JavaScript導論

JavaScript包括：JavaScript基礎語法，WebAPI（DOM操作），JavaScript面向對象

jQuery基於JavaScript封裝。

JavaScript基礎語法

## 1. 編程語言

### （1）編程

編程：讓計算機為解決某個問題而使用某種程序設計語言編寫程序代碼，並最終得到結果的過程。

計算機程序：計算機所執行的一系列的指令集合。

這裡所定義的計算機是任何能夠執行代碼的設備，可能是智能手機、ATM機、服務器等。

### （2）計算機語言

計算機語言指用於人與計算機之間通訊的語言，它是人與計算機之間傳遞信息的媒介。

計算機語言種類非常多，總的來說可以分成機器語言，彙編語言和高級語言三大類。

計算機最終所執行的都是機器語言，它是由0和1組成的二進制數，二進制是計算機語言的基礎。

### （3）編程語言

可以通過類似於人類語言的“語言”來控制計算機，讓計算機為我們做事情，這樣的語言就叫做編程語言（Programming Language）。

編程語言是用來控制計算機的一系列指令，它由固定的格式和詞彙（不同編程語言的格式和詞彙不一樣），必須遵守。

常用的編程語言有兩種形式：彙編語言和高級語言。

彙編語言和機器語言實質是相同的，都是直接對硬件操作，只不過指令採用了英文縮寫的標識符，容易識別和記憶。

高級語言主要是相對於低級語言而言，它並不是特指某一種具體的語言，而是包括了許多編程語言，常用的有C語言、C++、Java、C#、Python、PHP、JavaScript、Go語言、Objective-C、Swift等。

### （4）翻譯器

高級語言所編制的程序不能直接被計算機識別，必須經過轉換才能被執行，為此，我們需要一個翻譯器。翻譯器可以將我們所編寫的源代碼轉換為機器語言，這也被稱為二進制化，記住1和0。



### （5）編程語言和標記語言區別

編程語言有很強的邏輯和行為能力。在編程語言裡，有很多if else、for、while等具有邏輯性和行為能力的指令，這是主動的。

標記語言（HTML）不用於向計算機發出指令，常用於格式化和鏈接。標記語言被計算機讀取，是被動的。

## 2. 計算機基礎

### （1）計算機組成



### （2）數據存儲

計算機內部使用二進制0和1來表示數據。

所有數據，包括文件、圖片等最終都是以二進制數據（0和1）的形式存放在硬盤中的。

所有程序，包括操作系統，本質都是各種數據，也以二進制數據的形式存放在硬盤中。平時所說的安裝軟件，就是把程序文件複製到硬盤中。

硬盤、內存都是保存的二進制數據。

### （3）數據存儲單位

bit < byte < kb < GB < TB…

位（bit）：1bit可以保存一個0或者1（最小的存儲單位）

字節（Byte）：1B=8b

千字節（KB）：1KB=1024B

兆字節（MB）：1MB=1024KB

吉字節（GB）：1GB=1024MB

太字節（TB）: 1TB=1024GB

……

### （4）程序運行



step① 打開某個程序時，先從硬盤中把程序的代碼加載到內存中

step② CPU執行內存中的代碼

需要內存的原因，CPU運行速度非常快，如果只從硬盤中讀取數據，會浪費CPU性能，所以使用存取速度更快的內存來保存運行時的數據。（內存是電，硬盤是機械。）

## 3. 初識JavaScript

### （1）JavaScript歷史

布蘭登·艾奇 Brendan Eich （1961年出生）在1995年用10天設計了JavaScript。

網景公司最初將其命名為LiveScript，後來與Sun（famous for Java）合作之後將其改名為JavaScript

JavaScript和Java沒有關係。

### （2）JavaScript是什麼

JavaScript是世界上最流行的語言之一，是一種運行在客戶端的腳本語言（script是腳本的意思）。

腳本語言：不需要編譯，運行過程中由js解釋器（js引擎）逐行來進行解釋並執行。

現在也可以基於Node.js技術進行服務器端編程。



客戶端（自己的電腦） 遠程服務器

### （3）JavaScript作用

* 表單動態校驗（密碼強度檢測）（JS產生最初的目的）
* 網頁特效
* 服務端開發（Node.js）
* 桌面程序（Electron）
* App（Cordova）
* 控制硬件-物聯網（Ruff）
* 遊戲開發（cocos2d-js）

### （4）HTML / CSS / JS 的關係

HTML決定網頁結構和內容（決定看到什麼），相當於人的身體。

CSS決定網頁呈現給用戶的模樣（決定好不好看），相當於給人穿衣服、化妝。

JS腳本語言（編程類語言）實現業務邏輯和頁面控制（決定功能），相當於人的各種動作。

### （5）瀏覽器執行JS

瀏覽器分成兩部分，渲染引擎和JS引擎。

渲染引擎：用來解析HTML和CSS，俗稱內核，比如Chrome瀏覽器的blink，老版本的webkit。

JS引擎：也稱為JS解釋器。用來讀取網頁中的JavaScript代碼，對其處理後運行，比如Chrome瀏覽器的V8（號稱“最快的JS引擎”）。

瀏覽器本身並不會執行JS代碼，而是通過內置JavaScript引擎（解釋器）來執行JS代碼。JS引擎執行代碼時逐行解釋每一句源碼（轉換為機器語言），然後由計算機去執行，所以JavaScript語言歸為腳本語言，會逐行解釋執行。

### （6）JS的組成



ECMAScript是由ECMA國際（原歐洲計算機製造商協會）進行標準化的一門編程語言，這種語言在萬維網上應用廣泛，它往往被稱為JavaScript或Jscript，但實際上這兩者都是ECMAScript語言的實現和拓展。



ECMAScript規定了JS的編程語法和基礎核心知識，是所有瀏覽器廠商共同遵守的一套JS語法工業標準。

更多参看MDN: [MDN手册]

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/JavaScript_technologies_overview>

DOM（Document Object Model）（文檔對象模型），是W3C組織推薦的處理可拓展標記語言的標準編程接口。通過DOM提供的接口可以對頁面上的各種元素進行操作（大小、位置、顏色等）。

BOM（Browser Object Model）（瀏覽器對象模型），提供了獨立於內容的，可以與瀏覽器窗口進行互動的對象結構。通過BOM可以操作瀏覽器窗口，比如彈出框、控制瀏覽器跳轉、獲取分辨率等。

### （7）書寫JS

JS有3種書寫位置，分別為行內、內嵌和外部。

**① 行內JS**

可以將單行或少量JS代碼寫在HTML標籤事件屬性中（已on開頭的屬性），如：onclick。

注意單雙引號：在HTML中推薦使用雙引號，在JS中推薦使用單引號。

可讀性差，在HTML中編寫大量JS代碼時，不方便閱讀。

引號易錯，引號多層嵌套匹配時，非常容易弄混。

特殊情況下使用。

**② 內嵌JS**

可以將多行JS代碼寫到<script>標籤中。

內嵌JS式學習時常用的方式。

**③ 外部JS文件**

利於HTML頁面代碼結構化，把大段JS代碼獨立到HTML頁面之外，既美觀，也方便文件級別的複用。

引用外部JS的script標籤中間不可以寫代碼。

<script src="my.js"></script> 兩個script中間不要寫代碼。

適合於JS代碼量比較大的情況。

以下為例子。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <!-- 2. 內嵌式js -->

    <script>

      alert("天邊彎月");

    </script>

    <!-- 3. 引入外部js 注意這裡是雙標籤-->

    <script src="my.js"></script>

  </head>

  <body>

    <!-- 1.行内式的js 直接写到元素的内部-->

    <input type="button" value="物華天寶" onclick="alert('人傑地靈')" />

  </body>

</html>

## 4. JavaScript注釋

    <script>

      // 單行注釋   快捷鍵 ctrl + /

      /\* 多行注釋   快捷鍵 shift + alt + a 修改為shift + ctrl + /

      多行注釋 \*/

    </script>

## 5. JavaScript輸入輸出語句

為了方便輸入輸出信息，JS提供了一些輸入輸出語句。常用語句：

    <script>

      //  給用戶的輸入框

      prompt("請輸入您的用戶名");

      // 展示給用戶看的

      alert("計算的結果是");

      // 控制台輸出 給程序員測試用的

      console.log("給程序員看的");

    </script>

## 6. 變量概述

### （1）什麼是變量

變量是用於存放數據的容器。通過變量名獲取數據、修改數據。

通俗解釋：變量是一個裝東西的盒子。

### （2）變量在內存中的存儲

本質：變量是程序在內存中申請的一塊用來存放數據的空間。

類比：變量名=酒店房間號碼，變量=酒店房間；聲明了1個變量=在酒店開了一間房。

## 7. 使用變量

使用變量有2步：①聲明變量，②賦值。

第①步：聲明變量

var age; //聲明一個名稱為age的變量。

var是一個JS關鍵字，用來聲明變量（variable）。使用該關鍵字聲明變量後，計算機會自動為變量分配內存空間。

age是程序員定義的變量名，程序員需要通過變量名來訪問內存中分配的空間。

第②步：賦值

age = 10；//給age這個變量賦值為10。

=用來把右邊的值賦給左邊的變量空間中，此處代表賦值。

變量值是程序員保存到變量空間裡的值。

變量的初始化

var age = 18; //聲明變量同時賦值為18。

聲明一個變量並賦值，叫變量的初始化。

## 8. 變量語法拓展

### （1）更新變量

一個變量被重新賦值後，它原有的值會被覆蓋，變量值將以最後一次賦的值為準。

### （2）同時聲明多個變量

同時聲明多個變量時，只需要寫一個var，多個變量名之間使用英文逗號隔開，最後一個變量後面加分號。

      var age = 18,

          address = "墨集",

          weather = "sunny";

### （3）聲明變量的特殊情況

|  |  |  |  |
| --- | --- | --- | --- |
| 情況 | 說明 | 結果 | 注釋 |
| var age;  console.log(age); | 只聲明，不賦值 | undefined |  |
| console.log(age) | 不聲明，不賦值，直接使用 | 報錯 |  |
| age = 10;  console.log(age); | 不聲明，只賦值 | 10 | 雖然可以使用，但不提倡這樣，因為它會變成全局變量 |

## 9. 變量命名規範

* 由字母A-Z a-z、數字0-9、下劃線\_、美元符號$組成，比如：usrAge，num01，\_name。
* 嚴格區分大小寫。var app; 和var App; 是兩個變量。
* 不能以數字開頭。18age是錯誤的。
* 不能使關鍵字、保留字。例如：var, for, while。也不要用name，雖然它既不是關鍵字也不是保留字。
* 變量名必須有意義。不要用中文拼音簡寫，用英文單詞。
* 遵守駝峰命名法。首字母小寫，後面單詞的首字母需要大寫。myFirstName

## 10. 交換變量案例

    <script>

      // 交換apple1和apple2的值

      var apple1 = "青蘋果";

      var apple2 = "紅蘋果";

      // 聲明一個臨時變量

      var temp;

      // 把右邊的值賦給左邊

      temp = apple1;

      apple1 = apple2;

      apple2 = temp;

    </script>

## 11 數據類型簡介

### （1）為什麼需要數據類型

在計算機中，不同的數據所需佔用的存儲空間是不同的，為了充分利用存儲空間，於是定義了不同的數據類型。

### （2）變量的數據類型

JavaScript是一種弱類型語言/動態語言，這意味著不用提前聲明變量的類型，在程序運行過程中，類型會被自動確定。

    <script>

      var x = 10; //x是數字型

      x = "spring"; //x是字符串型

    </script>

### （3）數據類型的分類

JS把數據類型分為兩類：

* 簡單數據類型（Number，String，Boolean，Undefined，Null）
* 複雜數據類型（object）

|  |  |  |
| --- | --- | --- |
| **簡單數據類型** | **說明** | **默認值** |
| Number | 數字型，包含整型值和浮點型值，如21，0.21 | 0 |
| Boolean | 布爾值類型，如true、false，等價於1和0 | false |
| String | 字符串類型，如”惠風和暢”。js裡，字符串都帶引號。 | “” |
| Undefined | var a; 聲明了變量a但是沒有給值，此時a = undefined | undefined |
| Null | var a=null; 聲明了變量a，為空值。 | null |

## 12 簡單數據類型

### （1）數字型

#### ①數字型進制

    <script>

      // 數字前加0代表八進制，用到的字符是0-7

      var num1 = 010;

      console.log(num1); //八進制數字010轉換為十進制是8

      var num2 = 012;

      console.log(num2); //八進制數字012轉換為十進制是10

      //數字前面加0x 表示十六進制

      var num3 = 0x9;

      console.log(num3); //十六進制數字0x9轉換為十進制是9

      var num4 = 0xa;

      console.log(num4); //十六進制數字0xa轉換為十進制是10

    </script>

#### ② 數字型範圍

JavaScript中數值的最大和最小值

    <script>

      console.log(Number.MAX\_VALUE); //1.7976931348623157e+308

      console.log(Number.MIN\_VALUE); //5e-324

    </script>

#### ③ 數字型3個特殊值

    <script>

      console.log(Infinity); //輸出：Infinity，代表無窮大，大於任何數值

      console.log(-Infinity); //輸出：-Infinity，代表無窮小，小於任何數值

      console.log(NaN); //輸出：NaN ，代表一個非數值

    </script>

#### ④ isNaN()

isNaN() 用來判斷是非數字，並且返回一個值。如果是數字，返回的是false；如果不是數字，返回的是true。

    <script>

      console.log(isNaN(12)); //false

      console.log(isNaN("summer")); //true

    </script>

### （2）字符串型 String

字符串型可以是引號中的任意文本，其語法為雙引號””和單引號’’。因為HTML標籤裡面的屬性使用的是雙引號，JS更推薦使用單引號。

如果引號裡面是數字、true、false，這也算String，比如’12’、’true’、’false’都屬於string。

#### ① 字符串引號嵌套

JS可以用單引號嵌套雙引號，或者用雙引號嵌套單引號（外雙內單，外單內雙）。

#### ② 字符串轉義符

類似HTML裡面的特殊字符，字符串中也有特殊字符，叫做轉義符。

轉義符都是\開頭，常用轉義符：

|  |  |
| --- | --- |
| 轉義符 | 解釋說明 |
| \n | 換行符，n是newline的意思 |
| \\ | 斜杠\ |
| \’ | 單引號 |
| \” | 雙引號 |
| \t | tab縮進 |
| \b | 空格，b是blank的意思 |

轉義字符都要寫在引號裡面。

#### ③ 字符串長度

字符串由若干字符組成，這些字符的數量就是字符串的長度。通過字符串的length屬性可以獲取整個字符串的長度。

    <script>

      var str = "Her name is Mia";

      console.log(str.length); //包括空格共15個字符。

    </script>

#### ④ 字符串拼接

多個字符串之間可以使用+進行拼接，其拼接方式為：字符串+任何類型數據=拼接之後的新字符串。

拼接前會把與字符串相加的任何類型轉換成字符串，再拼接成一個新的字符串。

數值相加，字符相連

    <script>

      console.log("有約不來" + " " + "閒敲棋子"); //打印出來是：有約不來 閒敲棋子

      console.log("有約不來" + 11); //打印出來是：有約不來11

      console.log("有約不來" + true); //打印出來是：有約不來true

      console.log("12" + 12); //打印出來是：1212

    </script>

變量不要寫到字符串裡面，要和字符串相連。

    <script>

      var num = 2356789;

      console.log("用戶一共有" + num + "位"); //打印出來是：用戶一共有2356789位

    </script>

字符串拼接案例：

先請用戶輸入年齡，然後彈框顯示用戶年齡

    <script>

      var age = prompt("請輸入您的年齡");

      alert("您今年" + age + "歲了");

    </script>

prompt取來的值是字符型的

### （3）布爾型 Boolean

布爾類型有兩個值：true和false，其中true表示真（對）（參與加法運算當作1），false表示假（錯）（參與加法運算當作0）。

console.log(true + 1); //2

console.log(false + 1); //1

### （4）Undefined和Null

一個聲明後沒有被賦值的變量會有一個默認值undefined。

注意undefined、Null和其它類型用+相連的結果：

    <script>

      var variable = undefined;

      console.log(variable + "天朗氣清"); //結果是字符串：undefined天朗氣清

      console.log(variable + 3); //結果是：NaN

      console.log(variable + true); //結果是：NaN

      var space = null;

      console.log(space + "天朗氣清"); //結果是字符串：null天朗氣清

      console.log(space + 3); //結果是：3

      console.log(space + true); //結果是：1

    </script>

## 13 獲取變量數據類型

### （1）獲取檢測變量的數據類型

typeof可用來獲取變量的數據類型

    <script>

      var num = 10;

      console.log(typeof num); //number

      var str = "天朗气清";

      console.log(typeof str); //string

      var timer = null;

      console.log(typeof timer); // object

    </script>

### （2）字面量

字面量表示如何表達一個值，讓人一眼看上去就明白其類型。

數字字面量：8,9,10

字符串字面量：“黑馬程序員”，“大前端”

布爾字面量：true，false

控制台裡不同顏色表示不同數據類型。



    <script>

      console.log(18); //蓝

      console.log("18"); //黑

      console.log(true); //深蓝

      console.log(undefined); //灰

      console.log(null); //灰

    </script>

## 14 數據類型轉換

### （1）為什麼需要數據類型轉換

使用表單、prompt獲取的數據默認是字符串類型的，不能直接用於數學運算。

通常實現3種方式的轉換：

* 轉換為字符串類型
* 轉換為數字型
* 轉換為布爾型

### （2）把其它類型轉換為字符串

|  |  |  |
| --- | --- | --- |
| 方式 | 說明 | 案例 |
| 變量.toString() | 轉成字符串 | var num=1;  num.toString(); |
| String(變量)強制轉換 | 轉成字符串 | var num=1;  String(num); |
| 加號拼接字符串（最重要）  （隱式轉換法） | 和字符串拼接的結果都是字符串 | var num=1;  alert(num + ‘字符串’);  加空字符串也行 |

### （3）把其它類型轉換為數字

|  |  |  |
| --- | --- | --- |
| **方式** | **說明** | **案例** |
| parseInt(string)函數（重要） | 將string類型轉成整數數值型 | parseInt(‘78’)  注意，parseInt只取整數部分：  console.log(parseInt(‘3.14’)); //3  console.log(parseInt(‘3.94’)); //3  console.log(parseInt(‘12px’)); //12  console.log(parseInt(‘rem12px’)); //NaN |
| parseFloat(string)函數 （重要） | 將string類型轉成浮點數（即小數）數值型 | parseFloat(’78.91’)  console.log(parseFloat(‘12px’)); //12  console.log(parseFloat(‘rem12px’)); //NaN |
| Number() 強制轉換函數 | 將string類型轉換為數值型 | Number(‘12) |
| js隱式轉換（- \* /） | 利用算術運算隱式轉換為數值型 | ‘12’-0  console.log(‘12’-0); //12  console.log(‘123’ – ‘120’); //3 |

### （4）把其它類型轉換為布爾型

|  |  |  |
| --- | --- | --- |
| 方式 | 說明 | 案例 |
| Boolean()函數 | 其它類型轉成佈爾值 | Boolean(‘true’); |

代表空、否定的值會被轉換為false，如’’、0、NaN、null、undefined

其餘的值都會被轉換為true

## 15 拓展學習：解釋型語言和編譯型語言，標識符，關鍵字，保留字

### （1）解釋型語言和編譯型語言

計算機不能直接理解任何除了機器語言以外的語言，所以必須要把程序員所寫的程序語言翻譯成機器語言才能執行程序。把程序語言翻譯成機器語言的工具，叫翻譯器。



翻譯器翻譯的方式有兩種：一個是編譯，另一個是解釋。兩種方式之間的區別在於翻譯的時間點不同。

編譯器是在代碼執行之前進行編譯，生成中間代碼文件。比如寫了一百行代碼，會把代碼全部編譯完成，然後再執行。好比先把所有的菜做好，才開始吃飯。

解釋器是在運行時進行及時解釋，並立即執行（當編譯器以解釋方式運行時，也稱為解釋器）。比如寫了一百行代碼，會解釋一行代碼，就執行一行代碼。好比吃火鍋，邊涮邊吃。



### （2）標識符，關鍵字，保留字

標識（zhi4聲）符：開發人員為變量、屬性、函數、參數取的名字。

標識符不能是關鍵字或保留字。

關鍵字：JS採用的詞。

比如：break , case , catch , continue ,default , delete , do , else , finally , for , function , if , in , instanceof, new , return , switch , this , throw , try , typeof , var , void , while , with 等。

保留字：預留的“關鍵字”。雖然還不是關鍵字，但是未來可能會成為關鍵字，同樣不能使用它們當變量名或方法名。

比如：boolean , byte , char , class , const , debugger, double , enum , export , extends , final , float , goto , implements , import , int , interface , long , native , package , private , protected , public , short , static , super , synchronized, throws , transient , volatile 等。

在VSCode中，如果出現波浪線，就很有可能寫的是關鍵詞和保留字，所以無需刻意記憶，根據VSCode的提示即可。

## 16 運算符

運算符（operator）也叫操作符，用於賦值、比較和執行運算功能的符號。

JavaScript中常見的運算符：

* 算數運算符
* 遞增和遞減運算符
* 比較運算符
* 邏輯運算符
* 賦值運算符

## 17 算數運算符



console.log(3 % 5); //3

判斷一個數是否能被整除，是%運算符的主要用途。主要取餘為0，就能被整除。

算數運算符也有優先級：有括號先算括號裡面的，沒括號先乘除後加減。

浮點數運算會有問題

    <script>

      console.log(0.3 + 0.52); //0.8200000000000001

      console.log(0.32788 \* 100); //32.788000000000004

      var sum = 0.3 + 0.52;

      console.log(sum == 0.82); //false

    </script>

原因，小數點右邊的數本身很小，還要轉換成2進制再運算，會出差錯，別的語言也會有這種現象。所以盡量避免用浮點數運算。

不要直接拿浮點數來進行比較是否相等。

浮點數值的最高精度是17位小數，但在進行算數計算時，其精度遠遠不如整數。

**表達式和返回值**

表達式：由數字、運算符、變量等能求得數值的有意義排列方式。

表達式是由數字、運算符、變量等組成的式子。

表達式最終都有一個結果返回給我們，這個結果是返回值。

在程序裡，表達式右邊計算完畢然後把返回值返回給左邊，比如var num = 3+6;

## 18 遞增和遞減運算符

### （1）遞增和遞減運算符概述

給變量加1或減1，通過遞增運算符（++）和遞減運算符（--）來完成。

++和--既可以放在變量前面（前置遞增/遞減運算符），也可以放在變量後面（後置遞增/遞減運算符）。

遞增和遞減運算符必須配合變量使用。

### （2）前置遞增運算符和後置遞增運算符

前置遞增運算符

++num相當於num = num + 1

先自加1，後返回值（參與運算）。

後置遞增運算符

num++也相當於num = num + 1

先返回原值參與運算，後自加。例子：

    <script>

      var age = 10;

      console.log(age++ + 10); //20

      console.log(age); //11

      var e = 10;

      var f = e++ + ++e; //e++参与运算的是10，然后e=11；经过++e后，e=12，参与运算的是12，所以f=10+12

      console.log(f); //22

    </script>

前置遞增運算符和後置遞增運算符如果單獨使用，效果是一樣的。

前端開發後置遞增/減用得更多，寫的時候單獨佔一行。

## 19 比較運算符

比較運算符（也叫關係運算符）是兩個數據進行比較時所使用的運算符。

比較運算後，會返回一個佈爾值（true/false）作為比較運算的結果。

|  |  |  |  |
| --- | --- | --- | --- |
| 運算符名稱 | 說明 | 案例 | 結果 |
| < | 小於號 | 1<2 | true |
| > | 大於號 | 1>2 | false |
| >= | 大於等於號 | 2>=2 | true |
| <= | 小於等於號 | 3<=2 | false |
| == | 等號  只要等號兩側的值相等即可，不需要數據類型也一致。  默認轉換數據類型，會把字符串類型的數據轉換為數字型 | 37==37  37==’37’ | true  true |
| != | 不等號 | 37!=37 | false |
| === !== | 全等，要求數值和數據類型都一致 | 37 === ‘37’ | false |

## 20 邏輯運算符

邏輯運算符是用來進行佈爾值運算的運算符，其返回值也是佈爾值。開發中經常用於多個條件的判斷。

|  |  |  |
| --- | --- | --- |
| 邏輯運算符 | 說明 | 案例 |
| && | 邏輯與，簡稱“與” and  只要兩側有1個結果為false，則結果為false | true&&false |
| || | 邏輯或，簡稱“或” or  只要兩側有1個結果為true，則結果為true | true||false |
| ! | 邏輯非，簡稱“非” not | !true |

    <script>

      console.log(!true); //false

      console.log(!false); //true

    </script>

短路運算（邏輯中斷）

短路運算的原理：當有多個表達式（值）時，左邊的表達式值可以確定結果時，就不再繼續運算右邊的表達式的值。

邏輯中斷-邏輯與

語法：表達式1&&表達式2

如果第一個表達式的值為真，則返回表達式2。

如果第一個表達式的值為假，則返回表達式1。

    <script>

      console.log(123 && 456); //456

      console.log(0 && 456); //0

      console.log(0 && 1 + 2 && 456 \* 567889); //0

      console.log("" && 1 + 2 && 456 \* 567889); //''

      //0 '' null undefined NaN这5个都为false，其余为true

    </script>

邏輯中斷-邏輯或

語法：表達式1 || 表達式2

如果第一個表達式的值為真，則返回表達式1。

如果第一個表達式的值為假，則返回表達式2。

    <script>

      console.log(123 || 456); //123

      console.log(0 || 456); //456

      console.log(123 || 456 || 789); //123

    </script>

## 21 賦值運算符

|  |  |  |
| --- | --- | --- |
| 賦值運算符 | 說明 | 案例 |
| = | 直接賦值 | var usrName=’我是值’; |
| +=，-= | 加、減一個數後再賦值 | var age = 10; age += 5; // 15  (相當於age = age + 5) |
| \*=，/=，%= | 乘、除、取模後再賦值 | var age = 2; age \*= 5; //10  （相當於age = age \* 5） |

## 22 運算符優先級

|  |  |  |
| --- | --- | --- |
| 優先級 | 運算符 | 順序 |
| 1 | 小括號 | () |
| 2 | 一元運算符  （只要1個變/常量就可以運算） | ++ -- !  一元運算符例子：++num !num  二元運算符例子： 2+3 |
| 3 | 算術運算符 | \* / % 比+ -權重高 |
| 4 | 關係運算符 | > >= < <= |
| 5 | 相等運算符 | == != === !== |
| 6 | 邏輯運算符 | && 比 || 權重高 |
| 7 | 賦值運算符 | = |
| 8 | 逗號運算符 | , |

## 23 流程控制

在一個程序的執行過程中，各條代碼的執行順序對程序的結果有直接影響。通過控制代碼的執行順序來實現各種功能。

流程控制指控制代碼按照什麼順序執行。

流程控制有三種結構：順序結構，分支結構，循環結構。



## 24 順序流程控制

順序結構是程序中最簡單、最基本的流程控制。它沒有特定的語法結構，程序會按照代碼的先後順序，依次執行，程序中的大多數代碼都是這樣執行的。

## 25 分支流程控制if語句

### （1）分支結構

由上到下執行代碼的過程中，根據不同的條件，執行不同的路徑代碼（執行代碼多選一的過程），從而得到不同的結果。



JS語言提供了兩種分支結構語句

* if語句
* switch語句

### （2）if 語句



if語法結構

if (條件表達式) {

執行語句；

}

執行思路：

如果條件表達式為真，才執行花括號裡面的執行語句。否則執行if語句後面的代碼。

### （3）if else 語句（雙分支語句）



語法結構

if (條件表達式) {

如果條件成立執行的代碼；

} else {

否則執行的代碼；

}

執行思路：

條件成立，執行if裡面的代碼；否則執行else裡面的代碼。

### （4）if else if語句（多分支語句）



利用多個條件來選擇不同的語句執行，得到不同的結果。多選1的過程。

語法：

if (條件表達式1) {

語句1；

} else if (條件表達式2) {

語句2；

} else if (條件表達式3) {

語句3；

} else {

最後的語句；

}

執行思路：

如果滿足條件表達式1，執行語句1；執行完畢後，退出整個if分支語句。如果不滿足條件表達式1，則判斷是否滿足條件表達式2。如果滿足條件表達式2，則執行語句2。以此類推。如果前面的條件都不滿足，則執行else的最後語句。

else if 可以寫任意多個。

## 26 三元表達式

有三元運算符組成的式子叫三元表達式。

++num（一元表達式，1個變量/常量）

3 + 5 （二元表達式，2個變量/常量）

條件表達式 ? 表達式1 : 表達式2 （三元表達式，3個變量/常量）

執行思路：

如果條件表達式結果為真，則返回表達式1的值；如果條件表達式結果為假，則返回表達式2的值；

## 27 分支流程控制switch語句

switch語句也是多分支語句，基於不同條件執行不同代碼。在針對變量設置一系列的特定值的選項時，就可以使用switch。

語法：

switch(表達式) {

case value1:

執行語句1;

break;

case value2:

執行語句2;

break;

…

default:

執行最後的語句;

}

執行思路：

利用表達式的值和case後面的value值匹配。如果匹配上，就執行該case裡面的語句；如果都沒有匹配上，那麼執行default裡面的語句。

注意：

① switch(表達式) 這表達式通常是個變量

② num的值和case的值匹配時，必須全等，即值和數據類型一致，才能匹配上。

③ 如果當前的case裡面沒有break，則不會退出switch，而是繼續執行下一個case。

switch語句和if else if語句的區別

① 一般情況下，兩者可互換

② switch…case語句通常處理case為確定值的情況，而if…else…語句更靈活，通常用於範圍判斷（大於、等於、小於某個值）

③ switch語句進行條件判斷後直接執行到程序的條件句，效率更高。而if…else…語句有多少種條件，就得判斷多少次。

④ 分支較少時，if…else語句寫起來更簡單。

## 28 循環

目的：簡單高效執行某些重複代碼。

## 29 for 循環

### （1）for循環的執行過程

示例：

for (var i = 1; i <= 100; i++) {

console.log(‘hi’);

}

① 首先執行var i = 1 （i for index）這句話只執行一次。

② i <= 100 是否成立，如果成立，執行花括號{}裡面的內容；若不成立，退出循環。

③ i++ ，讓i+1。這一輪結束。

④ 重複第②步和第③步。

### （2）斷點調試

F12→Sources→找到需要調試的文件



### （3）案例：求1-100之間的整數累加和

    <script>

      var sum = 0;

      for (var i = 1; i <= 100; i++) {

        sum += i;

      }

      console.log(sum);

    </script>

### （4）案例：學生成績

彈出窗口請用戶輸入學生人數，然後請用戶逐個輸入學生成績，接下來給用戶顯示所有學生的平均成績。

    <script>

      var stuN = parseInt(prompt("請輸入學生人數"));

      var sum = 0;

      for (var i = 1; i <= stuN; i++) {

        stuScore = parseInt(

          prompt(

            "請輸入第" +

              i +

              "個學生的成績（" +

              (stuN - i) +

              "個學生成績待輸入）"

          )

        );

        sum += stuScore;

      }

      var ave = sum / stuN;

      alert("平均成績是：" + ave);

    </script>

## 30 雙重for循環

### （1）雙重for循環概述

很多情況下，單層for循環不能滿足我們的需求，比如要打印一個5行5列的圖形，打印一個倒直角三角形等，此時就可以通過循環嵌套來實現。

循環嵌套是指在一個循環語句中再定義一個循環語句的語法結構，例如在for循環語句中，可以再嵌套一個for循環，這樣的for循環叫雙重for循環。

雙重for循環，外層每循環一次，內存for循環全部執行完。

### （2）案例：一行打印5顆星，一共5行

    <script>

      var star = "";

      for (i = 1; i <= 5; i++) {

        for (j = 1; j <= 5; j++) {

          star += "★";

        }

        star += "\n";

      }

      console.log(star);

    </script>

### （3）打印三角形

第1行10個星星，共10行，每一行星星數目依次遞減1個。



    <script>

      var star = "";

      for (i = 1; i <= 10; i++) {

        for (j = i; j <= 10; j++) {

          star += "★";

        }

        star += "\n";

      }

      console.log(star);

    </script>

第1行1個星星，共10行，每一行星星數目依次遞增1個。



    <script>

      var star = "";

      for (i = 1; i <= 10; i++) {

        for (j = 1; j <= i; j++) {

          star += "★";

        }

        star += "\n";

      }

      console.log(star);

    </script>

### （4）案例：打印99乘法表

    <script>

      var timesTable = "";

      for (var i = 1; i <= 9; i++) {

        for (var j = 1; j <= i; j++) {

          timesTable += j + "X" + i + "=" + j \* i + "\t";

        }

        timesTable += "\n";

      }

      console.log(timesTable);

    </script>

## 31 while循環

### （1）while循環介紹

while語句在條件表達式為真的前提下，循環執行指定的一段代碼，直到表達式不為真時結束循環。

語法：

while (條件表達式) {

//循環體代碼

}

執行思路：

① 執行條件表達式，如果結果為true，執行循環體代碼；如果結果為false，退出循環，執行後面的代碼。

② 執行循環體代碼。

③ 循環體代碼執行完畢後，繼續執行條件表達式。回到第①步。

while循環應該有計數器，循環體裡應該更新計數器，防止死循環。

### （2）案例：輸入正確的下聯，否則會一直彈框。

進入頁面，彈框請用戶輸入“物華天寶”正確的下聯（“人傑地靈”），否則會持續彈框。

    <script>

      var answer = prompt("物華天寶？");

      while (answer !== "人傑地靈") {

        answer = prompt("物華天寶？");

      }

    </script>

## 32 do while循環

### （1）do while循環介紹

do while循環是while循環的變體。和while循環相比，do while至少會執行一次循環體語句。

語法：

do {

//循環體語句

} while (循環判斷條件)

執行思路：

該循環會先執行一次代do{}裡的 碼塊，然後對while後面括號裡的條件表達式進行判斷，如果條件為真，就會重複執行do循環體，否則退出循環。

### （2）do while循環案例

打印1-100歲

    <script>

      var age = 1;

      do {

        console.log("今年" + age + "岁了");

        age++;

      } while (age <= 100);

    </script>

如果無法回答出正確的答案，會一直出現彈框。

    <script>

      var answer = "";

      do {

        answer = prompt("物華天寶");

      } while (answer !== "人傑地靈");

      alert("答對啦");

    </script>

## 33 循環小結

JS循環中有for、while、do while。

三個循環很多情況下都可以相互代替使用。

如果用來計次數，跟數字有關，三者使用基本相同，for最常用。

while和do…while可以做更複雜的條件判斷，比for循環更靈活。

while和do…while執行順序不一樣。while先判斷後執行，do…while先執行一次，再判斷執行。

while和do…while執行次數不一樣。do…while至少會執行一次循環體，而while可能一次也不執行。

實際工作中，更常用for循環。這是學習重點。

## 34 continue break

### （1）continue

continue關鍵字用於立即跳出本次循環，繼續下一次循環。

案例：

打印的時候不會打印i=3時候的循環。

    <script>

      for (let i = 0; i < 6; i++) {

        if (i === 3) {

          continue; //只要遇見continue，就會終止運行循環體，直接跳到i++

        }

        console.log("第" + i + "個循環");

      }

    </script>

求1-100之間，除了能被7整除以外的所有整數的和。（答案：4315）

    <script>

      var sum = 0;

      for (let i = 1; i <= 100; i++) {

        if (i % 7 === 0) {

          continue;

        }

        sum += i;

      }

      console.log(sum);

    </script>

### （2）break

break關鍵字用於立即跳出整個循環（循環結束）。

例子：如果i===7，則跳出整個循環：

    <script>

      for (let i = 0; i < 10; i++) {

        if (i === 7) {

          break;

        }

        console.log(i);

      }

    </script>

## 35 JS命名規範以及語法格式

（1）標識符命名規範

變量、函數的命名必須要有意義，不能是關鍵字、保留字。

變量的名稱一般用名詞。

函數的名稱一般用動詞。

（2）操作符規範

操作符（比如=， <=）的左右兩側各保留一個空格。

（3）單行注釋規範

// 單行注釋和“//”之間有個空格

（4）其它規範

比如寫for循環，括號外面左右各有1個空格，右側花括號和for對齊

## 36案例：簡易ATM

① alter讓用戶選擇操作：1. 存錢，2. 取錢，3. 顯示餘額，4. 退出。

② 根據用戶的相關選擇給用戶實現相關功能。

③ 原始餘額10000。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>ATM</title>

    <!-- 2. 內嵌式js -->

    <script>

      var balance = 10000;

      // 显示4个选项

      function showChoices() {

        return parseFloat(

          prompt("请输入您的选择：\n1. 存錢\n2. 取錢\n3. 顯示餘額\n4. 退出")

        );

      }

      var userChoice = showChoices();

      // 检查输入的金额，如果不是数字，或者小于0，则需要break。

      function checkNumber(*num*) {

        if (isNaN(num) || num < 0) {

          alert("请输入一个正数");

          return "break";

        }

      }

      // 显示余额

      function showBalance(*balance*) {

        return alert("您的余额是：￥" + balance);

      }

      //判断用户输入的是1-4之间的整数，则进入while循环，否则提示用户输入1-4之间的整数。

      if (userChoice > 0 && userChoice < 5) {

        while (userChoice > 0 && userChoice < 5) {

          if (userChoice === 1) {

            var save = parseFloat(prompt("请输入您要存入的金额"));

            if (checkNumber(save) === "break") {

              break;

            }

            balance += save;

            showBalance(balance);

            userChoice = showChoices();

          } else if (userChoice === 2) {

            var withdrawal = parseFloat(prompt("请输入您要取出的金额"));

            if (checkNumber(withdrawal) === "break") {

              break;

            }

            if (balance < withdrawal) {

              alert("您目前的账户余额是" + balance + "元，待取金额超过余额.");

              break;

            }

            balance -= withdrawal;

            showBalance(balance);

            userChoice = showChoices();

          } else if (userChoice === 3) {

            showBalance(balance);

            userChoice = showChoices();

          } else if (userChoice === 4) {

            break;

          }

        }

      } else {

        alert("请输入1-4之间的整数");

        location.reload();

      }

    </script>

  </head>

  <body>

    <!-- 1.行内式的js 直接写到元素的内部-->

    <input type="button" value="ATM" onclick="location.reload();" />

  </body>

</html>

## 37 數組的概念

數組可以把一組先關的數據一起存放，並提供方便的訪問（獲取）方式。

數組：把一組數據存儲在單個變量名下。

數組裡面的元素沒有限制，什麼類型都可以。

## 38 創建數組

兩種方式創建數組：

* new創建數組
* 數組字面量創建數組（更常用）

（1）利用new創建數組

var 數組名 = new Array();

var arr = new Array(); //創建一個新的空數組

這種方法暫且了解，等學完對象再看。

注意Array()，大寫A。

（2）利用數組字面量創建數組。

方括號[ ]是數組的字面量。

① 使用數組字面量方式創建空的數組

var 數組名 = [ ];

② 使用數組字面量方式創建帶初始值的數組

var music = [‘宮’, ‘商’, ‘角’, ‘徵’, ‘羽’];

## 39 獲取數組中的元素

索引（下標）：用來訪問數組元素的序號（數組下標從0開始）。

獲取數組元素，格式為 數組名[索引號]

例子：

var music = [‘宮’, ‘商’, ‘角’, ‘徵’, ‘羽’];

console.log(music[2]); //’角’

console.log(music[5]); //undefined 因為沒有這個數組元素

## 40 遍歷數組

### （1）遍歷數組方法

用for循環遍歷數組。

遍歷：把數組中的每個元素從頭到尾都訪問一次。

    <script>

      var music = ["宮", "商", "角", "徵", "羽"];

      for (let i = 0; i < music.length; i++) {

        console.log(music[i]);

      }

    </script>

### （2）案例：求數組最大值

    <script>

      var arr = [2, 6, 1, 77, 52, 25, 7, 333];

      var larger = arr[0];

      for (let i = 1; i < arr.length; i++) {

        if (arr[i] > larger) {

          larger = arr[i];

        }

      }

      console.log(larger);

    </script>

### （3）案例：數組轉換為字符串

    <script>

      var arr = ["宫", "商", "角", "徵", "羽"];

      var str = "";

      var sep = "|";

      for (let i = 0; i < arr.length; i++) {

        str += arr[i] + sep;

      }

      console.log(str);

    </script>

## 41 數組中新增元素

通過修改length長度和索引號增加數組元素。

不要直接給數組名字賦值（比如賦個string），否則數組會被新的值（比如string）代替。

## 42 數組案例

### （1）篩選數組

這裡的巧妙之處用到了.length作為新數組的序號。

    <script>

      var arr1 = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];

      var arr2 = [];

      for (let i = 0; i < arr1.length; i++) {

        if (arr1[i] >= 10) {

          arr2[arr2.length] = arr1[i];

        }

      }

      console.log(arr2);

    </script>

### （2）冒泡排序

冒泡排序是一種算法，把一系列的數據按照一定的順序進行排列顯示（從小到大或從大到小）。

例子：把數組按從小到大的順序排列

    <script>

      var arr = [3, 5, 15, 1, 600, 30];

      for (let i = 0; i < arr.length - 1; i++) {

        for (let j = 0; j < arr.length - 1 - i; j++) {

          if (arr[j] > arr[j + 1]) {

            var temp = arr[j];

            arr[j] = arr[j + 1];

            arr[j + 1] = temp;

          }

        }

      }

      console.log(arr);

    </script>

## 43 函數的概念

JavaScript中，有時會定義非常多的相同代碼或者功能相似的代碼，這些代碼可能需要大量重複使用。雖然for循環語句也能實現一些簡單的重複操作，但是功能有限，函數可以更好地解決代碼重複的問題。

函數封裝了一段可以被重複調用執行的代碼塊。通過調用函數，可以避免寫重複代碼。

類比榨汁機：輸入原料（水果），內部處理，輸出果汁。

function test(參數1，參數2，……){ 輸入參數

函數體; 內部處理

return 返回值; 返回結果

}

## 44 函數的使用

使用函數分為2步：①聲明函數，②調用函數。

### （1）聲明函數

function 函數名() {

// 函數體

}

function全部小寫。函數名一般用動詞。函數不調用的話，不會自動執行。

例子：

function sayHi(){

console.log(‘hi’);

}

### （2）調用函數

語法：

函數名();

調用函數時不要忘記加小括號()。

例子：

sayHi();

函數的封裝是把一個或者多個功能通過函數的方式封裝起來，對外只提供一個簡單的函數接口。

## 45 函數的參數

### （1）參數要點

利用參數可以根據需要調整函數體的某些代碼。

語法：

聲明函數：

function 函數名(形參1, 形參2, …){

//函數體

}

調用函數：

函數名(實參1, 實參2,…);

形參類似於變量，實參類似於變量的值。形參可以沒有，也可以有，個數不限。

形參和實參之間用逗號隔開。

    <script>

      function cook(*arg*) {

        console.log(arg);

      }

      cook("燕麦粥");

    </script>

### （2）案例：求任意两个整数及其之间的整数的和

    <script>

      // 求任意两个整数及其之间的整数的和

      function getSum2(*a*, *b*) {

        var start = a < b ? a : b;

        var end = a > b ? a : b;

        var sum = 0;

        for (let i = start; i <= end; i++) {

          sum += i;

        }

        console.log(sum);

      }

      getSum2(1, 10);

      getSum2(-1, -10);

    </script>

### （3）函數形參和實參個數不匹配問題。

如果實參的個數多於形參個數，函數只取前面的參數，多出來的參數不管了。

    <script>

      function getSum(*a*, *b*) {

        var sum = a + b;

        console.log(sum);

      }

      getSum(2, 3, 9);

    </script>

如果實參的個數小於形參的個數，沒對應實參的形參就是undefined。

下面這個例子，a=2，b=undefined，undefined加上數字/true/false，結果是NaN（參看undefined那一小節）

    <script>

      function getSum(*a*, *b*) {

        var sum = a + b;

        console.log(sum);

      }

      getSum(2);

    </script>

## 46 函數的返回值

### （1）返回值的功能

① 希望函數將值返回給調用者，通過使用return語句實現。

    <script>

      function getResults() {

        return "Fantastic";

      }

      console.log(getResults());

    </script>

② return只能返回一個值。如果用逗號隔開多個值，以最後一個為準。

比如

function test(num1, num2){

return num1, num2;

}

console.log(test(1,2)); //返回的是2

③ 如果要返回多個值，返回一個數組或者對象。

④ 如果沒有return，則返回undefined。

    <script>

      function test1() {

        return 666;

      }

      console.log(test1()); //666

      function test2() {

        666;

      }

      console.log(test2()); //undefined

    </script>

### （2）案例：利用函數求數組裡的最大值

    <script>

      var arr = [5, 2, 99, 101, 67, 777];

      function largest(*arr*) {

        var largest = arr[0];

        for (let i = 1; i < arr.length; i++) {

          if (largest < arr[i]) {

            largest = arr[i];

          }

        }

        return largest;

      }

      console.log(largest(arr));

    </script>

### （3）終止函數的功能

return除了返回數據外，還有終止函數的功能。return後面的代碼不會被執行。

所以return後面一般不寫代碼。

### （4）案例：簡易計算器

    <script>

      var num1 = parseFloat(prompt("请输入第1个数"));

      var mathSymbol = prompt("请输入运算符号");

      var num2 = parseFloat(prompt("请输入第2个数"));

      var result = 0;

      function cal(*num1*, *mathSymbol*, *num2*) {

        switch (mathSymbol) {

          case "+":

            result = num1 + num2;

            break;

          case "-":

            result = num1 - num2;

            break;

          case "\*":

            result = num1 \* num2;

            break;

          case "/":

            result = num1 / num2;

            break;

          default:

            break;

        }

        return result;

      }

      cal(num1, mathSymbol, num2);

      alert("结果是：" + result);

    </script>

### （5）案例：判斷是否為質數

    <script>

      function isPrimeNum(*num*) {

        for (var i = 2; i < num; i++) {

          if (num % i === 0) {

            return false;

          }

        }

        return true;

      }

      console.log(isPrimeNum(13337));

    </script>

## 47 arguments的使用

### （1）arguments講解

當不確定有多少個參數傳遞的時候，可以用arguments來獲取。

在JavaScript中，arguments是每個函數的一個內置對象，arguments對象中存儲了傳遞的所有實參。

arguments展示形式是一個偽數組（看上去像數組，但實際不是數組），因此可以進行遍歷。偽數組具有以下特點：

* 具有length屬性
* 按索引方式存儲數據
* 不具有數組的一些方法（比如push、pop）

### （2）案例：判斷輸入任意個值的最大值。

    <script>

      function max() {

        var max = arguments[0];

        for (let i = 1; i < arguments.length; i++) {

          if (max < arguments[i]) {

            max = arguments[i];

          }

        }

        return max;

      }

      var maxResult = max(1, 578, 8, 2, 55, 70, 2, 100);

      console.log(maxResult);

    </script>

## 48 函數案例

### （1）用函數翻轉數組

    <script>

      function reverseArr(*arr*) {

        var reverseArr = [];

        for (let i = arr.length - 1; i >= 0; i--) {

          reverseArr[reverseArr.length] = arr[i];

        }

        return reverseArr;

      }

      var arr1 = [1, 2, 3, 4, 5];

      console.log(reverseArr(arr1));

    </script>

### （2）冒泡排序由小到大

    <script>

      function arrNumSmToLg(*arr*) {

        for (let i = 0; i < arr.length - 1; i++) {

          for (let j = 0; j < arr.length - 1 - i; j++) {

            if (arr[j] > arr[j + 1]) {

              temp = arr[j];

              arr[j] = arr[j + 1];

              arr[j + 1] = temp;

            }

          }

        }

        return arr;

      }

      var arr1 = [80, 23, 1, 6, 999];

      console.log(arrNumSmToLg(arr1));

    </script>

### （3）判斷是否閏年

    <script>

      function isLeapYear(*year*) {

        var leapYearType = year % 100 === 0 ? year % 400 : year % 4;

        return leapYearType === 0 ? true : false;

      }

      console.log(isLeapYear(2200));

    </script>

### （4）函數可以調用另外一個函數

每個函數都是獨立的代碼塊，用於完成特殊任務，常有函數互相調用的情況。

案例：用戶輸入年份，然後為其展示這一年的二月一共多少天。這裡第2個函數裡面調用了第1個函數來判斷這一年是閏年還是平年。

    <script>

      function isLeapYear(*year*) {

        var leapYearType = year % 100 === 0 ? year % 400 : year % 4;

        return leapYearType === 0 ? true : false;

      }

      function febDayNum() {

        var year = parseInt(prompt("请输入年份"));

        var leapYear = isLeapYear(year);

        var febDayNum = leapYear ? 29 : 28;

        return alert("这一年的二月一共有" + febDayNum + "天");

      }

      febDayNum();

### （5）計算器。用了函數，do while循環，switch，while循環

    <script>

      function getChoice() {

        var choice = parseFloat(

          prompt(

            "歡迎使用計算器：\n1. 加法運算\n2. 減法運算\n3. 乘法運算\n4. 除法運算\n5. 退出\n請輸入您的選項："

          )

        );

        if (!Number.isInteger(choice) || !(choice >= 1 && choice <= 5)) {

          alert("請輸入1-5之間的整數");

          location.reload();

        }

        return choice;

      }

      function cal(*symbol*) {

        do {

          var num1 = parseFloat(prompt("請輸入第1個數字"));

        } while (isNaN(num1));

        do {

          var num2 = parseFloat(prompt("請輸入第2個數字"));

        } while (isNaN(num2));

        switch (symbol) {

          case "+":

            var result = num1 + num2;

            break;

          case "-":

            var result = num1 - num2;

            break;

          case "\*":

            var result = num1 \* num2;

            break;

          case "/":

            var result = num1 / num2;

            break;

          default:

            break;

        }

        alert(num1 + symbol + num2 + "的結果是" + result);

        choice = getChoice();

      }

      var choice = getChoice();

      while (Number.isInteger(choice) && choice >= 1 && choice <= 5) {

        switch (choice) {

          case 1:

            cal("+");

            break;

          case 2:

            cal("-");

            break;

          case 3:

            cal("\*");

            break;

          case 4:

            cal("/");

            break;

          case 5:

            choice = 0;

            break;

          default:

            break;

        }

      }

    </script>

## 49 函數的兩種聲明方式

### （1）命名函數

利用函數關鍵字自定義函數

function 函數名(){

//函數體

}

例如：

定義

function saving(save){

console.log(save);

}

調用

saving(100000)

### （2）匿名函數：函數表達式

var 變量名 = function(){};

這種命名方式和命名變量一樣，只不過變量裡存的是值，這裡存的是函數。

定義

var saving = function(save) {

console.log(save);

}

調用

saving(100000)

這種命名方式寫出的函數，是沒有名字的，saving是變量名，不是函數名，所以這種函數叫匿名函數。

## 50 作用域

### （1）全局作用域和局部作用域

限定一個名字（通常是變量）在什麼代碼範圍可用，這個代碼範圍就是作用域。

這樣可以減少命名衝突。

JS的作用域（es6之前），分為全局作用域和局部作用域。

① 全局作用域：整個script標籤，或者一個單獨的js文件。

② 局部作用域：函數內部。代碼的名字只在函數內部其效果。

    <script>

      var num = 10; // 全局作用域里面的变量

      function saving() {

        var num = 20; // 局部作用域里面的变量

      }

    </script>

以上2個num雖然名字相同，但作用域不同，所以不會有衝突，都能起作用。

### （2）變量根據作用域分類

變量根據作用域分類可以分為全局變量和局部變量。

① 全局變量是在全局作用域裡的變量。在全局（包括函數裡面）都可以使用。

如果在函數內部沒有聲明而直接賦值的變量，也是全局變量。

② 局部變量是局部作用域下的變量。只在局部（函數裡面）使用。

函數的形參也可看做局部變量，不能在全局使用。

    <script>

      var num1 = 10; // 全局作用域里面的变量，为全局变量。

      function test1() {

        var num2 = 20; // 局部作用域里面的变量，只能在本函数里面使用

        console.log(num1); //可以直接使用全局变量num1

        num3 = 30; // 没声明直接赋值的变量，属于全局变量

      }

      test1();

      console.log(num3); //30 （先调用test1函數才行，否则報錯：Uncaught ReferenceError: num3 is not defined）

      console.log(num2); //局部變量不能在全局使用，報錯：Uncaught ReferenceError: num2 is not defined

    </script>

### （3）執行效率

① 全局變量只有在瀏覽器關閉的時候才會被銷毀，比較佔用內存資源。

② 局部變量當程序執行完畢就銷毀，比較節約內存資源。

### （4）es6之前，JS沒有塊級作用域（了解）

JS在es6的時候，新增了塊級作用域。

用花括號{}包裹的，就是塊級作用域。比如if{ } for { }

有塊級作用域的概念，比如

if(XX){

int num = 10;

}

在外面不能調用num

而es6之前，JS沒有塊級作用域的概念，所以下面這樣寫沒問題。

var a =2;

if (a < 5) {

var num = 10;

}

console.log(num);

### （5）作用域鏈

如果函數中還有函數，在這個局部作用域中又有一個局部作用域。

根據內部函數可以訪問外部函數變量的機制，用鏈式查找決定哪些數據能被內部函數訪問。這種結構叫作用域鏈。

比如下圖，這樣一層一層往上查找的機制，就是鏈式查找。簡單來說，就是就近原則。



## 51 預解析

JavaScript代碼是由瀏覽器中的JavaScript解析器（引擎）來執行的。

JavaScript解析器在運行JavaScript代碼的時候分兩步：預解析和執行代碼。

① 預解析

js引擎會把js裡面所有的var和function提升到當前作用域的最前面。

預解析分為 變量預解析（變量提升）和 函數預解析（函數提升）。

② 執行代碼

按照代碼書寫的順序從上往下執行。

### （1）變量預解析

變量預解析（變量提升）：把所有的變量聲明提升到**當前的作用域**最前面，**不提升賦值操作**。

      console.log(num); //undefined

      var num = 10;

相當於執行了以下代碼

      var num;

      console.log(num);

      num = 10;

下面是個匿名函數的例子

    <script>

      fun1(); // 报错：Uncaught TypeError: fun1 is not a function

      var fun1 = function () {

        console.log(22);

      };

    </script>

這是個匿名函數 fun1是個變量。這個時候僅僅把變量fun1的聲明提到最前面了，但並沒有給fun1賦值一個函數，所以調用函數fun1()時會報錯。

相當於執行了以下代碼：

    <script>

      var fun1;

      fun1(); // 报错：Uncaught TypeError: fun1 is not a function

      fun1 = function () {

        console.log(22);

      };

    </script>

### （2）函數預解析（函數提升）

函數提升：把所有的函數聲明提升到當前作用域的最前面，**不調用函數**。

所以下列代碼能正常執行：

    <script>

      fun2(); //22

      function fun2() {

        console.log(22);

      }

    </script>

相當於執行了下列代碼：

    <script>

      function fun2() {

        console.log(22);

      }

      fun2(); //22

    </script>

### （3）預解析案例

#### 案例①

    <script>

      var num = 10;

      fun(); //undefined

      function fun() {

        console.log(num);

        var num = 20;

      }

    </script>

相當於執行了

    <script>

      var num;

      function fun() {

        var num;

        console.log(num); //找最近的num，即上一行的num，这个时候已被声明但还未被赋值，所以结果是undefined

        num = 20;

      }

      num = 10;

      fun(); //undefined

    </script>

#### 案例②

    <script>

      f1();

      console.log(c); //9

      console.log(b); //9

      console.log(a); //报错 Uncaught ReferenceError: a is not defined

      function f1() {

        var a = (b = c = 9); //有時寫成var a = b = c = 9

        console.log(a); //9

        console.log(b); //9

        console.log(c); //9

      }

    </script>

相當於運行了下面的代碼：

    <script>

      function f1() {

        var a;

        a = b = c = 9; // var a = b = c =9;相当于

        // var a = 9；

        // b = 9;

        // c = 9;

        //b和c直接赋值，没有var声明，当全局变量看。

        //var a = b = c =9;和集体声明var a = 9, b = 9, c = 9;完全不同，不要混淆。

        //集体声明var a = 9, b = 9, c = 9;相当于

        //var a = 9;

        //var b = 9;

        //var c = 9;

        console.log(a); //9

        console.log(b); //9

        console.log(c); //9

      }

      f1();

      console.log(c); //9

      console.log(b); //9

      console.log(a); //报错 Uncaught ReferenceError: a is not defined

    </script>

## 52 對象

### （1）什麼是對象

對象是一個具體的事物，比如一個人，一本書，一輛車。

明星不是對象，周星馳是對象；班主任不是對象，我們的班主任是對象；蘋果不是對象，這個蘋果是對象；手機不是對象，這個iPhoneX是對象；遊戲不是對象，絕地求生是對象。

### （2）JavaScript中的對象

在JavaScript中，對象是一組無序的相關屬性和方法的集合。

所有的事物都是對象，例如字符串、數值、數組、函數等。

對象屬於複雜數據類型。

對象由屬性和方法組成。

* 屬性：事物的**特征**，在對象中用屬性來表示（常用名詞）
* 方法：事物的**行為**，在對象中用方法來表示（常用動詞）

例子：手機這個對象

屬性：大小，顏色，重量，屏幕尺寸。

方法：打電話，發短信，玩遊戲。

### （3）變量、屬性、函數、方法的共同點和區別

① 變量和屬性相同點：都用來存儲數據。

* 變量：單獨聲明（加var）並賦值，使用的時候直接寫變量名。單獨存在。
* 屬性：在對象裡面，不需要加var聲明；使用的時候必須是對象.屬性或者對象[‘屬性’]。

② 函數和方法相同點：都用來實現某種功能，做某件事。

* 函數時單獨聲明，調用 函數名() 單獨存在。
* 方法在對象裡面，調用的時候 對象.方法()。
* 有時候函數和方法混著叫。

### （4）對象表達結構更清晰。

## 53 創建對象的三種方式

### （1）用字面量創建對象

對象字面量：花括號{}，裡面包含了表達這個具體事務（對象）的屬性和方法。

創建對象例子：

    <script>

      var obj = {

        uname: "凌风",

        age: 22,

        gender: "女",

        sayHi: function () {

          console.log("hi");

        },

      };

    </script>

注意：

① 對象裡面的屬性或者方法採取鍵值對的形式 鍵（屬性名）：值（屬性值）

屬性值可以是任意類型的數值（數字類型，字符串類型，布爾類型，函數類型等）

② 對個屬性或者方法中間用逗號隔開

③ 方法冒號後面跟的是一個匿名函數

調用對象屬性，採取 對象名.屬性名 的形式。例子：

console.log(obj.uname);

調用對象屬性，採取 對象名[‘屬性名’] （必須加引號） 的形式。例子：

console.log(obj["age"]);

調用對象方法，採取 對象名.方法名() （必須加小括號）的形式。例子：

      obj.sayHi();

### （2）用new Object創建對象

例子：

    <script>

      var obj = new Object();

      obj.uname = "凌风";

      obj.age = 22;

      obj.gender = "女";

      obj.sayHi = function () {

        console.log("hi");

      };

    </script>

注意：

① 利用等號 = 賦值的方法 添加對象的屬性和方法。

② 每個屬性和方法之間用分號結束。

### （3）用構造函數創建對象

前面兩種創建對象的方法，創建多個對象時，會有許多重複的代碼。可以用構造函數解決這種冗餘。

構造函數，是把對象裡面一些相同的屬性和方法抽象出來封裝到函數裡面。

語法：

function 構造函數名() {

this.屬性 = 值;

this.方法 = function() {}

}

new 構造函數名();

例子：

    <script>

      function Artist(*uname*, *age*, *gender*) {

        this.name = uname;

        this.age = age;

        this.gender = gender;

      }

      var lf = new Artist("凌风", 22, "女");

      console.log(typeof lf);

      var wy = new Artist("王佑", 23, "女");

      console.log(wy);

    </script>

注意：

① 構造函數名字首字母大寫。

② 構造函數不需要return，就可以返回結果。

③ 調用構造函數，必須使用new。

④ 只要new Artist()，就調用函數創建了一個對象。

⑤ 屬性和方法前面必須添加this。

#### ① 構造函數和對象的區別

構造函數是泛指的某一大類（比如藝術家），類似於java的類（class）。

對象是特指的某一個人事物（比如梵高）

用構造函數創建對象的過程也稱為對象的實例化。

#### ② new關鍵字執行過程

a. new構造函數在內存中創造了一個空的對象。

b. this指向這個空對象。

c. 執行構造函數裡面的代碼，給這個空對象添加通過參數傳進來的屬性和方法。

d. 返回這個對象。

## 54 遍歷對象屬性

for…in 語句用於對數組或者對象的屬性進行循環操作。

語法 for (變量 in 對象) {

}

例子

    <script>

      function Artist(*uname*, *age*, *gender*) {

        this.name = uname;

        this.age = age;

        this.gender = gender;

      }

      var lf = new Artist("凌风", 22, "女");

      for (var k in lf) {

        console.log(k); //得到的是属性名 name age gender

        console.log(lf[k]); //得到的是属性值 凌风 22 女

      }

    </script>

## 55 內置對象

### （1）什麼是內置對象

JavaScript中的對象分為3種：自定義對象（程序員自己創建的），內置對象，瀏覽器對象。

自定義對象和內置對象是JS基礎內容，屬於ECMAScript。瀏覽器對象是JS獨有的，詳見JS API章節。

內置對象是JS語言自帶的一些對象，供開發者使用，並提供了一些常用的功能（屬性和方法）。

類比，手機的打電話、發短信功能，汽車的發動機、剎車功能（不需要關注這些功能如何實現的，直接用就行。）

常用的JavaScript內置對象：Math，Date，Array，String。

### （2）查文檔

學習一個內置對象的使用方法，只要學會其常用成員的使用即可，可通過查閱MDN或者W3C文檔來學習。

講師更推薦MDN（網址<https://developer.mozilla.org/zh-CN/> ）。Mozilla開發者網絡（MDN）提供了有關開放網絡技術（Open Web）的信息，包括HTML，CSS和萬維網及HTML5應用的API。

學習對象中的方法：

① 學習該方法的功能

② 查看參數的意義和類型

參數如果寫在中括號[ 裡，表示可有可無

③ 查看返回值的意義和類型

④ 通過demo進行測試

### （3）Math對象

#### ① Math對象講解

Math數學對象不是一個構造函數，所以不需要new來調用，直接使用其屬性和方法即可。

常用的Math對象中的屬性和方法

Math.PI 圓周率

Math.floor() 向下取整

Math.ceil() 向上取整

Math.round() 四捨五入，就近取整。注意 Math.round(-3.5)，結果是-3；Math.round(-3.6)，結果是-4 以 .5 結尾的負數特殊，不遵守四捨五入的法則，而是直接把 .5 去掉。

Math.abs() 絕對值

Math.max() 最大值

Math.min() 最小值

#### ② Math.random()

a 返回一個隨機的小數，假設其為X：0≤X＜1。

b 這個方法不跟參數

    <script>

      // 包含最小值，不包含最大值

      function getRandomInt(*min*, *max*) {

        min = Math.ceil(min);

        max = Math.floor(max);

        return Math.floor(Math.random() \* (max - min) + min); //The maximum is exclusive and the minimum is inclusive

      }

      // 最小值和最大值都包含

      function getRandomIntInclusive(*min*, *max*) {

        min = Math.ceil(min);

        max = Math.floor(max);

        return Math.floor(Math.random() \* (max - min + 1) + min); //The maximum is inclusive and the minimum is inclusive

      }

    </script>

Math.random()案例 隨機點名

    <script>

      // 最小值和最大值都包含

      function getRandomIntInclusive(*min*, *max*) {

        min = Math.ceil(min);

        max = Math.floor(max);

        return Math.floor(Math.random() \* (max - min + 1) + min); //The maximum is inclusive and the minimum is inclusive

      }

      var nameList = ["凌風", "謝楨", "張文遠", "趙令畤", "林楓眠", "江澄"];

      var num = getRandomIntInclusive(0, nameList.length - 1);

      console.log(nameList[num]);

    </script>

#### ③ 案例：猜數字

任意生成一個猜1-10之間的整數，讓用戶來猜。用戶每次輸入數字，都和生成的數字比較，以此提示用戶猜大了還是猜小了。

    <script>

      function getRandomIntInclusive(*min*, *max*) {

        min = Math.ceil(min);

        max = Math.floor(max);

        return Math.floor(Math.random() \* (max - min + 1) + min);

      }

      var rnum = getRandomIntInclusive(1, 10);

      console.log(rnum);

      function getUnum() {

        var unum = parseInt(

          prompt("隨機生成的1個1-10之間的整數，你猜是哪個？")

        );

        while (!Number.isInteger(unum) || !(unum >= 1 && unum <= 10)) {

          unum = parseInt(prompt("請輸入1-10之間的整數"));

        }

        return unum;

      }

      var unum = getUnum();

      do {

        if (unum > rnum) {

          alert("你猜的數大了，得往小了猜");

          unum = getUnum();

        } else if (unum < rnum) {

          alert("你猜的數小了，得往大了猜");

          unum = getUnum();

        }

      } while (rnum > unum || rnum < unum);

      alert("猜中了");

    </script>

  </head>

  <body>

#### ④ 案例：猜數字（限定了次數）

這是案例③的修改版，限定了猜測次數為3次

    <script>

      function getRandomIntInclusive(*min*, *max*) {

        min = Math.ceil(min);

        max = Math.floor(max);

        return Math.floor(Math.random() \* (max - min + 1) + min);

      }

      var rnum = getRandomIntInclusive(1, 10);

      console.log(rnum);

      function getUnum(*count*) {

        var rightWord = count === 0 ? "共" : "還";

        var unum = parseInt(

          prompt(

            "隨機生成的1個1-10之間的整數，你猜是哪個？你" +

              rightWord +

              "有" +

              (round - count) +

              "次機會"

          )

        );

        while (!Number.isInteger(unum) || !(unum >= 1 && unum <= 10)) {

          unum = parseInt(prompt("請輸入1-10之間的整數"));

        }

        return unum;

      }

      var count = 0;

      var round = 3;

      var unum = getUnum(count);

      do {

        count++;

        if (count >= round) {

          break;

        }

        if (unum > rnum) {

          alert("你猜的數大了，得往小了猜");

          unum = getUnum(count);

        } else if (unum < rnum) {

          alert("你猜的數小了，得往大了猜");

          unum = getUnum(count);

        }

      } while (rnum > unum || rnum < unum);

      if (rnum === unum) {

        alert("猜中了！就是" + rnum);

      } else {

        alert("木有機會啦。正確的數字是" + rnum);

      }

    </script>

### （4）日期對象

#### ① 新建日期對象

和Math不一樣，日期對象是個構造函數，必須使用new來創建日期對象。

日期對象Date用來處理日期和時間。

如果Date()括號裡有時間參數，就返回參數裡的時間。

下面這三種寫法都可以

    <script>

      var date1 = new Date("2019-5-1");

      console.log(date1);

      var date2 = new Date("2019/5/1");

      console.log(date2);

      var date3 = new Date("2019-10-1 8:8:8");

      console.log(date3);

    </script>

#### ② 日期格式化

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 代碼 |
| getFullYear() | 獲取當年 | dObj.getFullYear() |
| getMonth() | 獲取當月（0-11） | dObj.getMonth() |
| getDate() | 獲取當天日期 | dObj.getDate() |
| getDay() | 獲取星期幾（週日0，週一到到週六1到6） | dObj.getDay() |
| getHours() | 獲取當前小時 | dObj.getHours() |
| getMinutes() | 獲取當前分鐘 | dObj.getMinutes() |
| getSeconds() | 獲取當前秒鐘 | dObj.getSeconds() |

獲取星期幾的例子

    <script>

      var date1 = new Date();

      var wk = ["週日", "週一", "週二", "週三", "週四", "週五", "週六"];

      var wkkey = date1.getDay();

      console.log("今天是" + wk[wkkey]); //今天是週五

    </script>

#### ③ 獲取日期的總毫秒形式

用Date對象可以獲取從1970年1月1日到現在的總毫秒數。

通過下面3種方法獲得

    <script>

      // 1. 通過valueOf() 和getTime()方法獲得

      var date1 = new Date();

      console.log(date1.valueOf());

      console.log(date1.getTime());

      // 2. 通過在new前面寫+獲得（最常用的寫法）

      var date2 = +new Date();

      console.log(date2);

      // 3. H5新增的獲得總的毫秒數（注意兼容性）

      console.log(Date.now());

    </script>

#### ④ 案例：倒計時的函數

    <script>

      function countdown(*futureTime*) {

        var nowTime = +new Date();

        var timeDiffBySec = (futureTime - nowTime) / 1000;

        var d = parseInt(timeDiffBySec / 60 / 60 / 24);

        h = h < 10 ? "0" + h : h;

        var h = parseInt((timeDiffBySec / 60 / 60) % 24);

        h = h < 10 ? "0" + h : h;

        var m = parseInt((timeDiffBySec / 60) % 60);

        m = m < 10 ? "0" + m : m;

        var s = parseInt(timeDiffBySec % 60);

        s = s < 10 ? "0" + s : s;

        return "倒計時：" + d + "天" + h + "時" + m + "分" + s + "秒";

      }

      var date1 = +new Date("2022-10-1 8:8:8");

      console.log(countdown(date1));

    </script>

### （5）數組對象

#### ① 創建數組對象

創建數組對象的兩種方式：

* 字面量方式
* new Array()

    <script>

      // 1. 利用數組字面量

      var arr1 = [1, 2, 3];

      console.log(arr1[0]);

      // 2. 利用new Array();

      var arr2 = new Array(); //創建了一個空數組

      var arr3 = new Array(2); //這個2表示數組的長度為2，裡面有2個空的數組元素

      var arr4 = new Array(2, 3); //等價於[2,3]。這樣寫表示裡面有2個數組元素，分別是2和3.

    </script>

#### ② 檢測是否為數組

    <script>

      // 方法1 instanceof Array

      var arr = [];

      var obj = {};

      console.log(arr instanceof Array); //true

      console.log(obj instanceof Array); //false

      // 方法2 Array.isArray(參數); H5新增的方法  ie9以上版本支持。

      console.log(Array.isArray(arr));  //true

      console.log(Array.isArray(obj)); //false

    </script>

#### ③ 添加刪除數組元素的方法

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 返回值 |
| push(參數1…) | 在數組末尾添加一個或多個元素，原數組改變。 | 返回新的長度 |
| pop() | 刪除數組最後一個元素，把數組長度減1。  無參數，原數組改變。 | 返回被刪除的元素的值 |
| unshift(參數1…) | 向數組的開頭添加一個或更多元素，原數組改變。 | 返回新的長度。 |
| shift() | 刪除數組的第一個元素，數組長度減1無參數，原數組改變。 | 返回第一個元素的值。 |

push() 注意

* push()可以給數組追加新的元素
* push()的參數直接寫數組的元素即可
* push()完畢後，返回的結果是新數組的長度
* 原數組會發生變化

    <script>

      // push() 在數組的末尾添加一個或多個數組元素。

      var arr1 = [1, 2, 3];

      arr1.push(4, "天朗氣清");

      console.log(arr1);

      console.log(arr1.push(4, "天朗氣清")); //5 返回添加元素後後數組的長度

    </script>

unshift() 注意：

* unshift()給數組前面加新的元素
* unshift()的參數直接寫數組的元素即可
* unshift()完畢後，返回的結果是新數組的長度
* 原數組會發生變化

    <script>

      // unshift() 在數組的開頭添加一個或多個數組元素。

      var arr1 = [1, 2, 3];

      arr1.unshift(0, "清泉");

      console.log(arr1);

      console.log(arr1.unshift(0, "清泉")); //5 返回添加元素後後數組的長度

    </script>

pop() 注意：

    <script>

      var arr1 = [1, 2, "清風"];

      arr1.pop();

      console.log(arr1.pop()); //清風

    </script>

* pop()刪除數組的最後一個元素，一次只能刪除一個元素。
* pop()沒有參數。
* pop()完畢後，返回的結果是被刪除的元素。
* 原數組會發生變化 。

shift() 注意

    <script>

      var arr1 = ["水波", 2, "清風"];

      arr1.shift();

      console.log(arr1.shift()); //水波

    </script>

* shift()刪除數組的第一個元素，一次只能刪除一個元素。
* shift()沒有參數。
* shift()完畢後，返回的結果是被刪除的元素。
* 原數組會發生變化 。

#### ④ 數組排序

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 該方法是否修改原數組 |
| reverse() | 顛倒數組中元素的順序，無參數。 | 會改變，返回新數組。 |
| sort() | 對數組的元素進行排序（大到小或者小到大）。 | 會改變，返回新數組。 |

例子：

    <script>

      // 1 翻轉數組

      var arr1 = [1500, 5200, 2000, 2100, 1800];

      arr1.reverse();

      console.log(arr1);

      // 2 數組排序（冒泡排序）

      var arr2 = [1500, 5200, 2000, 2100, 1800];

      arr2.sort(function (*a*, *b*) {

        // return a - b; //升序排列

        return b - a; //降序排列

      });

      console.log(arr2);

    </script>

#### ⑤ 數組索引

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 返回值 |
| indexOf() | 查找數組中第1個索引值。  只返回滿足條件的第1個索引號。 | 如果存在，返回索引號；  如果不存在，返回-1。 |
| lastIndexOf() | 查找數組中的最後一個索引值。  只返回滿足條件的第1個索引號。 | 如果存在，返回索引號；  如果不存在，返回-1。 |

例子

    <script>

      var arr1 = ["宮", "商", "角", "徵", "羽", "徵"];

      console.log(arr1.indexOf("角")); //2

      console.log(arr1.indexOf("徵")); //3 雖然有2個，但只返回第1個符合條件的

      console.log(arr1.lastIndexOf("徵")); //5  雖然有2個，但只返回第1個符合條件的

    </script>

#### 數組索引案例：查數組中的重複數據

查數組中的重複數據的索引（數組indexArr），以及一共出現幾次（indexSum）。

    <script>

      function checkRep(*arr*, *rep*) {

        var indexArr = [];

        var beginIndex = 0;

        var indexSum = 0;

        do {

          indexArr.push(arr.indexOf(rep, beginIndex));

          beginIndex = arr.indexOf(rep, beginIndex) + 1;

          indexSum++;

        } while (arr.indexOf(rep, beginIndex) !== -1);

        console.log(indexArr, indexSum);

      }

      var arr = ["a", "b", "c", "d", "b", "a", "c", "a", "c", "a", "d", "b"];

      checkRep(arr, "a");

    </script>

#### ⑥ 重點案例：刪除數組中的重複元素

目標：把舊數組裡面不重複的元素取出來放到新數組中，重複的元素只保留1個在新數組。

核心算法：遍歷舊數組，拿著舊數組的每個元素去新數組查詢，看看該元素在新數組中是否出現過，如果沒出現過，就把該元素添加到新數組中。

如何知道元素是否存在於數組中？利用 新數組.indexOf(數組元素) 如果返回-1，說明新數組中沒有該元素。

    <script>

      var arr = ["c", "a", "z", "a", "x", "a", "x", "c", "b"];

      function delRep(*arr*) {

        if (!(arr instanceof Array)) {

          alert("請輸入數組");

          return;

        }

        newArr = [];

        for (let i = 0; i < arr.length; i++) {

          if (newArr.indexOf(arr[i]) === -1) {

            newArr.push(arr[i]);

          }

        }

        return newArr;

      }

      console.log(delRep(arr)); //["c", "a", "z", "x", "b"]

    </script>

#### ⑦ 數組轉換為字符串

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 返回值 |
| toString() | 把數組轉換成字符串，用逗號分隔每一項。 | 返回1個字符串 |
| join(‘分隔符’) | 如果不加參數，默認用逗號分隔字符串裡的內容。  如果加了參數，參數為分隔符。 | 返回1個字符串 |

    <script>

      // 1 toString()

      var arr1 = [1, 2, 3];

      console.log(arr1.toString()); //字符串 1,2,3

      // 2 join('分隔符')

      var arr2 = ["萬", "家", "燈", "火"];

      console.log(arr2.join()); //默認用逗號分隔字符串裡面的內容    萬,家,燈,火

      console.log(arr2.join("-")); //字符串   萬-家-燈-火

    </script>

#### ⑧ 其它重要的數組方法

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 返回值 |
| concat() | 連接兩個或多個數組，不影響原數組。 | 返回一個新數組 |
| slice() | 數組截取slice(begin, end) | 返回截取出來的數組 |
| splice() | 刪除數組 splice(第幾個開始，要刪除個數) | 返回被刪除數組的新數組。  會影響原數組。 |

### （6）字符串對象

#### ① 基本包裝類型

String，Number和Boolean都可以被包裝成複雜數據類型。

    <script>

      var str = "andy";

      console.log(str.length);

      // 疑問：對象這種複雜數據類型才有屬性和方法，string這種簡單數據為什麼會有length屬性呢？

      // 答案：因為把簡單的數據類型包裝成了複雜數據類型，這樣基本數據類型就有了屬性和方法。

      // 基本包裝類型：把簡單數據類型包裝成複雜數據類型。有以下3步：

      // （1）把簡單數據類型包裝成為複雜數據類型

      var temp = new String("andy");

      // （2）把臨時變量的值給str

      str = temp;

      // （3） 銷毀這個臨時變量

      temp = null;

    </script>

#### ② 字符串的不可變

      var str;

      str = "andy";

      str = "萬";

上面的代碼中，在給str賦值”萬”的時候，並沒有佔用原來開闢給”andy”的空間，而是新開闢一個空間給”萬”，這叫字符串的不可變。如下圖所示。



所以在開發中，不要大量給字符串重新賦值，也不要大量地拼接字符串，否則太耗資源。

比如說下面的代碼，如果num = 100還好，因為值小，如果num數值非常大，運行速度就很慢了。

    <script>

      var str = "";

      var num = 100;

      for (let i = 0; i < num; i++) {

        str += i;

      }

      console.log(str);

    </script>

也因為如此，字符串中所有的方法都不會修改字符串本身，操作完成後會返回一個新的字符串。

#### ③ 根據字符串返回位置

|  |  |
| --- | --- |
| 方法名 | 說明 |
| indexOf(‘要查找的字符’, 開始的位置) | 返回指定內容在字符串中的位置，只返回第1個符合條件的，如果找不到就返回-1。開始的位置時index索引號。 |
| lastIndexOf() | 從後往前找，只返回第1個匹配的。 |

    <script>

      var str = "物華天寶人傑地靈天";

      console.log(str.indexOf("天")); //2

      // 從索引號是3的位置（即寶所在的位置）開始往後查找

      console.log(str.indexOf("天", 3)); //8

    </script>

#### ④ 案例：string中重複出現的字符

    <script>

      // 求string中重複出現的字符出現的位置和次數

      function checkRep(*str*, *rep*) {

        var oIndex = [];

        var oSum = 0;

        var beginPl = 0;

        do {

          oIndex.push(str.indexOf(rep, beginPl));

          oSum++;

          beginPl = str.indexOf(rep, beginPl) + 1;

        } while (str.indexOf(rep, beginPl) !== -1);

        console.log(oIndex, oSum);

      }

      var str = "asdowerovbdoweovdf";

      checkRep(str, "o");

    </script>

#### ⑤ 根據位置返回字符（重點）

|  |  |  |
| --- | --- | --- |
| 方法名 | 說明 | 使用 |
| charAt(index) | 返回指定位置（index索引號）的字符 | str.charAt(3) |
| charCodeAt(index) | 獲取指定位置處（index索引號）字符的ASCII碼。  鍵盤上所有的按鍵（包括shift，space，enter）都有對應的ASCII碼。在開發中用於判斷用戶按了哪個鍵。 | str.charCodeAt(2) |
| str[index] | 獲取指定位置處字符 | HTML5，IE8+才支持，和charAt()等效。使用時注意兼容性。 |

例子：

    <script>

      var str = "andy";

      console.log(str.charAt(3)); //y

      console.log(str.charCodeAt(0)); //97

      console.log(str[3]); //y

    </script>

#### ⑥ 案例：求字符串中出現頻率最高的字符及其出現次數

    <script>

      var str = "abacdabadcdabdadccd";

      // 建一個空對象。鍵為str的單個字符，值為元素在str中出現的次數。遍歷str的字符，如果該字符是obj中的鍵，則該鍵的值+1；如果該字符不是obj的鍵，則把該字符新增為obj的鍵，並設值為1。

      var obj = {};

      for (let i = 0; i < str.length; i++) {

        if (obj[str[i]]) {

          obj[str[i]]++;

        } else {

          obj[str[i]] = 1;

        }

      }

      console.log("obj:");

      console.log(obj);

      // 用var max = Math.max(...arr);求數組的最大值;Object.values(obj)可以返回對象所有鍵值對中值所組成的數組；

      var max = Math.max(...Object.values(obj));

      console.log("max:");

      console.log(max);

      // 求最大值在數組中的位置，可能有1個，也可能有幾個。因為最大值有可能只有1個，有可能有幾個。利用這位置，可以得知obj裡的對應的鍵是什麼

      var arrObjMaxK = [];

      for (let j = 0; j < Object.values(obj).length; j++) {

        if (max === Object.values(obj)[j]) {

          arrObjMaxK.push(j);

        }

      }

      console.log("arrObjMaxK:");

      console.log(arrObjMaxK);

      // 遍歷arrObjMaxK裡面的值，去obj裡把最大的鍵取出來；Object.keys(obj)可以返回對象所有鍵值對中鍵所組成的數組；

      var arrPopular = [];

      for (let k = 0; k < arrObjMaxK.length; k++) {

        for (let l = 0; l < Object.keys(obj).length; l++) {

          if (arrObjMaxK[k] === l) {

            arrPopular.push(Object.keys(obj)[l]);

          }

        }

      }

      console.log("arrPopular:");

      console.log(arrPopular);

    </script>

#### ⑦ 字符串操作方法（重點）

|  |  |
| --- | --- |
| 方法名 | 說明 |
| concat(str1, str2, str3…) | concat()方法用於連接兩個或多個字符串。拼接字符串，等效於+， +更常用。 |
| substr(start, length) | 從start（索引號）位置開始，length為取的個數。重點 |
| slice(start, end) | 從start位置開始，截取到end位置。end取不到。start和end都是索引號。 |
| substring(start, end) | 從start位置開始，截取到end位置。end取不到。基本和slice相同，不接受負值。 |
| substr，slice和substring基本功能一致，都是截取字符串，重點掌握substr(start, length)。 | |
| replace(‘被替換的字符’, ‘替換為的字符’) | 替換字符。如果有重複的，只替換第1個。如果需要全部替換，用循環來做。詳見下面的例子 |
| split(‘分隔符’) | 字符串轉換為數組 |

    <script>

      var str1 = "andy";

      var str2 = "linda";

      console.log(str1.concat("red")); //andyred

      console.log(str2.concat("red", "blue")); //lindaredblue

      var str3 = "物華天寶人傑地靈";

      console.log(str3.substr(2, 2)); //天寶  第1個參數2是索引號，指從第2個開始；第2個參數2是取幾個字符。

    </script>

    <script>

      // 替換字符串

      var str1 = "andyn";

      var str2 = "andynn";

      console.log(str1.replace("a", "b")); // bndyn;

      console.log(str2.replace("n", "t")); // atdynn

      var str3 = "qweqsdsaqxczqwqaqcz";

      while (str3.indexOf("q") !== -1) {

        str3 = str3.replace("q", "\*");

      }

      console.log(str3); //\*we\*sdsa\*xcz\*w\*a\*cz

      // 把字符串轉換為數組

      var str4 = "blue&yellow&green&red";

      console.log(str4.split("&")); //["blue", "yellow", "green", "red"]

    </script>

## 56 數據類型，堆棧，內存分配，傳參

### （1）簡單類型和複雜類型

簡單數據類型又叫做基本數據類型或者值類型，複雜類型又叫做引用類型。

值類型：簡單數據類型/基本數據類型，在存儲時變量中存儲的是值本身，因此叫做值類型。

string，number，boolean，undefined，null都是值類型

注意：

      // 簡單數據類型null 返回的是一個空對象 object

      var timer = null;

      console.log(typeof timer); //object

      // 如果有個變量打算以後存儲為對象，暫時沒想好放什麼，這時候就給null

引用類型：複雜數據類型在存儲時，變量中存儲的僅僅是地址（引用），因此叫做引用數據類型。

通過new關鍵字創建的對象（系統對象，自定義對象），如Object，Array，Date等。

### （2）堆和棧

① 棧（操作系統）：由操作系統自動分配釋放存放函數的參數值、局部變量的值等。其操作方式類似於數據結構中的棧。

簡單數據類型存放到棧裡面。

② 堆（操作系統）：一般由程序員分配釋放，若程序員不釋放，由垃圾回收機制回收。

複雜數據類型存放到堆裡面。



JavaScript中沒有堆棧的概念，通過堆棧的類比，可以更容易理解某些代碼的執行方式，便於將來學習其它語言。

### （3）內存分配



① 簡單數據類型，存放在棧裡面，裡面直接開闢一個空間，存放值。

② 複雜數據類型，在棧裡面存放地址（系統自動生成的16進制數），這個地址指向堆裡面的數據。

————

### （4）傳遞的參數是簡單類型

函數的形參可以看作一個變量，當把一個值類型變量作為參數傳遞給函數的形參時，就把變量在棧空間裡的值複製了一份給形參，那麼在方法內部對形參做任何修改，都不會影響到外部變量。



### （5）傳遞的參數是複雜類型

    <script>

      function Person(*name*) {

        this.name = name;

      }

      function f1(*x*) { // x=p

        console.log("2" + x.name); //劉

        x.name = "張";

        console.log("3" + x.name); //張

      }

      var p = new Person("劉");

      console.log("1" + p.name); //劉

      f1(p);

      console.log("4" + p.name); //張

    </script>



函數的形參也可以看作一個變量。把引用類型變量傳給形參時，其實是把變量在棧空間裡保存的堆地址複製給了形參，形參和實參保存的是同一個堆地址，所以操作的是同一個對象。

WebAPI（DOM操作）

## 1 Web APIs和JS基礎關聯性



JS基礎階段

* 學習ECMAScript標準規定的基本語法
* 要掌握JS基礎語法
* 但只學基礎語法，做不了頁面網頁交互效果。學基礎語法是為了JS後面的課程做鋪墊。

Web APIs階段

* Web APIs是W3C組織的標準
* Web APIs 階段主要學習DOM和BOM
* Web APIs 是JS獨有的部分
* 主要學習頁面交互功能
* 以JS基礎課程為基礎

總結：

JS基礎學習ECMAScript基礎語法為後面做鋪墊。

Web APIs是JS的應用，大量使用JS基礎語法做交互效果。

## 2 API和Web API

① API（Application Programming Interface，應用程序編程接口）是一些預先定義的函數，目的是提供應用程序與開發人員基於某軟件或硬件得以訪問一組例程的能力，而又無需訪問源碼，或理解內部工作機制的細節。

簡單理解：API是給程序員提供的一種工具，以便能更輕鬆地實現想要完成的功能。

類比：

手機的充電接口

* 不關心手機內部變壓器等怎麼工作的
* 不關心充電線如何製作的
* 只要知道拿著充電線插進充電接口就可以充電
* 這個充電接口就是一個API

② Web API 是瀏覽器提供的一套操作瀏覽器功能和頁面元素的API（BOM和DOM）。針對瀏覽器做交互效果。

MDN詳細API： <https://developer.mozilla.org/zh-CN/docs/Web/API>

例子：alert彈框，做輪播圖，獲得頁面元素。

Web API一般都有輸入和輸出（函數的傳參和返回值），Web API很多都是方法（函數）。

學習Web API可以結合前面學習內置對象方法的思路學習。（這個API是幹嘛的，是否需要參數，返回值是什麼，代碼驗證）

## 3 DOM 簡介

### （1）DOM定義

文檔對象模型（Document Object Model，簡稱DOM），是W3C組織推薦的處理可擴展標記語言（HTML或者XML）的標準編程接口。

W3C定義了一系列的DOM接口，通過這些DOM接口可以改變網頁的內容、結構和樣式。

### （2）DOM樹



文檔：一個頁面就是一個文檔，DOM中用document表示。

元素：頁面中所有標籤都是元素，DOM中用element表示。

節點：頁面中的所有內容都是節點（標籤、屬性、文本、注釋等），DOM中用node表示。

DOM把以上內容都看做對象。比如把a元素取出來，a可以看做對象，有自己的屬性和方法。

## 4 獲取元素

DOM在實際開發中主要用來操作元素。

獲取到的DOM元素是個對象（object），所以叫文檔對象模型（Document Object Model）。

要獲取頁面中的元素，可以通過如下方式：

* 根據ID獲取
* 根據標籤名獲取
* 通過HTML5新增的方法獲取
* 特殊元素獲取

### （1）根據ID獲取

使用getElementById()方法可以獲取帶有ID的元素對象。

ID是大小寫敏感的字符串，代表了所要查找的元素的唯一ID。

返回一個匹配ID的DOM Element對象，如果在當前Document下沒找到，返回null。

  <body>

    <div id="time">2002-1-1</div>

    <script>

      // 文檔頁面從上往下加載。要拿標籤的id，先得有標籤，所以script寫在標籤下面。

      // getElementById駝峰法命名

      // 參數id是大小寫敏感的字符串

      // 返回的是一個元素對象

      var timer = document.getElementById("time");

      console.log(timer);

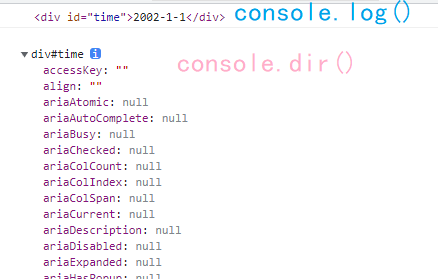
      console.log(typeof timer);

      // console.dir 打印返回的元素對象，可以更好地查看元素對象的屬性和方法

      console.dir(timer);

    </script>

  </body>



console.dir()和console.log()的區別

1.針對一個json對象而言,差別不大。

2.針對html來說。 console.log()打印的是element的內容樹,而console.dir()打印的是element的所有屬性和方法。

### （2）根據標籤名獲取

getElementsByTagName()方法可以返回帶有指定標籤名對象的集合。

返回的是獲取過來的元素對象的集合，以偽數組的形式存儲。for循環遍歷可以獲取裡面所有的對象。

getElementsByTagName()用法：

* document. getElementsByTagName()
* element. getElementsByTagName()

  <body>

    <ul>

      <li>物華天寶</li>

      <li>物華天寶</li>

    </ul>

    <ul id="rj">

      <li>人傑地靈</li>

      <li>人傑地靈</li>

    </ul>

    <script>

      var lis = document.getElementsByTagName("li");

      console.log(lis);

      console.log(lis[0]);

      var rj = document.getElementById("rj");

      var lirj = rj.getElementsByTagName("li");

      console.log(lirj);

    </script>

  </body>

### （3）通過HTML5新增的方法獲取

電腦端要注意兼容性。在移動端可以放心使用。

* document.getElementsByClassName(‘類名’); //根據類名返回元素對象集合
* document.querySelector(‘選擇器’); //根據指定選擇器返回第一個元素對象
  + 如果有多個相同的類/id/標籤，返回的是第1個
  + 傳入的參數是string，如果是類要加.，如果是id要加#
* document.querySelectorAll(‘選擇器’); //根據指定選擇器返回所有元素對象集合
  + 傳入的參數是string，如果是類要加.，如果是id要加#

  <body>

    <ul>

      <li class="wh">物華天寶1</li>

      <li class="wh">物華天寶2</li>

    </ul>

    <ul id="rj">

      <li>人傑地靈3</li>

      <li>人傑地靈4</li>

    </ul>

    <script>

      var lis = document.getElementsByClassName("wh");

      console.log(lis);

      var firstwh = document.querySelector(".wh"); //選出類為wh的第一個元素

      console.log(firstwh);

      var rj = document.querySelector("#rj"); //選出id為rj的第一個元素

      console.log(rj);

      var li = document.querySelector("li");

      console.log(li);

      var whs = document.querySelectorAll(".wh"); //選出類為wh的元素對象集合

      console.log(whs);

    </script>

  </body>

### （4）特殊元素獲取

  <body>

    <script>

      // 獲取body元素 document.body

      var bodyEle = document.body;

      console.log(bodyEle); //成功獲取

      // 獲取HTML元素 document.documentElement

      var htmlEle1 = document.html; //錯誤寫法

      var htmlEle2 = document.documentElement;

      console.log(htmlEle1); //undefined

      console.log(htmlEle2); //成功獲取

    </script>

  </body>

## 5 事件基礎

### （1）什麼是事件

事件是可以被JavaScript偵測到的行為。

例子：點擊按鈕，鼠標經過。

  <body>

    <button id="btn">天朗氣清</button>

    <script>

      // 事件由三部分組成：（1）事件源，（2）事件類型，（3）事件處理程序。也稱事件三要素。

      // （1）事件源 事件被觸發的對象：比如一個按鈕

      var btn = document.getElementById("btn");

      // （2）事件類型 發生了什麼事件 比如鼠標點擊（onclick），鼠標經過，鍵盤按下

      // （3）事件處理程序：函數賦值的方式

      btn.onclick = function () {

        alert("惠風和暢");

      };

    </script>

  </body>

### （2）執行事件的步驟：

a 獲取事件源

b 註冊事件（綁定事件）

c 添加事件處理程序（採取函數賦值形式）

### （3）常見的鼠標事件

|  |  |
| --- | --- |
| 鼠標事件 | 觸發條件 |
| onclick | 鼠標點擊左鍵觸發 |
| onmouseover | 鼠標經過觸發 |
| onmouseout | 鼠標離開觸發 |
| onfocus | 獲得鼠標焦點觸發 |
| onblur | 失去鼠標焦點觸發 |
| onmousemove | 鼠標移動觸發 |
| onmouseup | （按下鼠標後）鼠標彈起觸發 |
| onmousedown | 鼠標按下觸發 |

mouseover和mouseenter的區別：

mouseover鼠標經過盒子自身和其子盒子都會觸發。相對的是mouseout。

mouseenter只在鼠標經過盒子本身時觸發，因為mouseenter不會冒泡。相對的是mouseleave。

## 6 操作元素

JavaScript的DOM操作可以改變網頁內容、結構和樣式。

### （1）改變元素內容

改變元素內容，下面兩者都可行。也可用於讀取標籤的內容。

* element.innerText
  + 不識別HTML標籤
  + 非標準
* element.innterHTML
  + 識別HTML標籤
  + W3C標準，更常用

  <body>

    <button id="btn1">天朗氣清</button>

    <button id="btn2">惠風和暢</button>

    <div id="ci">

      燎沉香

      <p>消溽暑</p>

    </div>

    <script>

      var btn1 = document.getElementById("btn1");

      btn1.onclick = function () {

        btn1.innerText = "惠風和暢";

      };

      var btn2 = document.getElementById("btn2");

      btn2.onclick = function () {

        btn2.innerHTML = "<strong>惠風和暢</strong>";

      };

      var div = document.getElementById("ci");

      console.log(div.innerText); //只打印文字

      console.log(div.innerHTML); //連div裡面的p標籤一起打印出來

    </script>

  </body>

### （2）操作常用的元素屬性

innerText，innerHTML改變元素內容。

src，href，id，alt，title

案例，點擊不同的按鈕，顯示不同的圖片。

  <body>

    <button id="hanfu1">漢服1</button>

    <button id="hanfu2">漢服2</button>

    <img id="img" src="./hanfu1.jpg" alt="hanfu1" title="漢服1" />

    <script>

      var btn1 = document.getElementById("hanfu1");

      var btn2 = document.getElementById("hanfu2");

      var img = document.getElementById("img");

      btn1.onclick = function () {

        img.src = "./hanfu1.jpg";

        img.alt = "./hanfu1.jpg";

        img.title = "漢服1";

      };

      btn2.onclick = function () {

        img.src = "./hanfu2.jpg";

        img.alt = "./hanfu2.jpg";

        img.title = "漢服2";

      };

    </script>

  </body>

#### ① 案例：不同的時間顯示不同的問候語

  <body>

    <p id="greeting">上午好</p>

    <script>

      var p = document.getElementById("greeting");

      var date = new Date();

      var presentHour = date.getHours();

      console.log(presentHour);

      if (presentHour >= 18) {

        p.innerHTML = "晚上好";

      } else if (presentHour >= 14) {

        p.innerHTML = "下午好";

      } else if (presentHour >= 12) {

        p.innerHTML = "中午好";

      } else if (presentHour >= 6) {

        p.innerHTML = "上午好";

      } else if (presentHour >= 0) {

        p.innerHTML = "凌晨好";

      }

    </script>

  </body>

### （3）操作表單元素屬性

type，value，checked，selected，disabled

  <body>

    <button id="btn">按鈕</button>

    <input id="input" type="text" value="輸入內容" />

    <script>

      var btn = document.getElementById("btn");

      var input = document.getElementById("input");

      btn.onclick = function () {

        // 表單裡的文字，用input.value來修改，而非input.innerHTML。

        input.value = "點擊了按鈕";

        // 禁用表單，把disable值設為true

        btn.disabled = true; //這相當於this.disabled。this指向事件函數的調用者，這裡指btn。

      };

    </script>

  </body>

#### ① 案例：密碼解密/加密案例



點擊閉眼睛圖片，密碼解密；點擊睜開眼睛圖片，密碼加密。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>密碼解密加密案例</title>

    <style>

      div {

        width: 180px;

        height: 25px;

        border: 1px solid #333;

      }

      input {

        width: 150px;

        height: 23px;

        line-height: 23px;

        vertical-align: top;

        border: 0;

        outline: none;

      }

      img {

        margin-top: 5px;

        width: 15px;

      }

    </style>

  </head>

  <body>

    <div>

      <input id="input" type="password" />

      <img id="img" src="./close.png" alt="close" />

    </div>

    <script>

      var input = document.getElementById("input");

      var img = document.getElementById("img");

      var flag = "close";

      img.onclick = function () {

        if (flag === "close") {

          input.type = "text";

          this.src = "./open.png";

          this.alt = "open";

          flag = "open";

        } else if (flag === "open") {

          input.type = "password";

          this.src = "./close.png";

          this.alt = "close";

          flag = "close";

        }

      };

    </script>

  </body>

</html>

### （4）樣式屬性操作

通過JS修改元素的大小、顏色、位置等樣式。

這樣修改產生的是行內樣式，CSS權重比較高。

* element.style 行內樣式操作
  + style後面跟的樣式採取駝峰命名法，比如element.style.fontSize，element.style.backgroundColor
* element.className 類名樣式操作

例子

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>test</title>

    <style>

      #box {

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

    </style>

  </head>

  <body>

    <div id="box" class="box"></div>

    <script>

      var box = document.getElementById("box");

      box.onclick = function () {

        this.style.backgroundColor = "aquamarine";

      };

    </script>

  </body>

#### ① 案例：點擊關閉二維碼

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>點擊關閉二維碼</title>

    <style>

      #box {

        position: relative;

      }

      #close {

        position: absolute;

      }

      #close:hover {

        cursor: pointer;

      }

      #taobao {

        margin-left: 18px;

      }

    </style>

  </head>

  <body>

    <div id="box" class="box">

      <img id="close" src="./wrong.png" alt="close" />

      <img id="taobao" src="./tao.png" alt="taobao" />

    </div>

    <script>

      var close = document.getElementById("close");

      var box = document.getElementById("box");

      close.onclick = function () {

        box.style.display = "none";

      };

    </script>

  </body>

</html>

#### ② 案例：循環精靈圖

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <style>

        \* {

            margin: 0;

            padding: 0;

        }

        li {

            list-style-type: none;

        }

        .box {

            width: 250px;

            margin: 100px auto;

        }

        .box li {

            float: left;

            width: 24px;

            height: 24px;

            background-color: pink;

            margin: 15px;

            background: url(*images/sprite.png*) no-repeat;

        }

    </style>

</head>

<body>

    <div class="box">

        <ul>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

            <li></li>

        </ul>

    </div>

    <script>

        // 1. 获取元素 所有的小li

        var lis = document.querySelectorAll('li');

        for (var i = 0; i < lis.length; i++) {

            // 让索引号 乘以 44 就是每个li 的背景y坐标  index就是我们的y坐标

            var index = i \* 44;

            lis[i].style.backgroundPosition = '0 -' + index + 'px';

        }

    </script>

</body>

</html>

#### ③ 案例：顯示隱藏input中的文字

* 用到的事件：onfocus和onblur
* 獲得焦點，判斷表單裡面內容是否為默認文字，如果是默認文字，清空表單內容，改變文字顏色。
* 失去焦點，判斷表單內容是否為空，如果為空，則表單內容改為默認文字，改變文字顏色。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>顯示隱藏input中的文字</title>

    <style></style>

    <style>

      input {

        color: #999;

      }

    </style>

  </head>

  <body>

    <input type="text" id="input" value="請輸入搜索關鍵字" />

    <script>

      var input = document.getElementById("input");

      input.onfocus = function () {

        if (this.value === "請輸入搜索關鍵字") {

          this.value = "";

        }

        this.style.color = "#000";

      };

      input.onblur = function () {

        if (this.value === "") {

          this.value = "請輸入搜索關鍵字";

          this.style.color = "#999";

        }

      };

    </script>

  </body>

</html>

### （5）修改樣式屬性，使用element.className案例

點擊div後給它修改多個屬性。先把這些屬性寫在一個類裡，然後依次添加給div。

這會修改div原來的類名。比如這個div原來的類名是original，點擊後其類名為change。

如果想保持原來的類名，可以寫成：this.className = "original change";

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style></style>

    <style>

      .original {

        width: 100px;

        height: 100px;

        background-color: aquamarine;

      }

      div:hover {

        cursor: pointer;

      }

      .change {

        width: 200px;

        height: 200px;

        background-color: gold;

        margin-top: 200px;

        margin-left: 200px;

      }

    </style>

  </head>

  <body>

    <div id="box" class="original"></div>

    <script>

      var box = document.getElementById("box");

      box.onclick = function () {

        this.className = "change";

      };

    </script>

  </body>

</html>

#### ① 案例：判斷用戶輸入的密碼是否符合要求，並給出相應的提示信息

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <style>

        div {

            width: 600px;

            margin: 100px auto;

        }

        .message {

            display: inline-block;

            font-size: 12px;

            color: #999;

            background: url(*images/mess.png*) no-repeat left center;

            padding-left: 20px;

        }

        .wrong {

            color: red;

            background-image: url(*images/wrong.png*);

        }

        .right {

            color: green;

            background-image: url(*images/right.png*);

        }

    </style>

</head>

<body>

    <div class="register">

        <input type="password" class="ipt">

        <p class="message">请输入6~16位密码</p>

    </div>

    <script>

        // 首先判断的事件是表单失去焦点 onblur

        // 如果输入正确则提示正确的信息颜色为绿色小图标变化

        // 如果输入不是6到16位，则提示错误信息颜色为红色 小图标变化

        // 因为里面变化样式较多，我们采取className修改样式

        // 1.获取元素

        var ipt = document.querySelector('.ipt');

        var message = document.querySelector('.message');

        //2. 注册事件 失去焦点

        ipt.onblur = function() {

            // 根据表单里面值的长度 ipt.value.length

            if (this.value.length < 6 || this.value.length > 16) {

                // console.log('错误');

                message.className = 'message wrong';

                message.innerHTML = '您输入的位数不对，密碼必須6~16位';

            } else {

                message.className = 'message right';

                message.innerHTML = '您输入的密碼格式正确';

            }

        }

    </script>

</body>

</html>

### （6）算法：排他思想

如果有同一組元素，想要其中某個元素實現某種樣式，需要用到循環的排他思想，分為兩步：

a. 清除所有元素的樣式

b. 給目標元素設置樣式

案例：點擊一個button，將其變為粉紅色，其它的buttons不變。

  <body>

    <div id="box">

      <button>按钮1</button>

      <button>按钮2</button>

      <button>按钮3</button>

      <button>按钮4</button>

      <button>按钮5</button>

    </div>

    <script>

      var box = document.getElementById("box");

      var btns = box.getElementsByTagName("button");

      for (let i = 0; i < btns.length; i++) {

        btns[i].onclick = function () {

          var btnIndex = i + 1;

          console.log("按钮" + btnIndex);

          for (let j = 0; j < btns.length; j++) {

            btns[j].style.backgroundColor = "";

          }

          this.style.backgroundColor = "pink";

        };

      }

    </script>

  </body>

#### ① 案例：點擊li裡面的圖片，切換整個body的背景圖片



<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      body {

        background: url(*./images/1.jpg*) no-repeat;

      }

      #box {

        margin: 100px auto;

        width: 650px;

        height: 100px;

      }

      #box ul {

        list-style-type: none;

        padding: 0;

      }

      #box ul li {

        float: left;

        width: 160px;

        height: 100px;

        margin-right: 3px;

        cursor: pointer;

      }

      #box ul li:last-child {

        margin-right: 0;

      }

    </style>

  </head>

  <body>

    <div id="box">

      <ul>

        <li></li>

        <li></li>

        <li></li>

        <li></li>

      </ul>

    </div>

    <script>

      var box = document.getElementById("box");

      var lis = box.getElementsByTagName("li");

      var bodyEle = document.body;

      for (let i = 0; i < lis.length; i++) {

        var imgIndex = i + 1;

        lis[i].style.background =

          "url('./images/" + imgIndex + ".jpg') no-repeat";

        lis[i].style.backgroundSize = "160px 100px";

        lis[i].onclick = function () {

          console.log(this.style.background);

          bodyEle.style.background = this.style.background;

          bodyEle.style.backgroundSize = "100%";

        };

      }

    </script>

  </body>

</html>

#### ② 案例：鼠標經過li，加背景色；鼠標離開li，去掉背景色。

這種想法不可行：for循環得到所有li，並加上onmouseover事件。在onmouseover事件裡，先遍歷所有li去掉所有li的背景色，再給當前li加上背景色。

原因：只要onmouseover過，永遠會有1個li有背景色。

用這種方法：for循環得到所有li，給每個li加上onmouseover和onmouseout事件。onmouseover給li加上背景色，onmouseout給li去掉背景色。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      ul {

        list-style-type: none;

        padding: none;

      }

      li {

        cursor: pointer;

      }

    </style>

  </head>

  <body>

    <div id="box">

      <ul>

        <li>1</li>

        <li>2</li>

        <li>3</li>

        <li>4</li>

        <li>5</li>

      </ul>

    </div>

    <script>

      var box = document.getElementById("box");

      var lis = box.getElementsByTagName("li");

      for (let i = 0; i < lis.length; i++) {

        lis[i].onmouseover = function () {

          this.style.backgroundColor = "pink";

        };

        lis[i].onmouseout = function () {

          this.style.backgroundColor = "";

        };

      }

    </script>

  </body>

</html>

#### ③ 案例：是否全選（我的代碼）



<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      #box {

        width: 600px;

        margin: 100px auto;

      }

      #box table {

        border: 1px solid #999;

        border-collapse: collapse;

        text-align: center;

      }

      #box th {

        background-color: lightskyblue;

      }

      #box th,

      #box td {

        width: 200px;

        padding: 10px 0;

        border: 1px solid #999;

      }

      #box td {

        background-color: #d0d0d0;

      }

    </style>

  </head>

  <body>

    <div id="box">

      <table>

        <thead>

          <tr>

            <th>

              <input type="checkbox" id="ckAll" />

            </th>

            <th>商品</th>

            <th>价钱</th>

          </tr>

        </thead>

        <tbody id="tbd">

          <tr>

            <td>

              <input type="checkbox" />

            </td>

            <td>iPhone</td>

            <td>￥6000</td>

          </tr>

          <tr>

            <td>

              <input type="checkbox" />

            </td>

            <td>iPad</td>

            <td>￥5000</td>

          </tr>

          <tr>

            <td>

              <input type="checkbox" />

            </td>

            <td>Mac Mini</td>

            <td>￥16000</td>

          </tr>

        </tbody>

      </table>

    </div>

    <script>

      var ckAll = document.getElementById("ckAll");

      var tbd = document.getElementById("tbd");

      var cks = tbd.getElementsByTagName("input");

      var aCkFlag = false;

      ckAll.onclick = function () {

        // if (ckAll.checked === true) {

        //   for (let i = 0; i < cks.length; i++) {

        //     cks[i].checked = true;

        //   }

        // } else if (ckAll.checked === false) {

        //   for (let i = 0; i < cks.length; i++) {

        //     cks[i].checked = false;

        //   }

        // }

        // 這些被注釋了的代碼可以簡寫為下面兩行

        for (let i = 0; i < cks.length; i++) {

          cks[i].checked = this.checked;

        }

      };

      for (let i = 0; i < cks.length; i++) {

        cks[i].onclick = function () {

          //每次點擊了一下具體商品的複選框，都統計一下一共有幾個複選框選中了；

          var ckedNum = 0;

          for (let j = 0; j < cks.length; j++) {

            if (cks[j].checked === true) {

              ckedNum++;

            }

          }

          if (ckedNum === 3) {

            ckAll.checked = true;

          } else {

            ckAll.checked = false;

          }

        };

      }

    </script>

  </body>

</html>

#### ④ 案例：是否全選（導師的代碼）



<!DOCTYPE html>

<html>

<head lang="en">

    <meta charset="UTF-8">

    <title></title>

    <style>

        \* {

            padding: 0;

            margin: 0;

        }

        .wrap {

            width: 300px;

            margin: 100px auto 0;

        }

        table {

            border-collapse: collapse;

            border-spacing: 0;

            border: 1px solid #c0c0c0;

            width: 300px;

        }

        th,

        td {

            border: 1px solid #d0d0d0;

            color: #404060;

            padding: 10px;

        }

        th {

            background-color: #09c;

            font: bold 16px "微软雅黑";

            color: #fff;

        }

        td {

            font: 14px "微软雅黑";

        }

        tbody tr {

            background-color: #f0f0f0;

        }

        tbody tr:hover {

            cursor: pointer;

            background-color: #fafafa;

        }

    </style>

</head>

<body>

    <div class="wrap">

        <table>

            <thead>

                <tr>

                    <th>

                        <input type="checkbox" id="j\_cbAll" />

                    </th>

                    <th>商品</th>

                    <th>价钱</th>

                </tr>

            </thead>

            <tbody id="j\_tb">

                <tr>

                    <td>

                        <input type="checkbox" />

                    </td>

                    <td>iPhone8</td>

                    <td>8000</td>

                </tr>

                <tr>

                    <td>

                        <input type="checkbox" />

                    </td>

                    <td>iPad Pro</td>

                    <td>5000</td>

                </tr>

                <tr>

                    <td>

                        <input type="checkbox" />

                    </td>

                    <td>iPad Air</td>

                    <td>2000</td>

                </tr>

                <tr>

                    <td>

                        <input type="checkbox" />

                    </td>

                    <td>Apple Watch</td>

                    <td>2000</td>

                </tr>

            </tbody>

        </table>

    </div>

    <script>

        // 1. 全选和取消全选做法：  让下面所有复选框的checked属性（选中状态） 跟随 全选按钮即可

        // 获取元素

        var j\_cbAll = document.getElementById('j\_cbAll'); // 全选按钮

        var j\_tbs = document.getElementById('j\_tb').getElementsByTagName('input'); // 下面所有的复选框

        // 注册事件

        j\_cbAll.onclick = function() {

                // this.checked 它可以得到当前复选框的选中状态如果是true 就是选中，如果是false 就是未选中

                console.log(this.checked);

                for (var i = 0; i < j\_tbs.length; i++) {

                    j\_tbs[i].checked = this.checked;

                }

            }

            // 2. 下面复选框需要全部选中， 上面全选才能选中做法： 给下面所有复选框绑定点击事件，每次点击，都要循环查看下面所有的复选框是否有没选中的，如果有一个没选中的， 上面全选就不选中。

        for (var i = 0; i < j\_tbs.length; i++) {

            j\_tbs[i].onclick = function() {

                // flag 控制全选按钮是否选中

                var flag = true;

                // 每次点击下面的复选框都要循环检查者4个小按钮是否全被选中

                for (var i = 0; i < j\_tbs.length; i++) {

                    if (!j\_tbs[i].checked) {

                        flag = false;

                        break; // 退出for循环 这样可以提高执行效率 因为只要有一个没有选中，剩下的就无需循环判断了

                    }

                }

                j\_cbAll.checked = flag;

            }

        }

    </script>

</body>

</html>

### （7） 自定義屬性

自定義屬性目的：保持並使用數據。

這些數據只用保存在頁面中，而不用保存到數據庫中。

#### ① 獲取自定義屬性

兩種方法：

* element.屬性
  + 獲取內置屬性值（元素自帶屬性）
* element.getAttribute(‘屬性’);
  + 獲取程序員自定義的屬性
  + 自定義屬性例如下面a連接中的index="1"

  <body>

    <a href="www.lbc.com" id="link" alt="lbc" index="1"></a>

    <script>

      var link = document.getElementById("link");

      console.log(link.href);

      console.log(link.getAttribute("alt"));

    </script>

  </body>

#### ② 設置屬性值

兩種方法：

* element.屬性 = “值”; （只能修改自帶的屬性）
* element.setAttribute(“屬性”, “值”)；（既能修改自帶屬性又能修改自定義屬性）

  <body>

    <a href="www.lbc.com" id="link" alt="lbc" index="1"></a>

    <script>

      var link = document.getElementById("link");

      link.href = "www.666.com";

      link.setAttribute("alt", "隨風潛入夜");

      console.log(link.href);

      console.log(link.getAttribute("alt"));

    </script>

  </body>

特殊的，下面兩者等效，但注意一個寫className，一個寫class

div.className = ‘brd’;

div.setAttribute(‘class’, ‘brd’);

#### ③ 移除屬性

element.removeAttribute(‘屬性’);

#### ④ 設置H5自定義屬性

不容易判斷屬性是標籤的內置屬性還是自定義屬性，所以H5新增了自定義屬性。

H5規定自定義屬性data開頭作為屬性名並且賦值。

比如<div data-index=”1”></div>

或者使用JS設置：element.setAttribute(‘data-index’, 2);

獲取H5自定義屬性

* element.getAttribute(‘data-index’);
  + 兼容性很好
  + 任何自定義屬性都能獲取到
* element.dataset.index 或者 element.dataset[‘index’]
  + H5新增的，注意兼容性（ie11開始支持）
  + 只能獲取以data-開頭的自定義屬性
  + element.dataset中存放了所有以data開頭的自定義屬性
  + 注意，自定義屬性裡面有多個“-”連接的單詞，獲取該屬性時，採用駝峰命名法。

<div data-list-name=’andy’></div>

div.dataset.listName;

div.dataset[‘listName’];

## 7 節點操作

### （1）學習節點操作的目的

① 利用DOM提供的方法獲取元素，邏輯性不強，繁瑣。

② 利用節點層級關係（親子，同級）獲取元素，邏輯性強，兼容性稍差。

### （2）節點概述

網頁中的所有內容都是節點（標籤、屬性、文本、注釋等），在DOM中，節點用node表示。

HTML DOM樹中的所有節點均可通過JavaScript訪問，所有HTML元素（節點）均可被修改，可以被創建、刪除。



節點有三個基本屬性：

* nodeType（節點類型）
  + 元素節點nodeType為1
  + 屬性節點nodeType為2
  + 文本節點nodeType為3 （文本節點包含文字、空格、換行等）
* nodeName（節點名稱）
* nodeValue（節點值）

實際開發中，節點操作主要操作的是元素節點。

### （3）上級節點

parentNode得到的是最近的上級節點

  <body>

    <div id="c">

      <div id="b">

        <div id="a"></div>

      </div>

    </div>

    <script>

      var a = document.getElementById("a");

      var b = a.parentNode;

      console.log(b);

    </script>

  </body>

### （4）下級節點

#### ① parentNode.childNodes

返回包含指定節點的下級節點的集合，包含元素節點和文本節點（比如換行）。要拿到元素節點非常麻煩，所以一般不用childNodes

  <body>

    <ul id="a">

      <li>1</li>

      <li>2</li>

      <li>3</li>

      <li>4</li>

      <li>5</li>

    </ul>

    <script>

      var a = document.getElementById("a");

      var b = a.childNodes;

      console.log(b);

      var c = [];

      // 这个b不仅包含了li，还包含了换行，要得到li，必须for循环遍历筛选

      for (let i = 0; i < b.length; i++) {

        if (b[i].nodeType === 1) {

          c.push(b[i]);

        }

      }

      console.log(c);

    </script>

  </body>

#### ② 最常用 parentNode.children

雖然非標準，但是得到了各個瀏覽器的支持，可以放心使用。

只讀屬性。返回所有的子元素節點。

只返回子元素節點，其餘節點不返回。

  <body>

    <ul id="a">

      <li>1</li>

      <li>2</li>

      <li>3</li>

      <li>4</li>

      <li>5</li>

    </ul>

    <script>

      var a = document.getElementById("a");

      var b = a.children;

      console.log(b);

      // 第1个li

      console.log(b[0]);

      // 最后的li

      console.log(b[b.length - 1]);

    </script>

  </body>

#### ③ parentNode.firstChild 和parentNode.lastChild

firstChild和lastChild返回第一個和最後一個子節點（不論文本節點還是元素節點都會返回。），找不到則返回null。

#### ④ parentNode.firstElementChild和parentNode.lastElementChild

這兩個方法有兼容性問題，IE9以上才支持。

返回的是第一個和最後一個子元素節點。

#### ⑤ 案例：鼠標經過離開li顯示隱藏下拉菜單



<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      #a > li {

        float: left;

        margin-right: 100px;

      }

      #a ul {

        display: none;

      }

    </style>

  </head>

  <body>

    <ul id="a">

      <li>

        1

        <ul>

          <li>1-1</li>

          <li>1-2</li>

          <li>1-3</li>

        </ul>

      </li>

      <li>

        2

        <ul>

          <li>2-1</li>

          <li>2-2</li>

          <li>2-3</li>

        </ul>

      </li>

      <li>

        3

        <ul>

          <li>3-1</li>

          <li>3-2</li>

          <li>3-3</li>

        </ul>

      </li>

    </ul>

    <script>

      var a = document.getElementById("a");

      for (let i = 0; i < a.children.length; i++) {

        a.children[i].onmouseenter = function () {

          this.children[0].style.display = "block";

        };

        a.children[i].onmouseleave = function () {

          this.children[0].style.display = "none";

        };

      }

    </script>

  </body>

</html>

### （5）同級節點

#### ① node.nextSibling和node.previousSibling

返回當前元素的下一個和上一個同級節點（所有類型的節點），找不到則返回null。

#### ② node.nextElementSibling和node.previousSibling

返回當前元素的下一個和上一個同級元素節點，找不到則返回null。

有兼容性問題，IE9以上才支持。

#### ③ 封裝函數

要取得同級元素，又沒有兼容性問題，只能用node.nextSibling和node.previousSibling封裝函數，如下面的代碼。實際開發中用得很少，了解即可。

    <script>

      function getNextElementSibling(*element*) {

        var el = element;

        while ((el = el.nextSibling)) {

          if (el.nodeType === 1) {

            return el;

          }

        }

        return null;

      }

    </script>

### （6）創建和添加節點

#### ① 方法

創建節點：document.createElement(‘tagName’);

添加節點：node.appendChild(child); 將一個節點添加到指定父節點的子節點列表末尾，類似於數組的push。node是上級元素，child是待添加的元素。

添加節點：node.insertBefore(child, 指定元素); 將一個節點添加到父節點的指定子節點前面，類似於CSS裡的before偽元素。

  <body>

    <ul id="ul">

      <li>123</li>

    </ul>

    <script>

      var ul = document.getElementById("ul");

      var li = document.createElement("li");

      var lili = document.createElement("li");

      ul.appendChild(li); // 加在123后面

      ul.insertBefore(lili, ul.children[0]); // 加在123前面

    </script>

  </body>

#### ② 案例：發佈留言

  <body>

    <textarea name="" id="ta" cols="30" rows="10"></textarea>

    <button id="tj">添加</button>

    <ul id="ul">

      <li>123</li>

    </ul>

    <script>

      var ta = document.getElementById("ta");

      var tj = document.getElementById("tj");

      var ul = document.getElementById("ul");

      tj.onclick = function () {

        if (ta.value === "") {

          alert("评论不能为空");

          return;

        }

        var li = document.createElement("li");

        li.innerHTML = ta.value;

        // ul.appendChild(li); //在留言末尾添加新留言

        ul.insertBefore(li, ul.children[0]); //在留言前面添加新留言

        ta.value = "";

      };

    </script>

  </body>

### （7）三種動態創建元素區別

#### ① document.write() （了解）

* 直接將內容寫入頁面的內容流。
* 文檔執行完畢後，會導致頁面全部重繪。

下面的代碼中，點擊按鈕後，按鈕和abc都不見了，只有123這個div

  <body>

    <button>点击</button>

    <p>abc</p>

    <div class="inner"></div>

    <div class="create"></div>

<script>

        // window.onload = function() {

        //         document.write('<div>123</div>');

        //     }

      var btn = document.querySelector("button");

      btn.onclick = function () {

        document.write("<div>123</div>");

      };

    </script>

  </body>

#### ② element.innerHTML 和 document.createElement()

innerHTML 是將內容寫入某個DOM節點，不會導致頁面全部重繪。

innerHTML 創建多個元素效率更高（採用數組形式拼接效率非常高，而直接拼接字符串效率非常低，注意這個區別），結構稍微複雜。

createElement() 創建多個元素效率比innerHTML採用數組拼接字符串的形式稍低，但結構更清晰。

        // 2. innerHTML 创建元素

        var inner = document.querySelector('.inner');

        // for (var i = 0; i <= 100; i++) {

        //     inner.innerHTML += '<a href="#">百度</a>'

        // }

        var arr = [];

        for (var i = 0; i <= 100; i++) {

            arr.push('<a href="#">百度</a>');

        }

        inner.innerHTML = arr.join('');

        // 3. document.createElement() 创建元素

        var create = document.querySelector('.create');

        for (var i = 0; i <= 100; i++) {

            var a = document.createElement('a');

            create.appendChild(a);

        }

### （8）刪除節點

#### ① node.removeChild(child)

該方法從DOM中刪除一個子節點，返回被刪除的節點。

  <body>

    <button id="btn">点击删除li</button>

    <ul id="ul">

      <li>1</li>

      <li>2</li>

      <li>3</li>

    </ul>

    <script>

      var ul = document.getElementById("ul");

      var btn = document.getElementById("btn");

      btn.onclick = function () {

        if (ul.children.length === 1) {

          ul.removeChild(ul.children[0]);

          this.disabled = true;

          return;

        }

        ul.removeChild(ul.children[0]);

      };

    </script>

  </body>

#### ② 案例：刪除留言

阻止鏈接跳轉，給a標籤裡的href添加javascript:void(0);或者javascript:;

比如：

    <a href="javascript:void(0);">删除</a>

    <a href="javascript:;">删除</a>

案例代碼：

  <body>

    <textarea name="" id="ta" cols="30" rows="10"></textarea>

    <button id="tj">添加</button>

    <ul id="ul"></ul>

    <script>

      var ta = document.getElementById("ta");

      var tj = document.getElementById("tj");

      var ul = document.getElementById("ul");

      var lis = ul.children;

      // 添加留言

      tj.onclick = function () {

        if (ta.value === "") {

          alert("评论不能为空");

          return;

        }

        var li = document.createElement("li");

        li.innerHTML = ta.value;

        // 给li添加删除按钮

        var del = document.createElement("button");

        del.innerHTML = "删除";

        li.appendChild(del);

        // 或者这样添加删除按钮;

        // 阻止連接跳轉，添加javascript:void(0);或者javascript:;

        // li.innerHTML = ta.value + "<a href='javascript:;'>删除</a>";

        // ul.appendChild(li); //在留言末尾添加新留言

        ul.insertBefore(li, ul.children[0]); //在留言前面添加新留言

        ta.value = "";

        // 删除留言方法1

        // function dele1() {

        //   // 每一次添加留言，都遍历一下所有的留言，以便拿到最新留言数据的所有删除按钮

        //   for (let i = 0; i < lis.length; i++) {

        //     // 如果留言有删除按钮，才绑定点击事件

        //     if (lis[i].children[0]) {

        //       lis[i].children[0].onclick = function () {

        //         ul.removeChild(ul.children[i]);

        //         // 每一次删除留言，都遍历一下所有的留言，以便拿到最新留言数据的所有删除按钮。这样就有许多重复代码，所以写了dele这个函数

        //         dele();

        //       };

        //     }

        //   }

        // }

        // dele1();

        // 刪除留言方法2

        var delbtns = ul.getElementsByTagName("button");

        for (let i = 0; i < delbtns.length; i++) {

          delbtns[i].onclick = function () {

            ul.removeChild(this.parentNode);

          };

        }

      };

    </script>

  </body>

### （9）複製節點

node.cloneNode();

如果括號參數為空或者false，叫“淺拷貝”，只克隆節點本身，不克隆裡面的子節點。

若要連子節點一起拷貝，括號裡的參數為true，這叫“深拷貝”。

### （10）案例：動態創建表格

代碼：

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      table {

        width: 500px;

        margin: 100px auto;

        border-collapse: collapse;

        text-align: center;

      }

      td,

      th {

        border: 1px solid #333;

      }

      thead tr {

        height: 40px;

        background-color: #ccc;

      }

    </style>

  </head>

  <body>

    <table cellspacing="0">

      <thead>

        <tr>

          <th>姓名</th>

          <th>科目</th>

          <th>成绩</th>

          <th>操作</th>

        </tr>

      </thead>

      <tbody></tbody>

    </table>

    <script>

      // 1.先去准备好学生的数据

      var data = [

        {

          name: "胡正蒙",

          subject: "JavaScript",

          score: 100,

        },

        {

          name: "謝登之",

          subject: "JavaScript",

          score: 97,

        },

        {

          name: "莫如士",

          subject: "JavaScript",

          score: 99,

        },

        {

          name: "甘茹",

          subject: "JavaScript",

          score: 98,

        },

        {

          name: "韓璧",

          subject: "JavaScript",

          score: 95,

        },

      ];

      // 2. 往tbody 里面创建行： 有几个人（通过数组的长度）我们就创建几行

      var tbody = document.querySelector("tbody");

      for (var i = 0; i < data.length; i++) {

        // 外面的for循环管行 tr

        // 1. 创建 tr行

        var tr = document.createElement("tr");

        tbody.appendChild(tr);

        // 2. 行里面创建单元格(跟数据有关系的3个单元格) td 单元格的数量取决于每个对象里面的属性个数  for循环遍历对象 data[i]

        for (var k in data[i]) {

          // 里面的for循环管列 td

          // 创建单元格

          var td = document.createElement("td");

          // 把对象里面的属性值 data[i][k] 给 td

          // console.log(data[i][k]);

          // for(var k in obj) {

          //     k 得到的是属性名

          //     obj[k] 得到是属性值

          // }

          td.innerHTML = data[i][k];

          tr.appendChild(td);

        }

        // 3. 创建有删除2个字的单元格

        var td = document.createElement("td");

        td.innerHTML = '<a href="javascript:;">删除 </a>';

        tr.appendChild(td);

      }

      // 4. 删除操作 开始

      var as = document.querySelectorAll("a");

      for (var i = 0; i < as.length; i++) {

        as[i].onclick = function () {

          // 点击a 删除 当前a 所在的行(链接的爸爸的爸爸)  node.removeChild(child)

          tbody.removeChild(this.parentNode.parentNode);

        };

      }

    </script>

  </body>

</html>

## 8 事件進階

### （1）註冊事件（綁定事件）

註冊事件兩種方式：

* 傳統方式
* 方法監聽註冊方式

#### ① 傳統註冊方式

利用on開頭的事件，比如onclick

<button onclick=”alert(‘hi~’)”></button>

btn.onclick = function(){}

特點：唯一性。同一個元素同一個事件只能設置一個處理函數，後面註冊的處理函數將會覆蓋前面註冊的處理函數。

#### ② 方法監聽註冊方式

w3c標準推薦方式

可用addEventListener()

IE9之前不支持上述方法，可以使用attachEvent()代替。（非標準，盡量別在生產環境中使用。不提倡使用，了解即可）

特點：同一個元素同一個事件可以註冊多個監聽器（即監聽處理函數），按註冊順序依次執行。

##### a. eventTarget.addEventListener語法：

eventTarget.addEventListener(type, listener[, useCapture])

參數：

type：事件類型字符串，比如click，mouseover，不要帶on

listener：事件處理函數，事件發生時，調用該函數。

useCapture：可選參數，佈爾值，默認false。學完DOM事件流後，再進一步學習。

下面示例代碼中給同一個按鈕的同一個事件（click）添加了2個監聽器，會按順序2個都執行。

  <body>

    <button>点击</button>

    <script>

      var btn = document.querySelector("button");

      btn.addEventListener("click", function () {

        alert("hi");

      });

      btn.addEventListener("click", function () {

        alert("hi2");

      });

    </script>

  </body>

##### b. eventTarget.attachEvent語法：（了解）

eventTarget.attachEvent(eventNameWithOn, callback)

eventNameWithOn：事件類型字符串，比如onclick，onmouseover

callback：事件處理函數，當目標出發事件時回調函數被調用。

##### c. 兼容性的註冊事件函數（了解）

處理兼容性的原則：先照顧大多數瀏覽器，再處理特殊瀏覽器。



### （2）刪除事件（解綁事件）

#### ① 傳統註冊方式

eventTarget.onclick = null;

#### ② 方法監聽註冊方式

eventTarget.removeEventListener(type, listener [, userCapture]);

eventTarget.detachEvent(eventNameWithOn, callback);

示例代碼：

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <style>

        div {

            width: 100px;

            height: 100px;

            background-color: pink;

        }

    </style>

</head>

<body>

    <div>1</div>

    <div>2</div>

    <div>3</div>

    <script>

        var divs = document.querySelectorAll('div');

        divs[0].onclick = function() {

                alert(11);

                // 1. 传统方式删除事件

                divs[0].onclick = null;

            }

            // 2. removeEventListener 删除事件

        divs[1].addEventListener('click', fn) // 里面的fn 不需要调用加小括号

        function fn() {

            alert(22);

            divs[1].removeEventListener('click', fn);

        }

        // 3. detachEvent

        divs[2].attachEvent('onclick', fn1);

        function fn1() {

            alert(33);

            divs[2].detachEvent('onclick', fn1);

        }

    </script>

</body>

</html>

③ 處理兼容性的封裝函數（了解）



## 9 DOM事件流

### （1）DOM事件流的3個階段

事件流描述的是從頁面中接收事件的順序。

事件發生時會在元素節點之間按照特定的順序傳播，這個傳播過程即DOM事件流。

DOM事件流分為3個階段

1. 捕獲階段

事件捕獲：網景最早提出，由DOM最頂層節點開始，然後逐級向下傳播到最具體的元素接收的過程。

1. 當前目標階段
2. 冒泡階段

事件冒泡：IE最早提出，事件開始時由最具體的元素接收，然後逐級向上傳播到DOM最頂層節點的過程。



類比：向水裡扔一塊石頭，首先下降，類比從最頂層向事件發生的最具體元素（目標點）的捕獲過程。然後產生泡泡，從最低點（最具體元素）開始浮上水面，類比事件冒泡。



注意：

* JS代碼中只能執行捕獲或者冒泡其中的一個階段。
* onclick和attachEvent只能得到冒泡階段。
* 有的事件沒有冒泡：onblue，onfocus，onmouseover，onmouseleave
* addEventListener(type, listener[, useCapture])第三個參數如果是true，表示在事件捕獲階段調用事件處理程序；如果是false（不寫參數默認是false）表示在事件冒泡階段調用事件處理程序。
* 實際開發中很少使用事件捕獲，更關注事件冒泡。

### （2）捕獲示例代碼

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      .father {

        overflow: hidden;

        width: 300px;

        height: 300px;

        margin: 100px auto;

        background-color: pink;

        text-align: center;

      }

      .son {

        width: 200px;

        height: 200px;

        margin: 50px;

        background-color: purple;

        line-height: 200px;

        color: #fff;

      }

    </style>

  </head>

  <body>

    <div class="father">

      <div class="son">son盒子</div>

    </div>

    <script>

      // dom 事件流 三个阶段

      // 1. JS 代码中只能执行捕获或者冒泡其中的一个阶段。

      // 2. onclick 和 attachEvent（ie） 只能得到冒泡阶段。

      // 3. 捕获阶段 如果addEventListener 第三个参数是 true 那么则处于捕获阶段  document -> html -> body -> father -> son

      // 先弹出father再弹出son

      var son = document.querySelector(".son");

      son.addEventListener(

        "click",

        function () {

          alert("son");

        },

        true

      );

      var father = document.querySelector(".father");

      father.addEventListener(

        "click",

        function () {

          alert("father");

        },

        true

      );

    </script>

  </body>

</html>

### （3）冒泡示例代碼

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      .father {

        overflow: hidden;

        width: 300px;

        height: 300px;

        margin: 100px auto;

        background-color: pink;

        text-align: center;

      }

      .son {

        width: 200px;

        height: 200px;

        margin: 50px;

        background-color: purple;

        line-height: 200px;

        color: #fff;

      }

    </style>

  </head>

  <body>

    <div class="father">

      <div class="son">son盒子</div>

    </div>

    <script>

      // 4. 冒泡阶段 如果addEventListener 第三个参数是 false 或者 省略 那么则处于冒泡阶段  son -> father ->body -> html -> document

      // 执行顺序：son -> father -> document

      var son = document.querySelector(".son");

      son.addEventListener(

        "click",

        function () {

          alert("son");

        },

        false

      );

      var father = document.querySelector(".father");

      father.addEventListener(

        "click",

        function () {

          alert("father");

        },

        false

      );

      document.addEventListener("click", function () {

        alert("document");

      });

    </script>

  </body>

</html>

## 10 事件對象

### （1）什麼是事件對象

        div.onclick = function(*event*) {

                console.log(event);

            }

    <script>

      var div = document.querySelector("div");

      div.addEventListener("click", function (*e*) {

        console.log(e);

      });

    </script>

1. event是一個事件對象，寫在監聽函數的小括號裡面，當形參來看。
2. 事件對象只有有了事件才會存在，它是系統自動創建的，不需要我們傳遞參數。
3. 事件對象是事件的一系列相關數據的集合。比如鼠標點擊事件的事件對象包含了鼠標的相關信息，比如鼠標坐標。鍵盤事件的事件對象包含了鍵盤的信息，比如用戶按下了哪個鍵。
4. 它有很多屬性和方法。
5. 事件對象可以由程序員自行命名，不一定叫element，可以叫event，evt，e。
6. 事件對象也有兼容性問題，ie678的事件對象叫window.event。

可以這樣處理兼容性問題：

    <script>

      var div = document.querySelector("div");

      div.onclick = function (*e*) {

        e = e || window.event;

        console.log(e);

      };

    </script>

### （2）事件對象的常見屬性和方法

|  |  |
| --- | --- |
| 事件對象屬性方法 | 說明 |
| e.target | * 返回觸發事件的對象 標準 * e.target和this的區別：e.target返回的是觸發事件的對象（元素），this返回的是綁定事件的對象（元素）。 * 例如：給ul綁定onclick事件，點擊ul的下級元素li。e.target返回的是li，this返回的是ul。 * e.currentTarget和this非常相似，ie678不支持。因為this比e.currentTarget簡短，而且有兼容性問題，因此了解就好，開發時用this。 |
| e.srcElement | 返回觸發事件的對象 非標準 ie6-8使用 |
| e.type | 返回事件的類型，比如click mouseover，不帶on |
| e.returnValue | 阻止默認事件（默認行為）非標準，ie6-8使用 比如不讓鏈接跳轉 |
| e.preventDefault() | 阻止默認事件（默認行為）標準 比如不讓鏈接跳轉 |
| e.cancelBubble | 阻止冒泡，非標準，ie6-8使用 |
| e.stopPropagation() | 阻止冒泡 標準 |

#### ① e.target

處理兼容性（了解）：

div.onclick = function (*e*) {

        e = e || window.event;

        var target = e.target || e.srcElement;

        console.log(target);

      };

#### ② 阻止默認事件

    <script>

      // 2. 阻止默认行为（事件） 让链接不跳转 或者让提交按钮不提交

      var a = document.querySelector("a");

      a.addEventListener("click", function (*e*) {

        e.preventDefault(); //  dom 标准写法

      });

      // 3. 传统的注册方式

      a.onclick = function (*e*) {

        // 普通浏览器 e.preventDefault();  方法，不影响后面的代码执行

        e.preventDefault();

        // 低版本浏览器 ie678  returnValue  属性，不影响后面的代码执行

        e.returnValue;

        // 利用return false 也能阻止默认行为 没有兼容性问题 特点： return 后面的代码不执行了， 而且只限于传统的注册方式

        return false;

        alert(11);

      };

    </script>

#### ③ 阻止冒泡

下面的son，father，document都有點擊事件。阻止了son的冒泡後，點擊son時，father和document就都不會彈出對話框了。

<script>

      // 常见事件对象的属性和方法

      // 阻止冒泡  dom 推荐的标准 stopPropagation()

      var son = document.querySelector(".son");

      son.addEventListener(

        "click",

        function (*e*) {

          alert("son");

          e.stopPropagation(); // stop 停止  Propagation 传播

          e.cancelBubble = true; // 非标准 cancel 取消 bubble 泡泡

        },

        false

      );

      var father = document.querySelector(".father");

      father.addEventListener(

        "click",

        function () {

          alert("father");

        },

        false

      );

      document.addEventListener("click", function () {

        alert("document");

      });

    </script>

兼容性的寫法（了解）



實際開發常用e.stoppropagation();

### （3）事件委託（代理，委派）

事件委託也稱為事件代理，在jQuery裡面稱為事件委派。

事件委託原理：

不給每個子節點（比如li）單獨設置事件監聽器，而是把事件監聽器設置在其父節點（比如ul）上，然後利用冒泡原理影響設置每個子節點。

例子：給ul註冊點擊事件，利用事件對象的target找到當前點擊的li，點擊li，事件會冒泡到ul上，ul有註冊事件，就會觸發事件監聽器（監聽函數）。

示例代碼：

  <body>

    <ul>

      <li>日</li>

      <li>浩浩湯湯</li>

      <li>月</li>

      <li>天朗氣清</li>

      <li>星辰</li>

    </ul>

    <script>

      // 事件委托的核心原理：给父节点添加侦听器， 利用事件冒泡影响每一个子节点

      var ul = document.querySelector("ul");

      ul.addEventListener("click", function (*e*) {

        // alert('清風');

        // e.target 这个可以得到我们点击的对象

        e.target.style.backgroundColor = "pink";

      });

    </script>

  </body>

### （4）常用的鼠標事件

|  |  |
| --- | --- |
| 鼠標事件 | 觸發條件 |
| onclick | 鼠標點擊左鍵 |
| onmouseover | 鼠標經過 |
| onmouseout | 鼠標離開 |
| onfocus | 獲得鼠標焦點 |
| onblur | 失去鼠標焦點 |
| onmousemove | 鼠標移動 |
| onmousedown | 鼠標按下 |
| onmouseup | 鼠標按下後彈起 |

下面兩個事件了解即可。

  <body>

    我是一段不愿意分享的文字

    <script>

      // 1. contextmenu 我们可以禁用右键菜单

      document.addEventListener("contextmenu", function (*e*) {

        e.preventDefault();

      });

      // 2. 禁止选中文字 selectstart

      document.addEventListener("selectstart", function (*e*) {

        e.preventDefault();

      });

    </script>

  </body>

### （5）鼠標事件對象

MouseEvent

|  |  |
| --- | --- |
| 鼠標事件對象 | 說明 |
| e.clientX | 返回鼠標相對於瀏覽器窗口可視區的X坐標 |
| e.clientY | 返回鼠標相對於瀏覽器窗口可視區的Y坐標 |
| e.pageX | 返回鼠標相對於文檔頁面的X坐標 IE9+支持 |
| e.pageY | 返回鼠標相對於文檔頁面的Y坐標 IE9+支持 |
| e.screenX | 返回鼠標相對於電腦屏幕的X坐標 |
| e.screenY | 返回鼠標相對於電腦屏幕的Y坐標 |

    <script>

      // 鼠标事件对象 MouseEvent

      document.addEventListener("click", function (*e*) {

        // 1. client 鼠标在可视区的x和y坐标

        console.log(e.clientX);

        console.log(e.clientY);

        console.log("---------------------");

        // 2. page 鼠标在页面文档的x和y坐标

        console.log(e.pageX);

        console.log(e.pageY);

        console.log("---------------------");

        // 3. screen 鼠标在电脑屏幕的x和y坐标

        console.log(e.screenX);

        console.log(e.screenY);

      });

    </script>

#### 案例：隨鼠標移動的天使

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      img {

        position: absolute;

        top: 2px;

      }

    </style>

  </head>

  <body>

    <img src="images/angel.gif" alt="" />

    <script>

      var pic = document.querySelector("img");

      document.addEventListener("mousemove", function (*e*) {

        // 1. mousemove只要我们鼠标移动1px 就会触发这个事件

        // console.log(1);

        // 2.核心原理： 每次鼠标移动，我们都会获得最新的鼠标坐标， 把这个x和y坐标做为图片的top和left 值就可以移动图片

        var x = e.pageX;

        var y = e.pageY;

        console.log("x坐标是" + x, "y坐标是" + y);

        //3 . 千万記得给left 和top 添加px 单位，減掉的是圖片的長寬的一半

        pic.style.left = x - 50 + "px";

        pic.style.top = y - 40 + "px";

      });

    </script>

  </body>

</html>

### （6）常用的鍵盤事件

|  |  |
| --- | --- |
| 鍵盤事件 | 觸發條件 |
| onkeyup | 某個鍵盤按鍵被鬆開時 |
| onkeydown | 某個鍵盤按鍵被按下時 |
| onkeypress | 某個鍵盤按鍵被按下時 不識別功能鍵（如ctrl，shift，箭頭） |

不論代碼順序如何，如果上述代碼都有，會先執行onkeypress，再執行onkeydown，再執行onkeyup。

keydown和keypress在文本框裡的特點：這兩個事件觸發的時候，文字還沒有落入文本框中。

keyup事件觸發的時候，文字已經落入文本框裡面。

### （7）鍵盤事件對象

|  |  |
| --- | --- |
| 鍵盤事件對象屬性 | 說明 |
| keyCode | 返回該鍵的ASCII值 |

onkeydown和onkeyup不區分字母大小寫，onkeypress區分字母大小寫。

實際開發中，更多使用onkeydown和onkeyup，因為它們識別所有的鍵（包括功能鍵）。

Keypress不識別功能鍵，但是keyCode屬性能區別大小寫，返回不同的ASCII值。

#### ① 案例：按下s，輸入框獲得焦點

  <body>

    <input type="text" />

    <script>

      // 核心思路： 检测用户是否按下了s 键，如果按下s 键，就把光标定位到搜索框里面

      // 使用键盘事件对象里面的keyCode 判断用户按下的是否是s键。不要用keydown，因為會導致輸入框裡每次都會輸入s。

      // 搜索框获得焦点： 使用 js 里面的 focus() 方法

      var search = document.querySelector("input");

      document.addEventListener("keyup", function (*e*) {

        // console.log(e.keyCode);

        if (e.keyCode === 83) {

          search.focus();

        }

      });

    </script>

  </body>

#### ② 案例：在輸入框裡輸入內容，出現文本框同步放大顯示輸入框中的內容

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      #div {

        display: none;

        width: 200px;

        height: 20px;

        line-height: 20px;

        border: 1px solid coral;

        font-size: 20px;

      }

    </style>

  </head>

  <body>

    <input type="text" id="ipt" />

    <div id="div"></div>

    <script>

      var ipt = document.getElementById("ipt");

      var div = document.getElementById("div");

      ipt.addEventListener("focus", function (*e*) {

        if (this.value !== "") {

          div.style.display = "block";

        }

      });

      ipt.addEventListener("keyup", function (*e*) {

        div.style.display = "block";

        div.innerHTML = this.value;

      });

      ipt.addEventListener("blur", function (*e*) {

        div.style.display = "none";

      });

    </script>

  </body>

</html>

## 11 BOM概述

BOM（Browser Object Model）瀏覽器對象模型，提供了獨立於內容而與瀏覽器窗口進行交互的對象，其核心對象是window。

BOM由一系列相關的對象構成，每個對象都提供了許多方法和屬性。

BOM缺乏標準，JavaScript語法的標準化組織是ECMA，DOM的標準化組織是W3C，BOM最初是Nestscape瀏覽器標準的一部分。

|  |  |
| --- | --- |
| DOM | BOM |
| 文檔對象模型 | 瀏覽器對象模型 |
| 把文檔當作一個對象來看待 | 把瀏覽器當作對象看待 |
| DOM的頂級對象是document | BOM的頂級對象是window |
| 操作頁面元素 | 瀏覽器窗口交互的一些對象 |
| W3C標準規範 | 瀏覽器廠商在各自瀏覽器上定義的，兼容性較差 |

BOM比DOM更大，BOM包含DOM。



window對象是瀏覽器的頂級對象，它具有雙重角色。

1. 它是js訪問瀏覽器窗口的一個接口。
2. 它是一個全局對象。定義在全局作用域中的變量、函數都會變成window對象的屬性和方法。調用的時候可以省略window。

聲明變量的時候不要用name作為名字，因為window下面有個屬性：window.name。

    <script>

      var ipt = window.document.querySelector("input");

      var num = 10;

      console.log(window.num);

      function fn() {

        console.log("hi");

      }

      window.fn();

      window.alert("la");

    </script>

## 12 window對象的常見事件

### （1）窗口加載事件

#### ① onload

window.onload = function(){}

或者

window.addEventListener(“load”, function(){});

這是窗口（頁面）加載事件，當文檔內容（包括圖像，腳本文件，CSS文件）完全加載完成時觸發該事件，就調用處理函數。

有了這個window.onload，可以把JavaScript代碼放到HTML頁面中的任何位置。

window.onload這種傳統註冊事件方式只能寫一次，如果有多個，以最後一個window.onload為準。使用addEventListener則沒有這樣的限制。

#### ② DOMContentLoaded

document.addEventListener(‘DOMContentLoaded’, function(){})

DOMContentLoaded事件觸發時，僅當DOM加載完成，不包括CSS樣式表，圖片，flash等。

ie9以上才支持

如果頁面的圖片很多，從用戶訪問到onload觸發需要較長時間，影響用戶體驗，這時用DOMContentLoaded更合適。

### （2）調整窗口大小事件

window.onresize = function(){}

window.addEventListener(“resize”, function(){});

window.onresize窗口大小發生變化時加載事件並調用函數。

用來完成響應式佈局。window.innerWidth是當前屏幕的寬度

## 13 定時器

### （1）setTimeout()

#### ① 關於setTimeout()

window.setTimeout(調用函數, [延遲的毫秒數]);

* 在定時器到期時調用函數。類比：定時炸彈。
* window可以省略不寫。
* 函數可以直接寫函數，也可以寫函數名（函數事先定義好），或者字符串”函數名()”（不推薦）。
* 延遲時間的單位是毫秒。
* 延遲時間可以省略不寫，如果省略，則為默認值0。
* 一個頁面中有時會有很多定時器，所以通常會給定時器加標識符（即起個名字），加以區別。

比如

        var timer1 = setTimeout(callback, 3000);

        var timer2 = setTimeout(callback, 5000);

示例代碼：

      setTimeout(function () {

        console.log("时间到了");

      }, 2000);

或者這樣寫：

      function callback() {

        console.log("时间到了");

      }

      setTimeout(callback, 2000);

        // setTimeout('callback()', 3000); // 不提倡这个写法

#### ② setTimeout()的回調函數

setTimeout()裡面的函數稱為回調函數callback。

普通函數按照代碼順序直接調用。這個函數需要等待時間，時間到了才回頭去調用它，因此稱為回調函數。

其它回調函數例子：

element.onclick=function(){}或者addEventListener(“click”, fn)裡面的函數也是回調函數，因為要等點擊了以後才去調用。

案例：2秒以後自動隱藏div盒子

    <style>

      #div {

        width: 200px;

        height: 200px;

        background-color: coral;

      }

    </style>

  <body>

    <div id="div"></div>

    <script>

      var div = document.querySelector("div");

      function callback() {

        div.style.display = "none";

      }

      setTimeout("callback()", 2000);

    </script>

  </body>

#### ③ 停止setTimeout()定時器

window.clearTimeout(timeoutID)

* window可以省略
* 括號裡面的timeoutID是定時器標識符（即名字）

### （2）setInterval()

#### ① 關於setInterval()

window.setInterval(回調函數, [間隔的毫秒數]);

* setInterval()每間隔一段時間，就去調用一次回調函數。
* window可以省略不寫。
* 函數可以直接寫函數，也可以寫函數名（函數事先定義好），或者字符串”函數名()”（不推薦）。
* 間隔時間的單位是毫秒。
* 間隔時間可以省略不寫，如果省略，則為默認值0。
* 一個頁面中有時會有很多定時器，所以通常會給定時器加標識符（即起個名字），加以區別。
* 因為setInterval會等一秒鐘才執行，開啟定時器之前先執行一次函數，防止第一次刷新頁面有空白。

      countdown();

      setInterval(countdown, 1000);

#### ② 案例：倒數計時器

    <style>

      #div {

        width: 300px;

        background-color: lightblue;

      }

    </style>

  <body>

    <div id="div"></div>

    <script>

      var div = document.querySelector("div");

      function countdown() {

        var nowTime = +new Date();

        var futureTime = +new Date("2022-10-1 8:8:8");

        var timeDiffBySec = (futureTime - nowTime) / 1000;

        var d = parseInt(timeDiffBySec / 60 / 60 / 24);

        h = h < 10 ? "0" + h : h;

        var h = parseInt((timeDiffBySec / 60 / 60) % 24);

        h = h < 10 ? "0" + h : h;

        var m = parseInt((timeDiffBySec / 60) % 60);

        m = m < 10 ? "0" + m : m;

        var s = parseInt(timeDiffBySec % 60);

        s = s < 10 ? "0" + s : s;

        div.innerHTML = "倒計時：" + d + "天" + h + "時" + m + "分" + s + "秒";

      }

      // 因為setInterval會等一秒鐘才執行，開啟定時器之前先執行一次函數，防止第一次刷新頁面有空白。

      countdown();

      setInterval(countdown, 1000);

    </script>

  </body>

#### ③ 停止setInterval()定時器

window.clearInterval(intervalID)

* window可以省略
* 括號裡面的intervalID是定時器標識符（即名字）

#### ④ 案例：停止倒計時

    <style>

      #div {

        width: 300px;

        background-color: lightblue;

      }

    </style>

 <body>

    <button id="btn1">開始倒計時</button>

    <button id="btn2">停止倒計時</button>

    <div id="div"></div>

    <script>

      var div = document.querySelector("div");

      var btn1 = document.querySelector("#btn1");

      var btn2 = document.querySelector("#btn2");

      function countdown() {

        var nowTime = +new Date();

        var futureTime = +new Date("2022-10-1 8:8:8");

        var timeDiffBySec = (futureTime - nowTime) / 1000;

        var d = parseInt(timeDiffBySec / 60 / 60 / 24);

        h = h < 10 ? "0" + h : h;

        var h = parseInt((timeDiffBySec / 60 / 60) % 24);

        h = h < 10 ? "0" + h : h;

        var m = parseInt((timeDiffBySec / 60) % 60);

        m = m < 10 ? "0" + m : m;

        var s = parseInt(timeDiffBySec % 60);

        s = s < 10 ? "0" + s : s;

        div.innerHTML = "倒計時：" + d + "天" + h + "時" + m + "分" + s + "秒";

      }

      // 把定時器設為全局變量，並賦值為null（null的類別為對象，定時器就是個對象）。否則清除定時器時訪問不到。

      var timer = null;

      btn1.onclick = function () {

        // 因為setInterval會等一秒鐘才執行，開啟定時器之前先執行一次函數，防止第一次刷新頁面有空白。

        countdown();

        timer = setInterval(countdown, 1000);

      };

      btn2.onclick = function () {

        clearInterval(timer);

      };

    </script>

#### ⑤ 案例：隔幾秒再次發送短信

button的內容通過innerHTML修改而非value。

  <body>

    <input type="text" />

    <button id="btn1">發送</button>

    <script>

      var input = document.querySelector("input");

      var btn1 = document.querySelector("#btn1");

      var timer = null; //定義計時器

      var n = 3; //定義剩下的秒數

      function countdown() {

        // 當剩餘秒數為0時，停止interval計時，並把按鈕復原，把n重新設置為3

        if (n === 0) {

          clearInterval(timer);

          btn1.disabled = false;

          btn1.innerHTML = "發送";

          n = 3;

        } else {

          btn1.innerHTML = "已發送," + n + "秒後重新發送";

          n--;

        }

      }

      btn1.addEventListener("click", function () {

        // 按下按鈕後，按鈕被禁用，開啟倒計時

        this.disabled = true;

        countdown();

        timer = setInterval(countdown, 1000);

      });

    </script>

  </body>

## 14 this的指向

this的指向在函數定義的時候無法確定，只有函數執行時才能確定this指向誰。

this指向的是調用它的對象。

  <body>

    <button>点击</button>

    <script>

      // this 指向问题 一般情况下this的最终指向的是那个调用它的对象

      // 1. 全局作用域或者普通函数中this指向全局对象window（ 注意定时器里面的this指向window）

      console.log(this);

      function fn() {

        console.log(this);

      }

      window.fn();

      window.setTimeout(function () {

        console.log(this);

      }, 1000);

      // 2. 方法调用中谁调用this指向谁

      var o = {

        sayHi: function () {

          console.log(this); // this指向的是 o 这个对象

        },

      };

      o.sayHi();

      var btn = document.querySelector("button");

      // btn.onclick = function() {

      //     console.log(this); // this指向的是btn这个按钮对象

      // }

      btn.addEventListener("click", function () {

        console.log(this); // this指向的是btn这个按钮对象

      });

      // 3. 构造函数中this指向构造函数的实例

      function Fun() {

        console.log(this); // this 指向的是fun 实例对象

      }

      var fun = new Fun();

    </script>

  </body>

## 15 JS執行機制

### （1）同步和異步的來由

JavaScript語言的一大特點是單線程，即一次做一件事。因為JavaScript的使命：處理頁面中用戶的交互，操作DOM。比如對某個DOM元素進行添加和刪除操作，不能同時進行，應該先添加再刪除。

單線程意味著所有任務需要排隊：前一個任務結束，才會執行後一個任務。

導致問題：JS執行的時間過長，導致頁面渲染不連貫（頁面渲染阻塞）。

解決機制：同步和異步。

利用多核CPU的計算能力，HTML5提出Web Worker標準，允許JavaScript腳本創建多個線程。於是，JS中出現了同步和異步。

① **同步**：前一個任務執行結束，再執行後一個任務。依次執行任務。

② **異步**：如果一個任務執行時間很長，在執行這句任務時，可以同時執行其它任務。類比：燒水（10分鐘）的時候洗菜。

例子：下面log(3)的時間比較長，在等待這個任務的時候，先log(2)。

下面的打印結果是1 2 3

      console.log(1);

      setTimeout(function () {

        console.log(3);

      }, 1000);

      console.log(2);

### （2）同步任務和異步任務執行過程

下面的打印順序是1,2,3

      console.log(1);

      setTimeout(function () {

        console.log(3);

      }, 0);

      console.log(2);

**主線執行任務（執行棧）（同步任務）：**

conso.log(1);

setTimeout(fn, 0);

conso.log(2);

**任務列隊（消息列隊）（異步任務）：**

function() {

conso.log(3);

}

JS的異步是通過回調函數實現的。

異步任務有下列三種類型：

1. 普通事件：click，resize等
2. 資源加載：load，error等
3. 定時器：setInterval，setTimeout等

**JS執行機制：**

1. 先執行執行棧中的同步任務
2. 把異步任務（回調函數）放入任務隊列中。
3. 一旦執行棧中所有同步任務執行完畢，系統依次讀取任務隊列中的異步任務，這些異步任務結束等待狀態，進入執行棧，開始執行。

**如果有多個異步任務**

        console.log(1);

        document.onclick = function() {

            console.log('click');

        }

        console.log(2);

        setTimeout(function() {

            console.log(3)

        }, 3000)

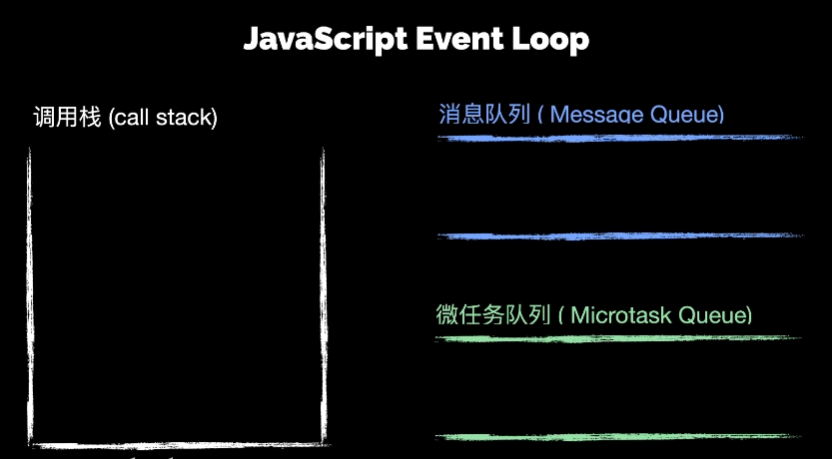
異步進程處理決定是否把異步任務放入任務隊列中。

如果沒有點擊，任務隊列中就不會有點擊任務。如果倒計時還沒有結束，任務隊列中也不會有console.log(3)的任務。

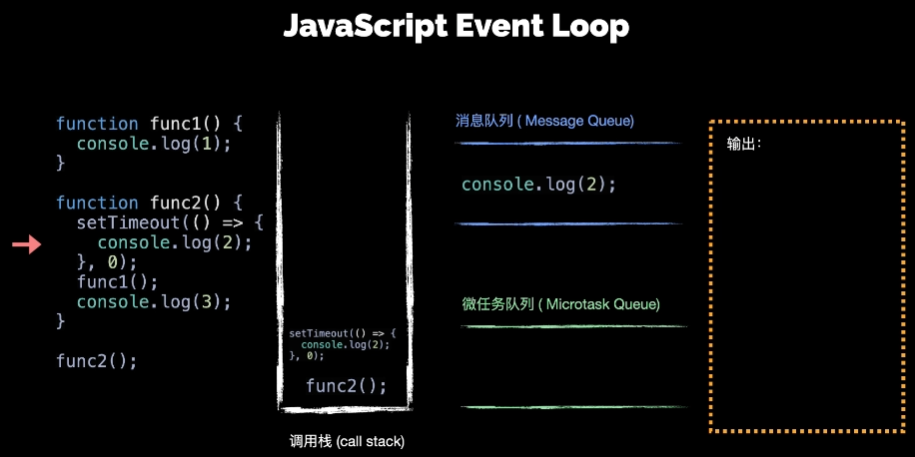


主線程執行完畢，異步進程處理會繼續檢查看看是否有新的異步程序產生，有的話就放入任務隊列，然後依次推送進入主線程去執行。這個過程就是事件循環。

### （3）JavaScript event loop 詳解



fetch，事件回調，setTimeout，setInterval中的回調會被放入消息隊列中。消息隊列中的代碼會在調用棧清空之後執行。



Promise，async，await創建的異步操作會加入微任務隊列，在調用棧被清空的時候立即執行。



最後輸出：



上面的講解來源：

<https://www.bilibili.com/video/BV1kf4y1U7Ln?from=search&seid=10931669757122508578&spm_id_from=333.337.0.0>

The event Loop keeps running continuously and checks the Main stack, if it has any frames to execute, if not then it checks Callback queue, if Callback queue has codes to execute then it pops the message from it to the Main Stack for the execution.

## 16 location對象

### （1）什麼是location對象

window對象提供了一個location屬性，用於獲取、設置解析窗口的URL。

因為這個屬性返回的是一個對象，所以將其稱為location對象。

### （2）URL

URL（Uniform Recourse Locator，統一資源定位符），是互聯網上標準資源的地址。互聯網上的每個文件都有一個唯一的URL，它包含的信息指出文件的位置以及瀏覽器應該怎麼處理它。

URL語法格式：

protocol://host[:port]/path/[?query]#fragment

例子：

<http://www.itcast.cn/index.html?name=andy&age=18#link>

|  |  |
| --- | --- |
| 組成 | 說明 |
| protocol | 通信協議，常用的有http，ftp，maito |
| host | 主機（域名）[www.itheima.com](http://www.itheima.com) |
| port | 端口號，可選，省略是使用默認端口，如http的默認端口為80 |
| path | 路徑，有零或多個’/’符號隔開的字符串，一般用來表示主機上的一個目錄或文件地址 |
| query | 參數，以鍵值對的形式通過&符號分隔開來 |
| fragment | 片段，#後面的內容，常見於鏈接，錨點 |

### （3）location對象的屬性

|  |  |
| --- | --- |
| location對象屬性 | 返回值 |
| location.href（重點） | 獲取或者設置整個URL |
| location.host | 返回主機（域名） |
| location.port | 返回端口號；如果沒寫，返回空字符串 |
| location.pathname | 返回路徑 |
| location.search（重點） | 返回參數，上面表格中的參數（query） |
| location.hash | 返回URL的#後面的內容，常見於連接，錨點 |

案例：進入頁面，2秒後自動跳轉到淘寶頁面

    <script>

      setTimeout(function () {

        location.href = "http://www.taobao.com";

      }, 2000);

    </script>

### （4）location對象的方法

|  |  |
| --- | --- |
| location對象的方法 | 作用 |
| location.assign() | 跳轉頁面（也稱重定向頁面）。括號裡的參數是個字符串，為目標網頁地址。  記錄了瀏覽歷史，可以後退頁面。 |
| location.replace() | 替換當前頁面。因為不記錄歷史，所以不能後退頁面。 |
| location.reload() | 重新加載頁面。相當於鼠標點擊瀏覽器的刷新按鈕，或者鍵盤按下f5。這樣刷新後，加載頁面速度會提高，因為許多數據緩存到了本地，是從本地讀取數據。  如果參數為true，為強制刷新=鍵盤按下ctrl+f5。強制刷新會清空網頁的本地緩存，直接從服務器獲取數據。 |

    <script>

      var btn = document.querySelector("button");

      btn.addEventListener("click", function () {

        location.assign("https://www.bing.com");

      });

    </script>

## 17 navigator對象

navigator對象包含有關瀏覽器的信息，它有很多屬性，最常用的是userAgent，該屬性可以返回由客戶機發送服務器的user-agent頭部的值。

下面前端代碼可以判斷用戶用什麼終點打開頁面，然後進行跳轉

    <script>

      if (

        navigator.userAgent.match(

          /(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mobile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS|Symbian|Windows Phone)/i

        )

      ) {

        window.location.href = ""; //手機

      } else {

        window.location.href = ""; //電腦

      }

    </script>

## 18 history對象

window對象提供了一個叫history的對象，與瀏覽器歷史進行交互。

history對象包含用戶在瀏覽器窗口中訪問過的URL。

|  |  |
| --- | --- |
| history對象方法 | 作用 |
| back() | 後退 |
| forward() | 前進 |
| go(參數) | 前進和後退。  如果參數=1，前進1個頁面。  如果參數=-1，後退1個頁面。 |

history對象在實際開發中比較少用；如果開發OA辦公系統，會用到。

## 19 元素偏移量offset系列

### （1）offset常見屬性

使用offset相關屬性可以動態地得到該元素的位置（偏移）、大小等。

* 獲得元素距離**帶有定位父元素**（父元素一定要設置了position。如果各個父級都沒有定位，則返回body）的位置
* 獲得元素自身的大小（寬高）
* 返回值為數字，不帶單位

|  |  |
| --- | --- |
| offset系列屬性 | 作用 |
| element.offsetParent | 返回該元素的父級元素（帶有定位position的。如果各個父級都沒有定位，則返回body） |
| element.offsetTop | 返回元素相對於 帶有定位的父元素 的上邊框的偏移 |
| element.offsetLeft | 返回元素相對於 帶有定位的父元素 的左邊框的偏移 |
| element.offsetWidth | 返回自身包括padding、邊框、內容區的寬度 |
| element.offsetHeight | 返回自身包括padding、邊框、內容區的高度 |

示例代碼：

    <style>

      \* {

        margin: 0;

        padding: 0;

      }

      .father {

        /\* position: relative; \*/

        width: 200px;

        height: 200px;

        background-color: pink;

        margin: 150px;

      }

      .son {

        width: 100px;

        height: 100px;

        background-color: purple;

        margin-left: 45px;

      }

      .w {

        height: 200px;

        background-color: skyblue;

        margin: 0 auto 200px;

        padding: 10px;

        border: 15px solid red;

      }

    </style>

  <body>

    <div class="father">

      <div class="son"></div>

    </div>

    <div class="w"></div>

    <script>

      // offset 系列

      var father = document.querySelector(".father");

      var son = document.querySelector(".son");

      // 1.可以得到元素的偏移 位置 返回的不带单位的数值

      console.log(father.offsetTop);

      console.log(father.offsetLeft);

      // 它以带有定位的父亲为准  如果么有父亲或者父亲没有定位 则以 body 为准

      console.log(son.offsetLeft);

      var w = document.querySelector(".w");

      // 2.可以得到元素的大小 宽度和高度 是包含padding + border + width

      console.log(w.offsetWidth);

      console.log(w.offsetHeight);

      // 3. 返回带有定位的父亲 否则返回的是body

      console.log(son.offsetParent); // 返回带有定位的父亲 否则返回的是body

      console.log(son.parentNode); // 返回父亲 是最近一级的父亲 亲爸爸 不管父亲有没有定位

    </script>

  </body>

### （2）offset和style的區別

|  |  |
| --- | --- |
| offset | style |
| 可得到任意樣式表中的樣式值 | 只能得到行內樣式表（和HTML代碼寫在一起）中的樣式值。 |
| 返回的是數字，沒有單位 | 返回的是字符串，數字+單位 |
| padding+border+width | 只有width |
| 只讀屬性，不能賦值。想要獲取值，用offset | 可讀寫屬性，可以賦值。想要改值，用style |

### （3）案例：點擊盒子，獲得鼠標距離盒子上邊框和左邊框的距離

    <script>

      var box = document.querySelector("div");

      // 拿到盒子距離頁面頂部和左邊的距離

      var boxT = box.offsetTop;

      var boxL = box.offsetLeft;

      var mToBW;

      var mToBH;

      box.addEventListener("click", function (*e*) {

        // 拿到鼠標距離頁面頂部和左邊的距離，並減去盒子距離頂部和左邊的距離，得到鼠標距離盒子頂部和左邊的距離。

        mToBW = e.pageX - boxL + "px";

        mToBH = e.pageY - boxT + "px";

        console.log(mToBW);

        console.log(mToBH);

      });

    </script>

### （4）案例：點擊按鈕，彈出模態框（這裡是登錄框）和遮罩層，能關閉模態框和遮罩層，能拖動模態框



<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Document</title>

    <style>

      #gray-layer {

        display: none;

        position: fixed;

        top: 0;

        left: 0;

        width: 100%;

        height: 100%;

        background-color: #678f8d;

        opacity: 0.8;

        z-index: 1;

      }

      #login {

        position: absolute;

        display: none;

        width: 512px;

        height: 280px;

        top: 200px;

        left: 50%;

        transform: translateX(-50%);

        background-color: #ffd000;

        z-index: 2;

      }

      .login-title {

        display: block;

        width: 100%;

        height: 100px;

        line-height: 100px;

        text-align: center;

        background-color: #ff6670;

        cursor: move;

      }

      #closebtn {

        position: absolute;

        top: 20px;

        right: 20px;

        display: block;

        width: 20px;

        height: 20px;

        background-color: #77a88d;

        border-radius: 50%;

      }

      #closebtn:hover {

        cursor: pointer;

      }

    </style>

  </head>

  <body>

    <button>點擊彈出登錄框</button>

    <!-- 遮罩層 -->

    <div id="gray-layer"></div>

    <!-- 登錄框 -->

    <div id="login">

      <span class="login-title" id="login-title">登錄框</span>

      <span id="closebtn"></span>

    </div>

    <script>

      var btn = document.querySelector("button");

      var grayLayer = document.querySelector("#gray-layer");

      var login = document.querySelector("#login");

      var closebtn = document.querySelector("#closebtn");

      var loginTitle = document.querySelector("#login-title");

      // 點擊按鈕彈出遮罩層和登錄框

      btn.addEventListener("click", function () {

        grayLayer.style.display = "block";

        login.style.display = "block";

      });

      // 點擊關閉按鈕關閉遮罩層和登錄框

      closebtn.addEventListener("click", function () {

        grayLayer.style.display = "none";

        login.style.display = "none";

      });

      // 點擊登錄框就能拖動登錄框

      loginTitle.addEventListener("mousedown", function (*e*) {

        // 每一次的拖拽，鼠標相對於登錄框的距離都是不變的

        // 鼠標到登錄框Top

        mouseLoginTop = e.pageY - login.offsetTop;

        // 鼠標到登錄框Left

        mouseLoginLeft = e.pageX - login.offsetLeft;

        // 鼠標移動時把新的top和left值賦值給登錄框

        loginTitle.addEventListener("mousemove", move);

        function move(*e*) {

          // 登錄框top = 鼠標e.pagY - 鼠標到登錄框Top

          loginTop = e.pageY - mouseLoginTop;

          // 登錄框left = 鼠標e.pagX - 鼠標到登錄框Left

          loginLeft = e.pageX - mouseLoginLeft;

          login.style.top = loginTop + "px";

          login.style.left = loginLeft + "px";

        }

        // 鼠標按鍵抬起時取消mousemove的監聽

        loginTitle.addEventListener("mouseup", function () {

          loginTitle.removeEventListener("mousemove", move);

        });

      });

    </script>

  </body>

</html>

### （5）案例：放大鏡效果

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      #img-wrap {

        position: relative;

        width: 200px;

        height: 270px;

        margin-top: 10px;

        margin-left: 300px;

      }

      #img-sm {

        width: 100%;

        height: 100%;

      }

      #mask {

        display: none;

        position: absolute;

        top: 0;

        left: 0;

        width: 70px;

        height: 70px;

        background-color: gold;

        opacity: 0.3;

        cursor: move;

      }

      #img-bg-wrap {

        display: none;

        position: absolute;

        top: 0;

        left: 230px;

        width: 295px;

        height: 400px;

        border: 1px solid #ccc;

        overflow: hidden;

      }

      #img-bg {

        position: absolute;

      }

    </style>

  </head>

  <body>

    <!-- 圖片本身寬651px高880px -->

    <!-- 1 佈局 -->

    <div id="img-wrap">

      <img id="img-sm" src="./h1.jpg" alt="h1" />

      <div id="mask"></div>

      <div id="img-bg-wrap">

        <img id="img-bg" src="./h1.jpg" alt="h1" />

      </div>

    </div>

    <script>

      // 2 把鼠標放在img-wrap上，遮罩層和大圖顯示；鼠標離開img-wrap，遮罩層和大圖隱藏

      var imgWrap = document.querySelector("#img-wrap");

      var mask = document.querySelector("#mask");

      var imgBgWrap = document.querySelector("#img-bg-wrap");

      var imgBg = document.querySelector("#img-bg");

      imgWrap.addEventListener("mouseenter", function () {

        mask.style.display = "block";

        imgBgWrap.style.display = "block";

      });

      imgWrap.addEventListener("mouseleave", function () {

        mask.style.display = "none";

        imgBgWrap.style.display = "none";

      });

      // 3 mask跟著鼠標移動

      imgWrap.addEventListener("mousemove", function (*e*) {

        // 3-1 先取得移入imgWrap那一瞬間，鼠標相對於imgWrap的top和left值

        var mouseImgWrapLeft = e.pageX - imgWrap.offsetLeft;

        var mouseImgWrapTop = e.pageY - imgWrap.offsetTop;

        // 3-2 鼠標位於mask中間

        var maskX = mouseImgWrapLeft - mask.offsetWidth / 2;

        var maskY = mouseImgWrapTop - mask.offsetHeight / 2;

        // 3-3 mask不能跑到imgWrap外面

        // mask在x轴上的最大距离

        var maxXMax = imgWrap.offsetWidth - mask.offsetWidth;

        // mask在y轴上的最大距离

        var maxYMax = imgWrap.offsetHeight - mask.offsetHeight;

        maskX = maskX <= 0 ? 0 : maskX;

        maskX = maskX >= maxXMax ? maxXMax : maskX;

        maskY = maskY <= 0 ? 0 : maskY;

        maskY = maskY >= maxYMax ? maxYMax : maskY;

        // 3-4 鼠標開始移動，把新位置賦值給mask

        mask.style.left = maskX + "px";

        mask.style.top = maskY + "px";

        // 4 让imgBig在imgBigWrap里面动

        // 公式maskX / maxXMax = imgBigX / imgBigXMax

        // igmBigMax在x轴上的最大距离

        var imgBigXMax = imgBg.offsetWidth - imgBgWrap.offsetWidth;

        // igmBigMax在y轴上的最大距离

        var imgBigYMax = imgBg.offsetHeight - imgBgWrap.offsetHeight;

        var imgBigX = maskX \* (imgBigXMax / maxXMax);

        var imgBigY = maskY \* (imgBigYMax / maxYMax);

        imgBg.style.left = -imgBigX + "px";

        imgBg.style.top = -imgBigY + "px";

      });

    </script>

  </body>

</html>

## 20 元素可視區client系列

### （1）關於client系列

動態地獲取元素可視區的相關信息。比如某元素的邊框大小，元素大小等。

|  |  |
| --- | --- |
| client系列屬性 | 作用 |
| element.clientTop | 返回元素上邊框的大小 |
| element.clientLeft | 返回元素左邊框的大小 |
| element.clientWidth | 返回自身包括padding、內容區的寬度，不含邊框，返回數值不帶單位 |
| element.clientHeight | 返回自身包括padding、內容區的高度，不含邊框，返回數值不帶單位 |

clientWidth和offsetWidth最大的區別就是前者不包含邊框

### （2）案例：淘寶flexible.js源碼分析

#### ① 立即執行函數

立即執行函數：不需要調用，能立即自動執行

寫法①

最後的小括號可以看作調用函數，可以傳遞參數

      (function () {})();

例子：

    <script>

      (function (*a*) {

        console.log(a);

      })(2);

      (function (*b*, *c*) {

        console.log(b + c);

      })(3, 5);

    </script>

給函數起名字也可以：

      (function sum(*e*, *f*) {

        console.log(e + f);

      })(6, 7);

寫法②

      (function(){}());

例子：

(function (e, f) {

console.log(e + f);

}(6, 7)); //打印結果13

還可以寫成有名字的函數：

(function sum(e, f) {

console.log(e + f);

}(6, 7));

立即執行函數作用：創建一個獨立的作用域，這樣裡面的變量都是局部變量，避免了命名衝突。

#### ② pagesshow事件和load事件

下面三種情況都會刷新頁面都會觸發load事件。

1. a標籤的超鏈接
2. F5或者瀏覽器的刷新按鈕（強制刷新）
3. 瀏覽器的前進後退按鈕
   1. 例外。火狐有“往返緩存”，保持了頁面數據，包含DOM和JavaScript的狀態，實際上將整個頁面都保存在了內存裡。所以後退按鈕不能刷新頁面。
      1. 這種情況下，可以用pageshow事件來觸發。這個事件在頁面顯示時觸發，無論頁面是否來自緩存。重新加載頁面時，pageshow會在load事件觸發後觸發；根據事件對象中的persisted來判斷是否是緩存中的頁面觸發的pageshow事件。
      2. pageshow事件要添加給window。

#### ③ 淘寶flexible.js源碼分析

(function flexible(*window*, *document*) {

    // 获取的html 的根元素

    var docEl = document.documentElement

    // dpr 物理像素比。如果能拿到，就用这个物理像素比；如果没有，就用1。

    var dpr = window.devicePixelRatio || 1

    // adjust body font size  设置我们body 的字体大小

    function setBodyFontSize() {

        // 如果页面中有body 这个元素 就设置body的字体大小

        if (document.body) {

            document.body.style.fontSize = (12 \* dpr) + 'px'

        } else {

            // 如果页面中没有body 这个元素（因为有时候程序员会在HTML页面的head中引入这个js文件，这时候页面的body还没有被加载），则等着页面主要的DOM元素加载完毕再去设置body的字体大小

            document.addEventListener('DOMContentLoaded', setBodyFontSize)

        }

    }

    setBodyFontSize();

    // set 1rem = viewWidth / 10    设置我们html 元素的文字大小

    function setRemUnit() {

        var rem = docEl.clientWidth / 10

        docEl.style.fontSize = rem + 'px'

    }

    setRemUnit()

    // reset rem unit on page resize  当我们页面尺寸大小发生变化的时候，要重新设置下rem 的大小

    window.addEventListener('resize', setRemUnit)

    // pageshow 是我们重新加载页面触发的事件

    window.addEventListener('pageshow', function (*e*) {

        // e.persisted 返回的是true 就是说如果这个页面是从缓存取过来的页面，也需要从新计算一下rem 的大小

        if (e.persisted) {

            setRemUnit()

        }

    })

    // detect 0.5px supports  有些移动端的浏览器不支持0.5像素的写法

    if (dpr >= 2) {

        var fakeBody = document.createElement('body')

        var testElement = document.createElement('div')

        testElement.style.border = '.5px solid transparent'

        fakeBody.appendChild(testElement)

        docEl.appendChild(fakeBody)

        if (testElement.offsetHeight === 1) {

            docEl.classList.add('hairlines')

        }

        docEl.removeChild(fakeBody)

    }

}(window, document))

## 21 元素滾動scroll系列

### （1）scroll相關屬性

scroll相關屬性可以動態得到該元素的大小、滾動距離等。

|  |  |
| --- | --- |
| scroll系列屬性 | 作用（都是返回數值不帶單位） |
| element.scrollTop（更常用） | 返回被卷去的上側距離 |
| element.scrollLeft（更常用） | 返回被卷去的左側距離 |
| element.scrollWidth | 返回內容的寬度，含padding，不含邊框 |
| element.scrollHeight | 返回內容的高度，含padding，不含邊框 |

比如內容很多，盒子不足以顯示整個內容，就會出現滾動條。滾動條向下滾動時，頁面上部被隱藏的高度，成為scrollTop（頁面被卷去的頭部）。

滾動條滾動時觸發onscroll事件。

下面的圖類似於把滾動條拉到了底部。



### （2）案例：頁面向上滾動，改變側邊欄的position屬性，並決定是否顯示“返回頂部”

頁面被卷去的頭部，可以通過window.pageYOffset獲得。被卷去的左側通過window.pageXOffset獲得。

注意區別：

元素被卷去的頭部是element.scrollTop，頁面被卷去的頭部則是window.pageYOffset。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      .w {

        width: 1200px;

        margin: 10px auto;

      }

      .header {

        height: 150px;

        background-color: #678f8d;

      }

      .banner {

        height: 250px;

        background-color: #678f8d;

      }

      .main {

        height: 1000px;

        background-color: #678f8d;

      }

      .side-bar {

        position: absolute;

        left: 50%;

        top: 300px;

        margin-left: 600px;

        width: 45px;

        height: 130px;

        background-color: #ff6670;

      }

      span {

        display: none;

        position: absolute;

        bottom: 0;

      }

    </style>

  </head>

  <body>

    <div class="header w">头部区域</div>

    <div class="banner w">banner区域</div>

    <div class="main w">主体部分</div>

    <div class="side-bar">

      <span class="go-top">返回顶部</span>

    </div>

    <script>

      // 获取元素

      var sideBar = document.querySelector(".side-bar");

      var goTop = document.querySelector(".go-top");

      var banner = document.querySelector(".banner");

      var main = document.querySelector(".main");

      // 側邊框被設為fixed定位的臨界值就是banner上邊框和屏幕上邊框貼合的時候，也就是banner上邊框距離body上邊框的距離

      var bannerTop = banner.offsetTop;

      // 側邊框被改為fixed定位的臨界點，然後定在這裡，側邊欄上邊框距離屏幕上邊框=側邊欄上邊框到body/頁面上邊框的距離-banner上邊框到body/頁面上邊框的距離

      var sideBarFixedY = sideBar.offsetTop - bannerTop;

      // 顯示返回頂部的臨界點，main的上邊框與屏幕上邊框重合

      var mainTop = main.offsetTop;

      // 頁面滾動，所以給監聽document

      document.addEventListener("scroll", function () {

        // 當頁面的banner的上邊框與屏幕上邊框重合的時候，把sideBar改為fix定位，重新設置top值；否則改回原樣。

        if (window.pageYOffset >= bannerTop) {

          sideBar.style.position = "fixed";

          sideBar.style.top = sideBarFixedY + "px";

        } else {

          sideBar.style.position = "absolute";

          sideBar.style.top = "300px";

        }

        // 當頁面的main的上邊框與屏幕上邊框重合的時候，顯示返回頂部；否則隱藏返回頂部

        if (window.pageYOffset >= mainTop) {

          goTop.style.display = "block";

        } else {

          goTop.style.display = "none";

        }

      });

    </script>

  </body>

</html>

### （3）頁面被卷去頭部的兼容性解決方案

注意兼容性問題，計算頁面被卷去的頭部有如下寫法：

① 聲明了DTD，使用document.documentElement.scrollTop

聲明了DTD指HTML頁面中有這句話：

<!DOCTYPE html>

② 未聲明DTD，使用document.body.scrollTop

③ 重點：新方法window.pageYOffset和window.pageXOffset，IE9開始支持

要解決兼容性，可以封裝一個函數（了解）：

    <script>

      function getScroll() {

        return {

          left:

            window.pageXOffset ||

            document.documentElement.scrollLeft ||

            document.body.scrollLeft ||

            0,

          top:

            window.pageYOffset ||

            document.documentElement.scrollTop ||

            document.body.scrollTop ||

            0,

        };

      }

      // 使用：

      getScroll().left;

    </script>

## 22 offset，client，scroll總結

element.offsetWidth 返回值帶邊框，其它兩個不帶邊框。

當內容很多，出現滾動條時，element.clientHeight值返回可視區的高度，而element.scrollHeight返回整個內容的高度。

offset系列常用於獲得元素位置：offsetLeft，offsetTop

client系列常用於獲取元素大小：clientWidth，clientHeight

scroll系列常永不獲取滾動距離：scrollTop，scrollLeft

頁面滾動距離通過window.pageXOffset獲得

## 23 動畫函數封裝

### （1）動畫原理

通過定時器setInterval()不斷移動盒子位置。

實現步驟：

1. 獲得盒子當前位置
2. 讓盒子在當前位置加1個移動距離（比如1px，10px）
3. 利用定時器重複上述2個步驟
4. 加一個結束定時器的條件
5. 要給元素添加定位，才能使用element.style.left

示例代碼：

打開頁面，盒子自動往右走，走到offsetLeft≥400的地方停下

    <style>

      div {

        position: absolute;

        left: 0;

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

    </style>

  <body>

    <div></div>

    <script>

      var box = document.querySelector("div");

      var timer = setInterval(function () {

        if (box.offsetLeft >= 100) {

          clearInterval(timer);

        }

        box.style.left = box.offsetLeft + 1 + "px";

      }, 10);

    </script>

  </body>

### （2）動畫函數簡單封裝

#### ① 初步封裝

這個函數需要傳遞2個參數：動畫對象和目標位置。

把上面的代碼封裝成一個函數，可以同時讓多個目標盒子調用：

    <style>

      div {

        position: absolute;

        left: 0;

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

      span {

        position: absolute;

        display: block;

        left: 0;

        top: 200px;

        width: 100px;

        height: 100px;

        background-color: aqua;

      }

    </style>

  <body>

    <div></div>

    <span></span>

    <script>

      function animate(*obj*, *target*) {

        var timer = setInterval(function () {

          if (obj.offsetLeft >= target) {

            clearInterval(timer);

          }

          obj.style.left = obj.offsetLeft + 1 + "px";

        }, 10);

      }

      var box = document.querySelector("div");

      var span = document.querySelector("span");

      animate(box, 100);

      animate(span, 200);

    </script>

  </body>

#### ② 修改封裝

給不同元素以不同定時器

**上述代碼的問題：**

① 每次調用函數animate時，var timer都會去內存開闢一個新空間，這樣很浪費性能。

② 每個item的定時器都叫timer，容易引起歧義。

解決方法：

給不同元素以不同定時器。

原理：JS是動態語言，可以方便地給當前對象添加屬性。

例子：

var obj = {};

obj.name = ‘andy’;

上面的函數改成這樣：

把var timer 改成 obj.timer，下面clearInterval(timer);改為clearInterval(obj.timer);

      function animate(*obj*, *target*) {

        obj.timer = setInterval(function () {

          if (obj.offsetLeft >= target) {

            clearInterval(obj.timer);

          }

          obj.style.left = obj.offsetLeft + 1 + "px";

        }, 10);

      }

**上述代碼還有個問題：**

如果這個函數放在click監聽函數裡，那麼每次點擊，都會開啟新的定時器，這樣導致盒子越走越快，也不會停在目標位置。

解決方法：每次開啟定時器前，先關閉一下定時器。clearInterval(obj.timer);

把函數變成這樣：

      function animate(*obj*, *target*) {

        clearInterval(obj.timer); //加上這句話

        obj.timer = setInterval(function () {

          if (obj.offsetLeft >= target) {

            clearInterval(obj.timer);

            return; // 防止達到目標位置後再移動

          }

          obj.style.left = obj.offsetLeft + 1 + "px";

        }, 10);

      }

### （3）緩動動畫

#### ① 單一方向

歡動動畫讓元素運動速度有變化，最常見的是讓速度變慢，讓元素慢慢停下。

思路：

1. 讓盒子每次移動的距離慢慢變小，速度就會慢慢變小。
2. 核心算法：（目標值-現在的值）/ 10 作為每次移動的步長。
   * 其中10可以變成其它的數字
   * 要對步長step取整，否則盒子到不了目標位置，總會差一點。

下面是沒有對step取整的結果。第一個盒子沒有到達目標500px的位置，第二個盒子也沒有到達目標200px的位置。取整step以後可以避免這種情況。



1. 停止定時器條件：盒子位置=目標位置

    <style>

      div {

        position: absolute;

        left: 0;

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

      span {

        position: absolute;

        display: block;

        left: 0;

        top: 200px;

        width: 100px;

        height: 100px;

        background-color: aqua;

      }

    </style>

  <body>

    <button>點擊</button>

    <div></div>

    <span></span>

    <script>

      function animate(*obj*, *target*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。

          var step = Math.ceil((target - obj.offsetLeft) / 10);

          if (obj.offsetLeft === target) {

            clearInterval(obj.timer);

            return;

          }

          obj.style.left = obj.offsetLeft + step + "px";

        }, 15);

      }

      var box = document.querySelector("div");

      var span = document.querySelector("span");

      var button = document.querySelector("button");

      button.addEventListener("click", function () {

        animate(box, 500);

      });

      animate(span, 200);

    </script>

  </body>

#### ② 可以前進可以退後

要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

    <style>

      div {

        position: absolute;

        left: 0;

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

    </style>

  <body>

    <button id="btn1">點擊到500</button>

    <button id="btn2">點擊到800</button>

    <div></div>

    <script>

      function animate(*obj*, *target*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

          var step = (target - obj.offsetLeft) / 10;

          step = step > 0 ? Math.ceil(step) : Math.floor(step);

          if (obj.offsetLeft === target) {

            clearInterval(obj.timer);

            return;

          }

          obj.style.left = obj.offsetLeft + step + "px";

        }, 15);

      }

      var box = document.querySelector("div");

      var btn1 = document.querySelector("#btn1");

      var btn2 = document.querySelector("#btn2");

      btn1.addEventListener("click", function () {

        animate(box, 500);

      });

      btn2.addEventListener("click", function () {

        animate(box, 800);

      });

    </script>

  </body>

### （4）動畫函數添加回調函數

回調函數原理：函數可以作為一個參數。將這個函數作為參數傳到另一個函數裡面，當那個函數執行完之後，再執行傳進去的這個函數，這個過程就叫做回調。

例子，點擊按鈕，盒子移動後變色：

    <style>

      div {

        position: absolute;

        left: 0;

        width: 100px;

        height: 100px;

        background-color: yellow;

      }

    </style>

  <body>

    <button id="btn1">點擊移動變色</button>

    <div></div>

    <script>

      function animate(*obj*, *target*, *callback*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

          var step = (target - obj.offsetLeft) / 10;

          step = step > 0 ? Math.ceil(step) : Math.floor(step);

          if (obj.offsetLeft === target) {

            clearInterval(obj.timer);

// 回調函數寫在定時器結束裡面

            if (callback) {

              callback();

            }

            return;

          }

          obj.style.left = obj.offsetLeft + step + "px";

        }, 15);

      }

      var box = document.querySelector("div");

      var btn1 = document.querySelector("#btn1");

      btn1.addEventListener("click", function () {

        animate(box, 500, function () {

          box.style.backgroundColor = "#ff6670";

        });

      });

    </script>

  </body>

### （5）案例：鼠標進入，盒子出來



鼠標hover盒子，紅色盒子從黃色盒子底下緩緩出來，動畫結束的時候，改變箭頭方向。



鼠標離開盒子，紅色盒子收起，，動畫結束的時候，改變箭頭方向。

代碼：

    <style>

      body,

      div {

        margin: 0;

        padding: 0;

      }

      #wrap {

        position: fixed;

      }

      #wrap:hover {

        cursor: pointer;

      }

      #tab {

        position: absolute;

        width: 50px;

        height: 50px;

        line-height: 50px;

        text-align: center;

        background-color: yellow;

        z-index: 2;

      }

      #strip {

        position: absolute;

        left: -100px;

        width: 150px;

        height: 50px;

        background-color: #ff6670;

        z-index: 1;

      }

    </style>

  <body>

    <div id="wrap">

      <div id="tab">→</div>

      <div id="strip"></div>

    </div>

    <script>

      function animate(*obj*, *target*, *callback*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

          var step = (target - obj.offsetLeft) / 10;

          step = step > 0 ? Math.ceil(step) : Math.floor(step);

          if (obj.offsetLeft === target) {

            clearInterval(obj.timer);

            // 回調函數寫在定時器結束裡面

            if (callback) {

              callback();

            }

            return;

          }

          obj.style.left = obj.offsetLeft + step + "px";

        }, 15);

      }

      var wrap = document.querySelector("#wrap");

      var strip = document.querySelector("#strip");

      var tab = document.querySelector("#tab");

      wrap.addEventListener("mouseover", function () {

        animate(strip, 50, function () {

          tab.innerHTML = "←";

        });

      });

      wrap.addEventListener("mouseout", function () {

        animate(strip, -100, function () {

          tab.innerHTML = "→";

        });

      });

    </script>

  </body>

## 24 常見網頁特效案例

### （1）案例：輪播圖

#### ① 輪播圖常見需求：

1. 鼠標經過輪播圖時，顯示左右按鈕，停止自動播放輪播圖
2. 點擊左右按鈕，左右播放圖片
3. 點擊左右按鈕，底部相應按鈕被選中
4. 點擊底部按鈕，切換到相應圖片
5. 鼠標離開輪播圖，左右按鈕消失，自動播放輪播圖，底部相應按鈕被選中

#### ② 關於圖片：

1. 給li左浮動後，li肯定會掉下來，因為輪播圖最外面的盒子寬度有限，所以要給ul設置足夠的寬度。如果是5張圖片，給以給ul的寬設置為500%。
2. 一定要記得給圖片ul定位設置position，否則無法通過animate讓其移動
3. 是移動ul，不是移動li

#### ③ 關於底部按鈕：

1. 要根據圖片張數，動態生成下部按鈕
2. 點擊編號為i的底部按鈕時，ul向左移動 i\*圖片寬度 的距離

#### ④ 輪播圖製作過程：

1. 搭建結構
2. 鼠標經過顯示隱藏左右按鈕
3. 動態生成底部按鈕
4. 點擊選中底部按鈕
5. 點擊底部按鈕顯示相應圖片
6. 點擊右側按鈕滾動圖片
   * 把第一張圖片用JS克隆一份，放在ul最後。
   * 當圖片滾動到克隆的圖片時，讓ul瞬間跳到第1張圖片
7. 點擊右側按鈕時，底部按鈕的選中狀態一起變化
8. 點擊左側按鈕滾動圖片
9. 自動播放
   * 類似於點擊了右側按鈕，使用手動調用右側按鈕事件 btnr.click()
   * 鼠標經過時，clearInterval停止調用btnr.click()。
   * 鼠標離開時，開啟定時器。
10. 節流閥
    * 防止連續點擊左右鍵時，播放過快
    * 原理：上一個函數動畫執行完畢，再去執行下一個函數動畫。
    * 思路：利用回調函數，添加一個變量來控制，鎖住函數和解鎖函數。
      + var flag = true;
      + 設置在左右按鈕點擊事件函數裡，包裹所有代碼。if(flag){flag = false; do something}
      + 利用animate的回調函數，動畫執行完畢時flag=true

#### ⑤ 代碼：

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      body,

      div,

      ul,

      li {

        margin: 0;

        padding: 0;

      }

      ul {

        list-style-type: none;

      }

      #carousel {

        position: relative;

        width: 870px;

        height: 500px;

        margin: 30px auto;

      }

      #carousel span {

        position: absolute;

        display: none;

        width: 50px;

        height: 50px;

        top: 50%;

        transform: translateY(-50%);

        border-radius: 50%;

        background-color: #ff6670;

      }

      #carousel span:hover {

        cursor: pointer;

      }

      #btnl {

        left: -50px;

        z-index: 2;

      }

      #btnr {

        right: -50px;

        z-index: 2;

      }

      .pic-wrap {

        width: 100%;

        height: 100%;

        overflow: hidden;

      }

      #pics {

        position: relative;

        width: 500%;

      }

      #pics li {

        float: left;

      }

      #pics img {

        width: 870px;

        height: 500px;

      }

      #dots {

        position: absolute;

        bottom: -30px;

        left: 50%;

        transform: translateX(-50%);

      }

      #dots li {

        float: left;

        width: 20px;

        height: 20px;

        margin-right: 10px;

        border-radius: 50%;

        background-color: #678f8d;

        cursor: pointer;

      }

      #dots .selected {

        background-color: #ff6670;

      }

    </style>

  </head>

  <body>

    <div id="carousel">

      <span id="btnl"></span>

      <span id="btnr"></span>

      <div class="pic-wrap">

        <ul id="pics">

          <li>

            <a href="#"><img src="./pic1.jpeg" alt="" /></a>

          </li>

          <li>

            <a href="#"><img src="./pic2.jpg" alt="" /></a>

          </li>

          <li>

            <a href="#"><img src="./pic3.jpg" alt="" /></a>

          </li>

          <li>

            <a href="#"><img src="./pic4.jpg" alt="" /></a>

          </li>

        </ul>

      </div>

      <ul id="dots"></ul>

    </div>

    <script>

      // 1 動畫函數

      function animate(*obj*, *target*, *callback*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

          var step = (target - obj.offsetLeft) / 10;

          step = step > 0 ? Math.ceil(step) : Math.floor(step);

          if (obj.offsetLeft === target) {

            clearInterval(obj.timer);

            // 回調函數寫在定時器結束裡面

            // if (callback) {

            //   callback();

            // }

            // 上面三行代碼相當於下面這一行，這叫短路運算。如果左邊為true才會執行右邊的代碼；如果左邊為false，就不再執行右邊的代碼了。

            callback && callback();

            return;

          }

          obj.style.left = obj.offsetLeft + step + "px";

        }, 15);

      }

      var carousel = document.querySelector("#carousel");

      var btnl = document.querySelector("#btnl");

      var btnr = document.querySelector("#btnr");

      var pics = document.querySelector("#pics");

      var dots = document.querySelector("#dots");

      // 單張圖片的寬度，一定要記得給圖片定位，否則無法通過animate讓其移動

      var picW = pics.querySelector("li").clientWidth;

      // 記錄被選中的下部按鈕

      var dotI = 0;

      // 記錄當前是第幾張圖片

      var picI = 0;

      // 2 鼠標經過，顯示隱藏左右按鈕

      // 11 鼠標經過，停止/開始自動播放

      carousel.addEventListener("mouseover", function () {

        btnl.style.display = "block";

        btnr.style.display = "block";

        clearInterval(timer);

        timer = null;

      });

      carousel.addEventListener("mouseout", function () {

        btnl.style.display = "none";

        btnr.style.display = "none";

        timer = setInterval(function () {

          // 手動調用點擊事件

          btnr.click();

        }, 2000);

      });

      // 3 根據圖片的張數，動態生成底部按鈕

      for (let i = 0; i < pics.children.length; i++) {

        var li = document.createElement("li");

        if (i === 0) {

          li.className = "selected";

        }

        dots.appendChild(li);

        // 4 點擊底部按鈕，選中相應的按鈕

        // 生成li的時候，直接綁定點擊事件

        li.addEventListener("click", function () {

          // 把所有按鈕變成非選中狀態

          for (let i = 0; i < dots.children.length; i++) {

            dots.children[i].className = "";

          }

          //選中點擊的按鈕

          this.className = "selected";

          // 記錄被選中的是第幾個按鈕，同時告訴左右button的按鈕記錄當前是第幾張圖片被選中

          dotI = i;

          picI = i;

          //5 切換到相應的圖片

          animate(pics, -picW \* i);

        });

      }

      // 6 點擊右側按鈕，每點擊一次，ul向左移動一張圖片的距離

      // 7 把第一張圖片克隆一下，然後放到pics UL的最後（這樣克隆出來的li，不會導致下部按鈕多出一個來）(因為這個克隆寫作了生成下部按鈕的後面。根據圖片生成下部按鈕的時候，還沒有這個克隆，所以不會多出下部按鈕)

      var pic1 = pics.children[0].cloneNode(true);

      pics.appendChild(pic1);

      // 12 （共12步）節流閥。把監聽點擊左右按鈕裡面所有的代碼寫到節流閥裡。一開始flag=true，監聽開始後flag=false，動畫結束時（通過animate的回調函數確定動畫結束，因為這個函數在動畫結束時執行）flag=false。

      var flag = true; //設置節流閥

      btnr.addEventListener("click", function () {

        if (flag) {

          flag = false; //關閉節流閥

          if (picI === pics.children.length - 1) {

            picI = 0;

            pics.style.left = 0;

          }

          picI++;

          animate(pics, -picW \* picI, function () {

            flag = true; //打開節流閥

          });

          // 8 點擊右側按鈕時，底部按鈕的選中狀態一起變化

          // 把底部所有按鈕變為非選中狀態

          dotI++;

          for (let i = 0; i < dots.children.length; i++) {

            dots.children[i].className = "";

          }

          // 選中相應的底部按鈕

          if (dotI < dots.children.length) {

            dots.children[dotI].className = "selected";

          } else if ((dotI = dots.children.length)) {

            //如果當前到了克隆圖片（即ul的最後一張）

            dots.children[0].className = "selected";

            dotI = 0;

          }

        }

      });

      // 9 點擊左側按鈕，移動圖片

      btnl.addEventListener("click", function () {

        if (flag) {

          flag = false;

          // 如果當前是首圖，先快速跳轉到ul末圖，即克隆圖片，然後再進行動畫

          if (picI === 0) {

            picI = pics.children.length - 1;

            dotI = picI;

            pics.style.left = -picW \* picI + "px";

          }

          picI--;

          animate(pics, -picW \* picI, function () {

            flag = true;

          });

          // 10 選中下部相應按鈕

          // 把底部所有按鈕變為非選中狀態

          dotI--;

          for (let i = 0; i < dots.children.length; i++) {

            dots.children[i].className = "";

          }

          // 選中相應的底部按鈕

            dots.children[dotI].className = "selected";

        }

      });

      // 11 鼠標經過，開始或者停止自動播放

      var timer = setInterval(function () {

        // 手動調用點擊事件

        btnr.click();

      }, 2000);

    </script>

  </body>

</html>

### （2）案例：帶有動畫的返回頂部

滾動窗口至文檔中的特定位置

window.scroll(x,y)

x和y不跟單位，比如window.scroll(0,100)

可以繼續使用封裝的動畫函數。只需要把所有的left相關的值改為跟頁面垂直滾動距離相關就可以了。

頁面滾動了多少，可以通過window.pageYOffset得到

代碼：

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      .w {

        width: 1200px;

        margin: 10px auto;

      }

      .header {

        height: 150px;

        background-color: #678f8d;

      }

      .banner {

        height: 250px;

        background-color: #678f8d;

      }

      .main {

        height: 1000px;

        background-color: #678f8d;

      }

      .side-bar {

        position: absolute;

        left: 50%;

        top: 300px;

        margin-left: 600px;

        width: 45px;

        height: 130px;

        background-color: #ff6670;

      }

      .go-top {

        display: none;

        position: absolute;

        bottom: 0;

      }

      .go-top:hover {

        cursor: pointer;

      }

    </style>

  </head>

  <body>

    <div class="header w">头部区域</div>

    <div class="banner w">banner区域</div>

    <div class="main w">主体部分</div>

    <div class="side-bar">

      <span class="go-top">返回顶部</span>

    </div>

    <script>

      // 获取元素

      var sideBar = document.querySelector(".side-bar");

      var goTop = document.querySelector(".go-top");

      var banner = document.querySelector(".banner");

      var main = document.querySelector(".main");

      // 側邊框被設為fixed定位的臨界值就是banner上邊框和屏幕上邊框貼合的時候，也就是banner上邊框距離body上邊框的距離

      var bannerTop = banner.offsetTop;

      // 側邊框被改為fixed定位的臨界點，然後定在這裡，側邊欄上邊框距離屏幕上邊框=側邊欄上邊框到body/頁面上邊框的距離-banner上邊框到body/頁面上邊框的距離

      var sideBarFixedY = sideBar.offsetTop - bannerTop;

      // 顯示返回頂部的臨界點，main的上邊框與屏幕上邊框重合

      var mainTop = main.offsetTop;

      // 頁面滾動，所以給監聽document

      document.addEventListener("scroll", function () {

        // 當頁面的banner的上邊框與屏幕上邊框重合的時候，把sideBar改為fix定位，重新設置top值；否則改回原樣。

        if (window.pageYOffset >= bannerTop) {

          sideBar.style.position = "fixed";

          sideBar.style.top = sideBarFixedY + "px";

        } else {

          sideBar.style.position = "absolute";

          sideBar.style.top = "300px";

        }

        // 當頁面的main的上邊框與屏幕上邊框重合的時候，顯示返回頂部；否則隱藏返回頂部

        if (window.pageYOffset >= mainTop) {

          goTop.style.display = "block";

        } else {

          goTop.style.display = "none";

        }

      });

      // 引入並修改之前寫好的緩動動畫函數，把裡面所有和左右相關的值，改為和上下相關的值。

      function animate(*obj*, *target*, *callback*) {

        clearInterval(obj.timer);

        obj.timer = setInterval(function () {

          // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

          var step = (target - window.pageYOffset) / 10;

          step = step > 0 ? Math.ceil(step) : Math.floor(step);

          if (window.pageYOffset === target) {

            clearInterval(obj.timer);

            // 回調函數寫在定時器結束裡面

            // if (callback) {

            //   callback();

            // }

            // 上面三行代碼相當於下面這一行，這叫短路運算。如果左邊為true才會執行右邊的代碼；如果左邊為false，就不再執行右邊的代碼了。

            callback && callback();

            return;

          }

          window.scroll(0, window.pageYOffset + step);

        }, 15);

      }

      // 點擊“返回頂部”，則返回到頁面頂部

      goTop.addEventListener("click", function () {

        // 因為是窗口的動畫，所以下面的第一個參數是window

        animate(window, 0);

      });

    </script>

  </body>

</html>

### （3）案例：筋斗雲

#### ① 需求

1. 鼠標經過某個li，筋斗雲跟到這個li的位置
2. 鼠標離開這個li，筋斗雲位置復原
3. 鼠標點擊了某個li，筋斗雲停留在li的位置

#### ② 分析

1. 利用動畫函數做動畫效果
2. 筋斗雲的起始位置是0
3. 鼠標經過li，把當前li的offsetLeft位置作為目標值即可
4. 鼠標離開li，目標值回復為0

#### ③ 代码

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      div,

      ul,

      li {

        padding: 0;

        margin: 0;

        list-style-type: none;

      }

      div {

        position: relative;

        height: 42px;

        margin-left: 50px;

      }

      ul {

        position: absolute;

        z-index: 2;

      }

      li {

        float: left;

        width: 83px;

        height: 42px;

        line-height: 36px;

        text-align: center;

        margin-right: 20px;

        color: #ccc;

      }

      li:hover {

        cursor: pointer;

      }

      span {

        position: absolute;

        left: 0;

        width: 83px;

        height: 42px;

        background: url(*./cloud.gif*) no-repeat;

      }

    </style>

  </head>

  <body>

    <div>

      <ul>

        <li>搗練子</li>

        <li>憶江南</li>

        <li>憶王孫</li>

        <li>如夢令</li>

        <li>漁家傲</li>

      </ul>

      <span></span>

    </div>

  </body>

  <script>

    // 引入緩動動畫函數

    function animate(*obj*, *target*, *callback*) {

      clearInterval(obj.timer);

      obj.timer = setInterval(function () {

        // 要對step取整，否則盒子到不了目標位置，總會差一點。要看step是正值還是賦值，如果是正值，就向上取整；如果是負值，就向下取整。

        var step = (target - obj.offsetLeft) / 10;

        step = step > 0 ? Math.ceil(step) : Math.floor(step);

        if (obj.offsetLeft === target) {

          clearInterval(obj.timer);

          // 回調函數寫在定時器結束裡面

          // if (callback) {

          //   callback();

          // }

          // 上面三行代碼相當於下面這一行，這叫短路運算。如果左邊為true才會執行右邊的代碼；如果左邊為false，就不再執行右邊的代碼了。

          callback && callback();

          return;

        }

        obj.style.left = obj.offsetLeft + step + "px";

      }, 15);

    }

    // 獲取元素

    var ul = document.querySelector("ul");

    var cloud = document.querySelector("span");

    var cloudBeginX = 0; //雲的起始位置

    for (let i = 0; i < ul.children.length; i++) {

      ul.children[i].addEventListener("mouseenter", function () {

        // 把雲移動到鼠標位置

        animate(cloud, this.offsetLeft);

      });

      ul.children[i].addEventListener("mouseleave", function () {

        // 把雲移回初始位置

        animate(cloud, cloudBeginX);

      });

      ul.children[i].addEventListener("click", function () {

        cloudBeginX = this.offsetLeft;

      });

    }

  </script>

</html>

## 25 觸屏事件

### （1）觸屏事件概述

移動端瀏覽器兼容性好，不需要考慮JS的兼容性問題。

移動端和Android、IOS有其特有屬性，比如觸屏事件touch。

touch對象代表一個觸摸點。觸摸點可能由手指或者觸摸筆產生。觸屏事件相應用戶手指/觸控筆對屏幕或者觸控板操作。

|  |  |
| --- | --- |
| 觸屏touch事件 | 說明 |
| touchstart | 手指/筆觸摸到一個DOM元素時觸發 |
| touchmove | 手指在一個DOM元素上滑動時觸發 |
| touchend | 手指從一個DOM元素上移開時觸發 |

  <body>

    <div>盒子</div>

    <script>

      var div = document.querySelector("div");

      div.addEventListener("touchstart", function () {

        console.log("九天攬月");

      });

    </script>

  </body>

### （2）觸屏事件對象

TouchEvent是一類描述手指在觸摸平面（觸摸屏、觸摸板等）的狀態變化的事件。

touchstart、touchmove、touchend三個事件各自有其事件對象。

  <body>

    <div>盒子</div>

    <script>

      var div = document.querySelector("div");

      div.addEventListener("touchstart", function (*e*) {

        console.log(e); //TouchEvent

        console.log(e.targetTouches[0]); //得到正在觸摸dom元素的第一個手指的關鍵信息，比如手指的坐標等

      });

    </script>

  </body>

TouchEvent觸摸事件重點對象列表

|  |  |
| --- | --- |
| 觸摸列表 | 說明 |
| touches | 正在觸摸屏幕的所有手指的列表，手指離開屏幕後沒有該列表。 |
| **targetTouches（最常用）** | 正在觸摸當前DOM元素上的手指的列表。手指離開屏幕後沒有該列表。 |
| changedTouches | 手指狀態發生了改變（從無到有，從有到無）的列表。 |

### （3）移動端拖動元素

touchstart、touchmove、touchend都可以拖動元素

拖動元素需要知道手指當前的坐標值，可以使用targetTouches[0]裡面的pageX和pageY。

移動端拖動的原理：手指移動中，計算出手指移動的距離。然後用盒子原來的位置+手指移動的距離。

手指移動的距離：手指滑動後的位置-手指剛開始觸摸的位置。

拖動元素三部曲：

1. 觸摸元素touchstart：獲取手指初始坐標和盒子原來的位置
2. 移動手指touchmove：計算手指的滑動距離，並且移動盒子
3. 離開手指touchend。

注意：

手指移動也會觸發滾動屏幕，所以要阻止默認的屏幕滾動e.preventDefault();

要給盒子設置position，否則無法拖動。

代碼：

    <style>

      div {

        position: relative;

        width: 50px;

        height: 50px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <div>盒子</div>

    <script>

      var div = document.querySelector("div");

      // 獲取手指初始坐標

      var startX = 0;

      var startY = 0;

      // 獲取盒子原來的位置

      var x = 0;

      var y = 0;

      div.addEventListener("touchstart", function (*e*) {

        // 1 觸摸元素 touchstart：獲取手指初始坐標，同時獲得盒子原來的位置

        startX = e.targetTouches[0].pageX;

        startY = e.targetTouches[0].pageY;

        x = this.offsetLeft;

        y = this.offsetTop;

      });

      div.addEventListener("touchmove", function (*e*) {

        // 2 計算手指的移動距離：手指移動之後的坐標減去手指初始的坐標

        var moveX = e.targetTouches[0].pageX - startX;

        var moveY = e.targetTouches[0].pageY - startY;

        // 3 移動盒子：盒子原來的位置+手指移動的距離

        this.style.left = x + moveX + "px";

        this.style.top = y + moveY + "px";

        // 4 阻止屏幕滾動的默認行為

        e.preventDefault();

      });

    </script>

  </body>

## 26 移動端常見特效

### （1）案例：移動端輪播圖

移動端輪播圖功能和PC端基本一致：

* 可以自動播放圖片
  + 開啟定時器
  + 使用translate移動
* 手指可以拖動播放輪播圖

#### ① 步驟

1. 搭建框架的時候最外面的div寬度100%；ul寬度500%（3張圖片+2張克隆），確保一排裝下所有li；li的寬度為ul的1/5即20%；img的寬度為100%，和父元素li一樣。
2. 記得給ul清除浮動
3. 在第首圖前面放一張末圖克隆；在末圖後面放一張首圖的克隆。
4. 設置ul的margin-lef: -100%，否則默認看見的是末圖克隆
5. 利用定時器自動輪播圖片
   * 用transform來移動圖片
   * 用過度效果來慢慢地游動圖片
   * 實現無縫輪播
     + 判斷條件時要等到圖片滾動完畢再去判斷，即過度完成後再判斷
     + 通過添加檢測過度完成事件transitioned即可
   * 下部按鈕選中相應按鈕
     + 把按鈕li中帶有selected的類名選出來去掉類名remove
     + 讓當前索引號所在的li加上selected add
     + 得等到過度結束之後，所以這個寫在transitionend事件裡面
     + 給下部按鈕的li的css部分加上transition: all 0.3s;這樣可以下部按鈕也有圖片那樣的動畫效果。也可以不加。
6. 手指拖動輪播圖
   * 本質：pics的ul跟隨手指移動，即移動端的手指拖動元素
   * 觸摸元素touchstart: 獲取手指初始坐標
   * 移動手指touchmove: 計算手指的滑動距離，並且移動盒子
   * 離開手指touchend：根據滑動的距離分不同的情況
     + 如果移動距離小於某個像素，就彈回原來位置
     + 如果移動距離大於某個像素，就向上一張/下一張滑動。

#### ② classList

classList屬性是HTML5新增的一個屬性，返回元素的類名。ie10以上版本支持。該屬性用於在元素中添加、移除、切換CSS類。

  <body>

    <div class="box1"></div>

    <span class="box2 box3"></span>

  </body>

  <script>

    var div = document.querySelector("div");

    var span = document.querySelector("span");

    console.log(div.classList);

    console.log(span.classList);

  </script>



**classList常見屬性**

a) 添加類

element.classList.add(‘類名’)

* 添加類名是在後面追加類名，不會覆蓋以前的類名
* 類名前面不要加 .

b) 移除類

* 移除其中的一個類名

element.classList.remove(‘類名’)

c) 切換類

element.classList.toggle(‘類名’);

示例代碼：

    <style>

      div {

        width: 100px;

        height: 100px;

      }

      .bgc {

        background-color: yellow;

      }

    </style>

  <body>

    <button>點擊切換div顏色</button>

    <div class="box1 box5"></div>

    <script>

      var div = document.querySelector("div");

      var button = document.querySelector("button");

      div.classList.add("box2"); //添加類名

      div.classList.remove("box1"); //刪除類名

      button.addEventListener("click", function () {

        div.classList.toggle("bgc"); //點擊button，給div加上bgc這個類，再點擊button，給div去掉bgc這個類

      });

    </script>

  </body>

#### ③ 代码

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>2-2 移動端輪播圖</title>

    <style>

      body,

      div,

      ul,

      li {

        margin: 0;

        padding: 0;

      }

      ul {

        list-style-type: none;

      }

      #carousel {

        position: relative;

        overflow: hidden;

      }

      #pics {

        /\* overflow: hidden; \*/

        width: 500%;

        margin-left: -100%;

      }

      #pics li {

        float: left;

        width: 20%;

      }

      #pics img {

        width: 100%;

      }

      #dots {

        position: absolute;

        bottom: 30px;

        left: 50%;

        transform: translateX(-50%);

      }

      #dots li {

        float: left;

        width: 20px;

        height: 20px;

        margin-right: 10px;

        border-radius: 50%;

        background-color: #678f8d;

        cursor: pointer;

      }

      #dots .selected {

        width: 40px;

        background-color: #ff6670;

        border-radius: 30%;

      }

      .test {

        width: 100px;

        height: 100px;

        background-color: aqua;

      }

    </style>

  </head>

  <body>

    <div id="carousel">

      <ul id="pics">

        <li>

          <a href="#"><img src="./images/2-2 picm1.jpg" alt="" /></a>

        </li>

        <li>

          <a href="#"><img src="./images/2-2 picm2.jpg" alt="" /></a>

        </li>

        <li>

          <a href="#"><img src="./images/2-2 picm3.jpg" alt="" /></a>

        </li>

      </ul>

      <ul id="dots"></ul>

    </div>

    <div class="test"></div>

    <script>

      var pics = document.querySelector("#pics");

      var dots = document.querySelector("#dots");

      // 圖片的寬度

      var w = pics.children[0].offsetWidth;

      // 目前所顯示的圖片索引號

      var index = 0;

      // 8-1 用於判斷用戶是否移動了手指,只有手指移動了，才把flag設為true

      flag = false;

      // 1 自動生成下部按鈕

      for (let i = 0; i < pics.children.length; i++) {

        var li = document.createElement("li");

        // 給第1個li加上選中的類

        if (i === 0) {

          li.className = "selected";

        }

        dots.appendChild(li);

      }

      // 2 克隆首圖和末圖，分別放在ul的末尾和開頭

      var picFirst = pics.children[0].cloneNode(true);

      var picLast = pics.children[pics.children.length - 1].cloneNode(true);

      pics.appendChild(picFirst);

      pics.insertBefore(picLast, pics.children[0]);

      // 3 利用定時器輪播圖片

      var timer = setInterval(function () {

        index++;

        var movePicsX = -index \* w;

        pics.style.transition = "all .3s";

        pics.style.transform = "translateX(" + movePicsX + "px)";

      }, 2000);

      // 3-2等過度完成後，再去判斷監聽過度完成的事件 transitionend

      pics.addEventListener("transitionend", function () {

        // 3-2-1如果圖片滾動到了ul末尾的克隆，則快速移動到ul的首圖，此時要去掉動畫效果

        if (index >= dots.children.length) {

          index = 0;

          pics.style.transition = "none";

          var movePicsX = -index \* w;

          pics.style.transform = "translateX(" + movePicsX + "px)";

        } //3-2-2 如果倒著走

        else if (index <= -1) {

          index = dots.children.length - 1;

          pics.style.transition = "none";

          var movePicsX = -index \* w;

          pics.style.transform = "translateX(" + movePicsX + "px)";

        }

        // 4 底部相應按鈕被選中

        // 4-1 把按鈕li中帶有selected的類名選出來去掉類名remove

        dots.querySelector(".selected").classList.remove("selected");

        // 4-2 讓當前索引號所在的li加上selected  add

        dots.children[index].classList.add("selected");

        // 4-3 得等到過度結束之後，所以這個寫在transitionend事件裡面

      });

      // 5 手指滑動輪播圖

      // 5-1 觸摸元素touchstart: 獲取手指初始坐標

      var startX = 0;

      // 設定移動距離

      var moveX = 0;

      pics.addEventListener("touchstart", function (*e*) {

        startX = e.targetTouches[0].pageX;

        // 只要開始拖動，就停止定時器

        clearInterval(timer);

        e.preventDefault;

      });

      // 5-2 移動手指touchmove: 計算手指的滑動距離，並且移動盒子

      pics.addEventListener("touchmove", function (*e*) {

        // 計算手指移動距離

        moveX = e.targetTouches[0].pageX - startX;

        // 移動盒子：盒子後來的位置=盒子原來的位置+手指移動的距離

        var movePicsX = -index \* w + moveX;

        //手指拖動的時候，不需要動畫效果，所以要取消過度效果

        pics.style.transition = "none";

        pics.style.transform = "translateX(" + movePicsX + "px)";

        flag = true; //8-2 只有手指移動了，才把flag設為true

        e.preventDefault(); // 取消默認行為，比如滾動屏幕

      });

      // 6 手指離開，根據移動距離判斷回彈還是播放上一張/下一張

      pics.addEventListener("touchend", function (*e*) {

        // 8-3 只有手指移動了，才把flag設為true，才去計算相應的移動距離

        if (flag) {

          // 6-1 如果移動距離大於50px，播放上一張/下一張

          if (Math.abs(moveX) > 50) {

            // moveX>0，為右滑，播放上一張

            if (moveX > 0) {

              index--;

            } else {

              // moveX<0，為左滑，播放下一張

              index++;

            }

            // 用最新的index值獲得最新的pics的位置

            var movePicsX = -index \* w;

            pics.style.transition = "all .3s";

            pics.style.transform = "translateX(" + movePicsX + "px)";

          } else {

            // 6-2 如果移動距離小於50px，就回彈

            var movePicsX = -index \* w;

            pics.style.transition = "all .1s";

            pics.style.transform = "translateX(" + movePicsX + "px)";

          }

        }

        // 7 手指離開屏幕就重新開啟定時器

        clearInterval(timer);

        timer = setInterval(function () {

          index++;

          var movePicsX = -index \* w;

          pics.style.transition = "all .3s";

          pics.style.transform = "translateX(" + movePicsX + "px)";

        }, 2000);

      });

    </script>

  </body>

</html>

### （2）案例：移動端返回頂部

#### ① 需求

當頁面滾動到某個地方，顯示 返回頂部按鈕，否則隱藏

點擊可以返回頂部

#### ② 代碼邏輯

1. 滾動到某個地方顯示按鈕
   * 事件：scroll頁面滾動事件
   * 如果被卷去的頭部（window.pageYOffset）大於某個數值，就顯示返回頂部按鈕
   * 某個數值為主體部分（main）的上邊框貼合屏幕上邊框時，即main.offsetTop
2. 點擊window.scroll(0,0)返回頂部

#### ③ 代碼

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <style>

      body,

      div,

      span {

        margin: 0;

        padding: 0;

      }

      .w {

        width: 100%;

        margin: 10px auto;

      }

      .header {

        height: 150px;

        background-color: #678f8d;

      }

      .banner {

        height: 250px;

        background-color: #678f8d;

      }

      .main {

        height: 1000px;

        background-color: #678f8d;

      }

      .go-top {

        display: none;

        position: fixed;

        right: 0;

        bottom: 100px;

        width: 25px;

        height: 100px;

        background-color: #ff6670;

      }

      .go-top:hover {

        cursor: pointer;

      }

    </style>

  </head>

  <body>

    <div class="header w">头部区域</div>

    <div class="banner w">banner区域</div>

    <div class="main w">主体部分</div>

    <div class="go-top">返回頂部</div>

    <script>

      // 获取元素

      var goTop = document.querySelector(".go-top");

      var main = document.querySelector(".main");

      var mainTop = main.offsetTop;

      window.addEventListener("scroll", function () {

        // 如果被卷去的頭部（window.pageYOffset）大於某個數值，就顯示返回頂部按鈕，否則隱藏。

        // 某個數值為主體部分（main）的上邊框貼合屏幕上邊框時，即main.offsetTop

        if (window.pageYOffset >= mainTop) {

          goTop.style.display = "block";

        } else {

          goTop.style.display = "none";

        }

      });

      goTop.addEventListener("click", function () {

        window.scroll(0, 0);

      });

    </script>

  </body>

</html>

#### ③ click延時解決方案

移動端click事件會有300ms延時，原因是移動端屏幕雙擊會縮放頁面（double tap to zoom）。

如果用戶在300毫秒（300ms）內單擊，則是click事件；300毫秒內雙擊，則是縮放頁面事件。這300毫秒就是系統用來等待用戶是否雙擊的時間。

解決方案：

1. 禁用縮放。瀏覽器禁用默認的雙擊縮放行為，從而去掉300ms的點擊延遲。
   * 在視口標籤裡，有這樣一句話：user-scalable=no
   * 例如：

<meta

      name="viewport"

      content="width=device-width, initial-scale=1.0 user-scalable=no"

    />

1. 利用touch事件自己封裝函數，解決300ms延遲。（了解）
   * 原理：
     + 手指觸摸屏幕，記錄當前觸摸時間
     + 手指離開屏幕，用離開的時間減去觸摸的時間
     + 如果時間小於150ms，且沒有滑動過屏幕，就定義為點擊
2. 使用插件fastclick解決300ms延遲。

## 27 移動端常用開發插件

### （1）什麼是插件

JS插件是JS文件，它遵循一定規範編寫，方便程序展示效果，擁有特定功能且方便調用。比如輪播圖和瀑布流插件。

JS插件一般是為了解決某個問題而存在，功能單一，比較小。

比如為了頁面中的盒子做動畫，而專門寫了個animate.js文件。這就是個插件。

### （2）fastclick插件

解決300ms延遲

GitHub官網地址：https://github.com/ftlabs/fastclick

使用方式：

* 在GitHub地址的lib文件裡，有個fastclick.js文件。在線打開這個文件，複製裡面的所有代碼。然後在要用到這個插件的項目裡新建一個fastclick.js文件，把剛剛複製的代碼都放進去。
* 然後在HTML文件中引入這個js文件。
* 然後在HTML文件的body中的script中寫上：

  <body>

    <script>

      if ("addEventListener" in document) {

        document.addEventListener(

          "DOMContentLoaded",

          function () {

            FastClick.attach(document.body);

          },

          false

        );

      }

    </script>

  </body>

### （3）Swiper觸摸滑動插件

中文官網地址：<https://www.swiper.com.cn/>

* 在獲取Swiper裡下載Swiper。
* 下載後解壓，demos文件夾裡有各種輪播圖樣式。Swiper網站上有“查看Swiper演示”，先在這裡確定要那種輪播圖樣式。比如要280的樣式，然後去demos裡找到280開頭的HTML文件。
* 打開280-autoplay.html後，在頁面單擊右鍵，選擇“查看網頁源代碼”，找到相關的HTML樣式，引入CSS、JS的路徑。
* 在自己的HTML文件裡引入上述css和js文件
* 如果自己的js文件依賴Swiper的js文件，自己的js文件要寫在Swiper的js文件的下面。
* 複製HTML代碼、CSS代碼和JS代碼到自己的HTML文件裡。

使用方法可以參看上面的導航欄：中文教程-Swiper使用方法

更改JS參數，在這裡改：

     var swiper = new Swiper(".swiper-container", {

不懂參數什麼意思的話，到Swiper首頁上面的導航欄：API文檔

如果要改css樣式，最簡單的方法是自己寫個樣式，覆蓋原來的。要改那個樣式，就先用f12找到那個類名，然後用這個類名重新寫一遍相關樣式，並在最後面用!important提高重要性。

範例代碼：

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <link

      rel="stylesheet"

      href="./swiper-6.4.15/swiper-master/package/swiper-bundle.min.css"

    />

    <title>Document</title>

    <style>

      html,

      body {

        position: relative;

        height: 100%;

      }

      body {

        background: #eee;

        font-family: Helvetica Neue, Helvetica, Arial, sans-serif;

        font-size: 14px;

        color: #000;

        margin: 0;

        padding: 0;

      }

      .swiper-container {

        width: 100%;

        /\* height: 100%; \*/

      }

      .swiper-slide {

        text-align: center;

        font-size: 18px;

        background: #fff;

        /\* Center slide text vertically \*/

        display: -webkit-box;

        display: -ms-flexbox;

        display: -webkit-flex;

        display: flex;

        -webkit-box-pack: center;

        -ms-flex-pack: center;

        -webkit-justify-content: center;

        justify-content: center;

        -webkit-box-align: center;

        -ms-flex-align: center;

        -webkit-align-items: center;

        align-items: center;

      }

      .swiper-slide img {

        width: 100%;

      }

    </style>

  </head>

  <body>

    <!-- Swiper -->

    <div class="swiper-container">

      <div class="swiper-wrapper">

        <div class="swiper-slide"><img src="./2-2 picm1.jpg" alt="" /></div>

        <div class="swiper-slide"><img src="./2-2 picm2.jpg" alt="" /></div>

        <div class="swiper-slide"><img src="./2-2 picm3.jpg" alt="" /></div>

      </div>

      <!-- Add Pagination -->

      <div class="swiper-pagination"></div>

    </div>

    <!-- Swiper JS -->

    <script src="./swiper-6.4.15/swiper-master/package/swiper-bundle.min.js"></script>

    <!-- Initialize Swiper -->

    <script>

      var swiper = new Swiper(".swiper-container", {

        loop: true, // 循环模式选项

        spaceBetween: 30,

        centeredSlides: true,

        autoplay: {

          delay: 2500,

          disableOnInteraction: false,

        },

        pagination: {

          el: ".swiper-pagination",

          clickable: true,

        },

      });

    </script>

  </body>

</html>

### （4）其它移動端常見插件

superslide: <http://www.superslide2.com>

iscroll: <https://github.com/cubiq/iscroll>

插件使用方法：

1. 明確插件功能
2. 去官網查看使用說明
3. 下載插件
4. 打開demo示例文件，查看需要引入的相關文件，然後依次引入
5. 複製demo示例文件中的HTML、css、js代碼
6. 根據需求，調整相關參數

### （5）案例：視頻插件zy.media.js

H5提供了video標籤，但是瀏覽器的支持情況不同。

不同的視頻格式文件，通過source解決。

但是外觀樣式、暫停、播放、全屏等功能只能自己寫代碼解決。

這時可以使用插件方式。

地址：<https://github.com/ireaderlab/zyMedia>

## 28 移動端常用開發框架

框架，是一套架構，向用戶提供一套較為完整的解決方案。開發者按照框架規範進行開發。

框架，大而全，一整套解決方案。前端常用的框架有Bootstrap、Vew、Angular、React等。既能開發PC端，也能開發移動端。

插件，小而專一，某個功能的解決方案。前端常用的移動插件有Swiper，superslide，iscroll等。

### （1）Bootstrap

Bootstrap是一個簡潔、高效的前端開發框架，能開發PC端，也能開發移動端。

Bootstrap JS插件使用步驟：

1. 引入相關JS文件
2. 複製HTML結構
3. 修改相對樣式
4. 修改相應JS參數
   * 比如修改輪播圖的寬高，重要的兩個參數是輪播圖最外層盒子的寬高和圖片的寬高。找到相應的類，重新寫寬高，並寫上! important提高重要性。

## 29 本地存儲

隨著互聯網快速發展，基於網頁的應用越來越普及，同時也變得越來越複雜，為了滿足各種需求，經常需要在本地存儲大量數據，HTML5規範提出了相關解決方案。

本地存儲特性

1. 數據存儲在用戶瀏覽器中
2. 設置、讀取方便，甚至頁面刷新都不丟失數據
3. 容量較大，sessionStorage約5M（能存幾百萬字），localStorage約20M
4. 只能存儲字符串，可以將對象JSON.stringify()編碼後存儲

### （1）window.sessionStorage

① 生命週期為關閉瀏覽器窗口。

② 在同一個窗口（頁面）下，數據可以共享。

③ 以鍵值對的形式存儲使用。

④ 存儲數據語法：sessionStorage.setItem(key, value)

* 值存在瀏覽器裡，按F12，選擇application，看左邊的Session Storage裡面的file://

⑤ 獲取數據語法：sessionStorage.getItem(key)

⑥ 刪除數據語法：sessionStorage.removeItem(key)

⑦ 刪除所有數據（慎用）：sessionStorage.clear()

示例代碼：

  <body>

    <input type="text" />

    <button class="set">存儲數據</button>

    <button class="get">獲取數據</button>

    <button class="remove">刪除數據</button>

    <button class="del">清空所有數據</button>

    <div></div>

    <input type="text" id="inpt2" />

    <button class="set" id="btn2">存儲數據2</button>

    <script>

      var inpt = document.querySelector("input");

      var set = document.querySelector(".set");

      var get = document.querySelector(".get");

      var div = document.querySelector("div");

      var remove = document.querySelector(".remove");

      var del = document.querySelector(".del");

      var inpt2 = document.querySelector("#inpt2");

      var btn2 = document.querySelector("#btn2");

      set.addEventListener("click", function () {

        // 點擊“存儲數據”按鈕，存儲表單的值

        var val = inpt.value;

        sessionStorage.setItem("uname", val);

      });

      get.addEventListener("click", function () {

        // 點擊“獲取數據”按鈕，獲取存儲的值

        var val = sessionStorage.getItem("uname");

        div.innerHTML = val;

      });

      remove.addEventListener("click", function () {

        // 點擊“刪除數據”按鈕，刪除存儲的值

        sessionStorage.removeItem("uname");

      });

      btn2.addEventListener("click", function () {

        // 點擊“存儲數據”按鈕，存儲表單的值

        var val = inpt2.value;

        sessionStorage.setItem("uname2", val);

      });

      del.addEventListener("click", function () {

        // 點擊“清空所有數據”按鈕，清空所有存儲的值

        sessionStorage.clear();

      });

    </script>

  </body>

### （2）window.localStorage

① 生命週期永久生效，除非手動刪除，否則關閉頁面也會存在

② 可以多窗口（頁面）共享（同一瀏覽器可以共享）

③ 以鍵值對的形式存儲使用

④ 存儲數據語法：localStorage.setItem(key, value)

* 值存在瀏覽器裡，按F12，選擇application，看左邊的Local Storage裡面的file://

⑤ 獲取數據語法：localStorage.getItem(key)

⑥ 刪除數據語法：localStorage.removeItem(key)

⑦ 刪除所有數據語法：localSotrage.clear()

jQuery

## 1 jQuery概述

### （1）JavaScript庫

倉庫：可以把很多東西放到這個倉庫裡面。

JavaScript庫：即library，封裝好的特定的（方法和函數）的集合。這裡封裝了許多函數，比如動畫animate、hide、show，比如獲取元素等。

簡單理解：JavaScript庫就是一個JS文件，對原生js代碼進行了封裝，這樣可以快速高效地使用這些封裝好的功能。

比如jQuery就是JavaScript的一個庫，裡面基本是函數（方法），可以便捷地操作DOM。

常見的JavaScript庫有jQuery，Prototype，YUI，Dojo，Ext JS，移動端的zepto。主要學習的是jQuery。

### （2）jQuery的概念

jQuery是一個快速、簡潔的JavaScript庫，其設計的宗旨是“Write Less，Do More”，即寫更少的代碼，做更多的事情。

jQuery封裝了JavaScript常用的功能代碼，優化了DOM操作、事件處理、動畫設計和Ajax交互。可以讓開發人員大大加快開發速度。

學習jQuery本質，就是學習調用這些函數（方法）。

優點：

* 輕量級。核心文件才幾十kb，不會影響頁面加載速度。
* 誇瀏覽器兼容。基本兼容了現在的主流瀏覽器。
* 鏈式編程，隱式迭代。
* 支持事件、樣式、動畫，大大簡化了DOM操作。
* 支持插件拓展開發。有著豐富的第三方插件，例如：樹形菜單、日期控件、輪播圖等。
* 免費，開源。

## 2 jQuery基本使用

### （1）下載jQuery

<https://jquery.com>

版本：

* 1X：兼容IE 678等低版本瀏覽器，官網不再更新
* 2X：不兼容IE 678 等低版本瀏覽器，官網不再更新
* 3X：不兼容IE 678 等低版本瀏覽器，是官方主要更新維護的版本

在jQuery的download頁面

Download the compressed, production jQuery 3.6.0 指生產用jQuery，被壓縮過的。

Download the uncompressed, development jQuery 3.6.0 指開發用jQuery，沒被壓縮過的。

使用jQuery：

在download頁面點擊上述任意1個文件，在線打開，複製所有代碼，拷貝到新建的jquery.mini.js或者jquery.js裡面。然後把這個js文件引入到相關HTML文件中。

### （2）jQuery的入口函數

等著頁面加載完畢再執行代碼，有兩種寫法：

$(function(){

…

});

$(document).ready(function(){

…

});

這兩種方法的特點：

① 等著DOM結構渲染完畢即可執行內部代碼，不必等到所有外部資源加載完成，jquery幫程序員完成了封裝。

② 相當於原生js中的DOMContentLoaded。

③ 不同於原生js中的load事件，是等頁面文檔、外部的js文件、CSS文件、圖片加載完畢才執行內部代碼。

④ 更推薦使用第一種方式，因為寫起來更簡潔。

示例代碼：

    <script src="./jquery.mini.js"></script>

    <style>

      div {

        width: 100px;

        height: 100px;

        background-color: cadetblue;

      }

    </style>

  <body>

    <script>

      // 和原生js代碼一樣，如果寫在HTML代碼上面，無法生效，所以需要入口函數;

      $(function () {

        $("div").hide();

      });

    </script>

    <div></div>

  </body>

### （3）jQuery的頂級對象$

① $是jQuery的別稱，在代碼中可以使用jQuery代替$，但一般為了方便，通常直接使用$。

$("div").hide();

相當於

jQuery("div").hide();

② $是jQuery的頂級對象，相當於原生JavaScript中的window。把元素利用$包裝成jQuery對象，就可以調用jQuery的方法。

比如上面的例子中，用$調用了div元素，把它包裝成jQuery對象，然後調用了jQuery的hide()方法。

## 3 DOM對象和jQuery對象

### （1） 區別

原生就是獲取過來的對象是DOM對象。DOM對象只能使用原生的JavaScript屬性和方法。

var box = document.querySelector("div");

jQuery方法獲取的對象是jQuery對象。

$("div");

jQuery對象本質：利用$對DOM對象包裝後產生的對象（偽數組形式存儲）。

jQuery對象只能使用jQuery方法。

### （2） 相互轉換

jQuery只提供了一些常用的屬性和方法，某些不常用的JS屬性和方法沒有封裝，要使用這些未封裝的屬性和方法，必須把jQuery對象轉換為DOM對象才能使用。

#### ① DOM對象轉換為jQuery對象

$(DOM對象) （不用加引號）

      var box = document.querySelector("div");

      $(box);

#### ② jQuery對象轉換為DOM對象

兩種方式：

$(‘div’)[index] index是索引號

$(‘div’).get(index) index是索引號

因為jQuery對象是偽數組，DOM對象是這偽數組裡的元素，所以通過讀取jQuery這偽數組的方式獲取DOM對象。

## 4 jQuery常用API

### （1）jQuery選擇器

原生JS獲取元素方式很多，且兼容性不一致，因此jQuery做了封裝，統一獲取元素的標準。

$(“選擇器”) //裡面選擇器直接寫CSS選擇器即可，但要加引號。

|  |  |  |
| --- | --- | --- |
| 名稱 | 用法 | 描述 |
| ID選擇器 | $(“#id”); | 獲取指定ID的元素 |
| 全選選擇器 | $(“\*”); | 匹配所有元素 |
| 類選擇器 | $(”.class”); | 獲取同一類class的所有元素 |
| 標籤選擇器 | $(“div”); | 獲取同一類標籤的所有元素 |
| 並集選擇器 | $(“div,p,li”); | 選取多個元素 |
| 交集選擇器 | $(“li.curent”); | 交集元素 |
| 子代選擇器 | $(“ul>li”); | 獲取子代元素，但不會獲得孫代及以後的元素。 |
| 後代選擇器 | $(“ul li”); | 獲取ul以下層級的所有元素。 |

### （2）jQuery隱式迭代（重要）

jQuery遍歷內部DOM元素（偽數組形式存儲）的過程就叫做隱式迭代。

在這個過程中，jQuery循環遍歷匹配到的所有元素，執行相應的方法，不用開發者自己去循環遍歷，簡化開發過程。

### （3）jQuery篩選選擇器

|  |  |  |
| --- | --- | --- |
| 語法 | 示例 | 描述 |
| :first | $(“li:first”) | 獲取第一個li元素 |
| :last | $(“li:last”) | 獲取最後一個li元素 |
| :eq(index) | $(“li:eq(2)”) | 獲取到的li元素中，選擇索引號為2的元素，索引號index從0開始。 |
| :odd | $(“li:odd”) | 獲取到的li元素中，選擇索引號為奇數的元素 |
| :even | $(“li:even”) | 獲取到的li元素中，選擇索引號為偶數的元素 |

### （4）jQuery篩選方法

|  |  |  |
| --- | --- | --- |
| 語法 | 示例 | 說明 |
| parent() 重點 | $(“li”).parent(); | 查找最近一級的上級元素 |
| children(selector)  重點 | $(“ul”).children(“li”); | 相當於$(“ul>li”)，最近一級子元素 |
| find(selector)  重點 | $(“ul”).find(“li”); | 相當於$(“ul li”)，後代選擇器 |
| siblings(selector)  重點 | $(”.first”).siblings(“li”); | 查找兄弟節點，不包括自己本身 |
| nextAll([expr]) | $(“.first”).nextAll(); | 查找當前元素之後的所有同級元素 |
| prevAll([expr]) | $(“.last”).prevAll(); | 查找當前元素之前的所有同級元素 |
| hasClass(class) | $(“div”).hasClass(“protected”) | 檢查當前的元素是否含有某個特定的類，如果有，則返回true |
| eq(index)  重點 | $(“li”).eq(2); | 相當於$(“li:eq(2)”), index從0開始  更推薦這種方法而非選擇器，因為index沒有寫在引號裡面，方便修改。 |
| parents(“選擇器”) | $(“.box”).paretns(“.wrap”)  返回.box的上級中，類為wrap的所有上級元素。 | 返回指定的上級元素（直接上級和上級的上級）。  parents()括號裡不寫參數，就會把所有的上級元素都拿到。括號裡寫參數，返回的是指定的上級元素。 |

#### ① 案例：鼠標經過，顯示下拉菜單；鼠標離開，隱藏下拉菜單

    <style>

      ul {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .nav > li {

        float: left;

        width: 100px;

        height: 30px;

        line-height: 30px;

        background-color: #678f8d;

        color: #ff6670;

      }

      .nav > li:hover {

        cursor: pointer;

      }

      .nav ul {

        display: none;

      }

    </style>

  <body>

    <ul class="nav">

      <li>

        1

        <ul>

          <li>1-1</li>

          <li>1-2</li>

          <li>1-3</li>

        </ul>

      </li>

      <li>

        2

        <ul>

          <li>2-1</li>

          <li>2-2</li>

          <li>2-3</li>

        </ul>

      </li>

      <li>

        3

        <ul>

          <li>3-1</li>

          <li>3-2</li>

          <li>3-3</li>

        </ul>

      </li>

    </ul>

    <script>

      $(".nav>li").mouseover(function () {

        // $(this) jQuery当前元素  this不要加引號

        $(this).children("ul").show();

      });

      $(".nav>li").mouseout(function () {

        $(this).children("ul").hide();

      });

    </script>

  </body>

#### ② 示例代碼

  <body>

    <ul>

      <li>1</li>

      <li class="selected">2</li>

      <li>3</li>

      <li>4</li>

      <li>5</li>

    </ul>

    <script>

      $(".selected").siblings("li").css("color", "#ff6670");

    </script>

  </body>

#### ③ 案例： 點擊一個按鈕選中變紅色，其它按鈕不變色

jQuery的排它思想

  <body>

    <button>按鈕1</button><button>按鈕2</button><button>按鈕3</button

    ><button>按鈕4</button><button>按鈕5</button>

    <script>

      //  1 隱式迭代，給所有的按鈕都綁定了點擊事件

      $("button").click(function () {

        // 2 當前的元素變化背景顏色

        $(this).css("background", "#ff6670");

        // 3 其餘的同級元素去掉背景顏色，隱式迭代

        $(this).siblings("button").css("background", "");

      });

    </script>

  </body>

#### ④ 案例：鼠標經過li顯示相應的盒子

核心原理：鼠標經過某個li，顯示相應的盒子內容，其餘盒子隱藏

jQuery得到當前元素索引號$(this).index();

    <style>

      ul,

      li {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .wrap {

        position: relative;

        width: 200px;

        height: 60px;

        border: 1px solid #ccc;

      }

      li {

        width: 20px;

        height: 20px;

        border: 1px solid #ccc;

        box-sizing: border-box;

      }

      li:hover {

        cursor: pointer;

      }

      .pic-wrap {

        position: absolute;

        width: 180px;

        height: 60px;

        top: 0;

        left: 20px;

      }

      .pic-wrap > div {

        position: absolute;

        display: none;

        top: 0;

        left: 0;

        width: 100%;

        height: 100%;

      }

      .pic-wrap .bck1 {

        display: block;

        background-color: #678f8d;

      }

      .bck2 {

        background-color: #ffd000;

      }

      .bck3 {

        background-color: #ff6670;

      }

    </style>

  <body>

    <div class="wrap">

      <ul>

        <li>1</li>

        <li>2</li>

        <li>3</li>

      </ul>

      <div class="pic-wrap">

        <div class="bck1">1</div>

        <div class="bck2">2</div>

        <div class="bck3">3</div>

      </div>

    </div>

    <script>

      $("ul")

        .children("li")

        .mouseover(function () {

          // 拿到當前的索引號

          var i = $(this).index();

          // 讓與當前索引號相同的div顯示，其餘div隱藏

          $(".pic-wrap").children("div").eq(i).show();

          $(".pic-wrap").children("div").eq(i).siblings("div").hide();

        });

    </script>

  </body>

### （5）鏈式編程

鏈式編程可以節省代碼量，讓代碼看起來更優雅

比如，把自己的顏色變紅色，把同級其它元素變為默認顏色，就可以這樣寫：

$(this).css(‘color’,’red’).sibling().css(‘color’,’’);

  <body>

    <button>按鈕1</button><button>按鈕2</button><button>按鈕3</button

    ><button>按鈕4</button><button>按鈕5</button>

    <script>

      $(function () {

        $("button").click(function () {

          //  讓當前字體顏色變為紅色，其餘同級元素字體顏色為空

          $(this).css("color", "#ff6670").siblings().css("color", "");

        });

      });

    </script>

  //  同級元素字體顏色為紅，本身不變色

          $(this).siblings().css("color", "#ff6670");

        });

          //  同級元素的父元素字體顏色為紅，本身不變色

          $(this).siblings().parent().css("color", "#ff6670");

### （6）jQuery樣式操作

#### ① 用css方法修改單個樣式

a 參數只寫屬性名，返回屬性值

$(this).css(“color”); （帶有單位的字符串）

b 參數是屬性名、屬性值、逗號分隔，是設置一組樣式，屬性必須加引號。值如果是數字可以不用跟單位和引號。

$(this).css(“color”,” #ff6670”);

$(this).css(“width”,”70”);

c 參數可以是對象的形式，用於設置多組樣式。屬性名和屬性值用冒號隔開，屬性可以不用加引號。復合屬性（fontSize，backgroundColor）要用駝峰命名法。如果值不是數字，要加引號。

$(this).css({“color”:”white”,”font-size”:”20px”})

  <body>

    <div>惠風和暢</div>

    <script>

      $("div").css({

        color: "#ff6670",

        fontSize: 20,

        width: 400,

        height: 400,

      });

    </script>

  </body>

#### ② 操作類修改多個樣式

作用等於classList。可以操作類樣式。注意參數名稱前面不用加點。

a 添加類。這是追加類名，不影響已有的類名。

$(“div”).addClass(“selected”);

b 刪除類。不影響已有的類名。

$(“div”).removeClass(“selected”);

c 切換類，沒有就加上這類，有就刪除這個類。不影響已有的類名。

$(“div”).toggleClass(“selected”);

示例代碼

    <style>

      .selected {

        width: 500px;

        height: 500px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <div>惠風和暢</div>

    <button>刪除類</button>

    <span>toggle類</span>

    <script>

      $("div").click(function () {

        $(this).addClass("selected");

      });

      $("button").click(function () {

        $("div").removeClass("selected");

      });

      $("span").click(function () {

        $("div").toggleClass("selected");

      });

    </script>

  </body>

#### ③ 案例：切換tab欄

    <style>

      ul,

      li {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .wrap {

        width: 300px;

        height: 100px;

        border: 1px solid #ccc;

        box-sizing: border-box;

      }

      .tab-list {

        height: 20px;

      }

      .tab-list li {

        float: left;

        width: 33.33%;

        height: 20px;

        border: 1px solid #ccc;

        box-sizing: border-box;

      }

      .tab-selected {

        background-color: #678f8d;

        color: #ff6670;

      }

      .tab-items li {

        display: none;

        width: 100%;

        height: 80px;

        border: 1px solid #ccc;

        box-sizing: border-box;

      }

      .tab-items li:first-child {

        display: block;

      }

    </style>

  <body>

    <div class="wrap">

      <ul class="tab-list">

        <li class="tab-selected">1</li>

        <li>2</li>

        <li>3</li>

      </ul>

      <ul class="tab-items">

        <li>(1)</li>

        <li>(2)</li>

        <li>(3)</li>

      </ul>

    </div>

    <script>

      $(".tab-list li").click(function () {

        // 點擊一個tab-list的li，添加選中的類，並移除其同級元素的選中類。鏈式編程。

        $(this).addClass("tab-selected").siblings().removeClass("tab-selected");

        // 獲取當前li的索引號

        var i = $(this).index();

        // 根據索引號得到相應的tab-item，將其設為顯示，並將其他tab-item隱藏.鏈式編程。

        $(".tab-items li")

          .eq(i)

          .css("display", "block")

          .siblings()

          .css("display", "none");

      });

    </script>

  </body>

#### ④ 類操作與className區別

原生JS中className會覆蓋元素原先的類名。

而jQuery的類操作只是對指定類進行操作，不影響原先的類名。

### （7）jQuery動畫效果

在jQuery官網API的effects頁面可以看到這些動畫效果的詳細說明

常見的jQuery動畫效果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 顯示隱藏 | 滑動 | 淡入淡出 | 自定義動畫 |
| show()  hide()  toggle() | slideDown()  slideUp()  slideToggle() | fadeIn()  fadeout()  fadeToggle()  fadeTo() | animate() |

#### ① 顯示隱藏

顯示語法：

show([speed, [easing],[fn]])

隱藏語法：

hide([speed, [easing],[fn]])

切換語法

toggle([speed, [easing],[fn]])

參數：

參數都可以省略，默認無動畫直接顯示。一般情況下不加參數。

speed：三種約定速度之一的字符串（”slow”, “normal”, “last”）或表示動畫的時長的毫秒數值（如：1000）。

easing：用於指定切換效果，默認值是”swing”，還可用”linear”。

fn：回調函數，在動畫完成時執行的函數，每個元素執行一次。

    <style>

      div {

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <script>

      $(function () {

        $("button")

          .eq(0)

          .click(function () {

            $("div").show("slow", function () {

              console.log("顯示");

            });

          });

        $("button")

          .eq(1)

          .click(function () {

            $("div").hide(1000, "linear", function () {

              console.log("隱藏");

            });

          });

        $("button")

          .eq(2)

          .click(function () {

            $("div").toggle(1000);

          });

      });

    </script>

    <button>顯示</button><button>隱藏</button><button>切換</button>

    <div></div>

  </body>

#### ② 滑動

語法：

slideDown([speed, [easing],[fn]])

slideUp([speed, [easing],[fn]])

slideToggle([speed, [easing],[fn]])

參數顯示隱藏。

參數都可以省略，默認無動畫直接顯示。

speed：三種約定速度之一的字符串（”slow”, “normal”, “last”）或表示動畫的時長的毫秒數值（如：1000）。

easing：用於指定切換效果，默認值是”swing”，還可用”linear”。

fn：回調函數，在動畫完成時執行的函數，每個元素執行一次。

    <script>

      $(function () {

        $("button")

          .eq(0)

          .click(function () {

            $("div").slideDown("slow", function () {

              console.log("顯示");

            });

          });

        $("button")

          .eq(1)

          .click(function () {

            $("div").slideUp(1000, "linear", function () {

              console.log("隱藏");

            });

          });

        $("button")

          .eq(2)

          .click(function () {

            $("div").slideToggle(1000);

          });

      });

    </script>

#### ③ 事件切換

鼠標經過（mouseenter）和鼠標離開（mouseleave）的復合寫法

hover([over,]out)

over：鼠標移到元素上要觸發的函數（相當於mouseenter）

out：鼠標移出元素要觸發的函數（相當於mouseleave）

hover裡面如果只寫1個函數，那麼鼠標經過和鼠標離開都只執行這1個函數。

#### ④ 案例：事件切換案例，鼠標經過導航欄，下拉菜單

    <style>

      ul {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .nav > li {

        float: left;

        width: 100px;

        height: 30px;

        line-height: 30px;

        background-color: #678f8d;

        color: #ff6670;

      }

      .nav > li:hover {

        cursor: pointer;

      }

      .nav ul {

        display: none;

      }

    </style>

  <body>

    <ul class="nav">

      <li>

        1

        <ul>

          <li>1-1</li>

          <li>1-2</li>

          <li>1-3</li>

        </ul>

      </li>

      <li>

        2

        <ul>

          <li>2-1</li>

          <li>2-2</li>

          <li>2-3</li>

        </ul>

      </li>

      <li>

        3

        <ul>

          <li>3-1</li>

          <li>3-2</li>

          <li>3-3</li>

        </ul>

      </li>

    </ul>

    <script>

      $(".nav>li").hover(

        function () {

          $(this).children("ul").slideDown(200);

        },

        function () {

          $(this).children("ul").slideUp(200);

        }

      );

    </script>

  </body>

上面的script裡面的代碼還可改為：

    <script>

      // hover裡面如果只寫1個函數，那麼鼠標經過和鼠標離開都只執行這1個函數。

      $(".nav>li").hover(function () {

        $(this).children("ul").slideToggle(200);

      });

    </script>

#### ⑤ 動畫列隊及其停止排隊方法

動畫或者效果一旦觸發就會執行，如果多次觸發，就造成多個動畫或者效果排隊執行。

比如剛剛的鼠標經過導航欄觸發下拉菜單，如果及其快速經過，會觸發多次下拉，而且都會依次完成。

要解決這個問題，用：

stop()

stop()方法用於停止動畫或效果，不管上一次動畫是否結束，都將其結束，然後再開始新的動畫。

stop()寫到動畫或者效果的前面，用於停止上一次動畫；而不能寫在動畫或者效果後面，否則動畫或者效果剛一執行，就被停止了。

所以把這個案例的script修改一下，得到：

    <script>

      // hover裡面如果只寫1個函數，那麼鼠標經過和鼠標離開都只執行這1個函數。

      $(".nav>li").hover(function () {

        $(this).children("ul").stop().slideToggle(200);

      });

    </script>

#### ⑥ 淡入淡出

淡入語法：

fadeIn([speed, [easing],[fn]])

淡出語法：

fadeOut([speed, [easing],[fn]])

切換語法

fadeToggle([speed, [easing],[fn]])

參數：

參數都可以省略，默認無動畫直接顯示。

speed：三種約定速度之一的字符串（”slow”, “normal”, “last”）或表示動畫的時長的毫秒數值（如：1000）。

easing：用於指定切換效果，默認值是”swing”，還可用”linear”。

fn：回調函數，在動畫完成時執行的函數，每個元素執行一次。

調整透明度

fadeTo([[speed], opacity, [easing],[fn]])

參數：

opacity透明度必須寫，取值0-1之間。

其它參數同上。

speed和opacity必須寫

    <style>

      div {

        display: none;

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <script>

      $(function () {

        $("button")

          .eq(0)

          .click(function () {

            $("div").fadeIn("slow", function () {

              console.log("淡入");

            });

          });

        $("button")

          .eq(1)

          .click(function () {

            $("div").fadeOut(1000, "linear", function () {

              console.log("淡出");

            });

          });

        $("button")

          .eq(2)

          .click(function () {

            $("div").fadeToggle(2000);

          });

        $("button")

          .eq(3)

          .click(function () {

            $("div").fadeTo(2000, 0.5);

          });

      });

    </script>

    <button>淡入</button><button>淡出</button><button>切換</button

    ><button>半透明</button>

    <div></div>

  </body>

#### ⑦ 案例：淡入淡出調整透明度

    <style>

      ul,

      li {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      ul {

        width: 768px;

        height: 273px;

        margin: 30px auto;

      }

      li {

        float: left;

        margin-left: 10px;

      }

      li:hover {

        cursor: pointer;

      }

    </style>

  <body>

    <ul>

      <li><img src="./images/2-3 pic1.jpg" alt="2-3 pic1.jpg" /></li>

      <li><img src="./images/2-3 pic2.jpg" alt="2-3 pic2.jpg" /></li>

      <li><img src="./images/2-3 pic3.jpg" alt="2-3 pic3.jpg" /></li>

      <li><img src="./images/2-3 pic4.jpg" alt="2-3 pic4.jpg" /></li>

    </ul>

    <script>

      $(function () {

        // $("ul>li").mouseenter(function () {

        //   $(this).siblings().stop().fadeTo(1000, 0.5);

        // });

        // $("ul>li").mouseleave(function () {

        //   $(this).siblings().stop().fadeTo(1000, 1);

        // });

        // 上述代碼可改為

        $("ul>li").hover(

          function () {

            $(this).siblings().stop().fadeTo(1000, 0.5);

          },

          function () {

            $(this).siblings().stop().fadeTo(1000, 1);

          }

        );

      });

    </script>

  </body>

#### ⑧ 自定義動畫animate

animate(params, [speed], [easing], [fn])

參數裡面params是想要更改的樣式屬性，以對象的形式傳遞，必須寫。屬性名可以不帶引號，如果是符合屬性則需採取駝峰命名法，比如borderLeft。其餘參數都可以省略。

其它參數同上。

speed：三種約定速度之一的字符串（”slow”, “normal”, “last”）或表示動畫的時長的毫秒數值（如：1000）。

easing：用於指定切換效果，默認值是”swing”，還可用”linear”。

fn：回調函數，在動畫完成時執行的函數，每個元素執行一次。

    <style>

      div {

        position: relative;

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <button>動起來</button>

    <div></div>

    <script>

      $(function () {

        $("button").click(function () {

          $("div").animate({ left: 500, opacity: 0.3 }, "slow");

        });

      });

    </script>

  </body>

#### ⑨ 案例：手風琴

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>手風琴</title>

    <script src="./jquery.mini.js"></script>

    <style>

      ul,

      li {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .accordion-wrap li {

        float: left;

        position: relative;

        width: 69px;

        height: 69px;

      }

      .accordion-wrap a img {

        height: 69px;

      }

      .accordion-wrap a .sm-pic {

        position: absolute;

        left: 0;

        top: 0;

      }

      .accordion-wrap a .big-pic {

        display: none;

      }

      .accordion-wrap .current {

        width: 224px;

      }

      .current a .sm-pic {

        display: none;

      }

      .current a .big-pic {

        display: block;

      }

    </style>

  </head>

  <body>

    <div class="accordion-wrap">

      <ul>

        <li class="current">

          <a href="#">

            <img src="./images/2-4/c1.jpg" alt="c1" class="sm-pic" />

            <img src="./images/2-4/c.png" alt="c" class="big-pic" />

          </a>

        </li>

        <li>

          <a href="#">

            <img src="./images/2-4/h1.jpg" alt="h1" class="sm-pic" />

            <img src="./images/2-4/h.png" alt="h" class="big-pic" />

          </a>

        </li>

        <li>

          <a href="#">

            <img src="./images/2-4/l1.jpg" alt="l1" class="sm-pic" />

            <img src="./images/2-4/l.png" alt="l" class="big-pic" />

          </a>

        </li>

        <li>

          <a href="#">

            <img src="./images/2-4/m1.jpg" alt="m1" class="sm-pic" />

            <img src="./images/2-4/m.png" alt="m" class="big-pic" />

          </a>

        </li>

        <li>

          <a href="#">

            <img src="./images/2-4/w1.jpg" alt="w1" class="sm-pic" />

            <img src="./images/2-4/w.png" alt="w" class="big-pic" />

          </a>

        </li>

        <li>

          <a href="#">

            <img src="./images/2-4/z1.jpg" alt="z1" class="sm-pic" />

            <img src="./images/2-4/z.png" alt="z" class="big-pic" />

          </a>

        </li>

      </ul>

    </div>

    <script>

      $(function () {

        $(".accordion-wrap li").mouseenter(function () {

          // 鼠標經過，當前li的寬度變為224px，裡面的小圖fadeOut，大圖fadeIn

          $(this)

            .stop()

            .animate({ width: 224 }, "slow")

            .find(".sm-pic")

            .stop()

            .fadeOut("slow")

            .siblings()

            .stop()

            .fadeIn("slow");

          // 當前li的siblings的li寬度變為69，裡面的小圖fadeIn，大圖fadeOut

          $(this)

            .siblings("li")

            .stop()

            .animate({ width: 69 }, "slow")

            .find(".sm-pic")

            .stop()

            .fadeIn("slow")

            .siblings()

            .stop()

            .fadeOut("slow");

        });

      });

    </script>

  </body>

</html>

### （8）jQuery屬性操作

#### ① 獲取和設置元素固有屬性值prop()

元素固有屬性是元素本身自帶屬性，比如<a>元素裡面的href，比如<input>元素裡面的type。

獲取屬性語法：

element.prop(“屬性”)

設置屬性語法

element.prop(“屬性”,”屬性值”)

  <body>

    <a href="https://www.bing.com"></a>

    <script>

      $(function () {

        console.log($("a").prop("href"));

        $("a").prop("title", "乾坤");

      });

    </script>

#### ② 設置和獲取元素自定義屬性值attr()

獲取屬性語法：

element.attr(“屬性”) //類似原生getAttribute()

設置屬性語法

element.attr(“屬性”,”屬性值”) //類似原生setAttribute()

該方法也可以獲取和修改H5自定義屬性（比如data-index）

  <body>

    <a href="https://www.bing.com" index="1" data-index="5"></a>

    <script>

      $(function () {

        console.log($("a").attr("index"));

        $("a").attr("data-index", "乾坤");

        $("a").attr("mark1", "臨江仙");

      });

    </script>

  </body>

#### ③ 數據緩存data()

data()方法可以在指定的元素上存取數據，不會修改DOM元素結構。一旦頁面刷新，之前存放的數據會被移除。

附加數據方法

element.data(“name”,”value”)

獲取數據方法

element.data(“name”)

該方法還可以讀取HTML5自定義屬性，比如data-index（獲取的時候直接寫element.data(“index”)，不用加data），得到的是數字型

  <body>

    <a href="https://www.bing.com" index="1" data-index="5"></a>

    <script>

      $(function () {

        $("a").data("uname", "andy");

        console.log($("a").data("uname")); //andy

        console.log($("a").data("index")); //5

      });

    </script>

  </body>

#### ④ 案例：全選

:checked選擇器 查找被選中的表單元素

  <body>

    全選<input type="checkbox" name="" class="ckAll" />

    <div class="cart-wrap">

      商品1<input type="checkbox" name="" class="ck-item" /> 商品2<input

        type="checkbox"

        name=""

        class="ck-item"

      />

      商品3<input type="checkbox" name="" class="ck-item" />

    </div>

    全選<input type="checkbox" name="" class="ckAll" />

    <script>

      $(function () {

        // 當前這個全選按鈕的狀態，決定了其它按鈕的全選狀態

        $(".ckAll").click(function () {

          var flag = $(this).prop("checked");

          $(".ck-item, .ckAll").prop("checked", flag);

        });

        // 小按鈕的狀態決定全選按鈕是否選中

        $(".ck-item").click(function () {

          // if (被選中的小複選框個數 ===3) {

          //   選中全選按鈕

          // }else{

          //   不要選中全選按鈕

          // }

          // console.log($(".ck-item:checked"));可以輸出所有被選中的複選框（數組的形式）

          console.log();

          var flag = true;

          // 只要被選中的商品複選框個數<商品複選框總個數，就把flag設為false，這個flag是給全選框的checked的值。

          if ($(".ck-item:checked").length < $(".ck-item").length) {

            flag = false;

          }

          $(".ckAll").prop("checked", flag);

        });

      });

    </script>

  </body>

### （9）jQuery文本屬性值

#### ① 普通元素內容html() （相當於原生innerHTML）

html() //獲取元素的內容，把標籤和內容都拿到

html(‘內容’) //設置元素的內容

  <body>

    <div>

      揚州慢

      <span>汴河曲</span>

    </div>

    <script>

      $(function () {

        console.log($("div").html()); //揚州慢和<span>汴河曲</span>都會獲取

        $("div").html("夏夜"); //把div裡面的所有內容包括span替換為夏夜

      });

    </script>

  </body>

#### ② 普通元素文本內容text() （相當於原生innerText）

  <body>

    <div>

      揚州慢

      <span>汴河曲</span>

    </div>

    <script>

      $(function () {

        console.log($("div").text()); //揚州慢和汴河曲都會獲取，不包括標籤

        $("div").text("夏夜"); //把div裡面的所有內容包括span替換為夏夜

      });

    </script>

  </body>

#### ③ 獲取和設置表單值val()

  <body>

    <input type="text" name="" id="" value="和氏之璧" />

    <script>

      $(function () {

        console.log($("input").val());

        $("input").val("隋氏之珠");

      });

    </script>

  </body>

#### ④ 案例：增加商品數量

    <style>

      span:hover {

        cursor: pointer;

      }

    </style>

  <body>

    <div class="item">

      <span class="increment">-</span

      ><input type="number" name="" id="" value="1" />

      <span class="decrement">+</span>

    </div>

    <div class="item">

      <span class="increment">-</span

      ><input type="number" name="" id="" value="1" />

      <span class="decrement">+</span>

    </div>

    <script>

      $(function () {

        $(".increment").click(function () {

          var num = parseInt($(this).siblings("input").val());

          // 如果數量小於0，就不能再減了

          if (num === 0) {

            return false;

          }

          num--;

          $(this).siblings("input").val(num);

        });

        $(".decrement").click(function () {

          var num = parseInt($(this).siblings("input").val());

          num++;

          $(this).siblings("input").val(num);

        });

      });

    </script>

  </body>

#### ⑤ 案例：修改商品數量，修改商品總價

  <body>

    <div class="item">

      單價￥<span class="price">133.33</span> <span class="decrement">-</span

      ><input type="number" name="" id="" value="1" class="item-num" />

      <span class="increment">+</span>總價￥<span class="priceT">133.33</span>

    </div>

    <div class="item">

      單價￥<span class="price">236.66</span> <span class="decrement">-</span

      ><input type="number" name="" id="" value="1" class="item-num" />

      <span class="increment">+</span>總價￥<span class="priceT">236.66</span>

    </div>

    <script>

      $(function () {

        $(".decrement").click(function () {

          // 用戶點擊減號

          var num = parseInt($(this).siblings("input").val());

          // 如果數量小於0，就不能再減了

          if (num === 0) {

            return false;

          }

          num--;

          $(this).siblings("input").val(num);

          // 商品總價跟著數量變化

          getPriceT($(this), num);

        });

        // 用戶點擊加號

        $(".increment").click(function () {

          var num = parseInt($(this).siblings("input").val());

          num++;

          $(this).siblings("input").val(num);

          // 商品總價跟著數量變化

          getPriceT($(this), num);

        });

        // 用戶直接修改input的值

        $(".item-num").change(function () {

          // 拿到改變後的input值

          var num = parseFloat($(this).val());

          // 商品總價跟著數量變化

          getPriceT($(this), num);

        });

        // 封裝一個函數，用來單個商品的總價

        function getPriceT(*obj*, *num*) {

          var priceT = parseFloat(obj.siblings(".price").text()) \* num;

          // 計算結果保留兩位小數 priceT.toFixed(2)

          obj.siblings(".priceT").text(priceT.toFixed(2));

        }

      });

    </script>

  </body>

### （10） jQuery元素操作

#### ① 遍歷元素

jQuery隱式迭代是對同一類元素做了同樣的操作。如果想給同一類元素做不同操作，要用到遍歷。

語法1：

$(“div”).each(function(index, domEle){ xxx; })

* each()方法遍歷匹配的每一個元素，主要用DOM處理。
* 裡面的回調函數有2個參數：index和domEle。
* index是每個元素的索引號，不一定叫index，可以自行指定，比如叫i、key、k。索引號從0開始。
* domEle是每個DOM元素對象，不是jQuery對象。想使用jQuery方法，必須把dom元素轉換為jQuery對象$(domEle)。domEle不一定叫這個名字，可以自行命名。

示例代碼：

  <body>

    <div>1</div>

    <div>2</div>

    <div>3</div>

    <script>

      $(function () {

        var divColor = ["#678f8d", "#ffd000", "#ff6670"];

        var sum = 0;

        $("div").each(function (*i*, *domEle*) {

          // 把三個盒子裡的文字顏色依次變為不同的顏色

          $(domEle).css("color", divColor[i]);

          // 把三個盒子裡的數字依次加起來求總和。

          sum += parseInt($(domEle).text());

        });

        console.log(sum);

      });

    </script>

  </body>

語法2：

$.each(object, function(index, element) {xxx;})

* $.each()方法可用於遍歷任何對象。主要用於數據處理，比如數組、對象。
* $.each()方法有2個參數，object寫上要遍歷的對象，第2個參數是函數。這兩個參數的名字不固定，可以改動。
* 裡面的函數有2個參數：index是每個元素的索引號，element遍歷內容（如果遍歷jQuery，得到dom對象）。
* 這個方法可以更好地處理數組和對象等數據。

  <body>

    <div>1</div>

    <div>2</div>

    <div>3</div>

    <script>

      $(function () {

        var divColor = ["#678f8d", "#ffd000", "#ff6670"];

        var obj = {

          name: "andy",

          age: 18,

          gender: "male",

        };

        $.each($("div"), function (*i*, *domEle*) {

          $(domEle).css("color", divColor[i]);

        });

        $.each(divColor, function (*i*, *element*) {

          console.log(element);

        });

        $.each(obj, function (*i*, *ele*) {

          console.log(i); //對象的屬性名name age gender

          console.log(ele); //對象的屬性值 andy 18 male

        });

      });

    </script>

  </body>

#### ② 案例：统计商品件数，总价

  <body>

    <div class="item">

      單價￥<span class="price">133.33</span> <span class="decrement">-</span

      ><input type="number" name="" id="" value="1" class="item-num" />

      <span class="increment">+</span>總價￥<span class="priceT">133.33</span>

    </div>

    <div class="item">

      單價￥<span class="price">236.66</span> <span class="decrement">-</span

      ><input type="number" name="" id="" value="1" class="item-num" />

      <span class="increment">+</span>總價￥<span class="priceT">236.66</span>

    </div>

    <div class="sum-wrap">

      共買了<span class="sumAll">0</span>件 共計￥<span class="priceAll"

        >0</span

      >

    </div>

    <script>

      $(function () {

        getSumPriceAll();

        $(".decrement").click(function () {

          // 用戶點擊減號

          var num = parseInt($(this).siblings("input").val());

          // 如果數量小於0，就不能再減了

          if (num === 0) {

            return false;

          }

          num--;

          $(this).siblings("input").val(num);

          // 商品總價跟著數量變化

          getPriceT($(this), num);

          // 商品總數量和所有總價跟著變化

          getSumPriceAll();

        });

        // 用戶點擊加號

        $(".increment").click(function () {

          var num = parseInt($(this).siblings("input").val());

          num++;

          $(this).siblings("input").val(num);

          // 商品總價跟著數量變化

          getPriceT($(this), num);

          // 商品總數量和所有總價跟著變化

          getSumPriceAll();

        });

        // 用戶直接修改input的值

        $(".item-num").change(function () {

          // 拿到改變後的input值

          var num = parseFloat($(this).val());

          // 商品總價跟著數量變化

          getPriceT($(this), num);

          // 商品總數量和所有總價跟著變化

          getSumPriceAll();

        });

        // 封裝一個函數，用來單個商品的總價

        function getPriceT(*obj*, *num*) {

          var priceT = parseFloat(obj.siblings(".price").text()) \* num;

          // 計算結果保留兩位小數 priceT.toFixed(2)

          obj.siblings(".priceT").text(priceT.toFixed(2));

        }

        //封裝一個函數，用來計算商品總個數和總價

        function getSumPriceAll() {

          var sum = 0;

          var priceAll = 0;

          // 遍歷輸入框拿到數量

          $(".item-num").each(function (*index*, *domEle*) {

            sum += parseInt($(domEle).val());

          });

          // 遍歷單個商品總價，拿到所有商品總價

          $.each($(".priceT"), function (*index*, *element*) {

            priceAll += parseFloat($(element).text());

          });

          $(".sumAll").text(sum);

          $(".priceAll").text(priceAll.toFixed(2));

        }

      });

    </script>

  </body>

#### ③ 創建、添加、刪除元素

創建语法：

$(“<li></li>”);

內部添加元素語法：

element.append(“內容”)

* 把內容放入匹配元素內部最後面，類似原生appendChild。

element.prepend(“內容”)

* 把內容放入匹配元素內部最前面。

外部添加元素語法：

element.after(“內容”) //把內容放到目標元素後面

element.before(“內容”) //把內容放到目標元素前面

刪除元素語法：

element.remove() //刪除匹配的元素（本身和所有的內容）

element.empty() //刪除匹配的元素集合中所有的子節點

element.html(“”) //清空元素內容。如果HTML括號裡面只寫””，會清空所有內容。如果寫了內容，會保留所寫的內容，其它的清空。

  <body>

    <ul id="box1">

      <li>li盒子</li>

    </ul>

    ——————

    <div id="box2">div盒子</div>

    ——————

    <div id="rem">

      刪除1

      <div>刪除1裡面的div</div>

    </div>

    ————————

    <div id="ept">

      刪除2

      <div>刪除2裡面的div</div>

    </div>

    ————————

    <div id="clr">

      刪除3

      <div>刪除3裡面的div</div>

    </div>

    <script>

      $(function () {

        // 創建元素

        var li1 = $("<li>創建的li1盒子</li>");

        var li2 = $("<li>創建的li2盒子</li>");

        var div1 = $("<div>創建的div1盒子</div>");

        var div2 = $("<div>創建的div2盒子</div>");

        // 內部添加元素(放在原有內容的最後)

        $("#box1").append(li1);

        // 內部添加元素(放在原有內容的最前)

        $("#box1").prepend(li2);

        // 外部添加元素（放在目標元素的後面）

        $("#box2").after(div1);

        // 外部添加元素（放在目標元素的前面）

        $("#box2").before(div2);

        // 刪除元素（本身和所有的內容）

        $("#rem").remove();

        // 刪除元素（刪除匹配的元素集合中所有的子節點）

        $("#ept").empty();

        // 刪除元素（清空元素內容。如果HTML括號裡面只寫””，會清空所有內容。如果寫了內容，會保留所寫的內容，其它的清空。）

        $("#clr").html("");

      });

    </script>

  </body>

#### ④ 案例：清空購物車

  <body>

    <div class="item">

      <input type="checkbox" name="" id="" class="ck-item" />商品1<span

        class="del-item"

        >刪除</span

      >

    </div>

    <div class="item">

      <input type="checkbox" name="" id="" class="ck-item" />商品2<span

        class="del-item"

        >刪除</span

      >

    </div>

    <div class="item">

      <input type="checkbox" name="" id="" class="ck-item" />商品3<span

        class="del-item"

        >刪除</span

      >

    </div>

    <div class="del-ck">刪除選中的商品</div>

    <div class="del-all">清空購物車</div>

    <script>

      $(function () {

        // 刪除單個商品

        $(".del-item").click(function () {

          $(this).parent().remove();

        });

        // 刪除選中的商品

        $(".del-ck").click(function () {

          // $(".item").each(function (index, domEle) {

          //   if ($(domEle).children(".ck-item").prop("checked")) {

          //     $(domEle).remove();

          //   }

          // });

          // 上面代碼可改為

          $(".ck-item:checked").parent().remove();

        });

        // 刪除所有商品

        $(".del-all").click(function () {

          $(".item").remove();

        });

        // 實際寫代碼時，記得每次刪除，都要去計算一下所選的商品件數和總價格數

      });

    </script>

  </body>

### （11）jQuery尺寸、位置操作

#### ① jQuery尺寸

|  |  |
| --- | --- |
| 語法 | 用法 |
| width()/ height() | 取得匹配元素寬度和高度值，只算width或者height |
| innerWidth()/ innerHeight() | 取得匹配元素寬度和高度值，包含padding |
| outerWidth()/ outerHeight() | 取得匹配元素寬度和高度值，包含padding和border |
| outerWidth(true) /outerHeight(true) | 取得匹配元素寬度和高度值，包含padding、border和margin |

* 以上參數為空，則是獲取相應的值。
* 返回的都是數字，不帶單位。
* 如果參數為數字，則是修改相應的值。比如

$("div").width(300);

示例代碼：

    <style>

      div {

        width: 100px;

        height: 100px;

        padding: 30px;

        margin: 20px;

        border: 10px solid #ff6670;

      }

    </style>

  <body>

    <div></div>

    <script>

      $(function () {

        // 返回的都是數字

        console.log($("div").width()); //width 100

        console.log($("div").innerWidth()); // width 100 + 左右padding各 30 = 160

        console.log($("div").outerWidth()); // width 100 + 左右padding各 30 +左右border各 10 = 180

        console.log($("div").outerWidth(true)); // width 100 + 左右padding各 30 +左右border各 10 + 左右margin各 20 = 220

      });

    </script>

  </body>

#### ② jQuery位置

位置主要有三個：offset()，position()，scrollTop()/ scrollLeft()。

#### ②-① offset() 獲取或設置元素偏移

* offset()方法設置或返回被選元素相對於文檔的偏移坐標，跟父級沒有關係。
* 該方法有2個屬性：left和top。offset().top用於獲取距離文檔頂部的距離，offset().left用於獲取距離文檔左側的距離。
* 可以設置元素的偏移：offset({top: 10, left:30})

#### ②-② position() 獲取元素偏移

* position()方法用於返回被選元素相對於帶有定位的父級偏移坐標，如果父級都沒有定位，則以文檔為準。
* 只能獲取，不能設置。

    <style>

      .box1 {

        position: relative;

        width: 300px;

        height: 300px;

        margin: 100px;

        background-color: #ff6670;

      }

      .box2 {

        position: absolute;

        top: 10px;

        left: 10px;

        width: 50px;

        height: 50px;

        background-color: #ffd000;

      }

      .box3 {

        width: 50px;

        height: 50px;

        background-color: #678f8d;

      }

    </style>

  <body>

    <div class="box1">

      <div class="box2"></div>

      <div class="box3"></div>

    </div>

    <script>

      $(function () {

        // 獲取元素到文檔的位置

        console.log($(".box2").offset()); //{top: 110, left: 118} body有8px的左右邊距

        // 修改元素到文檔的位置

        $(".box3").offset({ top: 300, left: 200 });

        // 獲取元素到帶有定位的父元素的位置

        console.log($(".box2").position()); //{top: 10, left: 10}

      });

    </script>

  </body>

#### ②-③ scrollTop()/ scrollLeft() 設置或獲取元素被卷去的頭部和左側

示例見下列案例代碼

#### ③ 案例：向下拉動滾動條，顯示返回頂部按鈕；點擊返回頂部按鈕，有動畫地返回頂部

* 使用animate動畫返回頂部。
* animate動畫函數裡有個scrollTop屬性，可以設置位置。
* 元素做動畫，因此$(“body,html”).animate({scrollTop:0})

#### ④ 案例：电梯导航

品优购的电梯导航：一开始看不见，页面滚动到今日推荐的位置，电梯导航才会显示出来。然后点击相应的模块（比如手机通讯），页面就滚动到手机通讯。

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Document</title>

    <script src="./jquery.mini.js"></script>

    <style>

      ul,

      li {

        margin: 0;

        padding: 0;

        list-style-type: none;

      }

      .top-nav {

        width: 100%;

        height: 200px;

        margin-top: 10px;

        background-color: #ff6670;

      }

      .banner {

        width: 100%;

        height: 300px;

        margin-top: 10px;

        background-color: #678f8d;

      }

      .main {

        width: 100%;

        height: 500px;

        margin-top: 10px;

      }

      .c1 {

        background-color: #ffd000;

      }

      .c2 {

        background-color: #ff6670;

      }

      .c3 {

        background-color: #678f8d;

      }

      .c4 {

        background-color: #ffd000;

      }

      .c5 {

        background-color: #ff6670;

      }

      footer {

        width: 100%;

        height: 600px;

        margin-top: 10px;

        background-color: #678f8d;

      }

      .elevator {

        position: fixed;

        display: none;

        top: 100px;

        left: 0;

        background-color: azure;

        text-align: center;

      }

      .elevator li {

        width: 50px;

      }

      .elevator li:hover {

        cursor: pointer;

      }

      .current {

        background-color: #ccc;

      }

    </style>

  </head>

  <body>

    <div class="top-nav">top-nav</div>

    <div class="banner">banner</div>

    <div class="main c1">宫</div>

    <div class="main c2">商</div>

    <div class="main c3">角</div>

    <div class="main c4">徵</div>

    <div class="main c5">羽</div>

    <footer>footer</footer>

    <div class="elevator">

      <ul>

        <li class="current">宮</li>

        <li>商</li>

        <li>角</li>

        <li>徵</li>

        <li>羽</li>

      </ul>

    </div>

    <script>

      $(function () {

        // 總共7-4步

        // 7 bug：每次點擊電梯按鈕，比如從5樓到2樓，會觸發滾動事件，在滾動中，會依次把4樓和3樓的按鈕也點亮。解決方案：點擊電梯按鈕時，不需要執行滾動事件中的點亮電梯按鈕代碼。可以用節流閥（或者叫互斥鎖）。

        var flag = true; //7-1

        // 4 刷新頁面的時候，即便頁面滾動距離超過了banner距離頁面上部的距離，電梯按鈕也不會出現，所以這個時候要執行一下第2步，可以把第2步封裝成函數

        function elevate() {

          if ($(document).scrollTop() >= $(".banner").offset().top) {

            // $(".elevator").show();或者

            $(".elevator").fadeIn();

          } else {

            // $(".elevator").hide(); 或者

            $(".elevator").fadeOut();

          }

        }

        elevate();

        // 1 頁面滾動事件

        $(window).scroll(function () {

          // 2 如果滾動的距離>=頁面距離頂端的距離，則顯示電梯按鈕；否則隱藏

          elevate();

          // 6 頁面滾動到相應區域，把相應區域的電梯按鈕加為current。

          // 需要用到each，遍歷電梯樓層。each能拿到電梯樓層的每一個模塊和索引號

          // 判斷條件：被卷去的頭部>=內容區裡面每個模塊的offset().top

          // 7-2 這是第7步裡的節流閥，只有flag為true時，才執行後續的代碼

          if (flag) {

            $(".main").each(function (*index*, *domEle*) {

              if ($(document).scrollTop() >= $(domEle).offset().top) {

                $(".elevator li")

                  .eq(index)

                  .addClass("current")

                  .siblings()

                  .removeClass("current");

              }

            });

          }

        });

        // 3 點擊相關的電梯按鈕，跳到相關的電梯層

        $(".elevator li").click(function () {

          // 7-3 每次點擊電梯按鈕，讓flag為false，這樣就不會執行滾動事件裡的點亮電梯按鈕的代碼了

          flag = false;

          // 拿到被點擊的電梯樓層號。

          i = $(this).index();

          // document被卷去的頭部=相應樓層的offset().top

          var dScrollTop = $(".main").eq(i).offset().top;

          // 頁面卷動

          // 7-4 在動畫結束時，把flag改為true，以便等會滾動頁面的時候，可以點亮電梯按鈕

          $("body,html")

            .stop()

            .animate(

              {

                scrollTop: dScrollTop,

              },

              function () {

                flag = true;

              }

            );

          // 5 把相應的電梯按鈕變成current，其它的電梯按鈕去掉current。鏈式編程

          $(this).addClass("current").siblings().removeClass("current");

        });

      });

    </script>

  </body>

</html>

## 5 jQuery事件

### （1）jQuery事件註冊

單個事件註冊

element.事件(function(){})

和原生基本一致

例子：

$(“div”).click(function(){事件處理程序})

其它常見的事件：mouseover，mouseout，blur，focus，change，keydown，keyup，resize，scroll等。

### （2）jQuery事件處理

on()方法在匹配元素上綁定一個或多個事件的事件處理函數。

on()方法優勢1：可以綁定多個事件。

#### on方法語法①：

element.on({

event1: fn1,

event2: fn2,

})

* on裡面的參數是個對象，鍵值對是事件和對應的函數。

示例代碼：

    <script>

      $(function () {

        $("div").on({

          mouseenter: function () {

            $(this).css("background", "#ffd000");

          },

          click: function () {

            $(this).css("background", "#678f8d");

          },

        });

      });

    </script>

#### on方法語法②：

element.on(events,[selector],fn)

* events：一個或多個用空格分隔的事件類型，如“click”或“keydown”。
* selector：元素的子元素選擇器。
* fn：回調函數即綁定在元素身上的偵聽函數。

如果2個事件觸發同一個函數：

    <style>

      div {

        width: 100%;

        height: 200px;

        background-color: #ff6670;

      }

      .current {

        background-color: #ffd000;

      }

    </style>

  <body>

    <div></div>

    <script>

      $(function () {

        $("div").on("mouseenter mouseleave", function () {

          $(this).toggleClass("current");

        });

      });

    </script>

  </body>

#### ③ on方法优势2：事件委派

可以事件委派操作：把原來加給子元素的事件綁定在父元素身上，這就是把事件委派給父元素。

    <script>

      $(function () {

        // 只給ul綁定了事件，但是觸發對象是子元素li

        $("ul").on("click", "li", function () {

          console.log("hello world");

        });

      });

    </script>

在此之前有bind()，live()，delegate()等方法來處理事件綁定或者事件委派，最新版本的用on代替它們。

#### ④ on方法优势3：給動態創建的元素綁定事件

動態創建的元素，click()沒法綁定事件，on()可以給動態生成的元素綁定事件。

示例代碼;

  <body>

    <ul>

      <li>1</li>

      <li>2</li>

      <li>3</li>

    </ul>

    <script>

      $(function () {

        // 先綁定點擊事件，後創建li，也能給新創建的li綁定點擊事件

        $("ul").on("click", "li", function () {

          console.log("hello world");

        });

        var li = $("<li>添加的li</li>");

        $("ul").append(li);

      });

    </script>

  </body>

#### ⑤ 案例：發佈微博/評論

    <style>

      li {

        display: none;

      }

    </style>

  <body>

    <div class="mic-blogs-wrap">

      發佈微博

      <textarea name="" id="" cols="30" rows="10"></textarea>

      <button>發佈</button>

      <ul class="mic-blogs"></ul>

    </div>

    <script>

      $(function () {

        $("button").on("click", function () {

          // var li = $(

          //   "<li>" +

          //     $("textarea").val() +

          //     "<a href='javascript:;'>删除</a></li>"

          // );

          // 上面的代碼可以改為

          var li = $("<li></li>");

          li.html($("textarea").val() + "<a href='javascript:;''>删除</a>");

          $(".mic-blogs").prepend(li);

          li.slideDown(); //要這個起作用，必須先給li設置display:none;

          $("textarea").val("");

        });

        $(".mic-blogs").on("click", "a", function () {

          // $(this).parent().remove(); //這行代碼沒有動畫效果

          // 有動畫效果的

          $(this) //這個$(this)指a

            .parent()

            .slideUp(function () {

              $(this).remove(); //這個$(this)指li

            });

        });

      });

    </script>

  </body>

#### ⑥ off()解綁事件和one()

    <style>

      div {

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <div></div>

    <ul>

      <li>1</li>

      <li>2</li>

      <li>3</li>

    </ul>

    <span>且</span>

    <script>

      $(function () {

        $("div").on({

          click: function () {

            console.log("點擊了");

          },

          mouseover: function () {

            console.log("鼠標經過了");

          },

        });

        $("ul").on("click", "li", function () {

          console.log("li");

        });

        // 解綁事件

        // 1 如果off括號裡為空，會把$("div")的所有事件都解綁

        $("div").off();

        // 2 要解除哪個事件，就把事件寫到off的括號裡

        $("div").off("click");

        // 3 解除事件委託

        $("ul").off("click", "li");

        // 4 有的事件只想觸發一次，可以用one()來綁定事件

        $("span").one("click", function () {

          console.log("span");

        });

      });

    </script>

  </body>

#### ⑦ 自動觸發事件：trigger()

比如輪播圖的自動播放功能，是設定定時器，自動觸發點擊右邊的箭頭。

    <style>

      div {

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <div></div>

    <script>

      $(function () {

        // 給div綁定了事件

        $("div").on({

          click: function () {

            console.log("點擊了");

          },

          mouseover: function () {

            console.log("鼠標經過了");

          },

        });

        // 自動觸發事件有3種方法

        // 1 元素.事件()。

        $("div").click(); //只要一刷新頁面，就會自動調用click

        // 2 element.trigger("事件")

        $("div").trigger("mouseover");

        // 3 element.triggerHandler("事件")。特點，不會觸發元素的默認行為。默認行為，比如input輸入框獲得焦點時（focus事件），會有光標在閃爍。

        $("div").triggerHandler("click");

      });

    </script>

  </body>

### （3）jQuery事件對象

element.on(events,[selector],function(event){})

阻止默認行為：event.preventDefault() 或者return false

阻止冒泡：event.stopPropagation()

  <body>

    <div></div>

    <script>

      $(function () {

        $(document).on("click", function () {

          console.log("點擊了document");

        });

        $("div").on("click", function () {

          console.log("點擊了div");

          event.stopPropagation(); //如果不寫這句話，點擊div，document的點擊事件也會被觸發

        });

      });

    </script>

  </body>

## 6 jQuery拷貝對象

### （1）語法

$.extend([deep], target, object1, [objectN])

* deep如果設為true為深拷貝，默認為false淺拷貝
  + 淺拷貝是把被拷貝的對象複雜數據類型中的地址拷貝給目標對象，修改目標對象會影響被拷貝對象。（見下面圖的分析）
  + 深拷貝，[deep]位置的參數寫true，完全克隆拷貝對象，而不是地址，修改目標對象不會影響被拷貝對象。
* target 要拷貝的目標對象
* object1：待拷貝到target的對象1

示例代碼：

  <body>

    <div></div>

    <script>

      $(function () {

        var targetObj = {};

        var targetObj2 = { id: 2 };

        var targetObj3 = { uid: 3 };

        var obj = {

          id: 1,

          name: "andy",

        };

        $.extend(targetObj, obj);

        console.log(targetObj); //{id: 1, name: "andy"}

        $.extend(targetObj2, obj);

        console.log(targetObj2); //{id: 1, name: "andy"} 如果有相同的鍵，會用新的值覆蓋原來的值

        $.extend(targetObj3, obj);

        console.log(targetObj3); //{uid: 3, id: 1, name: "andy"} 如果鍵不同，在原來的數據基礎上增加

      });

    </script>

  </body>

### （2）淺拷貝



    <script>

      $(function () {

        var targetObj = { id: 2 };

        var obj = {

          id: 1,

          name: "andy",

          msg: {

            age: 18,

          },

        };

        $.extend(targetObj, obj);

        targetObj.msg.age = 20;

        console.log(obj.msg.age); //20

      });

    </script>

再舉個例子：

    <script>

      $(function () {

        var targetObj = {

          id: 2,

          msg: {

            gender: "m",

          },

        };

        var obj = {

          id: 1,

          name: "andy",

          msg: {

            age: 18,

          },

        };

        $.extend(targetObj, obj);

        console.log(targetObj.msg); //{age: 18} 把targetObj.msg的內容完全覆蓋了

      });

    </script>

### （3）深拷貝



## 7 多庫共存

jQuery使用$作為標識符，隨著jQuery的流行，其它js庫也會用$作為標識符，這樣一起使用會引起衝突。

需要一個解決方案，讓jQuery和其它的js庫不存在衝突，可以同時存在，這叫多庫共存。

jQuery解決方案：

① 把裡面的$符號統一改為jQuery，比如jQuery(“div”)

② 讓jQuery釋放對$的控制權，讓用戶自己定義。比如：

        var mjq = $.noConflict();

        mjq("div"); //效果和$("div")一樣

      });

## 8 jQuery插件

jQuery功能比較有限，想要更複雜的特效，可以藉助於jQuery插件。

注意：這些插件依賴jQuery，所以得先引入jQuery文件。

jQuery插件常用網站：

jQuery插件庫：<http://www.jq22.com>

jQuery 之家（導師推薦這個）：<http://www.htmleaf.com>

### （1）瀑布流插件

1. 在<http://www.htmleaf.com>中，選擇jQuery庫裡面的瀑布流插件
2. 打開瀑布流插件列表後，選擇“兼容IE8的响应式网格瀑布流布局jQuery插件”
3. 下載插件
4. 根據頁面指引使用插件 or （最好參考demo。因為頁面指引裡的代碼可能過時；而demo能正常展示，就說明代碼能用）直接參考demo裡面的代碼
5. 使用插件分2步：
   * 引入js和css文件
   * 拷貝HTML、css和js代碼

### （2）圖片懶加載

圖片使用延遲加載可提高網頁加載速度，也能減輕服務器負載。

懶加載即頁面滑動到可視區域時，再顯示圖片。

使用jQuery懶加載插件可以實現該功能。

步驟：

1. 在<http://www.jq22.com>輸入“懶加載”，在搜索結果頁找到“EasyLazyload.js”
2. 下載
3. 按照demo和頁面說明操作

注意：

1. 将图片 src 替换为 data-lazy-src <img data-lazy-src="..." />

因為頁面中的圖片太多了，在visual studio code裡，按ctrl+h，查找<img src=，替換為<img data-lazy-src=，然後點擊Replace輸入欄的最後一個按鈕“Replace All”

1. 引入的js文件和js調用必須寫到DOM元素最後面，如demo所示。
2. 導師把js中下面4行代碼刪除了：

        coverColor: "white",

        coverDiv: "<h1>test</h1>",

        offsetBottom: 0,

        offsetTopm: 0

### （3）全屏滾動

常用的：fullpage.js，用於製作全屏網站

GitHub：<https://github.com/alvarotrigo/fullPage.js>

中文翻譯網站：<http://www.dowebok.com/demo/2014/77/>

### （4）Bootstrap組件

以v3版本為例，先看網站最上面的導航條。

1. getting started是寫好HTML框架代碼，並引入相關js和css文件。
2. css是使用Bootstrap組件時，先要寫的相關代碼。
3. components是各種組件
4. JavaScript是各種組件的交互效果
   * 比如Modal（點擊按鈕彈出對話框），調用JS效果，看usage部分。有2種方法：Via data attributes和Via JavaScript

## 9 案例：ToDoList

### （1）需求

1. 在ToDoList輸入框裡輸入內容，回車
2. 添加到正在進行的list中
3. 數據用localStorage存儲，這樣刷新頁面也不會丟失數據
4. 在正在進行的list的事件前面的複選框打鉤，該事件放入已經完成的list中
5. 在已完成的list中，取消事件複選框的打鉤，該事件放入正在進行的list中
6. 兩個list都能統計總共幾件事
7. 按刪除按鈕，刪除當前事項

### （2）注意

1. 不論按下回車，還是點擊複選框，都是把本地存儲的數據加載到頁面中，這樣保證刷新關閉頁面不會丟失數據
2. 存儲數據的格式：var todolist = [{title: ‘xxx’, done:false}, {title: ‘xxx’, done:false}]
3. 本地存儲只能存儲字符串的數據格式，把數組對象轉換為字符串格式 JSON.stringify()
4. 獲取本地存儲的數據，要把字符串轉換為數組對象格式JSON.parse()

### （3）步驟要點

1. 按下回車鍵讀取和保存localStorage
   * 利用事件對象.keyCode判斷用戶按下回車鍵（13）
   * 聲明一個數組，保持數據
   * 先讀取本地存儲的數據（聲明函數getData()），放到這個數組裡。
   * 把最新從表單獲取過來的數據，追加到數組裡。
   * 把數組存儲給本地存儲（聲明savaData()）。
2. 本地存儲數據渲染加載到頁面
   * 會經常渲染加載操作，所以聲明一個函數load，方便調用
   * 先要讀取本地存儲數據（記得把數據轉換為對象格式）
   * 之後遍歷這個數據（$.each()），有幾條數據，就生成幾個li
   * 每次渲染前，先把list中已渲染的內容清空，然後再渲染最新數據。
3. 刪除
   * 點擊a，不是刪除li，而是刪除localStorage對應的數據。只要localStorage裡的數據刪除了，li自然就不會被渲染了。
   * 核心原理：先獲取localStorage，刪除對應數據，保存localStorage，重新渲染li
   * 給鏈接自定義屬性，記錄當前的索引號。根據這個索引號刪除相關的數據。
   * 刪除數組用splice(i, n)方法。i是從第幾個元素開始刪除，n是刪除幾個元素
     + 比如要刪掉數組var arr = ['a’, ‘b’, ‘c’]裡面的’b’，arr.splice(1,1)
   * 因為a是動態創建的，使用on方法綁定事件
4. 事項放在doing或者done的list中
   * 點擊複選框獲取localStorage
   * 修改本地存儲數據：修改數據屬性done的值為true
   * 保存修改後的數據到localStorage
   * 重新渲染列表
     + load加載函數裡面，新增一個條件，如果當前數據的done為true，就是完成了的，就把列表渲染加載到done這個list裡。
     + 如果當前數據的done屬性為false，則是待辦事項，就將其渲染到doing list中
5. toDoList統計doing和done各自的事件數量
   * 在load函數裡面操作
   * 聲明2個變量：doingCount待辦個數，doneCount完成個數
   * 遍歷本地存儲數據的時候，如果屬性done的值為false，則doingCount++，否則doneCount++
   * 修改相應元素的text()

### （4）代碼

  <body>

    <div class="todo-wrap">

      請輸入待辦事項，回車添加待辦事項：<input type="text" id="todo" />

      <br />

      正在進行<span id="doing-count"></span>

      <ul id="doing"></ul>

      已經完成<span id="done-count"></span>

      <ul id="done"></ul>

    </div>

    <script>

      $(function () {

        // 總共4步

        // 1-7 每次頁面打開時，都從localStorage中讀取數據，把數據渲染到list中

        load();

        // 1(共7小條) 給todo輸入框綁定鍵盤抬起事件

        $("#todo").on("keyup", function (*e*) {

          // 1-1 如果抬起的鍵盤是13，則獲取輸入框裡的內容，存儲在本地，並且在doing list中顯示

          if (e.keyCode === 13) {

            if ($(this).val() === "") {

              alert("輸入不能為空");

            } else {

              // 1-2 讀取本地存儲原來的數據（因為這個功能經常用到，所以寫個函數）

              var local = getData();

              // 1-3 更新local數組，把最新的數據追加給local數組

              local.push({ title: $(this).val(), done: false });

              // 1-4 把local存儲給localStorage（因為這個功能經常用到，所以寫個函數）

              saveData(local);

              // 1-5 渲染加載數據

              load();

              // 1-6 清空輸入框

              $(this).val("");

            }

          }

        });

        // 1-2

        function getData() {

          var data = localStorage.getItem("todolist");

          if (data !== null) {

            // 本地存儲裡面的數據是字符串格式的，但我們需要的是對象格式的，所以要轉換

            return JSON.parse(data);

          } else {

            return [];

          }

        }

        //  1-4

        function saveData(*data*) {

          localStorage.setItem("todolist", JSON.stringify(data));

        }

        // 1-5

        function load() {

          var data = getData();

          $("#doing, #done").empty();

          // 4 統計doing和done的事件個數

          // 4-1 設置2個list統計數值的初始值

          var doingCount = 0;

          var doneCount = 0;

          $.each(data, function (*i*, *n*) {

            // 2-2-1 給a加個id屬性，以備刪除的時候用

            // 3-4-1 根據done的屬性來確定酒精把li渲染到哪個列表

            if (n.done) {

              $("#done").prepend(

                "<li><input type='checkbox' checked='checked'/><span>" +

                  n.title +

                  "</span><a href='javascript:;'' id=" +

                  i +

                  ">刪除</a></li>"

              );

              doneCount++;

            } else {

              $("#doing").prepend(

                "<li><input type='checkbox' /><span>" +

                  n.title +

                  "</span><a href='javascript:;'' id=" +

                  i +

                  ">刪除</a></li>"

              );

              doingCount++;

            }

          });

          // 4-2 修改統計數值

          $("#doing-count").text(doingCount);

          $("#done-count").text(doneCount);

        }

        // 2（共4小條） doToList 刪除操作

        $("#doing, #done").on("click", "a", function () {

          // 2-1 獲取本地存儲

          var data = getData();

          // 2-2 修改數據

          // 2-2-2 點了哪個a，就獲取哪個a的id

          var id = $(this).attr("id");

          data.splice(id, 1);

          // 2-3 保存到本地存儲

          saveData(data);

          // 2-4 重新渲染頁面

          load();

        });

        // 3 切換doing和done的狀態

        $("#doing, #done").on("click", "input", function () {

          // 3-1 獲取本地存儲的數據

          var data = getData();

          // 3-2 根據複選框的狀態修改數據

          // 點擊CheckBox，拿到它sibling a 的id值

          var id = $(this).siblings("a").attr("id");

          data[id].done = $(this).prop("checked");

          // 3-3 保存到本地存儲

          saveData(data);

          // 3-4 重新渲染頁面

          load();

        });

      });

    </script>

  </body>

Echarts數據可視化項目

## 1 什麼是數據可視化

### （1） 目的

數據可視化目的：藉助於圖形化手段，清晰有效地傳達與溝通信息。

數據可視化可以把數據從冰冷的數字轉換成圖形，揭示蘊含在數據中的規律和道理。

### （2） 使用場景

通用報表

移動端圖表

大屏可視化

圖編輯&圖分析

地理可視化

### （3） 常見的數據可視化庫

D3.js 目前Web端評價最高的JavaScript可視化工具庫（入手難）。

Echarts.js 百度出品的一個開源JavaScript數據可視化庫。

Highcharts.js 國外的前端數據可視化庫，非商用免費，被許多國外大公司所使用。

AntV螞蟻金服全新一代數據可視化解決方案。

Highcharts和ECharts就像office和WPS的關係。

## 2 數據可視化項目概述

### （1） 項目展示

### （2） 項目目的

市場需求：應對現在數據可視化的趨勢，越來越多企業需要在很多場景（營銷數據、生產數據、用戶數據）下使用，可視化圖表來展示體現數據，讓數據更加直觀，數據特點更加突出。

學習階段需求：

這個項目對前端學習者來說，起著承上啟下的作用。

承上：複習以前學習的內容：HTML5+CSS3佈局相關技術，JavaScript、jQuery相關技術

啟下：為學習服務器編程做鋪墊。如何把服務器裡面的數據渲染到頁面中？

### （3） 項目技術

HTML5+CSS3佈局

CSS3動畫、漸變

jQuery庫，原生JavaScript

flex佈局和rem適配方案

圖片邊框border-image

ES6模板字符

ECharts可視化庫等

## 3 ECharts簡介

ECharts是一個使用JavaScript實現的開源可視化庫，可以流暢地運行在PC和移動設備上，兼容當前絕大部分瀏覽器（IE8/9/10/11，Chrome，Firefox，Safari等），底層依賴矢量圖形庫ZRender，提供直觀、交互豐富、高度個性化定制的數據可視化圖表。

官網地址：<https://echarts.apache.org/zh/index.html>

特點：

* 豐富的可視化類型
* 多重數據支持格式
* 支持流數據
* 移動端優化
* 跨平台使用
* 絢麗的特效
* 詳細的文檔說明

## 4 ECharts基本使用

### （1）基本步驟

步驟1：下載並引入echarts.js文件

在下載頁面通過所需版本的dist連接進入相關GitHub頁面，找到echarts.js或者echarts.mini.js（壓縮版本）

步驟2：準備一個設置了寬高的DOM容器

步驟3：初始化echarts實例對象

通過echarts.init方法，官網有詳細代碼。

步驟4：指定配置項和數據

步驟5：將配置項設置給echarts實例對象

### （2）選擇不同類型圖表

官網-示例。

選擇所需的示例，拷貝相關代碼。

### （3）配置ECharts

* title：標題組件
* tooltip：提示框組件
* legend：圖例組件
* toolbox：工具欄
* grid：直角坐標系內繪圖網格
  + grid: {
    - left: '3%', （y軸距離包裹圖表的盒子的左邊框的距離）
    - right: '4%', （和y軸平行的最右邊的線，距離包裹圖表的盒子的右邊框的距離）
    - bottom: '3%', （x軸距離包裹圖表的盒子的下邊框的距離）
    - containLabel: true （值為true，顯示y軸的刻度，防止刻度標籤溢出（刻度標籤溢出即left值很小時，y軸刻度值跑到包裹圖表的盒子外面））。當left為0%時，如果containLabel: false，則刻度標籤不會顯示。
  + },
* xAxis：直角坐標系grid中的x軸
* yAxis：指教坐標系grid中的y軸
* series：系列列表。每個系列通過type決定自己的圖表類型（什麼類型的圖標）
  + type：類型。什麼類型的圖表，比如line折線，bar柱形。
  + name：系列名稱，用於tooltip的顯示，legend的圖例篩選。
  + stack：數據堆疊。如果設置相同值，會數據堆疊。
    - 數據堆疊：第2個數據值=第1個數據值+第2個數據值

第3個數據值=第2個數據值+第3個數據值，依次疊加。

* + - 如果給stack指定不同值或者去掉這個屬性，不會發生數據堆疊。



* color：調色盤顏色列表

上述為主要配置，其它配置在官網-文檔-配置項手冊

## 5 數據可視化項目開發

### （1）數據可視化項目適配方案

設計稿是1920px

PC端適配：寬度在1024-1920之間頁面元素寬高自適應

flexible.js可以檢測瀏覽器寬度，修改HTML文字大小。

rem單位。頁面元素根據rem適配大小，配合cssrem插件。

flex佈局。頁面快速佈局。

flexible.js把屏幕分成24等份。

PC端的效果圖是1920px。

cssrem插件的基準值是80px。（把visual studio code裡面的cssrem默認的Foot Font Size從16修改到80，然後重啟visual studio code）

rem值自動生成。

把屏幕寬度約束在1024-1920之間。小於1024和大於1920都不再縮放。

@media screen and (max-width: 1024px) {

  html {

    /\* 42.66=1024除以24份 \*/

    font-size: 42.66px !important;

  }

}

@media screen and (min-width: 1920px) {

  html {

    font-size: 80px !important;

  }

}

### （2）數據可視化項目開發

#### ① 搭建結構



引入文件後，在body裡寫個字，如果其font-size:12px，說明引入成功，因為flexible把body的font-size設置為12px

如果想要在visual studio code裡保存了以後自動刷新瀏覽器，可以安裝插件Preview on Web Server。安裝好以後，選擇vscode-preview-server: Launch on browser。

#### ② 初始化樣式

/\* 清除元素默认的外边距 \*/

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

/\* 让所有斜体不倾斜 \*/

em,

i {

  font-style: normal;

}

/\* 去掉列表前面的小点 \*/

li {

  list-style: none;

}

/\* 去掉图片边框，去掉图片底侧的空白缝隙 \*/

img {

  border: 0; /\*ie6\*/

  vertical-align: middle;

}

/\* 让button按钮变成小手 \*/

button {

  cursor: pointer;

}

/\* 取消连接的下划线 \*/

a {

  color: #666;

  text-decoration: none;

}

a:hover {

  color: #e33333;

}

#### ③ 設置body背景

#### ④ 項目基礎佈局

設置一個大盒子（viewport）包裹所有項目。大盒子裡面分成3列。

把logo作為viewport的背景。把logo的最小高度設置為780px（因為窗口最小寬度為1024px，而1024px寬通常和配780px高）。



#### ⑤ 邊框圖片

每個模塊的邊框是一樣的，可以統一做。

1. **邊框圖片的使用場景**

這些盒子的邊框都一樣，但是大小不同。無法通過以往學過的CSS樣式實現，得通過邊框圖片技術實現。

為了實現豐富多彩的邊框效果，CSS3中新增了border-image屬性，這個新屬性允許指定一幅圖像作為元素的邊框。

1. **邊框圖片的切圖原理**

把九宮格的四個角切出去，中間部分可以鋪排、拉伸或者環繞。



1. **邊框圖片語法規範**

|  |  |
| --- | --- |
| 屬性 | 描述 |
| border-image-source | 用在邊框的圖片的路徑。（哪個圖片？） |
| border-image-slice | 圖片邊框向內偏移。（裁剪的尺寸，不加單位，上右下左順序） |
| border-image-width | 圖片邊框的寬度（需要加單位）。給圖片邊框加減寬度，不影響盒子內容的位置。這個屬性如果不寫，默認和盒子邊框一樣寬。 |
| border-image-repeat | 圖片邊框是否平鋪（repeat）、鋪滿（round）、或拉伸（stretch）默認拉伸 |

範例：

    <style>

      div {

        width: 500px;

        height: 500px;

        border: 20px solid yellow;

        border-image-source: url(*./border.jpg*);

        border-image-slice: 165;

        /\* 上面一行代码相当于border-image-slice: 165 165 165 165; \*/

        border-image-repeat: round;

      }

    </style>

#### ⑤ 公共模塊樣式（邊框圖片和背景）



#### ⑥ 通過類名調用字體圖標

在HTML頁面引入字體圖標的css文件icomoon.css。然後直接引用對應的字體圖標類名。

引入css文件和聲明字體圖標的時候，一定要注意路徑。

#### ⑦ 概覽區域模塊製作

#### ⑧ 監控區域模塊製作

JS文件中，會有大量的變量命名，特別是ECharts使用中，需要大量初始化ECharts對象。

為了防止變量命名衝突（變量污染），採取立即執行函數策略：

(function(){})();

例子：

(function(){

var num = 10;

})();

(function(){})(

var num = 10;

);

注意：多个立即执行函数中间必须加分号隔开。

監控區域用到的JS效果：

1. 點擊切換tab
2. 克隆marquee裡面的row

#### ⑨ 點位分佈統計區域模塊製作

需要一個南丁格爾玫瑰圖，需要tooltip，color，series。

要定制圖表，必須先弄清每項參數的含義。

tooltip和series的參數含義：

option = {

    // 提示框组件

    tooltip: {

      // trigger 触发方式。  非轴图形，使用item的意思是放到数据对应图形上触发提示

      trigger: 'item',

      // 格式化提示内容：

      // a 代表series系列图表名称

      // b 代表series数据名称 data 里面的name

      // c 代表series数据值 data 里面的value

      // d代表  当前数据/总数据的比例

      formatter: "{a} <br/>{b} : {c} ({d}%)"

    },

    // 控制图表

    series: [

      {

        // 图表名称

        name: '点位统计',

        // 图表类型

        type: 'pie',

        // 南丁格尔玫瑰图 有两个圆  内圆半径10%  外圆半径70%

        // 饼形图半径。 可以是像素。也可以是百分比（ 基于DOM容器大小）第一项是内半径，第二项是外半径（通过它可以实现饼形图大小）。如果是百分比，必须加引号。非百分比的数字，直接写即可，不用加引号。

        radius: ['10%', '70%'],

        // 图表中心位置 left 50%  top 50%  距离图表DOM容器

        center: ['50%', '50%'],

        // radius 半径模式，另外一种是 area 面积模式

        roseType: 'radius',

        // 数据集 value 数据的值 name 数据的名称

        data: [

                {value:10, name:'rose1'},

                {value:5, name:'rose2'},

                {value:15, name:'rose3'},

                {value:25, name:'rose4'},

                {value:20, name:'rose5'},

                {value:35, name:'rose6'},

                {value:30, name:'rose7'},

                {value:40, name:'rose8'}

            ]

        }

    ]

};

新版和舊版的echarts.min.js不同，其中一個體現就是南丁格爾玫瑰圖的每一個花瓣的顏色及對應的字的顏色不同。

監聽瀏覽器縮放，圖表對象調用縮放resize函數：

  // (4) 当浏览器缩放的时候，图标等比例缩放

  window.addEventListener("resize", function () {

    myChart.resize();

  })

#### ⑩ 地圖區域製作

#### ⑪ 柱形圖 用戶統計

單獨修改某個柱形樣式

核心原理：series對象裡面的data數組，裡面的每一個數據影響每一個柱形。

data數組裡的元素，除了直接寫數字，還可以是對象。

#### ⑫ 銷售統計模塊

切換tab原理：

series裏面的data數據決定折綫的顯示

點擊不同的tab標簽，就讓series裏面的data調用不同的數據即可。

實現步驟：

① 準備切換時要用到的數據4組

② 綁定點擊事件

* 切換激活tab的樣式
* 切換圖表依賴的數據（重新渲染圖表）

③ 開啓定時器，進行切換，鼠標經過sales停止定時器，鼠標離開sales開啓定時器

#### ⑬ 一季度銷售進度

這裡有個半環形。做法：在ECharts中找個環形圖（doughnut chart），將其分為三段，2段平分上面半個環，一段是下半個環。把下半個環設為透明即可。

關於doughnut chart的startAngle



#### ⑭ 全國熱榜模塊

**技能點a) 把後台返回的真實數據渲染到頁面中**

* 準備後台返回的真實數據
* 利用數據渲染各省熱銷模塊sup模塊（拼接html格式字符串，進行渲染）
* 當數據進入tab的時候
  + 激活當前的tab樣式，刪除其它tab的樣式
  + 渲染各省熱銷sub模塊（拼接html格式字符串，進行渲染）
* 默認激活第一個tab的效果
* 開啟定時器，依次切換

**技能點b) ES6模板字符相關知識**

語法規範：

`${表達式}`

`是半角輸入法下數字1左邊那個鍵。

模板字符串允許自由換行。

以後會詳細講解ES6模板字符串，此處主要用於拼接字符串。

示例代碼：

    <script>

      var ci = {

        id: 1,

        name: "临江仙",

      };

      // 以前的写法，拼接的时候引号容易出问题

      console.log("id：" + ci.id + ",词牌名：" + ci.name);

      // ES6 模板字符写法

      console.log(`id：${ci.id}，词牌名：${ci.name}`);

    </script>

**注意**

這個功能裡，定時器裡不要用

        // lis.eq(index).mouseenter();

否則會和

 $('.province .sup').hover(

衝突。

因為hover就是mouseenter和mouseleave。

如果把下面注釋掉的代碼替換renderSupSubHTML()函數，會出錯。因為hover的第2個參數函數為mouseleave→激發lis.eq(index).mouseenter()→激發hover的第1個參數函數mouseenter→clearInterval(timer);→無法自動跳轉和渲染數據。

 // (5) 开启定时器

  var index = 0;

  var timer = setInterval(function () {

    index++;

    if (index >= 5) index = 0;

    // lis.eq(index).mouseenter();

    renderSupSubHTML(lis.eq(index));

  }, 1000);

  // (6) 鼠标经过停止和开启定时器

  $('.province .sup').hover(

    function () {

      clearInterval(timer);

    }, function () {

      clearInterval(timer);

      timer = setInterval(function () {

        index++;

        if (index >= 5) index = 0;

        // lis.eq(index).mouseenter();

        renderSupSubHTML(lis.eq(index));

      }, 1000);

    }

  )

#### ⑮ Echarts社區改版後叫MakeAPie

Echarts把社區裡的圖獨立出來了，專門做了個網站<https://www.makeapie.com/explore.html>

實現步驟：

需要下載china.js提供中國地圖的js 文件

因為這裡代碼太多，所以新建一個js文件：myMap.js

使用Echarts社區提供的配置

需要修改：

去掉圖例組件和標題組件

去掉背景顏色

修改地圖省份背景 #142957

地圖放大通過zoom設置為1.2即可

面向對象編程

## 1 面向對象編程介紹

量大編程思想：面向過程，面向對象

簡單的程序，一般用面向過程來寫；複雜的程序，一般用面向對象來寫。

### （1）面向過程編程

POP：process-oriented programming

面向過程是分析出解決問題所需要的步驟，然後用函數把這些步驟一步一步實現，使用的時候再一個一個一次調用。

例子：

目標：把大象裝進冰箱

過程：①打開冰箱門，②把大象裝進去，③關上冰箱門

適合跟硬件聯繫緊密的東西，比如單片機採用的是面向過程編程。

### （2）面向對象編程

OOP：object-oriented programming

面向對象把實物分解成一個個對象，然後由對象之間分工合作。

例子：

目標：把大象裝進冰箱

找出對象，並寫出這些對象的功能：

① 大象對象

* 進去

② 冰箱對象

* 開門
* 關門

③ 使用大象和冰箱的功能

面向對象編程特點：每一個對象都是功能中心，具有明確分工。

面向對象編程具有靈活、代碼可複用、容易維護、容易拓展開發的優點，更適合多人合作的大型軟件項目。

面向對象編程特性：

* 封裝性
* 繼承性
* 多態性

面向對象性能比面向過程低。

————

## 2 ES6中的類和對象

類class是ES6之後新增的。

ES6，全稱ECMAScript6.0,2015.06發佈。

抽象/泛指：手機

具體/特質：iPhone

面向對象的思維特點：

① 抽取（抽象）對象公用的屬性和行為組成（封裝成）一個類

② 對類進行實例化，獲取類的對象

### （1）對象

現實生活中：萬物皆對象，對象是一個具體的事務，看得見摸得著的實物。例如：一本書、一輛汽車、一個人、一個數據庫、一張網頁、一個與遠程服務器的連接。

JavaScript中，對象是一組無序的相關屬性和方法的集合，所有的事物都是對象。例如：字符串、數值、數組、函數等。

對象由屬性和方法組成：

* 屬性：事物的特征（名詞）
* 方法：事物的行為（動詞）

### （2）類

在ES6中新增了類的概念，可以使用class關鍵詞聲明一個類，之後以這個類來實例化對象。

類抽象了對象的公共部分（比如一輛寶馬車的圖紙）

對象特指某一個，通過類實例化一個具體的對象（比如一輛輛具體的寶馬車）

### （3）創建類

語法：

class Name {

// class body

}

創建實例：

var xx =new Name();

通過class關鍵詞聲明類，類名要首字母大寫

**類constructor構造函數**

constructor()方法是類的構造函數（默認方法），用於傳遞參數、返回實例對象，通過new命令生成對象實例時，自動調用該方法。如果程序員寫代碼時沒有定義constructor()，類內部會自動創建一個constructor()。

  <script>

    // 1. 創建類 class，比如明星類

    class Star {

      //一調用這個類Star，就會自動執行constructor這個函數

      // 類裡面所有的函數都無需加function

      constructor(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

      }

    }

    // 2. 利用類創建對象

    var bd = new Star("北斗星", 1110);

    console.log(bd.uname); //北斗星

    console.log(bd.id); //1110

  </script>

### （4）添加類方法

語法

class Person {  
 construction(name, age) {

this.name = name;

this.age = age;

}

say() {

console.log(this.name + ‘hi’);

}

sing(song) {

console.log(this.name + song);

}

}

類裡面的函數不用寫function關鍵字。

所有函數之間不用寫逗號來分隔。

## 3 類的繼承

程序中的繼承：子類可以繼承父類的一些屬性和方法。

### （1） 語法

class Father{ //父類

}

class Son extends Father { // 子類繼承父類

}

示例：

  <script>

    class Father {

      constructor() {}

      say() {

        console.log("hi");

      }

    }

    class Son extends Father {

      // 子類繼承父類

    }

    var son = new Son();

    son.say();

  </script>

### （2） super關鍵字

super關鍵字用於訪問和調用對象父類的函數。可以調用父類的構造函數，也可以調用父類的普通函數。

  <script>

    class Father {

      constructor(*x*, *y*) {

        this.x = x;

        this.y = y;

      }

      sum() {

        console.log(this.x + this.y);

      }

    }

    class Son extends Father {

      constructor(*x*, *y*) {

        super(x, y); //調用父類中的構造函數，把子類constructor的參數傳遞給父類constructor函數，從而使用父類constructor裡面的this

      }

    }

    var son = new Son(1, 2);

    son.sum();

  </script>

繼承中的屬性方法查找原則：如果實例化子類輸出一個方法，先看子類有沒有這個方法，有就先執行子類的。如果子類裡面沒有這個方法，就去查找父類有沒有這個方法，如果有，就執行父類的這個方法。

  <script>

    class Father {

      say() {

        return "father";

      }

    }

    class Son extends Father {

      say() {

        console.log(super.say() + "son");

      }

    }

    var son = new Son();

    son.say(); //fatherson

  </script>

子類在構造函數中使用super，必須放到this前面。必須先調用父類的構造方法，再使用子類構造方法。

  <script>

    // 父類有加法方法

    class Father {

      constructor(*x*, *y*) {

        this.x = x;

        this.y = y;

      }

      sum() {

        console.log(this.x + this.y);

      }

    }

    // 子類繼承父類的加法方法，同時有自己的減法方法

    class Son extends Father {

      constructor(*x*, *y*) {

        // 利用super調用父類的構造函數

        super(x, y);

        this.x = x;

        this.y = y;

      }

      subtract() {

        console.log(this.x - this.y);

      }

    }

    var son = new Son(5, 3);

    son.sum(); //8

    son.subtract(); //2

  </script>

### （3） 注意

ES6中沒有變量提升，所以必須先定義類，才能通過類實例化對象。

類裡面的共有屬性和方法一定要加this使用。

constructor裡面的this指向的是創建的實例對象。

類裡面的方法裡面的this，指向的是它的調用者（比如調用它的實例對象，或者按鈕）。

    <script>

      var that;

      var \_that;

      class Star {

        constructor(*uname*, *age*) {

          // constructor裡面的this指向的是創建的實例對象

          that = this;

          console.log(this);

          this.uname = uname;

          this.age = age;

          this.btn = document.querySelector("button");

          this.btn.onclick = this.sing;

        }

        sing() {

          // 這個sing方法的this指向的是btn這個按鈕，因為這個按鈕調用了這個函數

          console.log(this.uname);

          // 如果讓btn也能調用實例對象，可以在constructor構造函數中，把實例對象this保存到全局變量that中

          console.log(that.uname);

        }

        dance() {

          // 方法裡的this指向它的調用者。

          \_that = this;

          console.log(this);

        }

      }

      var bd = new Star("北斗星", 10000);

      console.log(that === bd); //true   "that === bd"

      bd.dance();

      console.log(\_that === bd); //true   "\_that === bd"

    </script>

## 4 面向對象案例

功能需求：

（1）點擊tab欄，切換。

（2）點擊+，添加tab和內容。

* 創建新的選項卡li和內容section
* 把創建的兩個元素追加到對應的父元素中
  + 以前的做法：動態創建元素createElement，但是元素裡面內容較多，需要innerHTML賦值，再appendChild追加到父元素裡面。
  + 現在更高級的做法：利用insertAdjacentHTML()可以直接把字符串合適元素添加到父元素中。
  + appendChild不支持追加字符串的子元素，insertAdjacentHTML支持追加字符串的元素。

（3）點擊x，刪除tab和內容。

* x沒有索引號，但是其上級元素li有。

（4）雙擊tab文字或者內容文字，可以修改文字內容。

* 雙擊事件：ondblclick
* 如果雙擊文字，會默認選定文字，此時需要雙擊禁止選中文字。用這樣一段代碼：
* window.getSelection ? window.getSelection().removeAllRanges() : document.selection.empty();
* 核心思路：雙擊文字的時候，在裡面生成一個文本框，當失去焦點或者按下回車，把文本框輸入的值給原先的元素。

抽取對象：Tab對象

（1）該對象有切換功能

（2）該對象有添加功能

（3）該對象有刪除功能

（4）該對象有修改功能

構造函數和原型

類class是ES6之後新增的。ES6之前是通過構造函數和原型來模擬類的實現機制。

## 1構造函數和原型講解

### （1） 導讀

在JS中，使用構造函數時要注意：

1. 構造函數用於創建某一類對象，其首字母要大寫
2. 構造函數要和new一起使用才有意義

new在執行時會做四件事情：

1. 在內存中創建一個新的空對象
2. 讓this指向這個新的對象
3. 執行構造函數裡面的代碼，給這個新對象添加屬性和方法
4. 返回這個新對象（所以構造函數裡面不需要return）

### （2） 實例成員和靜態成員

構造函數中的屬性和方法叫成員，可以添加成員。

實例成員是構造函數內部通過this添加的成員，實例成員只能通過實例化的對象來訪問，不能通過構造函數來訪問。

靜態成員是在構造函數本身上添加的成員。靜態成員只能通過構造函數訪問，不能通過對象來訪問。

    <script>

      function Star(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

        this.sing = function () {

          console.log("singing");

        };

      }

      var bd = new Star("北斗星", 89);

      // 實例成員是構造函數內部通過this添加的成員，uname, id, sing都是實例成員

      // 實例成員只能通過實例化的對象來訪問，不能通過構造函數來訪問。

      console.log(bd.uname); // 北斗星

      bd.sing(); // singing

      console.log(Star.uname); // undefined （不能通過構造函數來訪問。）

      // 靜態成員是在構造函數本身上添加的成員。

      Star.shade = "lightblue";

      console.log(Star.shade); //lightblue  (靜態成員只能通過構造函數訪問，不能通過對象來訪問。)

    </script>

### （3） 構造函數原型對象prototype

構造函數方法雖然好用，但存在浪費內存的問題。

每一次創建實例對象，都會專門給類裡面的函數在內存中開闢一個新的區域來儲存，這樣很浪費內存。

我們希望所有的對象使用同一個函數，這樣比較節省內存，該如何做呢？

構造函數通過原型分配的函數是所有對象所共享的。

JavaScript規定，每一個構造函數都有一個prototype屬性，指向另一個對象。注意這個prototype就是一個對象，這個對象的所有屬性和方法，都會被構造函數所擁有。

可以把一些不變的方法，直接定義在prototype對象上，這樣所有的對象的實例都可以共享這些方法。

一般情況下，公共屬性定義到構造函數裡面，公共方法放到原型對象身上

    <script>

      function Star(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

        // this.sing = function () {

        //   console.log("singing");

        // };

      }

      Star.prototype.sing = function () {

        console.log("singing");

      };

      var bd = new Star("北斗星", 89);

      var zw = new Star("紫薇星", 99);

      bd.sing();

      zw.sing();

    </script>

### （4） 對象原型屬性\_\_proto\_\_

對象都會有一個屬性\_\_proto\_\_指向構造函數的prototype原型對象，之所以對象可以使用構造函數prototype原型對象的屬性和方法，就是因為對象有\_\_proto\_\_原型存在。

    <script>

      function Star(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

      }

      Star.prototype.sing = function () {

        console.log("singing");

      };

      var bd = new Star("北斗星", 89);

      console.log(bd.\_\_proto\_\_ === Star.prototype); //true

    </script>

方法查找規則：

1. 首先看bd對象身上是否有sing方法，如果有就執行這個對象上的sing。
2. 如果沒有sing這個方法，因為有\_\_proto\_\_屬性的存在，就去構造函數原型對象prototype身上去查找sing這個方法。

\_\_proto\_\_屬性和原型對象prototype是完全相等的。

\_\_proto\_\_屬性的意義在於為對象的查找機制提供一個方向/路線。但它是個非標準屬性，因此實際開發中，不可以使用這個屬性，它只是內部指向原型對象prototype。

    <script>

      function Star(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

      }

      Star.prototype.sing = function () {

        console.log("singing");

      };

      var bd = new Star("北斗星", 89);

      // \_\_proto\_\_屬性和原型對象prototype是完全相等的

      console.log(bd.\_\_proto\_\_ === Star.prototype); //true

    </script>

### （5） 構造函數constructor

對象原型屬性（\_\_proto\_\_）和構造函數原型對象（prototype）裡面都有一個constructor屬性，它指回構造函數本身。

constructor主要用於記錄該對象引用了那個構造函數，它可以讓原型對象（prototype）重新指向原來的構造函數。

    <script>

      function Star(*uname*, *id*) {

        this.uname = uname;

        this.id = id;

      }

      // Star.prototype.sing = function () {

      //   console.log("singing");

      // };

      // 除了上述賦值方法外，還可以像下面這樣給原型對象添加函數。但這樣會存在一個問題，這樣直接把一個對象賦值給Star.prototype，會把Star.prototype對象原有的值，包括constructor，覆蓋掉。所以這樣賦值的時候要在最開頭把constructor寫上，並指回/賦值Star

      Star.prototype = {

        constructor: Star,

        sing: function () {

          console.log("singing");

        },

        dance: function () {

          console.log("dancing");

        },

      };

      var bd = new Star("北斗星", 89);

      console.log(bd.\_\_proto\_\_);

    </script>

### （6） 構造函數、實例、原型對象三者之間的關係



### （7） 原型鏈

只要是對象就有\_\_proto\_\_原型屬性，指向原型對象。

案例Star原型對象裡的\_\_proto\_\_原型屬性，指向Object.prototype



### （8） JavaScript的成員查找機制/規則

1. 訪問一個對象的屬性/方法時，先查找這個對象自身有沒有該屬性。
2. 如果對象自身沒有，就查找它的原型屬性\_\_proto\_\_指向的prototype原型對象。
3. 如果還沒有，就查找原型對象的原型：Object.prototype。
4. 還沒有，就是null，返回的結果是undefined。

如果對象自己有屬性id，類的原型對象上也有id，根據原型鏈的查找思路的就近原則，使用的是對象自己的屬性。

bd.id = 23;

Star.prtotype.id=56;

console.log(bd.id); //23

### （9） 原型對象this指向

1. 在構造函數中，this指向的是對象實例。
2. 原型對象裡面的函數裡面的this，指向的是對象實例。

### （10） 擴展內置對象

可以通過原型對象，對原來的內置對象進行自定義，拓展起方法屬性。

例子：給數組增加自定義求偶數和的功能。

    <script>

      Array.prototype.sum = function () {

        var sum = 0;

        for (let i = 0; i < this.length; i++) {

          sum += this[i];

        }

        return sum;

      };

      var arr = [2, 3, 5];

      console.log(arr.sum()); //10

    </script>

不能用覆蓋對象的方式來拓展原型對象Array.prototype={}，因為這樣會把原型對象原有的屬性和方法都覆蓋掉。只能用Array.prototype.xxx=function(){}的方式。

## 2 繼承

ES6之前沒有提供extends繼承，而是通過構造函數+原型對象模擬實現繼承，被稱為組合繼承。

### （1） call()

call()的作用：調用函數，修改函數運行時的this指向。

fun.call(thiaArg, arg1, arg2, …)

thisArg: 當前調用函數this的指向對象

arg1，arg2：普通參數

    <script>

      function fn(*x*, *y*) {

        console.log("手磨咖啡");

        console.log(this);

        console.log(x + y);

      }

      var o = {

        name: "andy",

      };

      // 1 call()可以調用函數

      fn.call();

      // 打印出來後可以看到this指向的是window

      // 2 call()可以改變這個函數的this指向

      fn.call(o);

      // 打印出來後可以看到this指向的是對象o

      fn.call(o, 5, 6);

    </script>

### （2） 借用構造函數和call()繼承父類屬性

核心原理：通過call()把父類（構造函數）的this指向子類（構造函數）的this

    <script>

      // 1 父構造函數

      function Father(*uname*, *age*) {

        // this指向父構造函數的實例對象

        this.uname = uname;

        this.age = age;

      }

      // 2 子構造函數

      function Son(*uname*, *age*) {

        // this指向子構造函數的實例對象

        Father.call(this, uname, age);

      }

      var son = new Son("Andy", 18);

      console.log(son);

    </script>

### （3） 利用原型對象繼承方法

不能通過Son.prototype = Father.prototype的方法來繼承。因為這樣做，是吧Son原型對象指向了Father原型對象；在修改Son原型對象的時候，會連Father原型對象一起修改：這樣不妥。



要用這樣的方式來繼承

      Son.prototype = new Father();



別忘了把繼承後的子原型對象的constructor指回子原型對象的constructor

Son.prototype.constructor = Son;

示例代碼

    <script>

      // 1 父構造函數

      function Father(*uname*, *age*) {

        // this指向父構造函數的實例對象

        this.uname = uname;

        this.age = age;

      }

      // 2 子構造函數

      function Son(*uname*, *age*) {

        // this指向子構造函數的實例對象

        Father.call(this, uname, age);

      }

      // Son.prototype = Father.prototype; 這樣直接賦值會有問題。如果修改了子原型對象，父原型對象也會跟著一起變化。

      // 用這樣的方式來繼承

      Son.prototype = new Father();

      // 這樣做，即便修改子構造函數的原型對象，也不會影響到父原型對象

      Son.prototype.dance = function () {

        console.log("dancing");

      };

      console.log(Father.prototype); //父原型對象沒有被子原型對象改變

      // 別忘了把繼承後的子原型對象的constructor指回子原型對象的constructor

      Son.prototype.constructor = Son; //如果沒有這句話，Son.prototype.constructor指向的是Father

      console.log(Son.prototype.constructor);

    </script>

## 3 ES5中新增的方法

### （1） 數組方法

迭代（遍歷）方法：forEach()，map()，filter()，some()，every()

forEach()、filter() 裡面的return不會終止迭代。

some() 裡面遇到return true會終止遍歷，迭代效率更高。

#### ① array.forEach(function(currentValue, index, arr))

* currentValue：數組當前項的值
* inde：數組當前項的索引
* arr：數組對象本身

    <script>

      // forEach 迭代（遍歷）數組

      var arr = [1, 2, 3];

      arr.forEach(function (*value*, *index*, *array*) {

        console.log("每個數組元素" + value);

        console.log("每個數組元素索引號" + index);

        console.log("數組本身" + array);

      });

    </script>

#### ② array.filter(function(currentValue, index, arr))

* filter()方法創建一個新的數組，新數組中的元素是通過檢查指定數組中符合條件的所有元素而得到。主要用於篩選數組。
* 它直接返回一個新數組
* currentValue：數組當前項的值
* index：數組當前項的索引
* arr：數組對象本身

    <script>

      // filter篩選數組

      var arr = [12, 66, 7, 99];

      var newArr = arr.filter(function (*value*, *index*) {

        return value >= 20;

      });

      console.log(newArr); //[66,99]

    </script>

#### ③ array.some(function(currentValue, index, arr))

* some()方法用於檢測數組中的元素是否滿足指定條件
* 返回的布爾值：如果找到這個元素，返回true；找不到就返回false。
* 如果找到第一個滿足條件的元素，則終止循環不再繼續查找。
* currentValue：數組當前項的值
* index：數組當前項的索引
* arr：數組對象本身

    <script>

      // some查找數組中是否有滿足條件的元素

      var arr = [10, 30, 50];

      var flag = arr.some(function (*value*) {

        return value >= 20;

      });

      console.log(flag); //true

    </script>

#### ④ 查詢商品案例

1. 把數據渲染到頁面中（forEach）
2. 根據價格顯示數據（filter）
3. 根據商品名稱顯示數據（some）

如果查詢數組中唯一的元素，用some方法更合適，因為它找到這個元素，就不再進行循環，效率更高

### （2） 字符串方法

trim()方法會從一個字符串的兩端刪除空白字符

str.trim()

trim()方法不影響原字符串，它返回的是一個新的字符串。

### （3） 對象方法

#### ① Object.key用於獲取對象自身所有的屬性

Object.keys(obj)

效果類似for…in

返回一個由屬性名組成的數組

#### ② Object.defineProperty() 用於定義新屬性或修改原有的屬性

Object.defineProperty(obj, prop, descriptor)

這三個參數都是必須的，不能省略

obj：目標對象

prop：要定義或修改的屬性的名字

descriptor：目標屬性所擁有的特性

* descriptor以對象形式書寫
* value：設置屬性的值，默認為undefined
* writable：值是否可以重寫。true | false 默認為false
* enumerable：目標屬性是否可以被枚舉（在遍歷時是否可以拿到這個屬性）。true | false 默認為false
* configurable：目標屬性是否可以被刪除或是否可以再次修改特性 true | false 默認為false

    <script>

      var obj = {

        id: 1,

        pname: "小米",

        price: 1999,

      };

      // 下面的代碼等價於obj.num = 1000;這是設置新的屬性

      Object.defineProperty(obj, "num", {

        value: 1000,

      });

      // 還可以修改已有的屬性

      Object.defineProperty(obj, "price", {

        value: 99,

      });

      console.log(obj);

    </script>

函數的進階

## 1 函數的定義

① 函數聲明方式function關鍵字（命名函數）

function fn() {};

② 函數表達式（匿名函數）

var fun = function(){}

③ new Function(‘參數1’, ‘參數2’, ‘函數體’)

    <script>

      var f = new Function("a", "b", "console.log(a+b)");

      f(3, 6); //9

      // 所有函數都是Function的實例對象

      console.dir(f); //裡面有\_\_proto\_\_屬性指向Function原型對象

      // 函數也屬於對象

      console.log(f instanceof Object); //true

    </script>

Function裡面參數必須都是字符串格式。

這種方法了解即可，因為效率低，書寫不方便，用得少。

由此可知：所有的函數都是Function的實例對象。



## 2 調用函數和this指向

  <body>

    <button>按鈕</button>

    <script>

      btn = document.querySelector("button");

      // 1 普通函數。this指向window

      function fn() {

        console.log("1 普通函數");

      }

      fn();

      fn.call();

      // 2 對象的方法。this指向對象o

      var o = {

        sayHi: function () {

          console.log("2 對象的方法");

        },

      };

      o.sayHi();

      // 3 構造函數。this指向bd這個實例對象。原型對象裡面的this指向的也是bd這個實例對象。

      function Star() {

        this.sing = function () {

          console.log("3 構造函數");

        };

      }

      var bd = new Star();

      bd.sing();

      // 4 綁定事件函數。this指向的是函數的調用者btn這個按鈕對象。

      btn.onclick = function () {

        console.log("4 綁定事件函數");

      };

      // 5 定時器函數。完整的調用寫法是window.setTimeout，所以this指向window

      setTimeout(function () {

        console.log("5 定時器函數");

      }, 1000);

      // 6 立即執行函數自動調用。立即執行函數和普通函數一樣，只不過不需要程序員調用，所以它的this指向的也是window。

      (function () {

        console.log("6 立即執行函數");

      })();

    </script>

  </body>

調用函數的時候才能確定this指向誰。調用方式不同決定了this的指向不同。

一般this指向調用者。

|  |  |
| --- | --- |
| **調用方式** | **this指向** |
| 普通函數調用 | window |
| 構造函數調用 | 實例對象。  原型對象裡面的方法也指向實例對象。 |
| 對象方法調用 | 該方法所屬對象 |
| 事件綁定方法 | 綁定事件對象 |
| 定時器函數 | window |
| 立即執行函數 | window |

## 3 改變函數內部this指向

JavaScript提供了一些方法來改變this指向，常用的有call()，apply()，bind()三種方法。

相同點：

都可以改變函數內部的this指向。

區別：

* call和apply會調用函數，並且改變函數內部this指向。
* call和apply傳遞的參數不一樣，call傳遞參數arg1，arg2…形式，apply的參數必須是數組形式。
* bind不會調用函數，可以改變函數內部this指向。

運用場景：

* call用於子構造函數繼承父構造函數属性，把父構造函數裡的this指向子構造函數。
* apply經常跟數組有關係，比如藉助於數學對象實現數組最大值最小值。
* bind不調用函數，但要改變this指向的應用：改變定時器內部的this指向。

### （1） call()

call()作用：①調用函數，②改變函數內this的指向，③繼承父構造函數

### （2） apply()

fun.apply(thisArg, [argsArray])

* this.Arg：在fun函數運行時指定的this值
* argsArray：傳遞的值，必須包含在數組裡面。如果把這個數組打印出來，打印的是數組的第一個元素。
* 返回值就是函數的返回值，因為它就是調用函數

apply()作用：①調用函數，②改變函數內this的指向

    <script>

      var o = {

        name: "andy",

      };

      function fn(*arr*) {

        console.log(this); //{name: "andy"}

        console.dir(arr); //1

      }

      fn.apply(o, [1, 2, 3]);

      // 利用apply藉助於數學內置對象Math.max()求最大值

      var arr = [1, 56, 89, 123];

      var max = Math.max.apply(Math, arr); //不改變this的指向，第1個參數寫Math

      console.log(max); //123

    </script>

### （3） bind()

bind()不會調用函數，但是能改變函數內部的this指向。

fn.bind(thisArg, arg1, arg2, …)

* thisArg：在fun函數運行時指定的this值
* arg1，arg2：傳遞的其它參數
* 返回由指定的this值和初始化參數改造的原函數拷貝

    <script>

      var o = {

        name: "andy",

      };

      function f(*a*, *b*) {

        console.log(this);

        console.log(a + b);

      }

      var fn = f.bind(o, 3, 6);

      fn(); //{name: "andy"}  9

    </script>

有的函數不需要立即調用，只要改變這個函數內部的this指向，這時使用bind()最合適。

  <body>

    <button>點擊</button>

    <script>

      // 需求，點擊了按鈕後禁用1秒鐘，然後再回復常態

      var btn = document.querySelector("button");

      btn.onclick = function () {

        this.disabled = true; //this指向btn按鈕

        setTimeout(

          function () {

            this.disabled = false; //定時器裡面的函數this指向的是window，所以要通過bind()來改變this指向。

          }.bind(this), //這個this指的是btn。因為這個bind()在定時器函數外面，在onclick函數裡面。這樣寫是把setTimeout裡面的函數裡面的this由指向window改為指向btn

          1000

        );

      };

    </script>

  </body>

## 4 嚴格模式

### （1）什麼是嚴格模式

JS中代碼模式可以分為非嚴格模式和嚴格模式。以前的代碼在非嚴格模式下寫的。ES5之后新增严格模式。IE10以上的版本支持嚴格模式。舊版本瀏覽器會忽略嚴格模，用普通模式來解析代碼。

嚴格模式：在嚴格的條件下運行JS代碼。

嚴格模式對普通模式的JavaScript語義做了一些更改：

1. 消除了JavaScript語法的一些不合理、不嚴謹之處，減少了一些怪異行為。

比如沒有聲明就可以賦值這種怪異之處。

1. 消除了代碼的一些不安全之處，保證代碼運行安全。
2. 提高編譯器效率，增加運行速度。
3. 禁用了在ECMAScript的未來版本中可能會定義的一些語法，為未來新版本的JavaScript做好鋪墊。比如一些保留字class，enum，export，extends，import，super不能做變量名。

### （2）如何開啟嚴格模式

嚴格模式可以應用到整個腳本或個別函數中。因此在使用時，可以將嚴格模式分為為腳本開啟嚴格模式和為函數開啟嚴格模式兩種情況。

#### ① 為腳本開啟嚴格模式

在所有語句之前放一個特定語句“use strict”; （單引號雙引號都可以）

    <script>

      // 為整個腳本(script標籤)開啟嚴格模式

      "use strict";

      // 下面的js代碼就會按照嚴格模式執行。

</script>

    <!-- 或者在立即執行函數裡開啟嚴格模式 -->

    <script>

      (function () {

        "use strict";

        // 下面的js代碼就會按照嚴格模式執行。

      })();

    </script>

#### ② 為函數開啟嚴格模式

把“use strict”; 聲明放在函數體所有語句之前。

    <script>

      function fn1() {

        "use strict";

        // 下面的代碼按照嚴格模式執行

      }

      function fn2() {

        // 這些代碼按照普通模式執行

      }

    </script>

### （3）嚴格模式中的變化

#### ① 變量規定

1. 在普通模式中，如果一個變量沒有聲明就賦值，默認是全局變量。嚴格模式禁止這種用法，變量都必須先用var命令聲明，然後再使用。
2. 嚴禁刪除已經聲明過的變量。例如，delete x; 語法是錯誤的。

#### ② this指向

1. 嚴格模式下，全局作用域中this是undefined。普通模式下，全局作用域中this指向window對象。
2. 嚴格模式下，如果構造函數不加new調用，構造函數中的this相關代碼會報錯。new實例化的構造函數指向創建的對象實例。

    <script>

      "use strict";

      function Star() {

        this.gender = "female";

      }

      // 沒有new Star()而直接調用Star()會報錯。因為this在普通模式下本來指向window，現在嚴格模式this是undefined，undefined.gender是錯誤。

      Star();

    </script>

1. 嚴格模式下，定時器函數裡的this指向的也是window。
2. 嚴格模式下，事件、對象還是指向調用者。

#### ③ 函數變化

1. 嚴格模式下，函數不能有重名的參數

非嚴格模式下函數參數重名的例子：

    <script>

      function fn(*a*, *a*) {

        console.log(a + a);

      }

      fn(1, 2);

      //打印出來是4  因為一開始a=1, 然後a=2，接著2+2=4

    </script>

1. 函數必須聲明在頂層。新版的JavaScript會引入“塊級作用域”（ES6中已引入）。為了與新版本接軌，不允許在非函數的代碼塊內聲明函數。

語法錯誤：

if(true){

function f() {}

}

for(var I = 0; i<5; i++){

function f() {}

}

符合語法規範：

function f1() {

function f2() {}

}

## 5 高階函數

高階函數是對其它函數進行操作的函數，它接收函數作為參數或將函數作為返回值輸出。

示例代碼：

    <style>

      div {

        position: relative;

        width: 100px;

        height: 100px;

        background-color: #ff6670;

      }

    </style>

  <body>

    <div></div>

    <script src="./jquery.mini.js"></script>

    <script>

      // 高階函數：函數作為參數傳遞

      function fn(*a*, *b*, *callback*) {

        console.log(a + b);

        callback && callback();

      }

      fn(1, 2, function () {

        console.log("callback");

      });

      // 動畫，移動以後變色

      $("div").animate(

        {

          left: 500,

        },

        function () {

          $("div").css("backgroundColor", "#ffd000");

        }

      );

    </script>

  </body>

## 6 閉包

### （1）變量作用域

全局變量（函數外部）和局部變量（函數內部）

1. 函數內部可以使用全局變量
2. 函數外部不可以使用局部變量
3. 當函數執行完畢，本作用域內的局部變量被銷毀

### （2）什麼是閉包

~~閉包（closure）指有權訪問另一個函數作用域中變量的函數。~~（講師的這個定義和他舉的例子矛盾）

閉包（closure）是函數。它的作用域中的變量可以被別的函數訪問到。

    <script>

      function fn() {

        var num = 10;

        function fun() {

          console.log(num);

        }

        fun();

      }

      fn();

      // fun函數作用域裡的代碼可以訪問fn函數作用域裡的代碼var num=10，fn()就是閉包函數

    </script>

另外一個例子，讓全局作用域訪問某個函數內的變量。通過return子函數實現。

    <script>

      function fn() {

        var num = 10;

        function fun() {

          console.log(num);

        }

        return fun;

      }

      var f = fn();

      // 上面這行代碼的意思是：

      // var f = function fun() {

      //   console.log(num);

      // };

      // 只要調用f，就能訪問fn裡面的num，這樣fn()也成了閉包

      f();

    </script>

### （3）閉包的作用：延伸了變量的作用範圍。

### （4）閉包案例：循環註冊點擊事件

  <body>

    <ul class="nav">

      <li>入塞</li>

      <li>子夜</li>

      <li>天香</li>

      <li>月慢</li>

    </ul>

    <script>

      // 閉包應用，點擊li輸出當前li的索引號

      // 1 動態添加屬性

      var lis = document.querySelector(".nav").querySelectorAll("li");

      for (var i = 0; i < lis.length; i++) {

        lis[i].index = i;

        lis[i].onclick = function () {

          // 如果直接console.log(i)；不論點擊那個li，輸出的都是4。因為循環是立馬執行的，而點擊事件是異步的，會在for循環之後執行。

          // console.log(i);

          console.log(this.index);

        };

      }

      // 2 利用閉包的方式得到當前li的索引號

      for (var i = 0; i < lis.length; i++) {

        // 利用for循環創建了4個立即執行函數。立即執行函數的第2個小括號是調用函數，這裡可以傳遞參數給函數。

        // 立即執行函數也稱為小閉包。因為立即執行函數裡面的任何一個函數都可以使用它的i這個變量。

        (function (*i*) {

          // console.log(i);

          lis[i].onclick = function () {

            console.log(i);

          };

        })(i);

      }

    </script>

  </body>

### （5）閉包案例：打印li標籤裡面的內容

  <body>

    <ul class="nav">

      <li>入塞</li>

      <li>秋水</li>

      <li>拜星</li>

      <li>春晴</li>

    </ul>

    <script>

      // 閉包應用：3秒鐘之後，打印所有li內容

      var lis = document.querySelector(".nav").querySelectorAll("li");

      for (var i = 0; i < lis.length; i++) {

        // 不能直接在定時器的回調函數裡console.log(lis[i].innerHTML);否則報錯說Cannot read property 'innerHTML' of undefined。原因：定時器裡的回調函數是異步任務。會放在任務隊列裡，等到同步任務（比如for循環）執行完了以後再執行。等執行定時器裡的回調函數時，i已經變成4了。

        // setTimeout(function () {

        //   console.log(lis[i].innerHTML);

        // }, 2000);

        // 要用一個立即執行函數來包裹定時器

        (function (*i*) {

          setTimeout(function () {

            console.log(lis[i].innerHTML);

          }, 2000);

        })(i);

      }

    </script>

  </body>

### （6）閉包案例：打車計算價格

    <script>

      // 閉包應用-計算打車價格

      // 打車起步價13（3公里內），之後每多一公里增加5塊錢，用戶輸入公里數就可以計算打車價格

      // 如果有擁堵情況，總價格多收取10塊錢擁堵費

      var car = (function () {

        var start = 13; //起步價  局部變量

        var total = 0; //總價  局部變量

        return {

          //正常的總價

          // price 和 congestion使用了car這個立即執行函數裡面的變量，所以car是閉包函數

          price: function (*n*) {

            if (n <= 3) {

              total = start;

            } else {

              total = start + (n - 3) \* 5;

            }

            return total;

          },

          //擁堵的費用

          congestion: function (*flag*) {

            return flag ? total + 10 : total;

          },

        };

      })();

      console.log(car.price(5)); //23

      console.log(car.congestion(true)); //33

    </script>

### （6）思考題1

    <script>

      var name = "The Window";

      var object = {

        name: "My Object",

        getNameFunc: function () {

          return function () {

            return this.name;

          };

        },

      };

      console.log(object.getNameFunc()()); //The Window

      // 分析：

      // var f = object.getNameFunc()，這裡返回的是function () {

      //   return this.name;

      // };

      // 然後調用函數f()，返回的是this.name。f()是個普通函數，this指向的是window，所以this.name="The Window";

    </script>

### （7）思考題2

    <script>

      var name = "The Window";

      var object = {

        name: "The Object",

        getNameFunc: function () {

          var that = this;

          return function () {

            return that.name;

          };

        },

      };

      console.log(object.getNameFunc()()); //The Object

      // 分析

      // var f = object.getNameFunc();  返回的是function () {

      //   var that = this;

      //   return function () {

      //     return that.name;

      //   };

      // }

      // 因為函數被object調用，根據函數裡的this指向調用者的規則，this指向object，於是that也是object。 然後f被賦予返回的值，值是return的function () {

      //     return that.name;

      //   };

      // 接下來調用f()，打印出來的是"The Object"

    </script>

## 7 遞歸

### （1）什麼是遞歸

如果一個函數在其內部可以調用其本身，這個函數就是遞歸函數。

遞歸函數和循環一樣，很容易發生“棧溢出”錯誤（stack overflow），所以必須要加退出條件return。

### （2）利用遞歸求階乘

    <script>

      function fn(*n*) {

        if (n === 1) {

          return 1;

        }

        return n \* fn(n - 1);

      }

      console.log(fn(3)); //6

      // 分析 fn(3)  return 3\*f(2)

      //                      return 2 \* f(1)

      //                                  return 1

    </script>

### （3）利用遞歸求斐波那契數列

斐波那契數列（俗稱兔子序列）：前兩項相加的和=第3項的值

    <script>

      function fn(*n*) {

        if (n === 1 || n === 2) {

          return 1;

        }

        return fn(n - 1) + fn(n - 2);

      }

      console.log(fn(5)); //5

    </script>

### （4）利用遞歸遍歷數據並打印

    <script>

      var data = [

        {

          id: 1,

          name: "家电",

          goods: [

            {

              id: 11,

              gname: "冰箱",

              goods: [

                {

                  id: 111,

                  gname: "海尔",

                },

                {

                  id: 112,

                  gname: "美的",

                },

              ],

            },

            {

              id: 12,

              gname: "洗衣机",

            },

          ],

        },

        {

          id: 2,

          name: "服饰",

        },

      ];

      // 需求：輸入id號，返回相應的數據對象

      // 利用forEach遍歷裡面每個對象

      function getData(*json*, *id*) {

        json.forEach(function (*item*) {

          // console.log(item);

          if (item.id === id) {

            console.log(item);

            // 要得到裡層的數據 比如 goods裡的id為11、12的數據，可以利用遞歸函數

            // 判斷一下item應該偶goods這個數組且數組長度不為0

          } else if (item.goods && item.goods.length > 0) {

            getData(item.goods, id);

          }

        });

      }

      getData(data, 11);

    </script>

### （5）利用遞歸遍歷數據並返回值

    <script>

      var data = [

        {

          id: 1,

          name: "家电",

          goods: [

            {

              id: 11,

              gname: "冰箱",

              goods: [

                {

                  id: 111,

                  gname: "海尔",

                },

                {

                  id: 112,

                  gname: "美的",

                },

              ],

            },

            {

              id: 12,

              gname: "洗衣机",

            },

          ],

        },

        {

          id: 2,

          name: "服饰",

        },

      ];

      // 需求：輸入id號，返回相應的數據對象

      // 利用forEach遍歷裡面每個對象

      function getData(*json*, *id*) {

        var o = {};

        json.forEach(function (*item*) {

          // console.log(item);

          if (item.id === id) {

            // console.log(item);

            o = item;

            // 要得到裡層的數據 比如 goods裡的id為11、12的數據，可以利用遞歸函數

            // 判斷一下item應該偶goods這個數組且數組長度不為0

          } else if (item.goods && item.goods.length > 0) {

            // 裡層循環return來的o，這裡要賦值給o，否則返回的是空對象

            o = getData(item.goods, id);

          }

        });

        return o;

      }

      console.log(getData(data, 1));

      console.log(getData(data, 11));

      console.log(getData(data, 111));

    </script>

### （6）淺拷貝和深拷貝

淺拷貝只是拷貝一層，更深層次對象級別的只拷貝引用（地址）。

Object.assign(target, ...sources) ES6新增方法可以淺拷貝。

    <script>

      var obj = {

        id: 1,

        name: "andy",

        msg: {

          age: 18,

        },

      };

      var o = {};

// for (var k in obj) {

// // k 是属性名 obj[k] 属性值

// o[k] = obj[k];

// }

// console.log(o);

      Object.assign(o, obj);

      console.log(o);

    </script>

深拷貝拷貝多層，每一級別的數據都會拷貝。

    <script>

      // 深拷贝拷贝多层, 每一级别的数据都会拷贝.

      var obj = {

        id: 1,

        name: "andy",

        msg: {

          age: 18,

        },

        color: ["pink", "red"],

      };

      var o = {};

      // 封裝函數

      function deepCopy(*newobj*, *oldobj*) {

        for (var k in oldobj) {

          // 獲取屬性值

          var item = oldobj[k];

          // 判斷屬性值屬於哪種數據類型

          // 要先判斷是否屬於數組，然後再判斷是否屬於對象。因為數組也屬於對象。

          if (item instanceof Array) {

            newobj[k] = [];

            deepCopy(newobj[k], item);

          } else if (item instanceof Object) {

            // 判斷這個值是否是對象

            newobj[k] = {};

            deepCopy(newobj[k], item);

          } else {

            // 屬於簡單數據類型

            newobj[k] = item;

          }

        }

      }

      deepCopy(o, obj);

      console.log(o);

    </script>

正則表達式

## 1 正則表達式概述

正則表達式（Regular Expression）是用於匹配字符串中字符組合的模式。

在JavaScript中，正則表達式也是對象。

正則表達式通常用來檢索、替換那些符合某個模式（規則）的文本。其作用：

* 匹配。例如驗證表單：用戶名表單只能輸入英文字母、數字或者下劃線，暱稱輸入框中可以輸入中文。
* 替換。正則表達式用於過濾頁面內容中的某些詞，比如髒話。
* 提取。從字符串中獲取我們想要的某個部分。

實際開發時，一般是直接複製寫好的正則表達式，但是開發者自己要會使用正則表達式並且根據實際情況修改正則表達式。

## 2 正則表達式在JavaScript中的使用

### （1）創建正則表達式

在JavaScript中，可以通過兩種方式創建一個正則表達式。

① 通過調用RegExp對象的構造函數創建

var 變量名 = new RegExp(/表達式/);

    <script>

      var regexp = new RegExp(/123/);

      console.dir(regexp);

      console.log(regexp);

    </script>

② 通過字面量創建

var 變量名 = /表達式/;

    <script>

      var rg = /123/;

    </script>

### （2）測試正則表達式

text()正則對象方法，用於檢測字符串是否符合該規則，該對象會返回true或false，其參數是被測試的字符串。

語法

regexObj.text(str)

* regexObj是正則表達式
* str是被測試的文本
* 這是在檢測str文本是否符合regexObj這個正則表達式的規範

    <script>

      var rg = /123/;

      console.log(rg.test("123")); //true

      console.log(rg.test(123)); //true

      console.log(rg.test("as")); //false

    </script>

### （3）正則表達式的組成

一個正則表達式可以由簡單的字符構成，比如/abc/；也可以是簡單和特殊字符的組合，比如/ab\*c/。其中特殊字符也被稱為元字符，在正則表達式中是具有特殊意義的專用符號，如^，$，+等。

正則表達式裡面不需要加引號，不管是數字型還是字符串型。

## 3 正則表達式中的特殊字符

### （1）邊界符

|  |  |
| --- | --- |
| 邊界符 | 說明 |
| ^ | 表示匹配行首的文本（以誰開始） |
| $ | 表示匹配行尾的文本（以誰結束） |

    <script>

      // 邊界符 ^ $

      var rg = /abc/; //正則表達式裡面不需要加引號，不管是數字型還是字符串型

      // /abc/表示只要包含abc，返回的都是true

      console.log(rg.test("abc")); //true

      console.log(rg.test("abcd")); //true

      console.log(rg.test("aabc")); //true

      console.log("——————分界線——————");

      var reg = /^abc/; //表示以abc開頭

      console.log(reg.test("abc")); //true

      console.log(reg.test("abcd")); //true

      console.log(reg.test("aabc")); //false

      console.log("——————分界線——————");

      var reg1 = /^abc$/; //精確匹配，必須是abc字符串

      console.log(reg1.test("abc")); //true

      console.log(reg1.test("abcd")); //false

      console.log(reg1.test("aabc")); //false

    </script>

### （2）字符類

[] 表示有一系列字符可供選擇

[-] 方括號內部範圍符號-

[^] 方括號內部取反符號^

    <script>

      var rg = /[abc]/; //只要包含a或者包含b或者包含c，都返回為true

      console.log(rg.test("andy")); //true

      console.log(rg.test("baby")); //true

      console.log(rg.test("color")); //true

      console.log(rg.test("red")); //false

      console.log("——————分隔符——————");

      var reg = /^[abc]$/; //三選一 只有是a 或者是b 或者是c 這三個字母才返回true

      console.log(reg.test("a")); //true

      console.log(reg.test("b")); //true

      console.log(reg.test("c")); //true

      console.log(reg.test("abc")); //false

      console.log("——————分隔符——————");

      var reg1 = /^[a-z]$/; //26個英文字母中任意一個小寫字母，返回的都是true

      console.log(reg1.test("A")); //false

      console.log(reg1.test("z")); //true

      console.log(reg1.test("aaa")); //false

      console.log("——————分隔符——————");

      // 字符組合

      var reg2 = /^[a-zA-Z0-9\_-]$/; //這些字符裡面的任意一個，返回的都是true

      // 取反。如果中括號裡面有^，表示取反，不要和邊界符^混淆

      var reg3 = /^[^a-zA-Z0-9\_-]$/; //表示除了括號裡的字符之外的字符

      console.log(reg3.test("aa"));

    </script>

### （3）量詞符

定義某個模式出現的次數

|  |  |
| --- | --- |
| 量詞 | 說明 |
| \* | 重複零次或更多次 |
| + | 重複一次或更多次 |
| ? | 重複零次或一次 |
| {n} | 重複n次 |
| {n,} | 重複n次或更多次 |
| {n,m} | 重複n到m次 |

    <script>

      console.log("——————讓a出現0次或者更多次 >=0——————");

      var rg1 = /^a\*$/;

      console.log(rg1.test("")); //true

      console.log(rg1.test("a")); //true

      console.log(rg1.test("aaa")); //true

      console.log("——————讓a出現1次或者更多次 >=1——————");

      var rg2 = /^a+$/;

      console.log(rg2.test("")); //false

      console.log(rg2.test("a")); //true

      console.log(rg2.test("aaa")); //true

      console.log("——————a有沒有出現 =1 || =0——————");

      var rg3 = /^a?$/;

      console.log(rg3.test("")); //true

      console.log(rg3.test("a")); //true

      console.log(rg3.test("aaa")); //false

      console.log("——————a出現n次——————");

      var rg4 = /^a{3}$/;

      console.log(rg4.test("")); //false

      console.log(rg4.test("a")); //false

      console.log(rg4.test("aaa")); //true

      console.log("——————a出現n次以上——————");

      var rg5 = /^a{3,}$/;

      console.log(rg5.test("")); //false

      console.log(rg5.test("a")); //false

      console.log(rg5.test("aaaa")); //true

      console.log("——————n ≤ a出現次數 ≤ m——————");

      var rg6 = /^a{3,6}$/;

      console.log(rg6.test("")); //false

      console.log(rg6.test("a")); //false

      console.log(rg6.test("aaaaa")); //true

      console.log(rg6.test("aaaaaaaa")); //false

    </script>

    <script>

      // 量詞是設定某個模式出現的次數

      var reg = /^[a-zA-Z0-9\_-]{6,16}$/; //這是個典型的用戶名正則表達式，用戶可以輸入任意的大小寫字母，0-9之間的數字，下劃線\_，中劃線-，這種模式可以重複6-19次。

      // {6,16}中間不要有空格

      console.log(reg.test("aaa")); //false

      console.log(reg.test("a210faL-")); //true

    </script>

### （4）案例

功能需求：

如果用戶名輸入合法，後面提示信息為“用戶名格式正確”，並且顏色為綠色。

如果用戶名輸入不合法，後面提示信息為“用戶名不符合規範”，並且顏色為紅色。

分析：

1. 用戶名只能是英文字母、數字、下劃線或者短橫線，並且用戶名長度為6-16位。
2. 首先準備好這種正則表達式模式/^[a-zA-Z0-9-\_]{6,19}$/。
3. 當表單失去焦點就開始驗證。
4. 如果符合正則規範，則讓後面的span標籤添加right類。
5. 如果不符合正則規範，則讓後面的span標籤添加wrong類

    <style>

      span {

        color: #aaa;

        font-size: 14pxl;

      }

      .right {

        color: green;

      }

      .wrong {

        color: red;

      }

    </style>

  <body>

    <input type="text" class="uname" /><span>請輸入用戶名</span>

    <script>

      var reg = /^[a-zA-Z0-9-\_]{6,19}$/;

      var uname = document.querySelector(".uname");

      var span = document.querySelector("span");

      uname.onblur = function () {

        if (reg.test(this.value)) {

          span.className = "right";

          span.innerHTML = "用戶名格式輸入正確";

        } else {

          span.className = "wrong";

          span.innerHTML = "用戶名格式輸入錯誤";

        }

      };

    </script>

  </body>

### （5）括號總結

    <script>

      // 中括號，字符集合，匹配方括號中的任意字符

      var reg1 = /^[abc]$/; // a可以，b也可以，c也可以  a||b||c

      // 大括號，量詞符，裡面表示重複次數

      var reg2 = /^abc{3}$/; //讓c重複3次

      console.log(reg2.test("abc")); //false

      console.log(reg2.test("abcabcabc")); //false

      console.log(reg2.test("abccc")); //true

      // 小括號表示優先級

      var reg3 = /^(abc){3}$/; //讓abc重複3次

      console.log(reg3.test("abc")); //false

      console.log(reg3.test("abcabcabc")); //true

      console.log(reg3.test("abccc")); //false

    </script>

### （6）正則表達式在線測試工具

<https://c.runoob.com/front-end/854>

### （7）正則表達式中的特殊符號

預定義類：某些常見模式的簡寫方式

|  |  |
| --- | --- |
| 預定義類 | 說明 |
| \d | 匹配0-9之間的任意一個數字，相當於[0-9] |
| \D | 匹配所有0-9以外的字符，相當於[^0-9] |
| \w | 匹配任意的字母、數字和下劃線，相當於[A-Za-z0-9\_] |
| \W | 除了所有字母、數字和下劃線以外的字符，相當於[^A-Za-z0-9\_] |
| \s | 匹配空格（包括換行符、製表符、空格符等），相當於[\t\r\n\v\f] |
| \S | 匹配非空格的字符，相當於[^\t\r\n\v\f] |

### （8）案例：驗證座機號碼

    <script>

      // 座機號驗證：全國座機號碼  兩種格式： 010-12345678   0530-1234567

      // 正則裡面的或者符號  |

      var reg = /^(010|02\d|0\d{3})-\d{7,8}$/; //電話號碼區號是010或者02\*，或者是0\*\*\*

    </script>

## 4 正則表達式中的替換

replace()方法可以實現替換字符串操作，用來替換的參數可以是一個字符串或是一個正則表達式。

複習一下replace()的用法

    <script>

      var str = "abc和efg";

      var newStr = str.replace("abc", "xyz");

      console.log(newStr); //xyz和efg

    </script>

stringObject.replace(regexp/substr,replacement)

1. 第一個參數：正則表達式或者被替換的字符串
2. 第二個參數：替換為的字符串
3. 返回值是一個替換完畢後的新字符串

正則表達式參數

/表達式/[switch]

switch也稱為修飾符，規定按照什麼樣的模式來匹配，有三種值：

* g：全局匹配
* i：忽略大小寫
* gi：全局匹配+忽略大小寫

案例：在textarea裡面輸入內容，點擊button提交，正則表達式過濾文本，文本顯示到div裡。

  <body>

    <textarea name="" id="message" cols="30" rows="10"></textarea

    ><button>提交</button>

    <div></div>

    <script>

      var text = document.querySelector("textarea");

      var btn = document.querySelector("button");

      var div = document.querySelector("div");

      btn.onclick = function () {

        div.innerHTML = text.value.replace(/洪水|啦|abc/g, "\*\*"); //如果不寫g，只會替換1次，後面的就不會替換了。

      };

    </script>

  </body>

ES6語法

## 1 什么是ES6

ES全稱ECMAScript，由ECMA國際標準化組織制定的一項腳本語言的標準化規範。它規定腳本語言如何實現。JavaScript是實現了這個標準的腳本語言。ES後面的6代表規範的版本。在這個版本中新增了很多好用的語法規範。

標準在每年的6月發佈一次，版本號用年號表示。比如2015年6月發佈了ES2015版本，2016年6月發佈了ES2016版本，2017年6月發佈了ES2017版本，2018年6月發佈了ES2018版本。ES6是泛指，泛指ES2015以及其後續的版本。

## 2 為什么使用ES6

每一次發佈新標準都意味著語言更完善，功能更強大。

JavaScript有不完美的地方，比如：

* 變量特性提升增加了程序運行時的不可預測性
* 語法過於鬆散，實現相同的功能，不同的程序員有不同的寫法

ES6會逐漸彌補這些不完美的地方。

## 3 let

let用於替換var

let特點：

### （1） let聲明的變量只在所處的塊級有效

好處：防止內層變量覆蓋外層變量；防止循環變量變成全局變量。

在ES6之前，JavaScript中只有全局作用域和局部作用域；在ES6中新增了塊級作用域（即一對大括號{}產生的作用域，比如if語句、for循環中都有大括號）。

    <script>

      if (true) {

        let a = 10;

      }

      console.log(a); //a is not defined

    </script>

使用let關鍵字聲明變量才能形成塊級作用域，使用var關鍵字聲明變量無法形成塊級作用域。

    <script>

      if (true) {

        var a = 10;

      }

      console.log(a); //10

    </script>

防止循環變量變成全局變量

    <script>

      // 在for循环的小括号里用let声明变量，在for循环外面访问不到

      for (var h = 0; h < 5; h++) {}

      console.log(h); //5

      for (let i = 0; i < 5; i++) {}

      console.log(i); //i is not defined

    </script>

### （2） let聲明的變量無變量提升

    <script>

      console.log(a); //undefined

      var a = 10;

      console.log(b); //报错：Cannot access 'b' before initialization

      let b = 20;

    </script>

ES6的變量使用更嚴格，只能先聲明，再使用。

### （3） let聲明的變量有“暫時性鎖區”的特點

    <script>

      var num = 10;

      if (true) {

        // 一旦在塊級區域聲明了變量，該變量和外部的變量就沒有任何關係了

        console.log(num); //报错：Cannot access 'num' before initialization

        let num = 20;

      }

    </script>

### （4） 經典面試題

    <script>

      var arr = [];

      for (var i = 0; i < 2; i++) {

        arr[i] = function () {

          console.log(i);

        };

      }

      // 執行函數的時候，會發現函數內部並沒有定義i，根據作用域鏈查找原則，會去上級作用域中查找。這個函數的上級作用域是全局作用域。循環執行時，恰好在全局作用域中創建了i。等到執行函數時，循環已經結束，此時i=2。所以下面兩個函數log出來的值都是2。

      arr[0](); //2

      arr[1](); //2

      // ——————————————————————————————————————————————————————

      let arr2 = [];

      // 用let聲明的循環，在循環的過程中，創建了2個塊級作用域，在這2個塊級作用域中分別i=0和i=1

      for (let i = 0; i < 2; i++) {

        arr2[i] = function () {

          console.log(i);

        };

      }

      //  執行函數的時候，會發現函數內部並沒有定義i，根據作用域鏈查找原則，會去上級作用域中查找。這個函數的上級作用域是塊級作用域，塊級作用域中相對應的i的值分別是i=0和i=1。

      arr2[0](); //0

      arr2[1](); //1

    </script>

## 4 const

作用：聲明常量。常量是值（內存地址）不能變化的量。

### （1） const聲明的常量只在所處的塊級有效

    <script>

      if (true) {

        const a = 10;

        if (true) {

          // 這個a只在這個if語句的塊級作用域裡起作用，和外面的不相干。

          const a = 20;

          console.log(a); //20

        }

        console.log(a); //10

      }

      console.log(a); //a is not defined

    </script>

### （2） const聲明常量時必須賦初始值，否則報錯

### （3） 常量賦值後，不能重新賦值

基本數據類型比如字符串、數字，常量賦值後不能更改。

    <script>

      const PI = 3.14;

      PI = 100; //報錯，不能分配值給常量：Assignment to constant variable.

    </script>

複雜數據類型比如數組、對象，常量賦值後不能重新再賦值，但是數組、對象內部的數據可以更改。

    <script>

      const ary = [100, 200];

      // 可以修改數組內部的值

      ary[0] = "a";

      ary[1] = "b";

      console.log(ary); //["a", "b"]

      // 不能給數組重新賦值

      ary = ["a", "b"]; //報錯：Assignment to constant variable.

    </script>

## 5 let、const、var的區別

|  |  |  |
| --- | --- | --- |
| var | let | const |
| 變量 | 變量 | 常量 |
| 可重新賦值 | 可重新賦值 | 不可重新賦值 |
| 函數級作用域 | 塊級作用域 | 塊級作用域 |
| 變量提升 | 無變量提升 | 無變量提升 |

編程時，如果使用的數據不需要不會，盡量用const定義。比如定義函數、數學公式中一些恆定不變的值。

JavaScript引擎不需要實時監控值的變化，所以const效率比let高。

## 6 解構賦值

ES6中允許從數組和對象中提取值，按照對應位置，對變量賦值。

### （1） 數組解構

    <script>

      let ary = [1, 2, 3];

      let [a, b, c] = ary;

      console.log(a); //1

      console.log(b); //2

      console.log(c); //3

    </script>

    <script>

      let ary = [1, 2, 3];

      let [a, b, c, d, e] = ary;

      console.log(a); //1

      console.log(b); //2

      console.log(c); //3

      console.log(d); //undefined

      console.log(e); //undefined

    </script>

### （2） 對象解構

對象解構允許使用變量的名字匹配對象的屬性，匹配成功將對象屬性的值賦值給變量。

    <script>

      let person = { name: "andy", age: 30, gender: "m" };

      let { name, age, gender } = person;

      console.log(name); //andy

      console.log(age); //30

      console.log(gender); //m

    </script>

按照一定模式，從數組或對象中提取值，將提取出來的值賦值給另外的變量。

    <script>

      let person = { name: "andy", age: 30, gender: "m" };

      let { name: uName, age: uAge, gender: uGender } = person;

      console.log(uName); //andy

      console.log(uAge); //30

      console.log(uGender); //m

    </script>

## 7 箭頭函數

### （1） 箭頭函數的形式

ES6中新增的定義函數的方式

() => {}

const fn = () => {}

    <script>

      const fn = () => {

        console.log(123);

      };

      fn(); //123

    </script>

如果函數體中只有一句代碼，且代碼的執行結果就是返回值，可以省略大括號。

下面兩個函數是1個函數

    <script>

      function sum(*num1*, *num2*) {

        return num1 + num2;

      }

      const sum = (*sum1*, *num2*) => num1 + num2;

    </script>

如果形參只有一個，可以省略形參外側的小括號。

下面兩個函數是1個函數

    <script>

      function fn(*v*) {

        alert(v);

      }

      const fn = (*v*) => {

        alert(v);

      };

    </script>

### （2） 箭頭函數中的this

箭頭函數不綁定this。箭頭函數中的this，指向的是函數定義位置的上下文this。

    <script>

      // 箭頭函數的this指向箭頭函數定義位置中的this。下面的示例中，箭頭函數被定義在了fn函數內部，箭頭函數的this會指向fn函數內部的this

      function fn() {

        console.log(this);

        return () => {

          console.log(this);

        };

      }

      const obj = { name: "andy" };

      const resFn = fn.call(obj); //fn把this指向了obj後，打印的是{name: "andy"}   ，同時返回了一個箭頭函數用resFn來接取，然後調用resFn，看看箭頭函數裡的this指向哪裡（指向和fn函數裡的this一樣，都是obj）

      resFn(); //{name: "andy"}

    </script>

### （3） 箭頭函數中的this-經典面試題

    <script>

      // 對象不能產生塊級作用域，所以say這個箭頭函數實際上被定義在了全局window裡面。而window是沒有age屬性的。所以打印結果是undefined。可以添加一個全局屬性var age =100;那麼打印出來的就是100

      // var age = 100;

      var obj = {

        age: 20,

        say: () => {

          console.log(this.age);

        },

      };

      obj.say(); //undefined

    </script>

## 8 ES6新增的語法

### （1） 剩餘參數

剩餘參數語法允許我們將一個不定數量的參數表示為一個數組。

    <script>

      function sum(*first*, ...*args*) {

        console.log(first); //10

        console.log(args); //[20,30]

      }

      sum(10, 20, 30);

    </script>

    <script>

      const sum = (...*args*) => {

        let total = 0;

        args.forEach((*item*) => {

          total += item;

        });

        return total;

      };

      console.log(sum(10, 20));

      console.log(sum(10, 20, 30));

    </script>

### （2） 剩餘參數配合解構

    <script>

      let students = ["Andy", "Harry", "Lily"];

      let [s1, ...s2] = students;

      console.log(s1); //Andy

      console.log(s2); //["Harry", "Lily"]

    </script>

### （3） 擴展運算符

擴展運算符可以將數組或者對象轉換為用逗號分隔的參數序列。

    <script>

      let ary = [1, 2, 3];

      console.log(...ary); //本來應該是打印出1,2,3  而實際上是1 2 3。為什麼輸出結果中無逗號？因為逗號被當做了console.log()的參數分割符了，所以輸出結果沒有逗號。

    </script>

### （4） 擴展運算符應用：合併數組

方法1：

    <script>

      let ary1 = [1, 2, 3];

      let ary2 = [4, 5, 6];

      let ary3 = [...ary1, ...ary2];

      console.log(ary3);// [1, 2, 3, 4, 5, 6]

    </script>

方法2：

    <script>

      let ary1 = [1, 2, 3];

      let ary2 = [4, 5, 6];

      ary1.push(...ary2);

      console.log(ary1); // [1, 2, 3, 4, 5, 6]

    </script>

### （5） 擴展運算符應用：將類數組或可遍歷對象轉換為真正的數組

  <body>

    <div>盒子6</div>

    <div>盒子5</div>

    <div>盒子2</div>

    <div>盒子3</div>

    <div>盒子4</div>

    <div>盒子1</div>

    <script>

      // 利用擴展運算符把偽數組轉換為真正的數組

      var oDivs = document.getElementsByTagName("div"); //返回值是個元素集合，偽數組

      console.log(oDivs);

      var ary = [...oDivs];

      console.log(ary);

      // 把偽數組轉換成真正的數組後，就可以調用數組下面的方法。

    </script>

  </body>

## 9 ES6中Array的擴展方法

### （1） Array.from()

Array.from()接收偽數組作為參數，返回值是個真正的數組。

    <script>

      let arrLike = {

        0: "a",

        1: "b",

        2: "c",

        length: 3,

      };

      let arr2 = Array.from(arrLike);

      console.log(arr2); //["a", "b", "c"]

    </script>

Array.from()的第二個參數是一個函數，該函數對數組的函數加工處理。數組中有多少個元素，函數就會被調用多少次。函數有個形參，代表當前被處理的數組元素。

    <script>

      let arrLike = {

        0: "0",

        1: "1",

        2: "2",

        length: 3,

      };

      let arr2 = Array.from(arrLike, *item* => item \* 2);

      console.log(arr2); //[0,2,4]

    </script>

### （2） Array.find()

用於找出第一個符合條件的數組成員，如果沒有找到則返回undefined。

    <script>

      let ary = [

        {

          id: 1,

          name: "Zhang",

        },

        {

          id: 2,

          name: "Wang",

        },

      ];

      let target = ary.find((*item*, *index*) => item.id === 2);

      console.log(target); //{id: 2, name: "Wang"}

    </script>

### （3） Array.findIndex()

查找第1個符合條件的元素在數組中的位置，如果沒有找到則返回-1

    <script>

      let ary = [1, 5, 9, 13, 22, 7];

      let index = ary.findIndex((*value*, *index*) => value > 9);

      console.log(index); //3 (index從0開始算)

    </script>

### （4） Array.includes()

表示某個數組是否包含給定的值，返回布爾值。

    <script>

      let incl1 = [1, 2, 3].includes(2);

      console.log(incl1); //true

      let incl2 = [1, 2, 3].includes(0);

      console.log(incl2); //false

    </script>

## 10 ES6中內置對象的擴展方法

### （1） 模板字符串

ES6新增的創建字符串的方式：使用反引號定義：

let name = `andy`

反引號`在Tab鍵上面

#### ① 模板字符串可以解析變量

    <script>

      let name = `Andy`;

      let sayHello = `Hello, my name is ${name}`;

      console.log(sayHello); //Hello, my name is Andy

    </script>

#### ② 模板字符串可以換行

    <script>

      let result = {

        name: "zhang",

        age: 20,

        gender: "f",

      };

      let html = `<div>

<span>${result.name}</span>

<span>${result.age}</span>

<span>${result.gender}</span>

</div>`;

      console.log(html);

    </script>

#### ③ 模板字符串可以調用函數

在調用函數的位置顯示函數的返回值。

    <script>

      const sayHello = function () {

        return "hello from fn";

      };

      let greeting = `${sayHello()}啦啦啦`;

      console.log(greeting); //hello from fn啦啦啦

    </script>

### （2） String的擴展方法

startsWith() 參數字符串是否在原字符串的頭部，返回布爾值。

endsWith() 參數字符串是否在原字符串的尾部，返回布爾值。

    <script>

      let str = `Hello world`;

      console.log(str.startsWith("hello")); //false

      console.log(str.startsWith("Hello")); //true

      console.log(str.endsWith("d")); //true

    </script>

repeat();

    <script>

      var str1 = "x".repeat(3);

      console.log(str1); //xxx

      var str2 = "hello".repeat(2);

      console.log(str2); //hellohello

    </script>

### （3） Set數據結構

ES6提供了新的數據結構Set。它類似於數組，但成員值都是唯一的，無重複值。

用途舉例：存儲用戶的搜索歷史記錄

Set是個構造函數，創建此構造函數的實例對象就是生成Set數據結構。

const s = new Set();

Set函數可以接受一個數組作為參數，用來初始化。

    <script>

      const set = new Set([1, 2, 3, 4, 4]);

      console.log(set); // {1, 2, 3, 4}

      const arr = [...set];

      console.log(arr); //[1, 2, 3, 4]

    </script>

### （4） Set對象實例方法

add(value)：添加某個值，返回Set結構本身，可以鏈式調用

delete(value)：刪除某個值，返回一個布爾值，表示是否刪除成功

has(value)：返回一個布爾值，表示該值是否為Set成員

clear()：清除所有成員，沒有返回值

const s = new Set();

s.add(1).add(2).add(3); //向set結構中添加值

s.delete(2); //刪除set結構中的2值

s.has(1); //看set結構中是否有1這個值，返回布爾值

s.clear(); //清除set結構中的所有值

遍歷set

Set結構的實例與數組一樣，也擁有forEach方法，對於每個成員執行某種操作，沒有返回值。

    <script>

      const s = new Set(["a", "b", "c"]);

      s.forEach((*value*) => console.log(value));

    </script>

——————

編程邏輯思維能力

案例：攜程移動端輪播圖，京東移動端輪播圖，品優購輪播圖，基於ECharts的數據可視化項目。