19   (10 + 9)
parseInt("11", 2);  // Returns 3    (2 + 1)
parseInt("17", 8);  // Returns 15   (8 + 7)
parseInt("1f", 16); // Returns 31   (16 + 15)
parseInt("10");     // Returns 10
```

```
parseInt("0x10");    // Returns 16
parseInt("010");      // Ambiguous: returns 10 or 8
```

Bug

在没有指定 *radix* 时，ECMAScript v3 允许实现将以 “0”（但不是 “0x” 或 “0X”）开头的字符串解析为八进制或十进制的数。要避免这种二义性，应该明确指定基数，或只有确定所有要解析的数字都是以 “0x” 或 “0X” 开头的十进制数或十六进制数时，才可以不指定基数。

参阅

`isNaN()`、`parseFloat()`

RangeError

ECMAScript v3

在数字超出合法范围时抛出

Object → Error → RangeError

构造函数

```
new RangeError()
new RangeError(message)
```

参数

message

可选的出错消息，提供异常的详细情况。如果指定了该参数，它将作为 RangeError 对象的 *message* 属性的值。

返回值

新构造的 RangeError 对象。如果指定了参数 *message*，该 Error 对象把它作为 *message* 属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果不用 `new` 运算符，把 `RangeError()` 构造函数当作函数调用，那么它的行为与使用 `new` 运算符调用时一样。

属性

message

提供异常细节的出错消息。该属性存放传递给构造函数的字符串，或实现定义的默认字符串。详见 “Error.message”。

name

声明异常类型的字符串。所有 RangeError 对象的 *name* 属性都继承值 “RangeError”。

描述

当数字超出合法范围时，RangeError 类的一个实例就会被抛出。例如，把数组的 *length* 属性设置成一个负数，就会使 RangeError 对象被抛出。关于抛出和捕捉异常的细节，请参阅 “Error”。

参阅

Error、Error.message、Error.name

ReferenceError

ECMAScript v3

在读取不存在的变量时抛出

Object → Error → ReferenceError

构造函数

new ReferenceError()

new ReferenceError(message)

参数

message

可选的出错消息，提供异常的详细情况。如果指定了该参数，它将作为ReferenceError对象的message属性的值。

返回值

新构造的ReferenceError对象。如果指定了参数message，该Error对象把它作为message属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果不用new运算符，把ReferenceError()构造函数当作函数调用，它的行为与使用new运算符调用时一样。

属性

message

提供异常细节的出错消息。该属性存放传递给构造函数的字符串，或实现定义的默认字符串。详见“Error.message”。

name

声明异常类型的字符串。所有ReferenceError对象的name属性都继承值“ReferenceError”。

描述

在读取一个不存在的变量的值时，ReferenceError类的一个实例就会抛出。关于抛出和捕捉异常的细节，请参阅“Error”。

参阅

Error、Error.message、Error.name

RegExp

ECMAScript v3

用于模式匹配的正则表达式

Object → RegExp

直接量语法

/pattern/attributes

构造函数

`new RegExp(pattern, attributes)`

参数

pattern

一个字符串，指定了正则表达式的模式或其他正则表达式。

attributes

一个可选的字符串，包含属性“g”、“i”和“m”，分别用于指定全局匹配、区分大小写的匹配和多行匹配。ECMAScript 标准化之前，不支持 m 属性。如果 *pattern* 是正则表达式，而不是字符串，则必须省略该参数。

返回值

一个新的 `RegExp` 对象，具有指定的模式和标志。如果参数 *pattern* 是正则表达式而不是字符串，那么 `RegExp()` 构造函数将用与指定的 `RegExp` 相同的模式和标志创建一个新的 `RegExp` 对象。如果不用 `new` 运算符，而将 `RegExp()` 作为函数调用，那么它的行为与用 `new` 运算符调用时一样，只是当 *pattern* 是正则表达式时，它只返回 *pattern*，而不再创建一个新的 `RegExp` 对象。

抛出

`SyntaxError`

如果 *pattern* 不是合法的正则表达式，或 *attributes* 含有“g”、“i”、“m”之外的字符，抛出该异常。

`TypeError`

如果 *pattern* 是 `RegExp` 对象，但没有省略 *attributes* 参数，抛出该异常。

实例属性

`global`

`RegExp` 对象是否具有性质 `g`。

`ignoreCase`

`RegExp` 对象是否具有性质 `i`。

`lastIndex`

上次匹配后的字符位置，用于在一个字符串中进行多次匹配。

`multiline`

`RegExp` 对象是否具有 `m` 性质。

`source`

正则表达式的源文本。

方法

`exec()`

执行强大的、通用的模式匹配。

`test()`

检测一个字符串是否含有某个模式。

描述

RegExp对象表示一个正则表达式，它是对字符串执行模式匹配的强大工具。要了解正则表达式的语法和用法，请参阅第 11 章。

参阅

第 11 章

RegExp.exec()

ECMAScript v3

通用的匹配模式

摘要

`regexp.exec(string)`

参数

string

要检索的字符串。

返回值

一个数组，存放的是匹配的结果。如果没有找到匹配，则值为 `null`。返回的数组的格式在下面介绍。

抛出

`TypeError`

调用该方法的对象不是 `RegExp` 时，抛出该异常。

描述

在所有的 `RegExp` 模式匹配方法和 `String` 模式匹配方法中，`exec()` 的功能最强大。它是一个通用的方法，使用起来比 `RegExp.test()`、`String.search()`、`String.replace()` 和 `String.match()` 都复杂。

`exec()` 将检索字符串 *string*，从中得到与正则表达式 *regexp* 相匹配的文本。如果 `exec()` 找到了匹配的文本，它就会返回一个结果数组。否则，返回 `null`。这个返回数组的第 0 个元素就是与表达式相匹配的文本。第 1 个元素是与 *regexp* 的第一个子表达式相

匹配的文本（如果存在）。第2个元素是与 *regexp* 的第二个子表达式相匹配的文本，依此类推。通常，数组的 *length* 属性声明的是数组中的元素个数。除了数组元素和 *length* 属性之外，*exec()* 还返回两个属性。*index* 属性声明的是匹配文本的第一个字符的位置。*input* 属性指的就是 *string*。在调用非全局 *RegExp* 对象的 *exec()* 方法时，返回的数组与调用方法 *String.match()* 返回的方法相同。

在调用非全局模式的 *exec()* 方法时，它将进行检索，并返回上述结果。不过，当 *regexp* 是一个全局正则表达式时，*exec()* 的行为就稍微复杂一些。它在 *regexp* 的属性 *lastIndex* 指定的字符处开始检索字符串 *string*。当它找到了与表达式相匹配的文本时，在匹配之后，它将把 *regexp* 的 *lastIndex* 属性设置为匹配文本的第一个字符的位置。这就是说，可以通过反复地调用 *exec()* 方法来遍历字符串中的所有匹配文本。当 *exec()* 再也找不到匹配的文本时，它将返回 *null*，并且把属性 *lastIndex* 重置为 0。如果在另一个字符串中完成了一次模式匹配之后要开始检索新的字符串，就必须手动地把 *lastIndex* 属性重置为 0。

注意，无论 *regexp* 是否是全局模式，*exec()* 都会将完整的细节添加到它返回的数组中。这就是 *exec()* 和 *String.match()* 的不同之处，后者在全局模式下返回的信息要少得多。事实上，在循环中反复地调用 *exec()* 方法是惟一种获得全局模式的完整模式匹配信息的方法。

例子

可以在循环中使用 *exec()* 来检索一个字符串中的所有匹配文本。例如：

```
var pattern = /\bJava\b/g;
var text = "JavaScript is more fun than Java or JavaBeans!";
var result;
while((result = pattern.exec(text)) != null) {
    alert("Matched '" + result[0] +
        "' at position " + result.index +
        " next search begins at position " + pattern.lastIndex);
}
```

参阅

RegExp.lastIndex、*RegExp.test()*、*String.match()*、*String.replace()*、*String.search()*，第 11 章

RegExp.global

ECMAScript v3

正则表达式是否全局匹配

摘要

regexp.global

描述

RegExp对象的属性`global`是一个只读的布尔值。它声明了给定的正则表达式是否执行全局匹配，如创建它时是否使用了性质`g`。

RegExp.ignoreCase

ECMAScript v3

正则表达式是否区分大小写

摘要

`regexp.ignoreCase`

描述

RegExp对象的属性`ignoreCase`是一个只读的布尔值。它声明了一个给定的正则表达式是否执行区分大小写的匹配，例如创建它时是否使用了性质`i`。

RegExp.lastIndex

ECMAScript v3

下次匹配的起始位置

摘要

`regexp.lastIndex`

描述

RegExp对象的属性`lastIndex`是一个可读可写的值。对于设置了`g`性质的正则表达式来说，该属性存放的是一个整数，它声明了紧接着上次找到的匹配文本的字符的位置。上次匹配的结果是由方法`RegExp.exec()`或`RegExp.test()`找到的，它们都以`lastIndex`属性所指的位置作为下次检索的起始点。这样，就可以通过反复调用这两个方法来遍历一个字符串中的所有匹配文本。注意，不具有性质`g`和不表示全局模式的RegExp对象不能使用`lastIndex`属性。

由于这一属性是可读可写的，所以只要目标字符串的下一次搜索开始，就可以对它进行设置。当方法`exec()`或`test()`再也找不到可以匹配的文本时，它们会自动地把`lastIndex`属性重置为0。如果在成功地匹配了某个字符串之后就开始检索另一个字符串，需要明确地把这个属性设为0。

参阅

`RegExp.exec()`、`RegExp.test()`

RegExp.source

ECMAScript v3

正则表达式的文本

摘要

`regexp.source`

描述

RegExp 对象的属性 `source` 是一个只读的字符串。它存放的是 RegExp 模式的文本。该文本中不包括正则表达式直接量使用的定界符，也不包括性质 `g`、`i`、`m`。

RegExp.test()

ECMAScript v3

检测一个字符串是否匹配某个模式

摘要

`regexp.test(string)`

参数

string

要检测的字符串。

返回值

如果字符串 *string* 中含有与 *regexp* 匹配的文本，就返回 `true`，否则返回 `false`。

抛出

`TypeError`

调用该方法的对象不是 RegExp 时，抛出该异常。

描述

方法 `test()` 将检测字符串 *string*，看它是否含有与 *regexp* 相匹配的文本。如果 *string* 中含有这样的文本，该方法将返回 `true`，否则，返回 `false`。调用 RegExp 对象 *r* 的 `test()` 方法，给它传递字符串 *s*，等价于下面的表达式：

```
(r.exec(s) != null)
```

例子

```
var pattern = /java/i;
pattern.test("JavaScript"); // Returns true
pattern.test("ECMAScript"); // Returns false
```

参阅

`RegExp.exec()`、`RegExp.lastIndex`、`String.match()`、`String.replace()`、`String.substring()`，第 11 章

RegExp.toString()

ECMAScript v3

把正则表达式转换成字符串

覆盖 Object.toString()

摘要`regexp.toString()`**返回值**`regexp` 的字符串表示。**抛出**

TypeError

调用该方法的对象不是 `RegExp` 时，抛出该异常。**描述**`RegExp.toString()` 方法将以正则表达式直接量的形式返回正则表达式的字符串表示。

注意，不允许用实现添加转义序列，这样可以确保返回的字符串是合法的正则表达式直接量。考虑由表达式 `new RegExp("/", "g")` 创建的正则表达式。`RegExp.toString()` 的一种实现对该正则表达式返回 `"//g"`，此外它还能添加转义序列，返回 `"\\/g"`。

String

ECMAScript v1

对字符串的支持

从 Object 继承

构造函数

```
new String(s) // Constructor function
String(s)     // Conversion function
```

参数`s` 要存储在 `String` 对象中或转换成原始字符串的值。**返回值**

当 `String()` 与 `new` 运算符一起作为构造函数使用时，返回一个新创建的 `String` 对象，存放的是字符串 `s` 或 `s` 的字符串表示。当不用 `new` 运算符调用 `String()` 时，它只把 `s` 转换成原始的字符串，并返回转换后的值。

属性`length`

字符串中的字符数。

方法

`charAt()`

抽取字符串中指定位置处的字符。

`charCodeAt()`

返回字符串中指定位置处的字符编码。

`concat()`

把一个或多个值连接到字符串上。

`indexOf()`

在字符串中检索一个字符或一个子串。

`lastIndexOf()`

在字符串中向后检索一个字符或一个子串。

`localeCompare()`

使用基于本地的顺序来比较字符串。

`match()`

用正则表达式执行模式匹配。

`replace()`

用正则表达式执行查找、替换操作。

`search()`

检索字符串中与正则表达式匹配的子串。

`slice()`

返回字符串的一个片段或一个子串。

`split()`

把字符串分割成一个字符串数组，在指定的分界字符处或正则表达式处执行分割。

`substr()`

从字符串中抽取一个子串。该方法是 `substring()` 的一个变体。

`substring()`

从字符串中抽取一个子串。

`toLowerCase()`

将字符串中的所有字符都转换成小写的，然后返回一个副本。

`toString()`

返回原始的字符串值。

`toUpperCase()`

将字符串中的所有字符都转换成大写的，然后返回一个副本。

`valueOf()`

返回原始字符串值。

静态方法

`String.fromCharCode()`

用作为参数而传递的字符代码创建一个新的字符串。

HTML 方法

从 JavaScript 的较早版本开始, `String` 类定义了许多方法, 返回的字符串是把它放在 HTML 标记中修改后得到的。虽然 ECMAScript 没有标准化这些方法, 但它们在客户端和服务端动态生成 HTML 的脚本代码中非常有用。用这些非标准的方法, 就可以为黑体的红色超链接创建常见 HTML 源代码, 如下所示:

```
var s = "click here!";  
var html = s.bold().link("javascript:alert('hello')").fontcolor("red");
```

因为这些方法没有被标准化, 所以它们没有单独的参考页:

`anchor(name)`

在 `` 环境中返回字符串的一个副本。

`big()`

在 `<big>` 环境中返回字符串的一个副本。

`blink()`

在 `<blink>` 环境中返回字符串的一个副本。

`bold()`

在 `` 环境中返回字符串的一个副本。

`fixed()`

在 `<tt>` 环境中返回字符串的一个副本。

`fontcolor(color)`

在 `` 环境中返回字符串的一个副本。

`fontsize(size)`

在 `` 环境中返回字符串的一个副本。

`italics()`

在 `<i>` 环境中返回字符串的一个副本。

`link(url)`

在 `` 环境中返回字符串的一个副本。

`small()`

在 `<small>` 环境中返回字符串的一个副本。

`strike()`

在 `<strike>` 环境中返回字符串的一个副本。

`sub()`

在 `<sub>` 环境中返回字符串的一个副本。

`sup()`

在 `<sup>` 环境中返回字符串的一个副本。

描述

字符串是 JavaScript 的一种基本数据类型。String 类提供了操作原始字符串值的方法。String 对象的 `length` 属性声明了该字符串中的字符数。类 String 定义了大量操作字符串的方法，例如从字符串中提取字符或子串，或者检索字符或子串。注意，JavaScript 的字符串是不可变（immutable）的，String 类定义的方法都不能改变字符串的内容。像 `String.toUpperCase()` 这样的方法，返回的是全新的字符串，而不是修改原始字符串。

在较早的 Netscape 代码基的 JavaScript 实现中（例如 Firefox 的实现中），字符串的行为就像只读的字符数组。例如，从字符串 `s` 中提取第三个字符，可以用 `s[2]` 代替更加标准的 `s.charAt(2)`。此外，对字符串应用 `for/in` 循环时，它将枚举字符串中每个字符的数组下标（但要注意，ECMAScript 标准规定，不能枚举 `length` 属性）。因为字符串的数组行为不标准，所以应该避免使用它。

参阅

第 3 章

String.charAt()

ECMAScript v1

返回字符串中的第 n 个字符

摘要

`string.charAt(n)`

参数

n 应该返回的字符在 `string` 中的下标。

返回值

字符串 `string` 的第 n 个字符。

描述

方法 `String.charAt()` 返回字符串 `string` 中的第 n 个字符。字符串中第一个字符的下标值是 0。如果参数 n 不在 0 和 `string.length - 1` 之间，该方法将返回一个空字符串。注意，JavaScript 并没有一种有异于字符串类型的字符数据类型，所以返回的字符是长度为 1 的字符串。

参阅

`String.charCodeAt()`、`String.indexOf()` 和 `String.lastIndexOf()`

String.charCodeAt()ECMAScript v1

返回字符串中的第 n 个字符的代码

摘要

`string.charCodeAt(n)`

参数

n 返回编码的字符的下标。

返回值

`string` 中的第 n 个字符的 Unicode 编码。这个返回值是 0~65535 之间的 16 位整数。

描述

方法 `charCodeAt()` 与 `charAt()` 执行的操作相似，只不过前者返回的是位于指定位置的字符的编码，而后者返回的则是含有字符本身的子串。如果 n 是负数，或者大于等于字符串的长度，则 `charCodeAt()` 返回 NaN。

要了解从 Unicode 编码创建字符串的方法，请参阅 `String.fromCharCode()`。

参阅

`String.charAt()` 和 `String.fromCharCode()`

String.concat()ECMAScript v3

连接字符串

摘要

`string.concat(value, ...)`

参数

`value, ...`

要连接到 `string` 上的一个或多个值。

返回值

把每个参数都连接到字符串 `string` 上得到的新字符串。

描述

方法 `concat()` 将把它的所有参数都转换成字符串（如果必要），然后按顺序连接到字符串 `string` 的尾部，返回连接后的字符串。注意，`string` 自身并没有被修改。

`String.concat()` 与 `Array.concat()` 很相似。注意，使用 “+” 运算符来进行字符串的连接运算通常更简便一些。

参阅

`Array.concat()`

`String.fromCharCode()`

ECMAScript v1

从字符编码创建一个字符串

摘要

`String.fromCharCode(c1, c2, ...)`

参数

`c1, c2, ...`

零个或多个整数，声明了要创建的字符串中的字符的 Unicode 编码。

返回值

含有指定编码的字符的新字符串。

描述

这个静态方法提供了一种创建字符串的方式，即字符串中的每个字符都由单独的数字 Unicode 编码指定。注意，作为一种静态方法，`fromCharCode()` 是构造函数 `String()` 的属性，而不是字符串或 `String` 对象的方法。

`String.charCodeAt()` 是与 `String.fromCharCode()` 配套使用的实例方法，它提供了获取字符串中单个字符的编码的方法。

例子

```
// Create the string "hello"
var s = String.fromCharCode(104, 101, 108, 108, 111);
```

参阅

`String.charCodeAt()`

`String.indexOf()`

ECMAScript v1

检索字符串

摘要

`string.indexOf(substring)`

`string.indexOf(substring, start)`

参数

`substring`

要在字符串 `string` 中检索的子串。

start

一个选定的整数参数，声明了在字符串 *String* 中开始检索的位置。它的合法取值是 0（字符串中的第一个字符的位置）到 *string.length - 1*（字符串中的最后一个字符的位置）。如果省略了这个参数，将从字符串的第一个字符开始检索。

返回值

如果在 *string* 中的 *start* 位置之后存在 *substring*，返回出现的第一个 *substring* 的位置。如果没有找到子串 *substring*，返回 -1。

描述

方法 *String.indexOf()* 将从头到尾的检索字符串 *string*，看它是否含有子串 *substring*。开始检索的位置在字符串 *string* 的 *start* 处或 *string* 的开头（没有指定 *start* 参数时）。如果找到了一个 *substring*，那么 *String.indexOf()* 将返回 *substring* 的第一个字符在 *string* 中的位置。*string* 中的字符位置是从 0 开始的。

如果在 *string* 中没有找到 *substring*，那么 *String.indexOf()* 方法将返回 -1。

参阅

String.charAt()、*String.lastIndexOf()*、*String.substring()*

String.lastIndexOf()

ECMAScript v1

从后向前检索一个字符串

摘要

string.lastIndexOf(substring)
string.lastIndexOf(substring, start)

参数

substring

要在字符串 *string* 中检索的子串。

start

一个选用的整数参数，声明了在字符串 *string* 中开始检索的位置。它的合法取值是 0（字符串中的第一个字符的位置）到 *string.length - 1*（字符串中的最后一个字符的位置）。如果省略了这个参数，将从字符串的最后一个字符处开始检索。

返回值

如果在 *string* 中的 *start* 位置之前存在 *substring*，那么返回的就是出现的最后一个 *substring* 的位置。如果没有找到子串 *substring*，那么返回的是 -1。

描述

方法 `String.lastIndexOf()` 将从尾到头的检索字符串 *string*，看它是否含有子串 *substring*。开始检索的位置在字符串 *string* 的 *start* 处或 *string* 的结尾（没有指定 *start* 参数时）。如果找到了一个 *substring*，那么 `String.lastIndexOf()` 将返回 *substring* 的第一个字符在 *string* 中的位置。由于是从尾到头的检索一个字符串，所以找到的第一个 *substring* 其实是 *string* 中出现在位置 *start* 之前的最后一个 *substring*。

如果在 *string* 中没有找到 *substring*，那么该方法将返回 `-1`。

注意，虽然 `String.lastIndexOf()` 是从尾到头的检索字符串 *string*，但是它返回的字符位置仍然是从头开始计算的。字符串中第一个字符的位置是 `0`，最后一个字符的位置是 `string.length - 1`。

参阅

`String.charAt()`、`String.indexOf()`、`String.substring()`

String.length

ECMAScript v1

字符串的长度

摘要

`string.length`

描述

属性 `String.length` 是一个只读整数，它声明了指定字符串 *string* 中的字符数。对于任何一个字符串 *s*，它最后一个字符的下标都是 `s.length - 1`。用 `for/in` 循环不能枚举出字符串的 `length` 属性，用 `delete` 运算符也不能删除它。

String.localeCompare()

ECMAScript v3

用本地特定的顺序来比较两个字符串

摘要

`string.localeCompare(target)`

参数

target

要以本地特定的顺序与 *string* 进行比较的字符串。

返回值

说明比较结果的数字。如果 *string* 小于 *target*，则 `localeCompare()` 返回小于 `0` 的

数。如果 *string* 大于 *target*，该方法返回大于 0 的数。如果两个字符串相等，或根据本地排序规则没有区别，该方法返回 0。

描述

把 < 和 > 运算符应用到字符串时，它们只用字符的 Unicode 编码比较字符串，而不考虑当地的排序规则。以这种方法生成的顺序不一定是正确的。例如，西班牙语中，其中字母“ch”通常作为出现在字母“c”和“d”之间的字符来排序。

`localeCompare()` 方法提供的比较字符串的方法，考虑了默认的本地排序规则。ECMAScript 标准没有规定如何进行本地特定的比较操作，它只规定该函数采用底层操作系统提供的排序规则。

例子

可以用下列代码，按照地方特定的排序规则对一个字符串数组排序。

```
var strings; // The array of strings to sort; initialized elsewhere
strings.sort(function(a,b) { return a.localeCompare(b) });
```

String.match()

ECMAScript v3

找到一个或多个正则表达式的匹配

摘要

`string.match(regex)`

参数

regex

声明了要匹配的模式 `RegExp` 对象。如果该参数不是 `RegExp` 对象，则首先将它传递给 `RegExp()` 构造函数，把它转换成 `RegExp` 对象。

返回值

存放匹配结果的数组。该数组的内容依赖于 *regex* 是否具有全局性质 `g`。下面详细说明了这个返回值。

描述

方法 `match()` 将检索字符串 *string*，以找到一个或多个与 *regex* 匹配的文本。这个方法的行为很大程度上依赖于 *regex* 是否具有性质 `g`。关于正则表达式的完整细节，请参阅第 11 章。

如果 *regex* 没有性质 `g`，那么 `match()` 就只能在 *string* 中执行一次匹配。如果没有找到任何匹配的文本，`match()` 将返回 `null`。否则，它将返回一个数组，其中存放了与它找到的匹配文本有关的信息。该数组的第 0 个元素存放的是匹配文本，其余的元素存放的

是与正则表达式的子表达式匹配的文本。除了这些常规的数组元素之外，返回的数组还含有两个对象属性。`index`属性声明的是匹配文本的起始字符在 *string* 中的位置，`input` 属性声明的是对 *string* 的引用。

如果 *regexp* 具有标志 `g`，那么 `match()` 将执行全局检索，找到 *string* 中的所有匹配子串。如果没有找到任何匹配的子串，它将返回 `null`。如果找到了一个或多个匹配子串，它将返回一个数组。不过，全局匹配返回的数组的内容与前者大不相同，它的数组元素存放的是 *string* 中的所有匹配子串，而且它也没有 `index` 属性或 `input` 属性。注意，在全局匹配的模式下，`match()` 既不提供与子表达式匹配的文本的信息，也不声明每个匹配子串的位置。如果你需要这些全局检索的信息，可以使用 `RegExp.exec()`。

例子

下面的全局匹配可以找到字符串中的所有数字：

```
"1 plus 2 equals 3".match(/\d+/g) // Returns ["1", "2", "3"]
```

下面的非全局匹配使用了更加复杂的正则表达式，它具有几个用括号括起来的子表达式。与该表达式匹配的是一个 URL，与它的子表达式匹配的是那个 URL 的协议部分、主机部分和路径部分：

```
var url = /((\w+):\/\/([(\w.]+)\.(\S*)))/;
var text = "Visit my home page at http://www.isp.com/~david";
var result = text.match(url);
if (result != null) {
    var fullurl = result[0]; // Contains "http://www.isp.com/~david"
    var protocol = result[1]; // Contains "http"
    var host = result[2]; // Contains "www.isp.com"
    var path = result[3]; // Contains "~david"
}
```

参阅

`RegExp`、`RegExp.exec()`、`RegExp.test()`、`String.replace()`、`String.search()`，
第 11 章

String.replace()

ECMAScript v3

替换一个与正则表达式匹配的子串

摘要

```
string.replace(regexp, replacement)
```

参数

regexp

声明了要替换的模式的 `RegExp` 对象。如果该参数是一个字符串，则将它作为要检索的直接量文本模式，而不是首先被转换成 `RegExp` 对象。

replacement

一个字符串，声明的是替换文本或生成替换文本的函数。详见描述部分。

返回值

一个新字符串，是用 *replacement* 替换了与 *regexp* 的第一次匹配或所有匹配之后得到的。

描述

字符串 *string* 的方法 *replace()* 执行的是查找并替换的操作。它将在 *string* 中查找与 *regexp* 相匹配的子串，然后用 *replacement* 替换这些子串。如果 *regexp* 具有全局性质 *g*，那么 *replace()* 将替换所有的匹配子串。否则，它只替换第一个匹配子串。

replacement 可能是字符串或函数。如果它是一个字符串，那么每个匹配都将由字符串替换。但 *replacement* 中的 *\$* 字符具有特殊的含义。如下表所示，它说明从模式匹配得到的字符串将用于替换。

字符	替换文本
\$1、\$2、...、\$99	与 <i>regexp</i> 中的第 1 到第 99 个子表达式相匹配的文本
\$&	与 <i>regexp</i> 相匹配的子串
\$`	位于匹配子串左侧的文本
\$'	位于匹配子串右侧的文本
\$\$	直接量符号

ECMAScript v3 规定，*replace()* 方法的参数 *replacement* 可以是函数而不是字符串。在这种情况下，每个匹配都调用该函数；它返回的字符串将作为替换文本使用。该函数的第一个参数是匹配模式的字符串。接下来的参数是与模式中的子表达式匹配的字符串，可以有 0 个或多个这样的参数。接下来参数是一个整数，声明了匹配在 *string* 中出现的位置。最后一个参数是 *string* 自身。

例子

要确保单词“JavaScript”中的大写字母是正确的，可用下列代码：

```
text.replace(/javascript/i, "JavaScript");
```

要将名字“Doe, John”转换成“John Doe”的形式，可用下列代码：

```
name.replace(/(\w+)\s*,\s*(\w+)/, "$2 $1");
```

用花引号替换直引号，可用下列代码：

```
text.replace(/"([^"]*)"/g, "`$1'");
```

使字符串中所有单词的第一个字母都是大写的，可用下列代码：

```
text.replace(/\b\w+\b/g, function(word) {
```

```
        return word.substring(0,1).toUpperCase() +  
               word.substring(1);  
    });
```

参阅

RegExp、RegExp.exec()、RegExp.test()、String.match()、String.search(), 第 11 章

String.search()

ECMAScript v3

检索与正则表达式相匹配的子串

摘要

string.search(*regex*)

参数

regex

要在字符串 *string* 中检索的 RegExp 对象，该对象具有指定的模式。如果该参数不是 RegExp 对象，则首先将它传递给 RegExp() 构造函数，把它转换成 RegExp 对象。

返回值

string 中第一个与 *regex* 相匹配的子串的起始位置。如果没有找到任何匹配的子串，则返回 -1。

描述

方法 search() 将在字符串 *string* 中检索与 *regex* 相匹配的子串，并且返回第一个匹配子串的第一个字符的位置。如果没有找到任何匹配的子串，则返回 -1。

search() 并不执行全局匹配，它将忽略标志 g。它也忽略 *regex* 的 lastIndex 属性，并且总是从字符串的开始进行检索，这意味着它总是返回 *string* 的第一个匹配的位置。

例子

```
var s = "JavaScript is fun";  
s.search(/script/i) // Returns 4  
s.search(/a(.)a/)   // Returns 1
```

参阅

RegExp、RegExp.exec()、RegExp.test()、String.match()、String.replace(), 第 11 章

String.slice()

ECMAScript v3

抽取一个子串

摘要

`string.slice(start, end)`

参数

start

要抽取的片段的起始下标。如果是负数，那么该参数声明了从字符串的尾部开始算起的位置。也就是说，-1 指字符串中的最后一个字符，-2 指倒数第二个字符，以此类推。

end

紧接着要抽取的片段的结尾的下标。如果没有指定这一参数，那么要抽取的子串包括 *start* 到原字符串结尾的字符串。如果该参数是负数，那么它声明了从字符串的尾部开始算起的位置。

返回值

一个新字符串，包括字符串 *string* 从 *start* 开始（包括 *start*）到 *end* 为止（不包括 *end*）的所有字符。

描述

方法 `slice()` 将返回一个含有字符串 *string* 的片段的字符串或返回它的一个子串。但是该方法不修改 *string*。

String 对象的方法 `slice()`、`substring()` 和 `substr()`（不建议使用）都返回字符串的指定部分。`slice()` 比 `substring()` 要灵活一些，因为它允许使用负数作为参数。`slice()` 与 `substr()` 有所不同，因为它用两个字符的位置指定子串，而 `substr()` 则用字符位置和长度来指定子串。还要注意的，`String.slice()` 与 `Array.slice()` 相似。

例子

```
var s = "ABCDEFGH";
s.slice(0,4)    // Returns "ABCD"
s.slice(2,4)    // Returns "CD"
s.slice(4)      // Returns "EFGH"
s.slice(3,-1)   // Returns "DEF"
s.slice(3,-2)   // Returns "DE"
s.slice(-3,-1)  // Should return "EF"; returns "ABCDEF" in IE 4
```

Bug

在 Internet Explorer 4 中，参数 *start* 的值无效（但在 IE 后来的版本中修正了）。*start* 值指定的不是从字符串尾部开始算起的字符位置，而是指定第 0 个字符的位置。

参阅

`Array.slice()`、`String.substring()`

String.split()

ECMAScript v3

将字符串分割成字符串数组

摘要

```
string.split(delimiter, limit)
```

参数

delimiter

字符串或正则表达式，从该参数指定的地方分割 *string*。

limit

这个可选的整数指定了返回的数组的最大长度。如果设置了该参数，返回的子串不会多于这个参数指定的数字。如果没有设置该参数，整个字符串都会被分割，不考虑它的长度。

返回值

一个字符串数组，是通过在 *delimiter* 指定的边界处将字符串 *string* 分割成子串创建的。返回的数组中的子串不包括 *delimiter* 自身，但下面列出的情况除外。

描述

方法 `split()` 将创建并返回一个字符串数组，该数组中的元素是指定的字符串 *string* 的子串，最多具有 *limit* 个。这些子串是通过从头到尾检索字符串中与 *delimiter* 匹配的文本，在匹配文本之前和之后分割 *string* 得到的。返回的子串中不包括定界符文本（本部分结尾处提到的情况除外）。如果定界符从字符串开头开始匹配，返回的数组的第一个元素是空串，即出现在定界符之前的文本。同样，如果定界符与字符串的结尾匹配，返回的数组的最后一个元素也是空串（假定与 *limit* 没有冲突）。

如果没有指定 *delimiter*，那么它根本就不对 *string* 执行分割，返回的数组中只有一个元素，而不分割字符串元素。如果 *delimiter* 是一个空串或与空串匹配的正则表达式，那么 *string* 中的每个字符之间都会被分割，返回的数组的长度与字符串长度相等（假定 *limit* 不小于该长度）（注意，这是一种特殊情况，因为没有匹配第一个字符之前和最后一个字符之后的空串）。

前面说过，该方法返回的数组中的子串不包括用于分割字符串的定界符文本。但如果 *delimiter* 是包括子表达式的正则表达式，那么返回的数组中包括与这些子表达式匹配的子串（但不包括与整个正则表达式匹配的文本）。

注意，`String.split()` 执行的操作与 `Array.join()` 执行的操作相反。

例子

在使用结构复杂的字符串时，方法 `split()` 最有用。例如：

```
"1:2:3:4:5".split(":"); // Returns ["1","2","3","4","5"]
"|a|b|c|".split("|");   // Returns ["", "a", "b", "c", ""]
```

`split()` 方法的另一个常见用法是解析命令和与之相似的字符串,用空格将它们分割成单词:

```
var words = sentence.split(' ');
```

用正则表达式作为定界符,很容易把字符串分割成单词:

```
var words = sentence.split(/\s+/);
```

要把字符串分割成字符数组,可以用空串作为定界符。如果只想把字符串的前一部分分割成字符数组,需要使用 *limit* 参数:

```
"hello".split("");      // Returns ["h","e","l","l","o"]
"hello".split("", 3);   // Returns ["h","e","l"]
```

如果想使返回的数组包括定界符或定界符的一个或多个部分,可以使用带子表达式的正则表达式。例如,下面的代码将在 HTML 标记处分割字符串,返回的数组中包括这些标记:

```
var text = "hello <b>world</b>";
text.split(/(<[^>]*>)/); // Returns ["hello ", "<b>", "world", "</b>", ""]
```

参阅

`Array.join()`、`RegExp`, 第 11 章

String.substr()

JavaScript 1.2; 反对使用

抽取一个子串

摘要

string.substr(start, length)

参数

start

要抽取的子串的起始下标。如果是一个负数,那么该参数声明从字符串的尾部开始算起的位置。也就是说, -1 指字符串中的最后一个字符, -2 指倒数第二个字符,以此类推。

length

子串中的字符数。如果省略了这个参数,那么返回从 *string* 的开始位置到结尾的子串。

返回值

一个字符串的副本,包括从 *string* 的 *start* 处 (包括 *start* 所指的字符) 开始的 *length* 个字符。如果没有指定 *length*, 返回的字符串包含从 *start* 到 *string* 结尾的字符。

描述

`substr()` 将在 *string* 中抽取并返回一个子串。但是它并不修改 *string*。

注意, `substr()` 指定的是子串的开始位置和长度, 它是 `String.substring()` 和 `String.splice()` 的一种有用的替代方法, 后两者指定的都是起始字符的位置。但要注意, ECMAScript 没有标准化该方法, 因此反对使用它。

例子

```
var s = "abcdefg";
s.substr(2,2);    // Returns "cd"
s.substr(3);      // Returns "defg"
s.substr(-3,2);   // Should return "ef"; returns "ab" in IE 4
```

Bug

在 IE 4 中, 参数 *start* 的值无效 (这在 IE 后来的版本中已经修正了)。*start* 值指定的不是从字符串尾部开始算起的字符位置, 而是第 0 个字符的位置。

参阅

`String.slice()`、`String.substring()`

String.substring()

ECMAScript v1

返回字符串的一个子串

摘要

`string.substring(from, to)`

参数

from

一个非负的整数, 声明了要抽取的子串的第一个字符在 *string* 中的位置。

to

一个可选的非负的整数, 比要抽取的子串的最后一个字符在 *string* 中的位置多 1。如果省略了该参数, 返回的子串直到字符串的结尾。

返回值

一个新字符串, 其长度为 $to - from$, 存放的是字符串 *string* 的一个子串。这个新字符串含有的字符是从 *string* 中的 *from* 处到 $to - 1$ 处复制的。

描述

`String.substring()` 将返回字符串 *string* 的子串, 由 *from* 到 *to* 之间的字符构成, 包括位于 *from* 的字符, 不包括位于 *to* 的字符。

如果参数 *from* 与 *to* 相等, 那么该方法返回的就是一个空串 (即长度为 0 的字符串)。如果 *from* 比 *to* 大, 那么该方法在抽取子串之前会先交换这两个参数。

要记住，该子串包括 *from* 处的字符，不包括 *to* 处的字符。虽然这样看来有违直觉，但这种系统一个值得注意的重要特性是，返回的子串的长度总等于 *to - from*。

注意 `String.slice()` 和非标准的 `String.substr()` 都可以从一个字符串提取子串。和这些方法不同的是，`String.substring()` 不接受负的参数。

参阅

`String.charAt()`、`String.indexOf()`、`String.lastIndexOf()`、`String.slice()` 和 `String.substr()`

String.toLocaleLowerCase()

ECMAScript v3

把字符串转换小写

摘要

`string.toLocaleLowerCase()`

返回值

string 的一个副本，按照本地方式转换成小写字母。只有几种语言（如土耳其语）具有地方特有的大小写映射，所以该方法的返回值通常与 `toLowerCase()` 一样。

参阅

`String.toLocaleUpperCase()`、`String.toLowerCase()`、`String.toUpperCase()`

String.toLocaleUpperCase()

ECMAScript v3

将字符串转换成大写

摘要

`string.toLocaleUpperCase()`

返回值

string 的一个副本，按照本地方式转换成了大写字母。只有几种语言（如土耳其语）具有地方特有的大小写映射，所以该方法的返回值通常与 `toUpperCase()` 一样。

参阅

`String.toLocaleLowerCase()`、`String.toLowerCase()`、`String.toUpperCase()`

String.toLowerCase()

ECMAScript v1

将字符串转换成小写

摘要

`string.toLowerCase()`

返回值

string 的一个副本，其中所有大写字符都被转换成了小写字符。

String.toString()

ECMAScript v1

返回字符串

覆盖 Object.toString()

摘要

string.toString()

返回值

string 的原始字符串值。一般不会调用该方法。

抛出

TypeError

调用该方法的对象不是 String 时抛出该异常。

参阅

String.valueOf()

String.toUpperCase()

ECMAScript v1

将字符串转换成大写

摘要

string.toUpperCase()

返回值

string 的一个副本，其中所有小写字符都被转换成了大写的。

String.valueOf()

ECMAScript v1

返回字符串

覆盖 Object.valueOf()

摘要

string.valueOf()

返回值

string 的原始字符串值。

抛出

TypeError

调用该方法的对象不是 String 时抛出该异常。

参阅

`String.toString()`

SyntaxError

ECMAScript v3

抛出该错误用来通知语法错

Object → Error → SyntaxError

构造函数

`new SyntaxError()`

`new SyntaxError(message)`

参数

message

提供异常细节的出错消息（可选）。如果设置了该参数，它将作为 `SyntaxError` 对象的 `message` 属性的值。

返回值

新构造的 `SyntaxError` 对象。如果指定了参数 *message*，该 `Error` 对象将它作为 `message` 属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果不用 `new` 运算符，把 `SyntaxError()` 构造函数当作函数调用，它的行为与使用 `new` 运算符调用时一样。

属性

`message`

提供异常细节的出错消息。该属性存放传递给构造函数的字符串，或存放实现定义的默认字符串。详见“`Error.message`”。

`name`

声明异常类型的字符串。所有 `SyntaxError` 对象的 `name` 属性都继承值“`SyntaxError`”。

描述

`SyntaxError` 类的一个实例会被抛出以通知 JavaScript 代码中的语法错误。`eval()` 方法、`Function()` 构造函数和 `RegExp()` 构造函数都可能抛出这种类型的异常。关于抛出和捕捉异常的细节，请参阅“`Error`”。

参阅

`Error`、`Error.message`、`Error.name`

TypeError

ECMAScript v3

当一个值的类型错误时，抛出该异常

Object → Error → TypeError

构造函数

`new TypeError()``new TypeError(message)`

参数

message

提供异常细节的出错消息（可选）。如果设置了该参数，它将作为 `TypeError` 对象的 `message` 属性的值。

返回值

新构造的 `TypeError` 对象。如果指定了参数 *message*，该 `Error` 对象将它作为 `message` 属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果不用 `new` 运算符，则把 `TypeError()` 构造函数当作函数调用，它的行为与使用 `new` 运算符调用时一样。

属性

message

提供异常细节的出错消息。该属性存放传递给构造函数的字符串，或存放实现定义的默认字符串。详见“`Error.message`”。

name

声明异常类型的字符串。所有 `TypeError` 对象的 `name` 属性都继承值“`TypeError`”。

描述

当一个值的类型与要求不符时，`TypeError` 类的一个实例就会被抛出。在访问值为 `null` 或 `undefined` 的属性时，这种情况经常发生。如果由一个对象（其他类的实例）调用另一个类定义的方法，或者对于不是构造函数的值使用 `new` 运算符时，也会发生这种情况。当调用内部函数或方法时，如果传递的参数多于期望的个数，JavaScript 的实现也允许抛出 `TypeError` 异常。关于抛出和捕捉异常的细节，请参阅“`Error`”。

参阅

`Error`、`Error.message`、`Error.name`

undefined

ECMAScript v3

未定义值

摘要

`undefined`

描述

`undefined` 是全局属性，存放 JavaScript 的 `undefined` 值。它与尝试读取不存在的对象属性的值时返回的值相同。用 `for/in` 循环不能枚举出 `undefined` 属性，用 `delete` 运算符也不能删除它。注意，`undefined` 不是常量，可以设置为其他值。

只能用 `===` 运算符测试一个值是否是未定义的，因为 `==` 运算符认为 `undefined` 值等价于 `null`。

unescape()

ECMAScript v1;ECMAScript v3 反对使用

给转义字符串解码

摘要

`unescape(s)`

参数

s 要解码或“反转义”的字符串。

返回值

s 解码后的一个副本。

描述

`unescape()` 是全局函数，对 `escape()` 编码的字符串解码。该函数是通过找到形式为 `%xx` 和 `%uxxxx` 的字符序列（这里 *x* 表示十六进制的数字），用 Unicode 字符 `\u00xx` 和 `\uxxxx` 替换这样的字符序列进行解码的。

虽然 ECMAScript 的第一个版本标准化了 `unescape()`，但 ECMAScript v3 从标准中删除了它，反对使用该方法。虽然 ECMAScript 的实现可能实现了该函数，但这不是必需的。应该用 `decodeURI()` 和 `decodeURIComponent()` 代替 `unescape()`。详见“`escape()`”。

参阅

`decodeURI()`、`decodeURIComponent()`、`escape()`、`String`

URIError

ECMAScript v3

由 URI 的编码和解码方法抛出

Object → Error → URIError

构造函数

`new URIError()`

`new URIError(message)`

参数

message

提供异常细节的出错消息（可选）。如果设置了该参数，它将作为 `URIError` 对象的 `message` 属性的值。

返回值

新构造的 `URIError` 对象。如果指定了参数 *message*，该 `Error` 对象将它作为 `message` 属性的值，否则，它将用实现定义的默认字符串作为该属性的值。如果不用 `new` 运算符，而把 `URIError()` 构造函数当作函数调用，它的行为与使用 `new` 运算符调用时一样。

属性

`message`

提供异常细节的出错消息。该属性存放传递给构造函数的字符串，或存放实现定义的默认字符串。详见“`Error.message`”。

`name`

声明异常类型的字符串。所有 `URIError` 对象的 `name` 属性都继承值“`URIError`”。

描述

如果指定的字符串含有不合法的十六进制转义序列，则 `decodeURI()` 或 `decodeURIComponent()` 方法就会抛出 `URIError` 类的实例。如果指定的字符串含有不合法的 Unicode 替代对，`encodeURI()` 或 `encodeURIComponent()` 方法也会抛出该异常。关于抛出和捕捉异常的细节，请参阅“`Error`”。

参阅

`Error`、`Error.message`、`Error.name`



客户端 JavaScript

参考手册

本书的这一部分是客户端 JavaScript 中的所有对象、属性、函数、方法和事件处理程序的完整参考手册。这部分的使用方法参见第三部分开始处。

这部分文档包含的类和对象有：

Anchor	DOMException	JSObject	Table
Applet	DOMImplementation	KeyEvent	TableCell
Attr	DOMParser	Link	TableRow
Canvas	Element	Location	TableSection
CanvasGradient	Event	MimeType	Text
CanvasPattern	ExternalInterface	MouseEvent	Textarea
CanvasRenderingContext2D	FlashPlayer	Navigator	UIEvent
CDATASection	Form	Node	Window
CharacterData	Frame	NodeList	XMLHttpRequest
Comment	History	Option	XMLSerializer
CSS2Properties	HTMLCollection	Plugin	XPathExpression
CSSRule	HTMLDocument	ProcessingInstruction	XPathResult
CSSStyleSheet	HTMLElement	Range	XSLTProcessor
Document	IFrame	RangeException	
DocumentFragment	Image	Screen	
DocumentType	Input	Select	

客户端 JavaScript

参考手册

这一部分是一个参考手册，介绍了客户端 JavaScript 所定义的类、方法、属性和事件处理程序。这个介绍以及第三部分开始的示例参考页说明了如何使用这个参考部分以及最大程度地利用好这个参考手册。仔细阅读这些材料，你可以更容易地找到和使用自己需要的信息。

这个参考手册是按照字母顺序安排的。类的属性和方法的参考页都以它们全称的字母顺序排列，而这个全称包括定义它们的类的名称。例如，如果要阅读 Form 类的 `submit()` 方法，应该在“Form.submit”下查看，而不是在“submit”下查看。

为了节省本书的篇幅，这个参考手册中的大部分属性没有自己的参考页（但所有方法和事件处理程序都有自己的参考页）。

简单的属性在定义它们的类的参考页中都有完整的说明。例如，可以在“HTMLDocument”参考页中读到 HTMLDocument 类的 `images[]` 属性的说明。需要大量解释的重要属性都有自己的参考页，在定义这些属性的类或接口的参考页中可以找到对这些参考页的交叉引用。例如，在“HTMLDocument”参考页中查看 `cookie` 属性时，可以看到对该属性的简短描述和对“HTMLDocument.cookie”参考页的引用。

客户端 JavaScript 具有大量全局属性和函数，如 `window`、`history` 和 `alert()`。在客户端 JavaScript 中，Window 对象是全局对象，客户端 JavaScript 的全局属性和函数实际上都是 Window 类的属性。因此，全局属性和函数都在“Window”参考页或“Window.alert()”条目下介绍。

一旦找到了要查询的参考页，那么找到你需要的信息就没有太大的困难了。不过，如果知道参考页是如何编写和组织的，就能更好地使用参考页。第三部分的开头是一个示例参考页，标题为“示例条目”，它示范了每个参考页的结构，告诉你在哪里可以找到什么类型的信息。

Anchor

DOM 级别 0

超文本链接的目标

Node → Element → HTMLElement → Anchor

属性

String name

Anchor 对象的名字。该属性的值由标记 <a> 的 name 属性设置。

方法

focus()

滚动文档以使锚的位置可见。

HTML 语法

锚对象是由包含 name 属性的标准 HTML 标记 <a> 创建的：

```
<a name="name" // Links may refer to this anchor by this name
...
</a>
```

描述

锚是 HTML 文档中一个命名了的地点，由具有 name 属性的标记 <a> 创建。Document 对象具有包含 Anchor 对象的一个数组属性 anchors[]，它的元素代表文档中所包含的锚。Anchor 对象可以通过数组中的下标或者名称来引用。

可以通过把 Location 对象的 hash 属性设置为一个 # 符号后面跟着锚的名称或者只是调用 Anchor 对象自身的 focus() 方法，来让浏览器显示一个锚的位置。

注意，用来创建锚的标记 <a> 还可以用来创建超文本链接。虽然在 HTML 用语中，超文本链接常常被称为锚，但是在 JavaScript 中代表它的则是 Link 对象，而不是 Anchor 对象。

例子

```
// Scroll the document so the anchor named "_bottom_" is visible
document.anchors['_bottom_'].focus();
```

参阅

Document、Link、Location

Anchor.focus()

0 级 DOM

滚动以使锚位置变得可见

摘要

```
void focus();
```

描述

这个方法滚动文档以使 Anchor 对象的位置变为可见。

Applet

0 级 DOM

嵌在网页中的小程序

摘要

```
document.applets[i]  
document.appletName
```

属性

Applet 对象具有和 <applet> 标记的 HTML 属性相对应的属性（参阅 HTMLElement 了解更多细节）。它还有和它所代表的 Java 小程序的公有字段相对应的属性。

方法

Applet 对象的方法和它所代表的 Java 小程序的公有方法相同。

描述

Applet 对象代表嵌入在 HTML 文档中的 Java 小程序。一个文档的 Applet 对象可以通过 Document 对象的 applets[] 集合来获取。

它的属性代表这个 Java 小程序的公有字段，它的方法代表这个 Java 小程序的公有方法。我们知道 Java 是一种强类型语言。这意味着小程序的每一个字段都要明确地声明为某种数据类型，把其他类型的值赋给它会引发运行时错误。这种规则同样适用于小程序的方法，即每个参数都有一个指定的类型，而且不能像在 JavaScript 中那样随意地省略参数。参阅第 23 章了解更多细节。

参阅

JSObject，第三部分中的 JavaObject，第 12 章，第 23 章

Attr

1 级 DOM 核心

文档元素的一个属性

Node → Attr

属性

```
readonly String name
```

属性的名称

```
readonly Element ownerElement [2 级 DOM]
```

包含这个属性的 Element 对象，如果 Attr 对象当前不以任何 Element 相关，就是 null。

readonly boolean specified

如果属性是在文档源中显式指定的或者由一个脚本设置的, 就为 true; 如果属性没有显式地指定, 而是由文档的 DTD 的一个默认值指定的, 就为 false。

String value

属性的值。当读取这个属性的时候, 属性值作为一个字符串返回。当把这个属性设置为一个字符串的时候, 它自动地创建一个包含相同的文本的 Text 节点, 并且让这个 Text 节点作为 Attr 对象的唯一的子节点。

描述

一个 Attr 对象代表着一个 Element 节点的属性。Attr 对象和 Element 节点相关联, 但是, 却并不是文档树的直接的一部分 (并且有一个空的 parentNode 属性)。你可以通过 Node 接口的 attributes 属性来获得一个 Attr 对象, 或者通过调用 Element 接口的 getAttributeNode() 或 getAttributeNodeNS() 方法。

一个属性的值通过一个 Attr 节点的子节点来表示。在 HTML 文档中, 一个 Attr 节点总是只有一个 Text 节点孩子, 并且 value 属性提供了读取和写入这个孩子节点的值的快捷方式。

XML 语法允许 XML 文档拥有包含了 Text 节点和 EntityReference 节点的属性, 这就是为什么一个属性值无法用一个字符串来完整地表示。然而, 实际上, Web 浏览器扩展了 XML 属性值中的任何实体引用, 并且没有实现 EntityReference 接口 (该接口并没有在本书中介绍)。因此, 在客户端 JavaScript 中, value 属性是唯一需要读取和写入的属性值。

既然属性值可以完全由字符串来表示, 通常也根本不需要使用 Attr 接口。在大多数情况下, 使用属性的最容易方法就是把它和 Element.getAttribute() 和 Element.setAttribute() 方法一起使用。这些方法使用针对属性值的字符串并且避免同时使用 Attr 节点。

参阅

Element

Button

参阅 Input

Canvas

Firefox 1.5, Safari 1.3, Opera 9

一个用于脚本化绘图的 HTML 元素

Node → Element → HTMLElement → Canvas

属性

String height

画布的高度。和一幅图像一样, 这个属性可以指定为一个整数像素值或者是窗口高度的百分比。当这个值改变的时候, 在该画布上已经完成的任何绘图都会擦除掉。默认值是 300。

String width

画布的宽度。和一幅图像一样，这个属性可以指定为一个整数像素值或者是窗口宽度的百分比。当这个值改变的时候，在该画布上已经完成的任何绘图都会擦除掉。默认值是 300。

方法

getContext()

返回一个 CanvasRenderingContext2D 对象，使用该对象在画布上绘图。必须为这个方法传递一个字符串“2d”来表明你需要进行二维的绘图。

描述

Canvas 对象表示一个 HTML 画布元素。它没有自己的行为，但是定义了一个 API 支持脚本化客户端绘图操作。你可以直接在这个对象上指定宽度和高度，但是，其大多数功能都可以通过 CanvasRenderingContext2D 对象获得。这通过调用 Canvas 对象的 getContext() 方法并且把直接量字符串“2d”作为唯一的参数传递给它而获得。

<canvas> 标记在 Safari 1.3 中引入，在编写本书的时候，它在 Firefox 1.5 和 Opera 9 中也得到了支持。在 IE 中，<canvas> 标记及其 API 可以使用位于 <http://excanvas.sourceforge.net/> 的 ExplorerCanvas 开源项目来模拟。

参阅

CanvasRenderingContext2D, 第 22 章

Canvas.getContext()

返回一个用于在画布上绘图的环境

摘要

CanvasRenderingContext2D getContext(String contextID)

参数

contextID

这个参数指定了你想要在画布上绘制的类型。当前唯一的合法值是“2d”，它指定了二维绘图，并且导致这个方法返回一个环境对象，该对象导出一个二维绘图 API。

返回值

一个 CanvasRenderingContext2D 对象，使用它可以绘制到 Canvas 元素中。

描述

返回一个表示用来绘制的环境类型的环境。其本意是要为不同的绘制类型（2 维、3 维）提

供不同的环境。当前，唯一支持的是“2d”，它返回一个 `CanvasRenderingContext2D` 对象，该对象实现了一个画布所使用的大多数方法。

参阅

`CanvasRenderingContext2D`

CanvasGradient

Firefox 1.5, Safari 1.3, Opera 9

用于一个画布中的一个颜色渐变

Object → `CanvasGradient`

方法

`addColorStop()`

为渐变指定颜色和位置。

描述

`CanvasGradient` 对象表示一个颜色渐变，该渐变可以使用 `CanvasRenderingContext2D` 对象的 `strokeStyle` 属性和 `fillStyle` 属性来指定。`CanvasRenderingContext2D` 的 `createLinearGradient()` 方法和 `createRadialGradient()` 都返回 `CanvasGradient` 对象。

一旦创建了一个 `CanvasGradient` 对象，就使用 `addColorStop()` 方法来指定应该出现在渐变中各个位置的颜色。在指定的位置之间，颜色通过插值来产生一个平滑的渐变或过渡。在渐变的开始处和结束处显示地创建透明的黑色色标。

参阅

`CanvasRenderingContext2D.createLinearGradient()`

`CanvasRenderingContext2D.createRadialGradient()`

CanvasGradient.addColorStop()

在渐变中的某一点添加一个颜色变化

摘要

`void addColorStop(float offset, String color)`

参数

offset

这是一个范围在 0.0 到 1.0 之间的浮点值，表示渐变的开始点和结束点之间的一部分。*offset* 为 0 对应开始点，*offset* 为 1 对应结束点。

color

以一个 CSS 颜色字符串的方式，表示在指定 *offset* 显示的颜色。沿着渐变某一点的颜色是根据这个值以及任何其他颜色色标来插值的。

描述

`addColorStop()` 提供了一种描述渐变中颜色变化的方法。这个方法可以调用一次或多次来改变渐变的开始点和结束点之间的特定百分段的颜色。

如果这个方法无法在一个渐变上调用，这个渐变是透明的。至少必须指定一个颜色色标来产生一个可见的颜色渐变。

CanvasPattern

Firefox 1.5, Safari 1.3, Opera 9

用在一个 Canvas 中的基于图像的模式

Object → CanvasPattern

描述

CanvasPattern 对象由 CanvasRenderingContext2D 对象的 `createPattern()` 方法返回。一个 CanvasPattern 对象可以用作 CanvasRenderingContext2D 对象的 `strokeStyle` 和 `fillStyle` 属性的值。

CanvasPattern 对象没有自己的属性或方法。参阅 CanvasRenderingContext2D.`createPattern()` 详细了解如何创建这个对象。

参阅

CanvasRenderingContext2D.`createPattern()`

CanvasRenderingContext2D

Firefox 1.5, Safari 1.3, Opera 9

用来在一个画布上绘图的对象

Object → CanvasRenderingContext2D

属性

readonly Canvas canvas

这个环境可以绘制于其上的 Canvas 元素。

Object fillStyle

用来填充路径的当前的颜色、模式或渐变。这个属性可以设置为一个字符串或者一个 CanvasGradient 或 CanvasPattern 对象。当设置为一个字符串的时候，它被解析为一个 CSS 颜色值并且用来进行实心填充。当设置为一个 CanvasGradient 或 CanvasPattern 对象，通过使用指定的渐变或模式来完成填充。参阅 CanvasRenderingContext2D.`createLinearGradient()`、CanvasRenderingContext2D.`createRadialGradient()` 和 CanvasRenderingContext2D.`createPattern()`。

float globalAlpha

指定在画布上绘制的内容的不透明度。这个值的范围在 0.0（完全透明）和 1.0（一点也不透明）之间。默认值为 1.0。

String globalCompositeOperation

指定颜色如何与画布上已有的颜色组合（合成）。参阅这个属性的可能值的单独索引页。

String lineCap

指定线条的末端如何绘制。合法的值是“butt”、“round”和“square”。默认值是“butt”。参阅这个属性的单独参考页来了解更多细节。

String lineJoin

指定两条线条如何连接。合法的值是“round”、“bevel”和“miter”。默认值是“miter”。参阅这个属性的单独参考页了解更多细节。

float lineWidth

指定了画笔（绘制线条）操作的线条宽度。默认值是1.0，并且这个属性必须比0.0大。较宽的线条在路径上居中，每边有线条宽的一半。

float miterLimit

当lineJoin属性为“miter”的时候，这个属性指定了斜连接长度和线条宽度的最大比率。参阅这一属性的单独参考页来了解更多细节。

float shadowBlur

指定羽化阴影的程度。默认值是0。阴影效果得到 Safari 的支持，但是并没有得到 Firefox 1.5 或 Opera 9 的支持。

String shadowColor

把阴影的颜色指定为一个 CSS 字符串或者 Web 样式字符串，并且可以包含一个 alpha 部分来表示透明度。默认值是 black。阴影效果得到 Safari 的支持，但是并没有得到 Firefox 1.5 或 Opera 9 的支持。

float shadowOffsetX, shadowOffsetY

指定阴影的水平偏移和垂直偏移。较大的值使得阴影化的对象似乎漂浮在背景的较高的位置上。默认值是0。阴影效果得到 Safari 的支持，但是并没有得到 Firefox 1.5 或 Opera 9 的支持。

Object strokeStyle

指定了用于画笔（绘制）路径的颜色、模式和渐变。这个属性可能是一个字符串，或者一个 CanvasGradient 对象或 CanvasPattern 对象。如果是一个字符串，它被解析为一个 CSS 颜色值，并且画笔用所得的实色来绘制。如果这个属性的值是一个 CanvasGradient 对象或 CanvasPattern 对象，画笔使用这个渐变或模式来实现。参阅 CanvasRenderingContext2D.createLinearGradient()、CanvasRenderingContext2D.createRadialGradient() 和 CanvasRenderingContext2D.createPattern()。

方法

arc()

用一个中心点和半径，为一个画布的当前子路径添加一条弧线。

arcTo()

使用目标点和一个半径，为当前的子路径添加一条弧线。

beginPath()

开始一个画布中的一条新路径（或者子路径的一个集合）。

`bezierCurveTo()`

为当前的子路径添加一个三次贝塞尔曲线。

`clearRect()`

在一个画布的一个矩形区域中清除掉像素。

`clip()`

使用当前路径作为连续绘制操作的剪切区域。

`closePath()`

如果当前子路径是打开的，就关闭它。

`createLinearGradient()`

返回代表线性颜色渐变的一个 `CanvasGradient` 对象。

`createPattern()`

返回代表贴图图像的一个 `CanvasPattern` 对象。

`createRadialGradient()`

返回代表放射渐变的一个 `CanvasGradient` 对象。

`drawImage()`

绘制一幅图像。

`fill()`

使用 `fillStyle` 属性所指定的颜色、渐变或模式来绘制或填充当前路径的内部。

`fillRect()`

绘制或填充一个矩形。

`lineTo()`

为当前的子路径添加一条直线线段。

`moveTo()`

设置当前位置并开始一条新的子路径。

`quadraticCurveTo()`

为当前路径添加一条贝塞尔曲线。

`rect()`

为当前路径添加一条矩形子路径。

`restore()`

将画布重置为最近保存的图形状态。

`rotate()`

旋转画布。

`save()`

保存 `CanvasRenderingContext2D` 对象的属性、剪切区域和变换矩阵。

`scale()`

标注画布的用户坐标系统。

`stroke()`

沿着当前路径绘制或画一条直线。这条直线根据 `lineWidth`、`lineJoin`、`lineCap`、`strokeStyle` 属性绘制。

`strokeRect()`

绘制（但并不填充）一个矩形。

`translate()`

转换画布的用户坐标系统。

描述

`CanvasRenderingContext2D` 对象提供了一组用来在画布上绘制的图形函数。尽管对函数的说明文档被忽略了，但是可用的函数非常丰富。它们可以分为几类。

绘制矩形

可以使用 `strokeRect()` 和 `fillRect()` 来绘制矩形的边框和填充矩形。此外，可以使用 `clearRect()` 来清除矩形所定义的区域。

绘制图像

在 Canvas API 中，图像通过表示 HTML `` 元素的 `Image` 对象来指定，或者通过使用 `Image()` 构造函数所创建的屏幕外图像来指定（参阅 `Image` 参考页了解详细内容）。一个画布对象也可以用作一个图像来源。

可以使用 `drawImage()` 方法在一个画布上绘制图像；而更为常见的形式是，允许源图像的任意矩形区域缩放或绘制到画布中。

创建和渲染路径

画布的一项强大功能就是，它能够从基本的绘图操作来构建图形，然后，绘制这些图形的帧（勾勒它们）或者给这些图形的内容涂色（填充它们）。累积起来的操作统一叫做当前路径。一个画布只保持一条当前路径。

为了构建一个多条线段所构成的一个连接的封闭形状，绘制操作之间需要有一个连接点。为此，画布保存了一个当前位置。画布绘制操作显式地使用这个位置作为它们的起始点，并且更新它直到到达终点。可以把这看作是使用钢笔和纸来绘画：当你完成了一条具体的线段或曲线，当前位置就是完成这一操作之后钢笔所停留的点。

可以在当前路径中创建一系列相互不连接的形状，它们都使用同样的绘制参数一起渲染。要分隔开这些形状，使用 `moveTo()` 方法，这个方法把当前的位置移动到一个新的位置而不添加一条连接线段。当你这么做的时候，就创建了一条新的子路径，这是用来表示一组连接起来的操作的画布术语。

一旦你所想要的路径形成了，可以使用 `stroke()` 绘制其边框，使用 `fill()` 来绘制其内容；或者两件事情都做。

可用的图形操作有：用来绘制直线的 `lineTo()`，用于绘制矩形的 `rect()`，用于绘制部分圆形的 `arc()` 或 `arcTo()`，以及用于绘制曲线的 `bezierCurveTo()` 或 `quadraticCurveTo()`。

除了勾勒和填充，还可以使用当前路径来指定绘制时画布所使用的剪切区域。这个区域中的像素是显示的，区域之外的像素是不显示的。剪切区域是累加性的；调用 `clip()` 可以将当前路径和当前绘制区域取交集，产生一个新的区域。不幸的是，没有直接的方法把当前的剪切区域设置为画布的范围；要做到这一点，必须保存和恢复画布的整个图形状态（稍后在这个条目中描述）。

如果任何子路径中的线段没有形成一个闭合的图形，`fill()` 和 `clip()` 操作通过添加一条从子路径的起点到终点的、虚拟（勾勒的时候看不到）线段来闭合它。可选的是，也可以调用 `closePath()` 来显式地添加这条线段。

颜色、渐变和模式

在填充和勾勒路径的时候，可以用 `fillStyle` 和 `strokeStyle` 属性来指定线段或者绘制区域如何绘制。CSS 样式颜色字符串，以及描述渐变和模式的 `CanvasGradient` 和 `CanvasPattern` 都是可以接受的。要创建一个渐变，使用 `createLinearGradient()` 或 `createRadialGradient()` 方法。要创建一个模式，使用 `createPattern()`。

要用 CSS 表示法来指定不透明的颜色，就采用“#RRGGBB”形式的字符串，其中 RR、GG 和 BB 分别是指定颜色的红色、绿色和蓝色成分的十六进制数字，其值都在 00 和 FF 之间。例如，完全红色的值是“#FF0000”。要指定部分透明的颜色，使用一个“rgba(R,G,B,A)”形式的字符串。在这种形式中，R、G 和 B 将颜色的红色、绿色和蓝色成分指定为 0 到 255 之间的十进制整数，并且 A 把 alpha（不透明）成分指定为 0.0（完全透明）到 1.0（完全不透明）之间的一个浮点数值。例如，半透明的完全红色为“rgba(255,0,0,0.5)”。

线条宽度、线帽和线条连接

画布为调整各种线条显示提供了几个选项。可以使用 `lineWidth` 属性来指定线条的宽度，用 `lineCap` 属性来指定线条的端点如何绘制，并且用 `lineJoin` 属性来指定线条如何连接。

坐标空间和转换

默认情况下，一个画布的坐标空间使用画布的左上角的 (0,0) 作为原点，x 值向右增加，y 值向下增加。这个坐标空间中的一个单位通常转换为一个像素。

然而，可以转换坐标空间，产生你在绘图操作中所指定的用来移动、缩放或旋转的任何坐标或范围。这通过 `translate()`、`scale()` 和 `rotate()` 方法来实现，它们会对画布的变换矩阵产生影响。由于坐标空间可以像这样变换，你传递给 `lineTo()` 这样方法的坐标可能无法用像素来度量。因此，Canvas API 使用浮点数而不是整数。

变换按照它们被指定的顺序相反的顺序来处理。例如，调用 `scale()` 之后，紧接着调用 `translate()`，这会首先变换坐标系统，然后再缩放。

组合

通常，图形是一个绘制于另一个的上面，新的图形使得在它之前绘制在其下方的图形变得模糊。这是一个画布中的默认行为。然而，你可以通过为 `globalCompositeOperation` 属性指定不同的值来执行很多有趣的操作，范围包括从 XOR 操作到增亮或减暗图形区域。参阅 `CanvasRenderingContext2D.globalCompositeOperation` 了解所有可能的选项。

阴影

Canvas API 包含了可以自动为你所绘制的任何图形添加下拉阴影的属性。然而，在编写本书的时候，Safari 是唯一的实现了这一 API 的浏览器。阴影的颜色可以使用 `shadowColor` 来指定，并且其可以通过 `shadowOffsetX` 和 `shadowOffsetY` 来改变。另外，应用到阴影边缘的羽化量也可以用 `shadowBlur` 来设置。

保存图形状态

`save()` 和 `restore()` 方法允许你保存和恢复一个 `CanvasRenderingContext2D` 对象的状态。`save()` 把当前状态推入到栈中，而 `restore()` 从栈的顶端弹出最近保存的状态，并且根据这些存储的值来设置当前绘图状态。

`CanvasRenderingContext2D` 对象的所有属性（除了画布的属性是一个常量）都是保存的状态的一部分。变换矩阵和剪切区域也是这个状态的一部分，但是当前路径和当前点并不是。

参阅

Canvas

CanvasRenderingContext2D.arc()

使用一个中心点和半径，为一个画布的当前子路径添加一条弧

摘要

```
void arc(float x, float y, float radius,  
         float startAngle, endAngle,  
         boolean counterclockwise)
```

参数

x, y

描述弧的圆形的圆心的坐标。

radius

描述弧的圆形的半径。

startAngle, endAngle

沿着圆指定弧的开始点和结束点的一个角度。这个角度用弧度来衡量。沿着 X 轴正半轴的三点钟方向的角度为 0，角度沿着逆时针方向而增加。

counterclockwise

弧是沿着圆周的逆时针方向 (*true*) 还是顺时针方向 (*false*) 遍历。

描述

这个方法头 5 个参数指定了圆周的一个起始点和结束点。调用这个方法会在当前点和当前子路径的起始点之间添加一条直线。接下来，它沿着圆周，在子路径的起始点和结束点之间添加弧。最后一个参数指定了圆应该沿着哪个方向遍历来连接起始点和结束点。这个方法将当前位置设置为弧的终点。

参阅

`CanvasRenderingContext2D.arcTo()`

`CanvasRenderingContext2D.beginPath()`

`CanvasRenderingContext2D.closePath()`

`CanvasRenderingContext2D.arcTo()`

使用切点和一个半径，来为当前子路径添加一条圆弧

摘要

```
void arcTo(float x1, float y1,  
           float x2, float y2,  
           float radius)
```

参数

x1, y1

点 P1 的坐标。

x2, y2

点 P2 的坐标。

radius

定义圆弧的圆的半径。

描述

这个方法为当前的子路径添加了一条圆弧，但是，它所描述的这条圆弧和 `arc()` 方法所描述的圆弧大不相同。添加给路径的圆弧是具有指定 *radius* 的圆的一部分。该圆弧有一个点与从当前位置到 P1 的线段相切，还有一个点和从 P1 到 P2 的线段相切。这两个切点就是圆弧的起点和终点，圆弧绘制的方向就是连接这两个点的最短圆弧的方向。

在很多常见的应用中，圆弧开始于当前位置而结束于 P2，但情况并不总是这样。如果当前的位置和圆弧的起点不同，这个方法添加了一条从当前位置到圆弧起点的直线。这个方法总是将当前位置设置为圆弧的终点。

例子

可以绘制一个矩形的右上角，用如下的代码给它一个圆角：

```
c.moveTo(10,10);           // start at upper left
c.lineTo(90, 10)           // horizontal line to start of round corner
c.arcTo(100, 10, 100, 20, 10); // rounded corner
c.lineTo(100, 100);        // vertical line down to lower right
```

Bug

这个方法在 Firefox 1.5 中未实现。

参阅

`CanvasRenderingContext2D.arc()`

`CanvasRenderingContext2D.beginPath()`

在一个画布中开始子路径的一个新的集合

摘要

```
void beginPath()
```

描述

`beginPath()` 丢弃任何当前定义的路径并且开始一条新的路径。它把当前的点设置为 (0,0)。

当一个画布的环境第一次创建，`beginPath()` 被显式地调用。

参阅

`CanvasRenderingContext2D.closePath()`

`CanvasRenderingContext2D.fill()`

`CanvasRenderingContext2D.stroke()`

第 22 章

`CanvasRenderingContext2D.bezierCurveTo()`

为当前子路径添加一条三次贝塞尔曲线

摘要

```
void bezierCurveTo(float cpX1, float cpY1,
                  float cpX2, float cpY2,
                  float x, float y)
```

..

参数

cpX1, cpY1

和曲线的开始点（当前位置）相关联的控制点的坐标。

cpX2, cpY2

和曲线的结束点相关联的控制点的坐标。

x, y

曲线的结束点的坐标。

描述

`bezierCurveTo()` 为一个画布的当前子路径添加一条三次贝塞尔曲线。这条曲线的开始点是画布的当前点，而结束点是 (x, y) 。两个贝塞尔控制点定义了 $(cpX1, cpY1)$ 和 $(cpX2, cpY2)$ 定义了曲线的形状。当这个方法返回的时候，当前的位置为 (x, y) 。

参阅

`CanvasRenderingContext2D.quadraticCurveTo()`

CanvasRenderingContext2D.clearRect()

擦除一个画布的矩形区域

摘要

```
void clearRect(float x, float y,  
               float width, float height)
```

参数

x, y

矩形的左上角的坐标。

width, height

矩形的尺寸。

描述

`clearRect()` 擦除了指定的矩形，用一个透明的颜色填充它。

CanvasRenderingContext2D.clip()

设置一个画布的剪切路径

摘要

```
void clip()
```

描述

这个方法用当前剪切路径来剪切当前路径，然后使用剪切后的路径作为新的剪切路径。注意，没有方法来扩大剪切路径。如果想要一个临时的剪切路径，应该先调用 `save()` 以便使用 `restore()` 恢复最初的剪切路径。一个画布的默认剪切路径就是画布的矩形自身。这个方法重新设置了当前路径以使其为空。

CanvasRenderingContext2D.closePath()

关闭一条打开的子路径

摘要

```
void closePath()
```

描述

如果画布的当前子路径是打开的，`closePath()` 通过添加一条线条连接当前点和子路径起始点来关闭它。如果子路径已经闭合了，这个方法不做任何事情。一旦子路径闭合，就不能再为其添加更多的直线或曲线了。要继续向该路径添加，需要通过调用 `moveTo()` 开始一条新的子路径。

不需要在勾勒或填充一条路径之前调用 `closePath()`。当填充的时候（并且当你调用 `clip()` 的时候），路径是隐式闭合的。

参阅

```
CanvasRenderingContext2D.beginPath()  
CanvasRenderingContext2D.moveTo()  
CanvasRenderingContext2D.stroke()  
CanvasRenderingContext2D.fill()
```

CanvasRenderingContext2D.createLinearGradient()

创建一条线性颜色渐变

摘要

```
CanvasGradient createLinearGradient(float xStart, float yStart,  
                                   float xEnd, float yEnd)
```

参数

xStart, yStart

渐变的起始点的坐标。

xEnd, yEnd

渐变的结束点的坐标。

返回值

表示一条线性颜色渐变的一个 `CanvasGradient` 对象。

描述

这个方法创建并返回了一个新的 `CanvasGradient` 对象，它在指定的起始点和结束点之间线性地内插颜色值。注意，这个方法并没有为渐变指定任何颜色。使用返回对象的 `addColorStop()` 来做到这一点。要使用一个渐变来勾勒线条或填充区域，只需要把 `CanvasGradient` 对象赋给 `strokeStyle` 或 `fillStyle` 属性即可。

参见

`CanvasGradient.addColorStop()`、`CanvasRenderingContext2D.createRadialGradient()`

`CanvasRenderingContext2D.createPattern()`

为贴图图像创建一个模式

摘要

```
CanvasPattern createPattern(Image image,  
                             String repetitionStyle)
```

参数

image

需要贴图的图像。这个参数通常是一个 `Image` 对象，但是也可以使用一个 `Canvas` 元素。

repetitionStyle

说明图像如何贴图。可能的值如下所示：

值	含义
"repeat"	在各个方向上都对图像贴图。这是默认值
"repeat-x"	只在 X 方向上贴图
"repeat-y"	只在 Y 方向上贴图
"no-repeat"	不贴图，只使用它一次

返回值

表示模式的一个 `CanvasPattern` 对象。

描述

这个方法创建并返回了一个 `CanvasPattern` 对象，该对象表示一个贴图图像所定义的模式。

要使用一个模式来勾勒线条或者填充区域，可以把一个 `CanvasPattern` 对象用作 `strokeStyle` 或 `fillStyle` 属性的值。

Bug

Firefox 1.5 只支持 “repeat” 方式，其他的都被忽略。

参阅

`CanvasPattern`

`CanvasRenderingContext2D.createRadialGradient()`

创建一个放射颜色渐变

摘要

```
CanvasGradient createRadialGradient(float xStart, float yStart, float radiusStart,  
                                     float xEnd, float yEnd, float radiusEnd)
```

参数

xStart, yStart

开始圆的圆心的坐标。

radiusStart

开始圆的半径。

xEnd, yEnd

结束圆的圆心的坐标。

radiusEnd

结束圆的半径。

返回值

表示一个放射性颜色渐变的 `CanvasGradient` 对象。

描述

这个方法创建和返回一个新的 `CanvasGradient` 对象，该对象在两个指定圆的圆周之间放射性地插值颜色。注意，这个方法并没有指定任何用来渐变的颜色。使用返回对象的 `addColorStop()` 方法来做到这一点。要使用渐变来勾勒线条或者填充区域，只需要把一个 `CanvasGradient` 赋给 `strokeStyle` 或 `fillStyle` 属性即可。

放射性渐变这样绘制：使用第一个圆的圆周在偏移0处的颜色和第二个圆的圆周在偏移1处的颜色，在两个位置之间的圆上插入颜色值（红色、绿色、蓝色和 alpha）。

参阅

`CanvasGradient.addColorStop()`、`CanvasRenderingContext2D.createLinearGradient()`

CanvasRenderingContext2D.drawImage()

绘制一幅图像

摘要

```
void drawImage(Image image, float x, float y)
void drawImage(Image image, float x, float y,
               float width, float height)
void drawImage(Image image, integer sourceX, integer sourceY,
               integer sourceWidth, integer sourceHeight,
               float destX, float destY,
               float destWidth, float destHeight)
```

参数

image

所要绘制的图像。这必须是表示一个标记或者一个屏幕外图像的Image对象，或者是一个Canvas对象。

x, y

要绘制的图像的左上角的位置。

width, height

图像所应该绘制的尺寸。指定这些参数使得图像可以缩放。

sourceX, sourceY

图像将要被绘制的区域的左上角。这些整数参数用图像像素来度量。

sourceWidth, sourceHeight

图像所要绘制区域的大小，用图像像素表示。

destX, destY

所要绘制的图像区域的左上角的画布坐标。

destWidth, destHeight

图像区域所要绘制的画布大小。

描述

这个方法有3个变形。第一个变形把整个图像复制到画布，将其放置到指定点的左上角，并且将每个图像像素映射成画布坐标系统的一个单元。第二个变形也把整个图像复制到画布，但是允许你用画布单位来指定想要的图像的宽度和高度。第三个变形则是完全通用的，它允许你指定图像的任何矩形区域并复制它，对画布中的任何位置都可进行任意的缩放。

传递给这个方法的图像必须是Image对象或Canvas对象。一个Image对象能够表示文档中的一个标记或者使用Image()构造函数所创建的一个屏幕外图像。

参阅

Image

CanvasRenderingContext2D.fill()

填充路径

摘要

```
void fill()
```

描述

`fill()` 使用 `fillStyle` 属性所指定的颜色、渐变和模式来填充当前路径。这一路径的每一条子路径都单独填充。任何未闭合的子路径都被填充，就好像已经对它们调用了 `closePath()` 方法一样（但是，注意，实际上并没有这么让子路径成为闭合的）。

画布使用“非零匝数规则”来确定哪个点在路径的内部，而哪个点在路径的外部。这一规则的细节超出了本书的范围，但是，它们通常只和那些与自身相交的复杂路径相关。

填充一条路径并不会清除该路径。你可以在调用 `fill()` 之后再次调用 `stroke()`，而不需要重新定义该路径。

参阅

CanvasRenderingContext2D.fillRect()

CanvasRenderingContext2D.fillRect()

填充一个矩形

摘要

```
void fillRect(float x, float y,  
              float width, float height)
```

参数

x, y

矩形的左上角的坐标。

width, height

矩形的大小。

描述

`fillRect()` 使用 `fillStyle` 属性所指定的颜色、渐变和模式来填充指定的矩形。

`fillRect()` 的当前实现还清除了路径，就好像 `beginPath()` 已经调用了。这一令人惊讶的行为可能不会标准化，因此不应该指望它。

参阅

`CanvasRenderingContext2D.fill()`
`CanvasRenderingContext2D.rect()`
`CanvasRenderingContext2D.strokeRect()`

CanvasRenderingContext2D.globalCompositeOperation

说明如何在画布上组合颜色

摘要

`String globalCompositeOperation`

描述

这个属性说明了绘制到画布上的颜色是如何与画布上已有的颜色组合（或“合成”）的。下面的表格列出了可能的值及其含义。这些值中的“source”一词，指的是将要绘制到画布上的颜色，而“destination”指的是画布上已经存在的颜色。默认值是“source-over”。

值	含义
“copy”	只绘制新的图形，删除其他所有内容
“darker”	在图形重叠的地方，颜色由两个颜色值相减后决定
“destination- atop”	已有的内容只有在它和新的图形重叠的地方保留。新图形绘制于内容之后
“destination- in”	在新图形以及已有画布内容重叠的地方，已有内容都保留。所有其他内容成为透明的。
“destination- out”	在已有内容和新图形不重叠的地方，已有内容保留。所有其他内容成为透明的
“destination- over”	新图形绘制于已有内容的后面。
“lighter”	在图形重叠的地方，颜色有两种颜色值的加值来决定
“source- atop”	只有在新图形和已有内容重叠的地方，才绘制新图形
“source- in”	在新图形以及已有内容重叠的地方，新图形才绘制。所有其他内容成为透明的
“source- out”	只有在和已有图形不重叠的地方，才绘制新图形
“source- over”	新图形绘制于已有图形的顶部。这是默认的行为
“xor”	在重叠和正常绘制的其他地方，图形都成为透明的

Bug

Firefox 1.5 不支持“copy”值或“darker”值

CanvasRenderingContext2D.lineCap

指定线段的末端如何绘制

摘要

String lineCap

描述

lineCap 属性指定线段如何结束。只有绘制较宽线段时它才有效。这个属性的合法值如下表所示。默认值是“butt”。

值	含义
“butt”	这个默认值指定了线段应该没有线帽。线条的末点是平直的而且和线条的方向正交，这条线段在其端点之外没有扩展
“round”	这个值指定了线段应该带有一个半圆形的线帽，半圆的直径等于线段的宽度，并且线段在端点之外扩展了线段宽度的一半
“square”	这个值表示线段应该带有一个矩形线帽。这个值和“butt”一样，但是线段扩展了自己的宽度的一半

Bug

Firefox 1.5 没有正确地实现“butt”线帽样式。Butt 线帽被当作是“square”线帽一样地绘制。

参阅

CanvasRenderingContext2D.lineJoin

CanvasRenderingContext2D.lineJoin

说明如何绘制交点

摘要

String lineJoin

描述

当一条路径包含了线段或曲线相交的交点的时候，lineJoin 属性说明如何绘制这些交点。只有当绘制具有宽度的线条的时候，这一属性的效果才能表现出来。

这一属性的默认值是“miter”，它说明了两条线段的外边缘一直扩展到它们相交。当两条线段以一个锐角相交，斜角连接可能变得很长。miterLimit 属性为一个斜面的长度设置了上限。超过这一限制，斜面就变成斜角了。

值“round”说明定点的外边缘应该和一个填充的弧接合，这个弧的直径等于线段的宽度。“bevel”值说明顶点的外边缘应该和一个填充的三角形相交。

Bug

Firefox 1.5 并没有正确地实现斜切连接，而是把它们当作圆连接来绘制。另外，当采用一种部分透明的颜色来勾勒的时候，斜角连接也不能正确地显示。

参阅

`CanvasRenderingContext2D.lineCap`、`CanvasRenderingContext2D.miterLimit`

CanvasRenderingContext2D.lineTo()

为当前子路径添加一条直线

摘要

```
void lineTo(float x, float y)
```

参数

`x`, `y`

直线的终点的坐标。

描述

`lineTo()` 为当前子路径添加一条直线。这条直线从当前点开始，到 (x, y) 结束。当方法返回时，当前点是 (x, y) 。

参阅

`CanvasRenderingContext2D.beginPath()`、`CanvasRenderingContext2D.moveTo()`

CanvasRenderingContext2D.miterLimit

最大斜面长度和线条宽度的比例

摘要

```
float miterLimit
```

描述

当宽线条使用设置为“miter”的 `lineJoin` 属性绘制并且两条线段以锐角相交的时候，所得的斜面可能相当长。当斜面太长，就会变得不协调。`miterLimit` 属性为斜面的长度设置了一个上限。这个属性表示斜面长度和线条宽度的比值。默认值是 10，意味着一个斜面的长度不应该超过线条宽度的 10 倍。如果斜面达到这个长度，它就变成斜角了。当 `lineJoin` 为“round”或“bevel”的时候，这一属性无效。

Bug

Firefox 1.5 没有正确地实现这一属性。当一个斜角连接超过了 `miterLimit`，连接会转换为圆连接。

参阅

`CanvasRenderingContext2D.lineJoin`

CanvasRenderingContext2D.moveTo()

设置当前位置并开始一条新的子路径

摘要

```
void moveTo(float x, float y)
```

参数

x, *y*

新的当前点的坐标。

描述

`moveTo()` 将当前位置设置为 (*x*, *y*) 并用它作为第一点创建一条新的子路径。如果之前有一条子路径并且它包含刚才的那一点，那么从路径中删除该子路径。

参阅

`CanvasRenderingContext2D.beginPath()`

CanvasRenderingContext2D.quadraticCurveTo()

为当前子路径添加一条贝塞尔曲线

摘要

```
void quadraticCurveTo(float cpX, float cpY,  
                      float x, float y)
```

参数

cpX, *cpY*

控制点的坐标。

x, *y*

曲线终点的坐标。

描述

这个方法为当前的子路径添加一条贝塞尔曲线。这条曲线从当前点开始，到 (*x*, *y*) 结束。

控制点 (*cpX*, *cpY*) 说明了这两点之间的曲线的形状 (贝塞尔曲线的数学原理超出了本书的范围)。当这个方法返回的时候, 当前位置是 (*x*, *y*)。

Bug

Firefox 1.5 没有正确地实现这一方法。

参阅

`CanvasRenderingContext2D.bezierCurveTo()`

CanvasRenderingContext2D.rect()

为路径添加一个矩形子路径

摘要

```
void rect(float x, float y,  
          float width, float height)
```

参数

x, *y*

矩形的左上角的坐标。

width, *height*

矩形的大小。

描述

这个方法为路径添加了一个矩形。这个矩形是路径的一个子路径并且没有和路径中的任何其他子路径相连。当这个方法返回的时候, 当前位置是 (0, 0)。

参阅

`CanvasRenderingContext2D.fillRect()`, `CanvasRenderingContext2D.strokeRect()`

CanvasRenderingContext2D.restore()

将绘图状态置为保存值

摘要

```
void restore()
```

描述

这个方法从栈中弹出存储的图形状态并恢复 `CanvasRenderingContext2D` 属性、剪切路径和变换矩阵的值。参阅 `save()` 方法了解更多信息。

Bug

Firefox 1.5 不能正确地存储和恢复 `strokeStyle` 属性。

参阅

`CanvasRenderingContext2D.save()`

CanvasRenderingContext2D.rotate()

旋转画布的坐标系

摘要

```
void rotate(float angle)
```

参数

angle

旋转的量，用弧度表示。正值表示顺时针方向旋转，负值表示逆时针方向旋转。

描述

这个方法通过指定一个角度，改变了画布坐标和Web浏览器中的<canvas>元素的像素之间的映射，使得任意后续绘图在画布中都显示为旋转的。它并没有旋转<canvas>元素本身。注意，这个角度是用弧度来指定的。要把角度转换为弧度，乘以`Math.PI`并除以180。

参阅

`CanvasRenderingContext2D.scale()`、`CanvasRenderingContext2D.translate()`

CanvasRenderingContext2D.save()

保存当前图形状态的一份拷贝

摘要

```
void save()
```

描述

`save()` 把当前状态的一份拷贝压入到一个保存图形状态的栈中。这就允许你临时地改变图形状态，然后，通过调用 `restore()` 来恢复以前的值。

一个画布的图形状态包含了 `CanvasRenderingContext2D` 对象的所有属性（除了只读的画布属性以外）。它还包含了一个变换矩阵，该矩阵是调用 `rotate()`、`scale()` 和 `translate()` 的结果。另外，它包含了剪切路径，该路径通过 `clip()` 方法指定。可是要注意，当前路径和当前位置并非图形状态的一部分，并且不会由这个方法保存。

Bug

Firefox 1.5 无法保存和恢复 `strokeStyle` 属性。

参阅

`CanvasRenderingContext2D.restore()`

CanvasRenderingContext2D.scale()

缩放画布的用户坐标系

摘要

```
void scale(float sx, float sy)
```

参数

sx, *sy*

水平和垂直的缩放因子

描述

`scale()` 为画布的当前变换矩阵添加一个缩放变换。缩放通过独立的水平和垂直缩放因子来完成。例如，传递一个值 2.0 和 0.5 将会导致绘图路径宽度变为原来的两倍，而高度变为原来的 1/2。指定一个负的 *sx* 值，会导致 X 坐标沿 Y 轴对折，而指定一个负的 *sy* 会导致 Y 坐标沿着 X 轴对折。

参阅

`CanvasRenderingContext2D.rotate()`、`CanvasRenderingContext2D.translate()`

CanvasRenderingContext2D.stroke()

绘制当前路径

摘要

```
void stroke()
```

描述

`stroke()` 方法绘制当前路径的边框。路径定义的几何线条产生了，但线条的可视化取决于 `strokeStyle`、`lineWidth`、`lineCap`、`lineJoin` 和 `miterLimit` 等属性。

术语勾勒，指的是钢笔或笔刷的笔画。它意味着“画……轮廓”。和 `stroke()` 方法相对的是 `fill()`，该方法会填充路径的内部区域而不是勾勒出路径的边框。

参阅

`CanvasRenderingContext2D.fill()`
`CanvasRenderingContext2D.lineCap`
`CanvasRenderingContext2D.lineJoin`
`CanvasRenderingContext2D.strokeRect()`

CanvasRenderingContext2D.strokeRect()

绘制一个矩形

摘要

```
void strokeRect(float x, float y,  
                float width, float height)
```

参数

x, y
矩形的左上角的坐标。

width, height
矩形的大小。

描述

这个方法按照指定的位置和大小绘制一个矩形的边框(但并不填充矩形的内部)。线条颜色和线条宽度由 `strokeStyle` 和 `lineWidth` 属性指定。矩形边角的形状由 `lineJoin` 属性指定。

`strokeRect()` 的当前实现会清晰化路径,就好像 `beginPath()` 已经调用过了。这一令人吃惊的行为可能不会标准化,所以不能指望它。

参阅

`CanvasRenderingContext2D.fillRect()`
`CanvasRenderingContext2D.lineJoin`
`CanvasRenderingContext2D.rect()`
`CanvasRenderingContext2D.stroke()`

CanvasRenderingContext2D.translate()

转换画布的用户坐标系

摘要

```
void translate(float dx, float dy)
```

参数

dx, *dy*

转化的量的 X 和 Y 大小。

描述

`translate()` 为画布的变换矩阵添加水平的和垂直的偏移。参数 *dx* 和 *dy* 添加给后续定义路径中的所有点。

参阅

`CanvasRenderingContext2D.rotate()`、`CanvasRenderingContext2D.scale()`

CDATASection

1 级 DOM XML

XML 文档中的 CDATA 节

Node → CharacterData → Text → CDATASection

描述

这个常用的接口表示 XML 文档中的 CDATA 节。使用 HTML 文档的程序设计者绝不会遇到这种类型的节点，不需要使用该接口。

CDATASection 是 Text 接口的子接口，没有定义任何自己的属性和方法。通过从 Node 接口继承 `nodeValue` 属性，或通过从 CharacterData 接口继承 `data` 属性，可以访问 CDATA 节的文本内容。虽然通常可以把 CDATASection 节点作为 Text 节点处理，但要注意，`Node.normalize()` 方法不并入相邻的 CDATA 部分。使用 `Document.createCDATASection()` 来创建一个 CDATASection。

参阅

CharacterData、Text

CharacterData

1 级核心 DOM

Text 和 Comment 节点的常用功能

Node → CharacterData

子接口

Comment、Text

属性

String `data`

该节点包含的文本。

readonly unsigned long `length`

该节点包含的字符数。

方法

`appendData()`

把指定的字符串添加到该节点包含的文本上。

`deleteData()`

从该节点删除指定的文本，从指定位移量处的字符开始，包括其后指定数量的字符。

`insertData()`

把指定的字符串插入指定位移量处的节点文本。

`replaceData()`

用指定的字符串替换从指定位移量处开始，包括其后指定数量的字符。

`substringData()`

返回从指定位移量处的字符开始，包括其后指定数量的字符的文本副本。

描述

`CharacterData` 是 `Text` 节点和 `Comment` 节点的超接口。文档从不包含 `CharacterData` 节点，它们只包含 `Text` 节点和 `Comment` 节点。但由于这两种节点类型具有相似的功能，因此此处定义了这些函数，以便 `Text` 和 `Comment` 可以继承它。

注意，不一定非要使用此接口定义的字符串操作方法。`data` 属性是一个普通的 JavaScript 字符串，你可以使用 `+` 运算符来操作它进行字符串连接，而且可以对它使用各种 `String` 和 `RegExp` 方法。

参阅

`Comment`、`Text`

`CharacterData.appendData()`

1 级核心 DOM

把字符串附加到 `Text` 或 `Comment` 节点上

摘要

```
void appendData(String arg)
    throws DOMException;
```

参数

`arg`

要附加到 `Text` 或 `Comment` 节点的字符串。

抛出

如果调用该方法的节点是只读的，它将抛出具有代码为 `NO_MODIFICATION_ALLOWED_ERR` 的 `DOMException` 异常。

描述

该方法将把字符串 *arg* 附加到节点的 *data* 属性的末尾。

CharacterData.deleteData()

1 级核心 DOM

从 Text 或 Comment 节点删除文本

摘要

```
void deleteData(unsigned long offset,  
                unsigned long count)  
    throws DOMException;
```

参数

offset

要删除的第一个字符的位置。

count

要删除的字符的数量。

抛出

该方法可以抛出具有以下代码的 DOMException 异常：

INDEX_SIZE_ERR

参数 *offset* 或 *count* 是负数，或 *offset* 大于 Text 节点或 Comment 节点的长度。

NO_MODIFICATION_ALLOWED_ERR

节点是只读的，不能修改。

描述

该方法将从 *offset* 指定的字符开始，从 Text 或 Comment 节点中删除 *count* 个字符。如果 *offset* 加 *count* 大于 Text 或 Comment 节点中的字符数，那么删除从 *offset* 开始到字符串结尾的所有字符。

CharacterData.insertData()

1 级核心 DOM

把字符串插入 Text 节点或 Comment 节点

摘要

```
void insertData(unsigned long offset,  
                String arg)  
    throws DOMException;
```

参数

offset

要把字符串插入 Text 节点或 Comment 节点的字符位置。

arg

要插入的字符串。

抛出

该方法可以抛出具有以下代码的 DOMException 异常：

INDEX_SIZE_ERR

参数 *offset* 是负数，或 *offset* 大于 Text 节点或 Comment 节点的长度。

NO_MODIFICATION_ALLOWED_ERR

节点是只读的，不能修改。

描述

该方法将把指定的字符串 *arg* 插入指定位置 *offset* Text 节点或 Comment 节点的文本处。

CharacterData.replaceData()

1 级核心 DOM

用指定的字符串替换 Text 节点或 Comment 节点的字符

摘要

```
void replaceData(unsigned long offset,  
                 unsigned long count,  
                 String arg)  
    throws DOMException;
```

参数

offset

要替换的第一个字符在 Text 节点或 Comment 节点中的位置。

count

要替换的字符的数量。

arg

要替换 *offset* 和 *count* 指定的字符的字符串。

抛出

该方法可以抛出具有以下代码的 DOMException 异常：

INDEX_SIZE_ERR

参数 *offset* 是负数或大于 Text 节点或 Comment 节点的长度，或者 *count* 是负数。

NO_MODIFICATION_ALLOWED_ERR

节点是只读的，不能修改。

描述

该方法将用字符串 *arg* 替换从 *offset* 开始的 *count* 个字符。如果 *offset* 加 *count* 大于 Text 或 Comment 节点的长度，那么从 *offset* 开始的所有字符都将被替换。

注意，`insertData()` 方法和 `deleteData()` 方法都是这一方法的特例。

CharacterData.substringData()

1 级核心 DOM

从 Text 或 Comment 节点中提取子串

摘要

```
String substringData(unsigned long offset, ,  
                      unsigned long count)  
    throws DOMException;
```

参数

offset

要返回的第一个字符的位置。

count

要返回的子串中的字符数。

返回值

一个字符串，包含 Text 或 Comment 节点中从 *offset* 开始的 *count* 个字符。

抛出

该方法可以抛出具有以下代码的 DOMException 异常：

INDEX_SIZE_ERR

参数 *offset* 是负数或大于 Text 节点或 Comment 节点的长度，或者 *count* 是负数。

DOMSTRING_SIZE_ERR

指定的文本范围太长，在浏览器的 JavaScript 实现中不能填到一个字符串中。

描述

该方法将从 Text 或 Comment 节点中提取从 *offset* 开始的 *count* 个字符。只有当节点包含的文本的字符数大于浏览器的 JavaScript 实现中能填入的字符串的最大字符数，该方法才有用。在这种情况下，JavaScript 程序不能直接使用 Text 节点或 Comment 节点的 `data` 属性，而必须用节点文本的较短子串。在实际应用中，这种情况不太可能出现。

Checkbox

参阅 Input

Comment

1 级核心 DOM

HTML 或 XML 注释

Node → CharacterData → Comment

描述

Comment 节点表示 HTML 或 XML 文档中的注释。使用由 CharacterData 接口继承的 data 属性，或使用由 Node 接口继承的 nodeValue 属性，可以访问注释的内容（即 `<!--` 和 `-->` 之间的文本）。使用由 CharacterData 接口继承的各种方法可以操作注释的内容。使用 `Document.createComment()` 来创建一个注释对象。

参阅

CharacterData

CSS2Properties

2 级 DOM CSS2

所有 CSS2 性质及其值的集合

Object → CSS2 属性

属性

String cssText

这是一组样式属性及其值的文本表示。这个文本格式化为一个 CSS 样式表，去掉了包围属性和值的元素选择器的花括号。将这一属性设置为非法的值将会抛出一个代码为 SYNTAX_ERR 的 DOMException 异常。当 CSS2Properties 对象是只读的时候，试图设置这一属性将会抛出一个代码为 NO_MODIFICATION_ALLOWED_ERR 的 DOMException 异常。

除了 cssText 属性，CSS2Properties 对象还有针对浏览器所支持的每个 CSS 属性都有一个对应的属性。这些属性的名称与 CSS 的属性名紧密对应，但为了避免 JavaScript 中的语法错误而进行了一些改变。含有连字符的多词属性（如 font-family）在 JavaScript 中没有连字符，而是每个词的第一个字符大写（如 fontFamily）。此外，float 属性与保留字 float 冲突，所以被转换成 cssFloat。

和 CSS2 规范所定义的每个属性相对应的 CSS2Properties 属性名在下表中列出。但是，注意，有些浏览器并不支持所有的 CSS 属性，因而也可能并不支持列出的所有属性。由于这些属性直接对应于 CSS 性质，因此它们没有单独的说明。有关它们的含义和合法值，请参阅 CSS 的参考书，如《Cascading Style Sheets: The Definitive Guide》，由 Eric A. Meyer 著，O'Reilly 公司出版。所有属性都是字符串。设置这些属性，会抛出异常，其原因与调用 CSSText 属性相同。

azimuth	background	backgroundAttachment	backgroundColor
backgroundImage	backgroundPosition	backgroundRepeat	border
borderBottom	borderBottomColor	borderBottomStyle	borderBottomWidth

borderCollapse	borderColor	borderLeft	borderLeftColor
borderLeftStyle	borderLeftWidth	borderRight	borderRightColor
borderRightStyle	borderRightWidth	borderSpacing	borderStyle
borderTop	borderTopColor	borderTopStyle	borderTopWidth
borderWidth	bottom	captionSide	clear
clip	color	content	counterIncrement
counterReset	cssFloat	cue	cueAfter
cueBefore	cursor	direction	display
elevation	emptyCells	font	fontFamily
fontSize	fontSizeAdjust	fontStretch	fontStyle
fontVariant	fontWeight	height	left
letterSpacing	lineHeight	listStyle	listStyleImage
listStylePosition	listStyleType	margin	marginBottom
marginLeft	marginRight	marginTop	markerOffset
marks	maxHeight	maxWidth	minHeight
minWidth	orphans	outline	outlineColor
outlineStyle	outlineWidth	overflow	padding
paddingBottom	paddingLeft	paddingRight	paddingTop
page	pageBreakAfter	pageBreakBefore	pageBreakInside
pause	pauseAfter	pauseBefore	pitch
pitchRange	playDuring	position	quotes
richness	right	size	speak
speakHeader	speakNumeral	speakPunctuation	speechRate
stress	tableLayout	textAlign	textDecoration
textIndent	textShadow	textTransform	top
unicodeBidi	verticalAlign	visibility	voiceFamily
volume	whiteSpace	widows	width
wordSpacing	zIndex		

描述

CSS2Properties 对象表示一组 CSS 样式属性及其值。它为 CSS 规范所定义的每一个 CSS 属性都定义了一个 JavaScript 属性。一个 HTML 元素的 `style` 属性是一个可读写的 CSS2Properties 对象，就好像一个 CSSRule 对象的 `style` 属性一样。然而，`Window.getComputedStyle()` 的返回值是一个 CSS2Properties 对象，其属性是只读的。

参阅

CSSRule、HTMLElement、Window.getComputedStyle(), 第 16 章

CSSRule

2 级 DOM CSS, IE 5

CSS 样式表中的规则

Object → CSSRule

属性

String selectorText

这个选择器文本指定了这一样式规则所适用的文档元素。设置这一属性的时候，如果规则是只读的，将会导致一个代码为 `NO_MODIFICATION_ALLOWED_ERR` 的 `DOMException` 异常；如果新的值不允许 CSS 语法规则，将会导致代码为 `SYNTAX_ERR` 的 `DOMException` 异常。

readonly CSS2Properties style

要应用于 selectorText 所指定的元素的样式值。注意，尽管 style 属性本身是只读的，它所引用的 CSS2Properties 对象的这一属性是可读写的。

描述

CSSRule 对象表示 CSS 样式表中的一个规则：它表示了应用于特定的一组文档元素的样式信息。selectorText 是这一规则的元素选择器的字符串表示，而 style 是一个 CSS2Properties 对象，它表示应用于选择的元素的一组样式属性和值。

2 级 DOM CSS 规范实际上定义了 CSSRule 的一个多少有些复杂的层级，用来表示可以出现在一个 CSSStyleSheet 中各种类型的规则。这里的属性列表实际上是由 DOM CSSStyleRule 接口定义的。样式规则是样式表中最常见也是最重要的规则类型，并且这里列出的样式属性只是那些跨浏览器可移植使用的属性。IE 并没有很好地支持 2 级 DOM 规范（至少 IE7 的支持是不够的），但它确实实现了一个 CSSRule 对象，该对象支持这里列出的两个属性。

参阅

CSS2Properties、CSSStyleSheet

CSSStyleSheet

2 级 DOM CSS, IE 4

CSS 样式表

Object → CSSStyleSheet

属性

readonly CSSRule[] cssRules

这是一个只读的、类似数组的对象，保存了组成样式表的 CSSRule 对象。在 IE 中，使用 rule 属性来替代它。在 DOM 兼容的实现里，这个数组包含了代表样式表中所有规则的对象，包括 @import 指示符这样的 at 规则。这些规则实现了一个和 CSSRule 所描述的不同的接口。其他类型的规则对象没有得到浏览器的很好支持，也没有在本书中介绍。因此，注意，你必须测试这个数组中的所有条目，以确保它们在你尝试使用这些属性之前定义了这些 CSSRule 属性。

boolean disabled

如果为 true，样式表被关闭，并且不能应用于文档。如果为 false，样式表打开并且可以应用于文档。

readonly String href

一个样式表的 URL，它连接到文档，或者因为内联样式表而为 null。

readonly StyleSheet parentStyleSheet

包含这个样式表的一个样式表，如果样式表直接包含到一个文档中此属性为 null。

readonly CSSRule[] rules

IE 中和 DOM 标准的 cssRules[] 对等的数组。

readonly String title

如果指定的话，就是样式表的标题。一个标题可以通过引用该样式表的 `<style>` 或 `<link>` 元素的 `title` 属性来指定。

readonly String type

样式表的类型，以一个 MIME 类型表示。CSS 样式表的类型为 “text/css”。

方法

`addRule()`

为一个样式表添加一条 CSS 规则的特定于 IE 的方法。

`deleteRule()`

从指定位置删除规则的 DOM 标准方法。

`insertRule()`

向样式表中插入一条新规则的 DOM 标准方法。

`removeRule()`

删除一条规则的特定于 IE 的方法。

描述

该接口表示一个 CSS 样式表。它拥有用来关闭样式表以及查询、插入和删除样式表规则的属性和方法。IE 实现了和 DOM 标准略为不同的 API。在 IE 中，使用 `rules[]` 数组，而不是 `cssRules[]`，并且使用 `addRule()` 和 `removeRule()`，而不是 DOM 标准的 `insertRule()` 和 `deleteRule()`。

应用于一个文档的 `CSSStyleSheet` 对象是 `Document` 对象的 `styleSheets[]` 数组的成员。DOM 标准也要求：定义或连接到一个样式表的任何 `<style>` 或 `<link>` 元素或 `ProcessingInstruction` 节点，应该通过一个 `sheet` 属性让 `CSSStyleSheet` 对象可用（尽管在编写本书的时候，这一点还没有广泛实现）。

参阅

`CSSRule`、`Document` 对象的 `styleSheets[]` 属性，第 16 章

CSSStyleSheet.addRule()

IE 4

特定于 IE 的方法，向样式表中插入一条规则

摘要

```
void addRule(String selector,
             String style,
             integer index)
```

参数

selector

规则的 CSS 选择器。

style

应用于匹配该选择器的元素的样式。这个样式字符串是一个分号隔开的属性: 值对的列表。并没有使用花括号开始或结束。

index

规则数组中插入或附加规则的位置。如果这个可选参数被省略掉, 则新的规则会增加到规则数组的最后。

描述

这个方法在样式表的 `rules` 数组的指定 `index` 处插入 (或附加) 一条新的 CSS 样式规则。这是标准 `insertRule()` 方法的特定于 IE 的替代。注意, 这个方法的参数和 `insertRule()` 方法的参数不同。

CSSStyleSheet.deleteRule()

2 级 DOM CSS

从样式表中删除一个规则

摘要

```
void deleteRule(unsigned long index)
    throws DOMException;
```

参数

index

要删除的规则在 `cssRules` 数组中的下标。

抛出

如果 `index` 是负数或大于等于 `cssRules.length`, 该方法将抛出代码为 `INDEX_SIZE_ERR` 的 `DOMException` 异常。如果样式表是只读的, 它将抛出代码为 `NO_MODIFICATION_ALLOWED_ERR` 的 `DOMException` 异常。

描述

该方法将删除 `cssRules` 数组指定 `index` 处的规则。这是一个 DOM 标准方法, 参阅 `CSSStyleSheet.removeRule()`, 它是特定于 IE 的一个替代方法。

CSSStyleSheet.insertRule()2 级 DOM CSS

在样式表中插入一条规则

摘要

```
unsigned long insertRule(String rule,
                          unsigned long index)
    throws DOMException;
```

参数

rule

要添加到样式表的规则的完整的、可解析的文本表示。对于样式规则，该参数包括元素选择器和样式信息。

index

要把规则插入或附加到 `cssRules` 数组中的位置。

返回值

参数 *index* 的值。

抛出

该方法在下列情况下将抛出具有下列代码的 `DOMException` 异常：

`HIERARCHY_REQUEST_ERR`

CSS 语法不允许指定的规则出现在指定的位置。

`INDEX_SIZE_ERR`

index 是负数或大于 `cssRules.length` 的值。

`NO_MODIFICATION_ALLOWED_ERR`

该样式表是只读的。

`SYNTAX_ERR`

指定的 *rule* 文本具有语法错误。

描述

该方法将在样式表的 `cssRules` 数组的指定 *index* 处插入（或附加）新的 CSS *rule*。这是一个 DOM 标准方法，参阅 `CSSStyleSheet.addRule()`，它是特定于 IE 的一个替代方法。

CSSStyleSheet.removeRule()IE 4

特定于 IE 的方法，从一个样式表中移除一条规则

摘要

```
void removeRule(integer index)
```

参数

index

要移除的规则在 `rules[]` 数组中的下标。如果这个可选的参数省略掉了，数组的第一条规则被移除。

描述

这个方法从样式表的规则数组的指定 *index* 移除 CSS 样式规则。这是标准的 `deleteRule()` 方法的一个特定于 IE 的替代。

Document

1 级核心 DOM

HTML 文档或 XML 文档

Node → Document

子接口

HTMLDocument

属性

readonly Window defaultView

Web 浏览器的 Window 对象（DOM 中的术语叫做“视图”），文档在其中显示。

readonly DocumentType doctype

对于具有 `<!DOCTYPE>` 声明的 XML 文档来说，该属性声明了表示文档的 DTD 的 DocumentType 节点。对于没有 `<!DOCTYPE>` 声明的 HTML 文档和 XML 文档来说，该属性为 `null`。

readonly Element documentElement

对文档根元素的引用。对于 HTML 文档来说，该属性总是表示 `<html>` 标记的 Element 对象。通过从 Node 节点继承的 `childNodes[]` 数组，也可以访问这一根元素。参阅 HTMLDocument 的 `body` 属性。

readonly DOMImplementation implementation

表示创建该文档的实现的 DOMImplementation 对象。

readonly CSSStyleSheet[] styleSheets

表示嵌入或链接入文档的所有样式表的对象的集合。在 HTML 文档中，该集合通常包括 `<link>` 和 `<style>` 标记定义的样式表。

方法

addEventListener()

为这个文档的事件句柄集合添加一个事件句柄函数。这是除 IE 以外的所有的现代浏览器都支持的一个 DOM 标准方法。

`attachEvent()`

为这个文档的句柄集合添加一个事件句柄函数。这是 `addEventListener()` 方法的特定于 IE 的替代。

`createAttribute()`

用指定的名称创建新的 `Attr` 节点。

`createAttributeNS()`

用指定的名称和名字空间创建新的 `Attr` 节点。

`createCDATASection()`

创建包含指定文本的新 `CDATASection` 节点

`createComment()`

创建包含指定字符串的新 `Comment` 节点。

`createDocumentFragment()`

创建新的、空的 `DocumentFragment` 节点。

`createElement()`

用指定的标记名创建新的 `Element` 节点。

`createElementNS()`

用指定的标记名和名字空间创建新的 `Element` 节点。

`createEvent()`

创建一个新的合成 `Event` 对象。

`createExpression()`

创建一个代表编译过的 XPath 查询的新的 `XPathExpression` 对象。对于其特定于 IE 的替代，请参阅 `Node.selectNodes()`。

`createProcessingInstruction()`

用指定的标记和数据串创建新的 `ProcessingInstruction` 节点。

`createRange()`

创建一个新的 `Range` 对象。从技术上说，该方法是由 `DocumentRange` 接口定义的，只有在支持 `Range` 模块的实现中，`Document` 对象才定义了该方法。

`createTextNode()`

创建新的 `Text` 节点，表示指定的文本。

`detachEvent()`

从这个文档中移除一个事件句柄函数。这是标准的 `removeEventListener()` 方法的特定于 IE 的替代。

`dispatchEvent()`

为这个文档分派一个合成事件。

`evaluate()`

根据这个文档计算一个 XPath 查询。其特定于 IE 的替代请参阅 `Node.selectNodes()`。

`getElementById()`

返回该文档中具有指定 `id` 属性的子孙 `Element` 节点。如果文档中没有这样的 `Element` 节点，则返回 `null`。

`getElementsByTagName()`

返回文档中所有具有指定标记名的 `Element` 节点的数组（从技术上说，是 `NodeList`）。`Element` 节点出现在返回数组中的顺序就是它们在源文档中出现的顺序。

`getElementsByTagNameNS()`

返回所有具有指定标记名和名字空间的 `Element` 节点。

`importNode()`

对于其他适合插入该文档的文档，生成它的节点副本。

`loadXML()`（只适用于 IE）

解析 XML 标记的一个字符串，并将结果存储到这个文档对象中。

`removeEventListener()`

从这个文档的句柄集合中移除一个事件句柄函数。这是一个标准的 DOM 方法，除 IE 以外的所有现代浏览器都支持它。

描述

`Document` 接口是文档树的根节点。一个 `Document` 节点可以具有多个子节点，但它们中只能有一个 `Element` 节点，即它是文档的根元素。通过 `documentElement` 属性最容易访问根元素。使用 `doctype` 属性和 `implementation` 属性可以访问文档的 `DocumentType` 对象（如果存在的话）和 `DOMImplementation` 对象。

`Document` 接口定义的大多数方法都是“工厂方法”，用于创建可以插入文档的各种类型的节点。但 `getElementById()` 方法和 `getElementsByTagName()` 方法是例外，它们对查找文档树中指定的 `Element` 或相关 `Element` 节点的集合来说非常有用。其他的例外是像 `addEventListener()` 这样的事件句柄注册方法。这些和事件相关的方法也由 `Element` 接口定义了，并且也会在那里详细介绍。

你最常见到的情况是通过一个 `Window` 的 `document` 属性来获得一个 `Document` 对象。`Document` 对象也可以通过 `Frame` 和 `IFrame` 的 `contentDocument` 属性来获得，并且也可以通过添加到文档中的任何 `Node` 的 `ownerDocument` 属性获得。

如果你使用 XML（包括 XHTML），可以使用 `DOMImplementation` 的 `createDocument()` 方法来创建一个新的 `Document` 对象：

```
document.implementation.createDocument(namespaceURL, rootTagName, null);
```

在 IE 中，你可能会使用如下代码：

```
new ActiveXObject("MSXML2.DOMDocument");
```

参见例 21-1 来认识一个用来创建新的 `Document` 对象的跨平台的工具函数。

也有可能从网络载入一个 XML 文件，并且将它解析到一个 Document 对象中。参阅 XMLHttpRequest 对象的 responseXML 属性。你也可以把一个 XML 标记串解析为一个 Document 对象，参阅 DOMParser.parseFromString() 以及特定于 IE 的 Document.loadXML() (例 21-4 是一个跨平台的工具函数，它使用这些方法来解析 XML 标记)。

参阅 HTMLDocument 来了解专门用于 HTML 文档的其他属性和方法。

参阅

DOMImplementation、DOMParser、HTMLDocument、Window、XMLHttpRequest，第 15 章。

Document.addEventListener()

参阅 Element.addEventListener()

Document.attachEvent()

参阅 Element.attachEvent()

Document.createAttribute()

1 级核心 DOM

创建新的 Attr 节点

摘要

```
Attr createAttribute(String name)
    throws DOMException;
```

参数

name

新创建的属性的名称。

返回值

新创建的 Attr 节点，nodeName 属性设置为 *name*。

抛出

如果 *name* 含有不合法的字符，该方法将抛出代码为 INVALID_CHARACTER_ERR 的 DOMException 异常。

参阅

Attr、Element.setAttribute()、Element.setAttributeNode()

Document.createAttributeNS()2 级核心 DOM

创建具有指定的名称或名字空间的 Attr 节点

摘要

```
Attr createAttributeNS(String namespaceURI,  
                        String qualifiedName)  
    throws DOMException;
```

参数*namespaceURI*

Attr 的名字空间的惟一标识符。如果没有名字空间，则为 null。

qualifiedName

属性的限定名，应该包括名字空间前缀、冒号和本地名。

返回值

新创建的 Attr 节点，具有指定的名称和名字空间。

抛出

在下列环境中，该方法将抛出具有下列代码的 DOMException 异常：

INVALID_CHARACTER_ERR

qualifiedName 中含有不合法的字符。

NAMESPACE_ERR

qualifiedName 格式错误，或者 *qualifiedName* 与 *namespaceURI* 不匹配。

NOT_SUPPORTED_ERR

该实现不支持 XML 文档，因此没有实现该方法。

描述

createAttributeNS() 方法与 createAttribute() 方法相似，只是它创建的 Attr 节点除了具有指定的名称外，还具有指定的名字空间。只有使用名字空间的 XML 文档才会使用该方法。

Document.createCDATASection()1 级核心 DOM

创建新的 CDATASection 节点

摘要

```
CDATASection createCDATASection(String data)  
    throws DOMException;
```

参数

data

要创建的 CDATASection 的文本。

返回值

新创建的 CDATASection 节点，内容为指定的 *data*。

抛出

如果该文档是 HTML 文档，该方法将抛出代码为 NOT_SUPPORTED_ERR 的 DOMException 异常，因为 HTML 文档不支持 CDATASection 节点。

Document.createComment()

1 级核心 DOM

创建新的 Comment 节点

摘要

```
Comment createComment(String data);
```

参数

data

要创建的 Comment 节点的文本。

返回值

新创建的 Comment 节点，文本为指定的 *data*。

Document.createDocumentFragment()

1 级核心 DOM

创建新的、空的 DocumentFragment 节点

摘要

```
DocumentFragment createDocumentFragment();
```

返回值

新创建的 DocumentFragment 节点，没有子节点

Document.createElement()

1 级核心 DOM

创建新的 Element 节点

摘要

```
Element createElement(String tagName)  
    throws DOMException;
```

参数

tagName

要创建的 Element 的标记名。因为 HTML 不区分大小写，所以 HTML 的标记名可以采用任意的大小写形式。XML 标记名则是区分大小写的。

返回值

新创建的 Element 节点，具有指定的标记名。

抛出

如果 *tagName* 中含有不合法的字符，该方法将抛出代码为 `INVALID_CHARACTER_ERR` 的 `DOMException` 异常。

Document.createElementNS()

2 级核心 DOM

创建使用指定的名字空间的新 Element 节点

摘要

```
Element createElementNS(String namespaceURI,  
                        String qualifiedName)  
    throws DOMException;
```

参数

namespaceURI

新的 Element 的名字空间的惟一标识符。如果没有名字空间，则为 `null`。

qualifiedName

新的 Element 的限定名，应该包括名字空间前缀、冒号和本地名。

返回值

新创建的 Element 节点，具有指定的标记名称和名字空间。

抛出

在下列环境中，该方法将抛出具有下列代码的 `DOMException` 异常：

`INVALID_CHARACTER_ERR`

qualifiedName 中含有不合法的字符。

`NAMESPACE_ERR`

qualifiedName 格式错误，或者 *qualifiedName* 与 *namespaceURI* 不匹配。

`NOT_SUPPORTED_ERR`

该实现不支持 XML 文档，因此没有实现该方法。

描述

`createElementNS()` 方法与 `createElement()` 方法相似，只是它创建的 `Element` 节点除了具有指定的名称外，还具有指定的名字空间。只有使用名字空间的 XML 文档才会使用该方法。

Document.createEvent()

2 级 DOM Events

创建新的 `Event` 对象

摘要

```
Event createEvent(String eventType)
    throws DOMException;
```

参数

eventType

想获取的 `Event` 对象的事件模块名。关于有效的事件类型的列表，请参阅“描述”部分。

返回值

新创建的 `Event` 对象，具有指定的类型。

抛出

如果实现不支持需要的事件类型，该方法将抛出代码为 `NOT_SUPPORTED_ERR` 的 `DOMException` 异常。

描述

该方法将创建一种新的事件类型，该类型由参数 *eventType* 指定。注意，该参数的值不是要创建的事件接口的（单数）名称，而是定义那个接口的 DOM 模块的（复数）名称。下表列出了 *eventType* 的合法值和每个值创建的事件接口。

参数	事件接口	初始化方法
HTMLEvents	Event	initEvent()
MouseEvent	MouseEvent	initMouseEvent()
UIEvents	UIEvent	initUIEvent()

用该方法创建了 `Event` 对象后，必须用上表中所示的初始化方法初始化对象。关于初始化方法的详细信息，请参阅相关的 `Event` 接口参考页。

该方法实际上不是由 `Document` 接口定义，而是由 `DocumentEvent` 接口定义的。如果一个实现支持 `Event` 模块，那么 `Document` 对象就会实现 `DocumentEvent` 接口并支持该方法。

参阅

Event、MouseEvent、UIEvent

Document.createExpression()

Firefox 1.0, Safari 2.01, Opera 9

创建一个 XPath 表达式以供稍后计算

摘要

```
XPathExpression createExpression(String xpathText,  
                                Function namespaceURLMapper)  
    throws XPathException
```

参数

xpathText

表示要编译的 XPath 表达式的字符串。

namespaceURLMapper

从一个名字空间前缀映射到一个全称名字空间 URL 的一个函数。如果不需要这样的映射，则为 null。

返回值

一个 XPathExpression 对象。

抛出

如果 *xpathText* 包含一个语法错误，或者它使用了 *namespaceURLMapper* 无法解析的一个名字空间前缀，这个方法会抛出一个异常。

描述

这个方法接受表示 XPath 表达式的一个字符串，并且将其转换为一个编译过的表达式，即一个 XPathExpression。除了这个表达式，该方法还接受一个形如 `function(prefix)` 的函数，该函数解析一个名字空间前缀字符串，并返回一个全称名字空间 URL 字符串。

IE 不支持这个 API。参阅 `Node.selectNodes()` 了解一种特定于 IE 的替代方法。

参阅

`Document.evaluate()`、`Node.selectNodes()`、`XPathExpression`、`XPathResult`

Document.createProcessingInstruction()

1 级核心 DOM

创建 ProcessingInstruction 节点

摘要

```
ProcessingInstruction createProcessingInstruction(String target,  
                                                String data)  
    throws DOMException;
```

参数

target

处理指令的目标。

data

处理指令的内容文本。

返回值

新创建的 ProcessingInstruction 节点。

抛出

在下列环境中，该方法将抛具有下列代码的 DOMException 异常：

INVALID_CHARACTER_ERR

target 中含有不合法的字符。

NOT_SUPPORTED_ERR

这是一个 HTML 文档，不支持处理指令。

Document.createRange()

2 级 DOM Range

创建 Range 对象

摘要

```
Range createRange();
```

返回值

新创建的 Range 对象，两个边界点都被设置为文档的开头。

描述

该方法将创建一个 Range 对象，可以用来表示文档的一个区域或与该文档相关的 DocumentFragment 对象。

注意，该方法实际上不是由 Document 接口定义的，而是由 DocumentRange 接口定义的。如果一个实现支持 Range 模块，那么 Document 对象就会实现 DocumentRange 接口，并定义该方法。IE 6 不支持这个模块。

参阅

Range

Document.createTextNode()1 级核心 DOM

创建新的 Text 节点

摘要

```
Text createTextNode(String data);
```

参数

data

Text 节点的内容。

返回值

新创建的 Text 节点，表示指定的 *data* 字符串。

Document.detachEvent()

参阅 [Element.detachEvent\(\)](#)

Document.dispatchEvent()

参阅 [Element.dispatchEvent\(\)](#)

Document.evaluate()Firefox 1.0, Safari 2.01, Opera 9

计算一个 XPath 表达式

摘要

```
XPathResult evaluate(String xpathText,  
                      Node contextNode,  
                      Function namespaceURLMapper,  
                      short resultType,  
                      XPathResult result)  
    throws DOMException, XPathException
```

参数

xpathText

表示要计算的 XPath 表达式的字符串。

contextNode

文档中，对应要计算的表达式的节点。

namespaceURLMapper

把一个名字空间前缀映射为一个全称名字空间 URL 的函数。如果不需要这样的映射，就为 `null`。

resultType

指定了期待作为结果的对象的类型，使用 XPath 转换来强制结果类型。类型的可能的值是 XPathResult 对象所定义的常量。

result

一个复用的 XPathResult 对象；如果你要创建一个新的 XPathResult 对象对象，则为 null。

返回值

表示根据给定的 Context 节点计算的表达式的一个 XPathResult 对象。

抛出

如果 `xpathText` 包含一个语法错误，或者如果表达式的结果无法转换为想要的 *resultType*，或者如果表达式包含了 *namespaceURLMapper* 无法解析的名字空间，或者如果 *contextNode* 具有错误的类型或它和这个文档不相关，该方法都会抛出一个异常。

描述

这个方法根据给定的 Context 节点来计算指定的 XPath 表达式，并且返回一个 XPathResult 对象，该对象是 *type* 来确定结果类型应该是什么。如果你想要多次计算一个表达式，使用 `Document.createExpression()` 来把该表达式编译成一个 XPathExpression 对象，然后使用 XPathExpression 的 `evaluate()` 方法。

IE 不支持这个 API。参阅 `Node.selectNodes()` 和 `Node.selectSingleNode()` 来了解一个特定于 IE 的替代方法。

参阅

`Document.createExpression()`
`Node.selectNodes()`
`Node.selectSingleNode()`
`XPathExpression`
`XPathResult`

Document.getElementById()

2 级核心 DOM

查找具有指定的惟一 ID 的元素

摘要

```
Element getElementById(String elementId);
```

参数

elementId

想获取的元素的 id 属性的值。

返回值

表示具有指定的 `id` 属性的文档元素的 `Element` 节点。如果没有找到这样的元素，则返回 `null`。

描述

该方法将检索 `id` 属性的值为 `elementId` 的 `Element` 节点，并将它返回。如果没有找到这样的元素 `Element`，它将返回 `null`。`id` 属性的值在文档中是惟一的，如果该方法找到多个具有指定 `elementId` 的 `Element` 节点，它将随机返回一个这样的 `Element` 节点，或者返回 `null`。这是一个重要的常用方法，因为它为获取表示指定的文档元素的 `Element` 对象提供了简便的方法。注意，该方法的名称以 `Id` 结尾，不是 `ID`，不要拼错了。

在 HTML 文档中，该方法总是检索具有指定 `id` 的属性。使用 `HTMLDocument.getElementsByName()` 方法，根据它们的 `name` 属性中的值来查找 HTML 元素。

在 XML 文档中，这个方法则是使用类型为 `id` 的任一属性来查找，而不管这个属性的名称是什么。如果 XML 属性的类型是未知的（如 XML 解析器忽略了或不能定位文档的 DTD），该方法总是返回 `null`。在客户端 JavaScript 中，这个方法并不经常和 XML 文档一起使用。实际上，`getElementById()` 最初被定义为 `HTMLDocument` 接口的一个成员，但是在后来的 2 级 DOM 中移入到 `Document` 接口中。

参阅

`Document.getElementsByTagName()`
`Element.getElementsByTagName()`
`HTMLDocument.getElementsByName()`

Document.getElementsByTagName()

1 级核心 DOM

返回所有具有指定名称的 `Element` 节点

摘要

```
Element[] getElementsByTagName(String tagname);
```

参数

tagname

要返回的 `Element` 节点的标记名或通配符 “*”（返回文档中的所有 `Element` 节点，无论标记的名称是什么）。对于 HTML 文档，标记名不区分大小写（IE 6 以前的版本不支持这一通配符语法）。

返回值

文档树中具有指定标记名的 `Element` 节点的只读数组（从技术上说，是 `NodeList` 对象）。返回的 `Element` 节点的顺序就是它们在源文档中出现的顺序。

描述

该方法将返回一个 `NodeList` 对象（可以作为只读数组处理），该对象存放文档中具有指定标记名的所有 `Element` 节点，它们存放的顺序就是在源文档中出现的顺序。`NodeList` 对象是“活的”，即如果在文档中添加或删除了具有指定标记名的元素，它的内容会自动进行必要的更新。

HTML 文档不区分大小写，所以可以用任意的大小写形式指定 `tagname`，它将与文档中所有同名的标记匹配，无论这些标记在源文档中采用的大小写形式是什么。但 XML 文档区分大小写，`tagname` 只和源文档中名称与大小写形式完全相同的标记匹配。

注意，`Element` 接口定义了一个同名的方法，该方法只检索文档的子树。另外，`HTMLDocument` 接口定义了 `getElementsByName()` 方法，它基于 `name` 属性的值（而不是标记名）检索元素。

例子

可以用下列代码检索并遍历文档中的所有 `<h1>` 标记：

```
var headings = document.getElementsByTagName("h1");
for(var i = 0; i < headings.length; i++) { // Loop through the returned tags
    var h = headings[i];
    // Now do something with the <h1> element in the h variable
}
```

参阅

`Document.getElementById()`
`Element.getElementsByTagName()`
`HTMLDocument.getElementsByName()`

`Document.getElementsByTagNameNS()`

2 级核心 DOM

返回所有具有指定名字和名字空间的 `Element` 节点

摘要

```
Node[] getElementsByTagNameNS(String namespaceURI,
                               String localName);
```

参数

`namespaceURI`

要获取的元素的名字空间的惟一标识符，或者“*”，匹配所有名字空间。

`localName`

要获取的元素的本地名（或者“*”）来匹配所有本地名字。

返回值

文档树中具有指定的名字空间和本地名的 `Element` 节点的只读数组（从技术上说，是 `NodeList` 对象）。

描述

该方法与 `getElementsByTagName()` 相似，只是它根据名字空间和名称来检索元素。只有使用名字空间的 XML 文档才会使用它。

`Document.importNode()`

2 级核心 DOM

把一个节点从另一个文档复制到该文档以便应用

摘要

```
Node importNode(Node importedNode,  
                boolean deep)  
throws DOMException;
```

参数

importedNode

要导入的节点。

deep

如果为 `true`，还要递归复制 *importedNode* 节点的所有子孙节点。

返回值

importedNode（可能还有它的所有子孙节点）的副本，它的 `ownerDocument` 属性设置到该文档。

抛出

如果 *importedNode* 是 `Document` 节点或 `DocumentType` 节点，该方法将抛出代码为 `NOT_SUPPORTED_ERR` 的 `DOMException` 异常，因为不能导入这些类型的节点。

描述

该方法的参数是另一个文档中定义的节点，返回值是适合插入该文档的节点的副本。如果 *deep* 值为 `true`，那么还要复制该节点的所有子孙节点。无论如何，原始节点和它的子孙节点都不会被修改。返回的副本的 `ownerDocument` 属性被设置为当前文档，但 `parentNode` 属性为 `null`，因为它还没有插入文档。在原始节点或树中注册的事件监听器函数不会被复制。

当导入 `Element` 节点时，只有在源文档中明确设置的属性才会被导入。当导入 `Attr` 节点时，将自动把它的 `specified` 属性设置为 `true`。

参阅

`Node.cloneNode()`

`Document.loadXML()`

Internet Explorer

通过解析一个 XML 标记字符串来组成该文档

摘要

```
void loadXML(String text)
```

参数

`text`

要解析的 XML 标记。

描述

这个特定于 IE 的方法解析指定的 XML 文本串，然后在当前文档对象中构建一棵 DOM 节点的树，而丢弃此前存在于文档中的任何节点。

这个方法在表示 HTML 文档的 Document 对象上并不存在。在调用 `loadXML()` 之前，通常创建一个新的、空的 Document 对象来保存解析的内容：

```
var doc = new ActiveXObject("MSXML2.DOMDocument");
doc.loadXML(markup);
```

参阅 `DOMParser.parseFromString()` 以了解一种非 IE 的替代方法。

参阅

`DOMParser.parseFromString()`

`Document.removeEventListener()`

参阅 `Element.removeEventListener()`

DocumentFragment

1 级核心 DOM

邻接节点和它们的子树

Node → DocumentFragment

描述

`DocumentFragment` 接口表示文档的一部分（或一段）。更确切地说，它表示一个或多个邻接的 Document 节点和它们的所有子孙节点。`DocumentFragment` 节点不属于文档树，继承的 `parentNode` 属性总是为 `null`。不过它有一种特殊行为，该行为使得它非常有用，即当请求把一个 `DocumentFragment` 节点插入文档树时，插入的不是 `DocumentFragment` 自身，而是它的所有子孙节点。这使得 `DocumentFragment` 成了有用的占位符，暂时存放那些希望一次插入文档的节点。它还有利于实现文档的剪切、复制和粘贴操作，尤其是与 `Range` 接口一起使用时更是如此。

可以用 `Document.createDocumentFragment()` 方法创建新的空 `DocumentFragment` 节点, 也可以用 `Range.extractContents()` 或 `Range.cloneContents()` 方法获取包含现有文档的片段的 `DocumentFragment` 节点。

参阅

`Range`

DocumentType

1 级 DOM XML

XML 文档的 DTD

Node → DocumentType

属性

`readonly String internalSubset` [2 级 DOM]

DTD 的内部子集 (即出现在文档自身的 DTD, 而不是出现在外部文件中的 DTD)。内部子集的分界符 `[]` 不属于返回值。如果没有这样的内部子集, 该属性为 `null`。

`readonly String name`

文档的类型名。它是 XML 文档开头的 `<!DOCTYPE>` 后的标识符, 它与文档根元素的标记名相同。

`readonly String publicId` [2 级 DOM]

DTD 的外部子集的公有标识符。如果没有设置这种标识符, 值为 `null`。

`readonly String systemId` [2 级 DOM]

DTD 的外部子集的系统标识符。如果没有设置这种标识符, 值为 `null`。

描述

这种不常用的接口表示 XML 文档的 DTD。专门处理 HTML 文档的程序设计者不必使用该接口。

因为 DTD 不属于文档的内容, 所以 `DocumentType` 节点不会出现在文档树中。如果 XML 文档有 DTD, 通过 `Document` 节点的 `doctype` 属性可以访问那个 DTD 的 `DocumentType` 节点。

尽管 W3C DOM 包含了一个 API, 用来访问一个 DTD 中所定义的 `XMLEntity` 节点和 `Notation` 节点, 这里并没有介绍这个 API, 常见的 Web 浏览器不能为它们所载入文档解析 DTD, 并且客户端 JavaScript 不能访问那些 `Entity` 节点和 `Notation` 节点。对于客户端 JavaScript 编程来说, 这些接口只表示 `<!DOCTYPE>` 标记的内容, 而不是它所引用的 DTD 文件的内容。

`DocumentType` 是不可变的, 它的内容不能改变。

参阅

`Document`, `DOMImplementation.createDocument()` `DOMImplementation.createDocumentType()`

DOMException

1 级核心 DOM

通知核心 DOM 对象的异常或错误

Object → DOMException

常量

下面的常量定义了 DOMException 对象的 `code` 属性的合法值。注意，这些常量是 DOMException 的静态属性，不是个别异常对象的属性。

```
unsigned short INDEX_SIZE_ERR = 1
```

说明数组或字符串下标的溢出错误。

```
unsigned short DOMSTRING_SIZE_ERR = 2
```

说明请求的文本太大，当前 JavaScript 实现的字符串容纳不了它。

```
unsigned short HIERARCHY_REQUEST_ERR = 3
```

说明发生了要把节点放在文档树层次中的不合法位置的操作。

```
unsigned short WRONG_DOCUMENT_ERR = 4
```

说明发生了从创建节点的文档以外的文档使用该节点的操作。

```
unsigned short INVALID_CHARACTER_ERR = 5
```

说明（如在元素名中）使用了不合法的字符。

```
unsigned short NO_DATA_ALLOWED_ERR = 6
```

当前不采用。

```
unsigned short NO_MODIFICATION_ALLOWED_ERR = 7
```

说明发生了修改只读的、不允许修改的节点的操作。

```
unsigned short NOT_FOUND_ERR = 8
```

说明在期望的位置没有找到指定的节点。

```
unsigned short NOT_SUPPORTED_ERR = 9
```

说明当前的 DOM 实现不支持某个属性或方法。

```
unsigned short INUSE_ATTRIBUTE_ERR = 10
```

说明在一个 Attr 节点已经关联到另一个 Element 节点时，发生了把一个 Attr 节点关联到另一个 Element 节点的操作。

```
unsigned short INVALID_STATE_ERR = 11 [2 级 DOM]
```

说明使用了处于不允许使用状态或不再允许使用状态的对象。

```
unsigned short SYNTAX_ERR = 12 [2 级 DOM]
```

说明指定的字符串含有语法错误。通常由 CSS 属性声明使用。

```
unsigned short INVALID_MODIFICATION_ERR = 13 [2 级 DOM]
```

说明发生了修改 CSSRule 对象或 CSSValue 对象的操作。

```
unsigned short NAMESPACE_ERR = 14 [2 级 DOM]
```

说明有涉及元素或属性的名字空间的错误。

```
unsigned short INVALID_ACCESS_ERR = 15 [2 级 DOM]
```

说明以一种当前的实现不支持的方法访问对象。

属性

`unsigned short code`

出错代码，提供了引发异常的原因的详细情况。该属性的合法值（和它们的含义）由前面列出的常量定义。

描述

当错误使用或在不适合的环境中使用某个 DOM 属性或方法时，就会抛出一个 `DOMException` 对象。`code` 属性的值说明了发生的异常的一般类型。注意，读写对象的属性或调用对象的方法时，都有可能抛出 `DOMException`。

在对象的属性和方法的描述部分，列出了它们可能抛出的异常。但要注意，这些列表省略了某些通常抛出的异常。当想要修改只读节点（如 `Entity` 节点或它的某个子孙节点）的操作时，代码为 `NO_MODIFICATION_ALLOWED_ERR` 的 `DOMException` 就会被抛出。因此，`Node` 接口（和它的子接口）的大多数方法和读/写属性都可能抛出该异常。因为只读节点只出现在 XML 文档中，不出现在 HTML 文档中，而且它普遍地应用于 `Node` 对象的方法和可写属性，所以这些方法和属性的描述部分省略了 `NO_MODIFICATION_ALLOWED_ERR` 异常。

同样，许多返回字符串的 DOM 方法和属性都可以抛出代码为 `DOMSTRING_SIZE_ERR` 的 `DOMException` 异常，该异常说明返回的文本太长，当前 JavaScript 实现中的字符串容纳不下它。虽然理论上许多属性和方法都可能抛出这种异常，但在实际中它很少出现，所以这些属性和方法的描述部分省略了该异常。

注意，并非 DOM 中的所有异常都由 `DOMException` 通知：涉及 DOM Range 模块的异常，会导致抛出 `RangeException` 异常。

参阅

`RangeException`

DOMImplementation

1 级核心 DOM

独立于任何特殊文档的方法

[Object → DOMImplementation](#)

方法

`createDocument()`

创建带有指定类型的根元素（返回的 `Document` 对象的 `documentElement` 属性）的新 `Document` 对象。

`createDocumentType()`

创建新的 `DocumentType` 节点。

`hasFeature()`

检查当前实现是否支持具有指定特性的版本。

描述

DOMImplementation 接口是一个占位符，存放不专属于任何特定 Document 对象，而对 DOM 实现来说是“全局性”的方法。可以通过任何 Document 对象的 implementation 属性获得对 DOMImplementation 对象的引用。

DOMImplementation.createDocument()

2 级核心 DOM

创建一个新 Document 对象和指定的根元素

摘要

```
Document createDocument(String namespaceURI,  
                        String qualifiedName,  
                        DocumentType doctype)  
    throws DOMException;
```

参数

namespaceURI

为文档创建的根元素的名字空间的惟一标识符。如果没有名字空间，则为 null。

qualifiedName

为文档创建的根元素的名称。如果 *namespaceURI* 不为 null，该名称应该包括名字空间前缀和冒号。

doctype

新创建的 Document 对象的 DocumentType 对象。如果没有想得到的 DocumentType 对象，则为 null。

返回值

一个带有 documentElement 属性的 Document 对象设置为指定类型的 Element 根节点。

抛出

该方法将在下列环境中抛出具有下列代码的 DOMException 异常：

INVALID_CHARACTER_ERR

qualifiedName 含有不合法的字符。

NAMESPACE_ERR

qualifiedName 格式错误，或者 *qualifiedName* 和 *namespaceURI* 不匹配。

NOT_SUPPORTED_ERR

当前实现不支持 XML 文档，没有实现该方法。

WRONG_DOCUMENT_ERR

另一个文档使用了 *doctype*，或其他 DOMImplementation 对象创建了该 *doctype*。

描述

该方法将创建一个新的 Document 对象，并为文档指定它的根对象 documentElement。如果参数 *doctype* 不是 null，那么 DocumentType 对象的 *ownerDocument* 属性将被设置为新创建的文档。

该方法用于创建 XML 文档，只支持 HTML 的实现可能不支持它。

参阅

DOMImplementation.createDocumentType()

DOMImplementation.createDocumentType()

2 级核心 DOM

创建一个 DocumentType 节点

摘要

```
DocumentType createDocumentType(String qualifiedName,  
                                String publicId,  
                                String systemId)  
    throws DOMException;
```

参数

qualifiedName

文档类型的名字。如果使用 XML 名字空间，该参数可能是一个限定名，用来指定名字空间前缀和本地名，两者之间用冒号分隔。

publicId

文档类型的公有标识符或 null。

systemId

文档类型的系统标识符或 null。该参数通常声明了 DTD 文件的本地文件名。

返回值

新创建的 DocumentType 对象，ownerDocument 属性为 null。

抛出

该方法将在下列环境中抛出具有下列代码的 DOMException 异常：

INVALID_CHARACTER_ERR

qualifiedName 含有不合法的字符。

NAMESPACE_ERR

qualifiedName 格式错误。

NOT_SUPPORTED_ERR

当前实现不支持 XML 文档，没有实现该方法。

描述

该方法将创建一个新的 `DocumentType` 节点。该方法只声明文档类型的一个外部子集。从 2 级 DOM 起，DOM 标准就不再提供声明内部子集的方法，返回的 `DocumentType` 对象不定义任何 `Entity` 节点和 `Notation` 节点。该方法只和 XML 文档一起使用，只支持 HTML 文档的实现不支持该方法。

DOMImplementation.hasFeature()

1 级 DOM 核心

确定实现是否支持某个特性

摘要

```
boolean hasFeature(String feature,  
                   String version);
```

参数

feature

特性名，用于判断哪个支持被测试。后面的表中列出了 2 级 DOM 标准支持的有效特性名的集合。特性名不区分大小写。

version

版本号，用于判断哪个支持被测试，或者为 `null`，如果该特性的所有版本都被支持，则为空串（""）。在 2 级 DOM 标准中，支持的版本号是 1.0 和 2.0。

返回值

如果当前实现完全支持指定特性的指定版本，返回值为 `true`，否则为 `false`。如果没有指定版本号，而且实现完全支持指定特性的所有版本，该方法也返回 `true`。

描述

W3C DOM 标准是模块化的，不要求每种实现都实现标准中的所有模块或特性。该方法用于检测一种 DOM 实现是否支持 DOM 标准的指定模块。在这个 DOM 参考手册中，每个条目的可用性信息都包括模块名。注意，虽然 Internet Explorer 5 和 5.5 都部分地支持 1 级 DOM 标准，但在 IE 6 之前，没有实现支持这个重要的方法。

下表列出了可以作为 *feature* 参数的模块名的完整集合。

特性	描述
Core	实现 <code>Node</code> 、 <code>Element</code> 、 <code>Document</code> 、 <code>Text</code> 和其他所有 DOM 实现都要求实现的基本接口；所有遵守 DOM 标准的实现都必须支持该模块
HTML	实现 <code>HTMLElement</code> 、 <code>HTMLDocument</code> 和其他 HTML 专有接口
XML	实现 <code>Entity</code> 、 <code>EntityReference</code> 、 <code>ProcessingInstruction</code> 、 <code>Notation</code> 和其他 XML 文档专用的节点类型

特性	描述
StyleSheets	实现描述普通样式表的简单接口
CSS	实现 CSS 样式表专有的接口
CSS2	实现 CSS2Properties 接口
Events	实现基本的事件处理接口
UIEvents	实现处理用户界面事件的接口
MouseEvents	实现处理鼠标事件的接口
HTMLEvents	实现处理 HTML 事件的接口
MutationEvents	实现处理文档变化事件的接口
Range	实现操作文档范围的接口
Traversal	实现进行高级文档遍历的接口
Views	实现处理文档视图的接口

例子

可以在代码中如下使用该方法：

```
// Check whether the browser supports the DOM Level 2 Range API
if (document.implementation &&
    document.implementation.hasFeature &&
    document.implementation.hasFeature("Range", "2.0")) {
    // If so, use it here...
}
else {
    // If not, fall back on code that doesn't require Range objects
}
```

参阅

Node.isSupported()

DOMParser

Firefox 1.0, Safari 2.01, Opera 7.60

解析 XML 标记来创建一个文档

Object → DOMParser

构造函数

new DOMParser()

方法

parseFromString()

解析 XML 标记并返回一个 Document。

描述

一个DOMParser对象解析XML文本并返回一个XML Document对象。要使用DOMParser, 使用不带参数的构造函数来实例化它, 然后调用其parseFromString()方法:

```
var doc = (new DOMParser()).parseFromString(text);
```

IE 不支持 DOMParser 对象。相反, 它支持使用 Document.loadXML() 的 XML 解析。注意, XMLHttpRequest 对象也可以解析 XML 文档。参阅 XMLHttpRequest 的 responseXML 属性。

参阅

Document.loadXML()、XMLHttpRequest, 第 21 章

DOMParser.parseFromString()

解析 XML 标记

摘要

```
Document parseFromString(String text,  
                           String contentType)
```

参数

text

要解析的 XML 标记。

contentType

文本的内容类型。这可能是“text/xml”、“application/xml”或“application/xhtml+xml”中的一个。注意, 不支持“text/html”。

返回值

保存 text 解析后表示的一个 Document 对象。参阅 Document.loadXML() 以了解这个方法的一个特定于 IE 的替代。

Element

1 级核心 DOM

一个 HTML 元素或 XML 元素

Node → Element

子接口

HTMLElement

属性

readonly String tagName

元素的标记名。例如, 对于 HTML <P> 元素, 它是字符串“P”。对于 HTML 文档,

无论标记名在文档源代码中的大小写形式是什么，都以大写形式返回它。XML 文档区分大小写，返回的标记名与它在文档源代码中的形式完全一致。该属性的值与 Node 接口的 `nodeName` 属性的值相同。

方法

`addEventListener()`

在这个元素的事件句柄集合中添加一个事件句柄函数。这是一个除 IE 以外的所有现代浏览器都支持的 DOM 标准方法。

`attachEvent()`

在这个元素的句柄集合中添加一个事件句柄函数。这是 `addEventListener()` 的一个特定于 IE 的替代。

`detachEvent()`

从这个元素中移除一个事件句柄函数。这是标准的 `removeEventListener()` 方法的一个特定于 IE 的替代。

`dispatchEvent()`

为该节点分派一个合成事件。

`getAttribute()`

以字符串形式返回指定属性的值。

`getAttributeNS()`

以字符串形式返回由本地名和名字空间 URI 指定的属性的值。只有使用名字空间的 XML 文档才使用该方法。

`getAttributeNode()`

以 Attr 节点的形式返回指定属性的值。

`getAttributeNodeNS()`

以 Attr 节点的形式返回由本地名和名字空间 URI 指定的属性的值。只有使用名字空间的 XML 文档才使用该方法。

`getElementsByTagName()`

返回一个数组（从技术上说是 `NodeList`），元素是具有指定标记名的所有 Element 节点的子孙节点，它们在数组中的顺序就是出现在文档中的顺序。

`getElementsByTagNameNS()`

与 `getElementsByTagName()` 相似，只是元素标记名由本地名和名字空间 URI 指定。只有使用名字空间的 XML 文档才会使用该方法。

`hasAttribute()`

如果该元素具有指定名字的属性，则返回 `true`，否则返回 `false`。注意，如果在文档的源代码中明确设置了指定的属性，或者文档的 DTD 为指定的属性设置了默认值，该方法都返回 `true`。

`hasAttributeNS()`

与 `hasAttribute()` 相似，只是属性由本地名和名字空间 URI 共同指定。只有使用名字空间的 XML 文档才会使用该方法。

`removeAttribute()`

从元素中删除指定的属性。注意，该方法只删除在文档的源代码中明确设置过的属性。如果 DTD 为该属性设置了默认值，这个默认值将成为属性的新值。

`removeAttributeNode()`

从元素的属性列表中删除指定的 Attr 节点。注意，该方法只能删除在文档的源代码中明确设置过的属性。如果 DTD 为删除的属性设置了默认值，那么将创建一个新的 Attr 节点，用于表示该属性的默认值。

`removeAttributeNS()`

与 `removeAttribute()` 相似，只是要删除的属性由本地名和名字空间 URI 共同指定。只有使用名字空间的 XML 文档才会使用该方法。

`removeEventListener()`

从这个元素的句柄集合中删除一个事件句柄函数。这是一个除了 IE 以外的所有现代浏览器都支持的标准 DOM 方法，而 IE 则使用 `detachEvent()`。

`setAttribute()`

把指定的属性设置为指定的字符串值。如果具有指定名称的属性还不存在，则给元素添加一个新属性。

`setAttributeNode()`

把指定的 Attr 节点添加到元素的属性列表。如果已经存在同名的属性，那么该属性的值将被替换。

`setAttributeNodeNS()`

与 `setAttributeNode()` 相似，只是该方法适用于 `Document.createAttributeNS()` 返回的节点。只有使用名字空间的 XML 文档才会使用该方法。

`setAttributeNS()`

与 `setAttribute()` 相似，只是要设置的属性名由本地名和名字空间 URI 共同指定。只有使用名字空间的 XML 文档才会使用该方法。

描述

Element 接口表示 HTML 元素、XML 元素或标记。tagName 属性指定了元素的名称。一个 Document 的 documentElement 属性引用这个文档的根 Element 对象。HTMLDocument 对象的 body 属性也类似，它引用了文档的 <body> 元素。要在一个 HTML 文档中找到一个指定名称的元素，使用 `Document.getElementById()` (并通过 id 属性给该元素一个唯一的名称)。要通过标记名来定位一个元素，使用 `getElementsByTagName()`，这既是 Element 的方法也是 Document 的方法。在 HTML 文档中，也可以使用类似的

`HTMLDocument.getElementsByTagName()` 方法来根据元素的 `name` 属性来查找元素。最后，可以用 `Document.createElement()` 方法，创建一个插入文档的新 `Element` 元素。

`addEventListener()` 方法（及其特定于 IE 的替代方法 `attachEvent()`）提供了在该元素上为特定类型的事件注册事件句柄函数的一种方法。参阅第 17 章了解详细内容。从技术上讲，`addEventListener()`、`removeEventListener()` 和 `dispatchEvent()` 都是由 2 级 DOM Events 规范的 `EventTarget` 接口定义的。既然所有的 `Element` 对象都实现了 `EventTarget`，我们在这里列出这些方法。

这个接口的各种其他方法提供了对元素的属性的访问。在 HTML 文档中（以及许多 XML 文档中），所有属性都有简单的字符串值，并且你可以使用简单方法 `getAttribute()` 和 `setAttribute()` 进行所需的任何属性操作。

如果你在使用 XML 文档，它可能包含了 Entity 参考页作为属性值的一部分，你将必须使用 `Attr` 对象及其节点的子树。你可针对一个属性使用 `getAttributeNode()` 和 `setAttributeNode()` 来获取和设置 `Attr` 对象，或者可以在 `Node` 接口的 `attributes[]` 数组遍历 `Attr` 节点。如果你使用了一个用到 XML 名字空间的 XML 文档，需要使用名字带有“NS”的各种方法。

在 1 级 DOM 规范中，`normalize()` 方法是 `Element` 接口的一部分。在 2 级规范中，`normalize()` 则是 `Node` 接口的一部分。所有 `Element` 节点继承这个方法并且仍然可以使用它。

参阅

`HTMLElement`, `Node`; 第 15 章, 第 17 章

Element.addEventListener()

2 级 DOM Events

注册一个事件句柄。

摘要

```
void addEventListener(String type,  
                        Function listener,  
                        boolean useCapture);
```

参数

type

事件监听器调用所针对的事件的类型。例如，“load”、“click”或“mousedown”。

listener

当指定类型的事件分派给这个元素的时候，事件监听器函数被调用。调用的时候，这个监听器函数被传递给一个 `Event` 对象，然后作为在该元素上注册的一个方法来调用。

useCapture

如果为 `true`，那么只有在事件传播的捕捉阶段，指定的 *listener* 才会被调用。更常用的值是 `false`，意味着在捕捉阶段不会调用 *listener*，而当该节点是实际的事件目标或事件从它的原始目标起泡到该节点时，调用 *listener*。

描述

该方法将把指定的事件监听器函数添加到当前节点的监听器集合中，以处理指定类型 *type* 的事件。如果 *useCapture* 为 `true`，则监听器被注册为捕捉事件监听器。如果 *useCapture* 为 `false`，它就注册为普通事件监听器。

`addEventListener()` 可能被调用多次，在同一个节点上为同一种类型的事件注册多个事件句柄。但要注意，DOM 不能确定多个事件句柄被调用的顺序。

如果一个事件监听器函数在同一个节点上用相同的 *type* 和 *useCapture* 参数注册了两次，那么第二次注册将被忽略。如果正在处理一个节点上的事件时，在这个节点上注册了一个新的事件监听器，则不会为那个事件调用新的事件监听器。

当用 `Node.cloneNode()` 方法或 `Document.importNode()` 方法复制一个 Document 节点时，不会复制为原始节点注册的事件监听器。

这个方法也在 Document 和 Window 对象上定义了，而且工作方式类似。

参阅

Event, 第 17 章

Element.attachEvent()

IE 4

注册一个事件句柄

摘要

```
void attachEvent(String type,  
                  Function listener);
```

参数

type

事件监听器调用所针对的事件的类型，带有一个“on”前缀。例如，“onload”、“onclick”或“onmousedown”。

listener

当指定类型的事件分派给这个元素的时候，事件监听器函数被调用。不会为这个函数传递任何参数，但是它可以从 Window 对象的 `event` 属性获得 Event 对象。

描述

这个方法是一个特定于 IE 的事件注册方法。它和标准的 `addEventListener()` 方法（IE 不支持它）具有相同的作用，但是，它和该函数也有一些重要的差别：

- 由于 IE 事件模型不支持事件捕获，所以 `attachEvent()` 和 `detachEvent()` 只需要两个参数：事件类型和句柄函数。
- 传递给 IE 方法的事件句柄名应该包括“on”前缀。例如，和 `attachEvent()` 一起使用的是“onclick”，而不是和 `addEventListener()` 一起使用的“click”。
- 使用 `attachEvent()` 注册的函数调用的时候没有 `Event` 对象参数。相反，它们必须读取 `Window` 对象的 `event` 属性。
- 使用 `attachEvent()` 注册的函数作为全局函数调用，而不是作为事件发生其上的文档元素的方法来调用。也就是说，当使用 `attachEvent()` 注册的一个事件句柄执行的时候，`this` 关键字引用的是 `Window` 对象，而不是事件目标元素。
- `attachEvent()` 允许同一个事件句柄函数注册多次。当指定类型的一个事件发生的时候，注册的函数调用次数和它注册的次数相同。

这个方法也在 `Document` 和 `Window` 对象上定义了，而且工作方式类似。

参阅

`Element.addEventListener()`、`Event`，第 17 章

`Element.detachEvent()`

IE 4

删除一个事件监听器

摘要

```
void detachEvent(String type,  
                  Function listener)
```

参数

type

要删除的事件监听器所针对的事件的类型，带有一个“on”前缀。例如，“onclick”。

listener

要删除的事件监听器函数。

描述

这个方法解除掉由 `attachEvent()` 方法所执行的事件句柄函数注册。它是 `removeEventListener()` 方法的特定于 IE 的替代。要为一个元素删除一个事件句柄函数，只需要使用你最初传递 `attachEvent()` 的相同参数来调用 `detachEvent`。

这个方法也在 Document 和 Window 对象上定义了，而且工作方式类似。

Element.dispatchEvent()

2 级 DOM Events

给该节点分派一个合成事件

摘要

```
boolean dispatchEvent(Event evt)
    throws EventException;
```

参数

evt

要分派的 Event 对象。

返回值

如果在事件传播过程中调用了 evt 的 preventDefault() 方法，则返回 false，否则返回 true。

抛出

如果 Event 对象 evt 没有被初始化，或者它的 type 属性为 null 或空串，该方法将抛出异常。

描述

该方法将分派一个合成事件，它由 Document.createEvent() 创建，由 Event 接口或它的某个子接口定义的初始化方法初始化。调用该方法的节点将成为事件的目标节点，该事件在捕捉阶段中第一次沿着文档树向下传播。如果该事件的 bubbles 属性为 true，那么在事件的目标节点自身处理事件后，它将沿着文档树向上起泡。

参阅

Document.createEvent()、Event.initEvent()、MouseEvent.initMouseEvent()

Element.getAttribute()

1 级核心 DOM

返回指定属性的字符串值

摘要

```
String getAttribute(String name);
```

参数

name

要返回值的属性的名称。

返回值

指定属性的字符串值。如果在文档中没有设置该属性的值，而且文档类型也没有设置它的默认值，那么返回值为空串。而某些实现在这种情况下返回 `null`。

描述

方法 `getAttribute()` 将返回一个元素的指定属性的值。注意，`HTMLElement` 对象定义了和每个标准 HTML 属性对应的 JavaScript 属性，因此，只有当你查询非标准属性的值的时候，才需要和 HTML 文档一起使用此方法。

在 XML 文档中，属性值不能直接作为元素属性，必须通过调用来查询它们。对于使用名字空间的 XML 文档来说，需要使用 `getAttributeNS()` 方法。

例子

接下来的代码说明了两种获取 HTML 元素的性质值的方法：

```
// Get all images in the document
var images = document.body.getElementsByTagName("img");
// Get the src attribute of the first one
var src0 = images[0].getAttribute("src");
// Get the src attribute of the second simply by reading the property
var src1 = images[1].src;
```

参阅

`Element.getAttributeNode()`、`Element.getAttributeNS()`、`Node`

`Element.getAttributeNode()`

1 级核心 DOM

返回指定属性的 Attr 节点

摘要

`Attr getAttributeNode(String name);`

参数

name

想获取的属性的名称。

返回值

一个 Attr 节点，它的子孙表示指定的属性的值。如果该元素没有这样的属性，则为 `null`。

描述

方法 `getAttributeNode()` 将返回一个 Attr 节点，表示指定的属性的值。注意，通过从 `Node` 接口继承的 `attributes` 属性也可以获取该 Attr 节点。

参阅

`Element.getAttribute()`、`Element.getAttributeNodeNS()`

`Element.getAttributeNodeNS()`

2 级核心 DOM

返回具有名字空间的属性的 `Attr` 节点

摘要

```
Attr getAttributeNodeNS(String namespaceURI,  
                        String localName);
```

参数

namespaceURI

惟一标识属性的名字空间的 URI。若没有名字空间，则该参数为 `null`。

localName

声明该属性在名字空间中名称的标识符。

返回值

一个 `Attr` 节点，它的子孙表示指定属性的值。如果该元素没有这样的属性，则返回 `null`。

描述

该方法与 `getAttributeNode()` 方法相似，只是属性名由名字空间 URI 和在这个名字空间中定义的本地名共同指定。只有使用名字空间的 XML 文档才使用该方法。

参阅

`Element.getAttributeNode()`、`Element.getAttributeNS()`

`Element.getAttributeNS()`

2 级核心 DOM

获取使用指定名字空间的属性的值

摘要

```
String getAttributeNS(String namespaceURI,  
                     String localName);
```

属性

namespaceURI

惟一标识属性的名字空间的 URI。若没有名字空间，则该参数为 `null`。

localName

声明该属性在名字空间中名称的标识符。

返回值

指定属性的字符串值。如果在文档中没有明确设置该属性，该方法将返回空串，但有些实现返回 `null`。

描述

该方法与 `getAttribute()` 方法相似，只是属性由名字空间 URI 和名字空间中的本地名指定。只有使用名字空间的 XML 文档才使用该方法。

参阅

`Element.getAttribute()`、`Element.getAttributeNodeNS()`

Element.getElementsByTagName()

1 级核心 DOM

找到具有指定标记名的子孙元素

摘要

```
Element[] getElementsByTagName(String name);
```

参数

name

想获取的元素的标记名，或者为 “*”，表示应该返回所有的子孙节点，无论它们的标记名是什么。

返回值

一个只读数组（技术上说来，是 `NodeList` 对象），元素是 `Element` 对象，表示当前元素的子孙并具有指定标记名。

描述

该方法将遍历指定元素的子孙节点，返回一个 `Element` 节点的数组（实际上是 `NodeList` 对象），表示所有具有指定标记名的文档元素。元素在返回的数组中的顺序就是它们出现在文档源代码中的顺序。

注意，`Document` 接口也定义了 `getElementsByTagName()` 方法，它与该方法相似，但遍历整个文档，而不只是遍历某个元素的子孙节点。不要把该方法与 `HTMLDocument.getElementsByTagName()` 方法弄混淆，后者基于元素的 `name` 属性值检索元素，而不是基于它们的标记名检索元素。

例子

可以用下列代码找到文档中的所有 `<div>` 标记：

```
var divisions = document.body.getElementsByTagName("div");
```

用下面的代码可以找到 <div> 标记中的所有 <p> 标记：

```
var paragraphs = divisions[0].getElementsByTagName("p");
```

参阅

`Document.getElementById()`
`Document.getElementsByTagName()`
`HTMLDocument.getElementsByTagName()`

Element.getElementsByTagNameNS()

2 级核心 DOM

返回具有指定名称和名字空间的子孙元素

摘要

```
Node[] getElementsByTagNameNS(String namespaceURI,  
                               String localName);
```

参数

namespaceURI

惟一标识元素的名字空间的 URI。

localName

声明在名字空间中元素的名称的标识符。

返回值

一个只读的数组（从技术上说，是一个 `NodeList` 对象），元素是 `Element` 对象，表示当前元素的子孙节点和具有指定的名称和名字空间的元素。

描述

该方法与 `getElementsByTagName()` 相似，只是想获取的元素的标记名被指定为名字空间 URI 和在名字空间中定义的本地名的组合。只有使用名字空间的 XML 文档才使用该方法。

参阅

`Document.getElementsByTagNameNS()`、`Element.getElementsByTagName()`

Element.hasAttribute()

2 级核心 DOM

判断当前元素是否具有指定的属性

摘要

```
boolean hasAttribute(String name);
```

参数

name

要使用的属性的名称。

返回值

如果当前元素具有指定的属性，或者为指定的属性设置了默认值，则返回 `true`，否则返回 `false`。

描述

该方法将判断一个元素是否具有指定的属性，但不返回那个属性的值。注意，如果文档中明确设置了指定的属性，或者文档类型为该属性设置了默认值，`hasAttribute()` 都返回 `true`。

参阅

`Element.getAttribute()`、`Element.setAttribute()`

`Element.hasAttributeNS()`

2 级核心 DOM

判断当前元素是否具有指定的属性

摘要

```
boolean hasAttributeNS(String namespaceURI,  
                        String localName);
```

参数

namespaceURI

惟一标识属性的名字空间标识符，若没有名字空间，则为 `null`。

localName

声明在指定的名字空间中属性的名称。

返回值

如果该元素明确地设置了指定的值，或者给指定属性设置了默认值，则返回 `true`，否则返回 `false`。

描述

该方法与 `hasAttribute()` 方法相似，只是要检查的属性由名字空间和名称指定。只有使用名字空间的 XML 文档才使用该方法。

参阅

`Element.getAttributeNS()`、`Element.hasAttribute()`、`Element.setAttributeNS()`

Element.removeAttribute()1 级核心 DOM

从元素中删除指定的属性

摘要

```
void removeAttribute(String name);
```

参数

name

要删除的属性的名称。

抛出

如果该元素是只读的，并且不允许删除它的属性，则该方法将抛出代码为 NO_MODIFICATION_ALLOWED_ERR 的 DOMException 异常。

描述

方法 `removeAttribute()` 将从当前元素中删除指定的属性。如果文档类型为指定的属性设置了默认值，那么接下来调用 `getAttribute()` 方法将返回那个默认值。删除不存在的属性或没有设置但具有默认值属性的操作将被忽略。

参阅

`Element.getAttribute()`、`Element.setAttribute()`、`Node`

Element.removeAttributeNode()1 级核心 DOM

从元素中删除一个 Attr 节点

摘要

```
Attr removeAttributeNode(Attr oldAttr)  
    throws DOMException;
```

属性

oldAttr

要从元素中删除的 Attr 节点。

返回值

删除的 Attr 节点。

抛出

该方法将抛具有下列代码的 DOMException 异常：

NO_MODIFICATION_ALLOWED_ERR

当前元素是只读的，不允许删除属性。

NOT_FOUND_ERR

*oldAttr*不是当前元素的属性。

描述

该方法将从当前元素的属性集合中删除（并返回）一个 *Attr* 节点。如果 DTD 给删除的属性设置了默认值，那么该方法将添加一个新的 *Attr* 节点，表示这个默认值。用 *removeAttribute()* 方法代替该方法往往会更简单。

参阅

Attr、*Element.removeAttribute()*

Element.removeAttributeNS()

2 级核心 DOM

删除由名称和名字空间指定的属性

摘要

```
void removeAttributeNS(String namespaceURI,  
                        String localName);
```

参数

namespaceURI

属性的名字空间的惟一标识符，若没有名字空间，则为 *null*。

localName

声明在指定的名字空间中属性的名称。

抛出

如果当前元素是只读的，并且不允许删除它的属性，该方法将抛出代码为 NO_MODIFICATION_ALLOWED_ERR 的 *DOMException* 异常。

描述

方法 *removeAttributeNS()* 与方法 *removeAttribute()* 相似，只是要删除的属性由名称和名字空间共同指定，而不是只由名称指定。只有使用名字空间的 XML 文档才可以使用该方法。

参阅

Element.getAttributeNS()、*Element.removeAttribute()*、*Element.setAttributeNS()*

Element.removeEventListener()2 级 DOM Events

删除一个事件监听器

摘要

```
void removeEventListener(String type,  
                          Function listener,  
                          boolean useCapture);
```

参数*type*

要删除事件监听器的事件类型。

listener

要删除的事件监听器函数。

useCapture

如果要删除的是捕捉事件监听器，则为 true；如果要删除的是普通事件监听器，则为 false。

描述

该方法将删除指定的事件监听器函数。参数 *type* 和 *useCapture* 必须与调用 `addEventListener()` 方法的相应参数一样。如果没有找到与指定的参数匹配的事件监听器，该方法则什么都不做。

如果一个事件监听器函数被该方法删除，那么当节点发生指定类型的事件时，就不再调用它。即使一个事件监听器被同一节点上同类型事件注册的另一个事件监听器删除，它也不再被调用。

这个方法也被 Document 和 Window 对象所定义，并且工作方式类似。

Element.setAttribute()1 级核心 DOM

创建或改变元素的某个属性

摘要

```
void setAttribute(String name,  
                  String value)  
    throws DOMException;
```

参数*name*

要创建或修改的属性的名称。

value

属性的字符串值。

抛出

该方法将抛出具有下列代码的 `DOMException` 异常：

`INVALID_CHARACTER_ERR`

参数 *name* 含有 HTML 属性名或 XML 属性名不允许使用的字符。

`NO_MODIFICATION_ALLOWED_ERR`

当前元素是只读的，不允许修改它的属性。

描述

该方法将把指定的属性设置为指定的值。如果不存在具有指定名称的属性，该方法将创建一个新属性。

注意，一个 HTML 文档的 `HTMLElement` 对象还定义了对应所有标准 HTML 属性的 JavaScript 属性。因此，只有当你需要设置一个非标准属性时，才需要使用这一方法。

例子

```
// Set the TARGET attribute of all links in a document
var links = document.getElementsByTagName("a");
for(var i = 0; i < links.length; i++) {
    links[i].setAttribute("target", "newwindow");
    // Or more easily: links[i].target = "newwindow"
}
```

参阅

`Element.getAttribute()`、`Element.removeAttribute()`、`Element.setAttributeNode()`

`Element.setAttributeNode()`

1 级核心 DOM

给元素添加新的 `Attr` 节点

摘要

```
Attr setAttributeNode(Attr newAttr)
    throws DOMException;
```

参数

newAttr

表示要添加的属性或值要修改的属性的 `Attr` 节点。

返回值

被 *newAttr* 替换的 `Attr` 节点。如果没有替换任何属性，则返回 `null`。

抛出

该方法将抛出具有下列代码的 `DOMException` 异常：

`INUSE_ATTRIBUTE_ERR`

`newAttr` 已经是其他 `Element` 节点的属性集合的成员。

`NO_MODIFICATION_ALLOWED_ERR`

当前的 `Element` 节点是只读的，不允许修改它的属性。

`WRONG_DOCUMENT_ERR`

`newAttr` 的 `ownerDocument` 属性不同于要设置它的 `Element` 节点。

描述

该方法将给 `Element` 节点的属性集合添加新的 `Attr` 节点。如果当前 `Element` 已经具有一个同名的属性，该方法将用 `newAttr` 替换那个属性，返回被替换的 `Attr` 节点。如果不存在这样的属性，该方法将为 `Element` 定义一个新属性。

通常，用 `setAttribute()` 方法比用 `setAttributeNode()` 简单。

参阅

`Attr`、`Document.createAttribute()`、`Element.setAttribute()`

Element.setAttributeNodeNS()

2 级核心 DOM

给 `Element` 节点添加具有名字空间的 `Attr` 节点

摘要

```
Attr setAttributeNodeNS(Attr newAttr)  
    throws DOMException;
```

参数

`newAttr`

表示要添加的属性或值要修改的属性的 `Attr` 节点。

返回值

被 `newAttr` 替换的 `Attr` 节点，如果没有替换任何属性，则返回 `null`。

抛出

该方法抛出异常的原因和 `setAttributeNode()` 方法一样。此外，如果当前实现不支持 XML 文档和名字空间，该方法还将抛出代码为 `NOT_SUPPORTED_ERR` 的 `DOMException` 异常，说明该方法没有实现。

描述

该方法与 `setAttributeNode()` 方法相似, 只是它适用于由名称和名字空间共同指定的表示特性的 `Attr` 节点。

只有使用名字空间的 XML 文档才使用该方法。不支持 XML 文档的浏览器可能无法实现该方法 (即抛出代码为 `NOT_SUPPORTED_ERR` 的异常)。

参阅

`Attr`、`Document.createAttributeNS()`、`Element.setAttributeNS()`、`Element.setAttributeNode()`

Element.setAttributeNS()

2 级核心 DOM

创建或改变具有名字空间的属性

摘要

```
void setAttributeNS(String namespaceURI,  
                    String qualifiedName,  
                    String value)  
    throws DOMException;
```

参数

namespaceURI

惟一标识要设置或创建的属性的名字空间的 URI。若没有名字空间, 则为 `null`。

qualifiedName

属性的名称, 作为名字空间的前缀, 其后是冒号和名字空间中的一个名称。

value

属性的新值。

抛出

该方法将抛出具有下列代码的 `DOMException` 异常:

`INVALID_CHARACTER_ERR`

qualifiedName 参数含有 HTML 属性名或 XML 属性名中不允许出现的字符。

`NAMESPACE_ERR`

qualifiedName 格式错误, 或者 *qualifiedName* 和 *namespaceURI* 参数不匹配。

`NO_MODIFICATION_ALLOWED_ERR`

当前的元素是只读的, 不允许修改它的属性。

`NOT_SUPPORTED_ERR`

当前的 DOM 实现不支持 XML 文档。

描述

该方法与 `setAttribute()` 方法相似，只是要创建或设置的属性由名字空间 URI 和限定名（由名字空间前缀、冒号和名字空间中的本地名构成）共同指定。除了可以改变一个属性的值以外，使用该方法还可以改变属性的名字空间前缀。

只有使用名字空间的 XML 文档才会使用该方法。不支持 XML 文档的浏览器可能不会实现该方法（即抛出代码为 `NOT_SUPPORTED_ERR` 的异常）。

参阅

`Element.setAttribute()`、`Element.setAttributeNode()`

Event	2 级 DOM Events, IE
一个事件的信息	Object → Event

子接口

`UIEvent`

标准属性

下列属性都是 2 级 DOM 事件标准定义的属性。参阅 `KeyEvent`、`MouseEvent` 和 `UIEvent` 可以了解其他特定于类型的事件属性。

`readonly boolean bubbles`
如果事件是起泡类型的（即只有调用 `stopPropagation()` 方法才能停止起泡），则为 `true`，否则为 `false`。

`readonly boolean cancelable`
如果用 `preventDefault()` 方法可以取消与事件关联的默认动作，则为 `true`，否则为 `false`。

`readonly Object currentTarget`
当前处理该事件的 `Element`、`Document` 或 `Window`。在捕捉和起泡的过程中，它不同于 `target` 属性。

`readonly unsigned short eventPhase`
事件传播的当前阶段。其值是如下 3 个常量之一，它们分别代表捕获阶段、正常事件分派和起泡阶段：

eventPhase 常量	值
<code>Event.CAPTURING_PHASE</code>	1
<code>Event.AT_TARGET</code>	2
<code>Event.BUBBLING_PHASE</code>	3

readonly Object target

事件的目标节点，如生成事件的 Element、Document 或 Window。

readonly Date timeStamp

生成事件（技术上说来，是创建 Event 对象的事件）的日期和时间。实现并不是必须在这个域中提供有效的日期数据。如果它们没有提供该数据，则 Date 对象的 getTime() 方法将返回 0。参阅本书第三部分的 Date 对象。

readonly String type

当前 Event 对象表示的事件的名称。它与注册的事件句柄同名，或者是事件句柄属性删除前缀“on”。例如，“click”、“load”或“submit”。

IE 属性

IE 并不支持（至少在 IE 7 中）标准的 DOM 事件模型，并且 IE 的 Event 对象定义了一组完全不同的属性。IE 事件模型并没有为不同的事件定义继承层级，因此所有和任何事件的类型相关的属性都在这里列出。

boolean altKey

一个布尔值，指定事件发生时，**Alt** 键是否被按下并保持住了。

integer button

对于鼠标事件，button 属性声明了被按下的鼠标按钮或按钮。这个只读的整数是位掩码，如果鼠标左按钮被按下，设置第 1 位；如果鼠标右按钮被按下，设置第 2 位；如果（三按钮的）鼠标的中间按钮被按下，设置第 4 位。

boolean cancelBubble

如果事件句柄想阻止事件传播到包容对象，必须把该属性设为 true。

integer clientX, clientY

声明了事件发生的位置相对于浏览器页面的 X 坐标和 Y 坐标。

boolean ctrlKey

一个布尔值，指定事件发生时，**Ctrl** 键是否被按下并保持住了。

Element fromElement

对于 mouseover 和 mouseout 事件，fromElement 引用移出鼠标的元素。

integer keyCode

对于 keypress 事件，该属性声明了被敲击的键生成的 Unicode 字符码。对于 keydown 和 keyup 事件，它指定了被敲击的键的虚拟键盘码。虚拟键盘码可能和使用的键盘的布局相关。

integer offsetX, offsetY

发生事件的地点在事件源元素的坐标系统中的 X 坐标和 Y 坐标（参见 SrcElement）。

boolean returnValue

如果设置了该属性，它的值比事件句柄真正的返回值优先级高。把这个属性设置为 false，可以取消发生事件的源元素的默认动作。

integer screenX, screenY

事件发生的地点相对于屏幕的 X 坐标和 Y 坐标。

boolean shiftKey

一个布尔值，声明了事件发生时，**Shift** 键是否被按下并保持住了。

Object srcElement

对生成事件的 Window 对象、Document 对象或 Element 对象的引用。。

Element toElement

对于 mouseover 和 mouseout 事件，该属性引用移入鼠标的元素。

String type

一个字符串，声明了事件的类型。它的值是事件句柄的名称去掉前缀“on”。因此，在调用事件句柄 onclick() 时，Event 对象的 type 属性值就是“click”。

integer x, y

事件发生的位置的 X 坐标和 Y 坐标，它们相对于用 CSS 动态定位的最内层包容元素。

标准方法

如下的方法是 2 级 DOM Events 规范所定义的。在 IE 事件模型中，Event 对象没有这些方法：

initEvent()

初始化新创建的 Event 对象的属性。

preventDefault()

通知浏览器不要执行与事件关联的默认动作（如果存在这样的动作）。如果事件类型不是可取消的，则该方法不起作用。

stopPropagation()

终止事件在传播过程的捕捉、目标处理或起泡阶段进一步传播。调用该方法后，该节点上处理该事件的处理程序将被调用，事件不再被分派到其他节点。

描述

一个 Event 对象的属性提供了有关事件的细节（例如，事件在其上发生的元素）。一个 Event 对象的方法可以控制事件的传播。2 级 DOM Events 标准定义了一个标准的事件模型，它被除 IE 以外的所有现代浏览器所实现，而 IE 定义了自己的、不兼容的模型。这个参考页列出了标准 Event 对象的属性以及 IE Event 对象的属性。参见第 17 章了解有关两种事件模型的更多细节。然而，要特别注意：在标准事件模型中，Event 对象传递给事件句柄函数；但是在 IE 事件模型中，它被存储在 Window 对象的 event 属性中。

在标准事件模型中，Event 的各种子接口定义了额外的属性，它们提供了和特定事件类型相关的细节。在 IE 事件模型中，只有一种类型的 Event 对象，它用于所有类型的事件。

参阅

KeyEvent、MouseEvent、UIEvent，第 17 章

Event.initEvent()2 级 DOM Events

初始化新事件对象的属性

摘要

```
void initEvent(String eventTypeArg,  
               boolean canBubbleArg,  
               boolean cancelableArg);
```

参数

eventTypeArg

事件的类型。它可以是一种预定义的事件类型，如“load”或“submit”，也可以是你自己选择的一种定制类型。但以“DOM”开头的名称是保留的。

canBubbleArg

事件是否起泡。

cancelableArg

是否可以用 `preventDefault()` 方法取消事件。

描述

该方法将初始化 `Document.createEvent()` 方法创建的合成 Event 对象的 `type` 属性、`bubbles` 属性和 `cancelable` 属性。只有在新创建的 Event 对象被 Document 对象或 Element 对象的 `dispatchEvent()` 方法分派之前，才能调用 `Event.initEvent()` 方法。

参阅

`Document.createEvent()`、`MouseEvent.initMouseEvent()`、`UIEvent.initUIEvent()`

Event.preventDefault()2 级 DOM Events

取消事件的默认动作

摘要

```
void preventDefault();
```

描述

该方法将通知 Web 浏览器不要执行与事件关联的默认动作（如果存在这样的动作）。例如，如果 `type` 属性是“submit”，在事件传播的任意阶段可以调用任意的事件句柄，通过调用该方法，可以阻止提交表单。注意，如果 Event 对象的 `cancelable` 属性是 `false`，那么就是没有默认动作，或者不能阻止默认动作。无论哪种情况，调用该方法都没有作用。

Event.stopPropagation()2 级 DOM Events

不再分派事件

摘要

```
void stopPropagation();
```

描述

该方法将停止事件的传播，阻止它被分派到其他 Document 节点。在事件传播的任何阶段都可以调用它。注意，虽然该方法不能阻止同一个 Document 节点上的其他事件句柄被调用，但它可以阻止把事件分派到其他节点。

ExternalInterface

Flash 8 中的 ActionScript 对象

一个用于 Flash 的双向接口

静态属性

available

表示 Flash 是否可以和 JavaScript 通信。如果浏览器的安全策略阻止通信，这个属性将是 false。

静态函数

addCallback()

导出一个 ActionScript 方法，以便可以在 JavaScript 中调用它。

call()

从 ActionScript 调用一个 JavaScript 函数。

描述

ExternalInterface 是 Adobe Flash 插件第 8 版及其以后版本所定义的一个 ActionScript 对象。它定义了两个供 Flash 电影中的 ActionScript 使用的静态函数。这些函数使得 Web 浏览器中的 JavaScript 代码能够和 Flash 电影中的 ActionScript 代码通信。

参阅

FlashPlayer, 第 23 章

ExternalInterface.addCallback()

Flash 8 中的 ActionScript 函数

暴露一个 ActionScript 方法供 JavaScript 执行

摘要

```
boolean ExternalInterface.addCallback(String name,  
                                       Object instance,  
                                       Function func)
```

参数

name

要定义的 JavaScript 函数的名称。使用这个名称调用一个 JavaScript 函数使得 Flash Player 能够调用 ActionScript 函数 *func*，就好像它是实例对象的一个方法一样。

instance

基于它来调用 *func* 函数的对象，或者可以为 `null`。当 *func* 被调用的时候，这个参数变成 `this` 关键字的值。

func

当名为 *name* 的 JavaScript 函数调用的时候，这个 ActionScript 函数被调用。

返回值

成功则返回 `true`，失败则返回 `false`。

描述

供 Flash 电影中的 ActionScript 代码使用的静态函数，使得 Web 浏览器中的 JavaScript 代码能够调用 ActionScript 代码。当调用 `addCallback()` 的时候，它定义了一个名为 *name* 的顶级 JavaScript 函数，当这个函数调用的时候，就会调用作为 ActionScript 对象实例的一个方法的 ActionScript 函数 *func*。

JavaScript 函数的参数被转换并传递给 *func*，并且 *func* 的返回值被转换并且变成了 JavaScript 函数的返回值。参数和返回值是基本类型的数字、字符串、布尔值，以及包含基本数值的对象和数组。但是，不可能把一个像 Window 或 Document 这样的客户端 JavaScript 对象传递给一个 ActionScript 函数。也不能把一个像 MovieClip 这样的特定于 Flash 的 ActionScript 对象返回给 JavaScript。

参阅

FlashPlayer，第 23 章

ExternalInterface.call()

Flash 8 中的 ActionScript 函数

从 ActionScript 调用一个 JavaScript 函数

摘要

```
Object ExternalInterface.call(String name,  
                               Object args...)
```

参数

name

要调用的 JavaScript 函数的名称。

args...

转换并传递给 JavaScript 函数的 0 个或多个参数。

返回值

JavaScript 函数的返回值，转换为一个 ActionScript 值。

描述

这个静态函数供一个 Flash 电影中的 ActionScript 代码使用，以调用 Flash 电影所嵌入的 Web 浏览器中所定义的一个 JavaScript 函数。参阅 `ExternalInterface.addCallback()`，了解有关函数参数和返回值在 ActionScript 和 JavaScript 之间的转换。

FileUpload

参阅 `Input`

FlashPlayer

Flash 2.0

用于 Flash 电影的插件

方法

`GetVariable()`

返回 Flash 电影所定义的一个变量的值。

`GotoFrame()`

跳到一个电影中指定的帧数。

`IsPlaying()`

察看电影是否在播放。

`LoadMovie()`

载入一个附加的 Flash 电影，并且在当前电影的一个指定层或级显示它。

`Pan()`

移动电影的视口。

`PercentLoaded()`

确定载入了电影的多少。

`Play()`

开始播放一个电影。

`Rewind()`

倒片到电影的第一帧。

`SetVariable()`

设置一个 Flash 电影所定义的变量。

`SetZoomRect()`

设置 Flash 播放器显示电影的区域。

StopPlay()

停止电影。

TotalFrames()

返回电影的长度，以帧数表示。

Zoom()

改变电影的视口的大小。

描述

一个FlashPlayer对象表示嵌入到Web页面中的一个Flash电影，以及播放这个电影的Flash插件的一个实例。可以使用Document.getElementById()来获取一个FlashPlayer对象，例如，获取把电影嵌入到一个Web页面中的<embed>标记或<object>标记。

一旦你获得了一个FlashPlayer对象，就可以使用它所定义的各种JavaScript方法来控制电影的回放并且通过设置和查询变量来同电影交互。注意，FlashPlayer方法都以一个大写字母开头，这在客户端JavaScript并非常见的命名惯例。

参阅

第 23 章

FlashPlayer.GetVariable()

Flash 4

返回定义在一个Flash电影中的值

摘要

String GetVariable(String variableName)

参数

variableName

定义于Flash电影中的变量的名称。

返回值

指定的变量的值作为一个字符串返回，如果这个变量不存在，就返回null。

FlashPlayer.GotoFrame()

Flash 2

跳到一个电影的指定帧

摘要

void GotoFrame(integer frameNumber)

参数

frameNumber

要跳到的帧数。

描述

这个函数跳到电影的指定的帧，或者如果指定的帧还没有载入的话，就跳到最后一个可用的帧。要避免不确定的行为，可以用 `PercentLoaded()` 来确定电影有多少帧可用。

FlashPlayer.IsPlaying()

Flash 2

察看一个电影是否在播放

摘要

```
boolean IsPlaying()
```

返回值

如果电影在播放，就返回 `true`；否则返回 `false`。

FlashPlayer.LoadMovie()

Flash 3

载入附加的电影

摘要

```
void LoadMovie(integer layer,  
                String url)
```

参数

layer

在当前的电影之上第几层或级来显示新载入的电影。

url

要载入的电影的 URL。

描述

这个方法从指定的 *url* 载入一个附加的电影，并且在当前电影的指定 *layer* 显示它。

FlashPlayer.Pan()

Flash 2

移动电影的视口

摘要

```
void Pan(integer dx, integer dy,  
         integer mode)
```

参数

dx, *dy*

移动的水平量和垂直量。

mode

这个参数指定如何解释 dx 和 dy 值。如果这个参数是 0，那么其他参数被当作像素。如果这个参数是 1，则其他参数被当作百分比。

描述

Flashplayer 定义了一个视口，Flash 电影通过它可见。通常，这个视口的大小和电影的大小是相同的。但是，当 `SetZoomRect()` 或 `Zoom()` 被调用过的时候，情况就不是这样了：这些方法可以改变视口，以使得只能看到电影的一部分。

当视口只能显示电影的一部分的时候，这个 `Pan()` 方法就可以移动视口以使得电影的不同部分显示出来。但是，这个方法不允许移动超过电影的边缘。

参阅

`FlashPlayer.SetZoomRect()`、`FlashPlayer.Zoom()`

FlashPlayer.PercentLoaded()

Flash 2

确定电影载入了多少

摘要

`integer PercentLoaded()`

返回值

一个 0 到 100 之间的整数，表示电影已经载入到播放器中的近似百分比。

FlashPlayer.Play()

Flash 2

播放一个电影

摘要

`void Play()`

描述

开始播放电影。

FlashPlayer.Rewind()

Flash 2

倒片到电影的第一帧

摘要

`void Rewind()`

描述

这个方法把电影倒片到第一帧。

FlashPlayer.SetVariable()Flash 4

设置一个 Flash 电影所定义的变量

摘要

```
void SetVariable(String name, String value)
```

参数*name*

设置的变量的名称。

value

指定变量的新的值。这个值必须是一个字符串。

描述

这个方法为 Flash 电影所定义的命名变量指定一个新值。

FlashPlayer.SetZoomRect()Flash 2

设置一个电影的视口

摘要

```
void SetZoomRect(integer left, integer top,  
                  integer right, integer bottom)
```

参数*left, top*

视口的左上角的坐标，以缇 (twip) 为单位。

right, bottom

视口的右下角的坐标，以缇 (twip) 为单位。

描述

这个方法定义了电影的视口，也就是说，它定义了电影在 Flash Player 中显示的一个子区域。Flash 电影用一种叫做缇的单位来度量。一个点有 20 缇，而一英寸有 1 440 缇。

参阅

FlashPlayer.Pan()、FlashPlayer.Zoom()

FlashPlayer.StopPlay()Flash 2

停止电影

摘要

```
void StopPlay()
```

描述

停止电影。

FlashPlayer.TotalFrames()

Flash 2

返回电影的长度，以帧为单位

摘要

```
integer TotalFrames()
```

描述

这个方法以帧为单位返回电影的长度。

FlashPlayer.Zoom()

Flash.2

放大或缩小

摘要

```
void Zoom(integer percentage)
```

参数

percentage

缩放视口的百分比，当它为 0 的时候，表示将视口恢复到完整大小。

描述

这个方法把视口缩放到指定的百分比。在 1 到 99 之间的参数会减小视口的大小，这会使得电影中的对象显得比较大。大于 100 的参数会扩大视口（但不会超过电影的大小），并且会使得电影中的对象显得比较小。作为特定情况，参数 0 将视口恢复到完整的大小，以使得整个电影都可见。

Form

2 级 DOM HTML

HTML 文档中的一个 <form>

Node → Element → HTMLElement → Form

属性

readonly HTMLCollection elements

表单中所有元素的一个数组（HTMLCollection）。参阅 Form.elements[]。

readonly long length

表单中表单元素的数目。这是和 elements.length 同样的一个值。

除了这些属性，Form 还定义了下表所示的属性，它们和 HTML 属性直接对应：

属性	HTML 属性	描述
String acceptCharset	acceptcharset	服务器可接受的字符集
String action	action	表单句柄的 URL
String enctype	enctype	表单的编码
String method	method	用于表单提交的 HTTP 方法
String name	name	表单的名称
String target	target	表单提交结果的 Frame 或 Window 名

方法

reset()

把表单的所有输入元素重置为它们的默认值。

submit()

提交表单。

事件句柄

onreset

在重置表单元素之前调用。

onsubmit

在提交表单之前调用。该事件句柄可以在提交表单之前对表单的各个条目进行验证。

HTML 语法

Form 对象是由标准的 HTML 标记 `<form>` 创建的。表单所含有的各种输入元素则是由 `<input>` 标记、`<select>` 标记、`<textarea>` 标记和其他标记创建的。

```
<form
  [ name = "form_name" ]           // 在 JavaScript 中给表单命名
  [ target = "window_name" ]       // 要响应的窗口的名称
  [ action = "url" ]               // 表单提交的目的地 URL
  [ method = ( "get" | "post" ) ]  // 提交表单的方法
  [ enctype = "encoding" ]         // 表单数据的编码方法
  [ onreset = "handler" ]          // 当重置表单时调用的事件句柄
  [ onsubmit = "handler" ]         // 当提交表单时调用的事件句柄
>
// 这里是表单文本和输入元素
</form>
```

描述

Form 对象代表 HTML 文档中的一个 `<form>` 元素。elements 属性是一个 HTMLCollection，它提供了对表单的所有元素的方便访问。submit() 和 reset() 方法允许表单在程序控制下进行提交或重置。

文档中的每个表单都表示为数组 `Document.forms[]` 的一个元素。已命名的表单还被表示为文档的一个属性 `form_name`，属性名 `form_name` 由标记 `<form>` 的属性 `name` 指定。

表单中的元素（如按钮、输入字段、复选框，等等）都收集在数组 `Form.elements[]` 中。已命名的元素和已命名的表单一样，可以直接使用它的名称来引用，它的元素名就作为它在 `Form` 对象中的属性名。因此，要引用表单 `questionnaire` 中名为 `phone` 的 `Input` 对象元素，可以使用如下的 JavaScript 表达式：

```
document.questionnaire.phone
```

参阅

`Input`、`Select`、`Textarea`，第 18 章

Form.elements[]

2 级 DOM HTML

表单中的输入元素

摘要

`readonly HTMLCollection elements`

描述

`elements[]` 是出现在 HTML 表单中表单元素（如 `Input`、`Select` 和 `Textarea` 对象）的类似数组的 `HTMLCollection`。元素在数组中出现的顺序和它们在表单的 HTML 源代码中出现的顺序相同。每个元素都有一个 `type` 属性，其字符串值说明了元素的类型。

习惯用法

如果数组 `elements[]` 中的某个元素具有名称，这个名称是由 HTML 标记 `<input>` 的属性 `name="name"` 设置的，那么这个元素的名称就成了表单 `form` 的一个属性，使用这个属性可以引用该元素。因此，使用名称而不是数字来引用输入对象是完全可能的：

```
form.name
```

参阅

`Input`、`HTMLCollection`、`Select`、`Textarea`

Form.onreset

0 级 DOM

在重置表单时调用的处理器

摘要

`Function onreset`

描述

`Form` 对象的 `onreset` 属性指定了一个事件句柄函数。当用户单击了表单中的 **Reset** 按钮而

提交一个表单时，就会调用这个事件句柄函数。注意，这个句柄不会作为 `Form.reset()` 方法响应而被调用。如果 `onreset` 句柄返回 `false`，表单的元素就不会重置。参阅 `Element.addEventListener()` 了解注册事件句柄的另一种方法。

参阅

`Element.addEventListener()`、`Form.onsubmit`、`Form.reset`，第 17 章

Form.onsubmit

0 级 DOM

在提交表单时调用的事件句柄

摘要

Function onsubmit

描述

`Form` 对象的属性 `onsubmit` 指定了一个事件句柄函数。当用户单击了按钮 **Submit** 来提交表单时，就会调用这个处理器函数。注意，在调用方法 `Form.submit()` 时，该事件句柄函数并不会被调用。

如果 `onsubmit` 返回的是 `false`，表单的元素就不会被提交。如果该句柄返回了其他值，或者什么都没有返回，那么表单就会被正常提交。由于句柄 `onsubmit` 可以取消表单提交，所以它对于进行表单数据验证十分理想。

参阅 `Element.addEventListener()` 了解注册事件句柄的另一种方法。

参阅

`Element.addEventListener()`、`Form.onreset`、`Form.submit()`，第 17 章

Form.reset()

2 级 DOM HTML

将表单中的元素重置为它们的默认值

摘要

void reset();

描述

该方法把表单中的每个元素都恢复到它的默认值。调用这个方法的结果类似用户单击了 **Reset** 按钮的结果，只是表单的事件句柄 `onreset` 不被调用。

参阅

`Form.onreset`、`Form.submit()`

Form.submit()2 级 DOM HTML

将表单数据提交给 Web 服务器

摘要

```
void submit();
```

描述

这个方法把表单元素的值提交给由表单的 `action` 属性所指定的服务器。它提交表单的方式和用户单击了 **Submit** 按钮一样，只不过表单的 `onsubmit` 事件句柄没有触发。

参阅

`Form.onsubmit`、`Form.reset()`

Frame2 级 DOM HTML

HTML 文档中的一个 `<frame>`

Node → Element → HTMLElement → Frame

属性

正如在“描述”部分所介绍的，HTML 帧可以作为 Frame 对象或 Window 对象来访问。当作为 Frame 对象来访问的时候，它们从 HTMLElement 继承了属性，并且定义了如下的附加属性：

`Document contentDocument`

容纳帧内容的文档。

`String src`

帧内容自该 URL 下载。设置这个属性会导致帧载入一个新的文档。这个属性只是 HTML `<frame>` 标记的 `src` 属性的一个对应：它并不是 `Window.location` 这样的 `Location` 对象。

除了这些属性，Frame 对象还定义了如下的属性，这些直接和 HTML 属性对应：

属性	HTML 属性	描述
<code>String frameBorder</code>	<code>frameborder</code>	对于没有边框的帧设置为 0
<code>String longDesc</code>	<code>longdesc</code>	一个帧描述的 URL
<code>String marginHeight</code>	<code>marginheight</code>	帧空白的顶部和底部，以像素为单位
<code>String marginWidth</code>	<code>marginwidth</code>	帧空白的左边缘和右边缘，以像素为单位
<code>String name</code>	<code>name</code>	帧的名称，对于 0 级 DOM，查阅表单和链接目标
<code>boolean noResize</code>	<code>noresize</code>	如果为 <code>true</code> ，用户无法改变帧大小
<code>String scrolling</code>	<code>scrolling</code>	帧滚动策略：“auto”、“yes”或“no”

描述

帧有一个双重的特性，在客户端 JavaScript 中可以用 Window 或 Frame 对象来表示。在传统的 0 级 DOM 中，每个 <frame> 被当作一个独立的窗口，并且用一个 Window 对象表示，通过其名称来引用；或者作为包含 Window 的 frames[] 数组的一个元素。

```
// Get a frame as a Window object
var win1 = top.frames[0];      // Index frames[] array by number
var win2 = top.frames['f1'];   // Index frames[] array by name
var win3 = top.f1;             // Get frame as a property of its parent
```

当帧以这种方式访问，返回对象是一个 Window，前面列出的属性都不可用。相反，使用 Window 属性，如用 document 来访问帧的文档，用 location 来访问文档的 URL。

在 2 级 DOM 中，<frame> 元素可以用 ID 或标记名来访问，就好像其他任何文档元素那样：

```
// Get a frame as a Frame object
var frame1 = top.document.getElementById('f1');           // by id
var frame2 = top.document.getElementsByTagName('frame')[1]; // by tag name
```

当像这样使用 DOM 方法访问帧的时候，结果是一个 Frame 对象而不是一个 Window 对象，前面列出的属性也是可用的。使用 contentDocument 来访问帧的文档，使用 src 属性来查询文档的 URL 或者让帧载入一个新的文档。要获取一个 Frame 对象 f 的 Window 对象，则使用 f.contentDocument.defaultView。

• <iframe> 元素和 <frame> 元素非常相似。参阅 IFrame 参考页。

注意，同源策略（参见第 13.8.2 节）是用于多帧的文档。对于和包含脚本的内容具有不同载入来源的帧，浏览器不允许访问这些帧的内容。不管帧用一个 Window 还是一个 Frame 对象表示，这一点都是成立的。

参阅

IFrame、Window，第 14 章

Hidden

参见 Input

History

浏览器的 URL 历史

JavaScript 1.0

Object → History

摘要

window.history
history

属性

length

这个数字属性声明了浏览器历史列表中 URL 的个数。由于没有方法确定当前在这个列表中显示的文档的下标，所以知道列表的大小不是非常有用。

方法

back()

后退到以前已经访问过的 URL。

forward()

前进到以前已经访问过的 URL。

go()

转移到以前已经访问过的 URL。

描述

History 对象最初设计来表示窗口的浏览历史。但出于隐私权方面的原因，History 对象不再允许脚本访问已经访问过的实际 URL。唯一保持使用的功能只是 back()、forward() 和 go() 方法。

例子

下面一行代码执行的操作与单击 **Back** 按钮执行的操作一样：

```
history.back();
```

下面的代码执行的操作与点击两次 **Back** 按钮执行的操作一样：

```
history.go(-2);
```

参阅

Window 对象的 history 属性、Location

History.back()

JavaScript 1.0

返回到前一个 URL

摘要

```
history.back()
```

描述

方法 back() 会使 History 对象所属的窗口或帧再次访问位于当前 URL 之前的那个 URL (如果存在)。调用这个方法产生的效果与单击 **Back** 按钮产生的效果一样。它还等价于：

```
history.go(-1);
```

History.forward()JavaScript 1.0

访问下一个 URL

摘要

```
history.forward()
```

描述

方法 `forward()` 会使 `History` 对象所属的窗口或帧再次访问位于当前 URL 之后的那个 URL (如果存在)。调用这个方法产生的效果与单击 **Forward** 按钮产生的效果一样。它还等价于:

```
history.go(1);
```

注意, 如果用户没有使用过 **Back** 按钮或 **Go** 菜单在历史记录中移动, 而且 JavaScript 没有调用过方法 `History.back()` 或 `History.go()`, 那么调用方法 `forward()` 就不会产生任何效果, 因为浏览器已经处于 URL 列表的尾部, 没有可以前进访问的 URL 了。

History.go()JavaScript 1.0

再次访问一个 URL

摘要

```
history.go(relative_position)
```

```
history.go(target_string)
```

参数

relative_position

要访问的 URL 在 `History` 的 URL 列表中的相对位置。

target_string

要访问的 URL (或 URL 的子串), 如果一个相匹配的 URL 在历史列表中存在的话。

描述

方法 `History.go()` 的第一种形式有一个整型的参数, 指定了在 `History` 对象维护的历史列表中的位置, 该方法会使浏览器访问这个指定位置的 URL。参数值若为正数, 浏览器就会在历史列表中向前移动。参数值为负数, 浏览器就会在历史列表中向后移动。因此, 调用 `history.go(-1)` 等价于调用 `history.back()`, 而且这一调用的效果与用户单击了 **Back** 按钮相同。同样, 调用 `history.go(3)` 再次访问的 URL 与调用 `history.forward()` 三次所访问的 URL 相同。

方法 `History.go()` 的第二种形式采用一个字符串参数, 会使浏览器再次访问第一个含有指定的字符串的 URL (如最近一次访问过的 URL)。方法的这一表单没有很好地明确, 可能在不同的浏览器中工作方式不同。例如, Microsoft 的文档指明这个参数必须和前面指

定的站点的 URL 确实匹配，而较早的 Netscape 文档（Netscape 发明了 History 对象）则指出，参数必须是前面访问过的 URL 的一个子串。

HTMLCollection

2 级 DOM HTML

通过位置或名称访问的 HTML 元素的数组

Object → HTMLCollection

属性

readonly unsigned long length

集合中的元素数。

方法

item()

返回集合中指定位置的元素。也可以在数组方括号中指定位置，而不是明确地调用该方法。

namedItem()

返回集合中 name 属性或 id 属性具有指定值的元素。如果没有这样的元素，则返回 null。可以在方括号中使用元素名，而不是明确地调用该方法。

描述

HTMLCollection 对象是带有方法的 HTML 元素的集合，它可以通过元素在文档中的位置或它们的 id 属性、name 属性获取元素。在 JavaScript 中，HTMLCollection 对象的行为和只读的数组一样，可以使用 JavaScript 的方括号，通过编号或名称索引一个 HTMLCollection 对象，而不必调用 item() 方法和 namedItem() 方法。

HTMLDocument 接口的许多属性都是 HTMLCollection 对象，它提供了访问如表单、图像和链接等文档元素的便捷方式。Form.elements 和 Select.options 属性都是 HTMLCollection 对象。HTMLCollection 还提供了遍历 Table 的各行以及 TableRow 的各个单元格的一种方便方法。

HTMLCollection 对象是只读的，不能给它添加新元素，即使采用 JavaScript 的数组语法也是如此。它们是“活”的，即如果基本的文档改变了，那些改变通过所有 HTMLCollection 对象会立即显示出来。

HTMLCollection 对象和 NodeList 对象很相似，但后者可能既能用名称索引也能用数字索引。

例子

```
var c = document.forms;           // This is an HTMLCollection of form elements
var firstform = c[0];             // It can be used like a numeric array
var lastform = c[c.length-1];    // The length property gives the number of elements
var address = c["address"];       // It can be used like an associative array
var address = c.address;          // JavaScript allows this notation, too
```

参阅

HTMLDocument、NodeList

HTMLCollection.item()

2 级 DOM HTML

根据位置获取元素

摘要

```
Node item(unsigned long index);
```

参数

index

要返回的元素的位置。元素在HTMLCollection对象中出现的顺序与它们在文档源代码中出现的顺序一样。

返回值

指定 *index* 处的元素。如果 *index* 小于 0 或大于等于 *length* 属性，则返回 *null*。

描述

方法 *item()* 将返回HTMLCollection中的编码元素。在JavaScript中，将HTMLCollection作为数组处理，用数组的语法索引它更容易一些。

例子

```
var c = document.images; // This is an HTMLCollection
var img0 = c.item(0);    // You can use the item() method this way
var img1 = c[1];         // But this notation is easier and more common
```

参阅

NodeList.item()

HTMLCollection.namedItem()

2 级 DOM HTML

根据名称获取元素

摘要

```
Node namedItem(String name);
```

参数

name

要返回的元素的名称。

返回值

返回具有指定 `id` 或 `name` 属性值集合中的元素，如果 `HTMLCollection` 中没有这样的元素，则返回 `null`。

描述

该方法将查找并返回 `HTMLCollection` 中具有指定名称的元素。如果某个元素的 `id` 属性值是指定的名称，则返回那个元素。如果没有找到这样的元素，则返回 `name` 属性为指定值的元素。如果这样的元素也不存在，`namedItem()` 则返回 `null`。

注意，任何 HTML 元素都可能有 `id` 属性，但只有某些 HTML 元素（如表单、表单元素、图像和锚）可能具有 `name` 属性。

在 JavaScript 中，将 `HTMLCollection` 作为关联数组处理，并采用数组语法将 `name` 放在 `[]` 之间查找元素将更容易一些。

例子

```
var forms = document.forms;           // An HTMLCollection of forms
var address = forms.namedItem("address"); // Finds <form name="address">
var payment = forms["payment"] // Simpler syntax: finds <form name="payment">
var login = forms.login;              // Also works: finds <form name="login">
```

HTMLDocument

0 级 DOM

HTML 文档树的根

Node → Document → HTMLDocument

属性

Element[] `all` [IE 4]

这个非标准的属性是一个类似数组的对象，它提供了对文档中所有 `HTMLElement` 的访问。`all[]` 数组最初用于 IE4 中，尽管它已经被 `Document.getElementById()` 和 `Document.getElementsByTagName()` 这样的方法取代，它仍然用在已经部署的代码中。参阅 `HTMLDocument.all[]` 了解更多细节。

readonly `HTMLCollection` `anchors`

文档中所有 `Anchor` 对象的数组（`HTMLCollection` 对象）。

readonly `HTMLCollection` `applets`

文档中所有 `Applet` 对象的数组（`HTMLCollection` 对象）。

`HTMLElement` `body`

一个快捷属性，引用 `HTMLElement` 对象，该对象表示当前文档的 `<body>` 标记。对于定义了帧集的文档，该属性引用最外层的 `<frameset>` 标记。

`String` `cookie`

允许设置或查询当前文档的 `cookie`。参阅 `HTMLDocument.cookie` 了解详细信息。

String domain

提供下载文档的服务器的域名，如果不存在这样的域名，则为 null。这个属性可以用于在特定环境下减轻同源安全策略。参阅 HTMLDocument.domain 了解详细信息。

readonly HTMLCollection forms

文档中所有 Form 对象的数组（HTMLCollection 对象）。

readonly HTMLCollection images

文档中所有 Image 对象的数组（HTMLCollection 对象）。注意，为了与 0 级 DOM 兼容，该集合不包括由 <object> 标记定义的图像。

readonly String lastModified

指定了文档中最近修改的日期和时间。这个值来自于 Last-Modified HTTP 头部，它是由 Web 服务器发送的可选项。

readonly HTMLCollection links

文档中所有 Link 对象的数组（HTMLCollection 对象）。

readonly String referrer

链接到当前文档的文档 URL，如果当前文档不是通过超级链接访问的，则为 null。这个属性允许客户端 JavaScript 访问 HTTP 引用头部。注意拼写的不同处，HTTP 的头部有 3 个 r's，而 JavaScript 属性有 4 个 r's。

String title

文档的 <title> 标记的内容。

readonly String URL

文档的 URL。这个值往往和包含文档的 Window 的 location.href 属性相同。然而，当 URL 重定向发生的时候，这个 URL 属性保存了文档的实际 URL，而 location.href 保存了请求的 URL。

方法

close()

关闭由 open() 方法打开的文档流，强制性地显示所有缓存的输出内容。

getElementByname()

返回具有指定 name 属性值的所有元素的节点数组（一个 NodeList）。

open()

打开一个流，以便写入新文档的内容。注意，该方法将擦去当前文档的所有内容。

write()

把 HTML 文本串添加到打开的文档。

writeln()

把 HTML 文本串添加到打开的文档，后面附加一个换行符。

描述

该接口扩展了 Document 接口，定义了 HTML 专用的属性和方法。很多属性和方法都是 HTMLCollection 对象（实际上是可以使用数字或名称索引的只读数组），其中保存了对锚、表单、链接以及其他重要文档的可脚本元素的引用。这些集合属性都源自于 0 级 DOM。它们已经被 Document.getElementsByTagName() 所取代，但是仍然常常使用，因为它们很方便。

write() 方法值得注意，在文档载入和解析的时候，它允许一个脚本向文档中插入动态生成的内容。

注意，在 1 级 DOM 中，HTMLDocument 定义了一个名为 getElementById() 的非常有用的方法。在 2 级 DOM 中，这个方法已经被移到了 Document 接口，它现在由 HTMLDocument 继承而不是由它定义了。参阅 Document.getElementById() 了解更多细节。

参阅

Document、Document.getElementById()、Document.getElementsByTagName()

HTMLDocument.all[]

IE 4

文档中的所有 HTML 元素

摘要

```
document.all[i]
document.all[name]
document.all.tags(tagname)
```

描述

all[] 是一个多功能的类似数组的对象，它提供了对文档中所有 HTML 元素的访问。all[] 数组源自 IE 4 并且已经被很多其他的浏览器所采用。它已经被 Document 接口的标准的 getElementById() 方法和 getElementsByTagName() 方法以及 HTMLDocument 的标准 getElementsByName() 方法所取代。尽管如此，这个 all[] 数字在已有的代码中仍然使用。

all[] 包含的元素保持了最初的顺序，如果你知道它们在数组中的确切数字化位置，可以直接从数组中提起它们。然而，更为常见的是使用 all[] 数组，根据它们的 HTML 属性 name 或 id 来访问元素。如果多个元素拥有指定的名称，是用该名称来索引 all[]，得到共享同一名称的元素的一个数组。

参阅

Document.getElementById()、Document.getElementsByTagName()、HTMLElement

HTMLDocument.close()

0 级 DOM

关闭一个打开的文档并显示它

摘要

```
void close();
```

描述

该方法将关闭 `open()` 方法打开的文档流，并强制性地显示出所有缓存的输出内容。如果你使用 `write()` 方法动态地输出一个文档，必须记住当你这么做的时候要调用这个方法，以确保所有的文档内容都能显示。一旦调用了 `close()`，就不应该再次调用 `write()`，因为这会隐式地调用 `open()` 来擦除当前的文档并开始一个新的文档。

参阅

`HTMLDocument.open()`、`HTMLDocument.write()`

HTMLDocument.cookie

0 级 DOM

文档的 cookie

摘要

```
String cookie
```

描述

`cookie` 是一个字符串属性，使用它可以对应用于当前文档的 `cookie` 或 `cookies` 文件进行读操作、创建操作、修改操作以及删除操作。所谓 `cookie`，就是浏览器存储的少量具有名称的数据。它主要是给浏览器提供一块“内存”，以便它们能够在一个页面中使用另一个页面的数据，或者通过网络浏览会话恢复用户的优先级。为了使服务器端脚本能够读写 `cookie` 的值，`cookie` 数据会在适当的时候自动地在浏览器和服务器之间进行传输。客户端的 JavaScript 代码也可以使用这一属性来读写 `cookie` 数据。

`HTMLDocument.cookie` 属性的行为与常规的读 / 写属性不同。虽然既可以对 `HTMLDocument.cookie` 进行读操作，也可以进行写操作，但是从这个属性读到的值通常与写入的值不一样。要了解这个极其复杂的属性的完整细节，请参阅第 19 章。

习惯用法

`cookie` 主要用于不经常地存储少量数据。它们并不是通用的通信或程序设计机制，所以对它们的使用也是有限的。注意，Web 浏览器保留的每个服务器的 `cookie` 值不能超过 20 个，而且保留的 `cookie` 数据的名称 / 值对的长度也不能超过 4KB。

参阅

第 19 章

HTMLDocument.domain

0 级 DOM

文档的安全域

摘要

String domain

描述

根据 2 级 DOM HTML 标准, domain 属性只是一个只读的字符串, 它包含了 Web 服务器的主机名, 而文档正是从该服务器载入的。

这个属性还有另外一个重要的用法 (尽管这一用法还没有标准化)。同源安全策略 (第 13.8.2 节介绍) 阻止一个文档中的脚本读取另一个文档的内容 (例如显示于 <iframe> 中的一个文档), 除非两个文档具有相同的来源 (例如, 都从同一个 Web 服务器获取)。这给那些使用多个服务器的大网站带来了麻烦。例如, 位于主机 *www.oreilly.com* 上的脚本可能想要共享位于主机 *search.oreilly.com* 上某个脚本的属性。

属性 domain 解决了这一问题。可以设置这个属性, 但限制非常严格, 即只能将它设置为自己的域名后缀。例如, 从 *search.oreilly.com* 装载进来的脚本, 可以将自己的 domain 属性设置为 “oreilly.com”。如果来自 *www.oreilly.com* 的脚本在另一个窗口中运行, 而且也将自己的 domain 属性设置成了 “oreilly.com”, 那么这两个脚本就可以共享属性, 即使它们并非出自同一个服务器。但要注意, 来自 *search.oreilly.com* 的脚本不能将自己的 domain 属性设置为 “search.oreilly” 或 “.com”。

参阅

第 13.8.2 节

HTMLDocument.getElementsByName()

2 级 DOM HTML

找到具有指定 name 属性的元素

摘要

```
Element[] getElementsByName(String elementName);
```

参数

elementName

name 属性的期望值。

返回值

Element 对象的一个只读数组（技术上讲是一个 NodeList），其中的对象拥有一个 name 属性，该属性具有指定的值。如果没有找到这样的元素，则返回的数组是空的，其 length 为 0。

描述

该方法将搜索 HTML 文档树，查找 name 属性具有指定值的 Element 节点，返回包含所有匹配元素的 NodeList 对象（可以将其作为一个只读的数组）。如果没有匹配的元素，则返回的 NodeList 对象的 length 属性为 0。

不要将这个方法与 Document.getElementById() 方法混淆，后者查找的是具有惟一的 id 属性值的 Element 节点。也不要与 Document.getElementsByTagName() 方法混淆，它返回具有指定标记名的元素的 NodeList 对象。

参阅

Document.getElementById()、Document.getElementsByTagName()

HTMLDocument.open()

0 级 DOM

打开一个新文档，抹去当前文档的内容

摘要

```
void open();
```

描述

该方法将抹去当前 HTML 文档的内容，开始一个新文档，新文档用 write() 方法或 writeln() 方法编写。调用 open() 方法打开一个新文档并且用 write() 方法设置文档内容后，必须记住调用 close() 方法关闭文档，并且迫使它的内容显示出来。

属于被覆盖的文档的一部分的脚本或事件句柄不能调用该方法，因为脚本或事件句柄自身也会被覆盖。

例子

```
var w = window.open("");           // Open a new window
var d = w.document;                // Get its HTMLDocument object
d.open();                           // Open the document for writing
d.write("<h1>Hello world</h1>");    // Output some HTML to the document
d.close();                          // End the document and display it
```

参阅

HTMLDocument.close()、HTMLDocument.write()

HTMLDocument.write()0 级 DOM

向打开的文档添加 HTML 文本

摘要

```
void write(String text);
```

参数

text

要添加到文档的 HTML 文本。

描述

该方法将把指定的 HTML 文本添加到文档中。根据 DOM 标准，这个方法接受单个的字符串参数。而根据通常的经验，`write()` 可以接受任意多个参数。这些参数被转换为字符串，并按顺序添加到文档中。

`Document.write()` 通常按照两种方式之一使用。首先，它可以在一个 `<script>` 标记中的文档之上调用，或者在文档被解析的时候执行的一个函数的内部调用。在这种情况下，`write()` 方法写出自己的 HTML 输出，就好像该输出位于文件中调用该方法的代码的位置一样。

其次，可以使用 `Document.write()` 在调用正在运行的脚本之外的一个窗口、帧或者 `iframe` 中产生新的文档。如果目标文档是打开的，`write()` 向该文档添加内容。如果该文档没有打开，`write()` 丢弃已有的文档并打开一个新的（空的）文档，向其中添加自己的参数。

一旦文档打开，`Document.write()` 可以添加任意数量的输出到文档的末尾。当通过这一技术完全生成一个新文档的时候，这个文档必须调用 `Document.close()` 来关闭。注意，尽管调用 `open()` 是可选的，但调用 `close()` 不是可选的。

调用 `Document.write()` 的结果可能不会在目标文档中立即可见。这是因为一个 Web 浏览器可能缓冲文本以供解析并且大块大块地显示。调用 `Document.close()` 是显示地强制缓冲的输出“刷新”并显示的唯一方法。

参阅

`HTMLDocument.close()`、`HTMLDocument.open()`

HTMLDocument.writeln()0 级 DOM

把 HTML 文本和换行符添加到打开的文档

摘要

```
void writeln(String text);
```

参数

`text`

要添加到文档的 HTML 文本。

描述

该方法与 `HTMLDocument.write()` 方法相似。只是在附加的文本后，它还要附加一个换行符。在编写标记 `<pre>` 的内容时，该方法很有用。

参阅

`HTMLDocument.write()`

HTMLElement

2 级 DOM HTML

HTML 中的一个元素

Node → Element → HTMLElement

属性

一个 HTML 文档的每个元素都有和元素的 HTML 属性对应的属性。所有 HTML 标记都支持的属性在这里列出。其他的属性，都特定于某种具体的 HTML 标记，在“描述”部分后面的一个长长的表格中列出。HTMLElement 从 Node 和 Element 继承了很多有用的标准属性，也实现了下面所描述的几个非标准属性：

`String className`

元素的 `class` 属性的值，声明了 0 个或多个空白隔开的 CSS 类的名称。注意，该属性名不是“`class`”，因为“`class`”是 JavaScript 中的保留字。

`CSS2Properties currentStyle`

这一特定于 IE 的属性应用于元素的所有 CSS 属性的级联组。它是 `Window.getComputedStyle()` 的一个仅用于 IE 的替代。

`String dir`

元素的 `dir` 属性的值，声明了文档文本的方向。

`String id`

`id` 属性的值。在一个文档中，没有两个元素具有相同的 `id` 值。

`String innerHTML`

一个可读可写的字符串，声明了包含在元素中的纯文本，不包括元素自身的开始标记和结束标记。查询这一属性会将元素的内容作为一个 HTML 文本串返回。将这一属性设置为一个 HTML 文本串，则可以用 HTML 的解析表示来替换元素的内容。在文档载入的时候，不能设置这一属性（要了解这一功能，参阅 `HTMLDocument.write()`）。这是一个源自 IE4 的非标准的属性。它已经被所有的现代浏览器所实现。

`String lang`

`lang` 属性的值，声明了元素内容的语言代码。

`int offsetHeight, offsetWidth`

元素及其所有内容的高度和宽度，以像素为单位，包括元素的CSS补白和边框，但不包括页边距。这是非标准的但却得到很好支持的属性。

`int offsetLeft, offsetTop`

元素的CSS边框左上角的X坐标和Y坐标，相对于包含元素的`offsetParent`。这是非标准的但却得到很好支持的属性。

`Element offsetParent`

声明了定义坐标系统的包含元素，属性`offsetLeft`和属性`offsetTop`都是以这个坐标系统来计量的。对于大多数元素来说，`offsetParent`指的就是包含它们的Document对象。但是，如果一个元素的包含器是动态定位的，那么`offsetParent`就是这个动态定位的元素。在一些浏览器中，表的单元格是相对于包含它们的行来定位的，而不是相对于包含文档来定位的。参阅第16章中可移植地使用这一属性的一个例子。这是非标准的但却得到很好支持的属性。

`int scrollHeight, scrollWidth`

一个元素的完整的高度和宽度，以像素为单位。当一个元素拥有滚动条的时候（例如，由于CSS的`overflow`属性），这些属性和`offsetHeight`与`offsetWidth`不同，`offsetHeight`与`offsetWidth`只是报告元素的可见部分的大小。这是非标准的但却得到很好支持的属性。

`int scrollLeft, scrollTop`

已经滚动过元素的左边界或滚动过元素的上边界的像素数。只有在元素有滚动条的时候，例如，元素的CSS `overflow`属性设置为`auto`的时候，这些像素才有用。这些属性也只在文档的`<body>`或`<html>`标记上定义（这和浏览器有关），并且一起来指定滚动文档的量。注意，这些属性并不会指定一个`<iframe>`标记中的滚动量。这是非标准的但却得到很好支持的属性。

`CSS2Properties style`

为当前元素设置内联CSS样式的`style`属性的值。注意，这个属性的值不是一个字符串。参阅`CSS2Properties`了解更多细节。

`String title`

元素的`title`属性的值。当鼠标悬停在元素上的时候，很多浏览器在元素的“工具提示”中显示这一属性的值。

方法

`HTMLElement`对象继承了`Node`和`Element`的标准方法。某些类型的元素实现了特定于标记的方法，这在描述部分后面的长长的表中列出，并且在`Form`、`Input`和`Table`等其他参考页介绍。大多数现代浏览器也都实现了如下的非标准方法：

`scrollIntoView()`

滚动文档，使该元素出现在窗口的顶部或底部。

事件句柄

响应纯鼠标和键盘事件的所有 HTML 元素都可以触发这里列出的事件句柄。某些元素，如链接和按钮，当这些事件发生的时候执行默认操作。对于像这样的元素，更多的细节可以在具体元素的参考页找到，例如，参见 `Input` 和 `Link`。

`onclick`

当用户单击该元素时调用。

`ondblclick`

当用户双击该元素时调用。

`onkeydown`

当用户按下一个键时调用。

`onkeypress`

当用户按下一个键或放开一个键时调用。

`onkeyup`

当用户放开一个键时调用。

`onmousedown`

当用户按下一个鼠标按钮时调用。

`onmousemove`

当用户移动鼠标时调用。

`onmouseout`

当用户把鼠标移开当前元素时调用。

`onmouseover`

当用户把鼠标移动过一个元素时调用。

`onmouseup`

当用户放开一个鼠标按钮时调用。

描述

HTML 文档中的每个标记都由一个 `HTMLElement` 对象来表示。`HTMLElement` 定义了属性，这些属性代表了所有 HTML 元素所共有的属性。如下的 HTML 标记所拥有的属性都不外乎前面列出的那些，这些标记将在 `HTMLElement` 接口中详细描述：

<code><abbr></code>	<code><acronym></code>	<code><address></code>	<code></code>
<code><bdo></code>	<code><big></code>	<code><center></code>	<code><cite></code>
<code><code></code>	<code><dd></code>	<code><dfn></code>	<code><dt></code>
<code></code>	<code><i></code>	<code><kbd></code>	<code><noframes></code>
<code><noscript></code>	<code><s></code>	<code><samp></code>	<code><small></code>
<code></code>	<code><strike></code>	<code></code>	<code><sub></code>
<code><sup></code>	<code><tt></code>	<code><u></code>	<code><var></code>

大多数标记定义的属性都不外乎前面明确地列出的那些。2 级 DOM HTML 规范为这些标记定义了特定于标记的接口，以便所有标准 HTML 属性都有一个对应的标准 JavaScript 属性。通常，一个名为 *T* 的标记拥有一个名为 *HTMLTElement* 的特定于标记的接口。例如，`<head>` 标记通过 *HTMLHeadElement* 接口来表示。在少数情况下，两个或多个相关的标记共享同一个单个的接口，例如，在 `<h1>` 到 `<h6>` 标记的情况中，它们都通过 *HTMLHeadingElement* 接口表示。

大多数标记指定的接口不过是为 HTML 标记的属性定义一个 JavaScript 属性。JavaScript 属性和 HTML 属性名称相同，采用全小写的形式（如 `id`），在 HTML 属性名由多个单词构成时，JavaScript 属性采用大小写混合的形式（如 `longDesc`）。当 HTML 属性名是 Java 或 JavaScript 的保留字时，JavaScript 属性名有轻微改变。例如，`<label>` 标记和 `<script>` 标记的 `for` 属性将成为 *HTMLLabelElement* 和 *HTMLScriptElement* 接口的 `htmlFor` 属性，因为 `for` 是保留字。这些属性的含义直接对应于 HTML 标准定义的 HTML 属性，对它们的介绍不属于本书的范围。

下表列出了所有具有相应的 *HTMLElement* 子接口的 HTML 标记。对于每个标记来说，该表列出了接口名和它定义的属性及方法名。如果没有明确说明，所有属性都是可读可写的字符串。对于不是可读可写的字符串的属性，属性名前的方括号中声明了它的属性类型。表中少数在 HTML 4 种不再使用的标记和属性使用 * 标记出来。

由于这些接口和它们的属性都非常直接地映射到 HTML 元素和属性，所以大多数接口在本书中都没有它们自己的参考页，并且你需要通过一本 HTML 参考书来了解细节。例外的是那些所代表的标记对于客户端 JavaScript 程序员特别重要的接口，例如代表 `<form>` 和 `<input>` 标记的接口。这些标记在本书中介绍了，分别在不包含“HTML”前缀或“Element”后缀的条目下。例如，参见 *Anchor*、*Applet*、*Canvas*、*Form*、*Image*、*Input*、*Link*、*Option*、*Select*、*Table* 和 *Textarea* 条目。

HTML 标记	DOM 接口、属性和方法
所有标记	<i>HTMLElement</i> : <code>id</code> , <code>title</code> , <code>lang</code> , <code>dir</code> , <code>className</code>
<code><a></code>	<i>HTMLAnchorElement</i> : <code>accessKey</code> , <code>charset</code> , <code>coords</code> , <code>href</code> , <code>hreflang</code> , <code>name</code> , <code>rel</code> , <code>rev</code> , <code>shape</code> , <code>[long]</code> <code>tabIndex</code> , <code>target</code> , <code>type</code> , <code>blur()</code> , <code>focus()</code>
<code><applet></code>	<i>HTMLAppletElement</i> *: <code>align</code> *, <code>alt</code> *, <code>archive</code> *, <code>code</code> *, <code>codeBase</code> *, <code>height</code> *, <code>hspace</code> *, <code>name</code> *, <code>object</code> *, <code>vspace</code> *, <code>width</code> *
<code><area></code>	<i>HTMLAreaElement</i> : <code>accessKey</code> , <code>alt</code> , <code>coords</code> , <code>href</code> , <code>[boolean]</code> <code>noHref</code> , <code>shape</code> , <code>[long]</code> <code>tabIndex</code> , <code>target</code>
<code><base></code>	<i>HTMLBaseElement</i> : <code>href</code> , <code>target</code>
<code><basefont></code>	<i>HTMLBaseFontElement</i> *: <code>color</code> *, <code>face</code> *, <code>size</code> *
<code><blockquote></code> , <code><q></code>	<i>HTMLQuoteElement</i> : <code>cite</code>
<code><body></code>	<i>HTMLBodyElement</i> : <code>aLink</code> *, <code>background</code> *, <code>bgColor</code> *, <code>link</code> *, <code>text</code> *, <code>vLink</code> *

HTML 标记	DOM 接口、属性和方法
 	HTMLBRElement: clear*
<button>	HTMLButtonElement: [readonly HTMLFormElement] form, accessKey, [boolean] disabled, name, [long] tabIndex, [readonly] type, value
<caption>	HTMLTableCaptionElement: align*
<col>,	HTMLTableColElement: align, ch, chOff, [long] span, vAlign, width
<colgroup>	
, <ins>	HTMLModElement: cite, dateTime*
<dir>	HTMLDirectoryElement*: [boolean] compact*
<div>	HTMLDivElement: align*
<dl>	HTMLDListElement: [boolean] compact*
<fieldset>	HTMLFieldSetElement: [readonly HTMLFormElement] form
	HTMLFontElement*: color*, face*, size*
<form>	HTMLFormElement: [readonly HTMLCollection] elements, [readonly long] length, name, acceptCharset, action, enctype, method, target, submit(), reset()
<frame>	HTMLFrameElement: frameBorder, longDesc, marginHeight, marginWidth, name, [boolean] noResize, scrolling, src, [readonly Document] contentDocument
<frameset>	HTMLFrameSetElement: cols, rows
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	HTMLHeadingElement: align*
<head>	HTMLHeadElement: profile
<hr>	HTMLHRElement: align*, [boolean] noShade*, size*, width*
<html>	HTMLHtmlElement: version*
<iframe>	HTMLIFrameElement: align*, frameBorder, height, longDesc, marginHeight, marginWidth, name, scrolling, src, width, [readonly Document] contentDocument
	HTMLImageElement: align*, alt, [long] border*, [long] height, [long] hspace*, [boolean] isMap, longDesc, name, src, useMap, [long] vspace*, [long] width
<input>	HTMLInputElement: defaultValue, [boolean] defaultChecked, [readonly HTMLFormElement] form, accept, accessKey, align*, alt, [boolean] checked, [boolean] disabled, [long] maxLength, name, [boolean] readOnly, size, src, [long] tabIndex, type, useMap, value, blur(), focus(), select(), click()

HTML 标记	DOM 接口、属性和方法
<ins>	See
<isindex>	HTMLIsIndexElement*: [readonly HTMLFormElement] form, prompt*
<label>	HTMLLabelElement: [readonly HTMLFormElement] form, accessKey, htmlFor
<legend>	HTMLLegendElement: [readonly HTMLFormElement] form, accessKey, align*
	HTMLLIElement: type*, [long] value*
<link>	HTMLLinkElement: [boolean] disabled, charset, href, hreflang, media, rel, rev, target, type
<map>	HTMLMapElement: [readonly HTMLCollection of HTMLAreaElement] areas, name
<menu>	HTMLMenuElement*: [boolean] compact*
<meta>	HTMLMetaElement: content, httpEquiv, name, scheme
<object>	HTMLObjectElement: code, align*, archive, border*, codeBase, codeType, data, [boolean] declare, height, hspace*, name, standby, [long] tabIndex, type, useMap, vspace*, width, [readonly Document] contentDocument
	HTMLOListElement: [boolean] compact*, [long] start*, type*
<optgroup>	HTMLOptGroupElement: [boolean] disabled, label
<option>	HTMLOptionElement: [readonly HTMLFormElement] form, [boolean] defaultSelected, [readonly] text, [readonly long] index, [boolean] disabled, label, [boolean] selected, value
<p>	HTMLParagraphElement: align*
<param>	HTMLParamElement: name, type, value, valueType
<pre>	HTMLPreElement: [long] width*
<q>	See <blockquote>
<script>	HTMLScriptElement: text, htmlFor, event, charset, [boolean] defer, src, type
<select>	HTMLSelectElement: [readonly] type, [long] selectedIndex, value, [readonlylong] length, [readonly HTMLFormElement] form, [readonly HTMLCollection of HTMLOptionElement] options, [boolean] disabled, [boolean] multiple, name, [long] size, [long] tabIndex, add(), remove(), blur(), focus()
<style>	HTMLStyleElement: [boolean] disabled, media, type

HTML 标记	DOM 接口、属性和方法
<table>	HTMLTableElement : [HTMLTableCaptionElement] caption, [HTMLTableSectionElement] tHead, [HTMLTableSectionElement] tFoot, [readonly HTMLCollection of HTMLTableRowElement] rows, [readonly HTMLCollection of HTMLTableSectionElement] tBodies, align*, bgColor*, border, cellPadding, cellSpacing, frame, rules, summary, width, createTHead(), deleteTHead(), createTFoot(), deleteTFoot(), createCaption(), deleteCaption(), insertRow(), deleteRow()
<tbody>, <tfoot>, <thead>	HTMLTableSectionElement : align, ch, chOff, vAlign, [readonly HTMLCollection of HTMLTableRowElement] rows, insertRow(), deleteRow()
<td>, <th>	HTMLTableCellElement : [readonly long] cellIndex, abbr, align, axis, bgColor*, ch, chOff, [long] colSpan, headers, height*, [boolean] noWrap*, [long] rowSpan, scope, vAlign, width*
<textarea>	HTMLTextAreaElement : defaultValue, [readonly HTMLFormElement] form, accessKey, [long] cols, [boolean] disabled, name, [boolean] readOnly, [long] rows, [long] tabIndex, [readonly] type, value, blur(), focus(), select()
<tfoot>	See <tbody>
<th>	See <td>
<thead>	See <tbody>
<title>	HTMLTitleElement : text
<tr>	HTMLTableRowElement : [readonly long] rowIndex, [readonly long] sectionRowIndex, [readonly HTMLCollection of HTMLTableCellElement] cells, align, bgColor*, ch, chOff, vAlign, insertCell(), deleteCell()
	HTMLUListElement : [boolean] compact*, type*

* 表示不再使用的元素和属性

参阅

Anchor、Element、Form、HTMLDocument、Image、Input、Link、Node、Option、Select、Table、TableCell、TableRow、TableSection、Textarea, 第15章

HTMLElement.onclick

0 级 DOM

当用户单击该元素时调用的事件句柄

摘要

Function onclick

描述

HTMLElement 对象的 onclick 属性声明了用户单击元素时要调用的事件句柄。注意，onclick 与 onmousedown 不同。单击事件是在同一元素上发生了鼠标按下事件之后又发生了鼠标放开事件时才发生的。

参阅

Element.addEventListener()、Event、MouseEvent，第 17 章

HTMLElement.ondblclick

0 级 DOM

当用户双击该元素时调用的事件句柄

摘要

Function ondblclick

描述

HTMLElement 对象的 ondblclick 属性声明了一个事件句柄函数，该函数是在用户双击元素元素时调用的。

参阅

Element.addEventListener()、Event、MouseEvent，第 17 章

HTMLElement.onkeydown

0 级 DOM

当用户按下一个键时调用的事件句柄

摘要

Function onkeydown

描述

HTMLElement 对象的 onkeydown 属性声明了一个事件句柄函数，如果当该元素有键盘焦点的时候用户按下了某个键，该事件句柄函数就会被调用。确定哪个键或几个键被按下，多少和浏览器有关。参阅第 17 章了解详细信息。

onkeydown 句柄通常是功能按键的首选句柄，但是使用 onkeypress 响应一般的文字和数字键按下。

参阅

HTMLElement.onkeypress，第 17 章

HTMLElement.onkeypress

0 级 DOM

当用户按下一个键时调用的事件句柄

摘要

Function onkeypress

描述

HTMLElement 对象的 onkeypress 属性声明了一个事件句柄函数。如果当元素拥有键盘焦点的时候，用户按下并释放了一个键，该句柄就会被调用。keypress 事件在一个 keydown 事件之后并且在响应的 keyup 事件之前发生。keypress 事件和 keydown 事件很相似，尽管一个 keypress 事件通常对文字和数字按键更为有用，而一个 keydown 句柄可能比功能按键更为有用。

确定哪个按键按下，以及此时什么修改键有效，这多少有些复杂，而且和浏览器相关。参阅第 17 章了解详细内容。

参阅

HTMLElement.onkeydown, 第 17 章

HTMLElement.onkeyup

0 级 DOM

当用户放开一个键时调用的事件句柄

摘要

Function onkeyup

描述

HTMLElement 对象的 onkeyup 属性声明了一个事件句柄函数，如果当元素拥有键盘焦点时用户释放了某个键，该句柄就会被调用。

参阅

HTMLElement.onkeydown, 第 17 章

HTMLElement.onmousedown

0 级 DOM

当用户按下一个鼠标按钮时调用的事件句柄

摘要

Function onmousedown

描述

HTMLElement 对象的 onmousedown 属性声明了一个事件句柄函数，当用户在元素上按下了一个鼠标按钮时，该句柄就会被调用。

参阅

`Element.addEventListener()`、`Event`、`MouseEvent`，第 17 章

HTMLElement.onmousemove

0 级 DOM

当用户在该元素中移动鼠标时调用的事件句柄

摘要

`Function onmousemove`

描述

`HTMLElement` 对象的 `onmousemove` 属性声明了一个事件句柄函数，当用户在元素中移动鼠标指针时，该句柄就会被调用。

如果你定义了一个 `onmousemove` 事件句柄函数，那么当鼠标在元素之中移动时就会生成大量的鼠标移动事件，它们都会被一一地汇报出来。在编写这个事件句柄函数要调用的函数时，一定要谨记这一点。如果你对跟踪鼠标拖拽感兴趣，注册这种类型的一个句柄以便响应一个 `mousedown` 事件，然后当 `mouseup` 事件到达后，取消注册。

参阅

`Element.addEventListener()`、`Event`、`MouseEvent`，第 17 章

HTMLElement.onmouseout

0 级 DOM

当用户把鼠标指针移出该元素时调用的事件句柄

摘要

`Function onmouseout`

描述

`HTMLElement` 对象的 `onmouseout` 属性声明了一个事件句柄函数，当用户把鼠标指针移出了元素 `element` 的时候，该句柄就会被调用。

参阅

`Element.addEventListener()`、`Event`、`MouseEvent`，第 17 章

HTMLElement.onmouseover

0 级 DOM

当用户将鼠标指针移动过该元素时调用的事件句柄

摘要

`Function onmouseover`

描述

HTMLElement 对象的 `onmouseover` 属性声明了一个事件句柄函数，当用户把鼠标指针移动过元素时，该句柄就会被调用。

参阅

`Element.addEventListener()`、`Event`、`MouseEvent`，第 17 章

HTMLElement.onmouseup

0 级 DOM

当用户放开一个鼠标按钮时调用的句柄

摘要

Function onmouseup

描述

HTMLElement 对象的 `onmouseup` 属性声明了一个事件句柄函数，当用户在元素上放开一个鼠标按钮的时候，该句柄就会被调用。

参阅

`Element.addEventListener()`、`Event`、`MouseEvent`，第 17 章

HTMLElement.scrollIntoView()

Firefox 1.0, IE 4, Safari 2.02, Opera 8.5

使元素可见

摘要

`element.scrollIntoView(top)`

参数

top

一个可选的布尔参数，声明了元素应该滚动到屏幕的顶部 (`true`) 还是底部 (`false`)。这个参数并不是被所有的浏览器支持，并且，靠近一个文档的顶部或底部的元素也无法滚动到窗口相反的一边，因此这个参数应该只被看作是一个提示。

描述

如果一个 HTML 元素当前在窗口中不可见，这个方法可以滚动文档以使该元素变得可见。*top* 参数是一个可选的提示，它和元素是否应该滚动到窗口的顶部或底部相关。对于接受键盘焦点的元素，如 `Link` 和 `Input` 元素，`focus()` 隐式地执行相同的滚动到视图的操作。

参阅

`Anchor.focus()`、`Input.focus()`、`Link.focus()`、`Window.scrollTo()`

IFrame

2 级 DOM HTML

HTML 文档中的一个 <iframe>

Node → Element → HTMLElement → IFrame

属性

正如后面的“描述”部分所介绍，iframe 元素可以作为一个 IFrame 对象或 Window 对象访问。当作为一个 IFrame 对象访问的时候，它们从 HTMLElement 继承了属性，并且定义了如下的额外属性：

Document contentDocument

保存 <iframe> 的内容的文档。

String src

iframe 内容自这个 URL 载入。设置这个属性会导致 iframe 载入一个新的文档。这个属性只是与 HTML <iframe> 标记的 src 属性相对应。

除了这些属性，IFrame 对象还定义了如下属性，它们直接和 <iframe> 标记的 HTML 属性对应：

属性	HTML 属性	描述
deprecated String align	align	内联内容的对齐
String frameBorder	frameborder	对于无边框的帧，设置为 0
String height	height	视口的高度，以像素或百分比表示
String longDesc	longdesc	帧描述的 URL
String marginHeight	marginheight	帧边缘的顶部和底部，以像素为单位
String marginWidth	marginwidth	帧边缘的左边和右边，以像素为单位
String name	name	帧的名称。对于 0 级 DOM 查看表单和链接目标
String scrolling	scrolling	帧滚动策略：“auto”、“yes”或“no”
String width	width	视口的宽度，以像素或百分比表示

描述

除了 HTML 属性有略为不同，<iframe> 元素的行为和客户端 JavaScript 中的 <frame> 非常相似。<iframe> 元素成为包含窗口的 frames[] 数组的一部分。当通过数组访问的时候，它们通过 Window 对象表示，前面列出的属性都不可用。

当一个 <iframe> 元素作为一个文档元素通过 ID 或标记名访问的时候，它通过一个 IFrame 对象表示，带有前面列出的属性。使用 src 来查询或设置 <iframe> 的 URL，使用 contentDocument 来访问 iframe 的内容。可是别忘了，同源策略（参见第 13.8.2 节）可能阻止对 contentDocument 的访问。

参见

Frame、Window, 第 14 章

Image

2 级 DOM HTML

嵌在 HTML 文档中的图像

Node → Element → HTMLElement → Input

构造函数

`new Image(integer width, integer height)`

参数

width 和 *height*

指定图像的宽度和高度, 这两个参数可选。

属性

String *name*

这个属性指定了图像对象的名称。如果一个 `` 有一个 `name` 属性, 就可以把相应的 `Image` 对象作为 `Document` 对象的命名属性来访问。

String *src*

一个可读可写的字符串, 声明了浏览器显示的图像的 URL。这个属性的初始值由标记 `` 的 `src` 性质设置。当把这个属性设置为新图像的 URL 时, 浏览器就会把那幅新图像装载并显示出来。这对于更新网页的图形外观以响应用户的动作非常有用。

除了这些属性, `Image` 对象还支持如下的属性, 它们只是 HTML 属性的简单对应:

属性	HTML 属性	描述
deprecated String <i>align</i>	<i>align</i>	和内联内容对其相关
String <i>alt</i>	<i>alt</i>	当图像无法显示时的替代文本
deprecated String <i>border</i>	<i>border</i>	图像边框大小
long <i>height</i>	<i>height</i>	图像高度, 以像素为单位
deprecated long <i>hspace</i>	<i>hspace</i>	左边缘和右边缘, 以像素为单位
boolean <i>isMap</i>	<i>ismap</i>	是否使用一个服务器端图像映射
String <i>longDesc</i>	<i>longdesc</i>	一个较长图像描述的 URI
String <i>useMap</i>	<i>usemap</i>	为图像指定一个客户端图像映射
deprecated long <i>vspace</i>	<i>vspace</i>	边缘的顶端和底端, 以像素为单位
long <i>width</i>	<i>width</i>	图像宽度, 以像素为单位

事件句柄

`Image` 从 `HTMLElement` 继承事件句柄并定义了如下事件句柄:

onabort

如果在图像完全载入之前页面载入被停止了, 则调用该事件句柄。

onerror

如果在装载图像的过程中发生了错误, 则调用该事件句柄。

onload

在成功地装载了图像时调用的事件句柄。

HTML 语法

Image 对象是由标准的 HTML 标记 `` 创建的。在下面的语法中, 某些 `` 属性被省略了, 这是因为 JavaScript 通常不使用它们:

```

```

描述

Image 对象表示使用 `` 标记嵌入到一个 HTML 文档中的图像。出现在文档中的图像都集中在 `document.images[]` 数组里。具有 `name` 属性的 Image 也可以通过 Document 对象的命名属性访问, 例如:

```
document.images[0] // The first image in the document
document.banner    // An image with name="banner"
```

Image 对象的 `src` 属性是最有趣的属性。当你设置该属性, 浏览器载入并显示新值指定的图像。这使得图像滚动和动画等可视化效果成为可能。参阅第 22 章中的例子。

可以在 JavaScript 代码中用 `Image()` 构造函数创建屏幕外 Image 对象。注意, 这个构造函数没有参数指定要装载的图像。与在 HTML 文件中创建图像一样, 应该明确地把 `src` 属性设置成要创建的图像, 这样就可以告诉浏览器要装载图像。没有一种方法可以让浏览器显示这样创建一个 Image 对象。你所能做的就是迫使 Image 对象通过设置 `src` 属性把一幅图像装载进来。不过这非常有用, 因为这样它就把一幅图像装载到浏览器的缓存中。如果这幅图像的 URL 和稍后用一个实际 `` 标记所使用的 URL 相同, 那么它已经被预装载进来了, 很快就会显示出来。

参阅

第 22 章

Image.onabort

2 级 DOM Events

当用户放弃图像的装载时调用的事件句柄

摘要

Function onabort

描述

Image对象的属性onabort声明了一个事件句柄函数，当用户在图像完成载入之前放弃图像的装载（如单击了 **Stop** 按钮）时，就会调用这个句柄。

Image.onerror

2 级 DOM Events

在装载图像的过程中发生错误时调用的事件句柄

摘要

Function onerror

描述

Image对象的属性onerror声明了一个事件句柄函数，当装载图像的过程中发生了错误时就会调用这个句柄。参阅 Window.onerror。

Image.onload

2 级 DOM Events

当图像装载完毕时调用的事件句柄

摘要

Function onload

描述

Image对象的属性onload声明了一个事件句柄函数，当图像装载完毕的时候就会调用这个句柄。参阅 Window.onload。

Input

2 级 DOM HTML

HTML 表单中的输入元素

Node → Element → HTMLElement → Input

属性

String accept

当 type 为 “file”，这个属性是一个用逗号分隔开的 MIME 类型的列表，它指定了可能上传的文件的类型。映射 accept 属性。

String accessKey

浏览器可以用来把键盘焦点转换到这个输入元素的键盘快捷键。映射 accesskey 属性。

deprecated String align

和包围文本或元素的左漂移或右漂移相关的垂直对齐。映射 align 属性。

String alt

当浏览器无法显示这个输入元素时的替代文本。当 type 为 “image” 时特别有用。映射 alt 属性。

boolean checked

当 type 为 “radio” 或 “checkbox” 时，这个属性指定了元素是否 “选中”。设置这一属性会改变输入元素的视觉效果。映射 checked 属性。

boolean defaultChecked

当 type 为 “radio” 或 “checkbox” 时，这个属性保存了元素出现在文档源中的 checked 属性的初始值。当表单被重置，checked 属性恢复为这个属性的值。改变这个属性的值就会改变 checked 属性的值以及元素的当前选中状态。

string defaultValue

当 type 为 “text”、“password” 或 “file” 的时候，这个属性保存了这些元素显示的初始值。当表单被重置，这些元素恢复为这个值。改变这个属性的值也会改变 value 属性以及当前显示的值。

boolean disabled

如果为 true，输入元素被关闭并且不能用来进行用户输入。映射 disabled 属性。

readonly HTMLFormElement form

表示 <form> 元素的 Form 对象包含了这个输入元素，如果输入元素不在一个表单中，则为 null。

long maxLength

如果 type 是 “text” 或 “password”，这个属性指定了允许用户输入的最大字符数。注意，这和 size 属性不同。映射 maxlength 属性。

String name

输入元素的名称，由 name 属性指定。参阅后面的 “描述” 部分了解有关表单元素名称的更多细节。

boolean readOnly

如果为 true 并且 type 为 “text” 或 “password”，不允许用户向元素输入文本。映射 readonly 属性。

unsigned long size

如果 type 是 “text” 或 “password”，这个属性指定了元素的字符宽度。映射 size 属性。参阅 maxLength。

String src

对于类型为 “image” 的输入元素，指定要显示的图像的 URL。映射 src 属性。

long tabIndex

输入元素在切换顺序中的位置。映射 tabindex 属性。

String type

输入元素的类型。映射 type 属性。参阅“描述”部分了解关于表单元素类型的更多细节。

String useMap

对于 type 为“image”的元素，这个属性指定了一个 <map> 元素的名称，该元素为图像元素提供一个客户端图像映射。

String value

当表单提交的时候，传递给 Web 服务器的值。对于 type 为“text”、“password”或“file”的元素来说，这个属性是输入元素包含的可编辑文本。对于 type 为“button”、“submit”或“reset”的元素来说，这是出现在按钮上的一个（不可编辑的）标签。出于安全原因，FileUpload 元素的属性 value 应该是只读的。同样，Password 元素这一属性的返回值不应该包含用户的实际输入。

方法

blur()

将键盘焦点从元素中移开。

click()

如果 type 为“button”、“checkbox”、“radio”、“reset”或“submit”，该方法模拟在元素上的一次鼠标单击。

focus()

把键盘焦点赋予该元素。

select()

如果 type 为“file”、“password”或“text”，这个方法选中元素显示的所有文本。在很多浏览器中，这意味着当用户接着输入一个字符，选中的文本被删除掉并且用新输入的字符替换。

事件处理器

onblur

当用户把键盘焦点从元素中移开时调用的事件句柄。

onchange

对于文本输入的元素，当用户改变显示的文本，然后通过切换或单击来把键盘焦点转移到另一个元素从而“确认”这些改变的时候，这个事件句柄会调用。这个句柄不会报告连续按键的编辑。对于类型为“checkbox”和“radio”的开关按钮元素，当用户切换它们的时候，可能也会触发此事件（还要加上 onclick 事件）。

onclick

对于那些按钮表单元素和开关按钮来说，当用户通过鼠标或键盘来激活该按钮时，就调用此事件句柄。

onfocus

当用户把键盘焦点给予该元素时调用的事件句柄。

描述

一个Input对象代表了一个定义了可脚本化的表单输入元素的HTML `<input>` 标记。Input对象的3个最重要的属性是 `type`、`value` 和 `name`。这些属性在接下来的部分介绍。参见第18章了解有关HTML表单和表单元素的更多信息。

输入元素类型

HTML `<input>` 标记的 `type` 属性指定了所要创建的输入元素的类型。对于客户端JavaScript来说，这一属性作为Input对象的 `type` 属性可用，而且在确定一种未知的表单元素的类型的时候很有用，例如，当遍历一个Form对象的 `elements[]` 数组的时候。

合法的 `type` 值如下：

“button”

输入元素是一个图形化的按钮，显示了 `value` 属性指定的纯文本。这个按钮没有默认的行为，但是必须有一个 `onclick` 事件句柄以便使用。对于提交重置一个表单的按钮，`type` 使用 “submit” 和 “reset”。注意，HTML `<button>` 标记可以创建一个显示任意HTML而不是纯文本的按钮。

“checkbox”

这种类型的输入元素显示一个开关按钮，用户可以选中或不选中它。`checked` 属性保存了这个按钮的当前状态，不管何时，这个值发生变化的时候，`onclick` 事件句柄就会触发（浏览器也可能会触发 `onchange` 句柄）。`value` 属性是提交给Web服务器的一个内部值，不会显示给用户。要把一个标签和一个复选按钮联系起来，只要把标签文本放置到靠近 `<input>` 标记的地方即可，通常使用一个 `<label>` 标记。复选按钮元素常常成组出现，一组的每个成员有时候都具有相同的 `name` 属性和不同的 `value` 属性，这样就给表单要提交的Web服务器带来了方便。

“file”

这个类型创建一个 “file upload” 元素。这个元素包含了一个文本输入字段，用来输入文件名，还有一个按钮，用来打开文件选择对话框以便图形化选择文件。`value` 属性保存了用户指定的文件的名称，但是当包含一个file-upload元素的表单被提交的时候，浏览器会向服务器发送选中的文件的内容而不仅仅是发送文件名（要做到这一点，表单必须使用 “multipart/form-data” 编码并且使用POST方法）。

为安全起见，file-upload元素不允许HTML作者或JavaScript程序员指定一个默认的文件名。HTML `value` 属性被忽略，并且对于此类元素来说，`value` 属性是只读的，这意味着只有用户可以输入一个文件名。当用户选择或编辑一个文件名，file-upload元素触发 `onchange` 事件句柄。

“hidden”

这种类型的输入元素实际上是隐藏的。这个不可见的表单元素的 `value` 属性保存了一个要提交给 Web 服务器的任意字符串。如果想要提交并非用户直接输入的数据的话，就使用这种类型的元素。

“image”

这种类型的输入元素是显示一幅图像（由 `src` 属性指定）的一个表单提交按钮，而不是显示一个文本标签。`value` 属性并不使用。参见“submit”元素类型了解更多细节。

“password”

这个文本输入字段用来输入敏感数据，如密码等。当用户输入的时候，他的输入是被掩盖的（例如，使用星号*），以防止旁边的人从他背后看到输入内容。注意，用户的输入仍然是没有以任何方式加密的，当表单提交的时候，输入用明文发送。出于安全考虑，一些浏览器可能阻止 JavaScript 代码读取 `value` 属性。在其他方面，一个密码输入元素就像一个类型为“text”的元素一样。当用户改变显示的值的时候，它触发 `onchange` 事件句柄。

“radio”

这种类型的输入元素显示一个单独的图形化单选按钮。单选按钮是表示一组互斥选项按钮中的一个。当一个按钮被选中，之前选中的按钮就变为非选中的（类似早期汽车收音机的机械式频道预调按钮）。为了使一组单选按钮能够展示它们的互斥行为，它们必须出现在同一个 `<form>` 中，而且必须拥有相同的名称。由于开关按钮没有互斥行为，它使用“checkbox”类型。注意，HTML `<select>` 标记也可以用来表示互斥的或非互斥的选择（参阅 Select）。

`checked` 属性表示单选按钮是否被选中。没有办法来确定互斥的一组单选按钮中哪个按钮被选中，必须查看每个按钮的 `checked` 属性。当选中或不被选中的时候，单选按钮触发 `onclick` 事件句柄。

`value` 属性指定要提交给 Web 服务器的值，并且这个值不会在表单中显示。要为单选按钮指定一个标签，在 `<input>` 标记外部来实现，例如使用一个 `<label>` 标记。

“reset”

这种类型的输入元素就像一个用“button”类型创建的按钮，但是它有更为专门的用途。当一个重置按钮元素被单击，包含它的表单中的所有输入元素的值都重置为它们的默认值（默认值由 HTML `value` 属性或 JavaScript 的 `defaultValue` 属性指定）。`value` 属性指定了要显示在按钮中的文本。重置按钮在重置表单之前触发 `onclick` 句柄，并且这个句柄可以通过返回 `false` 来取消重置，或者使用第 17 章中所描述的其他事件取消方法也可以取消重置。参阅 `Form.reset()` 方法和 `Form.onreset` 事件句柄。

“submit”

这种类型的元素是一个按钮并且在单击时提交包含 `<form>`。`value` 属性指定了出现在按钮中的文本。在表单提交前，触发 `onclick` 事件句柄，并且一个句柄可以通过返回 `false` 来取消表单提交。参阅 `Form.submit()` 方法和 `Form.onsubmit` 事件句柄。

“text”

这是 `type` 属性的默认值，它创建一个单行的文本输入字段。HTML `value` 属性指定了出现在字段中的默认文本，而 JavaScript 的 `value` 属性保存了当前显示的文本。当用户编辑显示的文本并随后把输入焦点转移到其他元素的时候，触发了 `onchange` 事件句柄。使用 `size` 来指定输入字段的宽度，使用 `maxLength` 来指定允许输入的最大字符数。当一个表单只包含一个类型为 “text” 的输入元素，按下 **Enter** 键提交这个表单。

对于多行文本输入，使用 HTML `<textarea>` 标记（参阅 `Textarea`）。对于掩码文本输入，把 `type` 设置为 “password”。

输入元素值

`Input` 对象的 `value` 属性是可读写的字符串属性，它指定了表单所包含的输入元素被提交的时候要发送给 Web 服务器的文本。

根据 `type` 属性的值，`value` 属性也可能包含用户可见的文本。对于 `type` 为 “text” 和 “file” 的输入元素，这个属性保存了用户输入的文本。对于 `type` 为 “button”、“reset” 和 “submit” 的元素，这个属性指定了显示在按钮中的文本。对于其他元素类型，例如 “checkbox”、“radio” 和 “image”，`value` 属性的内容不会显示给用户，只有在表单提交的时候才使用。

输入元素名称

`Input` 对象的 `name` 属性是一个字符串，它提供了输入元素的名称。它的值来自 HTML `name` 属性。表单元素的名称有两个用途。第一，在提交表单时可以使用它。表单中所有元素的数据通常都是以下面这种形式提交的：

```
name=value
```

这里对 `name` 和 `value` 都进行了编码，这是传输必需的。如果没有给表单元素指定名称，那么元素的数据就不能提交给 Web 服务器。

`name` 属性的第二个用途是在 JavaScript 代码中引用表单元素。元素的名称将成为含有该元素的表单的一个属性。该属性的值就是对元素的引用。例如，假设有一个表单为 `address`，它含有一个名为 `zip` 的文本输入元素，那么 `address.zip` 引用的就是那个文本输入元素。

对于 `type` 为 “radio” 和 “checkbox” 输入元素来说，定义多个相关的对象，每个对象具有相同的 `name` 属性是很常见的。在这种情况下，数据传递给 Web 服务器时采用如下格式：

```
name=value1,value2,...,valuen
```

同样，在 JavaScript 中，共享同一个名称的元素都会成为具有那个名称的数组中的元素。因此，如果表单 `order` 中的四个 `Checkbox` 对象共享名称 `options`，就可以将它们作为 `order.options[]` 数组的元素来访问。

相关表单元素

HTML `<input>` 允许你创建多个不同的表单元素。但 `<button>`、`<select>` 和 `<textarea>` 也创建表单元素。

参阅

`Form`、`Form.elements[]`、`Option`、`Select`、`Textarea`，第 18 章

`Input.blur()`

2 级 DOM HTML

将键盘焦点从表单元素中移开

摘要

```
void blur()
```

描述

表单元素的方法 `blur()` 不需要调用事件句柄 `onblur` 就可以把键盘焦点从元素中移开。它实质上与方法 `focus()` 作用相反。但是方法 `blur()` 不会把键盘焦点转移到别的地方，所以当你不想触发事件句柄 `onblur` 时，真正有用的调用时机是你想要用 `focus()` 方法将键盘焦点转移到别处之前。也就是说，应该明确地把键盘焦点从元素中移开，这样当别的元素调用 `focus()` 方法隐式地移除键盘焦点时就不会再通知你了。

`Input.click()`

2 级 DOM HTML

在表单元素上模拟鼠标单击事件

摘要

```
void click()
```

描述

表单元素的 `click()` 方法模拟了表单元素上的一次鼠标单击事件，但是并不调用该元素的事件句柄 `onclick`。

该方法并不常用。由于它不调用事件句柄 `onclick`，所以对于类型为“button”的元素来说，它没有太大的用途，因为除了 `onclick` 处理程序定义的行为之外，再没有其他的行为了。在类型为“submit”或“reset”的元素上调用 `click()` 方法可以提交或重置表单，但是调用 `Form` 对象自身的 `submit()` 方法和 `reset()` 方法来实现这一点更直接一些。

`Input.focus()`

2 级 DOM HTML

把键盘焦点赋予表单元素

摘要

```
void focus()
```

描述

表单元素的方法 `focus()` 无须调用事件句柄 `onfocus` 就可以把键盘焦点转移到输入元素。也就是说，就键盘导航和键盘输入而言，它可以激活元素。因此，如果在类型为 “text” 的元素上调用 `focus()` 方法，用户输入的所有文本都会显示在那个元素中。如果在类型为 “button” 的元素上调用 `focus()` 方法，那么用户就可以从键盘来调用那个按钮。

Input.onblur

0 级 DOM

当表单元素失去焦点时调用的事件句柄

摘要

Function onblur

描述

Input 对象的属性 `onblur` 声明了一个事件句柄函数，这个函数是在用户把键盘焦点从该输入元素中移开时调用的。虽然调用 `blur()` 方法也可以将焦点从元素中移开，但是它并不能调用那个对象的 `onblur` 句柄。但是要注意，调用 `focus()` 把焦点转移到其他元素可以引发当前具有焦点的元素的 `onblur` 事件句柄函数被调用。

参阅

`Element.addEventListener()`、`Window.onblur`，第 17 章

Input.onchange

2 级 DOM Events

当改变表单元素的值时调用的事件句柄

摘要

Function onchange

描述

Input 对象的属性 `onchange` 声明了一个事件句柄函数，当用户改变了表单元素显示的值时就会调用这个句柄。这种改变既可以是对类型为 “text”、“password” 和 “file” 的输入元素中显示的文本的编辑，也可以是选择或取消选择类型为 “radio” 或 “checkbox” 的开关按钮（单选按钮元素和复选按钮元素总是触发 `onclick` 句柄，并且也能触发 `onchange` 句柄）。注意，只有当用户做出上述改变时才会调用这个事件句柄函数，当 JavaScript 程序改变元素显示的值时并不会调用它。

还要注意，并非每次用户在文本表单元素中输入一个字符或删除一个字符都会调用 `onchange` 句柄。`onchange` 不是为逐个字符的事件处理类型设计的。相反，只有在用户的编辑全部完成后才会调用它。浏览器假定在键盘焦点从当前元素移动到其他元素（如用

户单击了表单中另一个元素) 时用户的编辑就完成了。要了解逐个字符的事件处理类型请参阅 `HTMLElement.onkeypress`。

对于类型为 “button”、“hidden”、“image”、“reset” 和 “submit” 的输入元素, `onchange` 事件句柄不能使用。这些类型的元素使用 `onclick` 句柄。

参阅

`Element.addEventListener()`、`HTMLElement.onkeypress`, 第 17 章

Input.onclick

2 级 DOM Events

单击表单元素时调用的事件句柄

摘要

Function onclick

描述

`Input` 对象的属性 `onclick` 声明了一个事件句柄函数, 当用户激活该输入元素时就会调用这个句柄。这通常在使用鼠标单击元素的时候完成, 但是, 当用户使用键盘遍历来激活该元素的时候, `onclick` 句柄也会触发。但是调用该元素的 `click()` 方法时并不会调用 `onclick` 句柄。

注意, 类型为 “reset” 和 “submit” 的元素在单击的时候执行一个默认的动作, 它们分别重置或提交包含它们的表单。可以使用 `onclick` 事件句柄让每个元素来执行这些默认动作之外的动作。可以通过返回 `false` 来阻止这些默认动作, 或者使用第 17 章介绍的其他事件取消技术来阻止它们。注意, 你可以用 `Form` 对象自己的 `onsubmit` 和 `onreset` 事件句柄来完成类似的事情。

参阅

`Element.addEventListener()`, 第 17 章

Input.onfocus

2 级 DOM Events

当表单元素获得焦点时调用的事件句柄

摘要

Function onfocus

描述

`Input` 对象的属性 `onfocus` 声明了一个事件句柄函数, 当用户把键盘焦点转移到该元素时就会调用这个句柄函数。但是当调用了该元素的 `focus()` 方法设置焦点时并不会调用它的 `onfocus` 句柄。

参阅

`Element.addEventListener()`、`Window.onfocus`，第 17 章

`Input.select()`

2 级 DOM HTML

选择表单元素中的文本

摘要

```
void select()
```

描述

方法 `select()` 可以选择在类型为 “text”、“password” 或 “file” 的元素中显示的文本。选择文本后产生的效果，则根据平台的不同有所区别，不过一般说来，文本会高亮显示，变成可以剪切和粘贴的，并且如果用户输入其他字符的话文本会被删除掉。

`JavaArray`, `JavaClass`, `JavaObject`, `JavaPackage`

参阅本书第三部分

`JSObject`

Java 插件中的 Java 类

JavaScript 对象的 Java 表示

摘要

```
public final class netscape.javascript.JSObject extends Object
```

方法

```
call()
```

调用 JavaScript 对象的方法。

```
eval()
```

在 JavaScript 对象环境中执行一个 JavaScript 代码串。

```
getMember()
```

获取 JavaScript 对象的一个属性值。

```
getSlot()
```

获取 JavaScript 对象的一个数组元素的值。

```
getWindow()
```

获取一个 “根” `JSObject` 对象，表示 JavaScript 中代表浏览器窗口的 `Window` 对象。

```
removeMember()
```

删除 JavaScript 对象的一个属性。

```
setMember()
```

设置 JavaScript 对象的一个属性的值。

`setSlot()`

设置 JavaScript 对象的一个数组元素的值。

`toString()`

调用 JavaScript 对象的 `toString()` 方法，并且返回该方法执行的结果。

描述

JSObject 是一个 Java 类，而不是一个 JavaScript 对象，在 JavaScript 程序中不能使用它。相反，JSObject 对象由 Java 小程序使用，这些小程序通过读写 JavaScript 属性和数组元素、调用 JavaScript 的方法以及执行 JavaScript 的代码串与 JavaScript 进行通信。显而易见，由于 JSObject 是一个 Java 类，所以要使用它，还要了解 Java 的程序设计方法。

要了解有关 JSObject 程序设计技术的详细内容，请参阅第 23 章。

参阅

第 23 章，第 12 章；第三部分中的 JavaObject

JSObject.call()

Java 插件中的 Java 方法

调用 JavaScript 对象的一个方法

摘要

```
public Object call(String methodName, Object args[])
```

参数

methodName

要调用的 JavaScript 方法的名称。

args[]

作为参数传递给该方法的 Java 对象数组。

返回值

一个 Java 对象，表示 JavaScript 方法的返回值。

描述

Java JSObject 类的方法 `call()` 将调用一个已命名的 JavaScript 方法，这个方法以 JSObject 表示。参数将以 Java 对象数组的形式传递给该方法。JavaScript 方法的返回值是一个 Java 对象。

有关把该方法的参数从 Java 对象转换成 JavaScript 值，以及把 JavaScript 方法的返回值从 JavaScript 值转换成 Java 对象执行的数据转换，请参见第 23 章中的详细介绍。

JSObject.eval()Java 插件中的 Java 方法

执行一个 JavaScript 代码串

摘要

```
public Object eval(String s)
```

参数

s 一个字符串，含有任意的 JavaScript 语句，各语句之间用分号隔开。

返回值

s 中最后一个表达式的 JavaScript 值，将被转换成一个 Java 对象。

描述

Java JSObject 类的 `eval()` 方法将在 JSObject 指定的 JavaScript 对象环境中执行字符串 *s* 中的 JavaScript 代码。它的行为与 JavaScript 的全局函数 `eval()` 相似。

参数 *s* 是一个含有任意多条 JavaScript 语句的字符串，其中的语句用分号隔开，执行顺序与它们出现的顺序相同。`eval()` 返回的是计算 *s* 中的最后一条语句或表达式得到的值。

JSObject.getMember()Java 插件中的 Java 方法

读 JavaScript 对象的一个属性

摘要

```
public Object getMember(String name)
```

参数

name

要读的属性的名称。

返回值

一个 Java 对象，含有指定的 JSObject 对象命名的属性的值。

描述

Java JSObject 类的 `getMember()` 方法将读取 JavaScript 对象的一个已命名属性的值，并且将它返回给 Java。返回的值是另一个 JSObject 对象或一个 Double 对象、Boolean 对象或者 String 对象，不过返回时这个值都是通用的 Object 对象，必须对它进行必要的类型转换。

JSObject.getSlot()Java 插件中的 Java 方法

读一个 JavaScript 对象的数组元素

摘要

```
public Object getSlot(int index)
```

参数

index

要读取的数组元素的下标。

返回值

位于指定的 *index* 的 JavaScript 对象的数组元素值。

描述

Java JSObject 类的 `getSlot()` 方法可以读取位于 JavaScript 对象的指定 *index* 处的数组元素值，并且把这个值返回给 Java。返回值是另一个 JSObject 对象或一个 Double 对象、Boolean 对象或者 String 对象，不过返回时这个值都是通用的 Object 对象，必须对它进行必要的类型转换。

JSObject.getWindow()

Java 插件中的 Java 方法

返回代表浏览器窗口的初始 JSObject 对象

摘要

```
public static JSObject getWindow(java.applet.Applet applet)
```

参数

applet

一个 Applet 对象，在这个窗口中运行的对象就是要获取 JSObject 对象。

返回值

一个 JSObject 对象，表示 JavaScript 中用于代表浏览器窗口的 Window 对象，这个窗口含有指定的小程序 *applet*。

描述

`getWindow()` 是所有的 Java 小程序都要调用的第一个 JSObject 方法。由于 JSObject 类没有定义构造函数，所以调用静态方法 `getWindow()` 是惟一一种获得初始“根” JSObject 对象的方法。

JSObject.removeMember()

Java 插件中的 Java 方法

删除 JavaScript 对象的一个属性

摘要

```
public void removeMember(String name)
```

参数

name

要从 JSObject 对象中删除的属性的名称。

描述

Java JSObject 类的方法 `removeMember()` 可以从 JSObject 对象表示的 JavaScript 对象中删除一个已命名的属性。

JSObject.setMember()

Java 插件中的 Java 方法

设置 JavaScript 对象的一个属性

摘要

```
public void setMember(String name, Object value)
```

参数

name

JSObject 对象中要设置的属性的名称。

value

给指定的属性设置的值。

描述

Java JSObject 类的方法 `setMember()` 可以从 Java 中设置 JavaScript 对象的一个已命名属性的值。指定的值 *value* 可以是任何类型的 Java 对象。不过原始的 Java 值不会传递给这个方法。在 JavaScript 中，指定的值 *value* 通过一个 `JavaObject` 对象访问。

JSObject.setSlot()

Java 插件中的 Java 方法

设置 JavaScript 对象的一个数组元素

摘要

```
public void setSlot(int index, Object value)
```

参数

index

要在 JSObject 对象中设置的数组元素的下标。

value

要给指定的数组元素设置的值。

描述

Java JSObject 类的方法 `setSlot()` 可以从 Java 中设置 JavaScript 对象的已编码数组的元

素值。指定的值 `value` 可以是任何类型的 Java 对象。不过原始的 Java 值不会传递给这个方法。在 JavaScript 中，指定的值 `value` 作为一个 `JavaScript` 对象访问。

`JSObject.toString()`

Java 插件中的 Java 方法

返回一个 JavaScript 对象的字符串值。

摘要

```
public String toString()
```

返回值

一个字符串，是调用 Java `JSObject` 对象所表示的 JavaScript 对象的 `toString()` 方法而返回的。

描述

Java `JSObject` 类的 `toString()` 方法将调用 `JSObject` 表示的 JavaScript 对象的 `toString()` 方法，并且返回调用后的结果。

`KeyEvent`

Firefox 及兼容的浏览器

和键盘事件相关的细节

Event → UIEvent → `KeyEvent`

属性

readonly boolean `altKey`

当事件发生时，**Alt** 按键是否被按下。

readonly integer `charCode`

这个数字是一个 `keypress` 事件（如果有的话）所产生的可打印字符的 Unicode 编码。对于不可打印的功能键，这个属性为 0，并且这个属性不能用于 `keydown` 和 `keyup` 事件。使用 `String.fromCharCode()` 把这个属性转换为一个字符串。

readonly boolean `ctrlKey`

当事件发生时，**Ctrl** 按键是否被按下。为所有鼠标事件类型而定义。

readonly integer `keyCode`

被按下的按键的虚拟按键码，这个属性用于所有键盘事件类型。按键码可能是浏览器相关的、操作系统相关的以及使键盘硬件相关的。通常，当按键在其上显示一个可打印的字符的时候，这个按键的虚拟按键码和字符的编码是相同的。不可打印的功能键的按键码变化很多，参见例 17-6 可以找到一组常用的编码。

readonly boolean `shiftKey`

当事件发生时，**Shift** 按键是否被按下。为所有鼠标事件类型而定义。

描述

KeyEvent对象提供关于一个键盘事件的细节,并且可以传递给keydown、keypress和keyup事件的事件句柄。2级DOM Events标准并不涉及键盘事件,而KeyEvent对象也没有标准化。这个条目描述的是Firefox的实现。这些属性中的很多也只是在IE事件模型中支持,参阅Event对象的特定于IE的属性描述。注意,除了这里列出的属性,KeyEvent还继承了Event和UIEvent的属性。

第17章包括几个关于使用KeyEvent对象的具体例子。

参阅

Event、UIEvent, 第17章

Layer

仅在 Netscape 4 中, 在 Netscape 6 中不再保留

一个废弃的 Netscape API

描述

Layer对象是支持动态定位的HTML元素的Netscape 4技术。它并没有标准化,而且现在废弃了。

参阅

第16章

Link

0级DOM

HTML文档中的一个超文本链接或锚

Node → Element → HTMLElement → Link

属性

Link对象最重要的属性就是href,这是它的链接的一个URL。Link对象还定义了很多其他属性来存储URL的一部分。在接下来的每条属性说明中,给出的例子都是下面这个(虚拟)URL的一部分:

`http://www.oreilly.com:1234/catalog/search.html?q=JavaScript&m=10#results`

String hash

指定了Link对象的URL中的锚部分,包括前导散列符(#)。例如,“#results”。URL的锚部分只链接在文档中引用的位置。在HTML文件中,这个位置由<a>标记的name属性指定(参阅Anchor)。

String host

指定了Link的URL中的主机名和端口部分。例如, `www.oreilly.com:1234`。

String hostname

指定了Link的URL中的主机名部分。例如, “`www.oreilly.com`”。

String href

指定了Link的URL的完整文本,不同于其他的Link URL属性,声明的只是部分URL。

String pathname

声明了Link的URL的路径部分。例如,“/catalog/search.html”。

String port

声明了Link的URL的端口部分。例如,“1234”。

String protocol

声明了Link的URL的协议部分,包括后缀冒号。例如,“http:”。

String search

声明了Link的URL的查询部分,其中包括前导问号。例如,“?q=JavaScript&m=10”。

除了这些和URL相关的属性,Link对象还定义了和HTML的<a>和<area>标记的属性对应的属性:

属性	HTML 的属性	描述
String accessKey	accesskey	键盘快捷方式
String charset	charset	目标文档的编码
String coords	coords	用于<area>标记
String hreflang	hreflang	链接的文档的语言
String name	name	锚名称,参阅Anchor
String rel	rel	链接类型
String rev	rev	反转链接类型
String shape	shape	用于<area>标记
long tabIndex	tabindex	链接在切换顺序中的位置
String target	target	目标文档在其中显示的帧或窗口的名称
String type	type	目标文档的内容类型

方法

blur()

从链接上移走键盘焦点

focus()

滚动文档以使链接可见,并且把键盘焦点给链接。

事件句柄

Link对象针对3个事件句柄有特定的行为:

onclick

当用户单击链接时调用的事件句柄。

onmouseout

当用户把鼠标移出链接时调用的事件句柄。

onmouseover

当用户把鼠标移动到链接上时调用的事件句柄。

HTML 语法

Link 对象是由标准的 HTML 标记 `<a>` 和 `` 创建的。href 性质对所有的 Link 对象都是必需的。如果还设置了该对象的 name 性质，那么还会创建一个 Anchor 对象：

```
<a href="url"                // The destination of the link
  [ name="anchor_tag" ]      // Creates an Anchor object
  [ target="window_name" ]   // Where the new document should be displayed
  [ onclick="handler" ],     // Invoked when link is clicked
  [ onmouseover="handler" ]  // Invoked when mouse is over link
  [ onmouseout="handler" ]   // Invoked when mouse leaves link
> link text or image        // The visible part of the link
</a>
```

描述

Link 对象表示文档中的一个超链接。Link 通常是由 `<a>` 标记创建的，这个标记定义了一个 href 属性，但是，它们也可以使用客户端图像映射中的 `<area>` 标记来创建。当一个 `<a>` 标记有一个 name 属性替代 href 属性，它就在文档中定义了一个指定的位置，并且用一个 Anchor 对象来表示，而不是 Link 对象。参阅 Anchor 了解更多细节。

一个文档中的所有链接（由 `<a>` 标记或 `<area>` 标记创建）都是通过 Document 对象的 links[] 数组中的 Link 对象表示的。

超文本链接的目的文件由 URL 指定，Link 对象的许多属性都用来设置 URL 的内容。在这一点上，Link 对象与 Location 对象相似。在 Location 对象中，这些属性描述的是当前显示的文档的 URL。

例子

```
// Get the URL of the first hyperlink in the document
var url = document.links[0].href;
```

参阅

Anchor、Location

Link.blur()

0 级 DOM

从超链接移走键盘焦点

摘要

```
void blur();
```

描述

对于允许超链接拥有键盘焦点的 Web 浏览器来说，这个方法从超链接移走键盘焦点。

Link.focus()

0 级 DOM

使一个链接可见并给它键盘焦点

摘要

```
void focus();
```

描述

这个方法滚动文档以使指定的超链接变得可见。如果浏览器允许链接拥有键盘焦点，这个方法就把焦点给超链接。

Link.onclick

0 级 DOM

单击链接时调用的事件句柄

摘要

```
Function onclick
```

描述

Link 对象的 onclick 属性声明了一个事件句柄函数，当用户单击了该链接时，它就会被调用。在事件句柄返回后的浏览器默认动作是遵从被单击的超链接。可以通过返回 false 或者使用第 17 章所介绍的其他事件取消方法来阻止这一默认动作。

参阅

Element.addEventListener()、MouseEvent，第 17 章

Link.onmouseout

0 级 DOM

当鼠标离开该链接时调用的事件句柄

摘要

```
Function Onmouseout
```

描述

Link 对象的 onmouseout 属性声明了一个事件处理器函数，当用户把鼠标从该链接上移开时，它就会被调用。它常常和 onmouseover 事件句柄一起使用。

参阅

Element.addEventListener()、Link.onmouseover、MouseEvent，第 17 章

Link.onmouseover

0 级 DOM

当鼠标经过该链接时调用的事件句柄**摘要**

Function onmouseover

描述

Link对象的onmouseover属性声明了一个事件处理器函数,当用户把鼠标移动到该链接上时,它就会被调用。当用户将鼠标停留在超链接上时,浏览器在状态栏为这个链接显示URL。在旧的浏览器中,也可以阻止这一默认动作并在状态栏显示你自己的文本。由于安全原因(例如,防止钓鱼攻击),大多数现代浏览器都关闭了这一功能。

参阅

Element.addEventListener()、Link.onmouseout、MouseEvent, 第 17 章

Location

JavaScript 1.0

表示并控制浏览器的位置

Object → Location

摘要Location
window.location**属性**

Location对象的属性引用了当前文档的URL的各个部分。在接下来的每条属性说明中,给出的例子都是下面这条(虚拟)URL的一部分:

```
http://www.oreilly.com:1234/catalog/search.html?q=JavaScript&m=10#results
```

hash

一个可读可写的字符串,指定了当前URL中的锚部分,包括前导散列符(#)。例如,“#results”。文档URL的这一部分指定了锚在文档中的名称。

host

一个可读可写的字符串,是当前URL中的主机名和端口部分。例如,“www.oreilly.com:1234”。

hostname

一个可读可写的字符串,声明了当前URL中的主机名部分。例如,“www.oreilly.com”。

href

一个可读可写的字符串,它声明了当前显示的文档的完整URL,与其他Location属性只声明部分URL不同。把该属性设置为新的URL会使浏览器读取并显示新URL的内容。

pathname

一个可读可写的字符串，声明了当前URL的路径部分。例如，“/catalog/search.html”。

port

一个可读可写的字符串（不是数字），声明了当前URL的端口部分。例如，“1234”。

protocol

一个可读可写的字符串，声明了URL的协议部分，包括后缀的冒号。例如，“http:”。

search

一个可读可写的字符串，声明了当前URL的查询部分，包括前导问号。例如，“?q=JavaScript&m=10”。

方法

reload()

从缓存或服务器中再次把当前文档装载进来。

replace()

用一个新文档替换当前文档，而不用在浏览器的会话历史中生成一个新的记录。

描述

Location对象存储在Window对象的location属性中，表示那个窗口中当前显示的文档的Web地址。它的href属性存放的是文档的完整URL，其他属性则分别描述了URL的各个部分。这些属性与Link对象的URL属性非常相似。当一个Location对象被转换为字符串，href属性的值被返回。这意味着你可以使用表达式location来替代location.href。

不过Link对象表示的是文档中的超链接，Location对象表示的却是浏览器当前显示的文档的URL（或位置）。但是Location对象所能做的远远不止这些，它还能控制浏览器显示的文档的位置。如果把一个含有URL的字符串赋予Location对象或它的href属性，浏览器就会把新的URL所指的文档装载进来，并显示出来。

除了设置location或location.href用完整的URL替换当前的URL之外，还可以修改部分URL，只需要给Location对象的其他属性赋值即可。这样做会创建一个新的URL，其中的一部分与原来的URL不同，浏览器会将它装载并显示出来。例如，假设设置了Location对象的hash属性，那么浏览器就会转移到当前文档中的一个指定位置。同样，如果设置了search属性，那么浏览器就会重新装载附加了新的查询字符串的URL。

除了URL属性外，Location对象还定义了两个方法。reload()可以重新装载当前文档，replace()可以装载一个新文档而无须为它创建一个新的历史记录，也就是说，在浏览器的历史列表中，新文档将替换当前文档。

参阅

Link、HTMLDocument 对象的URL属性

Location.reload()JavaScript 1.1

重新装载当前文档

摘要

```
location.reload()  
location.reload(force)
```

参数*force*

一个可选的布尔参数，声明了是否应该重新装载当前文档，即使服务器的报告表明自从上次装载完毕之后它还没有被修改过也是如此。如果省略了这个参数，或者它的值为 `false`，那么只有自上次装载之后已改变文档，该方法才会重新装载整个页面。

描述

`Location` 对象的方法 `reload()` 可以把它所在的窗口正在显示的文档重新装载一遍。如果调用时没有给它传递参数，或者传递的参数是 `false`，它就会用 HTTP 头 `If-Modified-Since` 来检测服务器上的文档是否改变了。如果文档已经改变了，`reload` 就会把它从服务器上再次下载下来。如果文档没有改变，该方法将从缓存中把它装载进来。这与用户单击了浏览器的 **Reload** 按钮时发生的动作完全相同。

如果调用方法 `reload()` 时传递给它的参数是 `true`，那么无论文档的最后修改日期是什么，它都将绕过缓存，从服务器上重新下载该文档。这与用户在单击浏览器的 **Reload** 按钮时按住 **Shift** 键发生的动作完全相同。

Location.replace()JavaScript 1.1

用另一个文档替换当前显示的文档

摘要

```
location.replace(url)
```

参数*url*

一个字符串，指定了要替换当前文档的新文档的 URL。

描述

`Location` 对象的方法 `replace()` 将下载并显示一个新文档。用这种方式装载文档与只设置属性 `location` 或 `location.href` 有一点重要的不同之处，即 `replace()` 方法不在 `History` 对象中生成一个新的记录。当使用 `replace()` 方法时，新的 URL 就会覆盖 `History` 对象中的当前记录。也就是说，在调用 `replace()` 方法之后，单击浏览器的 **Back** 按钮就不会返回到之前浏览的那个文档，而直接返回到那个文档之前的 URL。

参阅

History

MimeType

JavaScript 1.1; IE 不支持

代表 MIME 数据类型

Object → MimeType

摘要

```
navigator.mimeTypes[i]  
navigator.mimeTypes["type"]  
navigator.mimeTypes.length
```

属性

description

一个只读的字符串，提供了由 MimeType 对象描述的数据类型的英文描述。

enabledPlugin

对已安装的 Plugin 对象的引用，并启用了处理指定类型的插件。如果没有插件处理这种 MIME 类型（例如，如果它直接由浏览器处理），那么这个属性的值就是 null。当插件已经存在但是被关闭时，这个属性也为 null。

suffixes

一个只读字符串，它存放的是一个文件名后缀（不包括字符“.”）的列表，各后缀之间用逗号分开。其中的文件名后缀常用于指定的 MIME 类型文件。例如，MIME 类型 text/html 的后缀是“html, htm”。

type

一个只读的字符串，声明了 MIME 类型的名称。这种字符串是惟一的，如“text/html”或“image/jpeg”，它将各种 MIME 类型区别开了。此外，它还描述了数据的一般类型以及使用的数据格式。这个属性的值还可以用作访问数组 navigator.mimeTypes[] 元素的下标。

描述

MimeType 对象表示 Netscape 支持的 MIME 类型（即数据格式）。这种格式可以是浏览器直接支持的，也可以是外部的辅助程序或插件支持的。MimeType 对象是 Navigator 对象的 mimeTypes[] 数组的成员。在 IE 中，mimeTypes[] 数组总是空的，并且没有这一功能的替代。

习惯用法

引用数组 navigator.mimeTypes[] 中的元素时，既可以使用该元素的下标，又可以使用 MIME 类型的名称（即属性 type 的值）。要检测 Netscape 支持的 MIME 类型，可以遍历

数组 `navigator.mimeTypes[]`。如果想检测 Netscape 是否支持某种指定的类型，可以使用如下的代码：

```
var show_movie = (navigator.mimeTypes["video/mpeg"] != null);
```

参阅

Navigator、Plugin

MouseEvent

2 级 DOM Events

鼠标事件的详细情况

Event → UIEvent → MouseEvent

属性

readonly boolean altKey

在鼠标事件发生时，**Alt** 键是否被按下并保持住了。所有类型的鼠标事件都定义了该属性。

readonly unsigned short button

在 `mousedown`、`mouseup` 或 `click` 事件中，哪个鼠标键的状态改变了。0 表示左键，2 表示右键，1 表示鼠标的中间键。注意，只有当改变鼠标键的状态时，该属性才会被定义，例如，在 `mousemove` 事件中，它不能用于报告鼠标键是否被按下并保持住了。此外，该属性不是位图，不能报告是否多个鼠标键被按下并保持住了。

readonly long clientX, clientY

声明鼠标指针相对于“客户区”或浏览器窗口的 X 坐标和 Y 坐标。注意，这些坐标不考虑文档滚动，如果事件发生在窗口的最上边，`clientY` 的值就是 0，无论文档向下滚动了多远都是如此。所有类型的鼠标事件都定义了这两个属性。

readonly boolean ctrlKey

在鼠标事件发生时，**Ctrl** 键是否被按下并保持住了。所有类型的鼠标事件都定义了该属性。

readonly boolean metaKey

在鼠标事件发生时，**Meta** 键是否被按下并保持住了。所有类型的鼠标事件都定义了该属性。

readonly Element relatedTarget

引用与事件的 `target` 节点相关的节点。对于 `mouseover` 事件，它是鼠标移到目标节点前要离开的 `Element`。对于 `mouseout` 事件，它是鼠标离开目标节点后要进入的 `Element`。其他类型的鼠标事件没有定义 `relatedTarget` 属性。

readonly long screenX, screenY

鼠标指针相对于用户显示器左上角的 X 坐标和 Y 坐标。所有类型的鼠标事件都定义了这两个属性。

readonly boolean shiftKey

在鼠标事件发生时，**Shift** 键是否被按下并保持住了。所有类型的鼠标事件都定义了该属性。

方法

initMouseEvent()

初始化新创建的 `MouseEvent` 对象的属性。

描述

该接口定义了传递给 `click`、`mousedown`、`mousemove`、`mouseout`、`mouseover` 和 `mouseup` 事件的 `Event` 对象的类型。注意，除了这里列出的属性之外，该接口还继承了 `UIEvent` 和 `Event` 接口的属性。

参阅

`Event`、`UIEvent`，第 17 章

MouseEvent.initMouseEvent()

2 级 DOM Events

初始化 `MouseEvent` 对象的属性

摘要

```
void initMouseEvent(String typeArg,  
                    boolean canBubbleArg,  
                    boolean cancelableArg,  
                    AbstractView viewArg,  
                    long detailArg,  
                    long screenXArg,  
                    long screenYArg,  
                    long clientXArg,  
                    long clientYArg,  
                    boolean ctrlKeyArg,  
                    boolean altKeyArg,  
                    boolean shiftKeyArg,  
                    boolean metaKeyArg,  
                    unsigned short buttonArg,  
                    Element relatedTargetArg);
```

参数

该方法的许多参数用于设置 `MouseEvent` 对象的属性的初始值，其中包括从 `Event` 和 `UIEvent` 接口继承的属性。因为每个参数的名称清楚地说明了它们要设置的值的属性，所以这里不再一一列出。

描述

该方法将初始化新创建的 `MouseEvent` 对象的各种属性。只有 `Document.createEvent()`

方法创建的 `MouseEvent` 对象可以调用该方法，而且必须在把 `MouseEvent` 对象传递给 `Element.dispatchEvent()` 方法之前调用才行。

参阅

`Document.createEvent()`、`Event.initEvent()`、`UIEvent.initUIEvent()`

Navigator

JavaScript 1.0

正在使用的浏览器的信息

Object → Navigator

摘要

`navigator`

属性

`appName`

一个只读字符串，声明了浏览器的代码名。在所有以 Netscape 代码为基础的浏览器中（Netscape、Mozilla 和 Firefox），它的值是“Mozilla”。为了兼容起见，在 Microsoft 的浏览器中，它的值也是“Mozilla”。

`appName`

一个只读字符串，声明了浏览器的名称。在基于 Netscape 的浏览器中，这个属性的值是“Netscape”。在 IE 中，这个属性的值是“Microsoft Internet Explorer”。其他浏览器可以正确地表示自己或者伪装为其他的浏览器以达到兼容性。

`appVersion`

一个只读的字符串，声明了浏览器的平台和版本信息。这个字符串的第一部分是版本号。把该字符串传递给 `parseInt()` 只能获取主版本号，传递给 `parseFloat()` 可以以浮点值的形式获得主版本号和副版本号。该属性的其余部分提供了有关浏览器版本的其他细节，包括运行它的操作系统的信息。但是，不同浏览器提供的信息的格式不同。

`cookieEnabled`

一个只读布尔值，如果浏览器启用了 cookie，该属性值为 `true`。如果禁用了 cookie，该属性值为 `false`。

`mimeTypes[]`

一个 `MimeType` 对象的数组，其中的每个元素代表浏览器支持的一种 MIME 类型（如“text/html”和“image/gif”）。这个数组可以以数字索引，也可以以 MIME 类型的名称来索引。虽然 `mimeTypes[]` 数组是由 IE 4 定义的，但是在 IE 4 中它却总是空的，因为 IE 4 不支持 `MimeType` 对象。

`platform`

一个只读字符串，声明了运行浏览器的操作系统和（或）硬件平台。虽然该属性没有标准的值集合，但它有些常用值，如“Win32”、“MacPPC”和“Linuxix86”，等等。

plugins[]

一个 Plugin 对象的数组，其中的元素代表浏览器已经安装的插件。Plug-in 对象提供的是有关插件的信息，其中包括它所支持的 MIME 类型的列表。

虽然 plugins[] 数组是由 IE 4 定义的，但是在 IE 4 中它却总是空的，因为 IE 4 不支持插件和 Plugin 对象。

userAgent

一个只读的字符串，声明了浏览器用于 HTTP 请求的用户代理头的值。一般说来，它是在 navigator.appCodeName 的值之后加上斜线和 navigator.appVersion 的值构成的。例如：

```
Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
```

函数

navigator.javaEnabled()

检测当前的浏览器是否支持并启用了 Java。

描述

Navigator 对象包含的属性描述了正在使用的浏览器。可以使用这些属性进行平台专用的配置。虽然这个对象的名称显而易见指的是 Netscape 的 Navigator 浏览器，但其他实现了 JavaScript 的浏览器也支持这个对象。Navigator 对象的实例是惟一的，可以用 Window 对象的 navigator 属性来引用它。

从历史来看，Navigator 对象已经用作“客户机嗅探”来根据所用的浏览器而运行不同的代码。例 14-3 展示了做到这一点的一种简单方法，例中的文字描述了依赖 Navigator 对象的很多缺陷。第 13.6.3 节介绍了跨浏览器兼容性的一种更好方法。

参阅

MimeType、Plugin

Navigator.javaEnabled()

JavaScript 1.1

检测 Java 是否可用

摘要

navigator.javaEnabled()

返回值

如果当前浏览器支持 Java，并启用了它，则返回 true，否则返回 false。

描述

可以使用 navigator.javaEnabled() 来检测当前浏览器是否支持 Java，从而知道浏览器是否能显示 Java 小程序。

Node

1 级核心 DOM

文档树中的一个节点

子接口

Attr、CDATASection、CharacterData、Comment、Document、DocumentFragment、DocumentType、Element、ProcessingInstruction、Text

常量

每个 Node 对象都实现了上面列出的一种子接口。每个 Node 对象都有 `nodeType` 属性，声明了它们实现的是哪种子接口。下面的常量是该属性的合法值，它们名称的含义很明显，不需要解释。注意，它们是 `Node()` 构造函数的静态属性，不是个别 Node 对象属性。此外还要注意，Internet Explorer 不支持这些常量。为了实现 IE 中的兼容性，必须直接使用数字直接量。例如，用 1 来代替 `Node.ELEMENT_NODE`：

```
Node.ELEMENT_NODE = 1;           // Element
Node.ATTRIBUTE_NODE = 2;         // Attr
Node.TEXT_NODE = 3;              // Text
Node.CDATA_SECTION_NODE = 4;     // CDATASection
Node.PROCESSING_INSTRUCTION_NODE = 7; // ProcessingInstruction
Node.COMMENT_NODE = 8;           // Comment
Node.DOCUMENT_NODE = 9;          // Document
Node.DOCUMENT_TYPE_NODE = 10;    // DocumentType
Node.DOCUMENT_FRAGMENT_NODE = 11; // DocumentFragment
```

属性

`readonly Attr[] attributes`

如果这是一个 Element 节点，那么 `attributes` 属性是一个只读的、类似数组的 Attr 节点，它表示这个元素的属性。注意，这个数组是“活”的：对该元素属性的任何改变都会立即通过数组看到。

从技术上讲，数组 `attributes[]` 是一个 `NamedNodeMap` 对象。1 级 DOM 标准指定了 `NamedNodeMap` 接口并且定义了很多方法用来查询、设置和删除元素。Element 接口定义了更好的方法来设置和查询元素属性，并且这里没有其他和客户端 JavaScript 相关的 `NamedNodeMap` 的使用。因此，本书并没有介绍 `NamedNodeMap`。把 `attributes` 属性当作是 Attr 对象的一个只读的数组，或者使用 Element 定义的方法来查询、设置和删除属性。

`readonly Node[] childNodes`

存放当前节点的子节点。该属性绝不会为 `null`，因为对于没有子节点的节点来说，`childNodes` 是一个 `length` 为 0 的数组。从技术上说来，该属性是一个 `NodeList` 对象，但它的行为更像 Node 对象的数组。注意，返回的 `NodeList` 对象是“活”的，子元素属性发生任何改变，通过它都可以立刻看到。

readonly Node firstChild

当前节点的第一个子节点，如果当前节点没有子节点，则为 null。

readonly Node lastChild

当前节点的最后一个子节点，如果当前节点没有子节点，则为 null。

readonly String localName[2 级 DOM]

在使用名字空间的 XML 文档中，它声明了该元素的本地部分或属性名。HTML 文档从不使用该属性。参阅 namespaceURI 属性和 prefix 属性。

readonly String namespaceURI[2 级 DOM]

在使用名字空间的 XML 文档中，声明一个 Element 节点或 Attribute 节点的名字空间的 URI。HTML 文档从不使用该属性。参阅 localName 属性和 prefix 属性。

readonly Node nextSibling

在 parentNode 的 childNodes[] 数组中，紧接着当前节点的兄弟节点。如果没有这样的节点，则为 null。

readonly String nodeName

节点的名称。对于 Element 节点，该属性声明了该元素的标记名，用 Element 接口的 tagName 属性也可以获取它。对于其他类型的节点，它的值由节点类型决定。参阅“描述”部分中的表。

readonly unsigned short nodeType

节点的类型，如该节点实现了哪种子接口。前面列出的常量定义了它的合法值。但由于 Internet Explorer 不支持那些常量，所以可以用硬编码的值来代替这些常量。在 HTML 文档中，该属性对于 Element 节点的值是 1，对 Text 节点该属性值是 3，对 Comment 节点该属性的值是 8，对顶层 Document 节点该属性的值是 9。

String nodeValue

节点的值。对于 Text 节点来说，该属性存放的是文本的内容。对于其他类型的节点，该属性的值由 nodeType 决定。参阅下面的表。

readonly Document ownerDocument

该节点所属的 Document 对象。对于 Document 节点来说，该属性为 null。

readonly Node parentNode

当前节点的父节点（或包容节点），如果没有父节点，则为 null。注意，Document 节点、DocumentFragment 节点和 Attr 节点绝不会有父节点。另外，从文档中删除的节点或新创建的、还没有插入文档树的节点，它们的 parentNode 属性都为 null。

String prefix [2 级 DOM]

对于使用名字空间的 XML 文档，该属性声明了 Element 节点或 Attribute 节点的名字空间的前缀。HTML 文档从不使用该属性。参阅 localName 属性和 namespaceURL 属性。

readonly Node previousSibling

在 parentNode 的 childNodes[] 数组中，紧挨着当前节点、位于它之前的兄弟节点。如果没有这样的节点，则为 null。

readonly String xml [仅用于 IE]

如果节点是一个 XML 文档或者是 XML 文档中的一个 Element，这个特定于 IE 的属性就把元素或文档的文本作为一个字符串返回。比较这一属性和 HTML 元素的 innerHTML 属性，参阅 XMLSerializer 了解一种跨平台的替代方法。

方法

appendChild()

通过把一个节点附加到当前节点的 childNodes[] 数组，给文档树添加节点。如果文档树中已经存在了该节点，则把它删除，然后在新位置插入它。

cloneNode()

复制当前节点，或复制当前节点及其所有子孙节点。

hasAttributes()[2 级 DOM]

如果当前节点是 Element 节点，而且具有属性，则返回 true。

hasChildNodes()

如果当前节点具有子节点，就返回 true。

insertBefore()

在文档树中插入一个节点，插入到当前节点的指定子节点之前。如果文档树中已经存在要插入的节点，则删除该节点，重新插入到它的新位置。

isSupported()[2 级 DOM]

如果当前节点支持指定特性的特定版本，则返回 true。

normalize()

通过删除当前节点的所有空 Text 节点，合并相邻的 Text 节点，“规范化”它的所有 Text 子孙节点。

removeChild()

从文档树中删除（并返回）指定的子节点。

replaceChild()

从文档树中删除（并返回）指定的子节点，用另一个节点替换它。

selectNodes()[仅用于 IE]

这个特定于 IE 的方法，使用这一节点作为根执行一个 XPath 查询，并把结果作为一个 NodeList 返回。参阅 Document.evaluate() 和 Document.createExpression() 了解基于 DOM 的替代。

selectSingleNode()[仅用于 IE]

这个特定于 IE 的方法，使用这一节点作为根执行一个 XPath 查询，并把结果作为单个的节点返回。参阅 Document.evaluate() 和 Document.createExpression() 了解基于 DOM 的替代。

transformNode()[仅用于 IE]

这个特定于 IE 的方法，对该节点应用一个 XSLT 样式表，并把结果作为一个 String 返回。参阅 XSLTProcessor 了解一个非 IE 的替代。

`transformNodeToObject()` [仅用于 IE]

这个特定于 IE 的方法，对该节点应用一个 XSLT 样式表，并把结果作为一个新的 Document 对象返回。参阅 `XSLTProcessor` 了解一个非 IE 的替代。

描述

文档树中的所有对象（包括 Document 对象自身）都实现了 Node 接口，它提供了遍历和操作文档树的基本属性和方法（在 IE 中，Node 接口还定义了一些特定于 IE 的属性和方法，它们和 XML 文档、XPath 表达式以及 XSLT 转换一起使用。参阅第 21 章了解详细情况）。

`parentNode` 属性和 `childNodes[]` 数组可以在文档树中上下移动。通过遍历 `childNodes[]` 数组或使用 `firstChild` 和 `nextSibling` 属性进行循环操作（或用 `lastChild` 和 `previousSibling` 属性反向循环），可以枚举指定节点的子节点。调用 `appendChild()`、`insertBefore()`、`removeChild()` 和 `replaceChild()` 方法，可以通过改变一个节点的子节点修改文档树。

文档树中的每个对象都实现了 Node 接口和更专用的接口，如 Element 接口或 Text 接口。`nodeType` 属性声明了该节点实现的是何种子接口。在使用专用接口的方法和属性前，可以用这个属性检测节点的类型。例如：

```
var n;    // Holds the node we're working with
if (n.nodeType == 1) {           // Or use the constant Node.ELEMENT_NODE
    var tagname = n.tagName; // If the node is an Element, this is the tag name
}
```

属性 `nodeName` 和 `nodeValue` 声明了更多的节点信息，但它们的值由 `nodeType` 决定，如下表所示。注意，子接口通常都定义了专门的属性（如 Element 节点的 `tagName` 属性和 Text 节点的 `data` 属性）来获取该信息。

<code>nodeType</code>	<code>nodeName</code>	<code>nodeValue</code>
<code>ELEMENT_NODE</code>	元素标记名	<code>null</code>
<code>ATTRIBUTE_NODE</code>	属性名	属性值
<code>TEXT_NODE</code>	<code>#text</code>	节点文本
<code>CDATA_SECTION_NODE</code>	<code>#cdata-section</code>	节点文本
<code>PROCESSING_INSTRUCTION_NODE</code>	PI 目标	PI 的剩余部分
<code>COMMENT_NODE</code>	<code>#comment</code>	注释的文本
<code>DOCUMENT_NODE</code>	<code>#document</code>	<code>null</code>
<code>DOCUMENT_TYPE_NODE</code>	文档类型名	<code>null</code>
<code>DOCUMENT_FRAGMENT_NODE</code>	<code>#document-fragment</code>	<code>null</code>

参阅

Document、Element、Text、XMLSerializer、XPathExpression、XSLTProcessor，第 15 章

Node.appendChild()

1 级核心 DOM

插入一个节点，作为当前节点的最后一个子节点

摘要

```
Node appendChild(Node newChild)  
    throws DOMException;
```

参数

newChild

要插入文档的节点。如果该节点是 DocumentFragment 节点，则不会直接插入它，而是插入它的每个子节点。

返回值

加入的节点。

抛出

该方法将在下列环境中抛出如下代码的 DOMException 异常：

HIERARCHY_REQUEST_ERR

当前节点不能有子节点，或者它不能有指定类型的子节点，或者 *newChild* 是该节点的祖先（或是该节点自身）。

WRONG_DOCUMENT_ERR

newChild 的 ownerDocument 属性与当前节点的 ownerDocument 属性不同。

NO_MODIFICATION_ALLOWED_ERR

当前节点是只读的，不允许添加子节点，或者被添加的节点已经是文档树的一部分，它的父节点是只读的，不允许删除子节点。

描述

该方法将把节点 *newChild* 添加到文档中，使它成为当前节点的最后一个子节点。如果文档树中已经存在了 *newChild*，它将从文档树中删除，然后重新插入它的新位置。如果 *newChild* 是 DocumentFragment 节点，则不会直接插入它，而是把它的子节点按序插入当前节点的 *childNodes[]* 数组的末尾。注意，来自一个文档的节点（或由一个文档创建的节点）不能插入另一个文档。也就是说，*newChild* 的 ownerDocument 属性必须与当前节点的 ownerDocument 属性相同。

例子

下面的函数将在文档末尾插入一个新段：

```
function appendMessage(message) {  
    var pElement = document.createElement("p");
```

```
    var messageNode = document.createTextNode(message);
    pElement.appendChild(messageNode);    // Add text to paragraph
    document.body.appendChild(pElement);  // Add paragraph to document body
}
```

参阅

`Node.insertBefore()`、`Node.removeChild()`、`Node.replaceChild()`

Node.cloneNode()

1 级核心 DOM

复制节点及其所有子孙节点

摘要

`Node cloneNode(boolean deep);`

参数

deep

如果该参数为 `true`，`cloneNode()` 就会递归复制当前节点的所有子孙节点。否则，它只复制节点自身。

返回值

当前节点的副本。

描述

该方法将复制并返回调用它的节点的副本。如果传递给它的参数是 `true`，它还将递归复制当前节点的所有子孙节点。否则，它只复制当前节点。返回的节点不属于文档树，它的 `parentNode` 属性为 `null`。当复制的是 `Element` 节点时，它的所有属性都将被复制。但要注意，当前节点上注册的事件监听器函数不会被复制。

Node.hasAttributes()

2 级核心 DOM

判断当前节点是否具有属性

摘要

`boolean hasAttributes();`

返回值

如果当前节点具有一个或多个属性，则返回 `true`，否则返回 `false`。注意，只有 `Element` 节点可以具有属性。

参阅

`Element.getAttribute()`、`Element.hasAttribute()`、`Node`

Node.hasChildNodes()1 级核心 DOM

判断当前节点是否具有子节点

摘要

```
boolean hasChildNodes();
```

返回值

如果当前节点具有一个或多个子节点，则返回 `true`，否则返回 `false`。

Node.insertBefore()1 级核心 DOM

在文档树中的指定节点前插入一个节点

摘要

```
Node insertBefore(Node newChild,  
                  Node refChild)  
    throws DOMException;
```

参数

newChild

要插入文档树的节点。如果它是 `DocumentFragment` 节点，则插入它的子节点。

refChild

位于要插入的 *newChild* 之前的子节点。如果该参数为 `null`，则把 *newChild* 节点插入到当前节点的最后一个子节点。

返回值

被插入的节点。

抛出

该方法将在下列环境中抛出如下 `code` 的 `DOMException` 异常：

`HIERARCHY_REQUEST_ERR`

当前节点不支持子节点，或者它不能有指定类型的子节点，或者 *newChild* 是该节点的祖先节点（或是该节点自身）。

`WRONG_DOCUMENT_ERR`

newChild 的 `ownerDocument` 属性与当前节点的 `ownerDocument` 属性不同。

`NO_MODIFICATION_ALLOWED_ERR`

当前节点是只读的，不允许添加子节点，或者 *newChild* 的父节点是只读的，不允许删除子节点。

NOT_FOUND_ERR

*refChild*不是当前节点的子节点。

描述

该方法将把节点*newChild*作为当前节点的子节点插入文档树。新节点将存放在当前节点的*childNodes[]*数组中,紧接在*refChild*节点前。如果*refChild*为null,*newChild*将插入到*childNodes[]*数组的末尾,如*appendChild()*方法一样。注意,如果*refChild*不是当前节点的子节点,调用该方法是不合法的。

如果文档树中已经存在*newChild*节点,那么它将从文档树中删除,然后重新插入到它的新位置。如果*newChild*是DocumentFragment节点,那么插入文档树的不是它自身,而是它的子节点,这些子节点将按顺序插在指定的位置上。

例子

下面的函数将把一个新段插入到文档的开头:

```
function insertMessage(message) {  
    var paragraph = document.createElement("p"); // Create a <p> Element  
    var text = document.createTextNode(message); // Create a Text node  
    paragraph.appendChild(text); // Add text to the paragraph  
    // Now insert the paragraph before the first child of the body  
    document.body.insertBefore(paragraph, document.body.firstChild)  
}
```

参阅

Node.appendChild()、Node.removeChild()、Node.replaceChild()

Node.isSupported()

2 级核心 DOM

判断当前节点是否支持某个特性

摘要

```
boolean isSupported(String feature,  
                     String version);
```

参数

feature

要检测的特性的名称。

version

要检测的特性的版本号,如果要检测对该特性的版本的支持,则为空串。

返回值

如果当前节点支持指定特性的指定版本,则返回 true,否则返回 false。

描述

W3C DOM 标准被模块化，它的实现不必实现标准规定的所有模块或特性。该方法用于检测当前节点的实现是否支持指定特性的指定版本。关于参数 *feature* 和 *version* 的值的列表，请参阅“DOMImplementation.hasFeature()”参考页。

参阅

DOMImplementation.hasFeature()

Node.normalize()

1 级核心 DOM

合并相邻的 Text 节点并删除空的 Text 节点

摘要

```
void normalize();
```

描述

该方法将遍历当前节点的所有子孙节点，通过删除空的 Text 节点，以及合并所有相邻的 Text 节点来规范化文档。该方法在进行节点的插入或删除操作后，对于简化文档树的结构很有用。

参阅

Text

Node.removeChild()

1 级核心 DOM

删除（并返回）当前节点的指定子节点

摘要

```
Node removeChild(Node oldChild)
    throws DOMException;
```

参数

oldChild
要删除的子节点。

返回值

被删除的节点。

抛出

该方法将在下列环境中抛出如下代码的 DOMException 异常：

NO_MODIFICATION_ALLOWED_ERR

当前节点是只读的，不允许删除子节点。

NOT_FOUND_ERR

oldChild 不是当前节点的子节点。

描述

该方法将从当前节点的 `childNodes[]` 数组中删除指定的子节点。如果调用该方法时传递的节点不是当前节点的子节点，将引发错误。该方法在删除 *oldChild* 后，将返回它。*oldChild* 仍然是有效节点，可以再次插入文档。

例子

可以用下面的代码删除文档主体的最后一个子节点：

```
document.body.removeChild(document.body.lastChild);
```

参阅

`Node.appendChild()`、`Node.insertBefore()`、`Node.replaceChild()`

Node.replaceChild()

1 级核心 DOM

用新节点替换一个子节点

摘要

```
Node replaceChild(Node newChild,  
                  Node oldChild)  
    throws DOMException;
```

参数

newChild

新的替换节点。

oldChild

要被替换的节点。

返回值

从文档树中删除和被替换的节点。

抛出

该方法将在下列环境中抛出如下代码的 `DOMException` 异常：

HIERARCHY_REQUEST_ERR

当前节点不能有子节点，或者它不能有指定类型的子节点，也可能 *newChild* 是该节点的祖先节点（或是该节点自身）。

WRONG_DOCUMENT_ERR

newChild 的 `ownerDocument` 属性与当前节点的 `ownerDocument` 属性不同。

NO_MODIFICATION_ALLOWED_ERR

当前节点是只读的，不允许替换子节点，或者 *newChild* 是子节点，它的父节点不允许删除子节点。

NOT_FOUND_ERR

oldChild 不是当前节点的子节点。

描述

该方法将用一个节点替换文档树的节点。*oldChild* 是被替换的节点，它必须是当前节点的子节点。*newChild* 将在当前节点的 `childNodes[]` 数组中代替 *oldChild*。

如果文档树中已经存在 *newChild* 节点，那么它将被从文档树中删除，然后重新插入它的新位置。如果 *newChild* 是 `DocumentFragment` 节点，那么插入文档树的不是它自身，而是它的子节点，按顺序插在原来由 *oldChild* 占有的位置上。

例子

下面的代码将用元素 `` 替换节点 *n*，然后把替换掉的节点插入 `` 元素，这种操作是重定节点 *n* 的父节点，使它以粗体字显示：

```
// Get the first child node of the first paragraph in the document
var n = document.getElementsByTagName("p")[0].firstChild;
var b = document.createElement("b"); // Create a <b> element
n.parentNode.replaceChild(b, n);      // Replace the node with <b>
b.appendChild(n);                      // Reinsert the node as a child of <b>
```

参阅

`Node.appendChild()`、`Node.insertBefore()`、`Node.removeChild()`

Node.selectNodes()

IE 6

用一个 XPath 查询选择节点

摘要

`NodeList selectNodes(String query)`

参数

query

XPath 查询串。

返回值

包含了匹配查询的节点的一个 `NodeList`

描述

这个特定于 IE 的方法计算一个 XPath 表达式，使用这个节点作为查询的根节点，并且将结果作为一个 NodeList 返回。这个 `selectNodes()` 方法只在 XML 文档节点上存在，在 HTML 文档中不存在。注意，既然 Document 对象是它们自己的节点，这个方法可以应用于整个 XML 文档。

参阅 `Document.evaluate()` 了解一个跨浏览器的替代。

参阅

`Document.evaluate()`、`XPathExpression`，第 21 章

Node.selectSingleNode()

IE 6

查找和 XPath 查询匹配的一个节点

摘要

`Node selectSingleNode(String query)`

参数

query

XPath 查询串。

返回值

匹配查询的一个单独的 Node，如果没有，则为 `null`。

描述

这个特定于 IE 的方法计算一个 XPath 表达式，使用这个节点作为 Context 节点。它返回找到的第一个匹配的节点，如果没有匹配的节点就返回 `null`。这 `selectSingleNode()` 方法只在 XML 文档节点上存在，在 HTML 文档中不存在。注意，既然 Document 对象是它们自己的节点，这个方法可以应用于整个 XML 文档。

参阅 `Document.evaluate()` 了解一个跨浏览器的替代。

参阅

`Document.evaluate()`、`XPathExpression`，第 21 章

Node.transformNode()

IE 6

使用 XSLT 把一个节点转换为一个字符串

摘要

`String transformNode(Document xslt)`

参数

xslt

一个 XSLT 样式表，解析为一个 Document 对象

返回值

对该节点及其子孙应用指定的样式而产生的文本。

描述

这个特定于 IE 的方法根据一个 XSLT 样式表指定的规则来转换一个 Node 及其子孙，并将结果作为一个为解析的字符串返回。`transformNode()` 方法只存在于 XML 文档的节点上，HTML 文档则没有该方法。注意，既然 Document 对象本身是节点，这个方法可以应用于整个 XML 文档。

参阅 `XSLTProcessor` 了解其他浏览器中的相似功能。

参阅

`XSLTProcessor`、`Node.transformNodeToObject()`，第 21 章

Node.transformNodeToObject()

IE 6

使用 XSLT 把一个节点转换为一个文档

摘要

`Document transformNodeToObject(Document xslt)`

参数

xslt

一个 XSLT 样式表，解析为一个 Document 对象。

返回值

转换的结果，解析为一个 Document 对象。

描述

这个特定于 IE 的方法根据一个 XSLT 样式表指定的规则来转换一个 Node 及其子孙，并将结果作为一个 Document 对象返回。`transformNodeToObject()` 方法只存在于 XML 文档的节点上，HTML 文档则没有该方法。注意，既然 Document 对象本身是节点，这个方法可以应用于整个 XML 文档。

参阅 `XSLTProcessor` 了解其他浏览器中的相似功能。

参阅

XSLTProcessor、Node.transformNode(), 第 21 章

NodeList

1 级核心 DOM

节点的只读数组

Object → NodeList

属性

readonly unsigned long length
数组中的节点数。

方法

item()
返回数组的指定元素。

描述

NodeList 接口定义 Node 对象的只读有序列表（即数组）。length 属性声明了列表中有多少节点。用 item() 方法可以获取列表中指定位置的节点。NodeList 的元素都是有效的 Node 对象，它不会包含 null 元素。

在 JavaScript 中，NodeList 对象的行为和 JavaScript 数组相似，可以用数组符号[]从列表中查询元素，而不必调用 item() 方法。但不能用[]给 NodeList 对象添加新节点。由于把 NodeList 对象看做 JavaScript 只读数组进行处理更容易，所以本书采用 Element[] 或 Node[]（即 Element 数组或 Node 数组）代替 NodeList。本书中介绍的 Document.getElementsByTagName()、Element.getElementsByTagName() 和 HTMLDocument.getElementsByTagName() 方法，它们都返回一个 Element[] 而不是 NodeList 对象。同样，虽然技术上说，Node 对象的 childNodes 属性是 NodeList 对象，但“Node”参考页将它定义为 Node[]，属性自身通常被称作“childNodes[] 数组”。

注意，NodeList 对象是“活”的，它们不是静态的，会立刻反映出文档树的变化。例如，如果有一个 NodeList 对象，表示指定节点的子节点，然后删除其中一个子节点，那么该子节点将从 NodeList 中删除。如果在遍历 NodeList 的元素时，循环的主体修改了文档树（如删除节点），那么将影响 NodeList 的内容，所以要仔细操作。

参阅

Document、Element

NodeList.item()

1 级核心 DOM

获取 NodeList 中的元素

摘要

Node item(unsigned long index);

参数

index

要获取的节点在 `NodeList` 中的位置（或下标）。`NodeList` 中的第一个节点下标为 0，最后一个节点的下标为 `length - 1`。

返回值

`NodeList` 中指定位置的节点。如果 *index* 小于 0 或大于等于 `NodeList` 的长度，则返回 `null`。

描述

该方法将返回 `NodeList` 中的指定元素。在 JavaScript 中，可以用数组符号 `[]` 代替调用方法 `item()`。

Option

2 级 DOM HTML

Select 框中的一个选项

Node → Element → HTMLElement → HTMLOptionElement

构造函数

可以使用 `Document.createElement()` 来创建 `Option` 对象，就像其他标记一样。在 0 级 DOM 中，可以使用 `Option()` 构造函数动态地创建 `Option` 对象：

```
new Option(String text, String value,  
           boolean defaultSelected, boolean selected)
```

参数

text

一个可选的字符串参数，指定了 `Option` 对象的 `text` 属性。

value

一个可选的字符串参数，指定了 `Option` 对象的 `value` 属性。

defaultSelected

一个可选的布尔参数，指定了 `Option` 对象的 `defaultSelected` 属性。

selected

一个可选的布尔参数，指定了 `Option` 对象的 `selected` 属性。

属性

`boolean defaultSelected`

`<option>` 元素的 `selected` 属性的初始值。如果表单被重置，则 `selected` 属性重置为这个属性的值。设置这个属性也会设置 `selected` 属性的值。

`boolean disabled`

如果为 `true`，这个 `<option>` 元素将被禁用，用户不能选择它。映射 HTML 属性 `disabled`。

`readonly HTMLFormElement form`

对包含这个元素的 `<form>` 元素的引用。

`readonly long index`

这个 `<option>` 元素在包含它的 `<select>` 元素中的位置。

`String label`

选项显示的文本。映射HTML属性 `label`。如果没有设置该属性,将采用 `<option>` 元素的纯文本内容。

`boolean selected`

该选项的当前状态:如果为 `true`,该选项被选中。这个属性的初始值来自 `selected` 属性。

`readonly String text`

包含在 `<option>` 元素中的纯文本。这个文本会作为选项的标签出现。

`String value`

当表单提交发生的时候,如果这个选项被选中的话, `value` 会和表单一起提交。映射 `value` 属性。

HTML 语法

Option对象是由 `<select>` 标记中的 `<option>` 标记创建的,其中 `<select>` 标记位于 `<form>` 标记中。一般说来,可以在 `<select>` 标记中放多个 `<option>` 标记。

```
<form ...>
<select ...>
  <option
    [ value="value" ] // The value returned when the form is submitted
    [ selected ] >   // Specifies whether this option is initially selected
    plain_text_label // The text to display for this option
  [ </option> ]
  ...
</select>
...
</form>
```

描述

Option对象描述的是Select对象中显示的一个选项。该对象的属性声明了在默认情况下它是否被选中了,当前它是否被选中了,它在包容的Select对象的 `options[]` 数组中的位置,它要显示的文本以及如果在提交表单时它处于选中的状态,要传递给服务器的值。

注意,虽然选项显示的文本是在标记 `<option>` 的外部设置的,但它必须是纯文本,即不能具有任何HTML标记的非格式化,这样它才能在不支持HTML格式的列表框或下拉式菜单中正确地显示出来。

可以用构造函数 `Option()` 动态地创建一个新Option对象。一旦创建了新的Option对象,它就会通过 `Select.add()` 添加到Select对象的选项列表中。参阅 `Select.options[]` 了解更多细节。

参阅

Select、Select.options[], 第 18 章

Packages

参阅本书第三部分。

Password

参阅 Input。

Plugin

JavaScript 1.1; IE 不支持

描述已安装的插件

Object → Plugin

摘要

```
navigator.plugins[i]  
navigator.plugins['name']
```

属性

description

一个只读字符串，包含人们可以读懂的插件说明。该说明文本由插件的创建者提供，包括厂商的信息、版本信息和插件功能的简短说明。

filename

一个只读字符串，声明了硬盘上存放插件程序的文件名。不同平台采用的名称不同。对于标识插件，name 属性比 filename 属性更有用。

length

每个 Plugin 对象也都是 MimeType 对象的一个数组，其中的元素声明了该插件支持的数据格式。和其他所有的数组一样，属性 length 声明了数组中的元素个数。

name

一个只读字符串，声明了插件的名称。每个插件都应该有惟一标识它的名称。插件的名称可以用作数组 navigator.plugins[] 的下标。利用这一事实，可以轻松地判断出当前浏览器是否安装了指定的插件：

```
var flash_installed = (navigator.plugins["Shockwave Flash"] != null);
```

数组元素

Plugin 对象的数组元素是 MimeType 对象，它们声明了插件支持的数据格式。length 属性说明了这个数组中的 MimeType 对象的个数。

描述

所谓插件，就是一个软件模块，浏览器可以调用这个模块显示嵌入浏览器窗口的特殊数据

类型。插件是用 Plugin 对象表示的，而 Navigator 对象的 `plugins[]` 属性是 Plugin 对象的数组，表示浏览器中已经安装的插件。IE 不支持 Plugin 对象，并且该浏览器上的 `navigator.plugins[]` 数组总是为空。

当你想遍历已安装插件的列表、查找符合要求的插件时（例如，查找一个支持你想嵌入网页的数据的 MIME 类型的插件），可以以数字作为数组 `navigator.plugins[]` 的下标。还可以使用插件名作为数组 `navigator.plugins[]` 的下标。也就是说，如果你要查看一个具体的插件是否在用户的浏览器中安装了，使用的代码如下：

```
var flash_installed = (navigator.plugins["Shockwave Flash"] != null);
```

用作数组下标的插件名与 Plugin 对象的属性 `name` 的值完全相同。

Plugin 对象多少有些不寻常，因为它既有常规对象的属性又有数组元素。Plugin 对象的属性提供了有关插件的各种片段信息，其数组元素是 `MimeType` 对象，说明了插件所支持的嵌入的数据格式。Plugin 对象存放在 Navigator 对象的数组中，而 Plugin 对象自身是 `MimeType` 对象的数组，不要混淆了它们。因为其中涉及两个数组，所以可能会得到如下代码：

```
navigator.plugins[i][j]           // The jth MIME type of the ith plug-in  
navigator.plugins["LiveAudio"][0] // First MIME type of LiveAudio plug-in
```

最后要注意的是，不仅用 Plugin 对象的数组元素可以声明插件支持的 MIME 类型，使用 `MimeType` 对象的 `enabledPlugin` 属性也可以判断插件是否支持给定的 MIME 类型。

参阅

Navigator、MimeType

ProcessingInstruction

1 级 DOM XML

XML 文档中的处理指令

Node → ProcessingInstruction

属性

String data

处理指令的内容（即从目标开始后的第一个非空格字符到结束字符“`?>`”之间的字符，但不包括“`?>`”）。

readonly String target

处理指令的目标。它是“`<?`”后的第一个标识符，指定了处理指令的处理器。

描述

这个不常用的接口表示 XML 文档中的一个处理指令（或 PI）。使用 HTML 文档的程序设计者不会遇到 `ProcessingInstruction` 节点。

参阅

`Document.createProcessingInstruction()`

Radio

参阅 Input

Range

2 级 DOM Range

表示文档中的连续范围

Object → Range

常量

这些常量指定了如何比较 Range 对象的边界点。它们是 compareBoundaryPoints() 方法的参数 how 的合法值。参阅 “Range.compareBoundaryPoints()” 参考页。

unsigned short START_TO_START = 0

用指定范围的开始点与当前范围的开始点进行比较。

unsigned short START_TO_END = 1

用指定范围的开始点与当前范围的结束点进行比较。

unsigned short END_TO_END = 2

用指定范围的结束点与当前范围的结束点进行比较。

unsigned short END_TO_START = 3

用指定范围的结束点与当前范围的开始点进行比较。

属性

Range 接口定义了下列属性。注意，所有属性都是只读的，不能通过设置这些属性改变范围的开始点或结束点，必须调用 setEnd() 方法或 setStart() 方法实现这一点。还要注意，调用 Range 对象的 detach() 方法后，对这些属性的任何读操作都会抛出代码为 INVALID_STATE_ERR 的 DOMException 异常。

readonly boolean collapsed

如果范围的开始点和结束点在文档的同一位置，则为 true，也就是说，范围是空的，或“折叠”的。

readonly Node commonAncestorContainer

包含范围的开始点和结束点的（即它们的祖先节点）、嵌套最深的 Document 节点。

readonly Node endContainer

包含范围的结束点的 Document 节点。

readonly long endOffset

endContainer 中的结束点位置。

readonly Node startContainer

包含范围的开始点的 Document 节点。

readonly long startOffset

开始点在 startContainer 中的位置。

方法

Range 接口定义了下列方法。注意，如果调用了范围的 `detach()` 方法，那么接下来调用 Range 对象的任何方法都会抛出代码为 `INVALID_STATE_ERR` 的 `DOMException` 异常。因为在该接口中的这种异常普遍存在，所以 Range 方法的参考页中没有列出它。

`cloneContents()`

返回新的 `DocumentFragment` 对象，它包含该范围表示的文档区域的副本。

`cloneRange()`

创建一个新 Range 对象，表示与当前的 Range 对象相同的文档区域。

`collapse()`

折叠该范围，使它的边界点重合。

`compareBoundaryPoints()`

比较指定范围的边界点和当前范围的边界点，根据它们的顺序返回 -1、0 和 1。比较哪个边界点由它的第一个参数指定，它的值必须是前面定义的常量之一。

`deleteContents()`

删除当前 Range 对象表示的文档区域。

`detach()`

通知实现不再使用当前的范围，可以停止跟踪它。如果调用了范围的这个方法，那么接下来调用的该范围任何方法都会抛出代码为 `INVALID_STATE_ERR` 的 `DOMException` 异常。

`extractContents()`

删除当前范围表示的文档区域，并且以 `DocumentFragment` 对象的形式返回那个区域的内容。该方法和 `cloneContents()` 方法与 `deleteContents()` 方法的组合很相似。

`insertNode()`

把指定的节点插入文档范围的开始点。

`selectNode()`

设置该范围的边界点，使它包含指定的节点和它的所有子孙节点。

`selectNodeContents()`

设置该范围的边界点，使它包含指定节点的子孙节点，但不包含指定的节点自身。

`setEnd()`

把该范围的结束点设置为指定的节点和偏移量。

`setEndAfter()`

把该范围的结束点设置为紧邻指定节点的节点之后。

`setEndBefore()`

把该范围的结束点设置为紧邻指定节点之前。

`setStart()`

把该范围的开始点设置为指定的节点中的指定偏移量。

`setStartAfter()`

把该范围的开始点设置到紧邻指定节点的节点之后。

`setStartBefore()`

把该范围的开始点设置到紧邻指定节点之前。

`surroundContents()`

把指定的节点插入文档范围的开始点，然后重定范围中所有节点的父节点，使它们成为新插入的节点的子孙节点。

`toString()`

返回该范围表示的文档区域的纯文本内容。

描述

`Range` 对象表示文档的连续范围或区域，如用户在浏览器窗口中用鼠标拖动选中的区域。如果一个实现支持 `Range` 模块，那么 `Document` 对象就定义了 `createRange()` 方法，调用它可创建新的 `Range` 对象（但要注意，Internet Explorer 定义了不兼容的 `Document.createRange()` 方法，它返回的对象与 `Range` 接口相似，但不兼容）。`Range` 接口为指定文档“选中”的区域定义了大量的方法，此外还有几个方法可以用于在选中的区域上进行剪切和粘贴类型的操作。

一个范围具有两个边界点，即一个开始点和一个结束点。每个边界点由一个节点和那个节点中的偏移量指定。该节点通常是 `Element` 节点、`Document` 节点或 `Text` 节点。对于 `Element` 节点和 `Document` 节点，偏移量指该节点的子节点。偏移量为 0，说明边界点位于该节点的第一个子节点之前。偏移量为 1，说明边界点位于该节点的第一个子节点之后，第二个子节点之前。但如果边界节点是 `Text` 节点，偏移量则指的是文本中两个字符之间的位置。

`Range` 接口的属性提供了获取范围的边界节点和偏移量的方法。它的方法提供了设置范围边界的方法。注意，范围的边界可以设置为 `Document` 或 `DocumentFragment` 对象中的节点。

一旦定义了范围的边界点，就可以使用 `deleteContents()`、`extractContents()`、`cloneContents()` 和 `insertNode()` 方法实现剪切、复制和粘贴的操作。

当通过插入或删除操作改变了文档时，表示文档一部分的所有 `Range` 对象都将被改变（如果必要的话），以便使它们的边界点保持有效，并且让它们（尽可能接近地）表示同样的文档内容。

参阅

`Document.createRange()`、`DocumentFragment`

`Range.cloneContents()`

2 级 DOM `Range`

把范围的内容复制到一个 `DocumentFragment` 对象

摘要

```
DocumentFragment cloneContents()  
    throws DOMException;
```

返回值

一个 DocumentFragment 对象，包含该范围中的文档内容的副本。

抛出

如果当前的范围包含 DocumentType 节点，该方法将抛出代码为 HIERARCHY_REQUEST_ERR 的 DOMException 异常。

描述

该方法将复制当前范围的内容，把它存在一个 DocumentFragment 对象中，返回该对象。

参阅

DocumentFragment、Range.deleteContents()、Range.extractContents()

Range.cloneRange()

2 级 DOM Range

复制该范围

摘要

```
Range cloneRange();
```

返回值

新创建的 Range 对象，与当前范围的边界点相同。

参阅

Document.createRange()

Range.collapse()

2 级 DOM Range

使范围的边界点重合

摘要

```
void collapse(boolean toStart)  
    throws DOMException;
```

参数

toStart

如果该参数为 true，该方法将把范围的结束点设置为与开始点相同的值。否则，它将把范围的开始点设置为与结束点相同的值。

描述

该方法将设置范围的一个边界点，使它与另一个边界点相同。要修改的边界点由参数 *toStart* 指定。该方法返回后，范围将“折叠”，即表示文档中的一个点，没有内容。当范围被折叠后，它的 *collapsed* 属性将被设置为 *true*。

Range.compareBoundaryPoints()

2 级 DOM Range

比较两个范围的位置

摘要

```
short compareBoundaryPoints(unsigned short how,
                             Range sourceRange)
    throws DOMException;
```

参数

how

声明如何执行比较操作（即比较哪些边界点）。它的合法值是 Range 接口定义的常量。

sourceRange

要与当前范围进行比较的范围。

返回值

如果当前范围的指定边界点位于 *sourceRange* 指定的边界点之前，则返回 -1。如果指定的两个边界点相同，返回 0。如果当前范围的指定边界点位于 *sourceRange* 指定的边界点之后，则返回 1。

抛出

如果 *sourceRange* 表示的文档不同于当前范围表示的文档，该方法将抛出代码为 *WRONG_DOCUMENT_ERR* 的 *DOMException* 异常。

描述

该方法将比较当前范围的边界点和指定的 *sourceRange* 的边界点，并返回一个值，声明它们在源文档中的相对位置。参数 *how* 指定了比较两个范围的哪个边界点。该参数的合法值和它们的含义如下：

Range.START_TO_START

比较两个 Range 节点的开始点。

Range.END_TO_END

比较两个 Range 节点的结束点。

Range.START_TO_END

用 *sourceRange* 的开始点与当前范围的结束点比较。

`Range.END_TO_START`

用 `sourceRange` 的结束点与当前范围的开始点比较。

该方法的返回值是一个数字，声明了当前范围相对于 `sourceRange` 的位置。因此，你可能认为，首先需要用参数 `how` 的范围常量指定当前范围的边界点，然后再用它指定 `sourceRange` 的边界点。但事实上，常量 `Range.START_TO_END` 指定与当前范围的 `end` 点和 `sourceRange` 的 `start` 点进行比较。同样，常量 `Range.END_TO_START` 常量指定比较当前范围的 `start` 点和指定范围的 `end` 点。

`Range.deleteContents()`

2 级 DOM Range

删除文档的区域

摘要

```
void deleteContents()  
    throws DOMException;
```

抛出

如果当前范围表示的部分文档是只读的，该方法将抛出代码为 `NO_MODIFICATION_ALLOWED_ERR` 的 `DOMException` 异常。

描述

该方法将删除当前范围表示的所有文档内容。当该方法返回时，当前范围的边界点将重合。注意，这种删除操作可以生成相邻的 `Text` 节点，调用 `Node.normalize()` 方法可以合并这些节点。

关于复制文档内容的方法，请参阅 `Range.cloneContents()`。关于进行文档内容的复制和删除操作的方法，请参阅 `Range.extractContents()`。

参阅

`Node.normalize()`、`Range.cloneContents()`、`Range.extractContents()`

`Range.detach()`

2 级 DOM Range

释放一个 `Range` 对象

摘要

```
void detach()  
    throws DOMException;
```

抛出

和所有 `Range` 方法一样，如果在已经被释放了的 `Range` 对象上调用 `detach()`，它将抛出代码为 `INVALID_STATE_ERR` 的 `DOMException` 异常。

描述

DOM 实现将跟踪为文档创建的所有 Range 对象，因为在修改文档时，它们需要改变范围的边界点。当确认 Range 对象不再被使用时，可以调用 detach() 方法，通知实现不必再跟踪该范围。注意，一旦调用了 Range 对象的 detach() 方法，再使用 Range 对象，就会抛出异常。对 detach() 方法的调用不是必需的，但在修改了文档的情况下，调用它可以提高性能。Range 对象不会被立刻回收。

Range.extractContents()

2 级 DOM Range

删除文档内容并以 DocumentFragment 对象的形式返回它

摘要

```
DocumentFragment extractContents()  
    throws DOMException;
```

返回值

一个 DocumentFragment 节点，包含该范围的内容。

抛出

如果要提取的文档内容是只读的，该方法将抛出代码为 NO_MODIFICATION_ALLOWED_ERR 的 DOMException 异常。如果当前范围包括 DocumentType 节点，该方法将抛出代码为 HIERARCHY_REQUEST_ERR 的 DOMException 异常。

描述

该方法将删除文档的指定范围，并返回包含被删除的内容（或被删除的内容的副本）的 DocumentFragment 节点。当返回该方法时，范围将折叠，文档中可能出现相邻的 Text 节点（用 Node.normalize() 方法可以合并）。

参阅

DocumentFragment、Range.cloneContents()、Range.deleteContents()

Range.insertNode()

2 级 DOM Range

在范围的开头插入一个节点

摘要

```
void insertNode(Node newNode)  
    throws RangeException,  
           DOMException;
```


参数

newNode

要插入文档的节点。

抛出

如果 *newNode* 是 `Attr`、`Document`、`Entity` 或 `Notation` 节点，该方法将抛出代码为 `INVALID_NODE_TYPE_ERR` 的 `RangeException` 异常。

在下列条件下，该方法还将抛出如下代码的 `DOMException` 异常：

`HIERARCHY_REQUEST_ERR`

包含范围的开始点的节点不能有子节点，它也不能有指定类型的子节点，或者 *newNode* 是该节点的祖先节点（或是该节点自身）。

`NO_MODIFICATION_ALLOWED_ERR`

包含范围的开始点的节点（或它的祖先节点）是只读的。

`WRONG_DOCUMENT_ERR`

newNode 与范围所属于的文档不同。

描述

该方法将把指定的节点（和它的所有子孙节点）插入文档范围的开始点。当该方法返回时，当前范围将包括新插入的节点。如果 *newNode* 已经是文档的一部分，那么它将被从当前位置删除，然后重新插入范围的开始点。如果 *newNode* 是 `DocumentFragment` 节点，那么插入的不是它自身，而是它的所有子节点，按顺序插入范围的开始点。

如果包含当前范围的开始点的节点是 `Text` 节点，那么在发生插入操作前，它将被分割成两个相邻的节点。如果 *newNode* 是 `Text` 节点，在插入文档后，它不会与任何相邻的 `Text` 节点合并。要合并相邻的节点，需要调用 `Node.normalize()` 方法。

参阅

`DocumentFragment`、`Node.normalize()`

Range.selectNode()

2 级 DOM Range

把范围边界设置为一个节点

摘要

```
void selectNode(Node refNode)
    throws RangeException, DOMException;
```

参数

refNode

被选中的节点（即将成为当前范围的内容的节点）。

抛出

如果 *refNode* 是 *Attr*、*Document* 或 *DocumentFragment* 节点，该方法将抛出代码为 *INVALID_NODE_TYPE_ERR* 的 *RangeException* 异常。

如果 *refNode* 所属的文档与创建该范围的文档不同，该方法将抛出代码为 *WRONG_DOCUMENT_ERR* 的 *DOMException* 异常。

描述

该方法将把范围的内容设置为指定的 *refNode* 节点。也就是说，“选中”那个节点和它的子孙节点。

参阅

`Range.selectNodeContents()`

Range.selectNodeContents()

2 级 DOM Range

把范围的边界设置为一个节点的子节点

摘要

```
void selectNodeContents(Node refNode)
    throws RangeException, DOMException;
```

参数

refNode

其子节点将成为当前范围的内容的节点。

抛出

如果 *refNode* 或它的一个祖先节点是 *DocumentType*、*Entity* 或 *Notation* 节点，该方法将抛出代码为 *INVALID_NODE_TYPE_ERR* 的 *RangeException* 异常。

如果 *refNode* 所属的文档与创建该范围的文档不同，该方法将抛出代码为 *WRONG_DOCUMENT_ERR* 的 *DOMException* 异常。

描述

该方法将设置范围的边界点，使该范围包含 *refNode* 的子节点。

参阅

`Range.selectNode()`

Range.setEnd()

2 级 DOM Range

设置范围的结束点

摘要

```
void setEnd(Node refNode,  
            long offset)  
    throws RangeException, DOMException;
```

参数

refNode

包含新的结束点的节点。

offset

结束点在 *refNode* 中的位置。

抛出

如果 *refNode* 或它的一个祖先节点是 `DocumentType` 节点，该方法将抛出代码为 `INVALID_NODE_TYPE_ERR` 的 `RangeException` 异常。

如果 *refNode* 所属的文档与创建该范围的文档不同，该方法将抛出代码为 `WRONG_DOCUMENT_ERR` 的 `DOMException` 异常。如果 *offset* 是负数，或者大于 *refNode* 中的子节点数或字符数，该方法将抛出代码为 `INDEX_SIZE_ERR` 的 `DOMException` 异常。

描述

该方法将把范围的结束点设置为 `endContainer` 属性和 `endOffset` 属性指定的值。

Range.setEndAfter()

2 级 DOM Range

在指定的节点后结束范围

摘要

```
void setEndAfter(Node refNode)  
    throws RangeException, DOMException;
```

参数

refNode

一个节点，要设置的范围的结束点位于该节点之后。

抛出

如果 *refNode* 是 `Document`、`DocumentFragment` 或 `Attr` 节点，或者 *refNode* 的根包容节点不是 `Document`、`DocumentFragment` 或 `Attr` 节点，该方法将抛出代码为 `INVALID_NODE_TYPE_ERR` 的 `RangeException` 异常。

如果 *refNode* 所属的文档与创建该范围的文档不同，该方法将抛出代码为 `WRONG_DOCUMENT_ERR` 的 `DOMException` 异常。

描述

该方法将把范围的结束点设置为紧邻指定的 *refNode* 节点的位置。

Range.setEndBefore()

2 级 DOM Range

在指定的节点之前结束范围

摘要

```
void setEndBefore(Node refNode)
    throws RangeException, DOMException;
```

参数

refNode

一个节点，要设置的范围的结束点位于该节点之前。

抛出

该方法将在与 `Range.setEndAfter()` 方法一样的环境中抛出相同的异常。详见 `Range.setEndAfter()` 方法的参考页。

描述

该方法将把范围的结束点设置为指定的 *refNode* 节点之前的位置。

Range.setStart()

2 级 DOM Range

设置范围的开始点

摘要

```
void setStart(Node refNode,
              long offset)
    throws RangeException, DOMException;
```

参数

refNode

包含新的开始点的节点。

offset

新开始点在 *refNode* 中的位置。

抛出

该方法抛出异常的原因和 `Range.setEnd()` 方法相同，抛出的异常也相同。详见 `Range.setEnd()` 方法的参考页。

描述

该方法将把范围的开始点设置为 `startContainer` 属性和 `startOffset` 属性指定的值。

Range.setStartAfter()

2 级 DOM Range

在指定的节点后开始范围

摘要

```
void setStartAfter(Node refNode)
    throws RangeException, DOMException;
```

参数

refNode

一个节点，要设置的范围的开始点位于该节点后。

抛出

该方法将出于和 `Range.setEndAfter()` 方法同样的原因抛出同样的异常。详见 `Range.setEndAfter()` 方法的参考页。

描述

该方法将把范围的开始点设置为紧邻指定的 *refNode* 节点的位置。

Range.setStartBefore()

2 级 DOM Range

在指定的节点之前开始范围

摘要

```
void setStartBefore(Node refNode)
    throws RangeException, DOMException;
```

参数

refNode

一个节点，要设置的范围的开始点位于该节点之前。

抛出

该方法将出于和 `Range.setEndAfter()` 方法同样的原因抛出同样的异常。详见 `Range.setEndAfter()` 方法的参考页。

描述

该方法将把范围的开始点设置为指定的 *refNode* 节点之前的位置。

Range.surroundContents()2 级 DOM Range

用指定的节点包围范围的内容

摘要

```
void surroundContents(Node newParent)  
    throws RangeException, DOMException;
```

参数

newParent

将成为当前范围内容的新父节点的节点。

抛出

该方法将在下列环境中抛出具有如下代码的 DOMException 异常或 RangeException 异常：

DOMException.HIERARCHY_REQUEST_ERR

当前范围的开始点的包容节点不能有子节点，不能有 *newParent* 类型的子节点，或者 *newParent* 是包容节点的祖先节点。

DOMException.NO_MODIFICATION_ALLOWED_ERR

当前范围的边界点的祖先节点是只读的，不允许进行插入操作。

DOMException.WRONG_DOCUMENT_ERR

newParent 和它的范围是用不同 Document 对象创建的。

RangeException.BAD_BOUNDARYPOINTS_ERR

当前范围部分地选择了一个节点(除了Text节点外的)，所以不能包围文档的这个区域。

RangeException.INVALID_NODE_TYPE_ERR

newParent 是 Document、DocumentFragment、DocumentType、Attr、Entity 或 Notation 节点。

描述

该方法将把当前范围的父节点重定为 *newParent*，然后把 *newParent* 插在文档中范围的开始位置。例如，把文档的一个区域放入 <div> 或 元素中，可以使用该方法。

如果 *newParent* 已经是文档的一部分，那么它首先将从文档中删除，它的子节点也将被舍弃。当该方法返回时，该范围将以 *newParent* 之前的位置为开始点、*newParent* 之后的位置为结束点。

Range.toString()2 级 DOM Range

以纯文本串形式获取范围的内容

摘要

```
String toString();
```

返回值

以纯文本（不含标记）的形式返回当前范围的内容。

RangeException

2 级 DOM Range

通知发生了范围特有的异常

Object → RangeException

常量

下面的常量为 RangeException 对象的 code 属性定义了合法的值。注意，这些常量是 RangeException 接口的静态属性，不是个别异常对象的属性。

```
unsigned short BAD_BOUNDARYPOINTS_ERR = 1
```

对于请求的操作，范围的边界点不合法。

```
unsigned short INVALID_NODE_TYPE_ERR = 2
```

尝试把范围边界点的包容节点设置为一个无效节点或具有无效祖先的节点。

属性

unsigned short code

提供引发异常的详细信息的出错代码。该属性的合法值（及其含义）由前面列出的常量定义。

描述

RangeException 对象由 Range 接口的某些方法抛出，用于通知某种类型的问题。注意，Range 方法抛出的大多数异常都是 DOMException 对象。只有当现有的 DOMException 错误常量不适合表示当前的异常时，才生成 RangeException 对象。

Reset

参阅 Input

Screen

JavaScript 1.2

提供有关显示器的信息

Object → Screen

摘要

screen

属性

availHeight

声明了显示 web 浏览器的屏幕的可用高度，以像素计。在 Windows 这样的操作系统中，这个可用的高度不包括分配给半永久特性（如屏幕底部的任务栏）的垂直空间。

availWidth

声明了显示 web 浏览器的屏幕的可用宽度，以像素计。在 Windows 这样的操作系统

中, 这个可用的宽度并不包括分配给半永久特性 (如应用程序的快捷方式栏) 的水平空间。

colorDepth

声明了屏幕的颜色深度, 以每像素的位数计。

height

声明了显示 web 浏览器的屏幕的高度, 以像素计。参阅 availHeight。

width

声明了显示 web 浏览器的屏幕的宽度, 以像素计。参阅 availWidth。

描述

每个 Window 对象的 screen 属性都引用一个 Screen 对象。这个全局对象的属性存放了有关显示浏览器屏幕的信息。JavaScript 程序将利用这些信息来优化它们的输出, 以达到用户的显示要求。例如, 一个程序可以根据显示器的尺寸选择使用大图像还是使用小图像, 它还可以根据显示器的颜色深度选择使用 16 位色的图像还是使用 8 位色的图像。另外, JavaScript 程序还能够根据有关屏幕尺寸的信息将新的浏览器窗口定位在屏幕中间。

参阅

Window 对象的 screen 属性

Select

2 级 DOM HTML

图形化选项列表

Node → Element → HTMLElement → Select

属性

readonly Form form

包含这个 <select> 元素的 <form> 元素。

readonly long length

这个 <select> 元素所包含的 <option> 元素的个数。它的值与属性 options.length 的值相同。

readonly HTMLCollection options

Option 对象的一个数组 (HTMLCollection), 它表示包含在这个 <select> 元素中的 <option> 元素, 按照它们出现的顺序。参阅 Select.options[] 了解更多细节。

long selectedIndex

选中的选项在选项数组中的位置。如果没有选项选中, 这个属性就是 -1。如果多个选项选中, 这个属性保存第一个选中选项的下标。

设置这个属性的值, 会选中某个特定的选项并取消对所有其他选项的选定, 即使 Select 对象具有指定的 multiple 属性也是如此。在采用列表框的选择样式 (size>1) 时, 可以把 selectedIndex 设置为 -1 而取消对所有选项的选定。注意, 以这种方式改变选择不会触发事件句柄 onchange()。

readonly String type

如果 `multiple` 属性为 `true`, 那么这个属性的值就为 “select-multiple”, 否则, 它就是 “select-one”。这个属性的存在是为了和 `Input` 对象的 `type` 属性兼容。

除了上述的属性, `Select` 对象还映射了如下的 HTML 属性:

属性	HTML 属性	描述
boolean disabled	disabled	用户元素是否关闭
boolean multiple	multiple	是否可有多个选项选中
String name	name	用于表单提交的元素名
long size	size	一次显示的选项数
long tabIndex	tabindex	Select 元素在切换顺序中的位置

方法

`add()`

向选项数组中插入一个新的 `Option` 对象, 要么将它添加到数组的末尾, 要么把它插入到另一个指定选项的前面。

`blur()`

从这个元素中取走键盘焦点。

`focus()`

把键盘焦点转移到这个元素。

`remove()`

移除指定位置的 `<option>` 元素。

事件句柄

`onchange`

在用户选中了一个选项或取消了对某个选项的选定时调用的事件句柄。

HTML 语法

`Select` 元素是由标准的 HTML 标记 `<select>` 创建的。`Select` 元素中的选项由 `<option>` 标记创建:

```
<form>
...
<select
  name="name" // A name that identifies this element; specifies name property
  [ size="integer" ] // Number of visible options in Select element
  [ multiple ] // Multiple options may be selected, if present
  [ onchange="handler" ] // Invoked when the selection changes
>
<option value="value1" [selected]> option_label1
```

```
<option value="value2" [selected]> option_label2 // Other options here
</select>
...
</form>
```

描述

Select 元素表示一个 HTML `<select>` 标记，它显示一个图形化的选项列表供用户选择。如果在该元素的 HTML 定义中出现了 `multiple` 属性，那么用户就可以从列表中选择任意多个选项。如果没有设置这个属性，那么用户就只能选择一个选项，此时选项的行为就和单选按钮一样，即无论选中了哪一个选项，在它之前选中的选项就会被取消选定。

显示 Select 元素的选项的方式有两种。如果它的 `size` 属性值大于 1，或者设置了 `multiple` 属性，那么可以在一个列表框中显示它的选项，这个列表框在浏览器窗口中的高度为 `size` 行。如果 `size` 的值比选项数小，那么列表框就会加入一个滚动条以便访问所有的选项。不过，如果 `size` 的值为 1 而且又没有设置 `multiple` 属性，那么就会将当前选中的选项显示在一行之中，通过下拉式菜单可以访问其他的选项。第一种显示方式可以清楚地显示出选项，但是占用浏览器窗口的空间比较大。第二种显示方法需要占用的空间最小，但是又不能明确地显示出可选项。

Select 元素的 `options[]` 属性是最为有趣的属性。它是 `Option` 对象的数组，它描述了 Select 元素中显示的一个选项。属性 `length` 声明了数组的长度（与 `options.length` 相同）。要了解详情，请参阅 `Option` 对象的说明。

对于没有指定 `multiple` 属性的一个 Select 元素，可以使用 `selectedIndex` 属性来判断哪个选项被选中。允许多个选项的时候，这个属性告诉你惟一的第一个被选中选项的下标。要确定被选中选项的完整集合，必须要遍历 `options[]` 数组并且察看每个 `Option` 对象的 `selected` 属性。

Select 元素所显示的选项可以动态地修改。使用 `add()` 方法和 `Option()` 构造函数来增加一个新的选项，使用 `remove()` 方法来移除一个选项。直接操作 `options` 数组也可以进行改变。

参阅

Form、Option、Select.options[]，第 18 章

Select.add()

2 级 DOM HTML

插入一个 `<option>` 元素

摘要

```
void add(HTMLElement element,
         HTMLElement before)
    throws DOMException;
```

参数

element

要添加的选项元素。

before

在选项数组的该元素之前增加新的元素。如果这个参数是 `null`，元素添加到选项数组的末尾。

抛出

如果参数 *before* 指定的对象并非 `options` 数组的成员，这个方法抛出一个代码为 `NOT_FOUND_ERR` 的 `DOMException`。

描述

这个方法为这个 `<select>` 元素增加一个新的 `<option>` 元素。*element* 是一个 `Option` 对象，它代表着要添加的 `<option>` 元素。*before* 指定了在哪个元素前添加该 `Option`。如果 *before* 是一个 `<optgroup>` 的一部分，*element* 总是作为同一组的一部分插入。如果 *before* 为 `null`，则 *element* 成为 `<select>` 的最后一个孩子。

参阅

`Option`

Select.blur()

2 级 DOM HTML

从这个元素移走键盘焦点

摘要

```
void blur();
```

描述

这个方法从这个元素移走键盘焦点。

Select.focus()

2 级 DOM HTML

给这个元素一个键盘焦点

摘要

```
void focus();
```

描述

这个方法把键盘焦点转移到这个 `<select>` 元素，以便用户可以使用键盘而不是鼠标和它交互。

Select.onChange

0 级 DOM

当改变选择时调用的事件句柄

摘要

Function onchange

描述

Select 对象的 onchange 属性引用一个事件句柄函数。当用户选中了一个选项，或者取消了对一个选项的选定时，就会调用这个句柄。这个事件不会指定新的选中选项是什么，必须通过 Select 对象的 selectedIndex 属性，或者各个 Option 对象的 selected 属性来确定这一点。

参阅 Element.addEventListener() 了解注册事件句柄的另一种方法。

参阅

Element.addEventListener()、Option, 第 17 章

Select.options[]

2 级 DOM HTML

Select 对象中的选项

摘要

readonly HTMLCollection options

描述

options[] 属性是一个类似数组的、Option 对象的 HTMLCollection。每个 Option 对象描述了 Select 对象中表示的一个选项。

options[] 属性并非一个普通的 HTMLCollection。为了和早期的浏览器向后兼容，这个集合有某种特殊的行为：允许通过 Select 对象来改变显示的选项：

- 如果把 options.length 属性设置为 0，Select 对象中的所有选项都会被清除。
- 如果 options.length 属性的值比当前值小，Select 对象的选项数就会减少，那么出现在选项数组尾部的元素就被舍弃。
- 如果把 options[] 数组中的一个元素设置为 null，那个选项就会从 Select 对象中删除，位于该元素之后的元素将被改变下标值以使它们的位置向前提，补上数组中的空缺（参阅 Select.remove()）。
- 如果用构造函数 Option() 创建了一个新的 Option 对象（参阅 Option），可以通过把这个新建的选项添加到 options[] 数组中，从而把那个选项添加到 Select 对象的选项列表的末尾。要实现这一点，只需要设置 options [options.length] 即可（参阅 Select.add()）。

参阅

Option

Select.remove()

2 级 DOM HTML

移除一个 <option>

摘要

```
void remove(long index);
```

参数

index

options 数组中要移除的 <option> 元素的位置。

描述

这个方法从选项数组的指定位置移除 <option> 元素。如果指定的下边比 0 小，或者大于或等于选项的数目，`remove()` 方法会忽略它并什么也不做。

参阅

Option

Style

参阅 CSS2Properties

Submit

参阅 Input

Table

2 级 DOM HTML

HTML 文档中的 <table>

Node → Element → HTMLElement → Table

属性

HTMLElement *caption*

对表的 <caption> 元素的引用，如果不存在该元素，则为 `null`。

readonly HTMLCollection *rows*

表示表中所有行的 TableRow 对象的数组 (HTMLCollection)。包括 <thead>、<tfoot> 和 <tbody> 中定义的所有行。

readonly HTMLCollection *tBodies*

表示表中的所有 <tbody> 段的 TableSection 对象的数组 (HTMLCollection)。

TableSection *tFoot*

表的 <tfoot> 元素，如果不存在该元素，则为 `null`。

TableSection tHead

表的 <thead> 元素，如果不存在该元素，则为 null。

除了前面列出的属性，这个接口还定义了下表中的属性，它们代表 <table> 元素的 HTML 属性：

属性	HTML 属性	描述
deprecated String align	align	表在文档中的水平对齐方式
deprecated String bgColor	bgcolor	表的背景颜色
String border	border	表的边线的宽度
String cellPadding	cellpadding	表元内容和边线之间的距离
String cellSpacing	cellspacing	表元边线之间的距离
String frame	frame	绘制哪种表边线
String rules	rules	在表中何处绘制行线
String summary	summary	表的概述
String width	width	表的宽度

方法

createCaption()

返回表现有的 <caption> 元素，如果不存在这样的元素，则创建（并插入）一个新的 <caption>。

createTFoot()

返回表现有的 <tfoot> 元素，如果不存在这样的元素，则创建（并插入）一个新的 <tfoot>。

createTHead()

返回表现有的 <thead> 元素，如果不存在这样的元素，则创建（并插入）一个新的 <thead>。

deleteCaption()

删除表的 <caption> 元素（如果表具有该元素）。

deleteRow()

删除表中指定位置的列。

deleteTFoot()

删除表的 <tfoot> 元素（如果表具有该元素）。

deleteTHead()

删除表的 <thead> 元素（如果表具有该元素）。

insertRow()

在表的指定位置插入一个新的空 <tr> 元素。

描述

Table 对象表示 HTML 的 <table> 元素，它为查询和修改表的各个部分定义了大量便捷的属性和方法。虽然这些方法和属性使得处理表更容易，但他们的功能和 DOM 的核心方法重复了。

参阅

TableCell、TableRow、TableSection

Table.createCaption()

2 级 DOM HTML

获取或创建 <caption> 元素

摘要

```
HTMLElement createCaption();
```

返回值

一个 HTMLElement 对象，表示该表的 <caption> 元素。如果该表已经有了标题，该方法只返回它。如果该表还没有 <caption> 元素，该方法将创建一个新的空 <caption> 元素，把它插入表中，并返回它。

Table.createTFoot()

2 级 DOM HTML

获取或创建 <tfoot> 元素

摘要

```
HTMLElement createTFoot();
```

返回值

一个 TableSection，表示该表的 <tfoot> 元素。如果该表已经有了脚注，该方法只返回它。如果该表还没有脚注，该方法将创建一个新的空 <tfoot> 元素，把它插入表中，并返回它。

Table.createTHead()

2 级 DOM HTML

获取或创建 <thead> 元素

摘要

```
HTMLElement createTHead();
```

返回值

一个 TableSection 对象，表示该表的 <thead> 元素。如果该表已经有了表头，该方法只返回它。如果该表还没有表头，该方法将创建一个新的空 <thead> 元素，把它插入表中，并返回它。

Table.deleteCaption()

2 级 DOM HTML

删除表的 <caption> 元素

摘要

```
void deleteCaption();
```

描述

如果该表有 <caption> 元素，这个方法将从文档树中删除它。否则，它什么也不做。

Table.deleteRow()

2 级 DOM HTML

删除表的一行

摘要

```
void deleteRow(long index)
    throws DOMException;
```

参数

index

指定了要删除的行在表中的位置。

抛出

如果 *index* 小于 0 或者大于等于表中行数，该方法将抛出代码为 INDEX_SIZE_ERR 的 DOMException 异常。

描述

该方法将删除表中指定位置的行。行的编码顺序就是它们在文档源代码中出现的顺序。<thead> 和 <tfoot> 中的行与表中其他行一起编码。

参阅

TableSection.deleteRow()

Table.deleteTFoot()

2 级 DOM HTML

删除表的 <tfoot> 元素

摘要

```
void deleteTFoot();
```

描述

如果该表有 <tfoot> 元素，这个方法将从文档树中删除它。否则，它什么也不做。

Table.deleteTHead()2 级 DOM HTML

删除表的 <thead> 元素

摘要

```
void deleteTHead();
```

描述

如果该表有 <thead> 元素，这个方法将从文档树中删除它。否则，它什么也不做。

Table.insertRow()2 级 DOM HTML

给表添加新的空行

摘要

```
HTMLElement insertRow(long index)  
    throws DOMException;
```

参数

index

要插入新行的位置。

返回值

一个 TableRow，表示新插入的行。

抛出

如果 *index* 小于 0 或者大于等于表中行数，该方法将抛出代码为 INDEX_SIZE_ERR 的 DOMException 异常。

描述

该方法将创建一个新的 TableRow 对象，表示一个 <tr> 标记，并把它插入表中的指定位置。新的行将被插入同一段并且就在 *index* 指定的位置所在的行之前。如果 *index* 等于表中的行数，新行将被附加到表的末尾。如果表初始时是空的，新行将被插入到一个新的 <tbody> 段，该段自身会被插入表中。

可以用 TableRow.insertCell() 方法给新创建的行添加内容。

参阅

TableSection.insertRow()

TableCell

2 级 DOM HTML

HTML 表中的 <td> 或 <th> 表元

Node → Element → HTMLElement → TableCell

属性

readonly long cellIndex

表元在它的行中的位置。

除了 cellIndex 属性, 该接口还定义了下表列出的属性, 它们直接对应于 <td> 和 <th> 元素的 HTML 属性。

属性	HTML 属性	描述
String abbr	abbr	参阅 HTML 标准
String align	align	表元的水平对齐方式
String axis	axis	参阅 HTML 标准
deprecated String bgColor	bgcolor	表元的背景颜色
String ch	char	对齐字符
String chOff	choff	对齐字符偏移量
long colSpan	colspan	表元跨越的列数
String headers	headers	表元头的 id 值
deprecated String height	height	表元的高度, 以像素计
deprecated boolean noWrap	nowrap	表元不自动换行
long rowSpan	rowspan	表元跨越的行数
String scope	scope	头表元的作用域
String vAlign	valign	表元的垂直对齐方式
deprecated String width	width	表元的宽度, 以像素计

描述

该接口表示 HTML 表中的 <td> 和 <th> 元素。

TableRow

2 级 DOM HTML

HTML 表中的 <tr> 元素

Node → Element → HTMLElement → TableRow

属性

readonly HTMLCollection cells

TableCell 对象的数组 (HTMLCollection 对象), 表示该行中的所有表元。

readonly long rowIndex

该行在表中的位置。

readonly long sectionRowIndex

该行在它的段（即 <thead>、<tbody> 或 <tfoot> 元素）中的位置。

除了上面列出的属性，该接口还定义了下表列出的属性，它们直接对应于 <tr> 元素的 HTML 属性。

属性	HTML 属性	描述
String align	align	表元在行中的默认水平对齐方式
deprecated String bgColor	bgcolor	行的背景颜色
String ch	char	表元在行中的对齐字符
String chOff	choff	表元在行中的对齐字符偏移量
String vAlign	valign	表元在行中的默认垂直对齐方式

方法

deleteCell()

删除行中指定的表元。

insertCell()

在 HTML 表的一行的指定位置插入一个空的 <td> 元素。

描述

该接口表示 HTML 表中的一行。

TableRow.deleteCell()	2 级 DOM HTML
-----------------------	--------------

删除表行中的一个表元

摘要

void deleteCell(long index)
throws DOMException;

参数

index
要删除的表元在行中的位置。

抛出

如果 index 小于 0 或者大于等于表中行的表元数，该方法将抛出代码为 INDEX_SIZE_ERR 的 DOMException 异常。

描述

该方法将删除表行中指定位置的表元。

TableRow.insertCell()2 级 DOM HTML

在表行中插入一个新的空 `<td>` 元素

摘要

```
HTMLElement insertCell(long index)
    throws DOMException;
```

参数

index

插入新表元的位置。

返回值

一个 `TableCell` 对象，表示新创建并被插入的 `<td>` 元素。

抛出

如果 *index* 小于 0 或者大于等于表中行的表元数，该方法将抛出代码为 `INDEX_SIZE_ERR` 的 `DOMException` 异常。

描述

该方法将创建一个新的 `<td>` 元素，把它插入行中指定的位置。新表元将被插入当前位于 *index* 指定位置的表元之前。如果 *index* 等于行中的表元数，新表元将被附加在行的末尾。注意，这种方法只能插入 `<td>` 数据表元。如果需要给行添加头表元，必须用 `Document.createElement()` 方法和 `Node.insertBefore()` 方法（或相关的方法）创建并插入一个 `<th>` 元素。

TableSection2 级 DOM HTML

表的头、脚注和主体段

Node → Element → HTMLElement → TableSection

属性

readonly HTMLCollection rows

`TableRow` 对象的数组（`HTMLCollection` 对象），表示表中该段中的所有行。

除了上面列出的 `rows` 属性，该接口还定义了下表列出的属性，它们直接对应于基础 HTML 元素的属性。

属性	HTML 属性	描述
String align	align	表元在段中的默认水平对齐方式
String ch	char	表元在段中的对齐字符
String chOff	choff	表元在段中的默认字符偏移量
String vAlign	valign	表元在段中的默认垂直对齐方式

方法

`deleteRow()`

删除段中指定位置的行。

`insertRow()`

在段的指定位置插入一个空行。

属性

该接口表示 HTML 表中的 `<tbody>`、`<thead>` 或 `<tfoot>` 段。Table 的 `tHead` 和 `tFoot` 属性都是 `TableSection` 对象，而 `tBodies` 属性是 `TableSection` 对象的一个数组。

TableSection.deleteRow()

2 级 DOM HTML

删除表段的一行

摘要

```
void deleteRow(long index)
    throws DOMException;
```

参数

index

指定了行在段中的位置。

抛出

如果 *index* 小于 0 或者大于等于段中行数，该方法将抛出代码为 `INDEX_SIZE_ERR` 的 `DOMException` 异常。

描述

该方法将删除段中指定位置的行。注意，*index* 指定的是行在段中的位置，不是行在整个表中的位置。

参阅

`Table.deleteRow()`

TableSection.insertRow()

2 级 DOM HTML

给表段添加新的空行

摘要

```
HTMLElement insertRow(long index)
    throws DOMException;
```

参数

index

要插入的新行在段中的位置。

返回值

一个 `TableRow` 对象，表示新创建并插入的 `<tr>` 元素。

抛出

如果 *index* 小于 0 或者大于等于段中行数，该方法将抛出代码为 `INDEX_SIZE_ERR` 的 `DOMException` 异常。

描述

该方法将创建一个新的 `<tr>` 元素，并把它插入段中的指定位置。如果 *index* 等于段中的行数，新行将被附加到段的末尾。否则新行将被插入当前位于 *index* 指定位置的行之前。注意，在该方法中，*index* 指定的是行在段中的位置，不是它在整个表中的位置。

参阅

`Table.insertRow()`

Text

1 级核心 DOM

HTML 或 XML 文档中的一系列文本

`Node` → `CharacterData` → `Text`

子接口

`CDATASection`

方法

`splitText()`

在指定的字符位置把一个 `Text` 节点分割成两个，并返回新的 `Text` 节点。

描述

`Text` 节点表示 HTML 或 XML 文档中的一系列纯文本。因为纯文本出现在 HTML 和 XML 的元素和属性中，所以 `Text` 节点通常作为 `Element` 节点和 `Attr` 节点的子节点出现。`Text` 节点继承了 `CharacterData` 接口，通过从 `CharacterData` 接口继承的 `data` 属性或从 `Node` 接口继承的 `nodeValue` 属性，可以访问 `Text` 节点的文本内容。用从 `CharacterData` 继承的方法或 `Text` 接口自身定义的 `splitText()` 方法可以操作 `Text` 节点。使用 `Document.createTextNode()` 来创建一个新的 `Text` 节点。`Text` 节点没有子节点。

关于从文档的子树中删除空 `Text` 节点与合并相邻的 `Text` 节点的方法，请参阅“`Node.normalize()`”参考页。

参阅

CharacterData、Node.normalize()

Text.splitText()

1 级核心 DOM

把一个 Text 节点分割成两个

摘要

```
Text splitText(unsigned long offset)
    throws DOMException;
```

参数

offset

分割 Text 节点的字符位置。

返回值

从当前节点分割出的 Text 节点。

抛出

该方法将抛出具有下列代码之一的 DOMException 异常：

INDEX_SIZE_ERR

Offset 是负数，或者大于 Text 或 Comment 节点的长度。

NO_MODIFICATION_ALLOWED_ERR

该节点是只读的，不能修改。

描述

该方法将在指定的 *offset* 处把 Text 节点分割成两个节点。原始的 Text 节点将被修改，使它包含 *offset* 指定的位置之前的文本内容（但不包括该文本内容）。新的 Text 节点将被创建，用于存放从 *offset* 位置（包括该位置上的字符）到原字符串结尾的所有字符。新的 Text 节点是该方法的返回值。此外，如果原始的 Text 节点具有 *parentNode*，新的 Text 节点将插入这个父节点，紧邻在原始节点之后。

CDATASection 接口继承了 Text 接口，CDATASection 节点也可以使用该方法，只是新建的节点是 CDATASection 节点，而不是 Text 节点。

参阅

Node.normalize()

Textarea

2 级 DOM HTML

一个多行文本输入区域

Node → Element → HTMLElement → Textarea

属性

String defaultValue

文本框的初始内容。在表单被重置时，文本框将被恢复为这个值。改变这个属性的值会改变文本框当前显示的文本。

readonly Form form

表示包含这个 Textarea 的 <form> 元素的 Form 对象，如果这个元素不在一个表单中，则为 null。

readonly String type

元素类型，以便和 Input 对象兼容。该属性的值总是“textarea”。

String value

当前显示在文本区域的文本。它的初始值与属性 defaultValue 的值相同。当用户在 Textarea 对象中输入字符时，value 属性的值就会随着用户的输入而更新。如果明确地设置了 value 属性，Textarea 对象就会显示指定的字符串。value 属性存放的字符串就是在提交表单时要发送给服务器的数据。

除了这些属性，Textarea 对象还映射了如下的 HTML 属性：

属性	HTML 属性	描述
String accessKey	accesskey	键盘快捷字符
long cols	cols	字符列的宽度
boolean disabled	disabled	Textarea 是否关闭
String name	name	用于表单提交和元素访问的 Textarea 名称
boolean readOnly	readonly	Textarea 是否可编辑
long rows	rows	行中 Textarea 的高度
long tabIndex	tabindex	Textarea 在切换顺序中的位置

方法

blur()

把键盘焦点从该元素上移开。

focus()

把键盘焦点转移到该元素。

select()

选中文本框的完整内容。

事件句柄

onchange

在用户编辑了这个元素中显示的文本，并且把键盘焦点移到了别处时调用的事件句柄。并非在 Textarea 元素中的每一次击键都会调用 onchange 句柄，只有当用户完成了一次编辑之后，才会调用这一句柄。

HTML 语法

Textarea 元素是由标准的 HTML 标记 <textarea> 和 </textarea> 创建的：

```
<form>
  ...
  <textarea
    [ name="name" ]           // A name that can be used to refer to this element
    [ rows="integer" ]       // How many lines tall the element is
    [ cols="integer" ]       // How many characters wide the element is
    [ onchange="handler" ]   // The onchange() event handler
  >
    plain_text                // The initial text; specifies defaultValue
  </textarea>
  ...
</form>
```

描述

Textarea 对象表示一个 HTML <textarea> 元素，它用于创建多行输入的文本框（通常在 HTML 表单中）。该文本框的初始内容在标记 <textarea> 和 </textarea> 之间设置。用户可以用 value 属性查询和设置文本。

Textarea 是一个像 Input 和 Select 一样的表单输入元素。和这些对象一样，它定义了 form、name 和 type 属性。

参阅

Form、Input，第 18 章

Textarea.blur()

2 级 DOM HTML

把键盘焦点从该元素中移开

摘要

```
void blur();
```

描述

该方法将把键盘焦点从该元素中移开。

Textarea.focus()

2 级 DOM HTML

给予该元素键盘焦点

摘要

```
void focus();
```

描述

该方法将把键盘焦点转移到当前元素，使用户可以不必单击它，就能编辑显示的文本。

Textarea.onchange

0 级 DOM

当输入值改变时候调用的事件句柄

摘要

```
Function onchange
```

描述

Textarea 的 onchange 属性引用一个事件句柄函数，当用户改变文本区域的值然后通过把键盘焦点移动到其他地方“确认”这些改变的时候，调用这个句柄。

注意，当一个 Text 对象的 value 属性通过 JavaScript 设置的时候，onchange 事件句柄不会调用。还要注意，这个句柄是为了处理对输入值的一次完整改变，因此，它不会在逐个按键的基础上调用。参阅 HTMLInputElement.onkeypress 来了解有关接受每个按键事件通知的内容，参阅 Element.addEventListener() 了解事件注册的另一种方法。

参阅

Element.addEventListener()、HTMLInputElement.onkeypress、Input.onchange, 第 17 章

Textarea.select()

2 级 DOM HTML

选择这个元素中的文本

摘要

```
void select();
```

描述

该方法将选中 <textarea> 元素显示的所有文本。在大多数浏览器中，这意味着文本将高亮显示，用户输入的新文本将替换高亮的文本，而不是附加到其后。

TextField

参阅 Input

UIEvent

2 级 DOM Events

用户界面事件的详细情况

Event → UIEvent

子接口

KeyEvent, MouseEvent

属性

readonly long detail

关于该事件的数字细节。对于 click、mousedown、mouseup 事件来说（参阅“MouseEvent”），这个字段填写单击的次数，1 表示单击，2 表示双击，3 表示击键三次，以此类推。对于 DOMActivate 事件，该字段为 1 表示普通激活，2 表示“超级激活”，如双击或 **Shift-Enter** 键的组合。

readonly Window view

生成事件的窗口（视图）。

方法

initUIEvent()

初始化新创建的 UIEvent 对象的属性，包括由 Event 接口继承的属性。

描述

UIEvent 接口是 Event 接口的子接口，定义了传递给 DOMFocusIn、DOMFocusOut 和 DOMActivate 类型事件的 Event 对象的类型。在 web 浏览器中，不常使用这些事件类型，关于 UIEvent 接口要记住的重要一点是，它是 MouseEvent 接口的父接口。

参阅

Event、KeyEvent、MouseEvent，第 17 章

UIEvent.initUIEvent()

2 级 DOM Events

初始化 UIEvent 对象的属性

摘要

```
void initUIEvent(String typeArg,  
                 boolean canBubbleArg,  
                 boolean cancelableArg,  
                 Window viewArg,  
                 long detailArg);
```

参数

typeArg

事件类型。

canBubbleArg

事件是否可以起泡。

cancelableArg

是否可以用 `preventDefault()` 方法取消该事件。

viewArg

发生事件的窗口。

detailArg

事件的 `detail` 属性。

描述

该方法将初始化 `UIEvent` 对象的 `view` 属性和 `detail` 属性以及从 `Event` 接口继承的 `type` 属性、`bubbles` 属性、`cancelable` 属性。只有新创建的 `UIEvent` 对象才能调用该方法，而且必须在把它传递给 `Element.dispatchEvent()` 方法之前调用。

参阅

`Document.createEvent()`、`Event.initEvent()`、`MouseEvent.initMouseEvent()`

Window

JavaScript 1.0

Web 浏览器窗口或帧

Object → Global → Window

摘要

`self`

`window`

`window.frames[i]`

属性

`Window` 对象定义了如下的属性，并且还继承了核心 JavaScript 的所有全局属性（参阅本书第三部分的 `Global`）：

`closed`

一个只读的布尔值，声明了窗口是否已经关闭。当浏览器窗口关闭时，表示该窗口的 `Window` 对象并不会消失，它将继续存在，不过它的 `closed` 属性设置为 `true`。

`defaultStatus`

一个可读可写的字符串，声明了显示在状态栏中的默认消息。参阅 `Window.defaultStatus` 参考页。

document

对描述窗口或帧中含有的文档的 Document 对象的只读引用。详见 Document 对象。

event[仅用于 IE]

在 IE 中, 这个属性引用描述最近事件的 Event 对象。这个属性用于 IE 事件模型中。在标准 DOM 事件模型中, Event 对象作为一个参数传递给事件句柄函数。参阅 Event 和第 17 章了解详细信息。

frames[]

Window 对象的数组, 每个 Window 对象在窗口中含有的一个帧或 <iframe>。属性 frames.length 存放数组 frames[] 中含有的元素个数。注意, frames[] 数组中引用的帧自身可能还含有帧, 它们自己也具有 frames[] 数组。

history

对窗口或帧的 History 对象的只读引用, 详见 History 对象。

innerHeight, innerWidth

只读属性, 声明了窗口的文档显示区的高度和宽度, 以像素计。这里宽度和高度不包括菜单栏、工具栏以及滚动条等的高度。IE 不支持这些属性。它用 document.documentElement 或 document.body (和 IE 的版本相关) 的 clientWidth 和 clientHeight 属性作为替代。参阅第 14.3.1 节了解详细内容。

location

用于窗口或帧的 Location 对象。该对象声明了当前装载进来的文档的 URL。把这个属性赋予一个新的 URL 字符串, 会引发浏览器装载并显示出那个 URL 所指的文档。要进一步了解该属性, 请参阅 Location 对象。

name

一个字符串, 存放了窗口的名称。这个名称是在 open() 方法创建窗口时指定的或者使用一个 <frame> 标记的 name 属性指定的。窗口的名称可以用作一个 <a> 或 <form> 标记的 target 属性的值。以这种方式使用 target 属性声明了超链接文档或表单提交结果应该显示于指定的窗口或帧中。

navigator

对 Navigator 对象的只读引用, 提供 web 浏览器的版本信息和配置信息。详见 Navigator 对象。

opener

一个可读可写的属性, 是对一个 Window 对象的引用, 该对象含有调用了 open() 方法的脚本以打开浏览器窗口的脚本。只有表示顶层窗口的 Window 对象的 opener 属性才有效, 表示帧的 Window 对象的 opener 属性无效。属性 opener 非常有用, 以致新创建的窗口可以引用创建它的窗口所定义的属性和函数。

outerHeight, outerWidth

只读的整数, 声明了整个窗口的高度和宽度, 以像素计。它们包括菜单栏、工具栏、滚动条和窗口边线等的高度和宽度, 等等。IE 不支持这些属性, 并且没有提供替代的属性。

`pageXOffset`, `pageYOffset`

只读的整数，声明了当前文档向右 (`pageXOffset`) 和向下 (`pageYOffset`) 滚动过的像素数。IE 不支持这些属性。IE 使用 `document.documentElement` 或 `document.body` (和 IE 的版本相关) 的 `scrollLeft` 和 `scrollTop` 属性。参阅第 3.1 节了解详细内容。

`parent`

对一个 Window 对象的只读引用，这个 Window 对象包含当前的窗口或帧。如果 window 是顶层窗口，那么 `parent` 引用的就是窗口自身。如果 window 是帧，那么 `parent` 引用的是包含 window 的窗口或帧。

`screen`

对一个 Screen 对象的只读引用，这个 Screen 声明了与屏幕有关的信息，即可用的像素数和可用的颜色数。详见 Screen 对象。

`screenLeft`, `screenTop`, `screenX`, `screenY`

只读整数，声明了窗口的左上角在屏幕上的 X 坐标和 Y 坐标。IE、Safari 和 Opera 支持 `screenLeft` 和 `screenTop`，而 Firefox 和 Safari 支持 `screenX` 和 `screenY`。

`self`

对窗口自身的只读引用。等价于 `window` 属性。

`status`

一个可读可写的字符串，声明了浏览器状态栏的当前内容。详见“`Window.status`”参考页。

`top`

对一个顶级窗口的只读引用，顶级窗口包含了这个窗口。如果窗口自身就是一个顶级窗口，`top` 属性存放了对窗口自身的引用。如果窗口是一个帧，那么 `top` 属性引用包含帧的顶层窗口。和 `parent` 属性相对。

`window`

`window` 属性等价于 `self` 属性，它包含了对窗口自身的引用。

方法

Window 对象定义了如下的方法，并且还继承了核心 JavaScript 定义的所有全局函数 (参阅第三部分的 Global)。

`addEventListener()`

向这个窗口的事件句柄集合中添加一个事件句柄函数。除 IE 以外的现代浏览器都支持这个方法。参阅用于 IE 的 `attachEvent()`。

`alert()`

在对话框中显示简单的消息。

`attachEvent()`

向这个窗口的句柄集合中添加一个事件句柄函数。这是 `addEventListener()` 的一个特定于 IE 的替代。

`blur()`

把键盘焦点从顶层浏览器窗口中移走。

`clearInterval()`

取消周期性执行的代码。

`clearTimeout()`

取消挂起超时操作。

`close()`

关闭窗口。

`confirm()`

用对话框询问一个回答为是或否的问题。

`detachEvent()`

从这个窗口移除一个事件句柄函数。这是 `removeEventListener()` 的一个特定于 IE 的替代。

`focus()`

把键盘焦点给予顶层浏览器窗口。在大多数平台上，这将使窗口移到最前边。

`getComputedStyle()`

确定应用到一个文档元素的 CSS 样式。

`moveBy()`

把窗口移动一个相对的数量。

`moveTo()`

把窗口移动到一个绝对的位置。

`open()`

创建并打开一个新窗口。

`print()`

模拟对浏览器的 **Print** 按钮的单击。只有 IE 5 和 Netscape 4 支持该方法。

`prompt()`

用对话框请求输入一个简单的字符串。

`removeEventListener()`

从这个窗口的句柄集合中移除一个事件句柄函数。除 IE 以外的所有现代浏览器都实现了这个方法。IE 提供了 `detachEvent()` 作为替代。

`resizeBy()`

把窗口大小调整指定的数量。

`resizeTo()`

把窗口大小调整到指定的大小。

`scrollBy`

让窗口滚动指定的数量。

`scrollTo()`

把窗口滚动到指定的位置。

`setInterval()`

周期性执行指定的代码。

`setTimeout()`

在经过指定的时间之后执行代码。

事件句柄

`onblur`

当窗口失去焦点时调用的事件句柄。

`onerror`

当发生 JavaScript 错误时调用的事件句柄。

`onfocus`

当窗口获得焦点时调用的事件句柄。

`onload`

当文档（或帧集）完全被装载进来时调用的事件句柄。

`onresize`

当调整窗口大小时调用的事件句柄。

`onunload`

当浏览器离开当前文档或帧集时调用的事件句柄。

描述

Window 对象表示一个浏览器窗口或一个帧。我们在第 14 章中对它进行了详细的说明。在客户端 JavaScript 中，Window 对象是全局对象，所有的表达式都在当前 Window 对象的环境中计算。也就是说，要引用当前窗口根本不需要特殊的语法，可以把那个窗口的属性作为全局变量来使用。例如，可以只写 `document`，而不必写为 `window.document`。同样，可以把当前窗口对象的方法当作函数来使用，如只写 `alert()`，而不必写 `window.alert()`。除了这里列出的属性和方法，Window 对象还实现了核心 JavaScript 所定义的所有全局属性和方法。参阅本书第三部分的 Global。

Window 对象的 `window` 属性和 `self` 属性引用的都是它自己。当你想明确地引用当前窗口，而不仅仅是隐式地引用它时，可以使用这两个属性。除了这两个属性之外，`parent` 属性、`top` 属性以及 `frames[]` 数组都引用了与当前 Window 对象相关的其他 Window 对象。

要引用窗口中的一个帧，可以使用如下的语法：

```
frames[i]           // Frames of current window
self.frames[i]      // Frames of current window
w.frames[i]         // Frames of specified window w
```

要引用一个帧的父窗口（或父帧），可以使用下面的语法：


```
parent      // Parent of current window
self.parent // Parent of current window
w.parent     // Parent of specified window w
```

要从顶层浏览器窗口含有的任何一个帧中引用它，可以使用如下的语法：

```
top          // Top window of current frame
self.top     // Top window of current frame
f.top        // Top window of specified frame f
```

新的顶层浏览器窗口由方法 `Window.open()` 创建。当调用这个方法时，应该把 `open()` 调用的返回值存储在一个变量中，然后使用那个变量来引用新窗口。新窗口的 `opener` 属性反过来引用了打开它的那个窗口。

一般说来，`Window` 对象的方法都是对浏览器窗口或帧进行某种操作。而 `alert()` 方法、`confirm()` 方法和 `prompt()` 方法则不同，它们通过简单的对话框与用户进行交互。

要深入地了解 `Window` 对象，请参阅第 14 章对 `Window` 对象的介绍，以及各个 `Window` 方法和事件句柄的参考页。

参阅

`Document`，本书第三部分的 `Global`，第 14 章

`Window.addEventListener()`

参阅 `Element.addEventListener()`

`Window.alert()`

JavaScript 1.0

在对话框中显示一条消息

摘要

```
window.alert(message)
```

参数

message

要在 `Window` 上弹出的对话框中显示的纯文本（而非 HTML 文本）。

描述

方法 `alert()` 将在对话框中把指定的 *message* 显示给用户。该对话框含有 **OK** 按钮，用户可以单击该按钮来关闭对话框。对话框通常是有模式的，在用户关闭该对话框之前，`alert()` 的调用将暂停执行。

习惯用法

在用户给某些表单元素输入了无效信息时会显示出错消息，这也许是 `alert()` 方法最常见的用法了。警报对话框可以把问题告诉用户，并且提示用户应该修正什么才能避免这一错误。

`alert()` 方法对话框的外观由平台决定, 不过通常它都具有表示错误、警告和某种警告信息的图形。尽管 `alert()` 可以显示任何消息, 但是对话框的警告图形却意味着该方法不适合显示简单的信息性消息, 诸如“欢迎访问我的 blog”和“你是本周的第 177 位访问者!”等。

注意, 对话框中显示的 *message* 是纯文本串, 而不是 HTML 格式的字符串。可以在这个字符串中使用换行符“\n”, 将消息放在多行显示。也可以使用空格来实现一些基本的格式化, 还可以使用带下划线的字符来模拟水平标尺, 不过所能达到的最终效果很大程度上依赖于对话框使用的字体, 因此它是依赖于系统的。

参阅

`Window.confirm()`、`Window.prompt()`

`Window.attachEvent()`

参阅 `Element.attachEvent()`

`Window.blur()`

JavaScript 1.1

把键盘焦点从顶层窗口移开

摘要

`window.blur()`

描述

方法 `blur()` 将把键盘焦点从顶层浏览器窗口中移走, 这个窗口由 `Window` 对象指定。哪个窗口最终获得键盘焦点并没有指定。在某些浏览器或平台上, 这个方法可能无效。

参阅

`Window.focus()`

`Window.clearInterval()`

JavaScript 1.2

停止周期性地执行代码

摘要

`window.clearInterval(intervalId)`

参数

intervalId

调用 `setInterval()` 方法返回的值。

描述

方法 `clearInterval()` 将停止周期性地执行指定的代码, 对这些代码的操作是调用方法 `setInterval()` 启动的。参数 *intervalId* 必须是调用方法 `setInterval()` 后的返回值。

参阅

`Window.setInterval()`

`Window.clearTimeout()`

JavaScript 1.0

取消对指定的代码的延迟执行

摘要

`window.clearTimeout(timeoutId)`

参数

timeoutId

方法 `setTimeout()` 返回的值，标识了要取消的延迟执行代码块。

描述

方法 `clearTimeout()` 取消对指定代码的执行，调用方法 `setTimeout()` 可以延迟执行这些代码。参数 *timeoutId* 是调用方法 `setTimeout()` 后的返回值，它标识了要取消的延期执行代码块（可以有多个）。

参阅

`Window.setTimeout()`

`Window.close()`

JavaScript 1.0

关闭浏览器窗口

摘要

`window.close()`

描述

方法 `close()` 将关闭由 *window* 指定的顶层浏览器窗口。一个窗口可以通过调用 `self.close()` 或只调用 `close()` 来关闭它自身。只有通过 JavaScript 代码打开的窗口才能够由 JavaScript 代码关闭。这阻止了恶意的脚本终止用户的浏览器。

参阅

`Window.open()`、`Window` 对象的 `closed` 属性和 `opener` 属性

`Window.confirm()`

JavaScript 1.0

询问一个回答为是或否的问题

摘要

`window.confirm(question)`

参数

question

要在对话框中显示的纯文本（而不是 HTML 格式的）。通常，它应该表达你想让用户回答的问题。

返回值

如果用户单击了 **OK** 按钮，返回值为 `true`；如果用户单击了 **Cancel** 按钮，返回值为 `false`。

描述

方法 `confirm()` 将在一个对话框中显示指定的问题 *question*。该对话框含有 **OK** 按钮和 **Cancel** 按钮，用户可以使用这两个按钮来回答问题。如果用户单击了 **OK** 按钮，那么 `confirm()` 将返回 `true`。如果用户单击了 **Cancel** 按钮，`confirm()` 将返回 `false`。

由 `confirm()` 方法显示的对话框是有模式的，也就是说，在用户单击 **OK** 按钮或 **Cancel** 按钮把对话框关闭之前，它将阻塞用户对浏览器窗口的所有输入。由于该方法返回的值由用户的响应决定，所以在调用 `confirm()` 时，将暂停对 JavaScript 代码的执行，在用户作出响应之前，不会执行下一条语句。

另外，对话框按钮的标签是不可改变的（例如想让它们显示为 **Yes** 和 **No**）。因此，应该小心地编写问题或消息，让它适合于用 **OK** 或 **Cancel** 来回答。

参阅

`Window.alert()`、`Window.prompt()`

Window.defaultStatus

JavaScript 1.0

默认的状态栏文本

摘要

`window.defaultStatus`

描述

属性 `defaultStatus` 是一个可读可写的字符串，声明了要在窗口的状态栏中显示的默认文本。web 浏览器通常用状态栏在装载文件的过程中显示浏览器的进度，或者在鼠标移到超文本链接时显示链接地址。在不显示这些瞬时消息时，默认情况下，状态栏是空白的。不过，可以通过设置 `defaultStatus`，指定在不使用状态栏时要显示的默认消息，也可以通过读取 `defaultStatus` 属性来获得默认的消息。虽然指定的文本可能会暂时被其他消息覆盖，如在鼠标移到超文本链接上时要显示的消息，但是在擦掉了瞬时消息后，会再次显示 `defaultStatus` 中的消息。

在某些现代浏览器中, `defaultStatus` 属性已经关闭。参阅 `Window.status` 了解详细信息。

参阅

`Window.status`

Window.detachEvent()

参阅 `Element.detachEvent()`

Window.focus()

JavaScript 1.1

把键盘焦点给予一个窗口

摘要

`window.focus()`

描述

方法 `focus()` 将把键盘焦点给予 `Window` 对象指定的浏览器窗口。

在大多数平台上, 得到了键盘焦点的顶层窗口都会被提到窗口堆栈的顶部, 以便它获得焦点后变得可见。

参阅

`Window.blur()`

Window.getComputedStyle()

2 级 DOM CSS

获取用来绘制一个元素的 CSS 样式

摘要

```
CSS2Properties getComputedStyle(Element elt,  
                                String pseudoElt);
```

参数

elt

需要的样式信息所属的文档元素。

pseudoElt

CSS 伪元素串, 例如 “:before” 或 “:first-line”, 如果没有则为 `null`。

返回值

一个只读的 `CSS2Properties` 对象, 表示用来在窗口中绘制指定的元素的样式属性和值。通过这一属性查询的任何长度值总是以像素或绝对数值表示, 而不会是相对数值或百分比数值。

描述

文档中的一个元素可以通过一个内联的 `style` 属性以及从样式表“级联”的任意数目的样式表中获取样式信息。在元素实际显示之前，它的样式信息必须从级联中提取出来，并且，用相对单位（如百分比或“ems”）指定的样式必须经过“计算”转换为绝对单位。

这个方法返回一个只读的 `CSS2Properties` 对象，它表示那些级联的和计算过的样式。DOM 规范要求任何表示长度的样式都要使用英寸或毫米这样的绝对单位。实际上，通常返回像素值，尽管还无法保证一个实现总是这样做。

把 `getComputedStyle()` 和 `HTMLElement` 的 `style` 属性进行对比：后者只允许访问一个元素的内联样式，用你所指定的任何单位都行，并且，不能告诉你应用到元素样式表样式的任何信息。

在 IE 中，类似的功能可以通过每个 `HTMLElement` 对象的 `currentStyle` 属性来实现。

参阅

`CSS2Properties`、`HTMLElement`

Window.moveBy()

JavaScript 1.2

把窗口移动一个相对的位置

摘要

`window.moveBy(dx, dy)`

参数

dx 要把窗口右移的像素数。

dy 要把窗口下移的像素数。

描述

方法 `moveBy()` 将把窗口 *window* 向右和向下分别移动 *dx* 和 *dy* 个像素。安全性和易用性方面的考虑在 `Window.moveTo()` 中介绍。

Window.moveTo()

JavaScript 1.2

把窗口移动到一个绝对位置

摘要

`window.moveTo(x, y)`

参数

x 窗口新位置的 X 坐标。

y 窗口新位置的 Y 坐标。

描述

方法 `moveTo()` 将移动窗口 `window`，使它的左上角处于 `x` 和 `y` 指定的位置处。出于安全方面的原因，浏览器限制方法使之不能把窗口移出屏幕。移动用户的浏览器窗口通常是个坏主意，除非用户明确地要求这么做。脚本通常只应该在他们自己通过 `Window.open()` 创建的窗口之上使用这个方法。

Window.onblur

JavaScript 1.1

当窗口失去键盘焦点时调用的事件句柄

摘要

Function onblur

描述

Window 对象的属性 `onblur` 声明了一个事件句柄函数，这个函数是在用户把键盘焦点从该窗口中移开时调用的。

这个属性的初始值是一个含有 JavaScript 语句的函数，其中的 JavaScript 语句由标记 `<body>` 或 `<frameset>` 的 `onblur` 性质指定的，各个语句之间用分号分隔。

习惯用法

如果网页具有动画效果，可以使用 `onblur` 事件句柄在窗口失去输入焦点时停止动画。从理论上说，如果窗口没有了焦点，那么用户就看不到它了，或者不再注意它了。

参阅

`Window.blur()`、`Window.focus()`、`Window.onfocus`，第 17 章

Window.onerror

JavaScript 1.1

发生 JavaScript 错误时调用的出错处理程序

摘要

可以使用如下代码注册一个 `onerror` 事件句柄：

```
window.onerror=handler-func
```

浏览器调用这个句柄的代码如下：

```
window.onerror(message, url, line)
```

参数

`message`

一个字符串，声明了出现的错误的消息。

url

一个字符串，声明了出现错误的文档的 URL。

line

一个数字，声明了出现错误的代码行的行号。

返回值

如果处理程序已经解决了发生的问题，JavaScript 代码不必再作进一步处理，那么返回值是 `true`。如果 JavaScript 程序还要显示出默认的出错消息对话框，那么返回值为 `false`。

描述

Window 对象的 `onerror` 属性声明了一个错误句柄。当出现了一个 JavaScript 错误并且没有被一个 `catch` 语句捕获时，就会调用这一错误句柄。你可以通过提供自己的 `onerror` 事件句柄函数来定制错误处理。

要定义一个 `onerror` 事件句柄函数，需要把 Window 对象的 `onerror` 属性设置成一个适当的函数。注意，`onerror` 是与其他事件句柄不同的一个错误句柄。特别是，一个错误句柄无法使用 `<body>` 标记的 `onerror` 属性来定义。

在调用 `onerror` 处理程序时，传递给它的参数有三个。第一个是声明错误消息的字符串，第二个是声明发生错误的文档的 URL 的字符串，第三个是发生错误的代码行的行号。有了这三个参数，错误句柄就能够完成任何想要执行的操作。例如，可以显示它自己的错误对话框，或以某种方式把这个错误记入日志等。当错误句柄执行完毕时，如果它已经完全解决了问题，不再需要浏览器执行进一步的操作了，就返回 `true`。如果它只是以某种方式提示或者记录了错误，还需要浏览器在它的默认对话框中显示出错消息，那么返回的就是 `false`。

Window.onfocus

JavaScript 1.1

在窗口得到焦点时调用的事件处理程序

摘要

Function onfocus

描述

Window 对象的属性 `onfocus` 声明了一个事件句柄函数，这个函数是在窗口得到了键盘焦点的时候调用的。

这个属性的初始值是一个含有 JavaScript 语句的函数，其中的 JavaScript 语句由标记 `<body>` 或 `<frameset>` 的 `onfocus` 性质指定，各个语句之间用分号分隔。

习惯用法

如果网页具有动画效果, 可以使用 `onfocus()` 事件句柄启动动画, 使用 `onblur` 处理程序停止它, 这样就可以在用户注意窗口时才运行动画。

参阅

`Window.blur()`、`Window.focus()`、`Window.onblur`, 第 17 章

Window.onload

JavaScript 1.0

结束文档装载时调用的事件处理程序

摘要

Function onload

描述

Window 对象的属性 `onload` 声明了一个事件句柄函数, 这个函数是在结束了文档 (或帧集) 装载时调用的。

这个属性的初始值是一个含有 JavaScript 语句的函数, 其中的 JavaScript 语句由标记 `<body>` 或 `<frameset>` 的 `onload` 性质指定, 各个语句之间用分号分隔。

一旦调用了 `onload` 事件句柄, 就可以确定文档已经完全装载进来了, 也就是说, 文档中所有的脚本都已经执行完毕, 脚本中的所有函数都已经被定义过, 脚本中的表单和其他文档元素也都已经解析过, 通过 `Document` 对象就可以访问它们。

可以使用 `Window.addEventListener()` 或 `Window.attachEvent()` 来为 `onload` 事件注册多个事件句柄函数。

参阅

`Window.onunload`, 第 13.5.7 节, 例 17-7, 第 17 章

Window.onresize

JavaScript 1.2

在调整窗口大小时调用的事件句柄

摘要

Function onresize

描述

Window 对象的属性 `onresize` 声明了一个事件句柄函数, 这个函数是在用户改变窗口或帧的大小时调用的。

这个属性的初始值是一个含有 JavaScript 语句的函数, 其中的 JavaScript 语句由定义窗口的 HTML 标记 `<body>` 或 `<frameset>` 的 `onresize` 性质指定。

Window.onunload

0 级 DOM

在浏览器卸载一个页面时调用的事件句柄

摘要

Function onunload

描述

Window 对象的属性 onunload 声明了一个事件句柄函数，这个函数是在浏览器卸载了一个文档或帧集，准备装载一个新文档时调用的。

这个属性的初始值是一个含有 JavaScript 语句的函数，其中的 JavaScript 语句由标记 <body> 或 <frameset> 的 onunload 性质指定，各个语句之间用分号分隔。

在装载新文档之前，应该清除浏览器的状态，onunload 事件句柄为此提供了机会。

onunload() 处理器是在用户指示浏览器离开当前页面，然后转移到其他页面时调用的。因此，要让 onunload 处理程序弹出一个对话框（调用 Window.confirm() 或者 Window.prompt()）以延迟新页面的装载是无论如何都不合适的。

参阅

Window.onload, 第 17 章

Window.open()

JavaScript 1.0

打开一个新的浏览器窗口或者查找一个已命名的窗口

摘要

window.open(url, name, features, replace)

参数

url

一个可选的字符串，声明了要在新窗口中显示的文档的 URL。如果省略这个参数，或者它的值是空字符串，那么新窗口就不显示任何文档。

name

一个可选的字符串，其中包括数字、字母和下划线，该字符串声明了新窗口的名称。这个名称可以作为标记 <a> 和 <form> 的性质 target 的值。如果这个参数指定一个已经存在的窗口，那么 open() 方法就不再创建一个新窗口，而只返回对指定窗口的引用。在这种情况下，参数 features 将被忽略。

features

一个字符串，声明了新窗口要显示的标准浏览器窗口的特征。在“Window Features”中对这个字符串格式进行了详细说明。该参数也是可选的，如果省略了它，新的窗口将具有所有标准特征。

replace

一个可选的布尔参数，声明了装载到窗口的 URL 是在窗口的浏览历史中创建一个新条目，还是用它替换浏览历史中的当前条目。如果这个参数的值为 `true`，那么就不创建新的历史条目。注意，它主要在改变已经存在的窗口的内容时使用。

返回值

对 Window 对象的引用，这个 Window 对象可能是新创建的，也可能是已经存在的，具体取决于参数 *name*。

描述

方法 `open()` 将查找一个已经存在的窗口或者打开一个新的浏览器窗口。如果参数 *name* 声明了一个已经存在的窗口，那么就返回对那个窗口的引用。返回的窗口将显示参数 *url* 指定的文档，但忽略参数 *features*。在只知道窗口名字的情况下，这是 JavaScript 获得对那个窗口引用的惟一方式。

如果没有指定参数 *name*，或者它指定的窗口不存在，那么 `open()` 方法将创建一个新的浏览器窗口。这个新窗口将显示参数 *url* 指定的 URL，它的名称由 *name* 指定，大小以及控件都由 *features* 参数指定（下面对该参数的格式进行了详细的说明）。如果 *url* 是一个空串，那么 `open()` 将打开一个空窗口。

参数 *name* 声明了新窗口的名称。这个名称中只能够出现数字、字母或下划线。它可以作为标记 `<a>` 和 `<form>` 的性质 `target` 的值，用来迫使文档在这个指定的窗口中显示。

当使用方法 `Window.open()` 给已有的指定窗口装载新文档时，可以给它传递参数 *replace*，用来声明新文档是在窗口的浏览历史中拥有自己的条目，还是替换当前文档的条目。如果 *replace* 的值为 `true`，新文档就会替换旧文档。如果它的值为 `false`，或者省略，那么新文档会在窗口的浏览历史中拥有自己的条目。这个参数提供的功能与方法 `Location.replace()` 提供的功能非常相似。

不要混淆了方法 `Window.open()` 与方法 `Document.open()`，这两者的功能完全不同。要使你的代码清楚明白，最好使用 `Window.open()`，而不要使用 `open()`。在定义为 HTML 性质的事件句柄中，通常把函数 `open()` 解释为 `Document.open()`，因此，在这种情况下，就必须使用 `Window.open()`。

窗口特征

参数 *features* 是一个窗口要显示的特征列表，其中各个特征之间用逗号分隔。如果这个可选参数的值为空，或者它被省略了，那么窗口将显示出所有特征。不过，如果 *features* 声明了某个特征，那么在这个列表中没有出现的特征就不会在窗口中显示出来。要注意的是，这个字符串不含任何空格或空白符，其中每个元素的格式如下所示：

`feature[=value]`

对于大多数的特征来说, *value* 的值是 *yes* 或 *no*。这些特征后的等号和 *value* 值都可以省略, 如果出现了该特征, 就假定它的 *value* 值为 *yes*, 如果没有出现, 就假定 *value* 值为 *no*。不过, 特征 *width* 和 *height* 的 *value* 值是必需的, 一定要指定它们的像素值。

可用的特征和它们的意义如下:

height

指定窗口的文档显示区的高度, 以像素计。

left

窗口的 X 坐标, 以像素计。

location

直接在浏览器中输入 URL 的输入字段。

menubar

菜单栏。

resizable

如果没有出现该特征, 或者设置为 *no*, 那么窗口就没有调整自己边界的操作 (根据平台的不同, 用户仍然可以使用其他方法调整窗口大小)。注意, 一个常见的错误是将这个特征误写为 “*resizeable*”, 即多写了一个 “*e*”。

scrollbars

在必要的情况下激活水平滚动条和垂直滚动条。

status

状态栏。

toolbar

浏览器的工具栏, 具有 **Back** 按钮和 **Forward** 按钮等。

top

窗口的 Y 坐标, 以像素计。

width

指定窗口的文档显示区的宽度, 以像素计。

参阅

`Location.replace()`、`Window.close()`、`Window` 对象的 `closed` 属性和 `opener` 属性

Window.print()

JavaScript 1.5

打印文档

摘要

`window.print()`

描述

调用 `print()` 方法所引发的行为就像用户单击了浏览器的 **Print** 按钮一样。通常，这会产生一个对话框，让用户可以取消或者定制打印请求。

Window.prompt()

JavaScript 1.0

获取对话框中输入的字符串

摘要

```
window.prompt(message, default)
```

参数

message

要在对话框中显示的纯文本（而不是 HTML 格式的文本）。它要求用户输入你想要得到的信息。

default

作为对话框中默认输入的字符串。要使 `prompt()` 显示一个空的输入框，就将这个参数设置为空串（`""`）。

返回值

用户输入的字符串，如果用户没有输入任何字符串，那么返回的就是空串，如果用户单击了 **Cancel** 按钮，返回的就是 `null`。

描述

方法 `prompt()` 将用一个对话框显示出指定的消息 *message*，这个对话框中含有文本输入字段、**OK** 按钮和 **Cancel** 按钮，其中由平台决定的图形向用户说明了需要他进行输入。如果用户单击了 **Cancel** 按钮，`prompt()` 将返回 `null`。如果用户单击了 **OK** 按钮，`prompt()` 将返回输入字段当前显示的文本。

由 `prompt()` 方法显示的对话框是有模式的，也就是说，在用户单击 **OK** 按钮或 **Cancel** 按钮来关闭它之前，它将阻塞用户对浏览器窗口的所有输入。由于该方法返回的值由用户的响应决定，所以在调用 `prompt()` 时，将暂停对 JavaScript 代码的执行。在用户作出响应之前，不会执行下一条语句。

参阅

`Window.alert()`、`Window.confirm()`

Window.removeEventListener()

参阅 `Element.removeEventListener()`

Window.resizeBy()JavaScript 1.2

将窗口大小调整一个相对的数量

摘要

```
window.resizeBy(dw, dh)
```

参数

dw 要使窗口宽度增加的像素数。

dh 要使窗口高度增加的像素数。

描述

方法 `resizeBy()` 将把窗口 *window* 的宽度和高度分别调整 *dh* 和 *dw* 个像素。参阅 `Window.resizeTo()` 下关于安全性和易用性考虑的讨论。

Window.resizeTo()JavaScript 1.2

调整窗口大小

摘要

```
window.resizeTo(width, height)
```

参数

width

想要调整到的窗口宽度。

height

想要调整到的窗口高度。

描述

方法 `resizeTo()` 将调整窗口 *window* 的大小，把它的宽度调整为 *width* 个像素，把它的高度调整为 *height* 个像素。出于安全性的原因，浏览器限制该方法，阻止脚本把窗口变得很小。出于易用性考虑，改变一个用户窗口的大小总是坏主意。如果一个脚本创建了一个窗口，脚本可以改变它的大小，但是一个脚本去改变它所载入的窗口的大小，这种形式不太好。

Window.scrollBy()JavaScript 1.2

将文档滚动一个相对的数量

摘要

```
window.scrollBy(dx, dy)
```

参数

dx 把文档向右滚动的像素数。

dy 把文档向下滚动的像素数。

描述

方法 `scrollBy()` 将把窗口中显示的文档向右滚动 *dx* 个像素、向下滚动 *dy* 个像素。

Window.scrollTo()

JavaScript 1.2

滚动文档

摘要

`window.scrollTo(x, y)`

参数

x 要在窗口文档显示区左上角显示的文档的 X 坐标。

y 要在窗口文档显示区左上角显示的文档的 Y 坐标。

描述

方法 `scrollTo()` 将滚动窗口 *window* 中显示的文档，使 *x* 和 *y* 指定的文档中的点显示在窗口的左上角（如果可能）。

Window.setInterval()

JavaScript 1.2

周期性地执行指定的代码

摘要

`window.setInterval(code, interval)`

参数

code

周期性地调用的一个函数或者周期性执行的 JavaScript 代码串。如果这个字符串含有多个语句，那么每个语句之间都用分号分隔开。在 IE 4 中，这个参数必须是一个字符串，但在此后的版本中就不是这样了。

interval

一个整数，声明了执行或调用 *code* 之间的时间间隔，以毫秒计。

返回值

一个可以传递给 `Window.clearInterval()` 从而取消对 *code* 的周期性执行的值。

描述

方法 `setInterval()` 可以重复地调用或执行 `code` 所指定的函数或字符串，每次执行的间隔为 `interval` 毫秒。

`setInterval()` 方法返回的值此后都可以用作方法 `Window.clearInterval()` 的参数，用来停止对 `code` 反复执行。

`setInterval()` 与 `setTimeout()` 有关。当你想延迟代码的执行、但不想重复执行时，可以使用 `setTimeout()`。参阅 `Window.setTimeout()` 对于 `code` 的执行环境的讨论。

参阅

`Window.clearInterval()`, `Window.setTimeout()`

Window.setTimeout()

JavaScript 1.0

延迟代码的执行

摘要

`window.setTimeout(code, delay)`

参数

`code`

延迟了 `delay` 时间后要调用的函数或者要执行的 JavaScript 代码串。如果这个参数是一个字符串，那么多条语句之间必须用分号相互隔开。在 IE 4 中，这个参数必须是一个字符串，在该浏览器中这个方法的函数形式不被支持。

`delay`

在执行 `code` 中的 JavaScript 语句之前要延迟的时间，以毫秒计。

返回值

一个不透明的值（“timeout id”），可以传递给 `clearTimeout()` 方法用来取消对 `code` 的执行。

描述

方法 `setTimeout()` 可以把一个 JavaScript 函数的调用或者一个 JavaScript 代码串的执行延迟了 `delay` 毫秒。注意，`setTimeout()` 只执行 `code` 一次。如果要多次调用，使用 `setInterval()` 或者让 `code` 自身再次调用 `setTimeout()`。

当 `code` 执行的时候，它是在 Window 对象的环境中执行的。如果 `code` 是一个函数，Window 对象就是 `this` 关键字的值。如果 `code` 是一个字符串，它是在全局作用域中执行的，而 Window 对象是作用域链上惟一的对象。即便 `setTimeout()` 的调用发生在一条长长的作用域链上的一个函数之中，这也是成立的。

参阅

`Window.clearTimeout()`, `Window.setInterval()`

Window.status

JavaScript 1.0

声明一条瞬时的状态栏消息

摘要

`String status`

描述

属性 `status` 是一个可读可写的字符串, 声明了要在窗口的状态栏中显示的一条瞬时消息。通常显示这条消息的时间是有限的, 直到其他的消息将它覆盖了, 或者用户把鼠标移动到窗口中的其他区域为止。当擦除了 `status` 声明的消息时, 状态栏要么是恢复为它默认的空白状态, 要么是再次显示出属性 `defaultStatus` 声明的默认消息。

在编写本书时, 很多浏览器已经关闭了脚本化它们的状态栏的功能。这是一种安全措施, 防止隐藏了超链接真正目标的钓鱼攻击。

参阅

`Window.defaultStatus`

XMLHttpRequest

Firefox 1.0, Internet Explorer 5.0, Safari 1.2, Opera 7.60

一个 HTTP 请求和响应

`Object` → `XMLHttpRequest`

构造函数

```
new XMLHttpRequest() // All browsers except IE 5 and IE 6
new ActiveXObject("Msxml2.XMLHTTP") // IE
new ActiveXObject("Microsoft.XMLHTTP") // IE with older system libraries
```

属性

`readonly short readyState`

HTTP 请求的状态。当一个 `XMLHttpRequest` 初次创建的时候, 这个属性的值从 0 开始, 直到接收到完整的 HTTP 响应, 这个值增加到 4。5 个状态中的每一个都有一个相关联的非正式的名称, 下表列出了状态、它们的名称及含义。

状态	名称	描述
0	Uninitialized	这是初始化状态。 <code>XMLHttpRequest</code> 对象已经创建或者已经被 <code>abort()</code> 方法重置
1	Open	<code>open()</code> 方法已经调用, 但 <code>send()</code> 还没有调用。请求还没有被发送

状态	名称	描述
2	Sent	send() 方法已经调用了, HTTP 请求已经发送到 Web 服务器。还没有接收到响应
3	Receiving	所有响应头部都已经接收到了。响应体开始接收但还没有完成
4	Loaded	HTTP 响应已经完全接收了

readyState 的值不会递减, 除非当一个请求已经在处理过程中的时候调用了 abort() 或 open() 方法。每次这个属性的值增加的时候, 都会触发 onreadystatechange 事件句柄。

readonly String.responseText

目前为止已经从服务器接收到的响应体 (不包括头部), 或者如果还没有数据接收到的话, 就是空字符串。如果 readyState 小于 3, 这个属性就是一个空字符串。当 readyState 为 3, 这个属性返回目前已经接收的响应部分。如果 readyState 为 4, 这个属性保存了完整的响应体。

如果响应包含了为响应体指定字符编码的头部, 就使用该编码。否则, 假定使用 Unicode UTF-8。

readonly Document.responseXML

对请求的响应, 解析为 XML 并作为 Document 对象返回。当下面 3 个条件都为真的时候, 这个属性为 null:

- readyState 为 4。
- 响应包含一个头部为 “text/xml”、“application/xml” 或任何以 “+xml” 结尾的 Content-Type, 表示响应是一个 XML 文档。
- 响应体由整齐的、解析没有错误的 XML 标记组成。

readonly short.status

由服务器返回的 HTTP 状态代码, 如 200 表示成功, 而 404 表示 “Not Found” 错误。当 readyState 小于 3 的时候读取这一属性将会导致一个异常。

readonly String.statusText

这个属性用名称而不是数字指定了请求的 HTTP 的状态代码。也就是说, 当状态为 200 的时候它是 “OK”, 当状态为 404 的时候它是 “Not Found”。和 status 属性一样, 当 readyState 小于 3 的时候读取这一属性将会导致一个异常。

方法

abort()

取消当前响应, 关闭连接并且结束任何未决的网络活动。

getAllResponseHeaders()

把 HTTP 响应头部作为未解析的字符串返回。

`getResponseHeader()`

返回指定的 HTTP 响应头部的值。

`open()`

初始化 HTTP 响应参数，例如 URL 和 HTTP 方法，但是并不发送请求。

`send()`

发送 HTTP 请求，使用传递给 `open()` 方法的参数，以及传递给该方法的可选请求体。

`setRequestHeader()`

向一个打开但未发送的请求设置或添加一个 HTTP 请求。

事件句柄

`onreadystatechange`

每次 `readyState` 属性改变的时候调用的事件句柄函数。当 `readyState` 为 3 时，它也可能调用多次。

描述

`XMLHttpRequest` 对象允许客户端 JavaScript 发布一个 HTTP 请求并接收来自 Web 服务器的响应（需要是 XML）。本书第 20 章的主题就是 `XMLHttpRequest`，其中包括很多使用它的例子。

`XMLHttpRequest` 具有较好的移植性，得到了所有现代浏览器较好的支持。唯一的浏览器依赖性涉及 `XMLHttpRequest` 对象的创建。在 IE 5 和 IE 6 中，必须使用特定于 IE 的 `ActiveXObject()` 构造函数，正如前面的构造函数部分所介绍的。

一旦创建了一个 `XMLHttpRequest` 对象，通常可以按照如下方式使用它：

1. 调用 `open()` 指定请求的 URL 和方法（通常是“GET”或“POST”）。当你调用 `open()`，也就指定了需要请求是同步的还是异步的。
2. 如果指定了异步请求，把 `onreadystatechange` 属性设置为请求进程需要通知到的函数。
3. 调用 `setRequestHeader()`，如果需要的话，指定额外的请求参数。
4. 调用 `send()` 来向 Web 服务器发送请求。如果这是一个 POST 请求，可能要给这个方法传递一个请求体。如果在调用 `open()` 的时候指定了一个异步请求，`send()` 方法会阻塞，直到请求完成并且 `readyState` 为 4。否则，`onreadystatechange` 事件句柄函数必须等到 `readyState` 属性达到 4（或者至少是 3）。
5. 一旦 `send()` 已经为同步请求而返回，或者 `readyState` 已经为异步请求而达到 4，就可以使用服务器响应了。首先，察看状态代码确保请求成功。如果是成功的，使用 `getResponseHeader()` 或 `getResponseHeaders()` 获取响应头部的值，并且使用 `responseText` 或 `responseXML` 属性来获取响应体。

XMLHttpRequest 对象还没有标准化，但是在编写本书的时候，W3C 已经开始了标准化的工作。这里介绍的内容都是基于标准化的工作草案。当前的 XMLHttpRequest 实现已经相当一致，但是和标准有细微的不同。例如，一个实现可能返回 `null`，而标准要求的是空字符串；或者实现可能把 `readyState` 设置为 3 而不保证所有的响应头部都可用。

参阅

第 20 章

XMLHttpRequest.abort()

取消一个 HTTP 请求

摘要

```
void abort()
```

描述

这个方法把 XMLHttpRequest 对象重置为 `readyState` 为 0 的状态，并且取消所有未决的网络活动。例如，如果请求用了太长时间，而且响应不再必要的时候，可以调用这个方法。

XMLHttpRequest.getAllResponseHeaders()

返回一个未解析的 HTTP 响应头部

摘要

```
String getAllResponseHeaders()
```

返回值

如果 `readyState` 小于 3，这个方法返回 `null`。否则，它返回服务器发送的所有 HTTP 响应的头部（但是没有状态栏）。头部作为单个的字符串返回，一行一个头部。每行用换行符“`\r\n`”隔开。

XMLHttpRequest.getResponseHeader()

获取一个指定的 HTTP 响应的头部

摘要

```
String getResponseHeader(String header)
```

参数

header

其值要被返回的 HTTP 响应头部的名称。可以使用任意大小写来指定这个头部名字，和响应头部的比较是不区分大小写的。

返回值

指定的 HTTP 响应头部的值，如果没有接收到这个头部或者 `readyState` 小于 3 则为空字符串。如果接收到多个有指定名称的头部，这些头部的值被连接起来并返回，使用逗号和空格分隔开各个头部的值。

XMLHttpRequest.onreadystatechange

`readyState` 改变的时候调用的事件句柄

摘要

Function onreadystatechange

描述

这个属性指定了一个事件句柄函数，每次 `readyState` 属性变化的时候就调用它。当 `readyState` 为 3 的时候，也有可能调用这个句柄多次来提供下载进度通知。

一个 `onreadystatechange` 句柄通常察看 `XMLHttpRequest` 对象的 `readyState`，看它是否达到了 4。如果达到了，它使用 `responseText` 或 `responseXML` 来执行某些操作。是否有任何参数传递给函数并不确定。特别是，事件句柄函数没有标准的方法来获取对它注册其上的 `XMLHttpRequest` 对象的一个引用。这意味着，不可能编写可以为多个请求所使用的一个通用句柄函数。

`XMLHttpRequest` 对象被假设遵从 DOM 事件模型，并且实现了一个 `addEventListener()` 方法来为 `readystatechange` 事件注册事件句柄（例如，参阅 `Event.addEventListener()`）。既然 IE 不支持 DOM 事件模型，并且既然每个请求很少需要多个事件句柄，那么只赋给 `onreadystatechange` 一个句柄函数是安全的。

XMLHttpRequest.open()

初始化 HTTP 请求参数

摘要

```
void open(String method,  
          String url,  
          boolean async,  
          String username, String password)
```

参数

method

用于请求的 HTTP 方法。可靠实现的值包括 GET、POST 和 HEAD。实现也可能支持方法。

url

这个 URL 是请求的主题。大多数浏览器实施了一个同源安全策略（参阅 13.8.2 节），并且要求这个 URL 和包含脚本的文档具有相同的主机名和端口。相关的 URL 也用标准的方式解析，使用包含脚本的文档的 URL。

async

请求是否应该异步地执行。如果这个参数是 `false`，请求是同步的，后续对 `send()` 的调用将阻塞，直到响应完全接受。如果这个参数为 `true` 或者省略，请求是异步的，并且通常需要一个 `onreadystatechange` 事件句柄。

username, password

这些可选的参数为使用 URL 所需的授权指明了认证资格。如果指定了，它们会覆盖 URL 自己所指定的任何资格。

描述

这个方法初始化请求参数以供 `send()` 方法稍后使用。它把 `readyState` 设置为 1，删除之前指定的所有请求头部，以及之前接收的所有响应头部，并且把 `responseText`、`responseXML`、`status` 和 `statusText` 属性设置为它们的默认值。当 `readyState` 为 0 的时候（当 `XMLHttpRequest` 对象刚刚创建或者 `abort()` 方法调用后）以及当 `readyState` 为 4（响应已经接受）的时候，调用这个方法是安全的。当针对任何其他状态调用的时候，`open()` 的行为是未指定的。

除了保存供 `send()` 使用的请求参数以及重置 `XMLHttpRequest` 对象以便复用，`open()` 方法没有其他的行为。要特别注意，当这个方法调用的时候，实现通常不会打开一个到 Web 服务器的网络连接。

参阅

`XMLHttpRequest.send()`，第 20 章

`XMLHttpRequest.send()`

发送一个 HTTP 请求

摘要

```
void send(Object body)
```

参数

body

如果通过调用 `open()` 指定的 HTTP 方法是“POST”或“PUT”，这个参数指定了请求体，作为一个字符串或者 `Document` 对象，如果请求体不是必须的话，这个参数就为 `null`。对于任何其他方法，这个参数是不用的，应该为 `null`（有些实现不允许省略这个参数）。

描述

这个方法导致一个 HTTP 请求发送。如果之前没有调用 `open()`，或者更具体地说，如果 `readyState` 不是 1，`send()` 抛出一个异常。否则，它发送一个 HTTP 请求，该请求由以下几部分组成：

- 之前调用 `open()` 时指定的 HTTP 方法、URL 以及认证资格（如果有的话）。
- 之前调用 `setRequestHeader()` 时指定的请求头部（如果有的话）。
- 传递给这个方法的 `body` 参数。

一旦请求发布了，`send()` 把 `readyState` 设置为 2，并触发 `onreadystatechange` 事件句柄。

如果之前调用 `open()` 的参数 `async` 为 `false`，这个方法会阻塞并不会返回，直到 `readyState` 为 4 并且服务器的响应被完全接受。否则，如果 `async` 参数为 `true`，或者如果这个参数省略了，`send()` 立即返回，并且正如后面所介绍的，服务器响应将在一个后台线程中处理。

如果服务器响应带有一个 HTTP 重定向，`send()` 方法或后台线程自动遵从重定向。当所有的 HTTP 响应头已经接受，`send()` 或后台线程把 `readyState` 设置为 3 并触发 `onreadystatechange` 事件句柄。如果响应较长，`send()` 或后台线程可能在状态 3 中触发 `onreadystatechange` 事件句柄：这可以作为一个下载进度指示器。最后，当响应完成，`send()` 或后台线程把 `readyState` 设置为 4，并最后一次触发事件句柄。

参阅

`XMLHttpRequest.open()`，第 20 章

`XMLHttpRequest.setRequestHeader()`

为请求添加一个 HTTP 请求头部

摘要

```
void setRequestHeader(String name, String value)
```

参数

name

要设置的头部的名称。这个参数不应该包括空白、冒号或换行符。

value

头部的值。这个参数不应该包括换行。

描述

`setRequestHeader()` 指定了一个 HTTP 请求的头部，它应该包含在通过后续的 `send()`

调用而发布的请求中。这个方法只有当 `readyState` 为 1 的时候才能调用，例如，在调用了 `open()` 之后，但在调用 `send()` 之前。

如果带有指定名称的头部已经被指定了，这个头部的新值就是：之前指定的值，加上逗号、空白以及这个调用指定的值。

如果 `open()` 调用指定了认证资格，`XMLHttpRequest` 自动发送一个适当的 `Authorization` 请求头部。但是，你可以使用 `setRequestHeader()` 来添加这个头部。类似地，如果 Web 服务器已经保存了和传递给 `open()` 的 URL 相关联的 cookie，适当的 `Cookie` 或 `Cookie2` 头部也自动地包含到请求中。可以通过调用 `setRequestHeader()` 来把这些 cookie 添加到头部。`XMLHttpRequest` 也可以为 `User-Agent` 头部提供一个默认值。如果它这么做，你为该头部指定的任何值都会添加到这个默认值后面。

有些请求头部由 `XMLHttpRequest` 自动设置而不是由这个方法设置，以符合 HTTP 协议。这包括如下和代理相关的头部：

```
Host
Connection
Keep-Alive
Accept-Charset
Accept-Encoding
If-Modified-Since
If-None-Match
If-Range
Range
```

参阅

`XMLHttpRequest.getResponseHeader()`

XMLSerializer

Firefox 1.0, Safari 2.01, Opera 7.60

序列化 XML 文档和节点

Object → XMLSerializer

构造函数

`new XMLSerializer()`

方法

`serializeToString()`

这个实例方法执行实际的序列化。

描述

`XMLSerializer` 对象使你能够把一个 XML 文档或 Node 对象转化或“序列化”为未解析的 XML 标记的一个字符串。要使用一个 `XMLSerializer`，使用不带参数的构造函数实例化它，然后调用其 `serializeToString()` 方法：

```
var text = (new XMLSerializer()).serializeToString(element);
```


IE 不支持 XMLSerializer 对象。相反，它通过 Node 对象的 xml 属性使 XML 文本变为可用。

参阅

DOMParser、Node，第 21 章

XMLSerializer.serializeToString()

把一个 XML 文档或节点转换为一个字符串

摘要

```
String serializeToString(Node node)
```

参数

node

要序列化的 XML 节点。这可能是文档中的一个 Document 对象或者任何 Element。

返回值

XML 标记的一个字符串，代表了指定节点及其所有子孙的序列化形式。

XPathExpression

Firefox 1.0, Safari 2.01, Opera 9

一个编译过的 XPath 查询

Object → XPathExpression

方法

```
evaluate()
```

为一个指定的 Context 节点计算这个表达式。

描述

XPathExpression 对象是一个 XPath 查询的编译过的表现形式，由 Document.createExpression() 返回。使用 evaluate() 方法根据一个特定的文档节点来计算该表达式。如果你需要仅计算一个 XPath 查询一次，可以使用 Document.evaluate()，它在一个步骤里编译并计算表达式。

IE 不支持 XPathExpression 对象。参阅 Node.selectNodes() 和 Node.selectSingleNode()，了解特定于 IE 的 XPath 方法。

参阅

```
Document.createExpression()
```

```
Document.evaluate()
```

```
Node.selectNodes()
```

```
Node.selectSingleNode()
```

第 21 章

XPathExpression.evaluate()

计算一个编译过的 XPath 查询

摘要

```
XPathResult evaluate(Node contextNode,  
                     short type,  
                     XPathResult result)
```

参数

contextNode

计算查询所应该依据的节点（或文档）。

type

期待的结果类型。这个参数应该是 XPathResult 定义的一个常量。

result

一个 XPathResult 对象，查询结果存储于其中。如果要想 evaluate() 方法创建并返回一个新的 XPathResult 对象，则为 null。

返回值

保存查询结果的一个 XPathResult。要么是作为 *result* 参数传递的对象，或者是 *result* 为 null 而新建的一个 XPathResult 对象。

描述

这个方法根据指定的节点或文档来计算 XPathExpression 并把结果返回到一个 XPathResult 对象。参阅 XPathResult，了解如何从返回对象中提取值。

参阅

Document.evaluate()、Node.selectNodes()、XPathResult，第 21 章

XPathResult

Firefox 1.0; Safari 2.01; Opera 9

XPath 查询的结果

Object → XPathResult

常量

如下的常量定义了一个 XPath 查询可能返回的类型。XPathResult 对象的 *resultType* 属性保存了这些值中的一个，用来指定对象保存何种结果。这些常量也同 Document.evaluate() 和 XPathExpression.evaluate() 方法一起使用，用来指定期待的结果类型。这些常量及其含义如下：

ANY_TYPE

把这个值传递给 Document.evaluate() 或 XPathExpression.evaluate() 来指定可接受的结果类型。属性 *resultType* 并不设置这个值。

NUMBER_TYPE

numberValue 保存结果。

STRING_TYPE

stringValue 保存结果。

BOOLEAN_TYPE

booleanValue 保存结果。

UNORDERED_NODE_ITERATOR_TYPE

这个结果是节点的无序集合，可以通过重复调用 `iterateNext()` 直到返回 `null` 来依次访问。在此迭代过程中，文档必须不被修改。

ORDERED_NODE_ITERATOR_TYPE

结果是节点的列表，按照文档中的属性排列，可以通过重复调用 `iterateNext()` 直到返回 `null` 来依次访问。在此迭代过程中，文档必须不被修改。

UNORDERED_NODE_SNAPSHOT_TYPE

结果是一个随即访问的节点列表。`snapshotLength` 属性指定了列表的长度，并且 `snapshotItem()` 方法返回指定下标的节点。节点可能和它们出现在文档中的顺序不一样。既然这种结果是一个“快照”，因此即便文档发生变化，它也有效。

ORDERED_NODE_SNAPSHOT_TYPE

这个结果是一个随机访问的节点列表，就像 `UNORDERED_NODE_SNAPSHOT_TYPE`，只不过这个列表是按照文档中的顺序排列的。

ANY_UNORDERED_NODE_TYPE

`singleNodeValue` 属性引用和查询匹配的一个节点，如果没有匹配的节点则为 `null`。如果有多个节点和查询匹配，`singleNodeValue` 可能是任何一个匹配节点。

FIRST_ORDERED_NODE_TYPE

`singleNodeValue` 保存了文档中的第一个和查询匹配的节点，如果没有匹配的节点，则为 `null`。

实例属性

这里的很多属性只有当 `resultType` 保存一个特定的值的时候才有效。访问并非为当前 `resultType` 定义的属性会导致异常。

`readonly boolean booleanValue`

当 `resultType` 为 `BOOLEAN_TYPE` 时，保存结果值。

`readonly boolean invalidIteratorState`

如果 `resultType` 是 `ITERATOR_TYPE` 常量中的一个并且文档已经修改，则为 `true`，它使迭代无效，因为结果已经返回。

`readonly float numberValue`

当 `resultType` 为 `NUMBER_TYPE` 时，保存结果值。

readonly short resultType

指定 XPath 查询返回何种结果。它的值是前面列出的常量之一。这个属性的值告诉你可以使用哪些其他属性和方法。

readonly Node singleNodeValue

当 resultType 为 XPathResult.ANY_UNORDERED_NODE_TYPE 或 XPathResult.FIRST_UNORDERED_NODE_TYPE 时，保存结果值。

snapshotLength

当 resultType 为 UNORDERED_NODE_SNAPSHOT_TYPE 或 ORDERED_NODE_SNAPSHOT_TYPE 时，指定返回的节点数。和 snapshotItem() 联合使用这一属性。

stringValue

当 resultType 为 STRING_TYPE 时，保存结果值。

方法

iterateNext()

如果 resultType 是 UNORDERED_NODE_ITERATOR_TYPE 或 ORDERED_NODE_ITERATOR_TYPE，使用这一方法。

snapshotItem()

返回结果节点列表中指定下边的节点。这个方法只有在 resultType 为 UNORDERED_NODE_SNAPSHOT_TYPE 或 ORDERED_NODE_SNAPSHOT_TYPE 的时候才能使用。snapshotLength 属性和这个方法一起使用。

描述

XPathResult 对象表示一个 XPath 表达式的值。这种类型的对象由 Document.evaluate() 和 XPathExpression.evaluate() 返回。XPath 查询可以计算为字符串、数字、布尔值、节点以及节点的列表。XPath 实现可以以几种不同的方式返回节点的列表，因此，这个对象为获取一个 XPath 查询的实际结果而定义了略为复杂的 API。

要使用一个 XPathResult，首先检查 resultType 属性。它将保存一个 XPathResult 常量。这个属性的值告诉你需要使用哪个属性和方法来确定结果值。调用不是为当前的 resultType 定义的方法或者读取不是为它定义的属性会导致异常。

IE 不支持 XPathResult API。要在 IE 中执行 XPath 查询，参阅 Node.selectNodes() 和 Node.selectSingleNode()。

参阅

Document.evaluate()、XPathExpression.evaluate()

XPathResult.iterateNext()

返回和一个 XPath 查询匹配的下一个节点

摘要

```
Node iterateNext()  
    throws DOMException
```

返回值

返回匹配节点列表中的下一个节点，如果没有其他节点，则为 null。

抛出

既然 XPathResult 被返回，如果文档已经修改了，这个方法会抛出异常。如果 returnType 不是 UNORDERED_NODE_ITERATOR_TYPE 或 ORDERED_NODE_ITERATOR_TYPE 的时候调用，方法也会抛出异常。

描述

iterateNext() 返回和 XPath 查询匹配的下一个节点，如果所有匹配的节点都已经返回了，则为 null。当 resultType 是 UNORDERED_NODE_ITERATOR_TYPE 或 ORDERED_NODE_ITERATOR_TYPE 的时候使用这个方法。如果类型是有序的，节点按照它们在文档中出现的顺序返回，否则，它们按照任意顺序返回。

如果 invalidIteratorState 属性为 true，文档已经被修改了，这个方法抛出异常。

XPathResult.snapshotItem()

返回和一个 XPath 查询匹配的一个节点

摘要

```
Node snapshotItem(index)
```

参数

index

要返回的节点的下标。

返回值

指定下标的节点，如果下标小于 0 或者大于或等于 snapshotLength，则为 null。

抛出

如果 resultType 不是 UNORDERED_NODE_SNAPSHOT_TYPE 或 ORDERED_NODE_SNAPSHOT_TYPE，这个方法抛出异常。

XSLTProcessor

Firefox 1.0, Safari 2.0.1, Opera 9

用 XSLT 样式表来转换 XML

Object → XSLTProcessor

构造函数`new XSLTProcessor()`**方法**`clearParameters()`

删除之前的任何设置参数。

`getParameter()`

返回指定的参数的值。

`importStyleSheet()`

指定要使用的 XSLT 样式表。

`removeParameter()`

删除指定的参数。

`reset()`

将 XSLTProcessor 重置为初始状态，清空所有的参数和样式表。

`setParameter()`

把指定的参数设置为一个指定的值。

`transformToDocument()`使用传递给 `importStylesheet()` 的样式表和传递给 `setParameter()` 的参数，来转换指定的文档或节点。结果作为一个新的 Document 对象返回。`transformToFragment()`

转换指定的文档或节点，把结果作为一个 DocumentFragment 返回。

描述

XSLTProcessor 使用 XSLT 样式表来转换 XML 文档。使用无参数的构造函数来创建一个 XSLTProcessor 对象，并通过 `importStylesheet()` 方法用一个 XSLT 样式表来初始化它。如果你的样式表使用参数，可以用 `setParameter()` 来设置这些参数。最后，使用 `transformToDocument()` 或 `transformToFragment()` 来执行实际的转换。

IE 支持 XSLT 但是没有实现 XSLTProcessor 对象。参阅特定于 IE 的 Node 的 `transformNode()` 和 `transformNodeToObject()` 方法，并参阅第 21 章的 XSLT 示例和跨平台工具函数。

参阅

`Node.transformNode()`、`Node.transformNodeToObject()`，第 21 章