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故事保留字。
* 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：3

    </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，怎執行語句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

}

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();

## 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 { }

java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，Java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>

WebAPI（DOM操作）

jQuery

面向對象編程

ES6語法

ECharts

熟悉Echarts即可，因為有的公司用，有的公司不用。

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。