JavaScript

網頁程式設計

Darren Yang 楊德倫

2021

目次

[1. 測試及除錯工具 1](#_Toc72782798)

[2. 常數和變數宣告 8](#_Toc72782799)

[3. 運算子 14](#_Toc72782800)

[4. 字串 19](#_Toc72782801)

[5. 取得標籤元素 21](#_Toc72782802)

[6. 流程控制 26](#_Toc72782803)

課程範例網址:

<https://github.com/telunyang/javascript_iii_bd>

# 1. 測試及除錯工具

開啟開發工具

* F12

檢查元素



圖: 檢查元素按鈕



(圖) 檢查元素

開發工具的位置



(圖) 按下三個點的圖示，也可以選擇 dock side



(圖) 切換 dock side，從下方到右側

切換裝置



(圖) 等同按下切換裝置工具欄



(圖) 可選擇不用的行動裝置，或自訂寬高，來顯示網頁



(圖) 選擇裝置來觀看網頁

Console面板

我們可以使用 Console 面板，進行 JavaScript 的開發測試與除錯。



(圖) Console 面板

常用快速鍵：

* Ctrl + R 或 F5 刷新頁面
* Ctrl + F5 清除快取後，刷新頁面(重新從伺服器端請求下載 HTML)
* Ctrl + L 清除 Console
* Shift + Enter 在 Console 中斷行(或多行)

以下是使用的範例：

* 使用 console.log 直接輸出



(圖) 輸出結果

* 使用 %s 輸入字串



(圖) 第二個字串引數值，可以帶入 %s 當中

備註: 若需要換行，可以在輸入第一行以後，按下 Shift + Enter，即可換行

* 使用 es6 的 template strings



(圖) 透過 template 進行字串輸出

* 使用 %c 輸出時，加入樣式 (css style)



(圖) 設定 css style 的文字輸出

* 使用 %i 伴隨字串格式來輸出數字的值



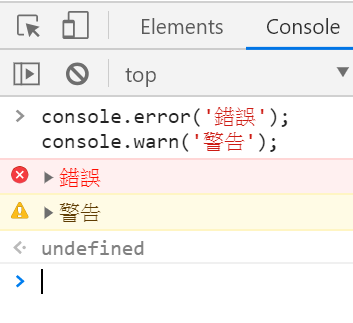
(圖) 輸出數字的值

常見的 % (格式化字串所使用的符號)：

* %s: 輸入字串
* %i 或 %d: 輸入數值
* %c: css style

若有特殊的訊息(例如錯誤、警告等)要顯示出來，可以使用 console.error、console.warn：

* console.error() JavaScript 出現錯誤訊息時，會出現紅色的文字訊息與圖示
* console.warn() JavaScript 出現警告訊息時，會出現黃色的文字訊息與圖示



(圖) 顯示結果會包括圖示

我們可以開啟 Preserve log，讓每次刷新頁面時，還能保有先前的輸出訊息。這個部分很重要，因為未來在使用非同步傳輸（Ajax、Fetch）結合Web API的時候，在資料傳輸上有問題，可以在刷新頁面、重新執行程式碼後，保留先前的輸出結果，讓我們在 debug 上更為便利。



圖：開啟 Preserve log

Network面板

Network 面板會顯示出所有網路請求的詳細訊息記錄，包括狀態、資源類型、大小、所需時間、HTTP request header 和 response header 等等，明確找出哪些請求比預期還要耗時，並加以調整，是優化網頁的重要工具。



(圖) Network面板會記錄任何的網路活動



(圖) 記錄網頁讀取的資訊與下載順序

我們可以透過 Headers，來了解網頁請求的狀況。開啟 Headers 的流程為：

1. 開啟 Network 面板
2. Ctrl + R 或是 F5 刷新頁面
3. 點選左側的檔案名稱
4. 點選 Headers



(圖) 觀看檔案的 Headers 內容

Request Headers (請求標頭，參考[維基百科](https://zh.wikipedia.org/wiki/HTTP%E5%A4%B4%E5%AD%97%E6%AE%B5))

| **標頭欄位** | **說明** | **範例** |
| --- | --- | --- |
| Accept | 能夠接受的回應內容類型（Content-Types）。 | Accept: text/plain |
| Cookie | 之前由伺服器通過 Set-Cookie傳送的一個 超文字傳輸協定Cookie | Cookie: \_ga=GA1.3.1322956465.1572335045;locale=zh\_TW; \_gid=GA1.3.1110994946.1584940974; \_gat\_gtag\_UA\_141775379\_1=1 |
| Content-Type | 請求多媒體類型 （用於POST和PUT請求中） | Content-Type: application/x-www-form-urlencoded |
| User-Agent | 瀏覽器的[瀏覽器身分標識字串](https://zh.wikipedia.org/wiki/%E7%94%A8%E6%88%B7%E4%BB%A3%E7%90%86) | User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:12.0) Gecko/20100101 Firefox/21.0 |

# 2. 常數和變數宣告

我們可以透過變數名稱的宣告，來為「值」（文字、數值、運算結果、暫存資料等）取個名字。我們使用關鍵字 var、let、const 來宣告變數，例如 var myName，而所謂「關鍵字」，是指在程式語言中具有意義的單詞，通常不適合拿來作為變數命名使用。原則上，變數的賦值，通常**由右往左**，我們稱之為「賦值運算子」（Assignment operators）。

變數命名的規則，有很多種方式，例如駱駝命名法、匈牙利命名法等。駱駝命名法是一種慣例，看起來像駱駝的駱峰，變數以小寫字母開頭，除了第一個單詞外，其它單詞的首個字母都大寫，例如 numberOfCandies，或是我們上方的 myName，或是 numberOfDays。課程中，原則上使用駱駝命名法，但沒有特別強制規定，你也可以**自由選擇自己慣用的方式**。

var、let 和 const

var、let、const 都是宣告變數的關鍵字，var 跟 let 都是宣告一般自訂變數，而 const 是用來宣告自訂常數（顧名思義，常數原則上不可修改它的值）。

|  |
| --- |
| 範例 2-1.html |
| <!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>          var myName = 'Darren';          console.log('我的名字: ' + myName);          let myAge = 18;          console.log('我的年紀: ' + myAge);          const PI = 3.1415926;          console.log('圓周率 = ' + PI);      </script>  </head>  <body>  </body>  </html> |

const 關鍵字用於「常數」的宣告，有別於我們先前看到的「變數」，原則上不宜重覆賦值，也常用大寫的方法與變數加以區隔，例如 const MAX\_LENGTH、DB\_HOST、DB\_PASSWORD等。

識別字的規則

識別字 (identifier) 為程式語言中依程式需求自行定義的名稱，舉凡程式中所用的各種名稱都屬於識別字，內建物件 (built-in object) 及 DOM 其內所用的名稱也屬於識別字，自行定義名稱時應該避免與其相衝突，同時，識別字也不可與保留字 (reserved words) 的名稱相同。

JavaScript 定義識別字可用任何 Unicode 符號，但是習慣上仍是以英文二十六的大小寫字母為主，另加上數字、底線符號及 dollar sign ，如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \_ | $ |  |  |  |  |  |  |  |  |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m |
| n | o | p | q | r | s | t | u | v | w | x | y | z |
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |  |  |

|  |
| --- |
| 範例 2-2-1.html |
| <!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>          var \_b;          var $a;          var $3;          var 3cats;      </script>    </head>  <body>    </body>  </html> |

|  |
| --- |
| 補充說明 |
| 我們另外補充說明「**保留字**」。在 JavaScript 中，有些識別字是保留給程式語言用的關鍵字，不能拿來作為變數名稱或是函式名稱。 |

我們應該避免使用 JavaScript 關鍵字：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | arguments | boolean | break | byte |
| case | catch | char | class\* | const |
| continue | debugger | default | delete | do |
| double | else | enum\* | eval | export\* |
| extends\* | false | final | finally | float |
| for | function | goto | if | implements |
| import\* | in | instanceof | int | interface |
| let | long | native | new | null |
| package | private | protected | public | return |
| short | static | super\* | switch | synchronized |
| this | throw | throws | transient | true |
| try | typeof | var | void | volatile |
| while | with | yield |  |  |

我們應該避免使用 JavaScript 的物件、屬性和方法名稱：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Date | eval | function | hasOwnProperty |
| Infinity | isFinite | isNaN | isPrototypeOf | length |
| Math | NaN | name | Number | Object |
| prototype | String | toString | undefined | valueOf |

我們在之後會應用到Window物件，我們也要避免使用到它的屬性和方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **alert** | **all** | **anchor** | **anchors** | **area** |
| **assign** | **blur** | **button** | **checkbox** | **clearInterval** |
| **clearTimeout** | **clientInformation** | **close** | **closed** | **confirm** |
| **constructor** | **crypto** | **decodeURI** | **decodeURIComponent** | **defaultStatus** |
| **document** | **element** | **elements** | **embed** | **embeds** |
| **encodeURI** | **encodeURIComponent** | **escape** | **event** | **fileUpload** |
| **focus** | **form** | **forms** | **frame** | **innerHeight** |
| **innerWidth** | **layer** | **layers** | **link** | **location** |
| **mimeTypes** | **navigate** | **navigator** | **frames** | **frameRate** |
| **hidden** | **history** | **image** | **images** | **offscreenBuffering** |
| **open** | **opener** | **option** | **outerHeight** | **outerWidth** |
| **packages** | **pageXOffset** | **pageYOffset** | **parent** | **parseFloat** |
| **parseInt** | **password** | **pkcs11** | **plugin** | **prompt** |
| **propertyIsEnum** | **radio** | **reset** | **screenX** | **screenY** |
| **scroll** | **secure** | **select** | **self** | **setInterval** |
| **setTimeout** | **status** | **submit** | **taint** | **text** |
| **textarea** | **top** | **unescape** | **untaint** | **window** |

我們應該避免使用 HTML 的事件名稱：

|  |  |  |  |
| --- | --- | --- | --- |
| onblur | onclick | onerror | onfocus |
| onkeydown | onkeypress | onkeyup | onmouseover |
| onload | onmouseup | onmousedown | onsubmit |

var 和 let 的主要差異

Javascript 透過 var 關鍵字宣告的變數，具有**抬升（Hoisting）**的性質，不易掌握變數的生命週期，同時 var 所宣告的全域變數，會成為 window 物件的屬性，我們日後會在課堂中提到。近年的 Javascript 版本，增加了 let 與 const 關鍵字，解決了 var 關鍵字的抬升問題（Hoisting）。

我們來看看 var 在區域變數中的特性：

一般來說，我們會這樣宣告變數、初始化變數（給變數初始值）、輸出變數內容

var name = 'Darren';

console.log(name);

//輸出 Darren

換個方式

console.log(name);

var name = 'Darren';

//這裡會輸出 undefined（Console 面板會不顯示輸出）

若是這樣子呢

console.log(name);

let name = 'Darren';

//應該輸出 undefined，卻拋出 **ReferenceError: name is not defined** 的訊息



(圖) 參考錯誤: name 變數沒有被定義

這是因為 Javascript 在使用var 宣告變數時，會**將變數抬升（hoisting）到作用域（程式區塊）的前面**，作用域內外的每一行都有機會使用到。

在JavaScript中有八種主要的型別：

* 三種基本型別：
* 布林(Boolean)
* 數值(Number)
* 字串(String)
* 兩種複合的型別：
* 陣列(Array)
* 物件(Object)
* 兩種簡單型別：
* 空值(null)
* 未定義(undefined)
* 一種特殊型別：
* 函式(Function)

# 3. 運算子

算術運算子

JavaScript 可以進行加（+）、減（-）、乘（\*）、除（/）的基本運算，傳統的數學計算概念，也可以在程式當中運作，例如先乘除、後加減等等的概念；關於下面的基本運算，我們稱為「算數運算子」（Arithmetic operators）。

| 運算子 | 範例 |
| --- | --- |
| 加 (+) | 1 + 2 = 3 |
| 減 (-) | 3 - 1 = 2 |
| 乘 (\*) | 3 \* 4 = 12 |
| 除 (/) | 8 / 2 = 4 |
| 取餘數  (%) | 12 % 5 = 2. |
| 遞增  (++) | 假如  x是 3，那  ++x  將把 x 設定為 4 並回傳 4，而  x++ 會回傳  3 ， 接著才把  x 設定為 4。 |
| 遞減  (--) | 假如  x是  3，那  --x  將把 x 設定為 2 並回傳 2，而  x-- 會回傳  3 ， 接著才把  x 設定為  2。 |
| (一元運算子) 減號  (-) | 假如x是3，-x 回傳 -3。 |
| (一元運算子) 加號  (+) | +"3"  回傳 3；+true  回傳 1. |
| 指數運算子  (\*\*) | 2 \*\* 3  回傳 8. |

|  |
| --- |
| 範例 3-1.html |
| <!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>          let x = 10;          let y = 5;          console.log(x + y); //15          console.log(x - y); //5          console.log(x \* y); //50          console.log(x / y); //2          console.log(x % y); //0          x++;          console.log(x); //11          y--;          console.log(y); //4          console.log(y\*\*3); //64      </script>  </head>  <body>  </body>  </html> |

關係運算子

以相互比較的方式，來呈現布林邏輯運算結果，稱之為「關係運算子」或「比較運算子」，例如常常提到的「大於 >、大於等於 >=、等於 ==（或 ===）、小於 <、小於等於 <=、不等於 !=」等概念。

|  |
| --- |
| 範例 3-2.html |
| <!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>          let price\_a = 100;          let price\_b = 50;          let price\_c = 100;          console.log(price\_a > price\_b);  //true          console.log(price\_a < price\_b);  //false          console.log(price\_a == price\_b); //false          console.log(price\_a >= price\_b); //true          console.log(price\_a <= price\_b); //false          console.log(price\_a > price\_c);  //false          console.log(price\_a < price\_c);  //false          console.log(price\_a == price\_c); //true          console.log(price\_a >= price\_c); //true          console.log(price\_a <= price\_c); //true      </script>  </head>  <body>    </body>  </html> |

|  |
| --- |
| 補充說明 |
| 「==」vs.「===」 相等運算子  有時候閱讀程式設計教課書，在討論是否等價這件事，有著不同的寫法，如同上面的「==」與「===」，兩個的差別在於：  「x == y」：若是型態不相等，變數會先強制轉換成相同的型態，再進行嚴格比對。  console.log(1 == 1); // 輸出 true  console.log("1" == "1"); // 輸出 true  console.log(1 == "1"); // 輸出 true  console.log("1" == 1); // 輸出 true  「===」：兩個型態不相等，**不**轉換型態，直接嚴格比對。**最常用的等價邏輯判斷方式**，若為 true，意味著兩個值相同，類型也相等。  console.log(1 === 1); // 輸出 true  console.log("1" === "1"); // 輸出 true  console.log(1 === "1"); // 輸出 false  console.log("1" === 1); // 輸出 false |

邏輯運算子

布林提供了 true (真) 與 false (假) 的判斷值，透過邏輯運算子（&&、||、!）來進行操作。

|  |
| --- |
| 範例 3-3-1.html |
| <!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>          let a = true;          let b = false;          console.log(a && b);          //輸出 false          let x = true;          let y = false;          console.log(x || y);          // 輸出 true          let z = true;          let result = !z;          console.log(result);          // 輸出 false      </script>  </head>  <body>  </body>  </html> |

賦值運算子的種類，常見的有以下幾種，提供給大家參考：

|  |  |  |
| --- | --- | --- |
| 名稱 | 簡化後運算子 | 說明 |
| 賦值 | x = y | x = y |
| 加法賦值 | x += y | x = x + y |
| 減法賦值 | x -= y | x = x - y |
| 乘法賦值 | x \*= y | x = x \* y |
| 除法賦值 | x /= y | x = x / y |
| 餘數賦值 | x %= y | x = x % y |
| 指數賦值 | x \*\*= y | x = x \*\* y |
| 左移賦值 | x <<= y | x = x << y |
| 右移賦值 | x >>= y | x = x >> y |
| 無號右移賦值 | x >>>= y | x = x >>> y |
| 位元 AND 賦值 | x &= y | x = x & y |
| 位元 XOR 賦值 | x ^= y | x = x ^ y |
| 位元 OR 賦值 | x |= y | x = x | y |

|  |
| --- |
| 範例 3-3-2.html |
| <!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>          //宣告變數          let x = 20;          let y = 10;          //將 x 加上 y 的結果，再賦值到 x 當中          x += y; //可以寫成 x = x + y;          console.log(x);          //變數初始化          x = 2, y = 8;          y \*= x; //可以寫成 y = y \* x;          console.log(y);          //變數初始化          x = 15, y = 4;          x %= y; //可以寫成 x = x % y;          console.log(x);      </script>  </head>  <body>  </body>  </html> |

# 4. 字串

字串的標示方式

* 透過兩個「'」將字串圍起來
* 透過兩個「"」將字串圍起來
* 透過 Template Literals 或 Template Strings 合併字串

|  |
| --- |
| 範例 4-1.html |
| <!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>          //宣告字串          let strName = 'Darren';          //用單引號圍住字串的方式輸出          console.log('Hello World! ' + strName + '...');          //用雙引號圍住字串的方式輸出          console.log("I love javascript! " + strName + "...");          //用 template strings 的方式輸出          console.log(`My name is ${strName} ...`);          //輸出 你好! 我是 Darren Yang          let fname = 'Darren';          let lname = 'Yang';          console.log(`你好! 我是 ${fname} ${lname}`);          //若是嵌入變數計算，可以這麼做          let a = 5;          let b = 10;          console.log(`${a + b}, ${2 \* a + b}`);      </script>  </head>  <body>  </body>  </html> |

字串的跳脫表示法

|  |
| --- |
| 範例 4-2.html |
| <!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>          //為了避免字串當中有單引號          console.log('Did you talk to Mary\'s brother?');          /\*\*           \* 綜合使用           \* \t : 縮排，等同於 tab 鍵           \* \n : 斷行，代表建立新的一行(new line)           \*/          console.log("嗨!\t\t\t\t近來好嗎?\n好久不見");      </script>  </head>  <body>  </body>  </html> |

# 5. 取得標籤元素

我們來解析一下 DOM。



(圖) DOM 的樹狀結構，裡面是由節點、樹葉、連結所組成

在 DOM 中，每個元素(element)、文字(text) 等等都是一個節點(node)，而節點通常分成以下四種：

* Document
  + Document 就是指這份文件，也就是這份 HTML 檔的開端，所有的一切都會從 Document 開始往下進行。
* Element
  + Element 就是指文件內的各個標籤，因此像是 <div>、<p> 等等各種 HTML Tag 都是被歸類在 Element 裡面。
* Text
  + Text 就是指被各個標籤包起來的文字，舉例來說在 <h1>Hello World</h1> 中，Hello World 被 <h1> 這個 Element 包起來，因此 Hello World 就是此 Element 的 Text
* Attribute
  + Attribute 就是指各個標籤內的相關屬性，例如 class、id、data-\* 等。



(圖) DOM 圖解

由於 DOM 為樹狀結構，樹狀結構最重要的觀念就是 Node 彼此之間的關係，這邊可以分成以下兩種關係：

* 父子關係(Parent and Child)
  + 簡單來說就是上下層節點，上層為 Parent Node ，下層為 Child Node 。
* 兄弟關係(Siblings)
  + 簡單來說就是同一層節點，彼此間只有 Previous 以及 Next 兩種。



(圖) table元素 - 節點間的關係

|  |
| --- |
| 補充說明 |
| window 物件（可以想成瀏覽器本身）代表了一個包含 DOM 文件的視窗，其中的 document 屬性指向了視窗中載入的 Document 物件：   * 代表所有在瀏覽器中載入的網頁，也是作為網頁內容 DOM 樹（包含如 <body>、<table> 與其它的元素）的進入點。 * 提供了網頁文件所需的通用函式，例如取得頁面 URL 或是建立網頁文件中新的元素節點等。 |

以下為常用的 DOM API：

* document.getElementById('idName')
  + 找尋 DOM 中符合此 id 名稱的元素，並回傳相對應的 element。
* document.getElementsBytagName('tag')
  + 找尋 DOM 中符合此 tag 名稱的所有元素。
* document.getElementsByClassName('className')
  + 找尋 DOM 中符合此 class 名稱的所有元素。
* document.querySelector(selectors)
  + 利用 selector 來找尋 DOM 中的元素，並回傳相對應的第一個 element。
* document.querySelectorAll(selectors)
  + 利用 selector 來找尋 DOM 中的所有元素。

| 補充說明 | | |
| --- | --- | --- |
| **選擇器** | **例子** | **例子描述** |
| **.class** | .intro | 選擇 class="intro" 的所有元素。 |
| **.class1.class2** | .name1.name2 | 選擇 class 屬性中同時有 name1 和 name2 的所有元素。 |
| **.class1 .class2** | .name1 .name2 | 選擇作為類名 name1 元素後代的所有類名 name2 元素。 |
| **#id** | #firstname | 選擇 id="firstname" 的元素。 |
| [**\***](https://www.w3school.com.cn/cssref/selector_all.asp) | \* | 選擇所有元素。 |
| [**element**](https://www.w3school.com.cn/cssref/selector_element.asp) | p | 選擇所有 <p> 元素。 |
| [**element.class**](https://www.w3school.com.cn/cssref/selector_element_class.asp) | p.intro | 選擇 class="intro" 的所有 <p> 元素。 |
| [**element,element**](https://www.w3school.com.cn/cssref/selector_element_comma.asp) | div, p | 選擇所有 <div> 元素和所有 <p> 元素。 |
| [**element element**](https://www.w3school.com.cn/cssref/selector_element_element.asp) | div p | 選擇 <div> 元素內的所有 <p> 元素。 |
| [**element > element**](https://www.w3school.com.cn/cssref/selector_element_gt.asp) | div > p | 選擇父元素是 <div> 的所有 <p> 元素。 |
| [**[attribute]**](https://www.w3school.com.cn/cssref/selector_attribute.asp) | [target] | 選擇帶有 target 屬性的所有元素。 |
| [**[attribute=value]**](https://www.w3school.com.cn/cssref/selector_attribute_value.asp) | [target=\_blank] | 選擇帶有 target="\_blank" 屬性的所有元素。 |
| [**[attribute~=value]**](https://www.w3school.com.cn/cssref/selector_attribute_value_contain.asp) | [title~=flower] | 選擇 title 屬性包含單詞 "flower" 的所有元素。 |
| [**[attribute^=value]**](https://www.w3school.com.cn/cssref/selector_attr_begin.asp) | a[href^="https"] | 選擇其 href 屬性值以 "https" 開頭的每個 <a> 元素。 |
| [**[attribute$=value]**](https://www.w3school.com.cn/cssref/selector_attr_end.asp) | a[href$=".pdf"] | 選擇其 href 屬性以 ".pdf" 結尾的所有 <a> 元素。 |

參考資料：

CSS 選擇器參考手冊

<https://www.w3school.com.cn/cssref/css_selectors.asp>

querySelector()

|  |
| --- |
| 範例 5-1.html |
| <!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>  </head>  <body>      <p class="myClass">問世間，</p>      <p class="myClass">情是何物，</p>      <p class="myClass">直教生死相許。</p>      <script>          //這裡是回傳第一個 p，是一種 element          let p = document.querySelector('p');          p.style.backgroundColor = '#7878ff';      </script>  </html> |

querySelectorAll()

|  |
| --- |
| 範例 5-2.html |
| <!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>  </head>  <body>      <p class="myClass">問世間，</p>      <p class="myClass">情是何物，</p>      <p class="myClass">直教生死相許。</p>      <script>          let p = document.querySelectorAll('p');          //可以使用類似陣列的操作方式，提供索引值來存取          p[0].style.backgroundColor = '#ff0000';          p[1].innerHTML = '<span style="color: #00ff00">...情是何物...</span>';          p[2].style.lineHeight = '24px';          p[2].style.backgroundColor = '#0000ff';          p[2].style.color = '#ffffff';      </script>  </body>  </html> |

# 6. 流程控制

我們把條件與廻圈，稱之為「流程控制」或「控制結構」，對於任何程式，都扮演著重要的角色。它們讓你定義的特定條件，控制在何時、何種頻率來執行哪些部分的程式碼。

選擇敘述

if 陳述句、if ... else 陳述句、if...else 陳述句鏈，它是依據條件做二選一區塊執行時，所使用的陳述句：

|  |
| --- |
| 範例 6-1-1.html |
| <!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>          let condition = false;          if (condition) {              //若條件為真，則執行這個區塊的程式碼              console.log('Hello World!');              document.write('Hello World!');          } else {              //若條件為假，則執行這個區塊的程式碼              console.log('May you have a nice day!');              document.write('May you have a nice day!');          }      </script>  </head>  <body>  </body>  </html> |

|  |
| --- |
| 範例 6-1-2.html |
| <!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>          let number = 7;          if (number > 10) {              console.log(1);          } else if (number === 7) {              console.log(2);          } else if (number > 5) {              console.log(3);          } else if (number === 6) {              console.log(4);          }      </script>  </head>  <body>  </body>  </html> |

|  |
| --- |
| 範例 6-1-3.html |
| <!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>          let bool = (1 > 2) ? true : false;          console.log(bool);          // 輸出 false            let x = 3;          let y = 4;          let answer = (x > y) ? 'x is greater than y' : 'x is less than y';          console.log(answer);          // 輸出 x is less than y      </script>  </head>  <body>    </body>  </html> |

switch 判斷

|  |
| --- |
| 範例 6-1-4.html |
| <!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>          let number = 7;          switch(number){              case '7':                  console.log('string 7');              break;                case 7:                  console.log('number 7');              break;                case 'number':                  console.log('string number');              break;                default:                  console.log('string default');          }          // 輸出 number 7      </script>  </head>  <body>    </body>  </html> |

迴圈

while 迴圈結合了條件判斷的概念，符合條件，則繼續執行區塊內的程式，直到條件不成立，才跳出程式區塊。以下提供範例來說明：

while 迴圈

|  |
| --- |
| 範例 6-2-1.html |
| <!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>          let count = 1;          while(count <= 7){              console.log(count);              count++;          }          // 輸出 1 2 3 4 5 6 7 (console.log 會斷行)      </script>  </head>  <body>    </body>  </html> |

for 迴圈

|  |
| --- |
| 說明 |
| for(init**;** condition**;** increment) {  //statements  }  init 是變數初始宣告的地方  condition 是變數狀態與判斷條件  increment 是變數賦值的方式  for(let i = 0; i < 10; i++) {  console.log('The value is ' + i);  }  // 輸出 The value is 0 …. The value is 9  迴圈內部也可以使用迴圈，通稱為「巢狀迴圈」  for(let i = 1; i <= 9; i++) {  for(let j = 1; j <= 9; j++){  console.log(i + ' \* ' + j + ' = ' + (i\*j));  }  }  for 的無限迴圈形式：  for(;;){  console.log('Hello World!');  }  while 的無限迴圈形式  while(1){ // 將 1 改成 true 亦同  …  } |

|  |
| --- |
| 範例 6-2-2.html |
| <!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>          for(let i = 0; i < 10; i++) {              console.log('The value is ' + i);          }          // 輸出 The value is 0 …. The value is 9      </script>  </head>  <body>    </body>  </html> |

|  |
| --- |
| 範例 6-2-3.html |
| <!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>      for(let i = 1; i <= 9; i++) {          for(let j = 1; j <= 9; j++){              //在 Console 面板輸出結果              console.log(`${i} \* ${j} = ${i\*j}`);          }      }      for(let i = 1; i <= 9; i++) {          for(let j = 1; j <= 9; j++){              //在網頁輸出結果              document.write(`${i} \* ${j} = ${i\*j} &nbsp;`);          }          document.write('<br />');      }      </script>  </head>  <body>    </body>  </html> |

break和continue

通常我們使用 break 來停止、跳出迴圈運作，直接往 for 或 while 迴圈block 結尾以後的程式區塊繼續執行；使用 continue 直接跳往下一個索引值的步驟繼續執行。

|  |
| --- |
| 範例 6-3-1.html |
| <!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>          //i 到 4 的時候，跳出迴圈          for(let i = 1; i <= 9; i++) {              if(i == 4){                  break;              }              console.log(`i = ${i}`);          }          /\*\*           \* j 為 4 的時候，只跳出內部迴圈，           \* 但外部迴圈依然會繼續執行，           \* 只有內部迴圈的 j 走到 4 時候，           \* 自行跳出           \*/          for(let i = 1; i <= 9; i++) {              for(let j = 1; j <= 9; j++){                  if(j == 4){                      break;                  }                  console.log(`i = ${i}, j = ${j}`);              }          }      </script>  </head>  <body>    </body>  </html> |

|  |
| --- |
| 範例 6-3-2.html |
| <!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>      let count = 0;      while(count <= 9){          //count 遞增到 5 的時候，跳出迴圈          if(count == 5){              break;          }          console.log(`count = ${count}`);          count++; //千萬記得要遞增，不然會變成無限迴圈      }      //i 為 5 的時候略過，直接往下一個 i (即是 6) 繼續執行      for(let i = 1; i <= 9; i++) {          if(i == 5){              continue;          }          console.log(`i = ${i}`);      }      //count 為 5 的時候略過，直接往下一個 count (即是 6) 繼續執行      count = 0;      while(count <= 9){          count++;          if(count == 5){              continue;          }          console.log(`count = ${count}`);      }      </script>  </head>  <body>    </body>  </html> |