JavaScript

HTML定義網頁內容

CSS指定網頁佈局

JavaScript編寫網頁行為

不同於伺服器端手稿語言，例如PHP與ASP，JavaScript主要被作為客戶端手稿語言在使用者的瀏覽器上運行，不需要伺服器的支援

包括以下部分:

ECMAScript描述了該語言的語法和基本物件

文件物件模型DOM描述處理網頁內容的函數和介面

瀏覽器物件模型BOM描述與瀏覽器進行互動的函數和介面

基本特點如下:

是一種解釋性程式語言，代碼不進行預編譯

主要用來向HTML頁面添加互動行為

可以直接嵌入HTML頁面，但寫成單獨的js檔案有利於結構和行為的分離

常用來完成以下任務:

嵌入動態文字於HTML頁面

對瀏覽器事件作出回應

讀寫HTML元素

在資料被提交到伺服器之前驗證資料

檢測訪客的瀏覽器資訊

控制cookie，包括建立和修改等

HTML中的<script>標籤

定義客戶端腳本，比如JavaScript，可以包含腳本語句，也可以通過src屬性指向外部腳本文件， type屬性不是必需的，規定腳本的MIME類型，JavaScript是HTML中的默認腳本語言，腳本可以放置在<body>或<head>，或者在兩者

<script type="text/javascript">

document.write("Hello World!")

</script>

外部JavaScript，myFunction存儲在名為myScript.js的外部文件中

<button type="button" onclick="myFunction()">Try it</button>

<script src="myScript.js"></script>

外部優點:

將HTML和JavaScript分開，易於閱讀和維護

緩存的JavaScript文件可以加快頁面加載速度

HTML事件

是發生在HTML元素上的事物，可以是瀏覽器執行的操作，也可以是用戶執行的操作，在HTML頁面中使用JavaScript時，JavaScript可以對這些事件做出反應

以下是一些HTML事件示例:

onchange HTML元素已更改

onclick 用戶單擊一個HTML元素

onmouseover 用戶將鼠標移到HTML元素上

onmouseout 用戶將鼠標從HTML元素移開

onkeydown 用戶按下鍵盤鍵

onload 瀏覽器已完成頁面加載

HTML允許使用JavaScript代碼將事件處理程序屬性添加到HTML元素中

單引號:

<element event='some JavaScript'>

雙引號:

<element event="some JavaScript">

例:

將onclick屬性(帶有代碼)添加到 <button>元素

<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

JavaScript代碼通常很長，查看事件屬性調用函數更為常見

<button onclick="displayDate()">The time is?</button>

<script>

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

</script>

JavaScript輸出

innerHTML寫入HTML元素

更改HTML元素的innerHTML屬性，是在HTML中顯示數據的常用函數

document.write()寫入HTML輸出

加載HTML文檔後使用document.write，將刪除所有現有的HTML，document.write應僅用於測試

window.alert()寫入警報框

JavaScript中，窗口對像是全局範圍對象，這意味著默認情況下變量，屬性和函數屬於窗口對象。這也意味著指定window關鍵字是可選的

console.log()寫入瀏覽器控制台

在瀏覽器中調用console.log()以顯示數據進行調試

JavaScript語法

JavaScript語句由以下組成:

值，運算符，表達式，關鍵字和註釋

JavaScript值

固定值稱為文字Literals

1.數字可以帶或不帶小數

2.字符串是文本，用雙引號或單引號引起來

變量值稱為變量Variables

1.變量用於存儲數據值

2.JavaScript使用var關鍵字聲明變量

變量命名規則:

(1)變量名稱必須以字母或$和\_開頭

(2)變量名稱可以包含字母，數字，下劃線和美元符號

(3)變量名稱區分大小寫

(4)關鍵字不能用於變量名稱

var x;

x = 6;

可以在聲明變量時為其賦值

var x = 6;

可以在一個語句中聲明多個變量，並用逗號分隔變量

var a, b, c;

重新聲明一個JavaScript變量，不會丟失其值

如果將數字用引號引起來，則其餘數字將被視為字符串並連接在一起

var x = "5" + 2 + 3; //523

var x = 2 + 3 + "5"; //55

JavaScript運算符

算術運算符:

+ 加

- 減

\* 乘

\*\* 冪、指數 //5\*\*2=25

/ 除

% 返回除法的餘數

++ 遞增

* 遞減

賦值運算符:

= 將值分配給變量

x += y等同x = x + y

x -= y等同x = x - y

x \*= y等同x = x \* y

x /= y等同x = x / y

x %= y等同x = x % y

x <<= y等同x = x << y

x >>= y等同x = x >> y

x >>>= y等同x = x >>> y

x &= y等同x = x & y

x ^= y等同x = x ^ y

x |= y等同x = x | y

x \*\*= y等同x = x \*\* y

比較運算符:

== 值等於

=== 值、數據類型皆等於

!= 值不等於

!== 值、數據類型皆不等於

> 值大於

< 值小於

>= 值大於或值等於

<= 值小於或值等於

? 三元運算符

condition ? exprIfTrue : exprIfFalse

等同

if (condition) { exprIfTrue }

else { exprIfFalse }

condition1 ? value1

: condition2 ? value2

: condition3 ? value3

: value4;

等同

if (condition1) { return value1; }

else if (condition2) { return value2; }

else if (condition3) { return value3; }

else { return value4; }

邏輯運算符:

&& 且

|| 或

! 不

類型運算符:

typeof返回一個變量數據類型的String

typeof "John" // Returns "string"

typeof 3.14 // Returns "number"

typeof NaN // Returns "number" (NaN)

typeof true // Returns "boolean"

typeof false // Returns "boolean"

typeof Symbol()// Returns "symbol"

typeof {name:'John', age:34} // Returns "object"

typeof null // Returns "object" (bug)

typeof [1,2,3,4] // Returns "object" (Array)

typeof new Date() // Returns "object" (Date)

typeof function () {} // Returns "function"

typeof undefined // Returns "undefined"

typeof myCar // Returns "undefined" (尚未分配值的變量)

instanceof判斷該對像是否使用構造函數創建

按位運算符:

JavaScript將數字存儲為64位浮點數，但是所有按位運算都是以32位二進制數執行

在執行按位運算前，將數字轉換為32位帶符號整數；執行按位運算後，將結果轉換回64位數字

& AND如果兩個位元均為1，則返回1 例: 5(0101) & 1(0001) = 1(0001)

| OR 如果其中一位元為1，則返回1例: 5(0101) | 1(0001) = 5(0101)

^ XOR如果兩個位元不同，則返回1例: 5(0101) ^ 1(0001) = 4(0100)

~ NOT反轉所有位元 例: ~5(0101) = 10(1010)

<< 從右側推入一個或多個零位元，左側的位元落下 例: 5(0101) << 1 = 10(1010)

>> 從左側推入一個或多個最左側位元的副本，右側的位元落下 例: 5(0101) >> 1 = 2(0010)

>>> 從左側推入一個或多個零位元，右側的位元落下 例: 5(0101) >>> 1 = 2(0010)

負數為正數反轉+1

5 = 0101

-5 = 1010(正數反轉)+1 =1011

JavaScript表達式

表達式是值，變量和運算符的組合，可以計算出一個值

JavaScript關鍵字

關鍵字是保留字，保留字不能用作變量的名稱

break跳出switch或loop

continue在loop中跳過一個疊代

debugger停止執行JavaScript，如果有調試功能，調用調試功能

do ... while當條件為真時，執行語句塊並重複該塊

for只要條件為真，就標記要執行的語句塊

function聲明一個函數

if ... else根據條件標記要執行的語句塊

return退出函數

switch根據不同的情況標記要執行的語句塊

this引用其所屬的對象

try ... catch對語句塊實施錯誤處理

var聲明一個以函數作用域區分作用域的變量

let聲明一個以塊級作用域區分作用域變量

const聲明一個以塊級作用域區分作用域且有恆定值的變量

this關鍵字

調用代碼所屬的對象

單獨使用時(包含嚴格模式下)，所有者是全局對象，this是指全局對象(在瀏覽器窗口中，全局對像是[object Window])

在函數中，函數的所有者默認綁定this，因此this是指調用該函數的對象

在箭頭函數中，this代表調用該函數的對象

在對象方法中，this是指方法的所有者(對象)

在構造函數中的this沒有值

在調用構造函數創建的新對象中，構造函數中的this是指新對象

在函數中，嚴格模式下不允許默認綁定，因此this為undefined

在HTML事件處理程序中，this指的是接收事件的HTML元素

call()和apply()方法是預定義函數方法，可以指定調用函數的this值到任意對象，可以將某個對象方法在不同對象上使用

JavaScript註釋

//單行註釋

/\*多行註釋\*/

JavaScript語句

分號分隔JavaScript語句，在末尾添加分號

var a, b, c;

當用分號分隔時，允許在一行上使用多個語句

var a, b, c; a = 5; b = 6; c = a + b;

JavaScript會忽略多個連續空格，在運算符周圍放置空格，以使其更具可讀性

var x = y + z;

避免使用超過80個字符的代碼行，如果JavaScript語句不適合一行，則最佳位置是在運算符之後

document.getElementById("demo").innerHTML =

"Hello Dolly!";

JavaScript數據類型

JavaScript具有動態類型，意味著可以使用同一變量來保存不同的數據類型，變量不用宣告數據類型

原始數據類型Primitive data type:

具有原始值的數據

原始值是一種沒有額外屬性和方法的單一簡單數據值，是硬編碼的，因此不能更改

x = 3.14

可以更改x的值，但不能更改3.14的值

1.字符串string

可以在字符串內使用引號，只要它們與字符串周圍的引號不匹配即可

var answer = 'He is called "Johnny"';

2.數字number

可以使用科學指數表示法

var y = 123e5; //12300000

3.布爾Boolean

4.未定義undefined

沒有值的變量具有值undefined、數據類型undefined

var car; //值undefined、數據類型undefined

將值設置為undefined，可以使變量具有值undefined、數據類型undefined

car = undefined; //值undefined、數據類型undefined

5.空值null

JavaScript中null的數據類型是Object是JavaScript最初實現中的一個bug

typeof null === 'object'

其原理是不同的對像在底層都是用二進制來表示的，在JavaScript中二進制前3位是0即判斷是為Object，null的二進製表示是全0，即前三位都是0，所以執行typeof返回的是object，實際上null為基本數據類型

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

person = null; //值null、數據類型Object

6. symbol

用來創建匿名的對象屬性

7. bigInt

一種特殊的數字類型，為任意長度的整數提供支持，通過

將n整數附加到整數文字的末尾

const bigint = 1234567890123456789012345678901234567890n;

調用BigInt從字符串

const sameBigint = BigInt("1234567890123456789012345678901234567890");

調用BigInt從數字

const bigintFromNumber = BigInt(10); // same as 10n

來創建bigint

引用數據類型Complex data type:

8.對象Object

對像用大括號編寫{}，對象屬性寫為名稱:值對，以逗號分隔

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

函數Function、數組Array也是一個Object

JavaScript函數Function

JavaScript函數是使用function關鍵字定義，可以使用函數聲明或函數表達式，函數可以用作值，函數是對象

函數聲明

聲明的函數不會立即執行，它們被保存以備後用，並且在被調用called upon時執行，函數聲明不是可執行語句，因此以分號結尾並不常見

function functionName(parameters) {

// code to be executed

}

函數表達式

函數表達式可以存儲在變量中，該函數是一個匿名函數(一個沒有名稱的函數)，不需要函數名稱，可以使用變量名稱來調用它們，以分號結尾，因為是可執行語句的一部分，建議使用const比使用var更加安全，因為函數表達式(包含箭頭函數)始終為常數

var x = function (a, b) {return a \* b};

var z = x(4, 3);

JavaScript函數語法

函數命名規則與變量相同

參數是函數調用時收到的值，在括號中列出參數名稱，用逗號分隔

在函數內部，參數表現為局部變量

到達return語句，函數將停止執行，並返回值，沒有return，返回值是undefined

訪問不帶()的函數將返回函數的函數表達式而不是函數的結果

匿名anonymous自調用Self-Invoking函數(立即呼叫函式表達式immediately-invoked function expression IIFE)

定義完立即執行，自動調用，而非被調用

(function() {

// 這裡的語句將獲得新的作用域

})();

若要將作用域外變數傳遞進函式

(function(a, b) {

// a == 'hello'

// b == 'world'

})('hello', 'world');

JavaScript箭頭函數

允許使用簡短的語法編寫函數表達式，箭頭函數跟函數表達式一樣必須先聲明再使用，在函數中，函數的所有者默認綁定this，因此this是指調用該函數的對象，包含箭頭函數

hello = function() {

return "Hello World!";

}

等同

hello = () => {

return "Hello World!";

}

如果函數只有一個語句，該語句返回一個值，可以省略{}和return(默認情況下，箭頭函數返回值)，僅當函數只有一個語句時，此方法才有效

hello = () => "Hello World!";

可以在括號內傳遞參數

hello = (val) => "Hello " + val;

如果只有一個參數，可以省略()

hello = val => "Hello " + val;

建議不要省略{}、return、()

JavaScript函數參數

函數參數是傳值by value，不是傳址by reference，函數僅了解參數的值；而不是參數的位置，如果函數更改了參數的值，不會更改參數的原始值(對於參數的更改在函數外部不可見)

參數規則

JavaScript函數未指定參數的數據類型、不對傳遞的參數執行類型檢查、不檢查接收到的參數數量

默認參數

如果使用缺少的參數(少於聲明的參數數量)調用函數，則將缺少的參數值設置為 undefined

建議將默認值分配給參數

function myFunction(x, y) {

if (y === undefined) {

y = 2;

}

}

ECMAScript 2015允許在函數聲明中使用默認參數值

function (x, y = 2) {

// function code

}

rest參數

在陣列中能在任何索引處將一個數組的所有元素插入另一個數組

let thisArray = ['sage', 'rosemary', 'parsley', 'thyme'];

let thatArray = ['basil', 'cilantro', ...thisArray, 'coriander'];

//thatArray等於['basil', 'cilantro', 'sage', 'rosemary', 'parsley', 'thyme', 'coriander']。

如果函式的最後一個命名參數以...開頭，會被視為一個陣列(theArgs)，該陣列的元素會被置於索引從0(含)到的theArgs.length(不含)位置，並且被視為一個函式的參數

//theArgs會將函式f中第三個(含)以後的參數收集起來

function f(a, b, ...theArgs) {

// ...

}

參數對象Arguments Object

JavaScript函數具有一個稱為arguments的內置對象，包含了調用函數時使用的參數數組

arguments object是一個類數組Array-like object，跟數組一樣擁有length屬性，以及從0開始的索引

如果調用的函數帶有太多的參數(超過了聲明的參數數量)，則可以使用arguments object來訪問這些參數

參數對象屬性

length屬性

提供了實際傳遞給函數的參數數量，可以大於或小於函數定義的參數數量

函數方法

預定義函數方法(在函數原型Function prototype中繼承的方法):

apply()方法

調用具有指定this值和參數的函數，參數以數組形式(數組對象array object或參數對象arguments object)提供

句法:

fun.apply(thisArg, [argsArray])

參數值:

thisArg(必須)用來指定函式內的this值

如果函數在非嚴格模式下執行，thisArg會被轉化為object:

若thisArg是null或undefined，則this為global object

若thisArg是其他原始值primitive value，則thisArg會被封裝為object

如果函數在嚴格模式下執行，thisArg會毫無改變的變成函式內的this:

若thisArg不是object，則this為調用函數的所有者

argsArray(可選)提供給函數的參數，一個數組對象或參數對象，數組中的第一個元素被視為函數的第一個參數，依此類推，如果是null或undefined或無值，函數不傳入參數；如果既不是數組對象，也不是參數對象arguments object，報錯為TypeError

返回值:

將this和參數傳入函數後得到的結果

call()方法

調用具有指定this值和參數的函數

句法:

fun.call(thisArg, arg1, arg2..., argn)

參數值:

thisArg(必須)用來指定函式內的this值

如果函數在非嚴格模式下執行，thisArg會被轉化為object:

若thisArg是null或undefined，則this為global object

若thisArg是其他原始值primitive value，則thisArg會被封裝為object

如果函數在嚴格模式下執行，thisArg會毫無改變的變成函式內的this:

若thisArg不是object，則this為調用函數的所有者

arg1, arg2..., argn(可選)提供給函數的參數，如果無值，函數不傳入參數

返回值:

將this和參數傳入函數後得到的結果

bind()方法

創建一個具有指定this值、指定參數、原函數的參數的新函數(綁定函數)

句法:

fun.bind(thisArg, arg1, arg2..., argn)

參數值:

thisArg(必須)用來指定新函數內的this值，使用new調用新函數時，該參數無效

arg1, arg2, ...argn(可選)新增的指定參數，當新函數被調用時，這些參數加上綁定函數本身的參數會按照順序作為原函數運行時的參數

返回值:

一個具有指定this值、指定參數、原函數的參數的新函數

JavaScript函數調用

1.調用函數作為函數

function myFunction(a, b) {

return a \* b;

}

myFunction(10, 2);

單獨使用時(包含嚴格模式下)，this是指全局對象

在JavaScript中，有一個默認的全局對象

在HTML中，全局對像是HTML頁面HTML page

在瀏覽器中，全局對像是瀏覽器窗口window

全局變量、函數、方法容易在全局對像中導致名稱衝突name conflict和bug

2.調用函數作為對象方法

var myObject = {

firstName:"John",

lastName: "Doe",

fullName: function () {

return this.firstName + " " + this.lastName;

}

}

myObject.fullName();

在對象方法中，this是指方法的所有者(對象)

3.調用函數作為構造函數

function myFunction(arg1, arg2) {

this.firstName = arg1;

this.lastName = arg2;

}

// This creates a new object

var x = new myFunction("John", "Doe");

調用函數前加上new關鍵字，則函數將作為構造函數被調用

構造函數將創建一個新對象，新對像從其構造函數繼承屬性和方法

4.用預定義函數方法Call、apply調用函數

Call、apply可以指定調用函數的this值，可以將某個對象方法在不同對象上使用

var person = {

fullName: function() {

return this.firstName + " " + this.lastName;

}

}

var person1 = {

firstName:"John",

lastName: "Doe"

}

person.fullName.call(person1); // Will return "John Doe"

person.fullName.apply(person1); // Will return "John Doe"

JavaScript函數作用域scope

全局Global變量

網頁上的所有腳本和函數都可以訪問

沒有使用var、let、const聲明的變量，將自動成為全局變量，即使在函數內創建；在嚴格模式下，則不會自動成為全局變量

JavaScript的全局作用域就是完整的JavaScript環境；HTML的全局作用域是window對象

在Web瀏覽器中，當您關閉瀏覽器窗口或選項卡時，全局變量將被刪除

局部Local變量

在函數中聲明的變量只能在函數內部訪問

具有相同名稱的變量可以在不同的函數中使用

函數啟動時創建局部變量，函數完成時將局部變量刪除

函數參數在函數內部充當局部變量

var關鍵字

聲明的變量具有全局作用域global scope(全局變量)或函數作用域function scope(局部變量)

不具有塊級作用域，除了函數，其他塊block訪問的變量範圍都是屬於全局作用域，可以從塊外部訪問在{}內部聲明的變量

let關鍵字

聲明的變量具有具有全局作用域global scope(全局變量)或塊級作用域(局部變量)

在塊{}中聲明的變量不能從該塊外部訪問

const關鍵字

聲明的變量具有全局作用域global scope(全局變量)或塊級作用域(局部變量)且具有恆定值

在塊{}中聲明的變量不能從該塊外部訪問

聲明時必須同時初始化(賦值)

不能更改常數原始值，但是可以更改常數對象的屬性、常數數組的元素

重新聲明

var可以重新聲明

var x = 2; // Now x is 2

var x = 3; // Now x is 3

不允許在相同作用域中用var重新聲明let

var x = 2; // Allowed

let x = 3; // Not allowed

{

var x = 4; // Allowed

let x = 5 // Not allowed

}

不允許在相同作用域中用let重新聲明let

let x = 2; // Allowed

let x = 3; // Not allowed

{

let x = 4; // Allowed

let x = 5; // Not allowed

}

不允許在相同作用域中用let重新聲明var

let x = 2; // Allowed

var x = 3; // Not allowed

{

let x = 4; // Allowed

var x = 5; // Not allowed

}

允許在另一個作用域用let重新聲明

let x = 2; // Allowed

{

let x = 3; // Allowed

}

{

let x = 4; // Allowed

}

JavaScript嵌套函數

所有函數都可以訪問它們上層的作用域，即可以使用父作用域的變量(自由變數)

JavaScript閉包Closure

函數以及該函數被聲明時詞法環境(所在作用域環境)的組合，利用作用域創造斯有變數

當捕捉閉包的時候，它的自由變數會在捕捉時被確定(保留一個到其父作用域的鏈接，防止父函數內的數據被垃圾回收)，這樣即便脫離了捕捉時的上下文，也能照常執行

透過閉包讓函數有私有變數

//定義函數addhome

function addhome() {

//count的自由變數

var counter = 0;

//add()時，真正執行的函數

function count() {

counter += 1;

console.log(counter)};

}

//返回真正執行函數的函數表達式

return count;

}

//add聲明、賦值

//賦值同時執行一次addhome()，使counter = 0、add成為count的函數表達式

var add = addhome()

add(); //1

add(); //2

等同

//add聲明、賦值

// IIFE定義完立即執行一次addhome()，使counter = 0、add成為count的函數表達式

var add = (function () {

var counter = 0;

return function () {

counter += 1;

console.log(counter);

}

})();

add(); //1

add(); //2

JavaScript對象Object

鍵:值對的集合，在JavaScript中，對象使用命名索引(元素名稱為字符串)

除原始值primitives外，所有JavaScript值都是對象

Booleans(原始值)可以是對象(不建議，如果使用new關鍵字定義)

Numbers(原始值)可以是對象(不建議，如果使用new關鍵字定義)

Strings(原始值)可以是對象(不建議，如果使用new關鍵字定義)

Date、Math、Regular expression、Array、Function、Object始終是對象

對像可以是變量

對像是傳址by reference，不是傳值by value，對像僅了解對象屬性值的位置；而不是對象屬性的值，如果函數更改了對象屬性的值，會更改對象屬性的原始值(對於對象屬性的更改在函數外部可見)

對象繼承其原型的屬性

JavaScript對象語法:

對象屬性Object Property:

鍵:值對中值是原始數據類型或對象

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

訪問對象屬性:

objectName.propertyName // person.age

如果屬性名稱中有空格或要使用變量來命名屬性，必須使用[]訪問對象屬性

objectName["property Name"] // person["age"]

或

objectName[expression] // x = "age"; person[x]

delete關鍵字

刪除屬性的屬性和值，不得在預定義的對象屬性上使用，可能會使應用程序崩潰，不會刪除繼承屬性

對象方法Object Methods:

對象方法語法

鍵:值對中值是函數

在對象方法中，this關鍵字是指方法的所有者(對象)

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

//從ES6開始，引入了針對對像初始化器方法定義的較短語法

fullName () {

return this.firstName + " " + this.lastName;

}

};

訪問對象方法

objectName.methodName() //person.fullName();

如果訪問不帶()的對象方法，將返回函數定義

objectName.methodName //person.fullName;

// function() {return this.firstName + " " + this.lastName;}

對象靜態方法(Object.Method())

keys()方法

返回object中所有可枚舉的屬性名稱(key)的數組

句法:

Object.keys(obj)

參數值:

obj指定對象

返回值:

指定對象所有可枚舉屬性的字符串數組

values()方法

返回object中所有的propertyvalue(value)的數組

句法:

Object.values(obj)

參數值:

obj指定對象

返回值:

指定對象所有可枚舉屬性值的字符串數組

entries()方法

返回object中所有的propertyname/propertyvalue對(key/value對)的數組

句法:

Object.entries(obj)

參數值:

obj指定對象

返回值:

指定對象所有可枚舉屬性鍵值對的字符串數組

defineProperty()方法

在一個object上定義新屬性或修改現有屬性，並返回這個object

句法:

Object.defineProperty(obj, prop, descriptor)

參數值:

obj定義或修改屬性的object

prop要被定義或修改的屬性名稱propertyname

descriptor被定義或修改的屬性的屬性描述器Property descriptor，屬性描述器主要有兩種(資料描述器data descriptor和訪問描述器accessor descriptor)，資料描述器是可以選擇能否覆寫的屬性；存取器描述符是由一對getter-setter函數描述的屬性，屬性描述器必須是這兩種形式之一，不能兩者兼有

資料描述器:

var obj = new Object();

Object.defineProperty(obj, 'name', {

configurable: false,

enumerable: true,

writable: true,

value: '張三'

})

console.log(obj.name) //張三

Configurable

表示能否通過delete刪除此屬性、能否修改屬性的特性、或能否修改把屬性修改為訪問器屬性，預設值為true

Enumerable

表示該屬性是否可列舉，即是否通過for-in迴圈或Object.keys()返回屬性，預設值為true

Writable

能否修改屬性的值，預設值為true

Value

該屬性對應的值，預設為undefined

訪問描述器:

var obj = {counter : 0};

Object.defineProperty(obj, "decrement", {

get : function () {this.counter--;}

});

Object.defineProperty(obj, "add", {

set : function (value) {this.counter += value;}

});

Configurable

表示能否通過delete刪除此屬性、能否修改屬性的特性、或能否修改把屬性修改為訪問器屬性，預設值為true

Enumerable

表示該屬性是否可列舉，即是否通過for-in迴圈或Object.keys()返回屬性，預設值為true

Get

一個給屬性提供getter的方法(訪問物件屬性時呼叫的函式)，如果沒有，則預設為 undefined，該方法返回值被用作屬性值

Set

一個給屬性提供setter的方法(給物件屬性設定值時呼叫的函式)，如果沒有，則設為undefined，該方法將接受唯一引數，並將該引數的新值分配給該屬性

返回值:

定義或修改屬性後的object

defineProperties()方法

在一個object上定義一個或多個新屬性或修改現有屬性，並返回這個object

句法:

Object.defineProperties(obj, props)

參數值:

obj定義或修改屬性的object

props該object的一個或多個鍵值對，定義將要為物件新增或修改的屬性的具體配置，屬性描述器Property descriptor主要有兩種(資料描述器data descriptor和訪問描述器accessor descriptor)，資料描述器是可以選擇能否覆寫的屬性；存取器描述符是由一對getter-setter函數描述的屬性，屬性描述器必須是這兩種形式之一，不能兩者兼有

var obj = new Object();

Object.defineProperties(obj, {

name: {

value: '張三',

configurable: false,

writable: true,

enumerable: true

},

age: {

value: 18,

configurable: true

}

})

console.log(obj.name, obj.age) // 張三, 18

資料描述器:

Configurable

表示能否通過delete刪除此屬性、能否修改屬性的特性、或能否修改把屬性修改為訪問器屬性，預設值為true

Enumerable

表示該屬性是否可列舉，即是否通過for-in迴圈或Object.keys()返回屬性，預設值為true

Writable

能否修改屬性的值，預設值為true

Value

該屬性對應的值，預設為undefined

訪問描述器:

Configurable

表示能否通過delete刪除此屬性、能否修改屬性的特性、或能否修改把屬性修改為訪問器屬性，預設值為true

Enumerable

表示該屬性是否可列舉，即是否通過for-in迴圈或Object.keys()返回屬性，預設值為true

Get

一個給屬性提供getter的方法(訪問物件屬性時呼叫的函式)，如果沒有，則預設為 undefined，該方法返回值被用作屬性值

Set

一個給屬性提供setter的方法(給物件屬性設定值時呼叫的函式)，如果沒有，則設為undefined，該方法將接受唯一引數，並將該引數的新值分配給該屬性返回值:

定義或修改屬性後的object

getOwnPropertyDescriptor()方法

返回一個對象，該對象包含指定對象的一個命名屬性(直接存在於對像上，而不是對象原型鏈中的屬性)的所有屬性描述符

const object1 = {

property1: 42,

property2: 43

};

const descriptor1 = Object.getOwnPropertyDescriptor(object1, 'property1');

console.log(descriptor1);

// expected output: Object { value: 42, writable: true, enumerable: true, configurable: true }

console.log(descriptor1.value);

// expected output: 42

句法:

Object.getOwnPropertyDescriptor(obj, prop)

參數值:

obj要在其中查找命名屬性的對象

prop要檢索其屬性描述符的命名屬性名稱或其Symbol

返回值:

一個對象，該對象包含指定對象的一個命名屬性的所有屬性描述符；否則返回undefined

getOwnPropertyDescriptors()方法

返回一個對象，該對象包含指定對象的所有命名屬性(直接存在於對像上，而不是對象原型鏈中的屬性)的所有屬性描述符

const object1 = {

property1: 42,

property2: 43

};

const descriptors1 = Object.getOwnPropertyDescriptors(object1);

console.log(descriptors1);

// expected output: Object { property1: Object { value: 42, writable: true, enumerable: true, configurable: true }, property2: Object { value: 43, writable: true, enumerable: true, configurable: true } }

console.log(descriptors1.property1);

// expected output: Object { value: 42, writable: true, enumerable: true, configurable: true }

console.log(descriptors1.property1.value);

// expected output: 42

句法:

Object.getOwnPropertyDescriptors(obj)

參數值:

obj要獲取其所有命名屬性的所有屬性描述符的對象

返回值:

一個對象，該對象包含指定對象的所有命名屬性的所有屬性描述符；否則返回一個空對象

getOwnPropertyNames()方法

返回一個數組，其中包含指定對象自身所有屬性(包含可枚舉和不可枚舉，除了使用Symbol的屬性)名稱

句法:

Object.getOwnPropertyNames(obj)

參數值:

obj要返回其可枚舉和不可枚舉屬性的對象

返回值:

一個字符串數組，包含指定對象自身所有屬性名稱

getPrototypeOf()方法

返回指定對象的原型(即內部的Prototype屬性值)

句法:

Object.getPrototypeOf(obj)

參數值:

obj要返回其原型的對象

返回值:

指定對象的原型，如果沒有繼承的屬性，則返回null

preventExtensions()方法

防止該對象進行擴展(將新屬性添加到對象)

句法:

Object.preventExtensions(obj)

參數值:

obj應使其不可擴展的對象

返回值:

不可擴展的對像

isExtensible()方法

確認對像是否可擴展(將新屬性添加到對象)

句法:

Object.isExtensible(obj)

參數值:

obj應該檢查的對象

返回值:

一個布爾值，如果可擴展，返回true；反之，則返回false

seal()方法

密封對象(防止向對象添加新屬性)，並將所有現有屬性標記為不可配置non-configurable，只要是writable，當前屬性值仍可更改

句法:

Object.seal(obj)

參數值:

obj應密封的對象

返回值:

被密封的對象

isSealed()方法

確認對像是否被密封

句法:

Object.isSealed(obj)

參數值:

obj應該檢查的對象

返回值:

一個布爾值，如果被密封，返回true；反之，則返回false

freeze()方法

凍結(防止向對象添加新屬性、刪除現有屬性、更改現有屬性的可枚舉性enumerability或可配置性configurability或可寫性writability、更改現有屬性的值、更改對象原型)對象，凍結的對象無法再更改

句法:

Object.freeze(obj)

參數值:

obj要凍結的對象

返回值:

傳遞給函數的相同對象

isFrozen()方法

確認對像是否被凍結

句法:

Object.isFrozen(obj)

參數值:

obj應該檢查的對象

返回值:

一個布爾值，如果被凍結，返回true；反之，則返回false

對象實例方法(Object.prototype.Method)

hasOwnProperty()方法

返回一個指示對像是否具有指定命名屬性(直接存在於對像上，而不是對象原型鏈中的屬性)的布爾值

句法:

obj.hasOwnProperty(prop)

參數值:

prop要測試屬性名稱的String或屬性的Symbol

返回值:

一個指示對像是否具有指定命名屬性的布爾值

new關鍵字

使用new聲明變量時，將作為對象創建(不要將字符串，數字和布爾值聲明為對象)

JavaScript Accessors訪問器

Getter和Setter可以定義對象訪問器，以屬性而非函數訪問對象(允許屬性和方法具有相同的語法)，提供了更簡單的語法，確保更好的數據質量，對於在幕後做事很有用

JavaScript Getter屬性獲取器(get關鍵字)

var person = {

language : "en",

get lang() {

return this.language;

}

};

// Display data from the object using a getter:

document.getElementById("demo").innerHTML = person.lang; //en

JavaScript Setter屬性設置器(set關鍵字)

var person = {

language : "",

set lang(lang) {

this.language = lang;

}

};

// Set an object property using a setter:

person.lang = "en";

// Display data from the object:

document.getElementById("demo").innerHTML = person.language; //en

JavaScript構造函數Constructor

設計用來創建新對象的函數，建議構造函數名稱的第一個字母需大寫

對像類型Object Types (擁有語法糖:類Class)

使用對象構造函數創建對像類型(Person)

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

this.name = function(){

return this.firstName + " " + this.lastName;

};

}

使用new關鍵字調用構造函數來創建相同類型的對象(myFather myMother)

var myFather = new Person("John", "Doe", 50, "blue");

var myMother = new Person("Sally", "Rally", 48, "green");

instanceof

判斷該對像是否使用構造函數創建(檢測構造函數的prototype屬性是否出現在該實例對象的原型鏈上)

myFather instanceof Person; //true

構造函數中的this沒有值，在調用構造函數創建新對象後，構造函數中的this是指新對象(myFather myMother)，因為在函數中，this關鍵字是指調用該函數的對象

可以向對象添加屬性、屬性，只會添加到該對象

myFather.nationality = "English";

myFather.name = function () {

return this.firstName + " " + this.lastName;

};

向對象構造函數中添加新屬性、方法，不能像對像一樣，必須將其添加到構造函數的內部

Person.nationality = "English"; //不能用

Person.name = function() {

return this.firstName + " " + this.lastName;

}; //不能用

內置JavaScript構造函數

var x1 = new Object(); // A new Object object

var x2 = new String(); // A new String object

var x3 = new Number(); // A new Number object

var x4 = new Boolean(); // A new Boolean object

var x5 = new Array(); // A new Array object

var x6 = new RegExp(); // A new RegExp object

var x7 = new Function(); // A new Function object

var x8 = new Date(); // A new Date object

Math是全局對象，new關鍵字不能使用，故不在內置JavaScript構造函數中

JavaScript有原始數據類型的對象版本(String、Number、Boolean)，但是沒有理由創建複雜的對象，因為原始值比較快

使用以下，而非用new

var x1 = {}; // new object

var x2 = ""; // new primitive string

var x3 = 0; // new primitive number

var x4 = false; // new primitive boolean

var x5 = []; // new array object

var x6 = /()/ // new regexp object

var x7 = function(){}; // new function object

JavaScript對象原型Prototype

原型繼承Inheritance

所有JavaScript對像都從原型繼承屬性和方法

一個對象prototype本身就是一個對象，所以prototype可以有它自己的prototype，例如Person.prototype的prototype是Object.prototype

Object.prototype是原型繼承鏈的頂端

原型鍊下端相同名稱的屬性、方法會覆蓋上端的屬性、方法

prototype屬性

允許向對象構造函數添加新屬性、方法(避免每次創建新對象實力時，重複創建相同的對象方法或屬性)

Person.prototype.nationality = "English";

Person.prototype.name = function() {

return this.firstName + " " + this.lastName;

};

將prototype設置為一個已經包含屬性的新對象，就可以一次添加所有屬性

Person.prototype = {

constructor: Person,

//手動將原型設置為新對像，會擦除該constructor屬性，記住定義constructor屬性以避免

nationality : "English",

name : function() {

return this.firstName + " " + this.lastName;

}

};

myFather instanceof Person; //true

不要修改標準JavaScript對象的原型

Object.prototype.constructor屬性

所有對像都會從它的原型上繼承一個constructor屬性

var o = {};

o.constructor === Object; // true

var a = [];

a.constructor === Array; // true

var n = new Number(3);

n.constructor === Number; // true

Object.prototype.isPrototypeOf()方法

用於測試一個對像是否存在於另一個對象的原型鏈上

Person.prototype.isPrototypeOf(myFather); //true

Object.create()方法

指定其原型物件與屬性，創建一個新物件

將多個子類別原型中重複的屬性、方法放於父類別原型中

//父類別Person

function Person(age) {

this.age = age;

}

//父類別的方法(放在原型，避免創建實例，重複創造副本)

Person.prototype = {

eat: function() {

console.log("nom nom nom");

}

};

//子類別Man

function Man() { }

//子類別擴展(extends)父類別

Man.prototype = Object.create(Person.prototype);

//手動將原型設置為新對像，會擦除該constructor屬性，需重新定義constructor屬性以避免

Man.prototype.constructor = Man;

//子類別實例

let mydad = new Man();

JavaScript Class類

JavaScript對象的模板template，ES6中引入了類別class作為原型程式prototype-based繼承(利用構造函數創建對象的對像類型(對象原型))的語法糖(類別並不是引入新的物件導向繼承模型，而是提供一個更簡潔的語法來建立物件和處理繼承)

JavaScript類語法

使用關鍵字class創建一個類，始終添加一個構造方法constructor()，然後添加任意數量方法，類語法必須以嚴格模式編寫

句法

class ClassName {

constructor() {

}

method\_1() {

}

}

使用class創建類(Person)，添加構造方法constructor()，然後添加類方法name

class Person {

constructor(first, last) {

this.firstName = first;

this.lastName = last;

}

name() {

return this.firstName + " " + this.lastName;

}

}

使用new關鍵字調用constructor()方法來創建相同類的對象(myFather myMother)

var myMother = new Person("Sally", "Rally");

類方法Class Method

創建類方法和對象方法(ES6引入對象方法定義的速寫語法)是相同的語法

構造方法Constructor Method

1.必須具有確切的名稱constructor

2.創建新對象時自動執行

3.用於初始化對象屬性

如果未定義構造函數方法，JavaScript將添加一個空的constructor()方法

JavaScript類繼承

super()引用父類，將調用父類的constructor()，以訪問父類的屬性和方法，super必須在this之前使用，以避免this產生reference error

句法

super([arguments]); // calls the parent constructor.

super.functionOnParent([arguments]);

使用extends 關鍵字創建類繼承，通過類繼承創建的類將從另一個類繼承所有方法

class Man extends Person {

constructor(first, last, age, eye) {

super(first, last); //調用父類(Person)的constructor(first, last)，以訪問父類的屬性和方法

this.age = age;

this.eyeColor = eye;

}

}

var myFather = new Man ("John", "Doe", 50, "blue");

屬性獲取器Getter和設置器Setter

在類中也可以使用get和set關鍵字添加getter和setter

Getter和Setter可以定義對象訪問器，以屬性而非函數訪問對象(允許屬性和方法具有相同的語法)

class Car {

constructor(brand) {

this.carname = brand;

}

get cnam() {

return this.carname;

}

set cnam(x) {

this.carname = x;

}

}

let myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;

類聲明不會被提升，必須先聲明一個類，才能使用

JavaScript靜態方法Static Method

使用關鍵字static來定義一個靜態方法static method給類別class，靜態方法只能由類別本身呼叫，在由類別所建立的對象實體instance上不能被呼叫，通常作為工具函式utility functions使用，用來建立對象或複製對象

class Car {

constructor(name) {

this.name = name;

}

static hello() {

return "Hello!!";

}

}

document.getElementById("demo").innerHTML = Car.hello();

實例方法

使用關鍵字static來實例方法定義一個靜態方法由類別所建立的對象實體instance呼叫

JavaScript全局屬性和函數

可以與所有內置JavaScript對像一起使用

JavaScript全局屬性Global Property

Infinity屬性

正無窮大

句法:

Infinity

返回值:

Infinity

-Infinity屬性

負無窮大

句法:

-Infinity

返回值:

-Infinity

NaN屬性

指示某個值是不是數字值

句法:

NaN

返回值:

NaN

undefined屬性

指示未定義的值

句法:

undefined

返回值:

undefined

JavaScript全局函數Global Function

在Web瀏覽器中，全局對像Global Object是瀏覽器窗口window

因為是Global，所以可以直接用Global Function替代Global Method，例如isNaN()替代window.isNaN()

decodeURI() 解碼某個編碼的URI

decodeURIComponent() 解碼一個編碼的URI 組件

encodeURI() 把字符串編碼為URI

encodeURIComponent() 把字符串編碼為URI 組件

eval() 計算JavaScript 字符串，並把它作為腳本代碼來執行

String()函數

將對象的值轉換為字符串

句法:

String(object)

參數值:

object(必須)JavaScript對象object

返回值:

一個字符串

isFinite()函數

確定一個值是否為有限數

句法:

isFinite(value)

參數值:

value(必須)判斷值

返回值:

如果值為有限數，則返回true；否則，返回false

isNaN()函數

確定一個值是否為NaN

句法:

isNaN(value)

參數值:

value(必須)判斷值

返回值:

如果值為NaN，則返回true；否則，返回false

String() 把對象的值轉換為字符串

unescape() 對由escape() 編碼的字符串進行解碼

將變量轉換為數字:

Number()函數

把對象的值轉換為數字

句法:

Number(object)

參數值:

object(可選) JavaScript對象object

返回值:

一個數字，如果該值不能轉換為合法數字，則返回NaN，如果未提供任何object參數，則返回0

parseFloat()函數

解析一個字符串並返回一個浮點數，允許有空格，僅返回第一個數字

句法:

parseFloat(string)

參數值:

string(必須)指定解析的字符串

返回值:

一個浮點數數字，如果第一個字符不能轉換為數字，則返回NaN

parseInt()函數

解析字符串並返回整數，允許有空格，僅返回第一個數字

句法:

parseInt(string, radix)

參數值:

string(必須)指定解析的字符串

radix (可選) 2至36之間任意整數，代表用幾進制轉換

返回值:

一個整數數字，如果第一個字符不能轉換為數字，則返回NaN

原始值(原始數據類型的值)的屬性和函數

原始值，例如"Hello World!"不能具有屬性或函數，因為它們不是對象，但是JavaScript在執行函數和屬性時會將其視為對象，所以屬性

var str = "Hello World!";

var n = str.length;

和函數

var str = "Hello World!";

var res = str.charAt(0)

也可用於原始數據類型的值

JavaScript字符串 String

用於存儲和處理文本

反斜杠轉義字符 \

由於字符串必須用引號引起來，因此在字符串中使用特殊字符(' ” /)，需使用反斜杠轉義字符將特殊字符轉換為字符串字符

\' 單引號字元

var x = 'It\'s alright.';

//It's alright.

\" 雙引號字元

var x = "We are the so-called \"Vikings\" from the north.";

//We are the so-called "Vikings" from the north.

\\ 反斜槓字元

var x = "The character \\ is called backslash.";

//The character \ is called backslash.

\t 水平制表符

\r Enter

\n 換行符

\b 退格

\f 換頁

字符串屬性String Property:

句法:

string.string\_property

str.string\_property(簡寫)

length屬性

返回字符串的長度

句法:

str.length

返回值:

字符串的長度

字符串方法String Methods:

句法:

string.string\_methods()

str.string\_ methods()(簡寫)

所有字符串函數都返回一個新值，不會更改原始變量

查找字符串:

JavaScript從零開始計數位置，0是字符串中的第一個位置

indexOf()方法

返回字符串中第一次出現指定文本的位置

句法:

str.indexOf(searchvalue, start)

參數值:

searchvalue(必須)指定搜索文本

start (可選)搜索的起始位置

返回值:

指定搜索值首次出現的位置；如果從未出現，則返回-1

lastIndexOf()方法

返回字符串中最後一次出現指定文本的位置

句法:

str.lastIndexOf(searchvalue, start)

參數值:

searchvalue(必須)指定搜索文本

start (可選)搜索的起始位置，從末尾開始搜索字符串，但返回從頭開始的索引

返回值:

指定搜索值最後一次出現的位置；如果從未出現，則返回-1

search()方法

返回字符串中第一次出現指定文本的位置，搜索值可以是字符串或正則表達式

句法:

str.search(searchvalue)

參數值:

searchvalue (必須)指定搜索文本，可以是字符串或正則表達式

返回值:

指定搜索值首次出現的位置；如果從未出現，則返回-1

match()方法

在字符串中搜索與正則表達式的匹配項，然後將匹配項作為數組對象返回

句法:

string.match(regexp)

參數值:

regexp(必須)正則表達式，如果不包含g修飾符(全局匹配)，則僅返回字符串中的第一個匹配項

返回值:

一個包含各個匹配項的數組對象，如果找不到匹配項，則返回null

提取部分字符串:

slice()方法

提取字符串的一部分

句法:

str.slice(start, end)

參數值:

start (必須)提取的起始位置，使用負數可以從字符串末尾進行提取

end(可選)提取的結束位置，不包含在字符串的提取部分

返回值:

一個包含提取字符的新字符串

substring()方法

提取字符串的一部分，不能接受負索引

句法:

str.substring(start, end)

參數值:

start (必須)提取的起始位置，如果小於0，則視為0

end(可選)提取的結束位置，不包含在字符串的提取部分，如果小於0，則視為0

返回值:

一個包含提取字符的新字符串

substr()方法

提取字符串的一部分

句法:

str.substr(start, length)

參數值:

start (必須)提取的起始位置，使用負數可以從字符串末尾進行提取

length(可選)提取的字符串長度，如果為0或負數，則返回一個空字符串

返回值:

一個包含提取字符的新字符串

替換字符串內容:

replace()方法

在字符串中搜索指定值或正則表達式，返回替換了指定值的新字符串

句法:

str.replace(regexp |substr, newSubstr|function)

參數值:

regexp |substr (必須)指定搜索文本，可以是字符串或正則表達式

newSubstr|function (必須)指定替換文本，可以是字符串或或函式(在每一次匹配時被呼叫)

返回值:

替換了指定值的新字符串

默認情況下，replace()函數僅替換第一個匹配項，要替換所有匹配項，請使用帶有標誌的正則表達式/g(全局匹配項)，正則表達式不帶引號

str = "Please visit Microsoft and Microsoft!";

var n = str.replace(/Microsoft/g, "W3Schools");

轉換為大寫和小寫:

toUpperCase()方法

將字符串轉換為大寫字母

句法

str.toUpperCase()

參數值:

無

返回值:

轉換為大寫的字符串

toLowerCase()方法

將字符串轉換為小寫字母

句法

str.toLowerCase()

參數值:

無

返回值:

轉換為小寫的字符串

連接字符串:

concat()方法

用於連接兩個或多個字符串

句法:

str.concat(string1, string2, ..., stringX)

參數值:

string1, string2, ..., stringX(必須)連接的字符串

返回值:

一個包含組合字符串的新字符串

可以使用加號運算符代替concat()函數，兩者的功能相同

var text = "Hello" + " " + "World!";

var text = "Hello".concat(" ", "World!");

刪除字符串空格:

trim()方法

從字符串的兩側刪除空格

句法:

str.trim()

參數值:

無

返回值:

兩端都刪除空格的字符串

字符串填充:

padStart()方法

在字符串的開頭填充

句法:

str.padStart(len, str);

參數值:

len(必須)填充後字符串的長度

str(必須)填充的字符串

返回值:

填充後的字符串

padEnd()方法

在字符串的結尾填充

句法:

str.padEnd(len, str);

參數值:

len(必須)填充後字符串的長度

str(必須)填充的字符串

返回值:

填充後的字符串

提取字符串字符:

charAt()方法

返回字符串中指定位置的字符

句法:

string.charAt(index)

參數值:

index(必須)指定搜索位置

返回值:

一個包含指定位置字符的字符串；如果找不到，則為空字符串

charCodeAt()方法

返回字符串中指定位置的字符Unicode(UTF-16)

句法:

string.charCodeAt(index)

參數值:

index(可選)指定搜索位置；默認為0

返回值:

指定位置字符的Unicode，如果指定位置沒有字符或索引小於0，則返回NaN

將字符串轉換為數組:

split()方法

用於將字符串拆分為子字符串數組，並返回新數組

句法:

string.split(separator, limit)

參數值:

separator (可選)字符串的分隔符，如果省略，則返回的數組將在[0]中包含整個字符串，如果將空字符串””用作分隔符，則該字符串將在每個字符之間分割

limit (可選)字符串的拆分長度，如果超出長度，則不會出現在數組中

其餘方法:

endsWith()方法

確定字符串是否以指定字符串的字符結尾，正確則返回true；否則返回false，區分大小寫

fromCharCode()方法

將Unicode值轉換為字符，這是String對象的靜態方法，語法始終為String.fromCharCode()

includes()方法

確定字符串是否包含指定字符串的字符，正確則返回true；否則返回false，區分大小寫

localeCompare()方法

比較當前語言環境中的兩個字符串

repeat()方法

返回一個新字符串，該字符串具有指定數量的被調用字符串的副本

startsWith()方法

確定字符串是否以指定字符串的字符開頭，正確則返回true；否則返回false，區分大小寫

toLocaleLowerCase()方法

根據主機的當前語言環境將字符串轉換為小寫字母，不會更改原始字符串toLocaleUpperCase()方法

根據主機的當前語言環境將字符串轉換為大寫字母，不會更改原始字符串toString()方法

返回String對象的值

valueOf()方法

返回String對象的原始值，通常由JavaScript在幕後自動調用，而不是在代碼中明確調用

JavaScript數字 Number

JavaScript數字始終是64位雙精度浮點數，不會定義不同類型的數字

數字 : 52位(0-51) 指數 : 11位(52-62) 符號 : 1位(63)

整數(不帶.或科學指數表示法的數字)的精度最高為15位

小數位數的最大值是17，但是浮點算術不總是準確

JavaScript將+用於加法(數字)和串聯(字符串)，從左到右起作用

var x = 10;

var y = 20;

var z = "30";

var result = x + y + z; //3030

JavaScript將嘗試在所有數字運算中將字符串轉換為數字，+用於串聯字符串

var x = "100";

var y = "10";

var z = x - y; //90

var x = "100";

var y = "10";

var z = x + y; //10010

數字屬性Number Property:

句法:

number.number\_property

num.number\_property(簡寫)

數字屬性只能用於數字，只能以Number. Number\_Properties訪問，否則將返回undefined

var x = 6;

var y = x.MAX\_VALUE; //undefined

MAX\_VALUE屬性

返回JavaScript中可能的最大數字，大於MAX\_VALUE的數字表示為無窮大

句法:

Number.MAX\_VALUE

返回值:

1.7976931348623157e + 308

MIN\_VALUE屬性

返回JavaScript中可能的最小正數，小於此的數字將轉換為0

句法:

Number.MIN\_VALUE

返回值:

5e-324

NEGATIVE\_INFINITY屬性

負無窮大

句法:

Number.NEGATIVE\_INFINITY

返回值:

-Infinity

NaN屬性

Not-a-Number，數字不是合法數字

句法:

Number.NaN

返回值:

NaN

POSITIVE\_INFINITY屬性

正無窮大

句法:

Number.POSITIVE\_INFINITY

返回值:

Infinity

EPSILON屬性

1與大於1可表示的最小浮點數的差值

句法:

Number.EPSILON

返回值:

1與大於1可表示的最小浮點數的差值，約等於2.2204460492503130808472633361816E-16

MIN\_SAFE\_INTEGER屬性

可在JavaScript中安全表示的最小數字

句法:

Number.MIN\_SAFE\_INTEGER

返回值:

可在 JavaScript 中安全表示的最小數字，等於−9007199254740991

MAX\_SAFE\_INTEGER屬性

可在JavaScript中安全表示的最大數字

句法:

Number.MAX\_SAFE\_INTEGER

返回值:

可在JavaScript中安全表示的最大數字，等於 9007199254740991

數字方法Number Methods:

句法:

number.number\_method

num.number\_method(簡寫)

isFinite()方法

確定一個類型為Number的值是否為有限數

句法:

Number.isFinite(value)

參數值:

value(必須)判斷值

返回值:

如果值的類型為Number且為有限數，則返回true；否則，返回false

isInteger()方法

確定一個類型為Number的值是否為整數

句法:

Number.isInteger(value)

參數值:

value(必須)判斷值

返回值:

如果值的類型為Number且為整數，則返回true；否則，返回false

isNaN()方法

確定一個類型為Number的值是否為NaN

句法:

Number.isNaN(value)

參數值:

value(必須)判斷值

返回值:

如果值的類型為Number且為NaN，則返回true；否則，返回false

isSafeInteger()方法

確定一個類型為Number的值是否為安全整數(2\*\*53-1至-2\*\*53-1間所有整數)

句法:

Number.isSafeInteger(value)

參數值:

value(必須)判斷值

返回值:

如果值的類型為Number且為安全整數，則返回true；否則，返回false

toExponential()方法

將數字轉換為以指數符號表示的字符串

句法:

number.toExponential(x)

參數值:

x(可選) 0至20之間任意整數，代表四捨五入至小數點後幾位數

返回值:

以指數符號表示的字符串

toFixed()方法

將數字轉換為四捨五入的字符串

句法:

number.toFixed(x)

參數值:

x(可選)代表四捨五入至小數點後幾位數，默認為0，即四捨五入至整數

返回值:

一個帶有精確小數位數數字的字符串

toPrecision()方法

將數字轉換為指定長度的字符串

句法:

number.toPrecision(x)

參數值:

x(可選)指定長度

返回值:

指定長度的字符串

toString()方法

將數字轉換為字符串

句法:

num.toString(radix)

參數值:

radix (可選)2至36之間任意整數，代表用幾進制轉換

返回值:

一個代表0數字的字符串

valueOf()方法

返回數字的原始值(typeof = object至typeof = number)

句法:

number.valueOf()

參數值:

無

返回值:

一個數字的原始值

JavaScript數組Array

在JavaScript中，數組使用編號索引(元素名稱為數字)，JavaScript不支持關聯數組(哈希)

數組用方括號括起來，數組項用逗號分隔，是一種特殊的Object，用於在單個變量中存儲多個值

句法:

var array\_name = [item1, item2, ...];

數組索引從零開始，意味著第一項是[0]

可以在數組中包含object、fuction、array

數組屬性Array Property:

length屬性

設置或返回數組中元素的數量

句法:

返回數組的長度

array.length

設置數組的長度

array.length = number

返回值:

一個代表數組的長度(數組中元素的數量)的數字，始終為最高數組索引+1

數組方法Array Methods:

當需要原始值時，JavaScript會自動將數組轉換為逗號分隔的字符串

將數組轉換為字符串:

toString()方法

返回一個包含所有數組值的字符串，用逗號分隔

句法:

array.toString()

參數:

無

返回值:

一個包含所有數組值的字符串，用逗號分隔

join()方法

返回一個包含所有數組值的字符串，用指定的分隔符分隔

句法:

array.join(separator)

參數值:

separator (可選)指定的分隔符，默認是逗號

返回值:

一個包含所有數組值的字符串，用指定的分隔符分隔

從數組中彈出或推入項目:

pop()方法

刪除數組的最後一個元素，並返回該刪除元素，此方法會更改現有數組的長度

句法:

array.pop()

參數:

無

返回值:

任何類型的刪除數組元素

push()方法

將新元素添加到數組的末尾，並返回新的長度，此方法會更改現有數組的長度

句法:

array.push(item1, item2, ..., itemX)

參數值:

item1, item2, ..., itemX(必須)數組的添加元素

返回值:

一個代表數組新長度的Number

shift()方法

刪除數組的第一個元素(其他元素會移位到較低的索引)，並返回該刪除元素，此方法會更改現有數組的長度

句法:

array.shift()

參數:

無

返回值:

任何類型的刪除數組元素

unshift()方法

將新元素添加到數組的開頭，並返回新的長度，此方法會更改現有數組的長度

句法:

array.unshift(item1, item2, ..., itemX)

參數值:

item1, item2, ..., itemX(必須)數組的添加元素

返回值:

一個代表數組新長度的Number

splice()方法

在數組中添加/刪除元素，並返回一個包含已刪除元素的數組，此方法會更改現有數組的長度

句法:

array.splice(index, howmany, item1, ....., itemX)

參數值:

index(必須)開始添加/刪除元素的位置，使用負數可以從末尾進行

howmany(可選)應被刪除元素的數量

item1, ..., itemX(可選)數組的添加元素

返回值:

一個包含已刪除元素的數組

連接數組:

concat()方法

用於連接兩個或多個數組，返回一個包含聯接數組值的新數組，此方法不會更改現有數組

句法:

array1.concat(array2, array3, ..., arrayX)

參數值:

array2, array3, ..., arrayX (必須)添加的數組

返回值:

一個包含聯接數組值的新數組

分割數組:

slice()方法

將數組中選取的元素作為新數組返回，此方法不會更改現有數組

句法:

array.slice(start, end)

參數值:

start (可選)開始選取位置，默認為0，使用負數可以從末尾進行

end (可選)結束選取位置(不包含在選取的元素)，使用負數可以從末尾進行

返回值:

一個包含選定元素的新數組

數組排序:

sort()方法

按照字母順序(字符編碼順序)對數組進行升序排序，如果數字按字符串排序，則25大於100，因為2大於1，此方法將更改原始數組

句法:

array.sort(compareFunction)

參數值:

compareFunction(可選)規定排序順序的函數

例:

升序(由小至大)數字排序

function(a, b){return a-b}

若a < b，則返回值(a-b)為負值，a將會出現在b之前

若a = b，則返回值(a-b)為0

若a > b，則返回值(a-b)為正值，a將會出現在b之後

降序(由大至小)數字排序

function(a, b){return b-a}

若a < b，則返回值(b-a)為正值，b將會出現在a之前(a將會出現在b之後)

若a = b，則返回值(b-a)為0

若a > b，則返回值(b-a)為負值，b將會出現在a之後(a將會出現在b之前)

返回值:

一個已排序的Array

reverse()方法

反轉數組中元素的順序，此方法將更改原始數組

句法:

array.reverse()

參數值:

無

返回值:

一個反轉後的數組

數組疊代:

forEach()方法

按順序對數組中的每個元素都調用一次函數，不對沒有值的數組元素執行該函數

句法:

array.forEach(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

undefined

map()方法

按順序對數組中的每個元素都調用一次函數，並將函數的結果儲存在一個新數組，不對沒有值的數組元素執行該函數

句法:

array.map(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

一個數組，其中包含為原始數組中的每個元素調用一次函數的結果

filter()方法

創建一個具有通過測試函數的數組元素的新數組，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.filter(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

一個包含所有通過測試的數組元素的數組，如果沒有元素通過測試，將返回一個空數組

reduce()方法

將數組減少為單個值，為數組的每個值(從左到右)執行提供的函數，函數的返回值存儲在累加器(total)中，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.reduce(function(total, currentValue, currentIndex, arr), initialValue)

參數值:

function(total,currentValue, index,arr)(必須)每個元素都調用的一次函數

total(必須) initialValue或之前函數返回值

currentValue(必須)當前元素的值

currentIndex(可選)當前元素的索引，如果有initialValue，則由索引0開始；否則自索引1開始

arr(可選)當前元素的數組

initialValue(可選)函數初始值，如果沒有initialValue，則第一個元素將成為初始的total

返回值:

最後一次呼叫回呼函式的回傳值

reduceRight()方法

將數組減少為單個值，為數組的每個值(從右到左)執行提供的函數，函數的返回值存儲在累加器(total)中，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.reduceRight(function(total, currentValue, currentIndex, arr), initialValue)

function(total,currentValue, index,arr)(必須)每個元素都調用的一次函數

total(必須) initialValue或之前函數返回值

currentValue(必須)當前元素的值

currentIndex(可選)當前元素的索引，如果有initialValue，則由索引0開始；否則自索引1開始

arr(可選)當前元素的數組

initialValue(可選)函數初始值，如果沒有initialValue，則第一個元素將成為初始的total

返回值:

最後一次呼叫回呼函式的回傳值

every()方法

檢查數組中的所有元素是否都通過測試函數，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.every(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

如果有數組元素不通過，則返回false，並且不檢查其餘值；如果皆通過，則返回true(全部為true才為true)

some()方法

檢查數組中的任意元素是否通過測試函數，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.some(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

如果有數組元素通過，則返回true，並且不檢查其餘值；如果皆不通過，則返回false (全部為false才為false)

indexOf()方法

在數組中(從頭到尾)搜索指定的項目，並返回其第一次出現的位置

句法:

array.indexOf(item, start)

參數值:

item(必須)指定搜索項目

start (可選)開始搜索位置，默認從開頭開始，使用負數可以從末尾進行

返回值:

一個指定項目位置的Number，如果找不到該項目，則返回-1，如果該項目多次出現，則返回第一次出現的位置

lastIndexOf()方法

在數組中(從尾到頭)搜索指定的項目，並返回其最後一次出現的位置

句法:

array.lastIndexOf(item, start)

參數值:

item(必須)指定搜索項目

start (可選)開始搜索位置，默認從結尾開始，使用負數可以從末尾進行

返回值:

一個指定項目位置的Number，如果找不到該項目，則返回-1，如果該項目多次出現，則返回最後一次出現的位置

find()方法

返回通過測試函數的數組中第一個元素的值，不會對空數組執行該函數，不會更改原始數組

句法:

array.find(function(currentValue, index, arr),thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

如果有數組元素通過，則返回該數組元素的值，並且不檢查其餘值；如果皆不通過，則返回undefined

findIndex()方法

返回通過測試函數的數組中第一個元素的位置索引，不會對沒有值的數組元素執行該函數，不會更改原始數組

句法:

array.findIndex(function(currentValue, index, arr), thisValue)

參數值:

function(currentValue, index, arr)(必須)每個元素都調用的一次函數

currentValue(必須)當前元素的值

index(可選)當前元素的索引

arr(可選)當前元素的數組

thisValue(可選)傳遞給函數的值，用於當作this值，如果這個參數為空，undefined會傳遞給函數當作this值

返回值:

如果有數組元素通過，則返回該數組元素的位置索引，並且不檢查其餘值；如果皆不通過，則返回-1

其餘方法:

copyWithin()方法

將數組元素複製到數組中的另一個位置，從而覆蓋現有值，不會將更多項目添加到數組中，會覆蓋原始數組

句法:

array.copyWithin(target, start, end)

參數值:

target(必須)將數組元素複製到的位置索引

start(可選)數組元素開始複製的位置索引，默認為0

end(可選)數組元素結束複製的位置索引，默認為array.length(數組最高索引+1)

返回值:

一個更改後的數組

entries()方法

返回帶有鍵/值對的數組疊代器對象，不會更改原始數組

句法:

array.entries()

參數值:

無

返回值:

數組疊代器對象

fill()方法

用靜態值填充數組中的指定元素，會覆蓋原始數組

可以指定開始和結束填充的位置。如果未指定，則將填充所有元素

句法:

array.fill(value, start, end)

參數值:

value(必須)填充的靜態值

start(可選)靜態值開始填充的位置索引，默認為0

end(可選)靜態值結束填充的位置索引，默認為array.length(數組最高索引+1)

返回值:

一個更改後的數組

from()方法

從具有length屬性或可疊代對象的任意對象object返回數組對象array

句法:

Array.from(object, mapFunction, thisValue)

參數值:

object(必須)轉換的對象object

mapFunction(可選)一個用於調用數組中的每個元素的映射函數map function

thisValue(可選)執行映射函數map function時用作this值

返回值:

數組對象

includes()方法

確定數組是否包含指定的元素，大小寫敏感

句法:

array.includes(element, start)

參數值:

element(必須)指定搜尋的元素

start(可選)開始搜尋的位置索引，默認為0

返回值:

如果數組包含元素，則返回true；否則，返回false

isArray()方法

確定對像是否為數組

句法:

Array.isArray(obj)

參數值:

obj(必須)測試的對象object

返回值:

如果對像是數組，則返回true；否則，返回false

keys()方法

返回帶有數組鍵的數組疊代器對象，不會更改原始數組

句法:

array.keys()

參數值:

無

返回值:

數組迭代器對象

valueOf()方法

返回數組的原始值(typeof = object至typeof = object)，將返回與數組相同的值，是數組對象的默認方法，不會更改原始數組

句法:

array.valueOf()

參數值:

無

返回值:

返回數組自身

JavaScript日期對象Date Object

日期對像是靜態的，一月是0，十二月是11，JavaScript將日期存儲為自1970年1月1日UTC(世界協調時間)以來的毫秒數(零時間是1970年1月1日)

四種創建新日期對象的方法:

1.使用當前日期和時間創建

new Date()

2.使用指定日期和時間創建，7個數字按順序分別為年、月、日、小時、分鐘、秒、毫秒，一、二位數的年份將解釋為19xx年

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

3.使用零時間加上毫秒創建

new Date(milliseconds)

4.從日期字符串創建

new Date(date string)

日期格式

通常有3種類型的JavaScript日期輸入格式:

1.ISO Date(ISO-8601)遵循JavaScript中的嚴格標準strict standard

2. Short Date "MM / DD / YYYY"("03/25/2015")

3. Long Date "MMM DD YYYY"("Mar 25 2015")or"DD MMM YYYY"("25 Mar 2015")

JavaScript默認以全文字符串格式輸出日期:

Wed Mar 25 2015 08:00:00 GMT+0800 (台北標準時間)

UTC(世界協調時間)與GMT(格林威治標準時間)相同

GMT是指位於本初子午經線上的英國倫敦郊區皇家格林威治天文台當地的平太陽時

原子鐘報時的UTC已經取代天文觀測的GMT

ISO-8601

表示日期和時間的國際標準

YYYY-MM-DDTHH:mm:ss.sssZ

日期和時間用T分隔，UTC時間以Z定義

如果要修改相對於UTC的時間，請刪除Z並添加+ HH:mm或-HH:mm代替

日期對象屬性

日期對象方法

顯示日期:

toString()方法

將日期對象轉換為字符串，JavaScript需要將Date Object顯示為String會自動調用此方法

句法:

Date.toString()

參數:

無

返回值:

一個日期和時間的字符串

toDateString()方法

將日期對象的日期轉換為字符串

句法:

Date.toDateString()

參數:

無

返回值:

一個日期的字符串

toTimeString()方法

將日期對象的時間轉換為字符串

句法:

Date.toTimeString()

參數:

無

返回值:

一個時間的字符串

toUTCString()方法

根據世界通用時間(UTC)將日期對象轉換為字符串

句法:

Date.toUTCString()

參數:

無

返回值:

一個UTC日期和時間的字符串

toISOString()方法

使用ISO標準將日期對象轉換為字符串，ISO-8601(YYYY-MM-DDTHH:mm:ss.sssZ)

句法

Date.toISOString()

參數:

無

返回值:

一個ISO標準格式日期和時間的字符串

toJSON()方法

使用JSON標準將日期對象轉換為字符串，JSON日期與ISO-8601(YYYY-MM-DDTHH:mm:ss.sssZ)標準相同

句法:

Date.toJSON()

參數:

無

返回值:

一個JSON標準格式日期和時間的字符串

toLocaleString()方法

使用語言環境設置將日期對象轉換為字符串，默認語言取決於計算機上的區域設置

句法:

Date.toLocaleString(locales, options)

參數值:

locales(可選)語言格式

ar-SA Arabic (Saudi Arabia)

bn-BD Bangla (Bangladesh)

bn-IN Bangla (India)

cs-CZ Czech (Czech Republic)

da-DK Danish (Denmark)

de-AT Austrian German

de-CH "Swiss" German

de-DE Standard German (as spoken in Germany)

el-GR Modern Greek

en-AU Australian English

en-CA Canadian English

en-GB British English

en-IE Irish English

en-IN Indian English

en-NZ New Zealand English

en-US US English

en-ZA English (South Africa)

es-AR Argentine Spanish

es-CL Chilean Spanish

es-CO Colombian Spanish

es-ES Castilian Spanish (as spoken in Central-Northern Spain)

es-MX Mexican Spanish

es-US American Spanish

fi-FI Finnish (Finland)

fr-BE Belgian French

fr-CA Canadian French

fr-CH "Swiss" French

fr-FR Standard French (especially in France)

he-IL Hebrew (Israel)

hi-IN Hindi (India)

hu-HU Hungarian (Hungary)

id-ID Indonesian (Indonesia)

it-CH "Swiss" Italian

it-IT Standard Italian (as spoken in Italy)

ja-JP Japanese (Japan)

ko-KR Korean (Republic of Korea)

nl-BE Belgian Dutch

nl-NL Standard Dutch (as spoken in The Netherlands)

no-NO Norwegian (Norway)

pl-PL Polish (Poland)

pt-BR Brazilian Portuguese

pt-PT European Portuguese (as written and spoken in Portugal)

ro-RO Romanian (Romania)

ru-RU Russian (Russian Federation)

sk-SK Slovak (Slovakia)

sv-SE Swedish (Sweden)

ta-IN Indian Tamil

ta-LK Sri Lankan Tamil

th-TH Thai (Thailand)

tr-TR Turkish (Turkey)

zh-CN Mainland China, simplified characters

zh-HK Hong Kong, traditional characters

zh-TW Taiwan, traditional characters

options(可選)設置屬性

|  |  |
| --- | --- |
| dateStyle | "full" "long" "medium" "short" |
| timeStyle | "full" "long" "medium" "short" |
| localeMatcher | "best-fit" (默認) "lookup" |
| timeZone |  |
| hour12 | false true |
| hourCycle | "h11" "h12" "h23" "h24" |
| formatMatcher | "basic" "best-fit" (默認) |
| weekday | "long" "short" "narrow" |
| year | "2-digit" "numeric" |
| month | "2-digit" "long" "narrow" "numeric" "short" |
| day | "2-digit" "long" |
| hour | "2-digit" "long" |
| minute | "2-digit" "long" |
| second | "2-digit" "long" |
| timeZoneName | "long" "short" |

返回值:

一個日期和時間的字符串

toLocaleDateString()方法

使用區域設置約定將日期對象的日期轉換為字符串

句法:

Date.toLocaleDateString()

返回值:

一個日期的字符串

toLocaleTimeString()方法

使用區域設置約定將日期對象的時間轉換為字符串

句法:

Date.toLocaleTimeString()

返回值:

一個時間的字符串

parse()方法

解析日期字符串，並返回日期字符串與1970年1月1日午夜之間的毫秒數

句法:

Date.parse(datestring)

參數值:

datestring(必須)指定的日期時間

返回值:

一個指定的日期時間和1970年1月1日午夜之間毫秒數的Number

獲取日期:

getTime()方法

返回1970年1月1日午夜到現在之間的毫秒數

句法:

Date.getTime()

參數值:

無

返回值:

一個自1970年1月1日午夜以來毫秒數的Number

getFullYear()方法

以四位數的數字返回指定日期的年份(1000年至9999年之間的日期為四位數)

句法:

Date.getFullYear()

參數值:

無

返回值:

一個指定日期年份的Number

getMonth()方法

根據本地時間返回指定日期的月份(0到11)

句法:

Date.getMonth()

參數值:

無

返回值:

一個指定日期月份的Number(0到11)

getDate()方法

返回指定日期的天數(1到31)

句法:

Date.getDate()

參數值:

無

返回值:

一個指定日期天數的Number(1到31)

getHours()方法

返回指定日期和時間的小時(0到23)

句法

Date.getHours()

參數值:

無

返回值:

一個指定日期和時間小時的Number(0到23)

getMinutes()方法

返回指定日期和時間的分鐘(0到59)

句法:

Date.getMinutes()

參數值:

無

返回值:

一個指定日期和時間分鐘的Number(0到59)

getSeconds()方法

返回指定日期和時間的秒數(0到59)

句法:

Date.getSeconds()

參數值:

無

返回值:

一個指定日期和時間秒數的Number(0到59)

getMilliseconds()方法

返回指定日期和時間的毫秒數(0到999)

句法:

Date.getMilliseconds()

參數值:

無

返回值:

一個指定日期和時間毫秒數的Number(0到999)

getDay()方法

返回指定日期的星期幾(0到6)

句法:

Date.getDay()

參數值:

無

返回值:

一個指定日期和時間星期幾的Number(0到6)，星期日Sunday為0

獲取UTC日期:

在假定日期對象具有本地時間和日期的情況下計算其日期

getUTCDate()等同UTC版的getDate()

getUTCDay()等同UTC版的getDay()

getUTCFullYear()等同UTC版的getFullYear()

getUTCHours() 等同UTC版的getHours()

getUTCMilliseconds()等同UTC版的getMilliseconds()

getUTCMinutes()等同UTC版的getMinutes()

getUTCMonth()等同UTC版的getMonth()

getUTCSeconds()等同UTC版的getSeconds()

設置日期:

setFullYear()方法

設置日期對象的年份(1000年至9999年之間的日期為四位數)，還可用於設置月份和天數

句法:

Date.setFullYear(year, month, day)

參數值:

year(必須)指定的年份，可以是負數

month(可選)指定的月份，0至11為今年月份，-1為去年12月

day(可選)指定的天數，1至31為本月天數，0為上個月最後一天

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setMonth()方法

設置日期對象的月份，還可用於設置每月的天數

句法:

Date.setMonth(month, day)

參數值:

month(必須)指定的月份，0至11為今年月份，-1為去年12月

day(可選)指定的天數，1至31為本月天數，0為上個月最後一天

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setDate()方法

設置日期對象的天數

句法:

Date.setDate(day)

day(必須)指定的天數，1至31為本月天數，0為上個月最後一天

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setHours()方法

設置日期對象的小時，還可用於設置分鐘，秒和毫秒

句法:

Date.setHours(hour, min, sec, millisec)

參數值:

hour (必須)指定的小時，0至23為本日時數，-1為昨天24時

min(可選)指定的分鐘，0至59為本小時分鐘，-1為上個小時60分

sec(可選)指定的秒數，0至59為本分鐘秒數，-1為上個分鐘60秒

millisec(可選)指定的毫秒數，0至999為本秒毫秒數，-1為上個秒1000毫秒

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setMinutes()方法

設置日期對象的分鐘，還可以用於設置秒和毫秒

句法:

Date.setMinutes(min, sec, millisec)

參數值:

min(必須)指定的分鐘，0至59為本小時分鐘，-1為上個小時60分

sec(可選)指定的秒數，0至59為本分鐘秒數，-1為上個分鐘60秒

millisec(可選)指定的毫秒數，0至999為本秒毫秒數，-1為上個秒1000毫秒

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setSeconds()方法

設置日期對象的秒數，還可以用於設置毫秒

句法:

Date.setSeconds(sec, millisec)

參數值:

sec(必須)指定的秒數，0至59為本分鐘秒數，-1為上個分鐘60秒

millisec(可選)指定的毫秒數，0至999為本秒毫秒數，-1為上個秒1000毫秒

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setMilliseconds()方法

設置日期對象的毫秒數

句法:

Date.setMilliseconds(millisec)

參數值:

millisec(必須)指定的毫秒數，0至999為本秒毫秒數，-1為上個秒1000毫秒

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

setTime()方法

通過在1970年1月1日午夜前後增加或減去指定的毫秒數來設置日期和時間

句法

Date.setTime(millisec)

參數值:

millisec(必須)指定與1970年1月1日午夜之間的毫秒數

返回值:

一個日期對象與1970年1月1日午夜之間毫秒數的Number

設置UTC日期:

setUTCDate()等同UTC版的setDate()

setUTCFullYear()等同UTC版的setFullYear()

setUTCHours() 等同UTC版的setHours()

setUTCMilliseconds()等同UTC版的setMilliseconds()

setUTCMinutes()等同UTC版的setMinutes()

setUTCMonth()等同UTC版的setMonth()

setUTCSeconds()等同UTC版的setSeconds()

其餘方法:

getTimezoneOffset()方法

以分鐘為單位返回UTC時間與本地時間之間的時差

句法:

Date.getTimezoneOffset()

參數值:

無

返回值:

一個UTC時間與本地時間之間的時差分鐘數的Number

now()方法

根據UTC返回自1970年1月1日以來的毫秒數

句法:

Date.now()

參數值:

無

返回值:

一個根據UTC自1970年1月1日午夜以來毫秒數的Number

UTC()方法

根據UTC返回指定日期與1970年1月1日午夜之間的毫秒數

句法:

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

參數值:

year(必須)指定的年份(四位數)，可以是負數

month(必須)指定的月份，0至11為今年月份，-1為去年12月

day(可選)指定的天數，1至31為本月天數，0為上個月最後一天

hour (可選)指定的小時，0至23為本日時數，-1為昨天24時

min(可選)指定的分鐘，0至59為本小時分鐘，-1為上個小時60分

sec(可選)指定的秒數，0至59為本分鐘秒數，-1為上個分鐘60秒

millisec(可選)指定的毫秒數，0至999為本秒毫秒數，-1為上個秒1000毫秒

返回值:

一個指定的日期時間和1970年1月1日午夜之間毫秒數的Number

valueOf()方法

返回日期對象的原始值，返回的原始值是自UTC 1970年1月1日午夜以來的毫秒數

句法:

Date.valueOf()

參數值:

無

返回值:

一個日期對象和1970年1月1日午夜之間毫秒數的Number

JavaScript數學對象Math Object

Math對像沒有構造函數，方法和屬性是靜態的，無需先創建Math對象即可使用所有方法和屬性(常數)

數學對象屬性(常數):

E屬性

返回e(自然對數函數的底數、尤拉數)，大約為2.718

句法:

Math.E

返回值:

一個e的Number

LN2屬性

返回ln 2，ln(自然對數(以e為底數的對數函數)) ，大約為0.693

句法:

Math.LN2

返回值:

一個ln 2的Number

LN10屬性

返回ln 10，ln(自然對數(以e為底數的對數函數)) ，大約為2.302

句法:

Math.LN10

返回值:

一個ln 10的Number

LOG2E屬性

返回log2 e，log(對數(指數運算的逆運算))，大約為1.442

句法:

Math.LOG2E

返回值:

一個log2 e的Number

LOG10E屬性

返回log e，大約為0.434

句法:

Math.LOG10E

返回值:

一個log e的Number

PI屬性

返回π(圓的面積與其半徑的平方之比)，大約為3.14

句法:

Math.PI

返回值:

一個π的Number

SQRT1\_2屬性

返回1/2的平方根，大約為0.707

句法:

Math.SQRT1\_2

返回值:

一個1/2平方根的Number

SQRT2屬性

返回2的平方根，約為1.414

句法:

Math.SQRT2

返回值:

一個2平方根的Number

數學對象方法:

round()方法

將數字四捨五入到最接近的整數

句法:

Math.round(x)

參數值:

x(必須)指定值

返回值:

一個四捨五入到最接近整數的Number

pow()方法

返回x^y

句法:

Math.pow(x, y)

參數值:

x(必須)底數

y(必須)指數

返回值:

一個x乘以y冪(次方)值的Number

sqrt()方法

返回數字的平方根

句法:

Math.sqrt(x)

參數值:

x(必須)指定值

返回值:

一個平方根的Number，如果x為負數，則返回NaN

abs()方法

返回數字的絕對值

句法

Math.abs(x)

參數值:

x(必須)指定值

返回值:

一個指定數字絕對值的Numbe；如果該值不是數字，則為NaN；如果該值為null，則為0

ceil()方法

將數字無條件進位到最接近的整數，如果是整數，則將不會進位

句法:

Math.ceil(x)

參數值:

x(必須)指定值

返回值:

一個無條件進位到最接近整數的Number

floor()方法

將數字無條件退位到最接近的整數，如果是整數，則將不會進位

句法:

Math.floor(x)

參數值:

x(必須)指定值

返回值:

一個無條件退位到最接近整數的Number

sin()方法

返回數字的正弦值

句法:

Math.sin(x)

參數值:

x(必須)指定值

返回值:

sin x，介於-1和1之間；如果無參數，則為NaN

cos()方法

返回數字的餘弦值

句法:

Math.cos(x)

參數值:

x(必須)指定值

返回值:

cos x，介於-1和1之間；如果無參數，則為NaN

max()方法

返回最大值

句法:

Math.max(n1, n2, n3, ..., nX)

參數值:

n1, n2, n3, ..., nX(可選)比較值

返回值:

一個最大值的Number；如果沒有給出參數，則為-Infinity；如果一個或多個參數不是數字，則為NaN

min()方法

返回最小值

句法:

Math.min(n1, n2, n3, ..., nX)

參數值:

n1, n2, n3, ..., nX(可選)比較值

返回值:

一個最小值的Number；如果沒有給出參數，則為Infinity；如果一個或多個參數不是數字，則為NaN

random()方法

返回一個從0(包括)到1(不包括)的隨機數

句法:

Math.random()

參數值:

無

返回值:

一個從0到1，但不包括1的Number

acos()方法

返回數字的反餘弦值(arccos x、cos-1 x)，介於0和PI之間，如果參數x超出-1到1，則返回NaN

acosh()方法

返回數字的雙曲反餘弦值(arccosh x、cosh-1 x)，如果參數x小於1，則返回NaN

asin()方法

返回數字的反正弦值(arcsin x、sin-1 x)，介於-PI / 2和PI / 2之間，如果參數x超出-1到1，則返回NaN

asinh()方法

返回數字的雙曲反正弦值(arcsinh x、sinh-1 x)

atan()方法

返回數字的反正切值(arctan x、tan-1 x)，介於-PI / 2和PI / 2之間，如果沒有參數，則為NaN

atan2()方法

返回其參數商的反正切值(arctan y/x、tan-1 y/x)，介於PI和-PI之間，返回的數字表示正X軸和點(x，y)之間以弧度(不是度)為單位的逆時針角度，y坐標為第一個參數，x坐標為第二個參數

atanh()方法

返回數字的雙曲反正切，參數x大於1或小於-1，則該方法將返回NaN；如果參數x為1，則該方法將返回Infinity；如果參數x為-1，則該方法將返回-Infinity

cbrt()方法

返回數字的立方根

clz32()方法

CountLeadingZeroes32的縮寫，返回以數字的32位二進製表示形式中的前導零位的數量(如果數字為0，則此方法返回32，因為所有位均為0)

cosh()方法

返回數字的雙曲餘弦值。

exp()方法

返回e^x

expm1()方法

返回e^x - 1

fround()方法

返回數字的最接近的32位單精度浮點表示形式

log()方法

返回loge x

log10()方法

返回log10 x

log1p()方法

返回1 + loge x

log2()方法

返回log2 x

sign()方法檢查數字是負數，正數還是零。

如果數字為正，則返回1

如果數字為負，則返回-1

如果數字為正零，則返回0

如果數字為負零，則返回-0

如果數字不是數字，則返回NaN

sinh()方法

返回數字的雙曲正弦值

tan()方法

返回數字的正切值

tanh()方法

返回數字的雙曲正切值

trunk()方法

刪除數字的小數部分，返回數字的整數部分

JavaScript布爾值Booleans

可以具有true或 false值

Boolean()函數

可以用來查找表達式或變量是否為真

不要使用布爾對象將非布爾值轉換為布爾值，而是使用Boolean作為執行此任務的函數

var x = Boolean(expression); // preferred

var x = new Boolean(expression); // don't use

句法:

Boolean

Boolean(value)

參數值:

value(可選)Boolean對象的初始值

一切帶有Value的東西都是True:

任何非空字符串都為true(包括字符串'false')

任何表達式都是true(0除外)

一切沒有Value的東西都是False:

0的布爾值為false

-0的布爾值為false

””空字符串的布爾值為false

undefined的布爾值是false

null的布爾值是false

false的布爾值是false

NaN的布爾值是false

布爾屬性

布爾方法

toString()方法

返回一個布爾值的字符串，在字符串操作中使用布爾值時，JavaScript都會自動調用此方法

句法:

boolean.toString()

參數值:

無

返回值:

“true”或”false”

valueOf()方法

返回布爾值的原始值，通常由JavaScript在幕後自動調用，而不是在代碼中明確調用

句法:

boolean.valueOf()

參數值:

無

返回值:

true或false

JavaScript條件語句Conditional Statements

if語句

句法

if (condition) {

//如果條件為true時，要執行的代碼塊

}

if (condition) {

//如果條件為true時，要執行的代碼塊

} else {

//如果條件為false時，要執行的代碼塊

}

if (condition1) {

//如果條件1為true時，要執行的代碼塊

} else if (condition2) {

//如果條件1為false且條件2為true時，要執行的代碼塊

} else {

//如果條件1為false且條件2為false時，要執行的代碼塊

}

Switch語句

句法

switch(expression) {

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

}

break關鍵字

跳出switch或loop

default關鍵字

指定如果沒有匹配的情況下應執行的代碼

如果多個匹配，則選擇第一個

JavaScript循環Loop

continue關鍵字

在loop中跳過一個疊代

For循環

多次遍歷代碼塊

句法

for (statement 1; statement 2; statement 3) {

// code block to be executed

}

statement 1在執行代碼塊之前執行一次

statement 2定義了執行代碼塊的條件

statement 3每次在執行代碼塊後都會執行

For / In循環

遍歷對象的屬性

句法

for (key in object) {

// code block to be executed

}

key每個鍵/值對的鍵(對象的屬性)

object對象

For / Of循環

遍歷可疊代對象的值

句法

for (variable of iterable) {

// code block to be executed

}

variable每個疊代的值

iterable一個具有一或多個鍵/值對且可以迭代的對象

一般對像不可迭代

const obj = { fname: 'foo', lname: 'bar' };

for (const value of obj) {

// TypeError: obj[Symbol.iterator] is not a function console.log(value);

}

While循環

只要條件為真，將重複該循環，遍歷代碼塊

句法

while (condition) {

// code block to be executed

}

Do / While循環

在檢查條件是否為真之前，此循環將執行一次代碼塊，然後只要條件為真，將重複該循環，遍歷代碼塊

句法

do {

// code block to be executed

}

while (condition);

JavaScript Labels

句法

labelname:

statements

//break帶有Label引用，用於跳出任何代碼塊

break labelname;

//continue帶有Label引用，用於跳過一個循環疊代

continue labelname;

constructor屬性

返回所有JavaScript變量的構造函數，null和undefined是沒有constructor屬性

"John".constructor // Returns function String() {[native code]}

(3.14).constructor // Returns function Number() {[native code]}

false.constructor // Returns function Boolean() {[native code]}

[1,2,3,4].constructor // Returns function Array() {[native code]}

{name:'John',age:34}.constructor // Returns function Object() {[native code]}

new Date().constructor // Returns function Date() {[native code]}

function () {}.constructor // Returns function Function(){[native code]}

JavaScript正則表達式Regular Expression

形成搜索格式的一系列字符，搜索格式可以用來描述要搜索的內容

句法

/pattern/modifiers;

/pattern1| pattern2/modifiers; // pattern1或pattern2

正則表達式修飾符Modifiers

g執行全局匹配(查找所有匹配，而不是在第一個匹配之後停止)

i執行不區分大小寫的匹配

m執行多行匹配，會變更 ^ 和 $ 的解譯，以比對每一行字符串(由\ n或\ r分隔)的開頭和結尾，而不是字符串的開頭和結尾

正則表達式格式Pattern

方括號Bracket

用於查找一系列字符

[abcde ..] 查找方括號之間的任意字符

/b[aiu]g/等同bag、big、bug

[A-Z] 查找從大寫A到大寫Z的任意字符

[a-z] 查找從小寫a到小寫z的任意字符

[A-z] 查找從大寫字母A到小寫字母z的任意字符

[0-9] 查找從0到9的任意字符(任意數字)

//可以部份查詢，[a-e] 等同a到e，[0-5] 等同0到5，[a-e0-5] 等同a到e或0到5

[^abc] 查找不在方括號之間的任意字符

[^0-9] 查找不在0到9的任意字符(任意非數字)

(x|y) 查找任意指定的選項

元字符Metacharacters

具有特殊含義的字符

. 查找單個字符，除了換行符或行終止符

/hu./等同hug、huh、hut、hum/

\w 查找單詞字符，包括a-z、A-Z、0-9、\_(下劃線) ，等同[A-Za-z0-9\_]

\W 查找非單詞字符，等同[^A-Za-z0-9\_]

\d 查找0-9之間的數字(任意數字)，等同[0-9]

\D 查找非數字字符(任意非數字)，等同[^0-9]

\s 查找空格字符，包括空格字符space、製表符tab、回車符carriage return(enter)、換行符new line、垂直製表符vertical tab、換頁字符form feed，等同[ \r\t\f\n\v]

\S 查找非空白字符，等同[^ \r\t\f\n\v]

\b 在單詞的開頭(\bLO)或結尾(LO\b)查找匹配項

\B 查找匹配項，但不位於單詞開頭(\BLO)或結尾(LO\B)

\0 查找NUL字符

\n 查找換行符

\f 查找換頁符

\r 查找回車符

\t 查找製表符

\v 查找垂直製表符

\xxx 查找由八進制數字xxx指定的拉丁字符

\xdd 查找由十六進制數字dd指定的拉丁字符

\uxxxx 查找由十六進制數字dddd指定的Unicode字符

量詞Quantifier

定義數量

n+ 匹配任何包含一個或多個n的字符串

/c+/等同cc、ccc

n\* 匹配任何包含零個或多個n的字符串

n? 匹配任何包含零個或一個n的字符串

n{X} 匹配X個n的序列字符串

n{X,Y} 匹配下限為X個(包含)、上限為Y個(包含) n的序列字符串

n{X,} 匹配下限為X個(包含) n的序列字符串

n$ 匹配任何結尾為n的字符串

^n 匹配任何開頭為n的字符串

?=n 匹配任何其後緊接指定字符串n的字符串

/q(?=u)/，qu符合，返回q；qk不符合

?!n 匹配任何其後沒有緊接指定字符串n的字符串

/a(?!t)/，au符合，返回a；at不符合

正則表達式對象RegExp Object

正則表達式對象屬性

constructor屬性 Returns the function that created the RegExp object's prototype

global屬性 正則表達式對像是否具有g修飾符，是則返回true；否則返回false

ignoreCase屬性 正則表達式對像是否具有i修飾符，是則返回true；否則返回false

lastIndex屬性 指定開始下一個匹配的索引，設置 g修飾符時，此屬性才有效

multiline屬性 正則表達式對像是否具有m修飾符，是則返回true；否則返回false

source屬性 返回RegExp pattern的文本

正則表達式對象方法

exec()方法

檢索字符串中的正則表達式的匹配項，如果找到匹配項，則返回匹配的文本，否則返回null

句法:

RegExpObject.exec(string)

參數值:

string(必須)檢索的字符串

返回值:

一個數組，其中存放匹配的結果(第一個匹配項)；如果未找到匹配，則返回值為null

test()方法

檢索字符串中的匹配項，如果找到匹配項，則返回true；否則返回false

句法:

RegExpObject.test(string)

參數值:

string(必須)檢索的字符串

返回值:

如果找到匹配項，則返回true；否則返回false

toString()方法

返回正則表達式的字符串

句法:

RegExpObject.toString()

參數值:

無

返回值:

一個正則表達式字符串的String

JavaScript錯誤

try語句定義一個代碼塊，該代碼塊在執行時將進行錯誤測試

catch如果在try代碼塊中發生錯誤，則該語句允許定義要執行的代碼塊

try和catch 成對出現

try {

Block of code to try

}

catch(err) {

Block of code to handle errors

}

throw an exception引發異常 (throw an error引發錯誤)

發生錯誤時，JavaScript通常會停止並生成錯誤消息

throw語句可以創建自定義錯誤消息

finally語句可以在try和catch之後執行代碼，而不管try和catch結果為何

finally {

Block of code to be executed regardless of the try / catch result

}

JavaScript錯誤對象

錯誤對象屬性

name屬性

設置或返回錯誤的名稱，可以返回六個不同的值:

EvalError 在eval() 函數中發生的錯誤，較新版本的JavaScript不會引發，請改用SyntaxError

RangeError 發生使用數字超出範圍的錯誤(將數字的有效位數設置為500)

ReferenceError 發生非法引用(使用尚未聲明的變量)

SyntaxError 發生語法錯誤

TypeError 發生類型錯誤

URIError 在encodeURI() 中發生的錯誤

message屬性

設置或返回錯誤消息(字符串)

JavaScript 提升Hoisting

是JavaScript的默認行為，即將所有聲明移到當前作用域(當前腳本或當前函數)

的頂部，適用於變量聲明和函數聲明，僅提升聲明，而不提升初始化(賦值)，未初始化的值為undefined

變量聲明

var聲明變量

變量會被提升，並且可以隨時初始化initialized(可以在聲明變量之前使用它)

let、const聲明變量

變量會被提升，但未初始化(代碼塊知道該變量，但是在聲明之前不能使用)，在聲明變量之前使用變量將導致ReferenceError，從塊開始到聲明為止，該變量位於臨時死區temporal dead zone

var x = 5;

未聲明變量，則嚴格模式下不允許使用變量

函數聲明

箭頭功能不會提升

函數聲明

函數可以在聲明之前調用

函數表達式聲明

函數不能在聲明之前調用

var x = function (a, b) {return a \* b};

JavaScript嚴格模式Strict Mode

"use strict"; //定義應在嚴格模式下執行JavaScript代碼

在腳本的開頭聲明，它具有全局作用域(腳本中的所有代碼將在嚴格模式下執行)

在函數內部聲明，它具有局部作用域(僅函數內部的代碼處於嚴格模式下)

不允許使用未聲明的變量(對像也是變量)

不允許刪除delete變量

不允許刪除函數

不允許重複函數參數名稱

不允許使用八進制數字文字

不允許使用八進制轉義字符

不允許寫入只讀read-only屬性

不允許寫入僅獲取get-only屬性

不允許刪除不可刪除的屬性

eval不能用作變量

arguments不能用作變量

with語句是不允許

eval()不允許在調用它的範圍內創建變量

this關鍵字是指調用該函數的對象，如果未指定對象，在嚴格模式下的函數將返回undefined；在正常模式下的函數將返回全局對象(window)

不能將未來JavaScript版本保留的關鍵字用作變量名，包括:

implements

interface

let

package

private

protected

public

static

yield

JavaScript異步Asynchronous

JavaScript是單線程

每個瀏覽器都有自己的JavaScript Runtime engine運行引擎，最著名的就是google chrome的V8 JavaScript引擎

異步函數

與其他函數並行運行的函數，例如JavaScript setTimeout()

1. JS Runtime engine運行至異步函數(setTimeout)

2.異步函數(setTimeout)由JS Runtime engine中Stack棧(後進先出、LIFO)至Web APIs，JS Runtime engine繼續運行

3.異步函數達成條件，由Web APIs至Callback Queue回調序列

4.確認Stack中沒有待執行的程式碼後，如果Callback Queue有準備好的Callback，Event Loop會將第一個準備好的Callback移動到Stack中執行

JavaScript回調Callback

JavaScript中函數是一級函數

一級函數

可以存儲在變量中、從一個函數返回、作為參數傳遞給另一個函數

高階函數

接受其他函數作為參數的函數

回調函數

作為參數傳遞給高階函數的函數，可以在高階函數完成後被調用和執行，最常將異步函數當作其高階函數一起使用

function myDisplayer(some) {

document.getElementById("demo").innerHTML = some;

}

function myCalculator(num1, num2, myCallback) {

let sum = num1 + num2;

myCallback(sum);

}

myCalculator(5, 5, myDisplayer); //將函數作為參數傳遞時，不要使用括號()

回調地獄callback hell

當使用回調執行多個異步操作時，會一次性寫多層的回調嵌套，導致難以讀取代碼

doSomething(function(result) {

doSomethingElse(result, function(newResult) {

doThirdThing(newResult, function(finalResult) {

console.log('Got the final result: ' + finalResult);

}, failureCallback);

}, failureCallback);

}, failureCallback);

JavaScript 承諾Promise

可以使用內置構造函數Promise()建立一個承諾對象(表示異步運算完成或失敗的對象)，建立同時必須傳入一個包含resolve、reject參數的函數作為參數，分別代表成功與失敗的回傳結果

承諾對象屬性(無法訪問)

myPromise.state和myPromise.result分別代表3種狀態:

"pending"和undefined等待承諾完成中

"fulfilled"和a result value承諾被兌現，使用.then()，繼續做預定好的下一件事

"rejected"和an error object承諾被打破，使用.catch()或.then的第二個參數，根據這個原因去做對應的動作

靜態方法

all()方法

返回一個Promise，作為輸入多個Promise後，結果數組的對象(匯總多個promise的結果)

句法:

Promise.all(iterable);

參數值:

iterable一個可疊代對象，例如Array

返回值:

當所有輸入的承諾都已兌現或輸入的可迭代的對象不包含任何承諾，返回的承諾將兌現；當任意輸入的承諾被打破，立即返回承諾，返回的承諾將被打破，並使用被打破的承諾結果

race()方法

返回一個Promise，作為輸入多個Promise後，第一個結果的對象(第一個運行完成的Promise的結果)

句法:

Promise.race(iterable);

參數值:

iterable一個可疊代對象，例如Array

返回值:

當任意輸入的承諾已兌現或被打破，立即返回承諾，返回的承諾將兌現或被打破，並使用已兌現或被打破的承諾結果

實例方法

then()方法

兩個參數分別用於的成功和失敗的回調函數，返回一個承諾對象，可以選擇將其用於鏈接

catch()方法

一個參數用於處理失敗的回調函數，返回一個承諾對象

finally()方法

一個參數用於處理成功或失敗的回調函數，返回一個承諾對象

//resolve 及 reject 的名稱可以自定義

function myPromise(x) {

return new Promise(function(resolve, reject) {

//生產代碼Producing code，需要一些時間的代碼

if (x == 0) {

resolve("success");//正確完成的回傳方法

} else {

reject("fail");//失敗的回傳方法

}

});

}

//消費代碼Consuming code是必須等待Producing code結果的代碼

myPromise(0)

.then(

function(success) {console.log(success);},

function(fail) {console.log(fail);}

);

承諾鏈Promise Chain

基於then()、catch()、finally()回傳的承諾對象，進行串接

// then、catch，如果運行正確則會繼續往下運行，當遭遇失敗則會跳到catch

myPromise(0)

.then(

function(success) {

console.log(success);

return myPromise(1); // 鏈接myPromise(1)

} // myPromise(0)成功

)

.catch(

function(fail) {

console.log(fail);

return myPromise(1); // 鏈接myPromise(1)

} // myPromise(0)失敗

)

.then(

function(success) {console.log(success);}, // myPromise(1)成功

function(fail) {console.log(fail);} // myPromise(1)失敗

);

Async、Await

基於Promise讓非同步的語法結構類似於同步語法

async function

用來定義一個異步函數，讓這個函式本體是屬於非同步，但其內部以同步的方式運行非同步程式碼

await

暫停異步函數的運行(中止Promise的運行)，直到異步進入resolve或reject，當接收完回傳值後繼續異步函數的運行

function myPromise(x) {

return new Promise(function(resolve, reject) {

//生產代碼Producing code，需要一些時間的代碼

if (x == 0) {

resolve("success");//正確完成的回傳方法

} else {

reject("fail");//失敗的回傳方法

}

});

}

//消費代碼Consuming code是必須等待Producing code結果的代碼

myPromise(0)

.then(

function(success) {console.log(success);}

)

.catch(

function(fail) {console.log(fail);}

);

等同

//用 try...catch管理流程，將原有的程式碼直接置入於try流程內，當遇到例外的錯誤時則撰寫在catch區塊內

async function getData() {

try { // 專注在成功

const data1 = await myPromise(0);

console.log(data1);

}

catch (err) { // 專注在錯誤

console.log(err);

}

}

getData();

JavaScript HTML DOM(文件物件模型Document Object Model)

當網頁加載時，瀏覽器會為網頁創建一個DOM(把一份HTML文件內的各個標籤，包括文字、圖片等等都定義成物件，而這些物件最終會形成一個樹狀結構)

通過DOM，JavaScript獲得了創建動態HTML所需的全部功能:

更改頁面中的所有HTML元素和屬性

更改頁面中的所有CSS樣式

刪除現有的HTML元素和屬性

添加新的HTML元素和屬性

對頁面中所有現有的HTML事件做出反應

在頁面中創建新的HTML事件

DOM是W3C標準，定義了用於訪問文檔的標準(處理網頁内容的方法和接口)

W3C DOM標準分為三個不同的部分:

(1)Core DOM適用於所有文檔類型的標準模型

(2)XML DOM適用於XML文檔的標準模型

(3)HTML DOM適用於HTML文檔的標準模型:

定義了HTML元素作為對象和所有HTML元素的屬性、訪問方法、事件(如何獲取，更改，添加或刪除HTML元素的標準)

DOM節點

在DOM中，節點通常分成以下五種:

Document Node文檔節點

整個文檔，是這份HTML檔的開端，所有的一切都會從Document開始往下進行

Element Node元素節點

各個HTML元素，文件內的各個HTML標籤Tag，像是<div>、<p>、…

Attribute Node屬性節點

各個HTML屬性，各個標籤內的相關屬性

Text Node文本節點

各個HTML元素內的文本，被各個標籤包起來的文字，例如在<h1>Hello</h1>中的Hello就是此Element的Text

Comment Node註釋節點

各個HTML註釋

節點關係，可以分成以下兩種:

父子關係Parent and Child

上下層節點，上層為Parent Node父節點，，每個節點都有父節點、除了根root (頂端節點)；下層為Child Node子節點，一個節點可擁有任意數量的子節點

兄弟關係Siblings

擁有相同父節點的節點，彼此間只有之前Previous以及之後Next兩種

節點屬性:

node.nodeName

返回節點的名稱

nodeName是只讀的

文檔節點的nodeName始終是#document

元素節點的nodeName與標籤名稱(大寫字母)相同

屬性節點的nodeName與屬性名稱相同

文本節點的nodeName始終是#text

node.nodeValue

返回或設置節點的值

文檔節點的nodeValue是null

元素節點的nodeValue是null

屬性節點的nodeValue是屬性值

文本節點的nodeValue是文本本身

node.nodeType

返回一個整數值，表示節點類型

比較重要的節點類型如下:

Element Node元素節點 (1) <h1 class="heading">W3Schools</h1>

Attribute Node屬性節點 (2) class = "heading" (不建議使用)

Text Node文本節點 (3) W3Schools

Comment Node註釋節點 (8) <!-- This is a comment -->

Document Node文檔節點 (9) HTML文檔本身 (<html>的父節點)

Document Type Node文檔類型節點(10) <!Doctype html>

節點導航:

node.parentNode

返回作為此節點的父級的Node；如果沒有，則返回null

node.childNodes[nodenumber]

返回NodeList，包含該節點所有子級即時更新的動態集合

node.firstChild

返回一個Node，表示節點的第一個直接子節點；如果沒有，則返回null node.lastChild

返回一個Node，表示節點的最後一個直接子節點；如果沒有，則返回null node.nextSibling

返回Siblings中下一個節點的Node；如果沒有，則返回null

node.previousSibling

返回Siblings中上一個節點的Node；如果沒有，則返回null

DOM根節點:

document.body

返回當前文件的<body>

document.documentElement

返回當前文件中的根元素，例如HTML文檔中的<html>

DOM操作

HTML DOM方法

可以對HTML元素做的的動作，例如添加或刪除HTML元素

HTML DOM屬性

HTML元素可以獲取、設置、更改的值，例如節點的名稱或內容

HTMLCollection

HTML元素的集合(具有即時性)，類似數組的對象列表(索引從0開始)

集合內元素為HTML element，Node type只能接受Element

屬性:

length

返回集合中的項目數

NodeList

文檔節點的集合(具有即時性)，類似數組的對象列表(索引從0開始) ，可能由getElementsByClassName()、Node.childNodes屬性、document.querySelectorAll()方法取得

集合內元素為Node，全部的Node都可以存放在NodeLists屬性:

length

返回集合中的項目數

查找HTML元素:

方法:

document.getElementById('idName')

找尋DOM中符合此id名稱的元素，並回傳相對應的element

document.getElementsBytagName('tagName')

找尋DOM中符合此標籤tag名稱的元素，並回傳所有相對應的element集合，集合為HTMLCollection

document.getElementsByClassName('className')

找尋DOM中符合此class名稱的元素，並回傳所有相對應的element集合，集合為HTMLCollection

document.querySelector('selector')

利用CSS選擇器selector來找尋DOM中的元素，並回傳相對應的第一個element

document.querySelectorAll('selector')

利用CSS選擇器selector來找尋DOM中的元素，並回傳所有相對應的element集合，集合為NodeList

更改HTML元素:

屬性:

element.innerHTML = new html content

設置或返回該HTML元素的HTML內容

element.attribute = new value

設置或返回該HTML元素的HTML屬性的值

element.style.property = new style

設置或返回該HTML元素的樣式

方法:

element.getAttribute(attribute)

返回該HTML元素的HTML屬性的值

element.setAttribute(attribute, value)

設置該HTML元素的HTML屬性的值

添加和刪除元素:

方法:

document.createElement(element)

創建元素節點

document.createAttribute()

創建屬性節點

document.createTextNode()

創建文本節點

node.remove()

從對象所屬的樹中刪除該對象(不支持remove()的瀏覽器，必須找到父節點才能使用removeChild()刪除元素)

document.removeChild(element)

刪除子節點

node.appendChild(element)

添加子節點到指定父節點末尾

node.insertBefore()

在指定子節點前面插入子節點

node.replaceChild(new, old)

替換子節點

document.write(text)

將文本字符串寫入由document.open()打開的文檔流中

JavaScript HTML DOM事件Event

HTML DOM允許JavaScript對HTML事件做出反應(事件發生時，執行JavaScript)

當事件觸發時，事件處理器被調用，例如觸發click event將調用onclick

load事件(onload)

加載完整個頁面(包括所有相關資源，例如樣式表和圖像)時觸發

onload可用於檢查訪問者的瀏覽器類型和瀏覽器版本，並根據該信息加載網頁的正確版本

unload事件(onunload)

卸載文檔或子資源時觸發

onload、onunload可以用來處理cookie

change事件(onchange)

在<input>、<select>、<textarea>元素中，當用戶提交對元素值的更改時觸發

onchange通常與輸入字段的驗證結合使用

mouseover事件(onmouseover)

當使用定點設備(鼠標、觸控板)將光標移動到元素或其子元素之一上時觸發

mouseout事件(onmouseout)

當使用定點設備(鼠標、觸控板)將光標移動到元素或其子元素之一外時觸發

click事件(onclick)

當光標在元素內部時，按下和釋放定點設備按鈕(滑鼠按鍵)以點擊元素時觸發

mousedown事件(onmousedown)

當光標在元素內部時，按下定點設備按鈕(滑鼠按鍵)時觸發

mouseup事件(onmouseup)

當光標在元素內部時，釋放定點設備按鈕(滑鼠按鍵)時觸發

使用HTML事件屬性將事件分配給HTML元素(不建議，容易被植入惡意程式碼)

<button onclick="displayDate()">Try it</button>

使用HTML DOM將事件分配給HTML元素(一個事件只能綁定一個函數)

<script>

document.getElementById(id).onclick = function(){code}

</script>

JavaScript HTML DOM EventListener事件監聽器

addEventListener()方法

設置了一個函數，只要將指定的事件傳遞給目標，該函數就會被調用

可以將多個事件處理程序添加到一個元素

可以將多個相同類型的事件處理程序添加到一個元素

可以將事件偵聽器添加到任何DOM對象

句法:

target.addEventListener(type, listener , options]);

target.addEventListener(type, listener , useCapture]);

參數值:

type(必須)區分大小寫的字符串，表示要偵聽的事件類型

listener(必須)當指定類型的事件發生時，接收通知的對象(實現EventListener接口的對象或JavaScript函數)

options(可選)指定有關事件偵聽器的特徵，可用的選項有:

capture一個布爾值，默認值為false，使用事件冒泡；當設置為true，使用事件捕獲

once一個布爾值，指示listener在添加後應至多調用一次，如果為true，則listene在調用時會自動將其刪除

passive一個布爾值，如果true，指示由指定的函數listener永遠不會調用 preventDefault()

signal AbortSignal接口，表示一個信號對象，可以與DOM請求進行通信，調用給定AbortSignal的abort()方法時，監聽器將被刪除

useCapture(可選)一個布爾值，默認值為false，使用事件冒泡；當設置為true，使用事件捕獲

返回值:

undefined

HTML DOM中有兩種事件傳播方式:

事件冒泡Bubbling:由內向外(子至父)觸發該事件

事件捕獲Capturing:由外向內(父至子)觸發該事件

stopPropagation()方法

阻止當前事件繼續進行捕獲及冒泡階段的傳遞

removeEventListener()方法

刪除先前由addEventListener()所註冊的事件監聽器

語法

target.removeEventListener(type, listener, options]);

target.removeEventListener(type, listener, useCapture]);

參數值:

type(必須)區分大小寫的字符串，表示要偵聽的事件類型

listener(必須)當指定類型的事件發生時，接收通知的對象(實現EventListener接口的對象或JavaScript函數)

options(可選)指定有關事件偵聽器的特徵，可用的選項有:

capture一個布爾值，默認值為false，使用事件冒泡；當設置為true，使用事件捕獲

once一個布爾值，指示listener在添加後應至多調用一次，如果為true，則listene在調用時會自動將其刪除

passive一個布爾值，如果true，指示由指定的函數listener永遠不會調用 preventDefault()

signal AbortSignal接口，表示一個信號對象，可以與DOM請求進行通信，調用給定AbortSignal的abort()方法時，監聽器將被刪除

useCapture(可選)一個布爾值，默認值為false，使用事件冒泡；當設置為true，使用事件捕獲

返回值:

undefined

JavaScript BOM(瀏覽器對象模型Browser Object Model)

BOM定義了與瀏覽器進行交互的方法和接口，沒有官方標準

window對象

代表瀏覽器的窗口，所有瀏覽器都支持

所有全局JavaScript對象、函數、變量都會自動成為window對象的成員

全局變量是window對象的屬性(包括HTML DOM的document對象)

全局函數是window對象的方法

屬性:

視窗尺寸:

window.innerHeight

獲得瀏覽器窗口內容區域的高度(包含水平滾動條)，以像素為單位

window.innerWidth

獲得瀏覽器窗口內容區域的寬度(包含垂直滾動條)，以像素為單位

方法:

window.open(strUrl, strWindowName, strWindowFeatures)

用指定的名稱將指定的資源加載到瀏覽器上下文；如果沒有指定名稱，則一個新的窗口會被打開，並且指定的資源會被加載到這個窗口的瀏覽器上下文

strUrl(必須)要在打開的新窗口中加載的URL

strWindowName(必須)新窗口的名稱

strWindowFeatures(可選)列出新窗口的特徵(大小，位置，滾動條等)作為一個DOMString

window.close()

關閉當前窗口或某個指定的窗口

window.moveTo(x, y)

移動當前窗口到指定的坐標位置

x要移動到的位置橫坐標

y要移動到的位置縱坐標

window.resizeTo(aWidth, aHeight)

動態調整窗口的大小

aWidth一個整數，表示新的outerWidth(包括滾動條、窗口邊框等)，以像素為單位

aHeight一個整數，表示新的outerHeight(包括滾動條、標題欄、窗口邊框等)，以像素為單位

window.screen對象

包含有關用戶屏幕的信息

可以在沒有window的前綴被寫入

屬性:

屏幕高、寬度:

screen.width

返回屏幕寬度，以像素為單位

screen.height

返回屏幕高度，以像素為單位

screen.availWidth

返回屏幕可用寬度(任務欄、歷史記錄、書籤側面板，可能會減少可用空間)，以像素為單位

screen.availHeight

返回屏幕可用高度，以像素為單位

屏幕顏色:

screen.colorDepth

返回屏幕顏色深度(用於顯示一種顏色的位數(24位True Colors或32位Deep Colors))

HTML中的#rrggbb表示True Colors

screen.pixelDepth

返回屏幕像素深度(顏色深度等同像素深度)

window.location對象

可用於獲取當前頁面地址(URL)，並將瀏覽器重定向到新頁面

可以在沒有window的前綴被寫入

屬性:

window.location.href

返回當前頁面的URL

window.location.hostname

返回當前頁面的主機名稱(域名)

window.location.pathname

返回當前頁面的路徑名稱

window.location.protocol

返回使用的Web協議(例如:http:或https:)

window.location.port

返回當前頁面的主機端口號

大多數瀏覽器不顯示默認端口號(例如:http是80、https是443)

方法:

window.location.assign(url)

根據指定的URL加載並顯示新文檔

window.history對象

瀏覽器的歷史記錄

可以在沒有window的前綴被寫入

方法:

history.back()

將歷史記錄中上一個URL加載(與在瀏覽器中單擊上一頁相同)

history.forward()

將歷史記錄中下一個URL加載(與在瀏覽器中單擊下一頁相同)

window.navigator對象

有關訪問者瀏覽器的信息

可以在沒有window的前綴被寫入

來自導航器對象的信息通常會產生誤導，並且不應用於檢測瀏覽器版本

屬性:

navigator.cookieEnabled

返回一個布爾值，表示當前頁面是否啟用了cookie

navigator.appName

返回瀏覽器的應用程序名稱(在任何瀏覽器中皆為Netscape，出於兼容性目的保留此屬性)

navigator.appCodeName

返回瀏覽器的應用程序代碼名稱(在任何瀏覽器中皆為Mozilla，出於兼容性目的保留此屬性)

navigator.platform

返回瀏覽器引擎的產品名稱(在任何瀏覽器中皆為Gecko，出於兼容性目的保留此屬性)

navigator.appVersion

返回有關瀏覽器的版本信息

navigator.userAgent

返回瀏覽器發送到服務器的使用者代理標頭

navigator.platform

返回表示瀏覽器平台(操作系統)的字符串

navigator.language

返回用戶偏好語言的字符串(通常是瀏覽器UI語言)

navigator.onLine

返回一個布爾值，表示瀏覽器的在線狀態，true為在線；false為離線

方法:

navigator.javaEnabled()

返回一個布爾值，表示是否啟用Java，true為啟用；false為未啟用

JavaScript彈出框Popup Box

要在彈出框中顯示換行符，使用\n

彈出框分為三種:

警告框Alert box、確認框Confirm box、提示框Prompt box

警報框

彈出警報框時，用戶將必須單擊確定以繼續，用於確保信息傳遞給用戶

window.alert(message)

顯示一個警告框，包含可選的指定內容、確定按鈕，可以在沒有window前綴下編寫

確認框

當彈出確認框時，用戶將必須單擊確定(返回true)或取消(返回 false)才能繼續，用於用戶驗證或接受某些內容

window.confirm(message)

顯示一個確認框，包含可選的指定內容、確定和取消按鈕，可以在沒有window前綴下編寫

提示框

當彈出提示框時，用戶必須在輸入值後單擊確定(返回輸入值)或取消(返回null)才能繼續，用於用戶在進入頁面之前輸入值

window.prompt(message, default)

顯示一個提示框，包含可選的指定內容，可以有默認輸入值，可以在沒有window前綴下編寫

JavaScript計時事件Timing Event

HTML DOM Window對象允許以指定的時間間隔執行代碼

window.setTimeout(function, delay, arg1, arg2, ...)

等待指定的毫秒數後執行函數，可以在沒有window前綴下編寫

參數值:

function(必須)執行函數

delay(可選)執行延遲毫秒數，默認為0(實際的延遲可能會比期待值長)

arg1, ..., argN(可選)附加參數(計時器到期，會作為參數傳遞給function)

返回值:

一個正整數，表示定時器的編號timeoutID(可以傳遞給clearTimeout()來取消該定時器)

window.setInterval(func, delay, arg1, arg2, ...)

等待指定的毫秒數後重複執行函數，每次調用之間有固定的時間延遲，可以在沒有window前綴下編寫

參數值:

function(必須)執行函數

delay(可選)重複執行間延遲毫秒數，默認為0(實際的延遲可能會比期待值長)

arg1, ..., argN(可選)附加參數(計時器到期，會作為參數傳遞給function)

返回值:

一個正整數，表示定時器的編號intervalID (可以傳遞給clearInterval()來取消該定時器)

window.clearTimeout(timeoutID)

取消調用setTimeout()建立的定時器，可以在沒有window前綴下編寫

window.clearInterval(intervalID)

取消調用setInterval()建立的定時器，可以在沒有window前綴下編寫

JavaScript Cookie

存儲在您計算機上的小型文本文件中的數據，可以將用戶信息存儲在網頁中，以名稱/值對(key=value)的形式保存

document.cookie屬性

創建，讀取和刪除Cookie

創建一個cookie:

document.cookie = "username=John Doe";

添加cookie到期日期(以UTC時間為單位)，cookie到期日期默認為關閉瀏覽器時:

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";

告訴瀏覽器cookie屬於哪個路徑，cookie路徑默認為當前頁面

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

用JavaScript讀取cookie，document.cookie將以一個字符串返回所有cookie:

var x = document.cookie;

刪除Cookie時只需將expires參數設置為經過的日期即可，應該定義cookie路徑，以確保刪除正確的cookie

AJAX 非同步的JavaScript與XML Asynchronous JavaScript and XML

非同步資料請求，網頁不用重新整理，就能即時地透過瀏覽器去跟伺服器交換數據

使用數個既有技術的新方法，這些技術包括HTML或XHTML、CSS、JavaScript、DOM、XML、XSLT以及最重要的XMLHttpRequest物件

一般由下列組成:

瀏覽器內置XMLHttpRequest對象(用於從Web服務器請求數據)

JavaScript和HTML DOM(顯示或使用數據)

伺服器對AJAX資料請求回應通常是HTML、XML、JSON三種資料格式，最常與Javascript做搭配就是JSON

出於安全原因，瀏覽器不允許跨域訪問，所以網頁及網頁嘗試加載的XML文件必須位於同一服務器上

瀏覽器可以使用Fetch API代替XMLHttpRequest對象，允許Web瀏覽器向Web服務器發出HTTP請求

XMLHttpRequest XHR對象

瀏覽器都具有內置XMLHttpRequest對象，可用於在後台與Web服務器交換數據(更新網頁的某些部分，而無需重新加載整個頁面)

創建XMLHttpRequest對象

var xhttp = new XMLHttpRequest();

將請求發送到服務器

get比post更簡單、快捷，一般用來讀取資料

post，一般用來傳送資料

在以下情況下，請始終使用POST:

不能選擇緩存文件(更新服務器上的文件或數據庫)

向服務器發送大量數據(POST沒有大小限制)

發送用戶輸入(POST可以包含未知字符，POST比GET更安全)

open()

指定請求的類型(初始化一個請求)

句法:

XMLHttpRequest.open(method, url, async, user, password)

參數值:

method HTTP的請求方法，例如GET、POST、PUT、DELETE等等，忽略非HTTP的URL

url一個DOMString，代表要向其發送請求的URL(服務器上文件的地址)

async(可選)布爾參數，指示是否異步執行操作，默認為true，不會等資料回傳就繼續下一個動作，使用事件偵聽器提供完成交易的通知；如果為false，則會等資料回傳才進行下一步動作，在send()收到響應之前該方法不會返回

user(可選)身份驗證的可選用戶名，默認為null

password(可選)身份驗證的可選密碼，默認為null

send()

將請求發送到服務器

句法:

XMLHttpRequest.send(body)

參數值:

body(可選)XHR請求中要發送的數據主體，可以是:

Document 在這種情況下，它在發送之前已序列化

XMLHttpRequestBodyInit 可以是Blob、BufferSource、FormData、URLSearchParams、USVString對象

null默認值

返回值:

undefined

setRequestHeader()

使用設定HTTP的表頭請求，必須在 open()之後、send()之前呼叫，之後每一次呼叫，都會把給定的文字附加在已存在的表頭內容之後

XMLHttpRequest.setRequestHeader(header, value)

參數值:

header想要設定所屬值的表頭名稱

value設定設定表頭本身的值

onreadystatechange屬性

定義當readyState更改時要執行的函數(每當readyState更改時，都會調用該函數)

readyState屬性

回傳一個XMLHttpRequest客戶端對象當前的狀態

0 XMLHttpRequest已被建立，但open()方法尚未被呼叫

1 open()方法已被呼叫

2 send()方法已被呼叫(可取得標題與狀態)

3 回應資料下載中(responseText會擁有部分資料)

4 完成下載操作

status屬性

包含Response的狀態碼，例如200表示成功

statusText屬性

包含了與該Response狀態碼一致的狀態信息，例如，OK對應200

responseText屬性

將服務器響應作為JavaScript字符串返回

responseXML屬性

將服務器響應作為XML DOM對象返回

getAllResponseHeaders()方法

從服務器響應中返回所有標頭信息

getResponseHeader()方法

從服務器響應中返回特定的標頭信息

JavaScript JSON(JavaScript Object Notation、JavaScript物件表示法)

用於存儲和傳輸數據的格式，輕量級的資料交換語言，以易於讓人閱讀的文字為基礎，用來傳輸由屬性值或者序列性的值組成的資料物件，JSON語法源自JavaScript對象表示法語法，但JSON是獨立於語言的文字格式

JSON語法規則

在JSON中，值必須是以下數據類型:

string(必須用雙引號)

number(整數或浮點數)

object (包含有效的JSON值)

array

boolean

null

JSON數據

寫為名稱/值對，由一個字段名(雙引號)、一個冒號、一個值組成，用逗號分隔

"firstName":"John"

JSON名稱必需雙引號，JavaScript名稱可以不用

JSON對象

寫在花括號內，就像JavaScript中一樣，對象可以包含多個名稱/值對{"firstName":"John", "lastName":"Doe"}

JSON數組

寫在方括號內，就像在JavaScript中一樣，數組可以包含對象

"employees":[

{"firstName":"John", "lastName":"Doe"},

{"firstName":"Anna", "lastName":"Smith"},

{"firstName":"Peter", "lastName":"Jones"}

]

JSON方法

parse()方法

解析一個字符串(以JSON格式編寫)，並返回一個JavaScript對象

句法

JSON.parse(string, function)

參數值:

string(必須)一個以JSON格式編寫的字符串

function(可選)用於轉換結果的函數，被每個item call一次，任何嵌套的object會在父代前轉換，如果函數返回有效值，則轉換值取代item值；如果函數返回undefined，則delete item

返回值:

JavaScript對像或JavaScript數組

JSON.stringify()方法

將JavaScript對象轉換為字符串(將數據發送到Web服務器時，數據必須是字符串)

句法:

JSON.stringify(obj, replacer, space)

參數值:

obj(必須) JavaScript對象

replacer(可選)用於轉換結果的函數或數組，被每個item call一次

space(可選) String或Number，如果是String，將String替代white space(最多10字符)；如果是Number，將Number數量的white space替代white space(最多10個)

返回值:

String

Web API應用程式介面Application Programming Interface

電腦作業系統或程式函式庫提供給應用程式呼叫使用的程式碼，其主要目的是讓應用程式開發人員得以呼叫一組常式功能，而無須考慮其底層的原始碼為何、或理解其內部工作機制的細節。API本身是抽象的，它僅定義了一個介面，而不涉及應用程式在實作過程中的具體操作

Web History API

提供了訪問windows.history對象的簡便方法，所有瀏覽器均支持

window.history.back();

上一步

window.history.forward();

下一步

window.history.go(x);

到當前相對x步位置

Web Storage API

用於在瀏覽器中存儲和檢索數據

localStorage對象

提供對特定網站的本地存儲的訪問，允許存儲，讀取，添加，修改和刪除該域的數據項

數據沒有到期日期，並且在關閉瀏覽器時不會刪除，將在幾天，幾周和幾年內可用

localStorage.setItem("name", "John Doe");

將數據項存儲在存儲器中，以名稱和值作為參數

localStorage.getItem("name");

從存儲中檢索數據項，以名稱作為參數

sessionStorage對象與localStorage對象不同之處在於關閉瀏覽器後，數據將被刪除

sessionStorage.setItem("name", "John Doe");

sessionStorage.getItem("name");

Web Workers API

一種JavaScript，它在後台運行，獨立於其他腳本，而不會影響頁面的性能

Fetch API

允許Web瀏覽器向Web服務器發出HTTP請求，不再需要XMLHttpRequest

fetch()方法是一個位於全域window物件的方法，語法結構完全是Promise的語法，被用來執行送出Request(要求)的工作，如果成功得到回應的話，它會回傳一個帶有Response(回應)物件的已實現Promise物件

fetch('http://abc.com/', {method: 'get'})

.then(function(response) {

//處理 response

}).catch(function(err) {

// Error :(

})

但要注意的是fetch在只要在伺服器有回應的情況下(只要不是網路連線問題，或是伺服器失連等等)，都會回傳已實現的Promise物件狀態，在這其中也會包含狀態碼為錯誤碼(404, 500...)的情況

function processStatus(response) {

// 狀態 "0" 是處理本地檔案 (例如Cordova/Phonegap等等)

if (response.status === 200 || response.status === 0) {

return Promise.resolve(response)

} else {

return Promise.reject(new Error(response.statusText))

}

}

fetch(request)

.then(processStatus)

.then()

.catch()

fetch由以下組成:

GlobalFetch提供全域的fetch方法

Request要求，其中包含method、url、headers、context、body等等屬性與clone方法

Response回應，其中包含headers、ok、status、statusText、type、body等等屬性與clone方法

Headers執行Request與Response中所包含的headers的各種動作，例如取回、增加、移除、檢查等等。設計這個介面的原因有一部份是為了安全性。

Body同時在Request與Response中均有實作(Mixin，將多個物件(或類別)中會共同使用(分享)的方法或屬性另外用一個物件或介面整合包裝起來，然後讓其他的物件(或類別)來使用其中的方法或屬性的設計樣式)，裡面有包含主體內容的資料，是一種ReadableStream(可讀取串流)的物件

Shadow DOM API

提供了一種將隱藏的單獨DOM附加到元素的方法

Shadow host影子主機:影子DOM附加到的常規DOM節點

Shadow tree影子樹:影子DOM內部的DOM樹

Shadow boundary影子邊界:影子DOM結束且常規DOM開始的位置

Shadow root影子根:影子樹的根節點

Element.attachShadow()方法

將影子DOM樹附加到指定的元素，並返回對其的引用ShadowRoot

句法:

var shadowroot = element.attachShadow(shadowRootInit);

參數值:

shadowRootInit

一個 ShadowRootInit字典，包含:

mode

一個字符串，指定影子DOM樹的封裝模式，包含:

open:可以從根外部的JavaScript訪問影子根的元素，例如，使用 Element.shadowRoot:

element.shadowRoot; // Returns a ShadowRoot obj

closed:拒絕從外部封閉JavaScript根目錄訪問其影子根節點:

element.shadowRoot; // Returns null

返回值:

一個ShadowRoot對象

Element.shadowRoot屬性

影子樹的根節點

路徑

「.」是代表目前所在目錄

「..」是代表上層目錄，若目前已經是根目錄則依然為目前所在目錄

「/」在各個目錄名稱之間的分隔符號，若放置在路徑之前則代表根目錄

1.絕對路徑：

這個檔案在本機端或是網路上的絕對位置

file:///c:/windows/search.html「本機C槽上的一個HTML 檔案」

http://www.csie.nuu.edu.tw「網路上一台 WWW Server 上的一個 HTML 檔案」

/jang/courses/n12345678「本機 WWW 根目錄下的一個目錄」

2.相對路徑：

相對於這個檔案的位置

text.html「表同一層目錄下的 text.html 檔案」

./text.html「表同一層目錄下的 text.html 檔案 」

image/text.html「表示 image 子目錄下的 text1.gif 檔案 」

../index.html「表示上一層目錄下的 index.html 檔案 」

../html40/cover.html「表示上一層目錄下 html40 子目錄的 index.html 檔案 」

JavaScript module

ES6模組

有各自的作用域，而且內部自動定義為use strict (嚴格模式)

一個檔案一個模組

在HTML文檔中創建一個type為module的腳本，以使用export和import

<script type="module" src="filename.js"></script>

將模組中的資料匯出給其他模組export:

1.定義後匯出:

// 定義一個變數

let name = 'doug';

// 匯出這個變數

export name;

// 定義一個函式

function say(){

console.log('hello ' + name);

}

// 匯出這個函式

export say;

// 定義一個類

class Student {

constructor(name, age){

this.name = name;

this.age = age;

}

}

// 匯出這個類

export Student;

2.定義同時匯出:

// 定義一個變數並匯出

export let name = 'doug';

// 定義一個函式並匯出

export function say(){

console.log('hello ' + name);

}

// 定義一個類並匯出

export class Student {

constructor(name, age){

this.name = name;

this.age = age;

}

}

從別的模組匯入資料import:

import { 資料1, 資料2, ... , 資料n } from 模組名

import { name, say } from './a.js';

匯入的資料是沒有辦法直接改變的(無法給name重新賦值)，但可以間接修改(例如在模組a中定義了一個可以修改name值的函數，通過呼叫這個函數來修改name值)

一次性匯入一個模組匯出出來的所有資料

import \* as aObj from './a.js'; // a 模組所有匯出出的資料都掛到了 aObj 上

// 通過 aObj 來訪問和呼叫 a 模組中的變數和方法

console.log(aObj.name);

aObj.say();

對匯入和匯出的資料重新命名as:

export {say as speak}; // 匯出時重新命名

import { speak as howling} from './a.js'; // 匯入時重新命名

將一個值作為預設值匯出出去 default:

每個模組只有一個預設值

1.定義時匯出

export default function(){

console.log('方法1匯出函式為預設值');

}

2.定義後匯出

let name = 'doug';

export default name;

3.將這個值用as命名為default後匯出

class Student{ // ... }

export {Student as default};

匯入其他模組匯出的預設值:

匯入一個模組匯出的預設值, 並不用{}來包裹, 而且不用使用as就可以將這個預設值重新命名

import familyName from './a.js'; //匯入name，將其重新命名為familyName

同時匯入預設值和非預設值:

import 預設值 { 非預設值 } from './a.js';

將從其他模組匯入的資料匯出:

1.先匯入再匯出

import { name } from './a.js'; // 匯入

export {name}; // 匯出

2.一行程式碼之內同時匯入和匯出

export { name } from './a.js';

也可以重新命名後再匯出出去

export { name as lastName } from './a.js';

解構賦值 Destructuring assignment

一種JavaScript運算式，可以把陣列或物件中的資料解開擷取成為獨立變數(在等式的左右如同鏡子一般，對映出陣列或物件的結構)

陣列解構:

let a, b, rest;

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(a); // 10

console.log(b); // 20

console.log(rest); // [30, 40, 50]

忽略某些值

const [a, , b] = [1, 2, 3] // a=1, b=3

交換值

const a = 1, b = 2;

[b, a] = [a, b] //a=2, b=1

變數可以設定預設值，當解構陣列對應的元素是undefined時

let a, b;

[a=5, b=7] = [1];

console.log(a); // 1

console.log(b); // 7

物件解構:

物件解構且該句沒有進行宣告時，必須在該句外加上括號，前句需要以分號結束，否則可能把上一句視為函式執行

{a, b} = {a: 1, b: 2} //無效，因為左邊的{a, b}被視為程式碼區塊而非物件

({a, b} = {a: 1, b: 2}) //有效的，視為const {a, b} = {a: 1, b: 2}

({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});

console.log(a); // 10

console.log(b); // 20

console.log(rest); // {c: 30, d: 40}

賦予新的變數名稱

const { prop: x, prop2: y } = { prop: 5, prop2: 10 } // x=5, y=10

變數可以設定預設值，當解構物件中對應的值是undefined時

const {a = 10, b = 5} = {a: 3};

console.log(a); // 3

console.log(b); // 5

樣板字面值Template literals、樣板字串template strings

允許嵌入運算式的字串字面值string literals，用於多行字串及字串內插string interpolation

透過反引號`包住的內容，會保留所有的換行和空行(字符串陣列)

console.log(`string text line 1

string text line 2`);

等同

console.log('string text line 1\n' +

'string text line 2');

// "string text line 1

// string text line 2"

${...}錢字元及花括號用於嵌入變量或任何的表達式(字符串)

var a = 5;

var b = 10;

console.log(`Fifteen is ${a + b} and

not ${2 \* a + b}.`);

等同

var a = 5;

var b = 10;

console.log('Fifteen is ' + (a + b) + ' and\nnot ' + (2 \* a + b) + '.');

// "Fifteen is 15 and

// not 20."

要在樣板字面值中跳脫一個反引號`，可以於反引號前加上一個反斜線\

var greeting = `\`Hello\` World!`;

console.log(greeting); // "`Hello` World!"

標籤模版tagged template

透過標籤函數操作樣板字面值的輸出直接在函式後面加上模版字符串，模版字符串的進階用法，適合用在前端用來過濾使用者所輸入的訊息

console.log `Hello EveryOne`; // Array ["Hello EveryOne"]

console.log('Hello EveryOne'); // String "Hello EveryOne"

把使用者輸入的內容${...}，和我們網頁原本的內容分開做處理，以達到過濾 HTML 字符串，防止用戶輸入惡意內容的效果

let myName = "PJCHENder";

let myCountry = "Taiwan";

tag `<p> My name is ${myName} and my coutry is ${myCountry}</p>`;

// tag(["<p> My name is ", " and my coutry is ", "</p>"], "PJCHENder", "Taiwan")

// 等同於上面這一段

function tag(template, ...values){

console.log(template); // ["<p> My name is ", " and my coutry is ", "</p>"]

console.log(values); // ["PJCHENder", "Taiwan"]

}

JavaScript調試Debug

JavaScript調試器Debugger

console.log()方法

將消息寫入控制台，該控制台可用於測試，請確保使控制台視圖可見(按F12鍵查看控制台Console)

句法:

console.log(message)

參數值:

message(必須)寫入控制台console的消息(String)或object(包含array)

設置斷點Breakpoint

在調試器窗口中，可以在JavaScript代碼中設置斷點，在每個斷點處，JavaScript將停止執行，並讓您檢查JavaScript值，之後可以繼續執行代碼(通常使用播放按鈕)

debugger關鍵字

停止執行JavaScript，並調用調試功能，與在代碼中設置斷點的功能相同，通常可以使用F12鍵在瀏覽器中激活調試，然後在調試器菜單中選擇控制台console，如果沒有調試可用，則debugger無效

句法

debugger;

JavaScript樣式指南

小駝峰式命名法lowerCamelCase

單字之間不以空格斷開或以連接號、底線連結，第一個單字以小寫字母開始；第二個單字的首字母開始大寫

變量名

使用lowerCamelCase，以字母開頭

運算符

在運算符( + - \* / = )前後以及逗號(,)後加空格

代碼縮進

使用2個空格來縮進代碼塊，不要使用製表符(tab)進行縮進(不同的編輯者解釋不同)

聲明規則

簡單語句:

以分號( ; )結束簡單語句

複雜(複合)語句:

將左花括號( { )放在第一行的末尾，在左花括號( { )之前使用一個空格

將右花括號( } )放}在新行上，不要有空格

不要用分號( ; )結束複雜的語句

對象規則

將左花括號( { )放在與對象名稱相同的行上

在每個屬性及其值之間使用冒號( : )加一個空格

如果值為String，請在兩邊使用引號( “ ‘ )

不要在最後一個屬性-值對之後添加逗號( , )

將右花括號( } )放}在新行上，不要有空格

始終用分號( ; )結束對象定義

線長

避免使用超過80個字符的行，打破的最佳位置是在運算符或逗號之後

使用小寫文件名

大多數Web服務器(Apache、Unix)對文件名區分大小寫

JavaScript最佳做法Best Practices

避免全局變量

減少使用全局變量，包括所有數據類型，對象和函數，全局變量和函數可以被其他腳本覆蓋

始終聲明局部變量

函數中使用的所有變量都必須使用var或let關鍵字聲明為局部變量，嚴格模式不允許未聲​​明的變量

在頂部聲明

將所有聲明放在每個腳本或函數的頂部

聲明同時初始化變量

聲明變量同時初始化變量

不聲明數字、字符串或布爾為對象

始終將數字，字符串或布爾值視為原始值，不作為對象，將這些類型聲明為對象，會降低執行速度並產生討厭的副作用

不要使用new Object()

使用{}代替new Object()

使用""代替new String()

使用0代替new Number()

使用false代替new Boolean()

使用[]代替new Array()

使用/()/代替new RegExp()

使用function (){}代替new Function()

當心自動類型轉換

數字可能會意外地轉換為字符串或NaN

使用===比較

使用參數默認值

調用函數時，缺少參數的值將設置為undefined，可能會破壞代碼，將默認值分配給參數以避免

使用default結束switch

避免使用eval()

eval()函數用於將文本作為代碼運行。在幾乎所有情況下，都沒有必要使用它，因為它允許運行任意代碼，所以它也表示一個安全問題

JavaScript性能

減少循環活動

循環中的每個語句(包括for語句)都針對循環的每次迭代執行，可以放在循環外部的語句或賦值將使循環運行更快

減少DOM訪問

如果要多次訪問DOM元素，請一次次訪問，並將其用作局部變量

減少DOM大小

使HTML DOM中的元素數量保持較小，將改善頁面加載，並加快渲染(頁面顯示)

避免不必要的變量

如果不打算保存值，請不要創建新變量

延遲JavaScript載入

將腳本放在頁面正文的底部可以使瀏覽器首先加載頁面，在下載腳本時，瀏覽器將不會開始其他任何下載，此外，所有解析和渲染活動都可能被阻止(HTTP規範定義瀏覽器不應並行下載兩個以上的組件)

另一種方法是defer="true"在<script>中使用defer屬性指定腳本應在頁面解析完成後執行，但僅適用於外部腳本

避免使用with關鍵字

with對速度有負面影響，還會弄亂JavaScript範圍，嚴格模式下不允許使用with

JavaScript保留字

在JavaScript 中，您不能把這些保留字作為變量、標記或函數名來使用

abstract arguments await\* Boolean break byte case catch

char class\* const continue debugger default delete do

double else enum\* eval export\* extends\* false final

finally float for function goto if implements import\*

in instanceof int interface let\* long native new

null package private protected public return short static

super\* switch synchronized this throw throws transient true

try typeof var void volatile while with yield

以下保留字已從ECMAScript 5/6中删除(依舊不能使用)

abstract boolean byte char

double final float goto

int long native short

synchronized throws transient volatile

避免使用 JavaScript内建對象的名稱、屬性和方法

Array Date eval function hasOwnProperty Infinity isFinite isNaN

isPrototypeOf length Math NaN name Number Object prototype

String toString undefined valueOf

JavaScript常與 Java一起使用，避免使用某些 Java對象和屬性

getClass java JavaArray javaClass JavaObject JavaPackage

避免使用HTML和Window對象和屬性的名稱

alert all anchor anchors area assign blur button checkbox clearInterval

clearTimeout clientInformation close closed confirm constructor crypto

decodeURI decodeURIComponent defaultStatus document element

elements embed embeds encodeURI encodeURIComponent escape event

fileUpload focus form forms frame innerHeight innerWidth layer

layers link location mimeTypes navigate navigator frames frameRate

hidden history image images offscreenBuffering open opener option

outerHeight outerWidth packages pageXOffset pageYOffset parent

parseFloat parseInt password pkcs11 plugin prompt propertyIsEnum

radio reset screen screen scroll secure select self setInterval

setTimeout status submit taint text textarea top unescape untaint

window

避免使用所有HTML事件處理程序的名稱

onblur onclick onerror onfocus onkeydown onkeypress onkeyup

onmouseover onload onmouseup onmousedown onsubmit

JavaScript版本

ECMAScript是JavaScript的正式名稱

ES1 ECMAScript 1(1997年)第一版

ES2 ECMAScript 2(1998年)編輯變更

ES3 ECMAScript 3(1999年)

添加正則表達式

添加try / catch

ES4 ECMAScript 4從未發布

ES5 ECMAScript 5(2009年)

添加嚴格模式

添加JSON支持

添加String.trim()

添加Array.isArray()

添加數組疊代方法

ES6 ECMAScript 2015

添加let和const

添加默認參數值default parameter values

添加Array.find()

添加Array.findIndex()

ECMAScript 2016

添加了指數運算符(\*\*)

添加了Array.prototype.includes

ECMAScript 2017

添加字符串填充string padding

添加Object.entries

添加Object.values

添加異步功能async functions

添加共享內存shared memory

ECMAScript 2018

添加rest / spread屬性

添加了異步迭代asynchronous iteration

添加了Promise.finally()

對RegExp的添加

ES5 ECMAScript 5(2009年)

ES5特徵

"use strict"嚴格模式

String.trim()從字符串的兩邊刪除空格

Array.isArray()檢查對像是否為數組

Array.forEach()為每個數組元素調用一次函數

Array.map()為每個數組元素調用一次函數，將函數的結果儲存在一個新數組

Array.filter()創建一個具有通過測試函數的數組元素的新數組

Array.reduce()將數組減少為單個值，為數組的每個值(從左到右)執行提供的函數

Array.reduceRight()將數組減少為單個值，為數組的每個值(從右到左)執行提供的函數

Array.every()檢查數組中的所有元素是否都通過測試函數

Array.some()檢查數組中的任意元素是否通過測試函數

Array.indexOf()在數組中(從頭到尾)搜索指定的項目，並返回其第一次出現的位置

Array.lastIndexOf()在數組中(從尾到頭)搜索指定的項目，並返回其最後一次出現的位置

JSON.parse()解析一個字符串(以JSON格式編寫)，並返回一個JavaScript對象

JSON.stringify()將JavaScript對象轉換為字符串

Date.now()根據UTC返回自1970年1月1日以來的毫秒數

屬性獲取器Getter和設置器Setter

新對象屬性方法:

Object.defineProperty()在一個object上定義新屬性或修改現有屬性，並返回這個object

Object.defineProperties()在一個object上定義一個或多個新屬性或修改現有屬性，並返回這個object

Object.getOwnPropertyDescriptor()返回一個對象，該對象包含指定對象的一個命名屬性(直接存在於對像上，而不是對象原型鏈中的屬性)的所有屬性描述符

Object.getOwnPropertyNames()返回一個數組，其中包含指定對象自身所有屬性(包含可枚舉和不可枚舉，除了使用Symbol的屬性)名稱

Object.keys()返回object中所有可枚舉的屬性名稱(key)的數組

Object.getPrototypeOf()返回指定對象的原型(即內部的Prototype屬性值)

Object.preventExtensions()防止該對象進行擴展(將新屬性添加到對象)

Object.isExtensible()確認對像是否可擴展(將新屬性添加到對象)

Object.seal()密封對象(防止向對象添加新屬性)，並將所有現有屬性標記為不可配置non-configurable，只要是writable，當前屬性值仍可更改

Object.isSealed()確認對像是否被密封

Object.freeze()凍結(防止向對象添加新屬性、刪除現有屬性、更改現有屬性的可枚舉性enumerability或可配置性configurability或可寫性writability、更改現有屬性的值、更改對象原型)對象，凍結的對象無法再更改

Object.isFrozen()確認對像是否被凍結

ES5語法更改

1.允許對字符串進行屬性訪問[]

charAt()返回字符串中指定位置的字符

2.ES5允許在對象和數組定義中使用逗號結尾(不建議，JSON不允許尾隨逗號)

3.如果使用反斜杠\進行轉義，ES5允許在多行上使用字符串文字(可能沒有通用支持，更安全的方法是使用字符串加法 + )

4.ES5允許保留字作為屬性名稱(不建議)

ES6 ECMAScript 2015

ES6特徵

let關鍵字 聲明塊級作用域的變量

const關鍵字 聲明塊級作用域的常數(具有恆定值JavaScript變量)

JavaScript箭頭函數 使用簡短的語法編寫函數表達式，不需要function關鍵字、return關鍵字和{}

JavaScript For/of遍歷可疊代對象的值

JavaScript Class JavaScript對象的範本

JavaScript Promise承諾 鏈接生產代碼Producing Code和消費代碼Consuming Code的JavaScript對象(生產代碼可能需要一些時間，消費代碼必須等待結果)

?JavaScript Symbol符號一種原始數據類型，代表唯一的隱藏標識符，沒有其他代碼可以意外訪問該標識符(符號始終是唯一的，如果使用相同的描述創建兩個符號，則它們將具有不同的值)

ES6允許函數參數具有默認值

rest參數(...)允許函數將不確定數量的參數視為數組

Array.find()返回通過測試函數的數組中第一個元素的值

Array.findIndex()返回通過測試函數的數組中第一個元素的位置索引

新數字屬性

ES6向Number對象添加了以下屬性:

EPSILON1與大於1可表示的最小浮點數的差值

MIN\_SAFE\_INTEGER可在JavaScript中安全表示的最小數字

MAX\_SAFE\_INTEGER可在JavaScript中安全表示的最大數字

新數字方法

Number.isInteger()確定一個類型為Number的值是否為整數

Number.isSafeInteger()確定一個類型為Number的值是否為安全整數(2\*\*53-1至-2\*\*53-1間所有整數)

新全局方法:

isFinite()確定一個值是否為有限數

isNaN()確定一個值是否為NaN

JavaScript Module模塊

ECMAScript 2016

ECMAScript 2016特徵

JavaScript指數(\*\*)

JavaScript指數分配(\*\*=)

?JavaScript Array.prototype.includes檢查數組中是否存在元素

ECMAScript 2017

ECMAScript 2017特徵

JavaScript字符串填充:

padStart在字符串的開頭填充

padEnd在字符串的結尾填充

JavaScript Object.entries返回帶有鍵/值對的疊代器對象

JavaScript Object.values返回帶有值的疊代器對象

JavaScript異步Async函數

?JavaScript共享內存

ECMAScript 2018

ECMAScript 2018特徵

?異步迭代:

在for/of循環中使用await關鍵字

?最後承諾Promise.finally

?對像rest屬性 可以銷毀一個對象，並將剩餘的東西收集到一個新的對像上

RegExp新特徵:

?Unicode屬性轉義符Property Escape(\ p {...})

向後斷言Assertion (?<=)和(?<!)

命名捕獲組Named Capture Group

s(dotAll)標誌Flag

JavaScript表格

JavaScript驗證API

約束驗證DOM方法:

checkValidity()方法 如果是有效值數據，返回true

setCustomValidity()方法 設置輸入元素的validationMessage驗證信息屬性

約束驗證DOM屬性:

validity屬性 與輸入元素的有效性相關的布爾

validationMessage屬性 當有效性為false時，瀏覽器顯示的消息

willValidate屬性 指示輸入元素是否將被驗證

有效性屬性:

customError屬性 如果設置了自定義有效性消息，則設置為true

patternMismatch屬性 如果元素的值與其pattern屬性不匹配，則設置為true

rangeOverflow屬性 如果元素的值大於其max屬性，則設置為true

rangeUnderflow屬性如果元素的值小於其min屬性，則設置為true

stepMismatch屬性 如果元素的值對於其每個step屬性無效，則設置為true

tooLong屬性 如果元素的值超過其maxLength屬性，則設置為true

typeMismatch屬性 如果元素的每個type屬性的值無效，則設置為true

valueMissing屬性 如果元素(具有必需屬性)沒有值，則設置為true

valid屬性 如果元素的值有效，則設置為true