JavaScript程式設計

張庭禎 老師

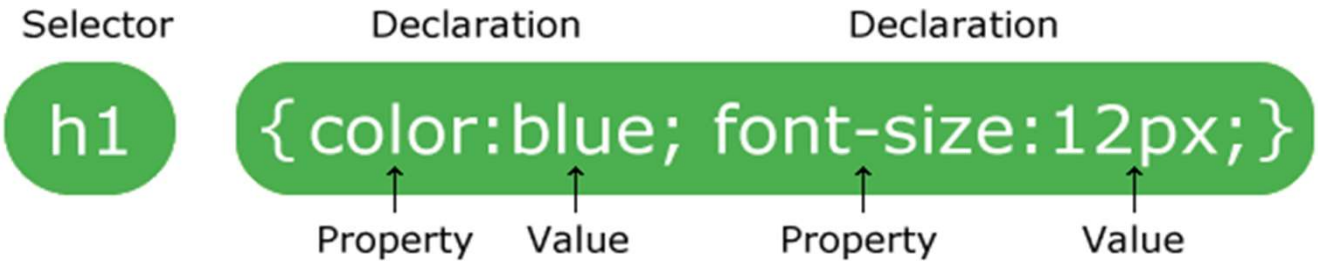
JavaScript程式設計

- HTML CSS 基本語法
- JavaScript 基本語法
- 認識物件
- 事件驅動
- Javascript form
- jQuery 框架

基本HTML範例

```
<!DOCTYPE html>
<html>
<head>
      <meta charset="utf-8"/>
      <title></title>
      <style></style>
</head>
<body>
  <p>Hello CSS!</p>
  <p>These paragraphs are styled with CSS.</p>
</body>
</html>
```

CSS 基本語法



- Selector(選擇器) - 通常是元素、類別或ID
- Declaration(宣告) - 由Property(屬性)和Value(值)所組成，宣告間以分號區隔，最外圍用大括號括住。

CSS 套用方法(1)

元素標籤後面

```
<p style="color:red; font-size:20px;">
```

CSS 套用方法(2)

放在<head>的<style>中

```
<head>
   <style>
      p {
         width: 200px;
         height: 40px;
         background: #b6ff00;
      }
   </style>
</head>
```

CSS 套用方法(3)

- 外部載入將CSS格式獨立寫成*.css檔
- 從HTML內部載入外部的檔案即可，寫法是在<head>使用<link>
- 在<style>中使用@import載入外部檔案

```
<head>
   <link href="basis.css" rel="stylesheet"/>
</head>
<head>
   <style type=text/css>
      @import url(basis.css);
   </style>
</head>
```

class選擇器

```
<style>
    .name
    {
        color: red;
    }
</style>
<body>
    <p class="name" ></p>
</body>
```

ID選擇器

```
<style>
   #id01
   {
      color: red;
   }
</style>
<body>
   <p id="id01" >Test</p>
</body>
```

群組選擇器

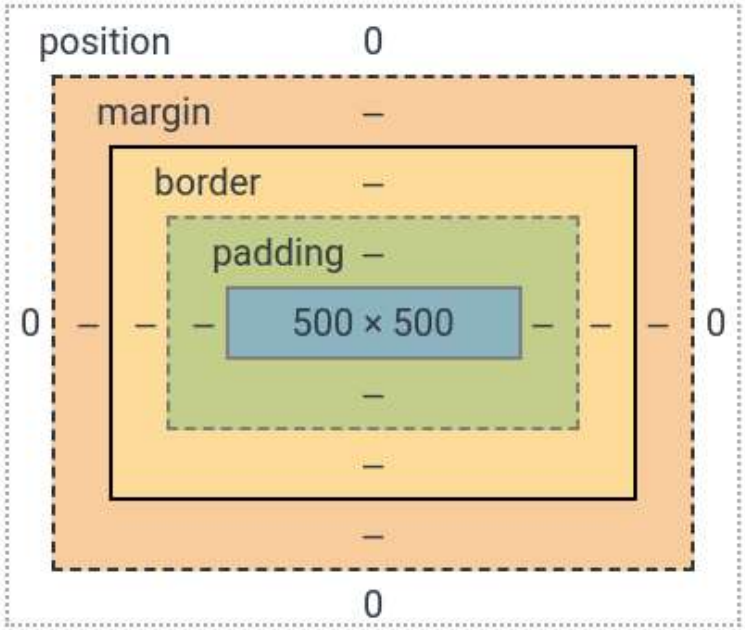
```
<style>
    h2,p
    {
        color: red;
    }
</style>
<body>
    <h2>Heading 2</h2>   <!--這項套用紅字-->
    <p>Paragraph</p>     <!--這項套用紅字-->
</body>
```

CSS 背景

background-color:顏色值　　設定背景顏色

background-image:url(圖片檔案路徑)　設定背景圖片

CSS 版面定位



margin-top:屬性值	設定上方外距
margin-bottom:屬性值	設定下方外距
margin-left:屬性值	設定左方外距
margin-right:屬性值	設定右方外距
margin:上 右 下 左	設定全方向外距(簡便寫法)

JavaScript 語言

- JavaScript (簡稱 JS) 被用於 Web 瀏覽器的腳本語言 (scripting language)
- JavaScript 屬於直譯語言 (interpreted language)，不需事先編譯 (compile)，直接在瀏覽器 (browser) 上執行。

JavaScript 如何使用

- <script></script> 標籤在瀏覽器執行。
- <script> 標籤可以放在網頁 HTML 的任何地方像是 <body> 或 <head> 中
- 需告語法 <script type="text/javascript" language="javascript">

```
<!doctype html>
<html><head>
<script>
    alert('Hello world!');
</script>
</head>
<body>
   My first JavaScript page!
</body>
</html>
```

引用 JavaScript 檔案

- 瀏覽器遇到 <script> 標籤時，會停止解析 HTML 文件，會先執行 <scipt> 裡面的 JS 程式碼等到執行完程式碼，才會繼續解析接下來的 HTML 文件。
- <script src="/hello.js"></script>
- <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
- 可以將 JS 程式碼和 HTML 分開，才不會雜在一起很難閱讀和查找。
- 不同邏輯的 JavaScript 程式碼，可以分開不同的檔案編寫和儲存，方便維護和管理

JavaScript 變數

```
function foo() {
  // 在 function 變數作用(存在)範圍只在 function 裡面
    var carName = 'Ferrari';
    alert(carName); // 會顯示 Ferrari
}
alert(carName); // 會發生錯誤，因為找不到變數
```

```
// carName 是一個全域變數
var carName = 'Ferrari';

function foo() {
   alert(carName); // 會顯示 Ferrari
}
alert(carName); // 會顯示 Ferrari
```

JavaScript 資料型態

- 基本資料型態包含了：
  - 布林值 (Boolean): 只包含兩種值 true / false
  - null: null 是一個特殊值 (keyword)，表示這變數裡面沒有記憶體
  - undefined: 表示值還沒有定義或還未指定
  - 數值 (Number): 數值類型的值，像是 12 / 3.14
  - 字串 (String): 像是 'hello world' / 汽車
- 複合資料型態包含了：
  - 陣列 (Array): 陣列用來儲存多個資料
  - 物件 (Object): 基本資料型態以外的都是物件型態

JavaScript Number

- NaN (Not a Number)

```
isNaN(NaN);        // true
isNaN(undefined); // true
isNaN({});         // true

isNaN(true);       // false
isNaN(null);       // false
isNaN(20);         // false

var v1=parseInt(x)
var v2=parseFloat(y)
```

型別判讀 typeof

- 運算子typeof用來判斷一個運算元是什麼資料型態。
- // 輸出 string
- console.log(typeof 'hello');
- // 輸出 number
- console.log(typeof 123);
- // 輸出 boolean
- console.log(typeof true);

運算子	例子	說明
==	3 == var1 '3' == var1	如果兩邊相等就返回 true
!=	var1 != 4 var2 != '3'	如果兩邊不相等就返回 true
===	3 === var1	跟 == 的差異在於，=== 不會自動嘗試轉型，型態和值都一樣才會返回 true
!==	var1 !== '3' 3 !== '3'	跟 != 的差異在於，!== 不會自動嘗試轉型，型態或值不一樣都會返回 true
>	var2 > var1 var1 > 2	如果左邊運算元大於右邊的就返回 true
>=	var2 >= var1 var1 >= 3	如果左邊運算元大於或等於右邊的就返回 true
<	var2 < var1 var1 < 2	如果左邊運算元小於右邊的就返回 true
<=	var2 <= var1 var1 <= 3	如果左邊運算元小於或等於右邊的就返回 true

If 語法

JavaScript 只有對下面這些值會判斷為 false
其他都是 true：

1. 布林值 false
2. undefined
3. null
4. 數值 0
5. NaN
6. 空字串 ''

If 語法

```
var text = '';
if (text) {
    alert(true);
} else {
    alert(false);
}
```

迴圈語法

- JavaScript提供三種迴圈語法
  - while：

```
while (GuestCheck( ))
{
  numberOfGuess += 1;
}
```

  - do while：

```
do {
  eatTimes++;
} while (StillHungry( ))
```

  - for：

```
for (var i=0; i<10; i++) {
  Count( );
}
```

迴圈語法(1)

```
var n = 0;
var x = 0;
while (n <= 10) {
    n++;
    x += n;
}
```

迴圈語法(2)

```
var x = 0;
for(var n=1;n<=10;n++){
    x += n;
}
```

彈出式視窗

```
alert('歡迎來到 google.com');

var yes = confirm('你確定嗎？');
if (yes) {
    alert('你按了確定按鈕');
} else {
    alert('你按了取消按鈕');
}

var nickname = prompt('請輸入你的暱稱');
alert('Hello ' + nickname);
```

迴圈猜數字

```
var rnd=parseInt(Math.random()*100)+1;
  var guess=0;
  while(rnd != guess){
     guess= prompt("Guess 1~100:");
     if(guess>rnd)
       alert(guess+" too big");
      else if(guess<rnd)
        alert(guess + " too small");
      else{
          alert("Bingo");
          break;
      }
  }
```

函數表達式

- Function expression 宣告函數，將匿名函數當作值指定給一個變數。

```
var square = function(number) {
    return number * number;
};
```

- 一般函數宣告

```
function square(number) {
    return number * number;
}
```

JavaScript Array (陣列)

- 陣列中的每一個值我們稱做一個元素
- 每一個元素儲存在陣列中固定的位置我們稱做索引 (index)，索引值從 0 開始，表示陣列中的第一個元素，第二個之後的元素索引值則依序加 1 (0, 1, 2, 3, ...)。
- 語法：

```
var arrayName = [item1, item2, ...];
```

- 例如：

```
var fruits = ['Apple', 'Banana'];
```

新增陣列元素

- push() 方法來新增元素到陣列最後面：

```
var fruits = ['Apple', 'Banana'];
fruits.push('Orange');
// 輸出 ["Apple", "Banana", "Orange"]
console.log(fruits);
```

- unshift() 方法來新增一個元素到陣列最前面：

```
var fruits = ['Apple', 'Banana'];
fruits.unshift('Orange');
// 輸出 ["Orange", "Apple", "Banana"]
console.log(fruits);
```

讀取陣列中的元素

```
var fruits = ['Apple', 'Banana'];
// Apple
var first = fruits[0];
// Banana
var last = fruits[fruits.length - 1];
```

刪除陣列元素

- pop() 方法來移除陣列中的最後一個元素：

```
var fruits = ['Apple', 'Banana'];
// pop() 除了移除元素，還會返回移除的元素值
var last = fruits.pop(); // Banana
// 輸出 ["Apple"]
console.log(fruits);
```

- shift() 方法來移除陣列中的第一個元素：

```
var fruits = ['Apple', 'Banana'];
// shift() 除了移除元素，還會返回移除的元素值
var first = fruits.shift(); // Apple
// 輸出 ["Banana"]
console.log(fruits);
```

陣列搜尋 indexOf()

- indexOf() 方法用來找出一個值出現在陣列中的哪個位置

語法：

```
ary.indexOf(searchElement)
ary.indexOf(searchElement, fromIndex)
```

```
var ary = [2, 6, 9];
// 返回 0
i=ary.indexOf(2);
// 返回 -1
i=ary.indexOf(7);
```

陣列 splice(1)

- 插入、刪除或替換陣列中的某一個或某個範圍的元素。
- 語法：

```
ary.splice(start)
ary.splice(start, deleteCount)
ary.splice(start, deleteCount, newElement1, newElement2, ...)
```

```
var fruits = ['Banana', 'Orange', 'Apple', 'Mango', 'Peach'];
var removed = fruits.splice(2, 2);
// 輸出目前 ["Banana", "Orange", "Peach"]
console.log(fruits);
// 輸出已經刪除 ["Apple", "Mango"]
console.log(removed);
```

陣列 splice(2)

刪除並新增元素

```
var fruits = ['Banana', 'Orange', 'Apple', 'Mango’ , 'Peach'];
var removed = fruits.splice(2, 2, 'Watermelon', 'Lemon');

// 輸出 ["Banana", "Orange", "Watermelon", "Lemon", "Peach"]
console.log(fruits);

// 輸出刪除 ["Apple", "Mango"]
console.log(removed);
```

JavaScript Object

- 宣告一個物件：

```
var myObj = new Object();
```

- {} 可以宣告一個物件：

```
var myObj = {};
```

- 物件的屬性

```
var myObj = {};
// 建立一個叫 color 的屬性，值是 blue
myObj.color = 'blue';
// 存取物件屬性
var myColor = myObj.color;
```

JavaScript Object Method

```
var user = {
    firstName: 'Mary',
    lastName: 'Lee',
    age: 30,
    fullName: function() {
        return this.firstName + ' ' + this.lastName;
    }
}

// name = 'Mary Lee'
var name = user.fullName();
```

JavaScript BOM

- 瀏覽器物件模型(BOM-Browser Object Model)

window: 讓你可以存取操作瀏覽器視窗

location: 讓你可以存取操作頁面的網址 (URL)

Timer: 讓你可以使用瀏覽器內建的計時器

cookie: 讓你可以管理瀏覽器的 cookie

Window物件

```
var windowObj= window.open(
    'http://tw.yahoo.com/',
    'yahoo’,
    'width=800,height=600,
     resizable=no,scrollbars=yes,
     status=no,location=no'
);
```

Location物件

- location.href

```
// 切換到另一個網站
location.href = 'https://www.google.com/';
```

- 取得當前網頁的網址路徑 location.pathname

Timer物件

- setTimeout() 用來設定一段時間過後，自動執行某個函數 (callback)，這計時器只會執行一次就停止
- setInterval() 用來設定每過一段時間，就自動執行某個函數 (callback)，這計時器會永遠一直執行下去
- clearInterval() 取消 setInterval()

Timer物件範例

```
var timeoutID = window.setTimeout(myAlert, 5000);
function myAlert() {
    alert('五秒鐘到了！');
}

var intervalID = window.setInterval(function() {
    alert('3秒鐘又到了！');
}, 3000);
```

Cookie物件

```
function btnSave(){
    var d = new Date();
    d.setTime(d.getTime() + ( 60 * 60 ));
    expires = "expires=" + d.toTimeString();
    alert(expires);
    document.cookie = 'username=Mary; '+expires+'; path=/';
  }
  function btnRead() {
    var cookieAry = document.cookie.split(';');
    for(var i=0;i< cookieAry.length;i++){
      alert(cookieAry[i]);
    }
  }
```

JSON資料格式

- 資料格式 名稱/值
- 資料值是以逗號做分隔
- { } 表示建立的物件
- [ ] 表示建立的物件陣列
- JavaScript程式有標準內建函式轉換JSON資料成JavaScript物件

製作JSON物件

- JSON格式建立物件方式

```
var attendees =
   {
      "name": "Eric Chang",
      " age":  20
   }
```

- 物件欄位 name , age
- 資料值 Eric Chang , 20

使用JSON定義物件陣列

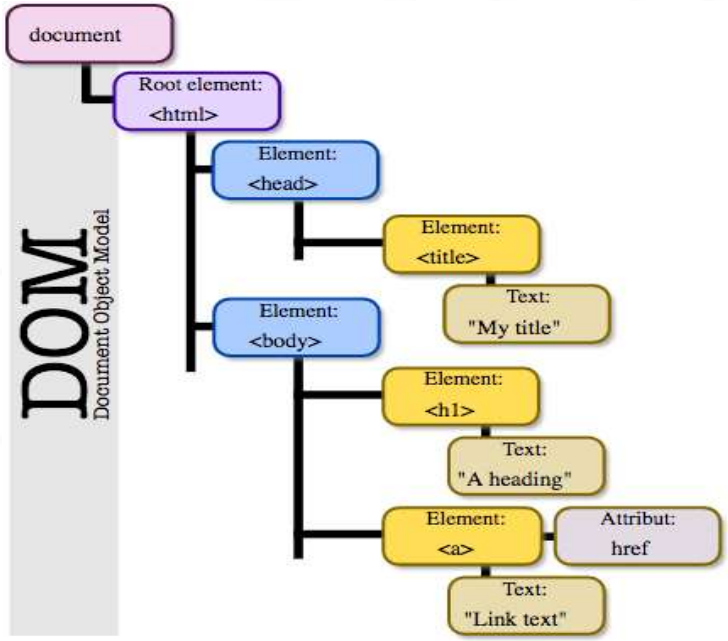
- JSON格式建物物件陣列

```
var attendees = [
    {
        "name": "Eric Gruber",
        " age": "18"
    },
    {
        "name": "Martin Weber",
        " age": "28"
    }
]
```

- JavaScript內建函式轉換JSON資料

DOM (Document Object Model)

- 定義了一組標準 API 讓我們可以用 JavaScript 對 HTML 文件做操作。
- 定義了HTML元素有哪些屬性 (properties) 可以來做存取
- 定義了HTML元素有哪些方法 (methods) 可以來被操作
- 定義了HTML元素事件讓我們可以針對特定元素來綁定事件處理函式



DOM 查找元素(1)

- document.getElementById 用來根據 id 取得 HTML元素。

```
<html>
<head>
  <title>getElementById example</title>
</head>
<body>
  <p id="anyText">Some text here</p>
</body>
</html>
var elem = document.getElementById('anyText');
```

DOM 查找元素(2)

```
function div1Element() {
    // 取得 id 為 div1 的 HTML 元素
    var div1 = document.getElementById('div1');

    // 取得 div1 下面所有的 <p> 元素
    var div1Param = div1.getElementsByTagName('p');

    // 取得 div1 下面總共有幾個 <p> 元素
    // num = 5
    var num = div1Param.length;

    // .....
}
```

```
function div2Elememt() {
    // 取得 id 為 div1 的 HTML 元素
    var div2 = document.getElementById('div2');

    // 取得 div2 下面所有的 <p> 元素
    var div2Param =
div2.getElementsByTagName('p');

    // 取得 div2 下面總共有幾個 <p> 元素
    // num = 2
    var num = div2Param.length;

    // .....
}
```

getElementsByClassName(name)

```
<div id="parent-id">
    <p>hello word1</p>
    <p class="test">hello word2</p>
    <p class="test">hello word3</p>
</div>

// 取得 id 為 parent-id 的 HTML 元素
var parentDOM = document.getElementById('parent-id');
// 取得 parentDOM 下面所有 class 是 test 的 HTML 元素
var test = parentDOM.getElementsByClassName('test');
// 輸出 2
console.log(test.length)
// 輸出 <p class="test">hello word2</p>
console.log(parentDOM.getElementsByClassName('test')[0].innerTerxt);
```

Node.childNodes

```
<body>
        <!- 以下標籤不可以換行 不然節點會出現undefined --!>
        <div id="foo"><p>P1</p><span>Span1</span><p>P2</p></div>
</body>
    <script>
        // 取得 id 為 foo 的元素
            var foo = document.getElementById('foo');

            // 元素的 hasChildNodes() 方法可以用來判斷一個元素下有沒子元素
            if (foo.hasChildNodes()) {
                // 取得 foo 元素的所有子元素集合
                var children = foo.childNodes;
                // 可以用 for 迴圈來遍歷每一個子元素
                for (var i = 0; i < children.length; ++i) {
                    // 用 children[i] 來取得遍歷到子元素
                    console.log("log1 "+children[i].innerHTML);
                    console.log("log2 " +children[i].nodeName);
                }
            }
    </script>
```

Node.firstChild

```
<p id="foo"> <!- 有換行 --!>
  <span>First span</span>
</p>
<script>
    var p = document.getElementById('foo');
    // 會顯示 "#text"，因為第一個子元素是空白的文字節點
    alert(p.firstChild.nodeName);
</script>
<!- 沒有換行 --!>
<p id="foo"><span>First span</span></p>
<script>
    var p = document.getElementById('foo');
    // 會顯示 "SPAN"
    alert(p.firstChild.nodeName);
</script>
```

Node.lastChild

```
<p id="foo"><span>First span</span><span>Second
span</span><span>Last span</span></p>

<script>
    var p = document.getElementById('foo');
    // 會顯示 "Last span"
    alert(p.lastChild.innerHTML);
</script>
```

nextSibling previousSibling

```
<div><span id="s1">s1</span><span id="s2">s2</span></div>

<script>
    // 會顯示 null
alert(document.getElementById('s1').previousSibling);
    // 會顯示 "s1"
alert(document.getElementById('s2').previousSibling.id);
</script>
// 會顯示 "s2"
  alert(document.getElementById('s1').nextSibling.id);
 // 會顯示 null
  alert(document.getElementById('s2').nextSibling);
```

DOM 修改節點的操作

```
<div id="outer">
  <div id="inner">blah</div>
</div>
<script>
    var outerDiv = document.getElementById('outer');
    var divs = document.getElementsByTagName('div');
    // 顯示 2
    alert(divs.length);
    // 清空 outer 下的節點
    outerDiv.innerHTML = '';
    // 顯示 1
    alert(divs.length);
</script>
```

DOM 新增節點的操作

```
<ul id="firstUL">
</ul>
<script>
    var head = document.getElementById('firstUL');
    for(var i=1;i<=3;i++){
      li01 = document.createElement("li");
      li01.innerHTML = "顯示的文字 "+i;
      head.appendChild(li01);
    }

</script>
```

DOM新增文件標籤

- 首先找尋要新增標籤根元素
- 新增新標籤元素資料
- 新增新新標籤元素資料到根元素中

```
<ul id="VList">
    <li>Room A</li>
    <li>Room B</li>
</ul>
var list=document.getElementById("VList");
var newItem=document.createElement("li");
newItem.textContent="Room C";
list.appendChild(newItem);
```

DOM 刪除節點的操作

```
<ul id="firstUL"><li>1</li><li>2</li><li>3</li></ul>
<script>
    var head = document.getElementById('firstUL');
    // 刪除 <li> 3 </li>
    head.removeChild(head.childNodes[2]);

</script>
```

Node屬性-Node.nodeValue

```
<div id="foo">hello world</div>
<script>
    var div = document.getElementById('foo');

    // 顯示 hello world
    alert(div.firstChild.nodeValue);

    // 顯示 foo
    alert(div.attributes.id.nodeValue);
</script>
```

Node屬性-innerHTML

```
<div id="foo"><span>hello world</span> 101</div>
<script>
    var div = document.getElementById('foo');
    // 顯示 <span>hello world</span> 101
     alert(div.innerHTML);

    // 將 div 的內容改成 123
    div.innerHTML = '123';

    // 顯示 123
    alert(div.innerHTML);
</script>
```

Node屬性-innerText

```
<div id="foo"><span>hello world</span> 101</div>
<script>
    var div = document.getElementById('foo');
    // 顯示 hello world 101
    alert(div.innerText);
</script>
<script>
    var div = document.getElementById('foo');
    // 將 div 的內容改成 123
    div.innerText = '<span>one</span><span>two</span>';
    // 顯示 &lt;span&gt;one&lt;/span&gt;&lt;span&gt;two&lt;/span&gt;
    alert(div.innerHTML);
</script>
```

Node- appendChild

```
<div id="foo"><span>hello</span></div>
<script>
    // 建立一個新 <div>
    var newDiv = document.createElement('div');
    // 建立一個新的文字節點
    var newContent = document.createTextNode('I love gjun.com!');
    // 將文字節點加到剛建立的 <div> 元素中
    newDiv.appendChild(newContent);

    // 取得目前頁面上的 foo 元素
    var currentDiv = document.getElementById('foo');
    // 將剛建立的 <div> 元素加入 foo 元素中
    currentDiv.appendChild(newDiv);

    // 顯示 <span>hello</span><div>I love gjun.com!</div>
    alert(currentDiv.innerHTML);
</script>
```

Node- insertBefore

```
<div id="foo"><span id="s1">hello</span><span id="s2">world</span></div>
<script>
    // 建立一個新的 <span>
    var newSpan = document.createElement('span');
    // 增添一些內容
    newSpan.innerHTML = 'my new span text';

    // 取得目前頁面上的 foo 元素
    var foo = document.getElementById('foo');
    // 取得目前頁面上的 s2 元素
    var s2 = document.getElementById('s2');

    // 將新的 span 元素放到 foo 元素中的 s2 子元素前面
    foo.insertBefore(newSpan, s2);

    // 顯示 <span id="s1">hello</span><span> my new span text </span><span
id="s2">world</span>
    alert(foo.innerHTML);
</script>
```

insertBefore移動元素的位置

```
<div id="foo"><span id="s1">hello</span><span id="s2">world</span></div>

<script>
    // 取得目前頁面上的 foo 元素
    var foo = document.getElementById('foo');

    // 取得目前頁面上的 s1 元素
    var s1 = document.getElementById('s1');
    // 取得目前頁面上的 s2 元素
    var s2 = document.getElementById('s2');

    // 將 s2 元素放到 foo 元素中的 s1 子元素前面
    foo.insertBefore(s2, s1);

    // 顯示 <span id="s2">world</span><span id="s1">hello</span>
    alert(foo.innerHTML);
</script>
```

removeChild 移除 DOM 節點

```
<div id="container"><div id="nested">12</div>34</div>
<script>
    // 取得目前頁面上的 container <div> 元素
    var container = document.getElementById('container');

    // 取得目前頁面上的 nested <div> 元素
    var nested = document.getElementById('nested');

    // 從頁面上移除 nested <div> 元素
    // removeChild 方法會返回被移除的元素
    var garbage = container.removeChild(nested);
    // 顯示 <div id="nested">12</div>
    alert(garbage.outerHTML);
    // 顯示 34
    alert(container.innerHTML);
</script>
```

DOM CSS

```
<p id="foo">hello world</p>
<script>
    var foo = document.getElementById('foo');
    // 將 <p> 的文字字體顏色改成綠色
    foo.style.color = 'green';
    // 將 <p> 的背景顏色改成灰色
    foo.style.background = 'gray';
</script>
<script>
    var foo = document.getElementById('foo');
    // 將背景顏色改為紅色
    foo.style['background-color'] = '#f00';
    // 將 margin-top 設為 100px
    foo.style.marginTop = '100px';
</script>
```

DOM cssText

```
<p id="foo">hello world</p>
<script>
    var foo = document.getElementById('foo');
    // 輸出 "" 空字串，因為 foo 上沒有設定 style 屬性
    alert(foo.style.cssText);

    // 將 foo 的字體大小設為 20px、字體顏色設為紫色
    foo.style.cssText = 'font-size: 20px; color: purple;';

    // 輸出 font-size: 20px; color: purple;
    alert(foo.style.cssText);
</script>
```

DOM-cssProperty

```
<style>
#elem {
    position: absolute;
    left: 100px;
    top: 200px;
    height: 100px;
}
</style>
<div id="elem" style="top:50px;">CSS Property Test</div>
<script>
    var elem = document.getElementById('elem');
    // 顯示 100px
    alert(elem.currentStyle.height);
    // 顯示 50px
    alert(elem.currentStyle.top);
</script>
```

DOM getAttribute 屬性

```
<a id="foo" href="http://www.gjun.com/" target="_blank" data-foo>
www.gjun.com</a>
<script>
    // 取得目前頁面上的 foo 元素
    var foo = document.getElementById('foo');

    // 顯示 null
    alert(foo.getAttribute('xyz'));

    // 顯示 http://www.gjun.com/
    alert(foo.getAttribute('href'));

    // 顯示 _blank
    alert(foo.getAttribute('target'));

    // 顯示 "" 空字串
    alert(foo.getAttribute('data-foo'));
</script>
```

DOM setAttribute 屬性

```
<a id="foo" href="http://www.gjun.com/">www.gjun.com</a>

<script>
    // 取得目前頁面上的 foo 元素
    var foo = document.getElementById('foo');

    // 顯示 null
    alert(foo.getAttribute('target'));

    // 將 target 屬性設定為 _blank
    foo.setAttribute('target', '_blank');

    // 顯示 _blank
    alert(foo.getAttribute('target'));
</script>
```

DOM onclick 事件

```
<html>
<head>
  <title>inline event handling example</title>
</head>
<body>
  <button onclick="triggerAlert(this);" data-name="Mike">click
me</button>
  <script>
    function triggerAlert(em) {
      alert('Hey ' + em.getAttribute('data-name'));
    }
  </script>
</body>
</html>
```

DOM onclick 事件

```
// 當使用者用滑鼠點擊螢幕 (或用手點擊觸控螢幕) 時
document.body.addEventListener('click', function(event) {
    // 把字體改成黃色
    event.target.style.color = 'yellow';
});
```

DOM addEventListener事件

```
<script>
    function myAlert() {
      alert('Hey!');
    }

    // 當使用者用滑鼠點擊頁面時，執行 myAlert 函數
    document.addEventListener('click', myAlert);

    // 當網頁載入時，執行這個 callback 函數
    window.addEventListener('load', function() {
      alert('頁面已載入！');
    });
  </script>
```

DOM Key 事件

```
// 當使用者在網頁中按下鍵盤按鍵時
document.onkeydown = function(event) {
    if (event.keyCode === 89 && event.ctrlKey) {
         alert('你同時按下 "control + y"');
     } else if (event.which === 90 && event.ctrlKey ){
         alert('你同時按下 "control + z"');
     }
};
```

DOM 滑鼠事件

- DOM定義了事件程序可以透過使用者來觸發
- 可以動態加入標籤事件程序

```
scene.addEventListener("mouseover",
    function( ) { window.alert('Some help text'); },
  false);
```

- HTML標籤可以定義回呼函式

```
var scene=document.getElementById("scene");
document.images.scene.onmouseover=
    function( ) { window.alert('Scene text'); };
```

HTML5表單清單資料元件

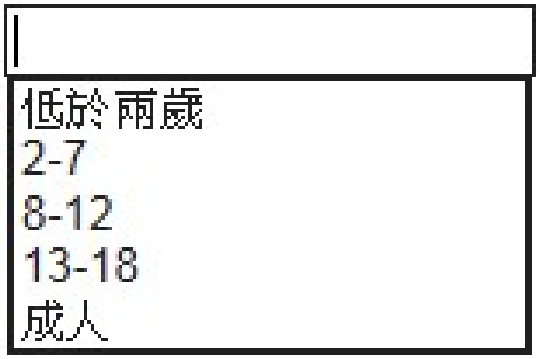
- optgroup

```
<select id="carManufacturer" name="carManufacturer">
  <optgroup label="歐洲車">
    <option value="volvo">Volvo</option>
    <option value="audi">Audi</option>
  </optgroup>
  <optgroup label="美國車">
    <option value="chrysler">
        Chrysler</option>
    <option value="ford">
        Ford</option>
  </optgroup>
</select>
```

HTML5表單清單資料元件

- <datalist>

```
<input id="ageCategory" name="ageCategory"
list="ageRanges" />
<datalist id="ageRanges">
<option value="低於2歲"></option>
<option value="2 - 7"></option>
<option value="8 - 12"></option>
<option value="13-18"></option>
<option value="成人"></option>
</datalist>
```



HTML5表單輸入屬性

- 表單輸入屬性能夠修飾驗證輸入的資料內容

- autocomplete

```
<input name="password" type="password" autocomplete="off" />
```

- required

```
<input id= "contactNo" name= "contactNo" type= "tel" placeholder
="Enter your phone number" required="required" />
```

- pattern

```
<input id="orderRef" name="orderRef" type="text"
    pattern="[0-9]{2}[A-Z]{3}" title="2 digits and 3 uppercase letters" />
```

利用JavaScript驗證輸入資料

- 在表單<form>標籤設定攔截事件
- <form action="test.aspx" onsubmit="return check( )" />
- 在check( ) javascript程序中執行資料驗證檢查，如果通過驗證就傳回true、不通過就傳回false

XMLHttpRequest物件存取遠端資料

- 傳送HTTP請求過程：
  1. 產生XMLHTTPRequest物件
  2. 指定HTTP方法及網址(URL)
  3. 設定請求資料表頭資料
  4. 送出請求

```
var request = new XMLHttpRequest( );
var url = "http://server.com/resources/...";
request.open( "GET", url );
request.send( );
```

- 請求會以非同步方式送出到伺服器

```
request.open( "GET", url , true);
```

XMLHttpRequest非同步存取遠端資料

- 建立javascript XMLHTTPRequest物件
- 指定伺服器端要執行程式網址(URL)
- 建立事件處理readystatechange程序
- 讀取傳回物件response的屬性responseText
- 建立伺服器端程式準備傳回之資料
- 最後送出請求執行

```
<div id="user"></div>
<script>
// 建立 XMLHttpRequest 物件
var httpRequest;
httpRequest = new XMLHttpRequest();
// AJAX callback
httpRequest.onreadystatechange = function() {
// 等狀態變成請求完成狀態
if (httpRequest.readyState === 4) {
// 只處理 server 返回正常的 HTTP 200 狀態
if (httpRequest.status == 200) {
// 解開 server 返回的 JSON 資料格式
var jsonResponse = JSON.parse(httpRequest.responseText);
// 更新頁面內容
document.getElementById('user').innerHTML =  jsonResponse.userName;
} else {
alert('ERROR - server status code: ' + httpRequest.status);
}
}
};
// 使用 HTTP GET 方法，從 URL {username:'Mary'}請求資料
httpRequest.open('GET', 'user.txt');
// 送出 HTTP 請求
httpRequest.send();
</script>
```

XMLHttpRequest傳送資料

- 傳送資料到伺服器端：
  1. 設定XMLHttpRequest資料表頭屬性
     Content-Type = application/x-www-form-urlencoded
  2. 透過HTTP POST方法傳送資料

```
var data = "fname=John&lname=Lee";
var request = new XMLHttpRequest( );
var url = …;
request.open("POST", url, true);
request.setRequestHeader("Content-Type",
                         "application/x-www-form-urlencoded");
request.send(data);
```

jQuery 前端程式庫特色

- jQuery 是一套物件導向式簡潔輕量級的 JavaScript程式庫
- 透過 jQuery 你可以用最精簡的程式碼來達到跨瀏覽器 DOM 操作、事件處理、設計頁面元素動態效果、AJAX 互動等
- jQuery 已經幫你作好了 (Edge, Firefox, Safari, Opera, Chrome)
- 支援 CSS3 選擇器
- <script src="//ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

jQuery基本觀念

- **jQuery 程式碼由 $ 開始後面會接著刮號「()」而刮號裡面的參數是你想 jQuery 幫你找什麼 (取得哪個(些)元素) 執行什麼動作 (或處理事件)。例如：**

```
// 選取 id 為 em 的元素，並綁定 onclick 事件
// 叫 jQuery 將其CSS的背景顏色屬性改成綠色
$('#em').click(function() {
  $('#em').css('background-color', 'green');
});
```

jQuery Tag selector(1)

- 語法 (Syntax)
  - $(selectors)
- <a>標籤選擇器 jQuery
  - $('a'); // 取得頁面中所有的 <a> 標籤元素
- <a>標籤在JavaScript DOM
  - document.getElementsByTagName('a');

jQuery class selector(2)

- 語法 (Syntax)
  - $(selectors)
- class 選擇器 jQuery
  - $('.item'); // 取得 class name 為 item 的所有元素
- 在JavaScript DOM
  - document.getElementsByClassName('item');

jQuery id selector(3)

- 語法 (Syntax)
  - $(selectors)
- id選擇器 jQuery
  - $('#el'); // 取得 id 為 el 的元素
- JavaScript DOM
  - document.getElementsById('el');

挑選 jQuery的標籤元素

- p標籤段落

```
$(document).ready(function( ){
  $("button").click(function( ){
    $("p").hide( );
  });
});
```

- p標籤段落

```
$("#test").hide( );
```

- p標籤段落

```
$(".test").hide( );
```

jQuery標籤套用CSS

- 改變標籤字型顏色

```
$(document).ready(function( ) {
    $("h2").each(function( ) {
        this.style.color = "red";
    });
});
```

- 改變第一個p標籤套用 "intro" CSS

```
$("button").click(function( ){
 $("p:first").addClass("intro");
});
```

jQuery方法與標籤

- css( )方法將所選擇標籤設定CSS屬性資料
  - $("button").click(function( ){
    $("p").css("color","red");
    });
- val( )取得或是設定標籤value屬性資料值
  - $("button").click(function( ){
    $(“#username").val("John");
    });

使用jQuery設定標籤內容

- This is a paragraph.
- $("#test1").text("Hello world!");
- $("#test1").html("Hello world!");
- <input type="text" id="test2" value="Edit Text">
- $("#test2").val("My JavaScript Text");

使用jQuery取得標籤內容

- some bold text
- alert("Text: " + $("#test").text( ));
- alert("HTML: " + $("#test").html( ));
- <input type="text" id="test" value="Edit Text">
- alert("Value: " + $("#test").val( ));

jQuery DOM 操作(1)

- jQuery.html() - 類似 JavaScript DOM 中的 innerHTML

```
// HTML
<div></div>

// jQuery
$('div').html('<p>Hello World</p>’ );

// 得到的結果
[<div><p>Hello World</p></div>]
```

jQuery DOM 操作(2)

- **jQuery.text() - 取得字串包含著所有匹配元素的純文字內容**

例如 HTML

```
<p><em>Test1.</em>Test2.</p><p>Test3</p>
```

jQuery

```
$('p').text();
```

得到的結果

Test1.Test2.Test3

- 設定所有匹配元素的純文字內容

string 裡面的 “<” 與 “>” 會自動被轉成 HTML 本文

jQuery.text(string)

jQuery DOM 操作(3)

- .append(content) - 在每個匹配的元素內部最後面加入內容 (內部插入)

例如 HTML

```
<p>I would like to say: </p>
```

```
// jQuery
$('p').append('<b>Hello</b>');
```

```
// 得到的結果
[<p>I would like to say: <b>Hello</b></p>]
```

.prepend(content) - 在每個匹配的元素內部最前面加入... (內部插入)

```
// 例如 HTML
<p>I would like to say: </p>
```

```
// jQuery
$('p').prepend('<b>Hello</b>');
```

```
// 得到的結果
[<p><b>Hello</b>I would like to say: </p>]
```

jQuery DOM 操作(4)

- .before(content) - 在每個匹配的元素前面加入... (外部插入)

```
//  HTML
    <p>I would like to say: </p>
// jQuery
    $('p').before('<b>Hello</b>');
// 結果
   [<b>Hello</b><p>I would like to say: </p>]
```

- .after(content) - 在每個匹配的元素後面加入... (外部插入)

```
//  HTML
    <p>I would like to say: </p>
// jQuery
   $('p').after('<b>Hello</b>');
// 結果
   [<p>I would like to say: </p><b>Hello</b>]
```

jQuery DOM 操作(5)

- .wrap(html) – 框住匹配到的元素

```
// HTML
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
// jQuery
$('.inner').wrap('<div class="new"></div>');
// 結果
<div class="container">
  <div class="new">
    <div class="inner">Hello</div>
  </div>
  <div class="new">
    <div class="inner">Goodbye</div>
  </div>
</div>
```

jQuery DOM 操作(6)

- .wrapAll(html) – 框住所有匹配的元素

```
// HTML
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
// jQuery
      $('.inner').wrapAll('<div class="new" />');
// 結果
<div class="container">
  <div class="new">
    <div class="inner">Hello</div>
    <div class="inner">Goodbye</div>
  </div>
</div>
```

jQuery DOM 操作(7)

- .wrapInner(html) – 進入匹配的元素內

```
// HTML
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
// jQuery
      $('.inner').wrapInner('<div class="new"></div>');
// 結果
<div class="container">
  <div class="inner">
    <div class="new">Hello</div>
  </div>
  <div class="inner">
    <div class="new">Goodbye</div>
  </div>
</div>
```

jQuery DOM 操作(8)

- .empty() – 清空匹配到的元素其所有子節點

HTML

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">Goodbye</div>
</div>
```

jQuery

```
$('.hello').empty();

// 結果
<div class="container">
  <div class="hello"></div>
  <div class="goodbye">Goodbye</div>
</div>
```

jQuery DOM 操作(9)

- .remove() – 刪除匹配到的元素其所有子節點

HTML

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">Goodbye</div>
</div>
```

jQuery

```
$('.hello').remove();
```

```
// 我們也可以多帶入一個 selector 參數，來過濾匹配到的元素
// 例如這樣寫會得到一樣的結果
$('div').remove('.hello');
```

```
// 結果
<div class="container">
  <div class="goodbye">Goodbye</div>
</div>
```

jQuery DOM 操作(10)

- .clone([true]) - 複製匹配元素的副本

HTML

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">Goodbye</div>
</div>
```

jQuery

```
    $('.hello').clone().appendTo('.goodbye');
```

結果

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">
    Goodbye
    <div class="hello">Hello</div>
  </div>
</div>
```

jQuery - 添加 CSS 類別

- jQuery 提供addClass()方法將 CSS 類別添加到匹配的 HTML 元素

```
<style>
  .big{ font-weight: bold; font-size:20px; }
  .small{ font-weight: normal; font-size:10px; }
</style>
<body>
<div class="container">
  <h2>jQuery addClass() Method</h2>
  <div class="hello">Hello</div>
  <div class="goodbye">Goodbye</div>
</div>
</body>
```

```
$( ".hello" ).addClass("big" );
$( ".goodbye" ).addClass("small" );
```

結果

```
<div class="hello big">Hello</div> <div class="goodbye small">Goodbye</div>
```

jQuery – 刪除 CSS 類別

- jQuery 提供removeClass()方法將 CSS 類別刪除匹配的 HTML 元素

```
<style>
  .big{ font-weight: bold; font-size:20px; }
  .small{ font-weight: normal; font-size:10px; }
</style>
<body>
<div class="container">
  <h2>jQuery addClass() Method</h2>
  <div class="hello big">Hello</div>
  <div class="goodbye small">Goodbye</div>
</div>
</body>
```

```
$( ".hello" ). removeClass("big" );
$( ".goodbye" ). removeClass("small" );
```

結果

```
<div class="hello">Hello</div> <div class="goodbye">Goodbye</div>
```

jQuery - 切換 CSS

```
<script>
  $(document).ready(function() {
    $("button").click(function(){
      $( ".hello" ).toggleClass("big" );
    });
  });
</script>
<style>   .big{ font-weight: bold; font-size:20px; }  </style>
</head>
<body>
  <div class="container">
    <h2>jQuery toggleClass() Method</h2>
    <div class="hello big">Hello</div>
    <div class="goodbye">Goodbye</div>
  </div> <br>
  <button>Toggle Class</button>
</body>
```

jQuery - 設置 CSS 屬性

- 採用第一個匹配的 的顏色，並使用 div 背景顏色更改所有 <p> 的文本顏色

```
<script>
  $(document).ready(function() {
    $("button").click(function(){
      var color = $("div").css("background-color");
      $("p").css("color", color);
    });
  });
</script>
<style>
  button{margin:10px;width:150px;cursor:pointer}
  div{ margin:10px;padding:12px; width:125px;}
</style>
</head>
<body>
  <p>Click the below button to see the result:</p>
  <div style="background-color:#9c9cff;">Blue</div>
  <div style="background-color:#93ff93;">Green</div>
  <button>Set CSS Property</button>
</body>
```

jQuery - 設置多個 CSS 屬性

- 使用單個 jQuery 方法css()在匹配的元素上套用多個 CSS 屬性

```
<script>
  $(document).ready(function() {
    $("button").click(function(){
      $("div").css({"background-color":"#fb7c7c", "font-size": "25px"});
    });
  });
</script>
<style>
  button{margin:10px;width:150px;cursor:pointer}
  div{ margin:10px;padding:12px; width:125px;}
</style>
</head>
<body>
  <p>Click the below button to see the result:</p>
  <div style="background-color:#9c9cff;">Blue</div>
  <div style="background-color:#93ff93;">Green</div>
  <button>Set CSS Property</button>
</body>
```