# JavaScript

# JavaScript課程大綱

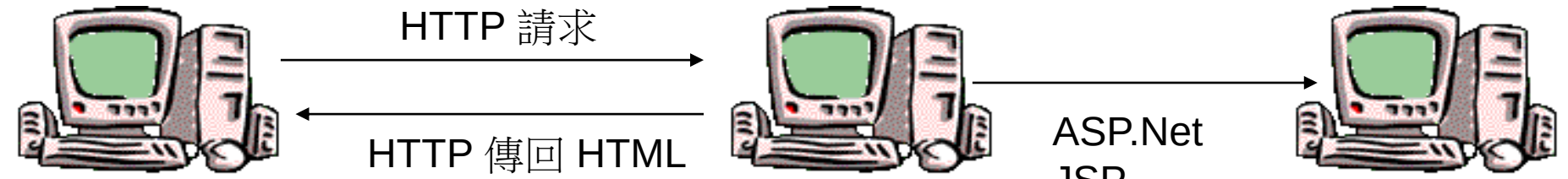
- 單元一：JavaScript簡介
- 單元二：資料型態與變數
- 單元三：運算子與敘述
- 單元四：函數
- 單元五：JavaScript物件
- 單元六：JavaScript的事件處理
- 單元七：Dynamic HTML
- 單元八：BOM 物件模型
- 單元九　：DOM 物件模型
- 單元十　：檔案處理
- 單元十一: Drag & Drop
- 單元十二: 網頁儲存區
- 單元十三: Geolocation API
- 附錄一　：Google Maps
- 附錄二　：Canvas 的使用

# 單元一：JavaScript簡介

- JavaScript 簡介
- JavaScript 的好處
- JavaScript 的基本架構
- JavaScript 的撰寫格式
- JavaScript 的輸出

# Web運作原理

- HTTP – HyperText Transfer Protocol
- HTML – HyperText Markup Language



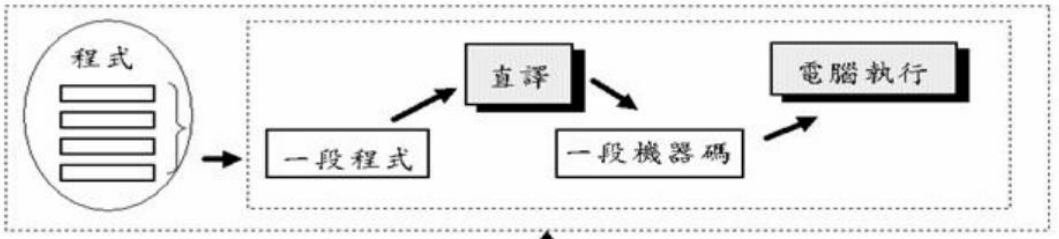
# JavaScript 簡介

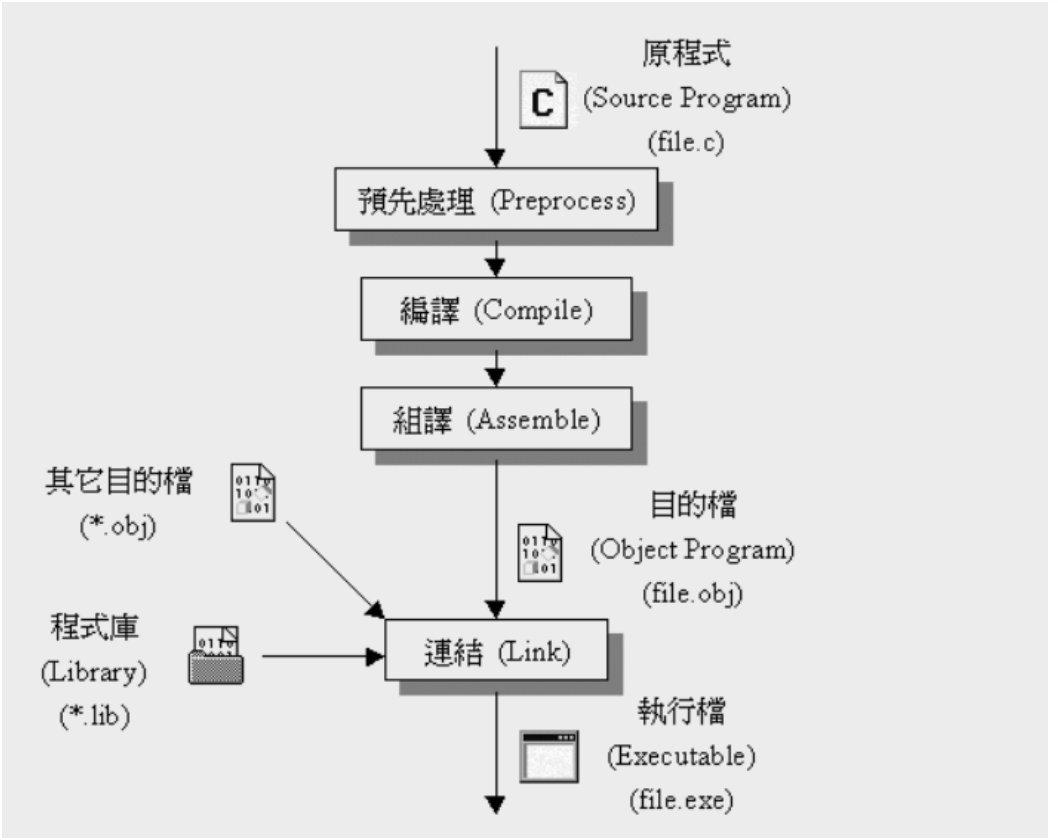
- 1995年由網景公司(Netscape co.)的Brendan Eich開發了一個名叫LiveScript的指令碼語言
- 1997年在歐洲電腦製造商協會(European Computer Manufacturers Association-ECMA International)協調下，確定統一標準:ECMA-262，稱為ECMAScript
- 是一種直譯程式(interpreter)，無須編譯器(compiler)
- 是一種以物件及事件驅動為基礎的程式語言
- 必須要直接嵌入HTML文件中，才可執行
- JavaScript包括三大部分:ECMAScript、Browser Object Model與Document Object Model
- 規格書:http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

# 直譯器與編譯器

- 直譯器(interpreter)：將原始程式碼一行一行或數行直接轉譯執行

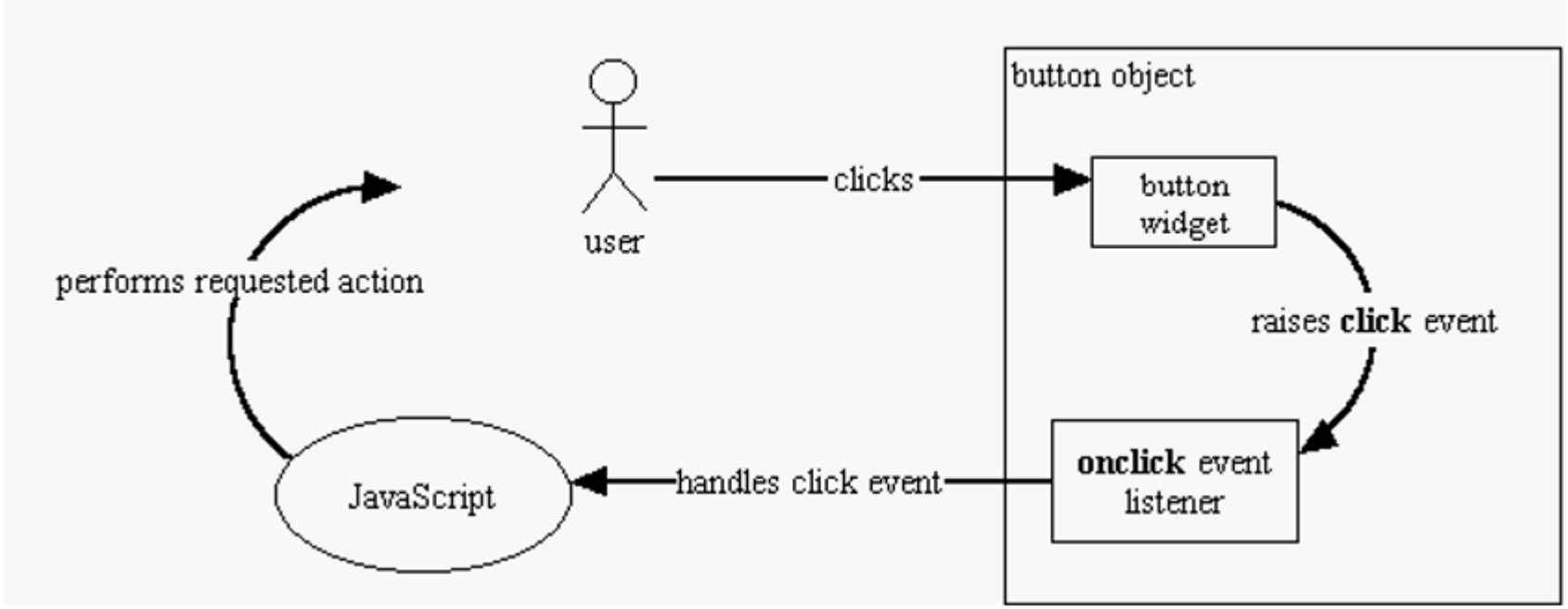
- 編譯器(compiler)： 將原始程式碼(source code)→ 預處理器(preprocessor) → 編譯器(compiler)→ 組譯程式(assembler) →目的碼(object code)→ 連結器(Linker)→ 執行檔(executables)，可直接執行。



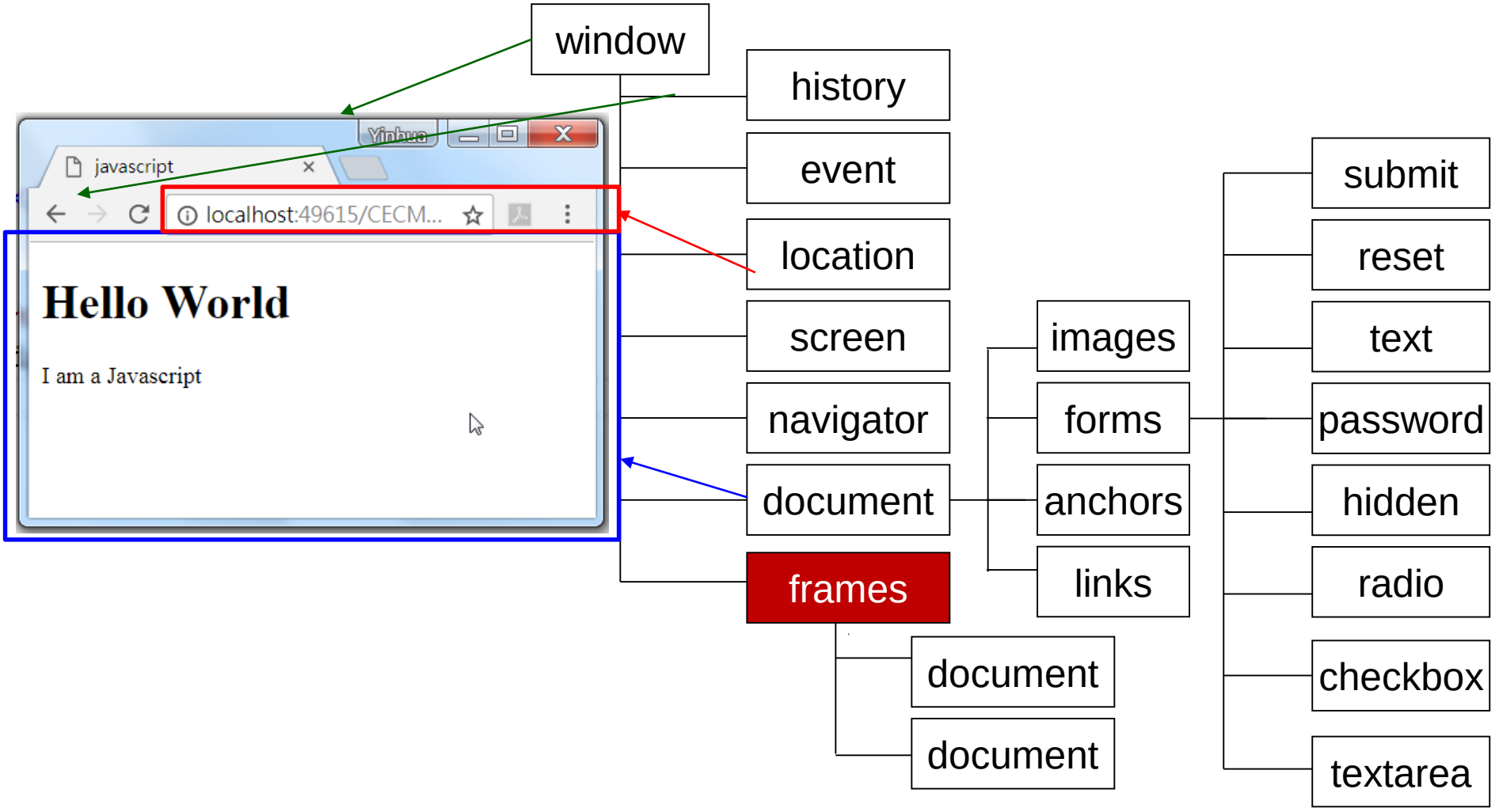



# 事件處理程序

```
<input type="button" value="ClickMe" onclick="window.alert('JavaScript');"/>
```



# BOM 物件模型架構(Browser Object Model)

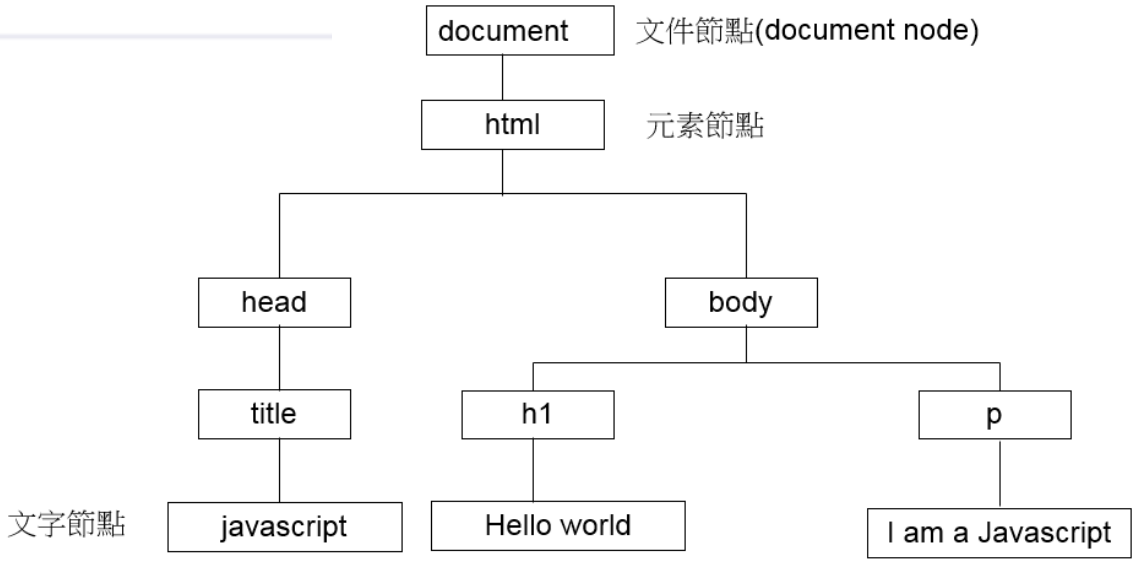



https://www.w3schools.com/jsref/

# DOM (Document Object Model)

DOM.html

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>javascript</title>
5   </head>
6   <body>
7       <h1>Hello World</h1>
8       <p>I am a Javascript</p>
9   </body>
10  </html>
```



# JavaScript 的好處

- 簡單易學
- 具跨瀏覽器之特性
- 提高網頁的互動性及趣味性
- 提供表單前端驗證
- 動態更新網頁部分內容

# JavaScript 的基本架構

- 直接嵌入在HTML文件中的任何位置
- 建議寫法為

```
<head>
   <script >
      <!--  // 程式碼寫在這裡
            window.alert("Hello!");
            // -->
   </script>
</head>
```

- 建立Javascript檔案

```
document.write("JavaScript");
```

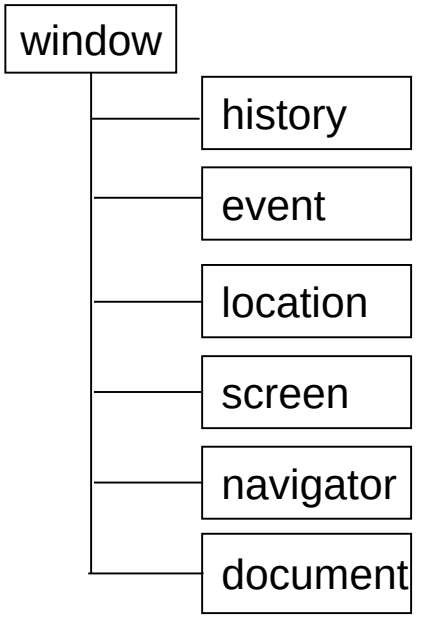
```
<head>
  <script src="first.js"></script>
</head>
```

# JavaScript 的撰寫格式

- 每個敘述不限制一定要寫成一行
- 行尾可加；號代表此敘述結束
- 程式區塊使用{}號包圍
- 程式的註解
  - 單行註解使用 //
  - 多行註解使用 /* */

# JavaScript 的輸出

- 使用window.alert(message)
  - 顯示對話方塊及按鈕
  - window.alert("Hello! \n Javascript");
- 使用document.write(exp1,exp2…)
  - 在文件上印出HTML 元素及字串
  - document.write("Hello! <br> Javascript");
- 使用console.log(exp)
  - 在除錯模式(F12，Ctrl+Shift+I)顯示字串
  - console.log("Hello! \n Javascript");
- 使用innerHTML
- 參考網站:http://www.w3schools.com



# 單元二：資料型態與變數

- JavaScript常數
- JavaScript變數
- JavaScript保留字
- JavaScript跳脫字元
- JavaScript資料型態
- 資料型態的轉換與取得

# JavaScript常數(literal)

- 直接在程式中出現的資料值，永遠不變的值
- 數字常數
  - 123.45,-6
- 字串常數
  - "javascript"
- 布林常數
  - true,false
- 物件常數(單元二)
  - {…..}
- 函數常數(function literal) (單元四)
  - function(){…}
- 陣列常數(單元五)
  - […….]
- RegExp常數(單元五)
  - /……../

# JavaScript變數(identifier)

- 變數的宣告
  - 語法： var identifier [=value];
  - 用 var宣告變數 var intAge = 25;
  - 直接指定變數值 strName = "Sherman";
  - 一般變數宣告後不分**資料型態**(loose typed language), 真正的資料型態將視 assigned value 而定(dynamic language)
- 變數的命名規則
  - 第一個字母應為大小寫英文字母或下底線(_)或$
  - 大小寫視為不同變數 (Java , java)
  - 除了第一個字母外，變數從第2個字元以後可為大小寫英文字, 數字, 下底線_
  - 變數名稱裡不可是保留字

# JavaScript 保留字

| | | | | |
|---|---|---|---|---|
| break | delete | function | return | typeof |
| case | do | if | switch | var |
| catch | else | in | this | void |
| continue | false | instanceof | throw | while |
| debugger | finally | new | true | with |
| default | for | null | try | |

| | | | | | | |
|---|---|---|---|---|---|---|
| class | const | enum | export | extends | import | super |

| | | | | |
|---|---|---|---|---|
| implements | let | private | public | yield |
| interface | package | protected | static | |

# 跳脫字元(escape sequence)

在字串中若要表示特定字元，或該字元具有特殊函義

| 跳脫字元 | 代表字元 |
|---|---|
| \0 | NULL字元(\u0000) |
| \b | 倒退鍵(\u0008) |
| \t | 水平定位字元(\u0009) |
| \n | 換行(\u000A) |
| \v | 垂直定位字元(\u000B) |
| \f | 下一頁(form feed)(\u000C) |
| \r | 游標歸位(\u000D) |
| \” | 雙引號(\u0022) |
| \’ | 單引號(\u0027) |
| \\ | 反斜線(\u005C) |
| \xXX | 兩位數的十六進位 |
| \uXXXX | 以 16 進位數指定 Unicode字元輸出 |

# JavaScript資料型態 Data types

- 基本資料型態
  - 數字(number)型態:雙精準度浮點數(精準度15位)，共64 bits
  - 字串(string)型態：以雙引號或單引號括起來，如:"JavaScript"
  - 布林(Boolean)資料型態：true , false
    - 若結合布林值作+、-、*、/等運算，true會被當作1，而false會被當作0
    - 若在真假判斷式中，任何值都可轉為布林值。**0 、 -0、NaN、""、null、undefined轉為false，其它的值，包括所有物件與陣列則轉為true。**
- 特殊資料型態
  - null(Empty Object)型態：無值或無物件
  - undefined (未定義值)型態：宣告時未指定給值或不存在的物件
- 參考資料型態
  - 物件
  - 陣列

# 物件資料型態 Data types

- 語法:
  - obj = new Object()
  - obj={property:value,property:value…}
- 存取屬性:
  - obj.property
  - obj["property"]
- 存取方法:
  - obj.method()

```
var person=new Object();
person.firstName="Mary";
person.lastName="Wang";

var peson={firstName:"Mary",lastName:"Wang"};
```

```
person.fullName=function(){return this.lastName+" "+this.firstName;}
```

  - this : 指向呼叫該函數的物件

# 資料型態的轉換與取得

- 取得變數的資料型態
  - 使用 typeof 運算子

```
<script >
  var strName ="JavaScript";
  window.alert (typeof strName);
</script>
```

- 資料型態的轉換
  - parseInt()方法　　將變數值轉換成整數
  - parseFloat()方法　　將變數值轉換成浮點數

```
parseInt("33P")
parseInt("12.522")
parseInt("P33")
```

```
parseFloat("5.22A22")
parseFloat("P5.22A22")
```

# 單元三：運算子與敘述

- JavaScript 的運算子
- 增減數運算子
- 流程控制

# JavaScript 的運算子

- 運算元與運算子

  X = 4　　"=" 為運算子 , x 與 4 為運算元

  - 算術運算子(arithmetic operators)
    - + , - , * , / , % , ++ , --
  - 比較性運算子 (comparision operators)
    - = =, != , < , <= , > , >=
  - 指定運算子
    - =
  - 邏輯運算子(logical operators)
    - && , || , !
  - 條件運算子(三元運算子)
    - ?:　　x > 0 ? x : -1

# 增減數運算子

- 增減數運算子置於運算元的前後會有不同的結果
  - 前置(prefix)型

    b = 10 * ++a

    q = 2 * -- a
  - 後置(postfix)型

    b = 10 * a ++

    q = 2 * a --

# 運算子優先順序(operator precedence)

| operator | 描述 |
|---|---|
| .[ ] ( ) | 欄位存取或物件屬性存取、陣列索引、函式呼叫和運算式群組 |
| ++ -- - ~ ! delete new typeof void | 一元運算子、物件建立、傳回資料類型、未定義的值 |
| * / % | 乘法、除法、取餘數 |
| + - + | 加法、減法、字串串連 |
| << >> >>> | 位元移位 |
| < <= > >= instanceof | 小於、小於或等於、大於、大於或等於、是否為某個特定類別的執行個體 |
| == != === !== | 等號比較、不等比較、嚴格等號比較和嚴格不等比較 |
| & | 位元 AND |
| ^ | 位元 XOR |
| \| | 位元 OR |
| && | 邏輯 AND |
| \|\| | 邏輯 OR |
| ?: | 條件式 |
| = *OP*= | 指派，以運算指派 (例如 += 和 &=) |
| , | 多重評估 |

# 流程控制

- 判斷結構
  - If 單一選項敘述
  - switch-case多重選項敘述
- 重複結構
  - for 迴圈敘述
  - while 迴圈敘述
    - Continue 與 break 敘述

# 流程控制 – if 敘述句 (1/4)

- if, if-else. 用 { } 來 group statements

```
if (condition)
   statement1
[else
   statement2]
```

```
if (a == 5)
{
     statements
}
else if (a == 6)
{
     statements
}
else
      statement
```

var *Stringvar* = window.prompt([String *message*],[String *default*])
顯示訊息對話方塊，讓使用者輸入值

# 流程控制 – switch (2/4)

- switch-case statement

語法

```
switch (expression) {
   case label :
      statementlist;break;
   case label :
      statementlist;break;
   ...
   default :
      statementlist
}
```

Example

```
switch (a)
{
      case 5 : I = 100; J = 100;
               break;
      case 6 : I=200; J=400;
               break;
      default : I=0; J=0;
                break;

}
```

# 流程控制 – for (3/4)

for loop

for (initialization ; test ; increment)

{statements }

break and continue

```
<script>
for (i=5;i>0;i--)
{
   if (i==3) break;
document.write("i="+i+"<br>");
}
</script>
```

# 流程控制 – while (4/4)

while loop

while (expression) statements}

```
<script>
var i=10;
while(i>0)
{
document.write("i="+i+"<br>");
i--;
}
</script>
```

# 單元四：函數

- 函數的功能
- 自訂函數
  - 宣告式函數(Declarative function)
  - 函數實字(function literal or function expression)
- 區域變數 VS 廣域變數
- this物件

# 函數的功能

- 獨立完整的程式碼，可完成一特定的任務
- 使用函數的好處
  - 避免重覆設定工作
  - 使程式更為模組化從而易閱讀或修正
- 函數寫法分為
  - 宣告式函數(Declarative function)
  - 函數實字(function literal or function expression or anonymous function)

# 宣告式函數

- 語法：

```
function 函數名稱(參數1,參數2, ….. parameters)
 {
     //將程式碼寫在這裡
   return 回傳值
 }
```

```
function f (x){
            return x*x;
}
```

- function –保留字
- 函數名稱
  - 可自訂
  - 不可使用保留字
- 參數列 – 可有可無 (參數1, 參數2, ……,parameters)
- return 如果沒有加上回傳值，回傳值為undefined

# 函數實字(function literal)

- 語法：

  var variable =function(params){

  statemenets

  }

```
Var f=function(x){
        return x*x;
}
```

- function –保留字
- 沒有函數名稱

- 語法:
  - obj = new Object()
  - obj={property:value,property:value…}
- 存取屬性:
  - obj.property
  - obj["property"]
- 存取方法:
  - obj.method()

```
var person=new Object();
person.firstName="Mary";
person.lastName="Wang";

var peson={firstName:"Mary",lastName:"Wang"};
```

```
person.fullName=function(){return this.lastName+" "+this.firstName;}
```

  - this : 指向呼叫該函數的物件

# JavaScript資料型態Data types



- 基本資料型態
  - 數字(number)型態: 整數、浮點數(–4.81 , 123.45)
  - 字串(string)型態：以雙引號或單引號括起來，如:"JavaScript"
  - 布林(Boolean)資料型態 : true , false
    - 若結合布林值作+、-、*、/等運算，true會被當作1，而false會被當作0
    - 若在真假判斷式中，任何值都可轉為布林值。**0 、 -0、NaN、""、null、undefined**轉為**false，其它的值，包括所有物件與陣列則轉為true。**
- 特殊資料型態
  - null(Empty Object)型態：無值或無物件
  - undefined (未定義值)型態：宣告時未指定給值或不存在的物件
- 參考資料型態
  - 物件
  - 陣列

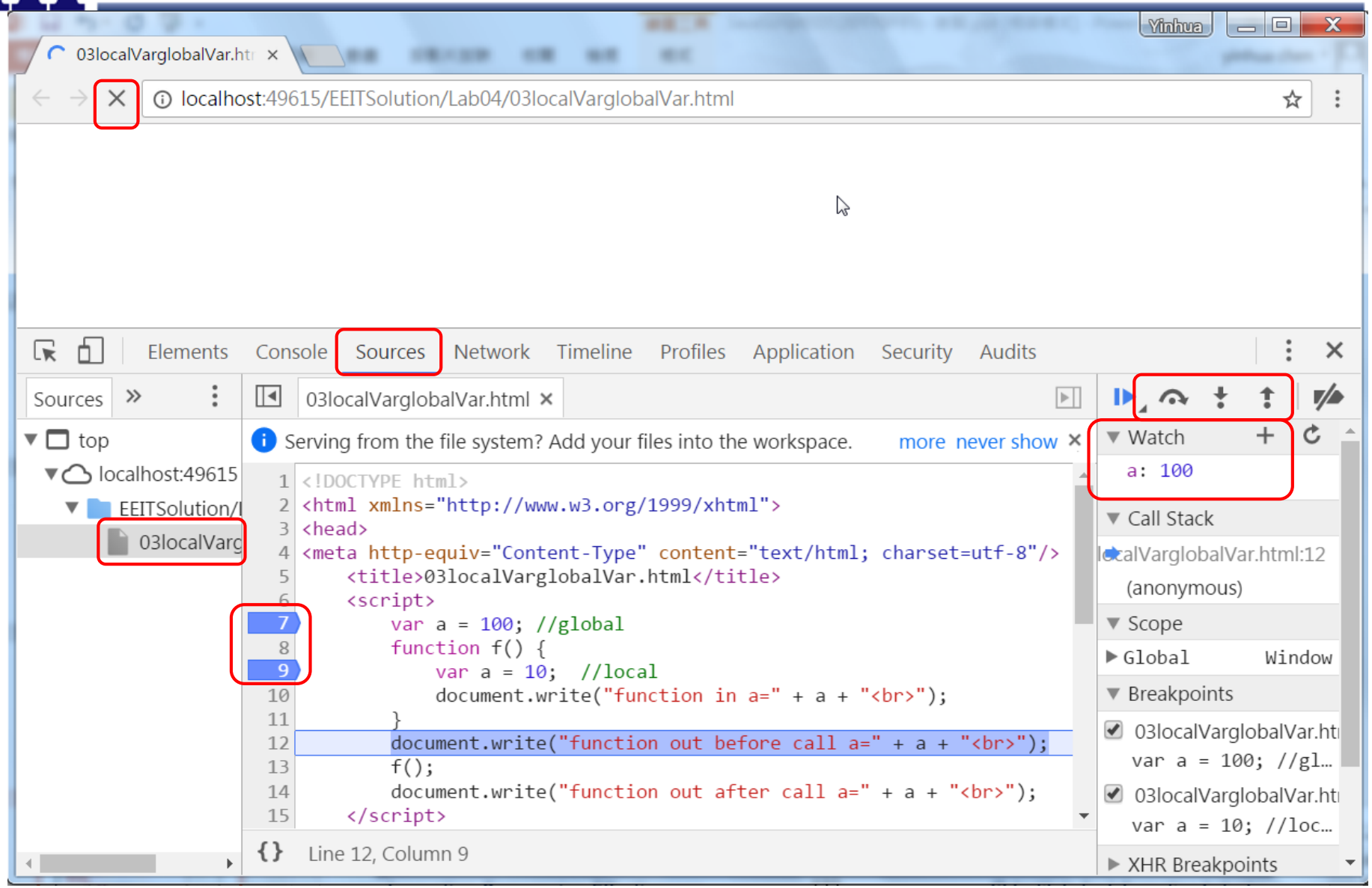
# 區域變數 VS 廣域變數

- 區域(local)變數
  - 用 var 在函數內宣告
- 廣域(global)變數
  - 在函數外宣告
  - 在函數內宣告，但不使用 var

# this物件

- 指向呼叫該函數的物件
  - 物件.函數(); //函數內的this指向該物件
- 指向全域物件
  - 函數(); //函數內的this指向全域物件

# 除錯(debug)



# 單元五：JavaScript物件

- JavaScript物件種類
- JavaScript 物件的屬性與方法
- Global物件
- String 物件
- Date物件
- Array物件
- RegExp 物件

# JavaScript物件種類

- 原生(Native)物件或內建物件
  - 是由ECMAScript規格所定義的物件或物件類別。例如，Date，Array，function，RegExp
- 主機(host)物件
  - 由JavaScript執行環境所定義的物件(例如Web瀏覽器)。客戶端用來表示網頁結構的HTMLElement物件 (https://developer.mozilla.org/en-US/docs/Web/API )
- 使用者定義(User-defined)物件
  - 使用JavaScript程式碼所建立的物件

# JavaScript 物件的屬性與方法

- JavaScript 是一個以物件為基礎的程式語言
- 語法如下:
  - 物件.屬性
  - 物件.方法()
- 原生物件
  - Global物件
  - String 物件
  - Date 物件
  - Array 物件
  - RegExp 物件

# Global 物件方法

- parseInt(*numString*) ：回傳整數值
- parseFloat(*numString*) ：回傳浮點數值
- eval(*string*) ： 計算字串所含的運算式
- isNaN(*numValue*) ： 測試字串是否為數值, 若是回傳 false , 否則回傳 true
- encodeURI(*string*),encodeURIComponent(string)
  ：將文字字串編碼, 以便資料正確傳送至網站伺服器
  - URI(Uniform Resource Identifier)**統一資源識別碼**，是一個用於標識網際網路資源名稱的字串
- decodeURI(*string*) , decodeURIComponent(string)
  ：解碼為文字字串

# 存取文件元素及值

- var *elementObj*=document.getElementById(elementID)
  - 取得文件上指定ID屬性的元素物件
- var *Stringvar* = *elementObject*.value
  - 取得文件上元素物件的值

```
<script>
      function check(){
          var theName=document.getElementById("idName").value;
       }
</script>
<body>
   name:<input type="text" id="idName" />
   <input type="button" value="check" onclick="check(); " />
</body>
```

02checkAge.html

```
    <script>
        function checkAge() {
            var theAgeObj = document.getElementById("idAge");
            //alert(typeof theAgeObj);
            var theAgeVal = theAgeObj.value;
            //alert(typeof theAgeVal);
            //alert(theAgeVal);
            var theAge = document.getElementById("idAge").value;

            if (isNaN(theAge))
                alert("please input number 0-9!!");
            else
                alert("Age=" + theAge);
        }
    </script>
</head>
<body>
    Age:<input type="text" id="idAge" size="3" maxlength="3" />
    <br />
    <input type="button" value="checkAge" onclick="checkAge();" />
</body>
```

# String 物件(String Object)1/3

- 字串變數與物件所運用的屬性與方法皆是一樣
- 語法:
  - stringObj = new String**(**["stringLiteral"]**)**
- 字串的方法
  - charAt(*index*) ：回傳指定索引的字元
  - charCodeAt(index) ：回傳指定索引的Unicode 編碼
  - indexOf (*subString*) ：回傳字元第一次出現的位置
  - substr (*start[, length]*) ： 從開始位置取得幾個字元
  - substring(*start, end*) ：從開始位置取到結束位置
  - lastIndexOf (*subString*) ：回傳字元最後出現的位置
  - split([*separator] [,limit]*) ：將字串分為數個子字串，回傳所產生的字串陣列

# String 物件(String Object)2/3

- ASC-II(American Standard Code for Information Interchange，美國資訊交換標準碼)是基於拉丁字母的一套電腦編碼系統。它主要用於英語系編碼。

- unicode（萬國碼、國際碼、統一碼）是電腦系統的一項業界標準。對世界上大部分的文字系統進行整理、編碼，使得電腦可以用更為簡單的方式來呈現和處理文字。

# String 物件(String Object)3/3

- UTF-8(8-bit Unicode Transformation Format)是一種針對unicode的可變長度字元編碼。
  - US-ASCII字元只需一個位元組編碼(U+0000至U+007F) 。
  - 帶有附加符號的拉丁文、希臘文、西里爾字母、亞美尼亞語、希伯來文、阿拉伯文、敘利亞文及它拿字母則需要兩個位元組編碼(U+0080至U+07FF) 。
  - 其他基本多文種平面（BMP）中的字元（這包含了大部分常用字，如大部分的漢字）使用三個位元組編碼(U+0800至U+FFFF)
  - 其他極少使用的Unicode輔助平面的字元使用四至六位元組編碼

- 字串的屬性
  - length : 字串的長度

# Date物件(Date Object)

- 建立日期物件語法：

  dateObj = new Date()

  dateObj = new Date(*milliseconds*)

  dateObj = new Date(*dateString*)

  dateObj = new Date (*year,month,date [,hour [, minutes[, seconds[, milliseconds]]]]*)

- 使用日期物件語法：

  dateObj.方法

# Date物件方法

- 日期物件可使用的方法如下:
  - getFullYear() ：回傳西元年份值
  - getMonth() ：回傳月份值(0-11)
  - getDate() ：回傳日數(1-31)
  - getDay() ：回傳星期數(0-6)
  - getHours() ：回傳時數(0-23)
  - getMinutes() ：回傳分數(0-59)
  - getSeconds() ：回傳秒數(0-59)
  - getTime() ：回傳自1970/1/1 0:0:0算起之毫秒數

# Array物件(Array Object)

- 陣列是放置在連續記憶體的變數的集合
- 語法:
  - arrayObj = new Array([size])
  - arrayObj = new Array([element0[, element1[, ...[, elementN]]]])
  - arrayObj = [element0 [, element1[, ...[, elementN]]]])
- 讀取陣列
  - var v = arrayObj[0];

# 陣列範例

- 產生陣列

```
//方法二
var myArray = new Array("baseball", "baseketball", "valleyball")
//方法三
var myArray = ["baseball", "baseketball", "valleyball"]
```

- 讀取陣列資料

```
for (var i=0;i<myArray.length;i++) {
      alert (myArray[i])
}
```

# 陣列的使用

- 取得陣列長度
  - length
- 在陣列後面新增一個元素
  - push(item1,...) : arrayObj.push("item3")
- 刪除陣列最後一個元素
  - pop() : arrayObj.pop()
- 常用方法
  - join(separator)：回傳所有元素串連
  - reverse() ：回傳元素反轉陣列
  - sort() ：回傳排序過陣列

# RegExp 物件

- Regular Expression 語法:
  - re = new RegExp("pattern", "flag")
  - re = /pattern/flag
    - pattern 是正規表示法的字串，flag 則是比對的方式
    - flag 的值可能有三種
      - g：全域比對（Global match）
      - i：忽略大小寫（Ignore case）
      - gi：全域比對並忽略大小寫
- 方法:
  - *re*.test(*string*)：以字串 string 比對物件 re，並回傳比對結果（true 代表比對成功，false 代表比對失敗）

# 正規式字元

| 字元 | 說明 | 簡單範例 |
|---|---|---|
| \ | 避開特殊字元 | /A*/ 可用於比對 "A*"，其中 * 是一個特殊字元，為避開其特殊意義，所以必須加上 "\" |
| ^ | 比對輸入列的啟始位置 | /^A/ 可比對 "Abcd" 中的 "A"，但不可比對 "aAb" |
| $ | 比對輸入列的結束位置 | /A$/ 可比對 "bcdA" 中的 "A"，但不可比對 "aAb" |
| * | 比對前一個字元零次或更多次 | /bo*/ 可比對 "Good boook" 中的 "booo"，亦可比對 "Good bk" 中的 "b" |
| + | 比對前一個字元一次或更多次，等效於 {1,} | /a+/ 可比對 "caaandy" 中的 "aaa"，但不可比對 "cndy" |
| ? | 比對前一個字元零次或一次 | /e?l/ 可比對 "angel" 中的 "el"，也可以比對 "angle" 中的 "l" |
| . | 比對任何一個字元（但換行符號不算） | /.n/ 可比對 "nay, an apple is on the tree" 中的 "an" 和 "on"，但不可比對 "nay" |
| xy | 比對 x 或 y | /a*b*/g 可比對 "aaa and bb" 中的 "aaa" 和 "bb" |
| {n} | 比對前一個字元 n 次，n 為一個正整數 | /a{3}/ 可比對 "lllaaalaa" 其中的 "aaa"，但不可比對 "aa" |
| {n,} | 比對前一個字元至少 n 次，n 為一個正整數 | /a{3,}/ 可比對 "aa aaa aaaa" 其中的 "aaa" 及 "aaaa"，但不可比對 "aa" |
| {n,m} | 比對前一個字元至少 n 次，至多 m 次，m、n 均為正整數 | /a{3,4}/ 可比對 "aa aaa aaaa aaaaa" 其中的 "aaa" 及 "aaaa"，但不可比對 "aa" 及 "aaaaa" |
| [xyz] | 比對中括弧內的任一個字元 | /[ecm]/ 可比對 "welcome" 中的 "e" 或 "c" 或 "m" |
| [^xyz] | 比對不在中括弧內出現的任一個字元 | /[^ecm]/ 可比對 "welcome" 中的 "w"、"l"、"o"，可見出其與 [xyz] 功能相反。（同時請同學也注意 /^/ 與 [^] 之間功能的不同。） |
| \b | 比對英文字的邊界 | 例如 /\bn\w/ 可以比對 "noonday" 中的 'no' ;<br>/\wy\b/ 可比對 "possibly yesterday." 中的 'ly' |

# 正規式字元

| | | |
|---|---|---|
| \d | 比對任一個數字，等效於 [0-9] | /[\d]/ 可比對 由 "0" 至 "9" 的任一數字 但其餘如字母等就不可比對 |
| \D | 比對任一個非數字，等效於 [^0-9] | /[\D]/ 可比對 "w" "a"... 但不可比對如 "7" "1" 等數字 |
| \f | 比對換頁(\u000C) | 若是在文字中有發生 "換頁" 的行為 則可以比對成功 |
| \n | 比對換行符號(\u000A) | 若是在文字中有發生 "換行" 的行為 則可以比對成功 |
| \r | 比對 游標返回(\u000D) | |
| \s | 比對任一個空白字元（White space character），等效於 [ \f\n\r\t] | /\s\w*/ 可比對 "A b" 中的 "b" |
| \S | 比對任一個非空白字元，等效於 [^ \f\n\r\t] | /\S/\w* 可比對 "A b" 中的 "A" |
| \t | 比對定位字元（Tab）(\u0009) | |
| \w | 比對數字字母字元（Alphanumerical characters）或底線字母（"_"），等效於 [A-Za-z0-9_] | 比對數字字母字元（Alphanumerical characters）或底線字母（"_"），等效於 [A-Za-z0-9_] |
| \W | 比對非「數字字母字元或底線字母」，等效於 [^A-Za-z0-9_] | /\W/ 可比對 ".A _!9" 中的 "."、" "、"!"，可見其功能與 /\w/ 恰好相反。 |
| \xxx | 比對八進位，其中*xxx是八進位數目* | /\123/ 可比對 與 八進位的ASCII中 "123" 所相對應的字元值。 |
| \x*hh* | 比對兩位數的十六進位，其中*hh是十六進位數目* | /\x57/ 可比對 與 16進位的ASCII中 "57" (W)所相對應的字元。 |
| \uhhhh | 比對四位數的十六進位 | /\u4E2D/ 可比對 與 16進位的unicode "4E2D" (中)所相對應的字元。 |
| \0 | 比對NULL字元(\u0000) | |
| (?=p) | 正向前置(positive lookahead)，比對後面字元中相符於樣式p | /windows(?=7\|8)/， /^(?=.*[0-9]).{2}$/ 可比對字串中至少有一個數字 |
| (?!p) | 負向前置，比對後面字元中不相符於樣式p | negative lookahead |

# 正規式字元(範例)

```
function checkEmail()
{
    var theEmail=document.getElementById("idEmail").value;
    re = /^.+@.+\..{2,3}$/;
    if (re.test(theEmail))
        alert("成功！符合「" + re + "」的格式！");
    else
        alert("失敗！不符合「" + re + "」的格式！");
}
```

# 單元六：JavaScript的事件處理

- 事件處理方式
- 事件處理範例
- 常見事件

# 事件處理方式

- 事件(Event)是使用者對瀏覽器或網頁內容所做的某個動作
- 事件處理程序(Event Handlers)定義事件發生時要做的事
- 事件處理方式
  - HTML屬性的事件處理程序
  - JavaScript屬性的事件處理程序
  - W3C DOM 處理程序

# 事件處理範例

- HTML屬性的事件處理程序

```
<input type="button" onclick="check() " value="Try it"/>
```

- JavaScript屬性的事件處理程序
  - *object.event=function;*

```
<script>
        document.getElementById("myBtn").onclick=check;
        function check(){   }
</script>
```

- W3C DOM處理程序
  - *object*.addEventListener(*event*, *function,usecapture*)

```
<script>
    document.getElementById("myBtn"). addEventListener("click", check,false);
</script>
```

# 函數(function)

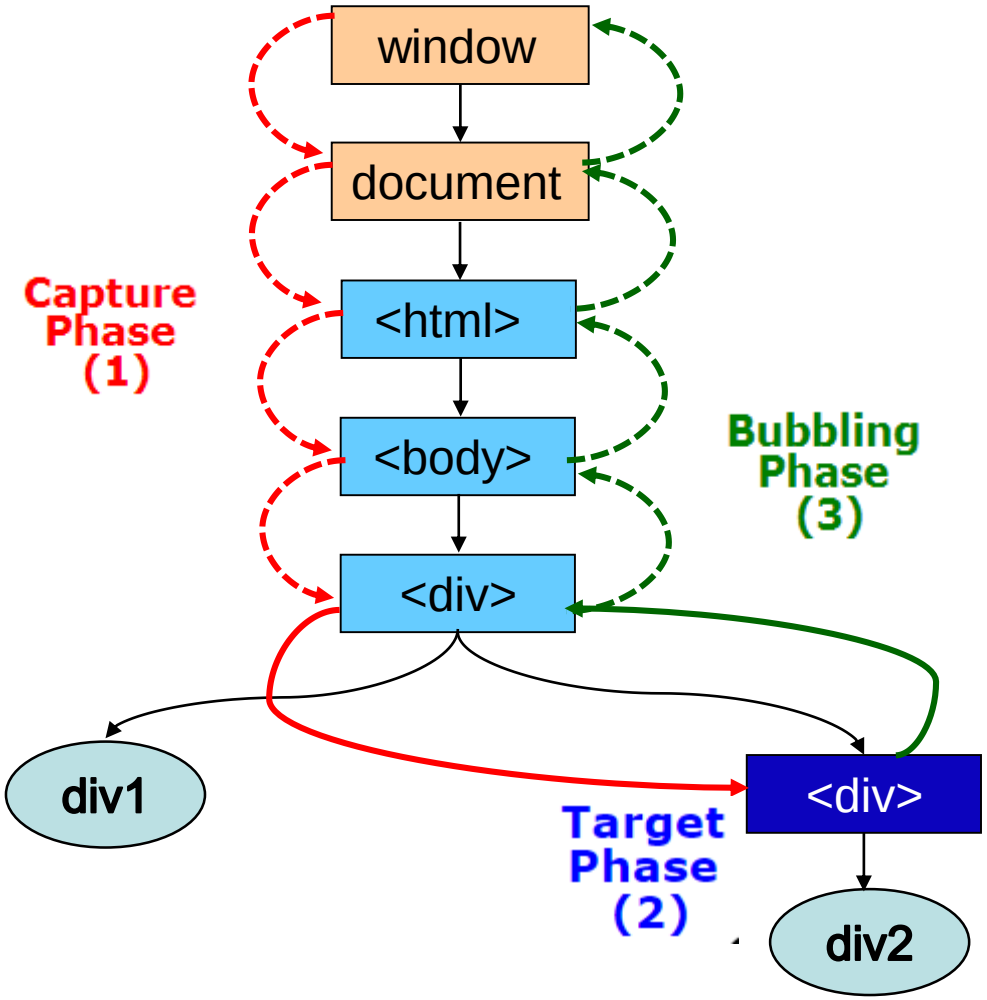
- 宣告式函數

```
function fAdd1(a,b){
        return (a+b);
}
```

- 函數實字(function literal)

```
var fAdd2=function(a,b){
        return (a+b);
}
```

# DOM2級事件(W3C)



# 常見的事件

- 常見的事件：
  - 與鍵盤相關的
    - onkeypress、onkeyup、onkeydown
  - 與滑鼠相關的
    - onclick、ondblclick、 onmouseover 、 onmouseout…
  - 與網頁載入
    - onload、onunload

```
HTML: <body onload= " init() " >
```

```
Javascript:
document.addEventListener("DOMContentLoaded",init);
```

  - 其它

```
Javascript:
document.addEventListener("DOMContentLoaded",function(){};);
```

    - onblur(焦點離開)、onfocus(取得焦點)、onchange(改變內容且離開)、onsubmit、onreset

# 單元七： Dynamic HTML

- Dynamic Styles 動態樣式
- Dynamic Content 動態內容

# 動態樣式

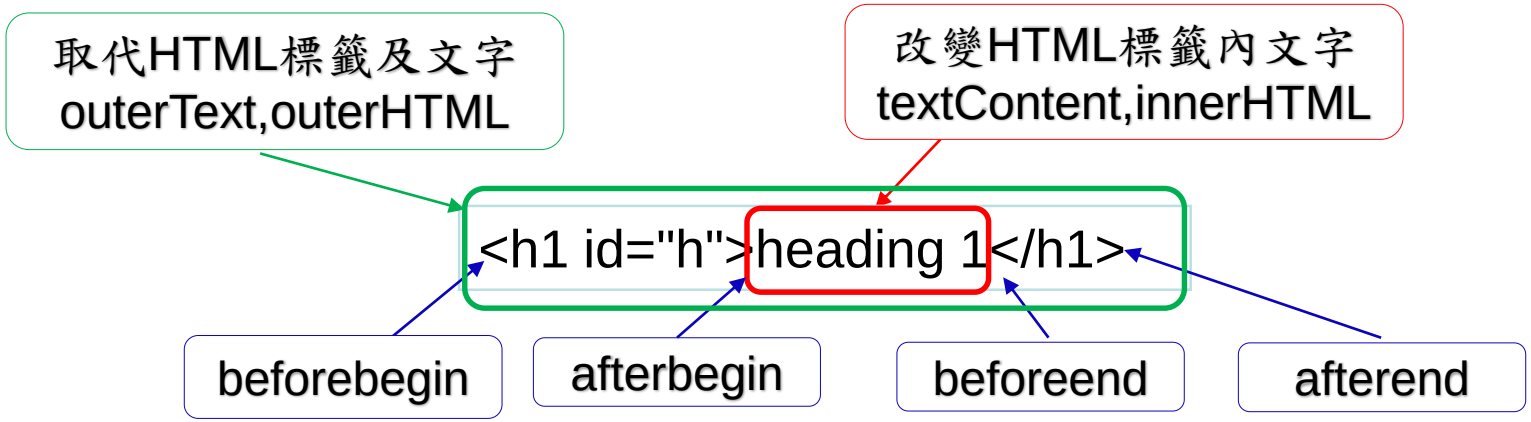
- 方法一
  - 透過 物件.style.css屬性 = css值 來改變樣式
- 方法二
  - 先預先定義class的樣式，再使用
    物件. className屬性=class樣式 來改變樣式

```
var theP = document.getElementById("myP");
theP.style.fontSize="20";
theP.style.textDecoration="underline";

theP.className = "s2";
```

# 動態修改顯示內容

- 改變HTML標籤內的文字
  - textContent ：將取代的內容視為純文字(text)
  - innerHTML：將取代的內容視為HTML標籤解譯

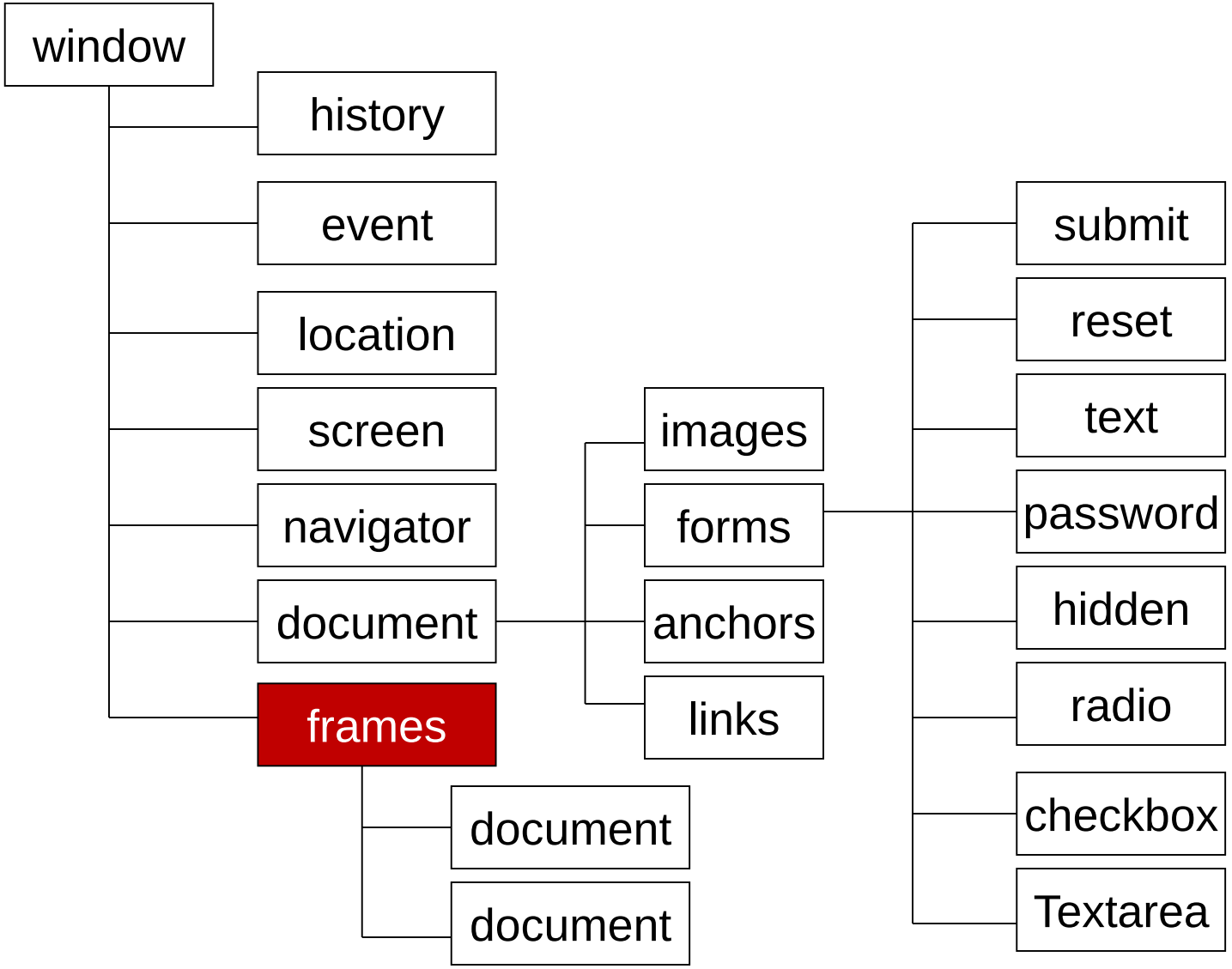



- 自訂文字加入的位置
  - insertAdjacentHTML(where, text)
  - insertAdjacentText (where, text)
    - Where : afterbegin , beforebegin(開始標籤之前) , beforeend , afterend (結束標籤之後)

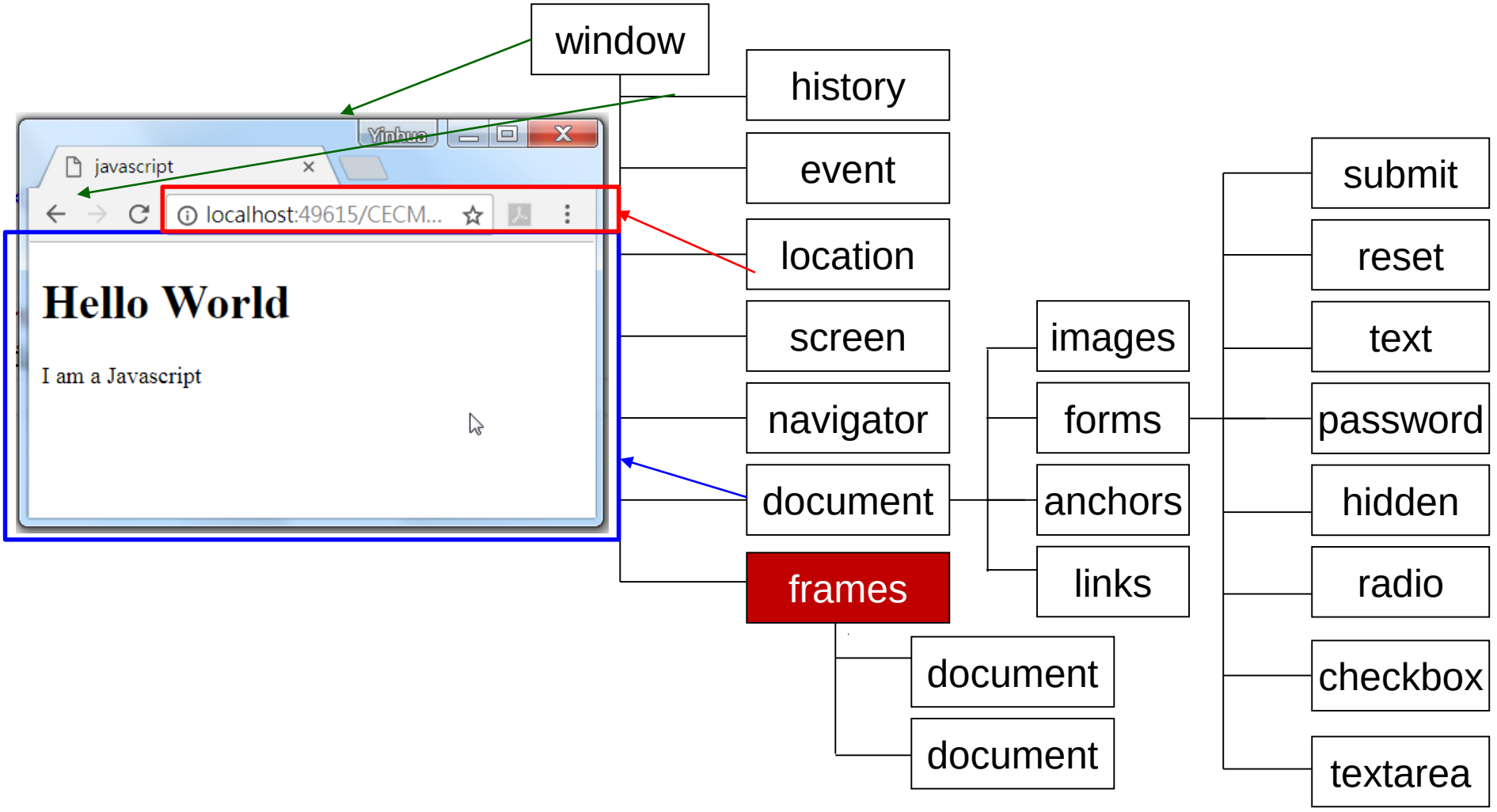
# 單元八：BOM 物件模型

- BOM 物件模型架構
- 視窗(window)物件
- 位址 (location)物件
- 歷史(history)物件
- 事件(event)物件
- 文件(document)物件
- 圖片(image)物件
- 表單 (form) 物件

# BOM 物件模型架構(Browser Object Model)



# BOM 物件模型架構(Browser Object Model) p8




https://www.w3schools.com/jsref/

# 視窗(window)物件

- window物件是瀏覽器物件模型的最高階層
- 代表目前正在使用的視窗
- window物件語法：

  window.方法()

  window.屬性
- 常用功能
  - 視窗的開啟與關閉
  - window 物件內建的對話方塊
  - window 物件的計時器

# 視窗的開啟與關閉

- 方法
  - open() 開啟新視窗
    - window.open ( [URL],[名稱],[視窗規格] )
      - 名稱 : _self, _parent, _top, _blank, name
      - 視窗規格 : width, height, top, left
  - close() 關閉視窗

```
window.open("a.html", "_blank", "top=50,left=100,width=500,height=100");
```

# window 物件內建的對話方塊

- 方法
  - alert(message)：顯示訊息對話方塊及按鈕
  - confirm(message) ：顯示訊息對話方塊，讓使用者選取確定或取消按鈕
    - var *Booleanvar* =window.confirm([String *message*]);
  - prompt(msg, defaultvalue) ：顯示訊息對話方塊，讓使用者輸入值

# Window 物件的計時器

- 要建立動態的網頁內容一定要用計時器
- 方法
  - var timeoutID=setTimeout(*function*, *milliseconds*) 時間到後只執行一次
  - clearTimeout(timeoutID)停止 setTimeout 方法啟動的計時器
  - var IntervalID=setInterval(*function*, *milliseconds*) 每隔多少時間執行一次
  - clearInterval (IntervalID)停止setInterval方法啟動的計時器
- 範例
  - 電子鐘

# 位址 (location)物件

- 儲存載入網頁URL的相關資訊
- 屬性
  - href：轉向連結到其它網址
  - protocol , host , hostname , port , pathname, hash
- 方法
  - reload() ：重新載入目前網頁
  - replace(url)：使用新網頁取代目前網頁

# 歷史(history)物件

- 主要功能是用來儲存用戶端最近查閱過的網址清單
- history物件屬性
  - length
- history物件方法
  - forward()
  - back()
  - go()　　ex go(-2) 回到上二頁

# 事件(event)物件(1)

- 主要功能是用來取得事件發生的類型與位置
- 屬性
  - type ：事件發生的類型
  - button：回傳滑鼠按下的按鈕
  - keyCode(IE8) ：回傳Unicode 編碼
  - which：回傳Unicode 編碼
  - altKey , ctrlKey , shiftKey ：回傳true 或 false
  - screenX , screenY：滑鼠游標相對於**客戶端螢幕**的位置座標
  - clientX,clientY ：滑鼠游標相對於**目前視窗**的位置座標
  - offsetX,offsetY ：滑鼠游標相對於**觸發事件的元素**的位置座標
  - srcElement(IE8) ：回傳觸發事件的元素
  - target：回傳觸發事件的元素
  - currentTarget：回傳事件正在處理時所在的元素

以event物件作為參數來使用，IE8以前是透過全域變數window.event來取用這個物件

# 事件處理範例



- HTML屬性的事件處理程序

```
<h1 id="m1"  onclick="f1()" >method1</h1>
```

- JavaScript屬性的事件處理程序

```
<h1 id="m2">method2</h1>
<script>
        document.getElementById("m2").onclick=f2;
        function f2(){alert("method3");}
</script>
```

- W3C DOM處理程序

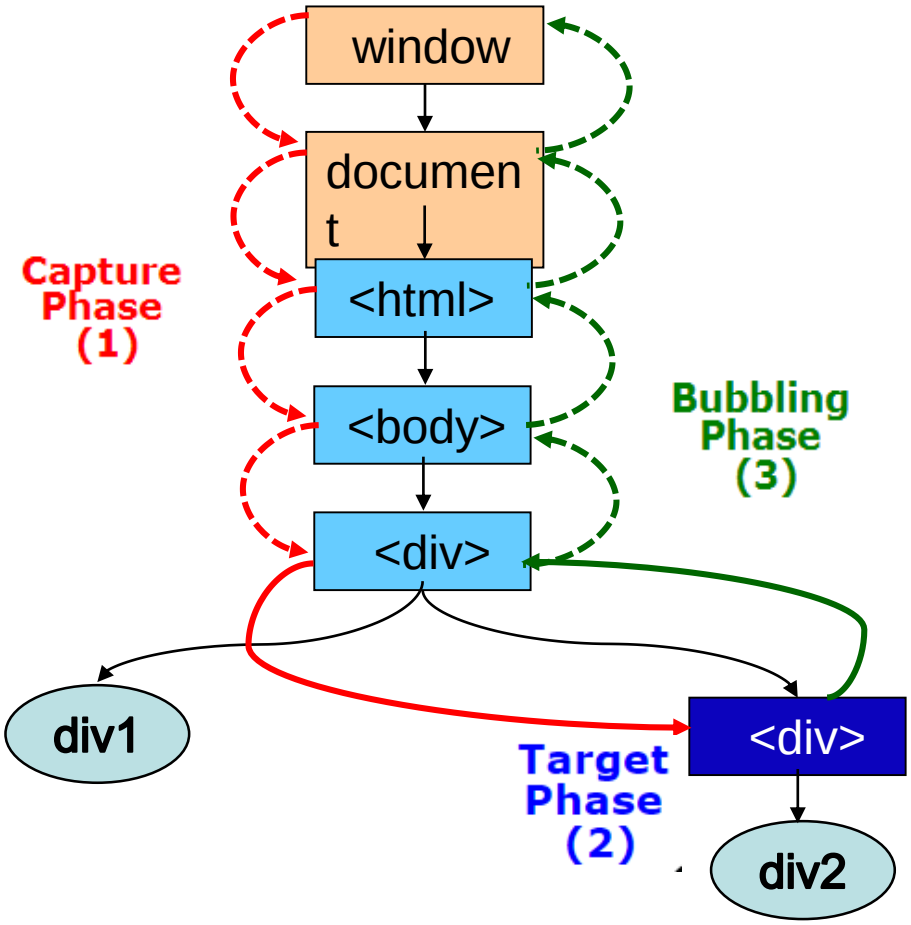
```
<h1 id="m3">method3</h1>
<script>
      document.getElementById("m3"). addEventListener("click", f3,false);
      function f3(){alert("method3");}
</script>
```

# JavaScript資料型態Data types

- 基本資料型態
  - 數字(number)型態: 整數、浮點數(–4.81 , 123.45)
  - 字串(string)型態：以雙引號或單引號括起來，如:"JavaScript"
  - 布林(Boolean)資料型態 : true , false
    - 若結合布林值作+、-、*、/等運算，true會被當作1，而false會被當作0
    - 若在真假判斷式中，任何值都可轉為布林值。**0 、 -0、NaN、""、null、undefined**轉為**false，其它的值，包括所有物件與陣列則轉為true。**
- 特殊資料型態
  - null(Empty Object)型態：無值或無物件
  - undefined (未定義值)型態：宣告時未指定給值或不存在的物件
- 參考資料型態
  - 物件
  - 陣列

# DOM2級事件(W3C)





# 事件(event)物件(2)

- 取消事件氣泡(event bubbling)
  - cancelBubble : IE8以前only
  - stopPropagation() : firefox only

- 停止(取消)預設的動作
  - returnValue: IE8以前only
  - preventDefault() : firefox only

# 文件(document)物件

- 主要功能是代表目前的文件
- 屬性
  - 控制文件外觀： bgColor, fgColor
  - 得到文件資訊：title, lastModified
  - 取得文件下的子元素: forms,images
- 方法
  - open() , close() , write() , writeln()

# 圖片(image)物件

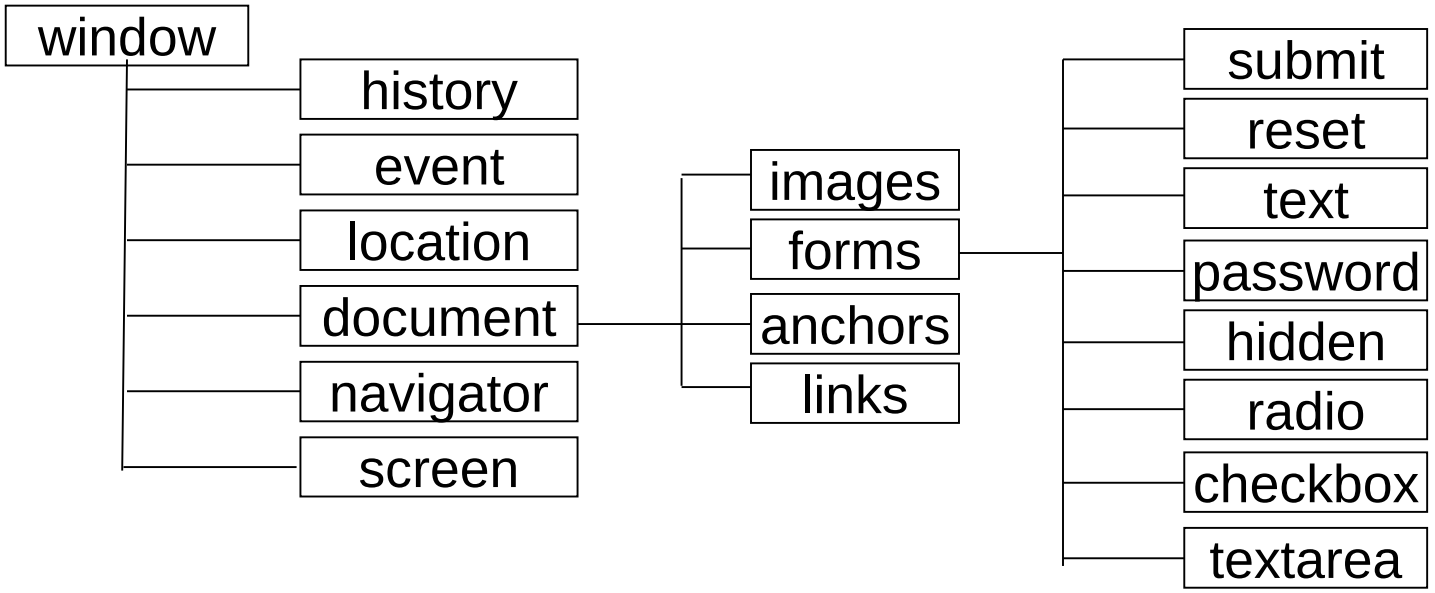
- image物件代表網頁中的圖片，在網頁中顯示圖片是使用<img>標籤
- 屬性:
  - src 如,document.images[0].src= "other.gif ";
  - useMap：設定或取得客戶端影像地圖

- 方法:
  - document.getElementsByTagName(" img ")：取得所有image元素
  - document. querySelectorAll(selectors)：取得所有selector元素

- https://www.w3.org/TR/2014/REC-html5-20141028/embedded-content-0.html#the-img-element
- https://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/idl-definitions.html

# 表單 (form) 物件(1/3)

- 使用欄位名稱取得欄位資料
  - document.forms[0].欄位名稱.value




- 使用forms和elements物件存取欄位資料
  - document.forms[0].elements[0].value

# 表單 (form) 物件(2/3)

- 屬性：
  - length ： 傳回表單擁有的欄位數
- 方法：
  - reset()
  - submit()

- 範例
  - JavaScript 的表單欄位驗證

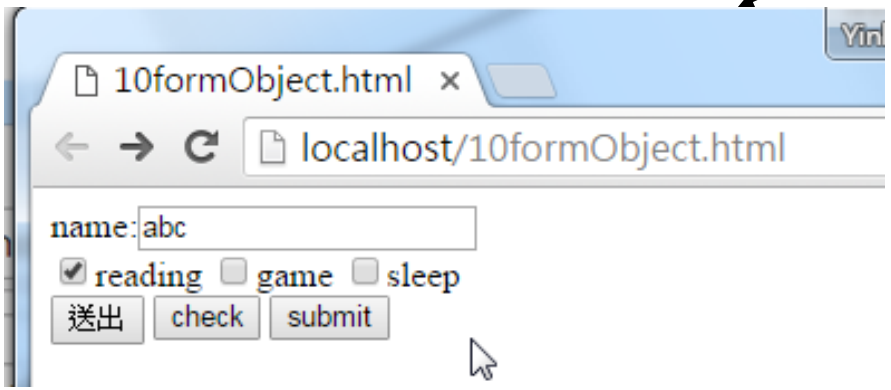
# 表單 (form) 物件(3/3)

http://localhost/10formObject.html

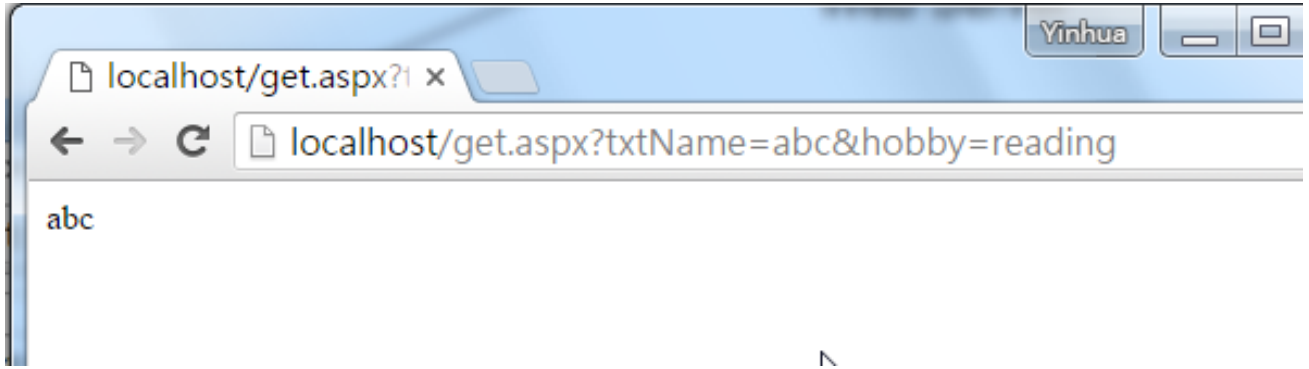
HTTP 傳回 HTML

Client(Brower)

Web Server



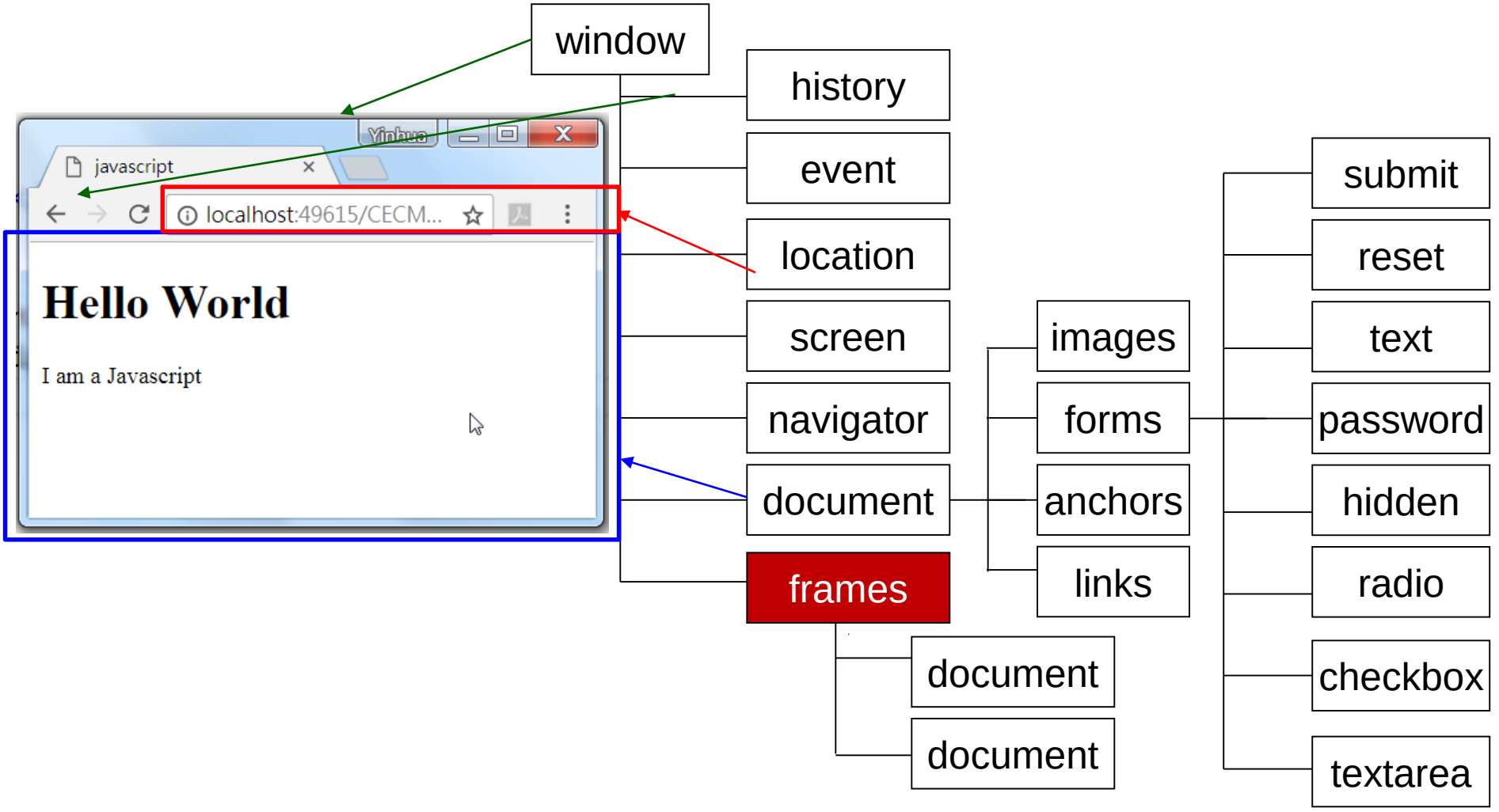

```
<form action="get.aspx" method="get">
name:
<input type="text" name="txtName“ value="abc" />
```



# 單元九：DOM (Document Object Model)

- 什麼是DOM
- HTML網頁
- HTML網頁解析出的樹狀結構
- 瀏覽節點
- 尋找及存取節點資料
- 選取文件元素
- 如何新增及移除元素
- 關於屬性資料

# BOM 物件模型架構(Browser Object Model) p8




https://www.w3schools.com/jsref/

# 什麼是DOM

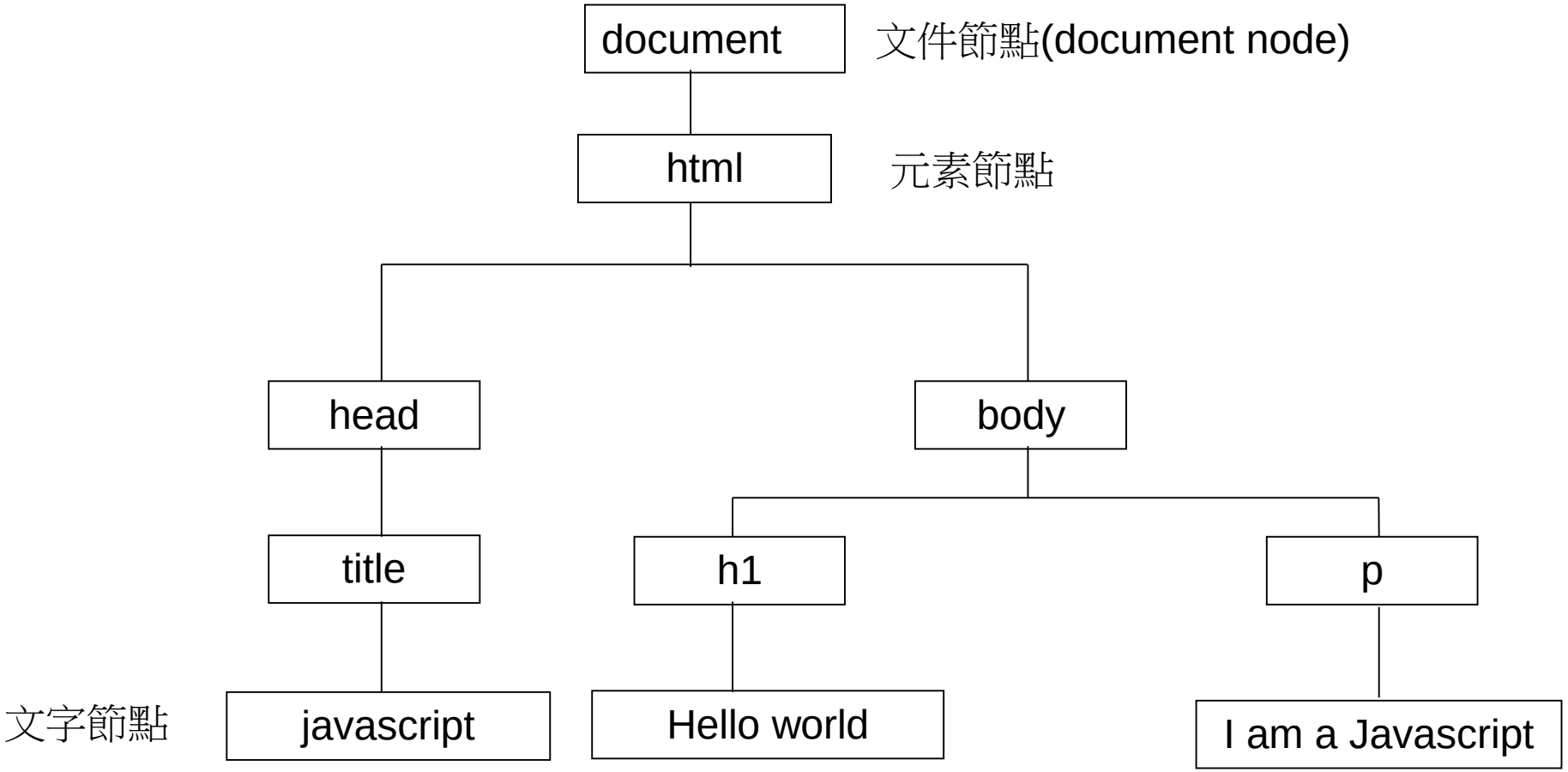
- 文件物件模型(Document Object Model)
  - 是一組用來處理HTML及XML文件的屬性與方法
  - 文件中的所有資料，皆可透過DOM來存取、修改、刪除及新增

- DOM Level4於2015年成為W3C的建議規格
  - https://www.w3.org/TR/2015/REC-dom-20151119/

# HTML網頁

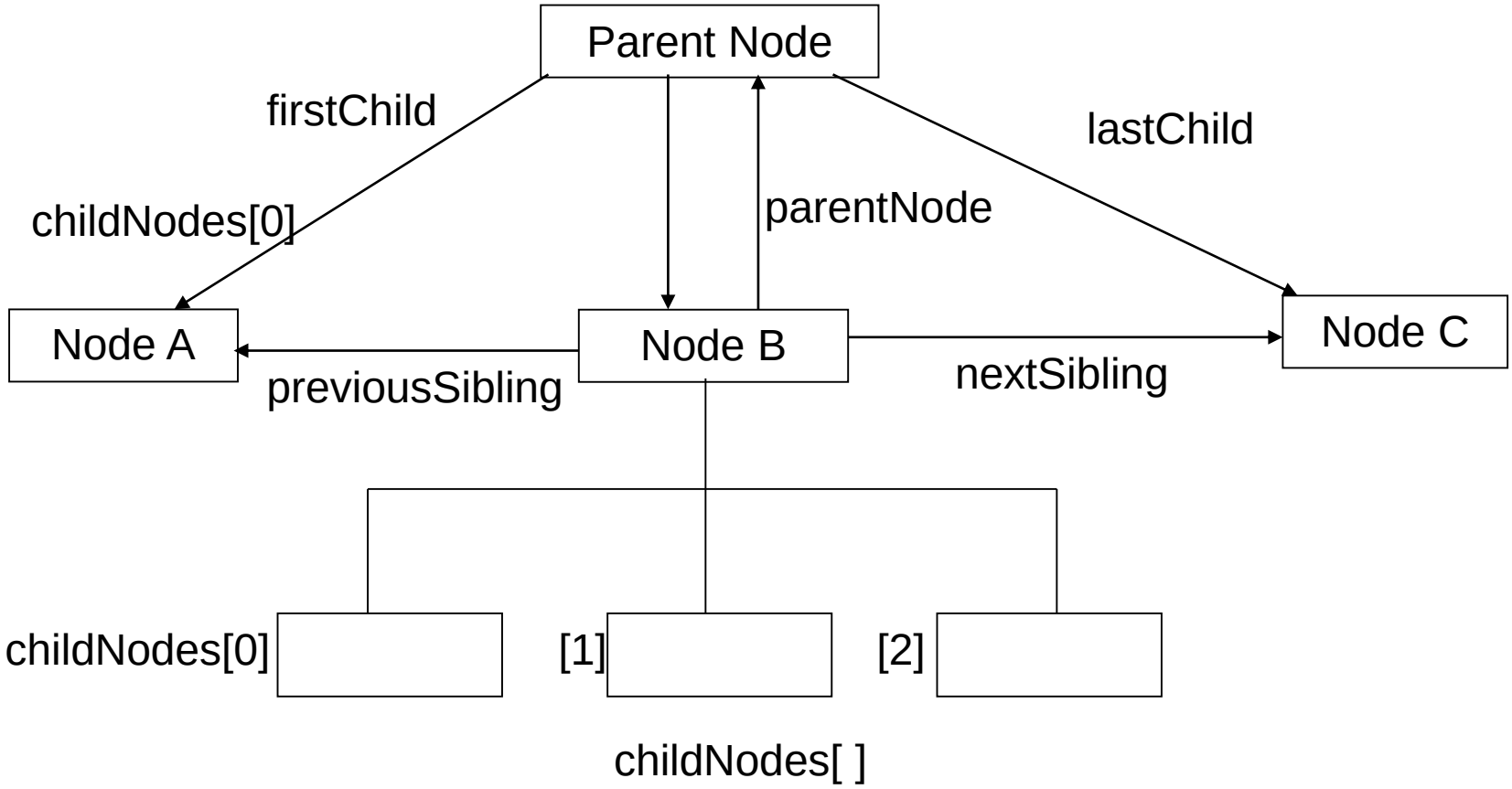
```
<!DOCTYPE>
<html>
    <head>
        <title>javascript</title>
    </head>
    <body>
        <h1>Hello world</h1>
        <p>I am a Javascript</p>
    </body>
</html>
```

# HTML網頁解析出的樹狀結構




參考文件：http://www.w3schools.com/xml/dom_nodetype.asp

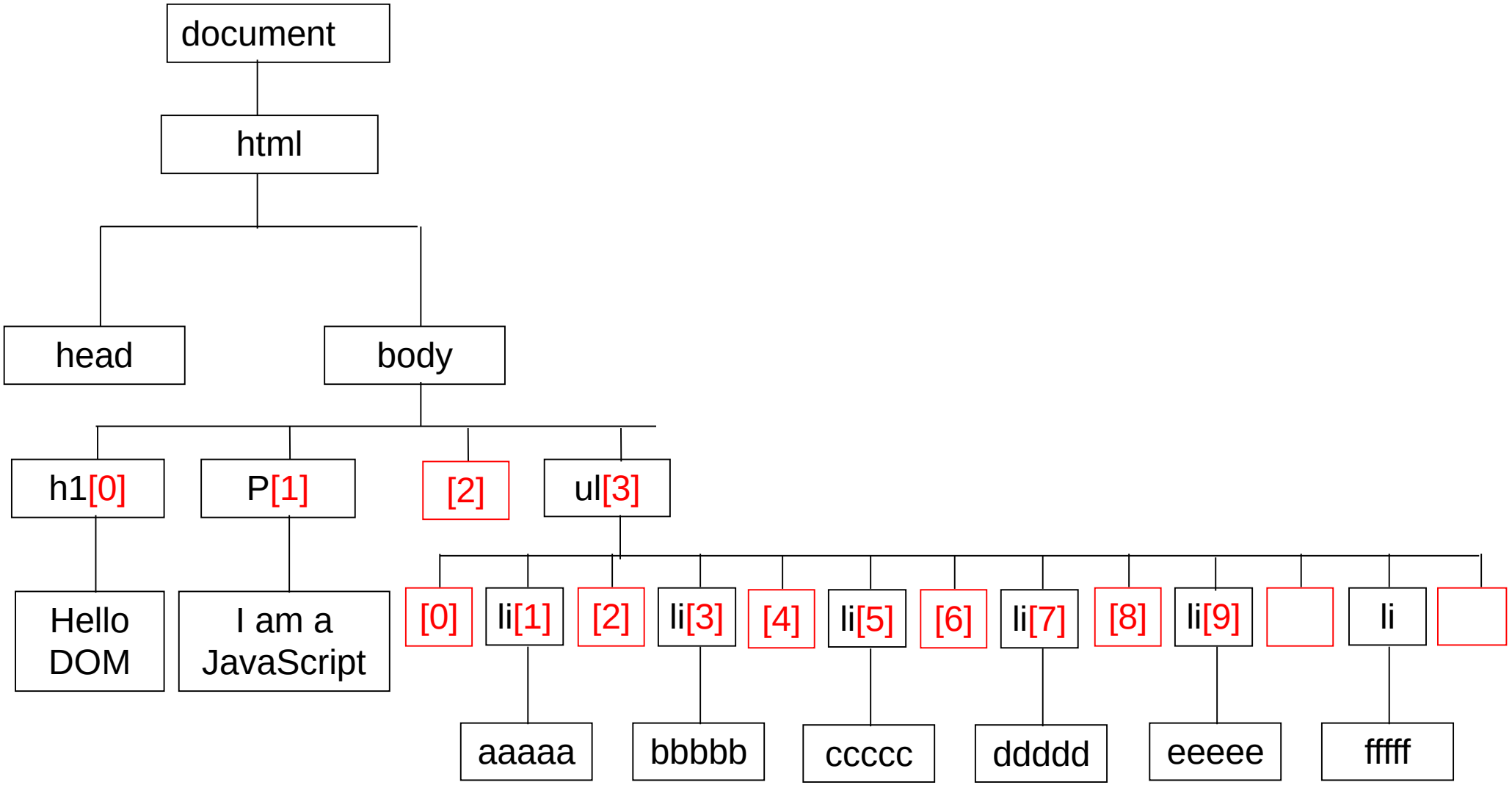
# 瀏覽節點



# 尋找及存取節點資料

- 透過document物件來使用節點物件的屬性及方法
- 節點彼此關係
  - documentElement
  - firstChild , lastChild , parentNode , childNodes
  - previousSibling , nextSibling
- 節點屬性
  - nodeType：節點類型
    - 9(Document)、1(Element)、3(Text)…..
    - https://www.w3.org/TR/2015/REC-dom-20151119/#dom-node-nodetype
    - http://www.w3schools.com/xml/dom_nodetype.asp
  - nodeName :大寫標籤名稱或#document 或 #text
  - nodeValue : Text或Comment節點的文字內容或null

# 範例



# 選取文件元素

- getElementById()：選取指定id屬性的元素
- getElementsByName()：選取指定name屬性元素
- getElementsByTagName()：選取指定標籤名稱元素
- getElementsByClassName()：選取指定類別名稱元素
- 根據CSS選擇器
  - https://www.w3.org/TR/css3-selectors/
  - https://www.w3.org/TR/selectors-api/
    - querySelector()
    - querySelectorAll()

# 範例程式

```
alert (document.documentElement.childNodes(1).nodeName);



var ps = document.getElementsByTagName("p");

for (var i=0;i<ps.length;i++)
{
     alert (ps.item(i).firstChild.nodeValue);
}
```

- item(index)—回傳指定索引值的節點

# 如何新增及移除元素

- 新增元素(成對標籤)
  - 步驟一 先建立文字內容 createTextNode
  - 步驟二
    - 再建立元素名稱 createElement
    - 或找到您要用的元素名稱
  - 步驟三 使用 appendChild 將文字內容依附在這個元素名稱之後
- 移除元素
  - 步驟一 找到您要刪除的元素
  - 步驟二 使用 node.parentNode.removeChild(node) 的方式來刪除節點

# 關於屬性資料

- 關於屬性資料(單一標籤)
  - 步驟一 找到相關的元素
  - 步驟二
    - 設定屬性
      - node.setAttribute ("屬性名稱", "屬性值")
    - 刪除屬性
      - node.removeAttribute ("屬性名稱")
    - 讀取屬性
      - node.getAttribute ("屬性名稱")

# 範例程式

```
var theDIV = document.getElementById("div1")

var eleP = document.createElement("p")
var txtP = document.createTextNode(“Hello World")
eleP.appendChild(txtP)

theDIV.appendChild(eleP)

var eleImg = document.createElement("img")
eleImg.setAttribute("src","images/b.gif")

theDIV.appendChild(eleImg)
```

# 單元十 :檔案處理

- File API
- Selecting Files
- Form input for selecting
- 讀取檔案資訊
- 讀取檔案內容
- 讀取檔案事件
- 以DataURL讀取檔案
- 顯示圖片

# File API

- 可直接在瀏覽器中讀取客戶端的文件，不需要將檔案傳到伺服端暫存後，瀏覽器再取回顯示。
  - 檢視上載圖片
  - Offline時的檔案儲存
- Interface
  - File : 讀取檔案相關訊息,如名稱,檔案大小
  - FileList : 表示透過檔案系統所選取的檔案陣列
  - FileReader : 提供不同的讀取檔案方式來讀取檔案內容
  - Blod : 裁切文件部分的二進制資料

# Selecting Files

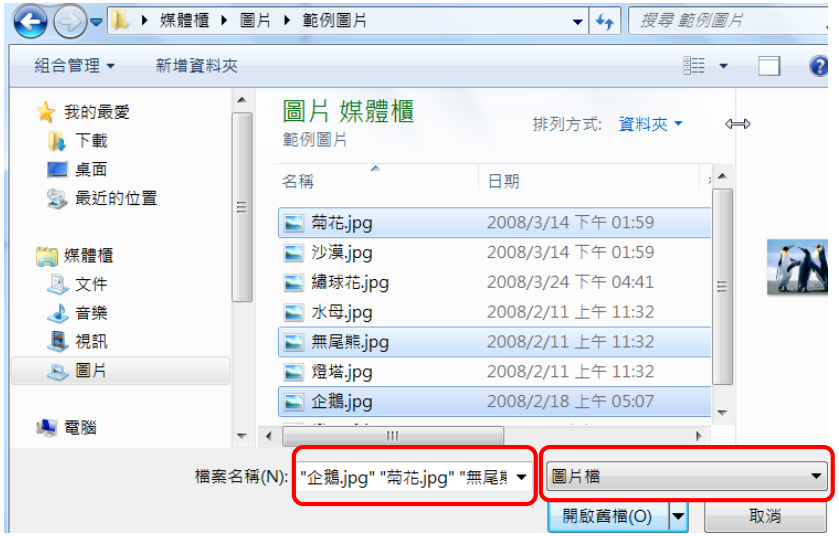
- 使用方式
  - 判斷瀏覽器是否支援FileAPI

```
if (window.File && window.FileReader && window.FileList && window.Blob) {
  // Great success! All the File APIs are supported.
} else {
  alert('The File APIs are not fully supported in this browser.');
}
```

  - 透過檔案選擇元件
    - Form input for selecting
    - Drag and drop Selecting

# Form input for selecting

- 可以選擇多個檔案
  - 設定multiple屬性
- 可過濾縮小選擇檔案類別
  - 設定accept屬性，指定MIME類型(image/* )




  - MIME(Multipurpose Internet Mail Extensions)類型和內容類型，在網際網路傳輸的標準。
    - 內容類型 `Content-Type: [type]/[subtype]; parameter`
    - 如：text/plain， text/html ， image/gif， image/jpeg， image/png
- 當使用者在檔案選擇標籤(<input type="file">)中選取檔案後，瀏覽器會建立對應的File Object，儲存於檔案選擇標籤的 **files** 屬性中

# 讀取檔案資訊

- file object
  - name,type,size,slice(start,end)

```
function fileInfo(){
    var files = document.getElementById("file1").files;
    for(var i=0;i<files.length;i++){
        var file = files[i];
        alert(file.name + ":" + file.type + "(" + file.size + ")");
    }
}
<input type="file" id="file1" multiple="multiple"
onchange="fileInfo();" />
```

# 讀取檔案內容

- FileReader object：使用非同步(asynchronous) 方法，即只負責讀取檔案的工作，但不等待資料回傳，讀取的資料會存於物件的 **result** 屬性。
  - 讀取方法(Method)：
    - readAsText(Blob|File, opt_encoding)：使用文字方式讀取資料內容，放到result屬性。
    - readAsDataURL(Blob|File)：使用dataURL編碼讀取資料內容，放到result屬性。
    - readAsBinaryString(Blob|File)：使用二進位方式讀取資料內容，放到result屬性。
    - readAsArrayBuffer(Blob|File)：以ArrayBuffer物件為讀取資料內容，放到rresult屬性。

# 讀取檔案事件

- Event：呼叫read..的方法時，會產生下列事件
  - onloadstart : 資料讀取開始時觸發
  - onprogress : 資料讀取中觸發
  - onload : 資料讀取成功完成時觸發
  - onabort : 資料讀取中斷時觸發
  - onerror: 資料讀取錯誤時觸發
  - onloadend: 資料讀取完成時觸發，無論成功或失敗都會觸發

# 讀取檔案內容範例

```
function fileviewer(){
   var reader = new FileReader();
   //onload資料讀取成功完成時觸發，表示資料已經準備好了
   reader.onload = function(e){
      var fileContent = e.target.result;
      var show = document.getElementById("show_box");
      show.innerHTML = fileContent;
   }
   reader.readAsText(document.getElementById("file1").files[0],
"UTF-8");
}
<input type="file" id="file1" multiple="multiple" onchange="fileviewer();" />
<hr />
<textarea id="show_box" readonly="true" cols="60" rows="20">
</textarea>
```

# 以DataURL讀取檔案

- Data URI scheme：即Uniform Resource Identifier(URI) scheme，直接把圖像的內容嵌入到網頁裡
- 語法:

```
data:[<media type>][;base64],<data>
```

  - Media type :即MIME(Multipurpose Internet Mail Extensions)類型和內容類型，在網際網路傳輸的標準。
    - 內容類型 `Content-Type: [type]/[subtype]; parameter`
    - 如：text/plain， image/gif， image/jpeg， image/png ，application/pdf
  - Base64： 將8-bit 資料轉成標準 ASCII 字元

```
<img src="
AAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO

9TXL0Y4OHwAAAABJRU5ErkJggg==" alt="Red dot" />
```

# 顯示圖片

```
function fileviewer(){
    var reader = new FileReader();
    reader.onload = function(e){
      var fileContent = e.target.result;
      var show = document.getElementById("img1");
      show.setAttribute("src", fileContent);
     }
     var file = document.getElementById(“file1”).files[0];
     reader.readAsDataURL(file);}
<input type="file" id="file1" multiple="multiple" onchange="fileviewer();" />
<hr />
<img id="img1" alt="Show Image" />
```

# 單元十一：Drag & Drop

- Drag & Drop API
- 元素拖曳
- 元素拖放

# Drag & Drop API

- 屬性
  - Draggable
- 事件
  - Dragstart：開始拖放操作
  - Dragenter：被拖放的元素開始進入目標元素範圍內
  - Dragover：被拖放的元素正在目標元素範圍內移動
  - Dragleave：被拖放的元素離開目標元素範圍
  - Drop：其他元素被拖放到目標元素中
  - Dragend：拖放操作結束

# Drag & Drop API

- dataTransfer物件 -- 拖放操作時產生的事件物件的屬性，儲存拖放時攜帶的資料
  - *event.dataTransfer*.setData(*format*, *data*) ：

    加入拖放攜帶的資料
  - *data* = event.*dataTransfer*.getData(*format*) ：

    取得拖放攜帶的資料
  - ***event.dataTransfer*.files**：

    取得拖放的檔案集合(FileList)

# 元素拖曳

- 做法
  - 設定draggable屬性為true
  - 透過dragstart事件，開始拖曳的動作，取得要拖曳元素的資料
  - 將取得的資料放進dataTransfer物件中

```
function dragstartHandler(e){
   e.dataTransfer.setData("text/plain",e.target.textContent);
}
<div id="dragItem" draggable="true"
ondragstart="dragstartHandler(event)">Drag me!!</div>
```

# 元素拖放

- 元素的拖放，要處理二個事件
  - dragover事件，被拖放的元素拖放到目標元素邊框上方時觸發
  - drop事件，將資料拖曳進入目標元素後放開滑鼠時觸發，處理從dataTransfer物件取出的資料

```
<div id="dropZone" ondragover="dragoverHandler(event) "
                        ondrop="dropHandler(event)">
    Drop Zone
</div>
```

# 拖放範例

```
function dragoverHandler(e){
    e.preventDefault();
}
function dropHandler(e){
    e.preventDefault();
    e.stopPropagation();
    var sourceid = e.dataTransfer.getData('text/plain');
    var source = document.getElementById(sourceid);
    e.currentTarget.appendChild(source.parentNode.
                                        removeChild(source));
}
```

# 單元十二:網頁儲存區

- 網頁儲存區
- Cookies
- Web Storage
- 存取儲存的值

# 網頁儲存區

- Cookie
- Web Storage
  - Local Storage
  - Session Storage
- IndexedDB

# Cookies

- 在用戶端的電腦裡儲存資料，最多可儲存4K。
- 使用 document.cookie 來存取cookie
- 存入cookie的字串格式

```
name=value;expires=date;path=pathname;domain=domainname
```

```
//寫入Cookie
document.cookie = "name=Jack;expires=Wed Dec 6 2016 20:47:11
GMT; path=/ ";

//讀取Cookie
var cookies = document.cookie;
alert(cookies.split("=")[1]);
```

# Web Storage

- localStorage
  - 不同的網站擁有自己的儲存空間，無法互相存取。
  - 資料是永存的(permanent)，會一直保存在用戶端的電腦中，直到Web apps 把資料刪除或使用者在瀏覽器刪除。
- sessionStorage
  - 不同的視窗(window)擁有自己的儲存空間，當視窗關閉後，儲存區的資料也會消失。
- 屬性
  - length
- 方法
  - Key(n) 、getItem(key) 、setItem(key,value) 、removeItem(key) 、clear()

# 存取儲存的值

- 設定值

```
localStorage.keyname = "value ";
```

- 取值

```
var theValue = localStorage.keyname;
```

- 刪除值

```
localStorage.removeItem("keyname ");          //刪除某筆資料

localStorage.clear();                         //刪除所有資料
```

# localStorage 範例

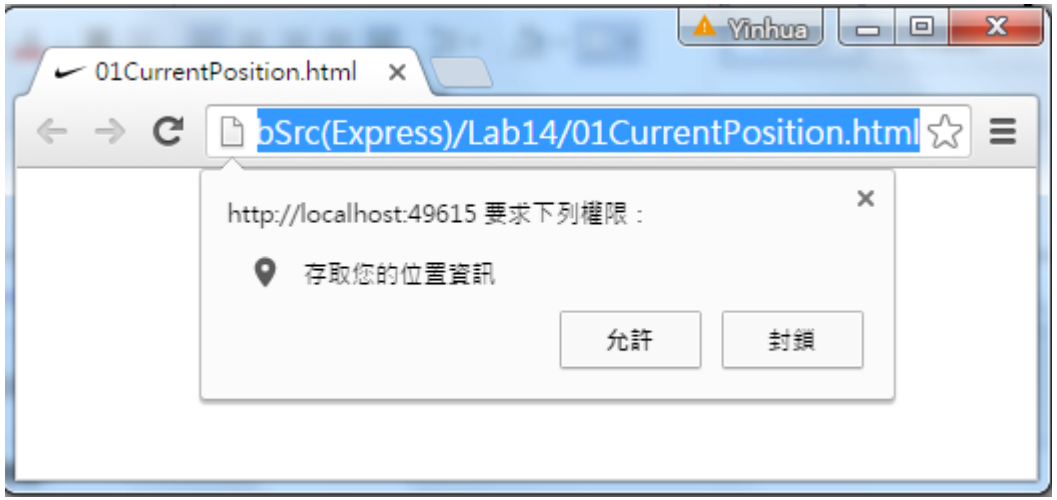
```
<p> You have viewed this page <span id="count">an untold number of</span>
time(s). </p>
<script>
    if (!localStorage.pageLoadCount) {
        localStorage.pageLoadCount = 0;
    }
    localStorage.pageLoadCount = parseInt(localStorage.pageLoadCount) + 1;
   document.getElementById('count').textContent = localStorage.pageLoadCount;
</script>
```

# 單元十三：Geolocation API

- Geolocation API
- 取得位置
- 取得位置成功時...
- 取得位置失敗處理
- 取得位置的參數

# Geolocation API

- 是Geolocation Work group 制訂的標準
- 透過瀏覽器取得電腦的IP及附近的無線網路狀況，傳給Google Location Services，計算您的地理位置
- 因隱私權的關係，瀏覽器需經過您的同意，才能送出訊息



# 取得位置

- 方法
  - getCurrentPosition(successCallback,errorCallback,options)
  - watchPosition()回傳使用者的目前位置，並且繼續回傳使用者移動的更新位置
  - clearWatch()停止 watchPosition() 方法

```
navigator.geolocation.getCurrentPosition(
    successCallback,              //取回地理位置成功呼叫此function
    errorCallback,                //取回地理位置失敗呼叫此function
    {maximumAge:600000});   //保留地理位置10分鐘
```

# 取得位置成功時...

- Position 物件
  - coords 屬性
    - Accuracy(位置準確性)、latitude(緯度)、longitude(經度)、altitude(海拔)、altitudeAccuracy(位置的海拔準確性)、heading(方向)、speed。
  - Timestamp(時間戳記) 屬性

```
function successCallback(position) {
    alert("Latitude: " + position.coords.latitude + ",
            Longitude: " + position.coords.longitude);
 }
```

# 取得位置失敗處理

- PositionError 物件
  - code 屬性
    - 0(UNKNOW_ERROR)
    - 1(PERMISSION_DENIED)
    - 2(POSITION_UNAVALIABLE)
    - 3(TIMEOUT)
  - message 屬性

```
function errorCallback(error) {
. alert("Error Code: " + error.code + ",
                                Error Message: " + error.message);
 }
```

# 取得位置的參數

- PositionOptions 物件
  - enableHighAccuracy ：取得高準確度的結果
  - maximumAge ：保留取得位置的時間
  - Timeout：取得位置超出時間會引發錯誤

```
navigator.geolocation.getCurrentPosition(
successCallback,
errorCallback,
{
    enableHightAccuracy:true,
    maximumAge:1000,              //Infinity,
    timeout:500
});
```

# 附錄一：Google Maps

- Google Map API用法
  - 參考網址：https://developers.google.com/maps/documentation/javascript/tutorial?hl=zh-TW
  - 準備HTML Form
  - 撰寫呼叫Google Map API的程式

# 準備HTML Form

- HTML form

```
<div id="map"></div>
```

- Javascript 程式碼
  - 載入 Google Map API

```
<script async defer
        src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap">
    </script>
```

  - 建立map物件，指定中心位置

```
var map;
function initMap() {
    map = new google.maps.Map(document.getElementById('map'), {
        center: { lat: 25.033656, lng: 121.54333599999995 },
        zoom: 13
    });

    var marker = new google.maps.Marker({ position: { lat: 25.033656, lng:
121.54333599999995 }, title: "Hello World!" });
    // To add the marker to the map, call setMap();
    marker.setMap(map);
}
```

# Google Maps API v3(1/2)

- LatLng(緯度, 經度) 類別
  - 建立指定的經度和緯度的地理座標，放在地圖中心點
- Map(map容器,選項)類別-在指定的 HTML 容器建立新地圖
  - 選項
    - zoom -- 初始的地圖縮放等級(0 代表完全縮小的地圖)
    - center -- 初始的地圖中心
    - mapTypeId -- 初始的地圖類型
      - ROADMAP 顯示 Google 地圖的正常、預設 2D 地圖方塊。
      - SATELLITE 可顯示攝影地圖方塊。
      - HYBRID 顯示混合攝影地圖方塊與重要地圖項 (道路、城市名稱) 的地圖方塊圖層
      - TERRAIN 顯示實際起伏的地圖方塊，以呈現海拔高度和水域圖徵 (山嶽、河流等)
- Marker(選項)-建立一個指定位置的標記
  - 選項
    - position -- 標記位置
    - title -- 變換文字

# Google Maps API v3(2/2)

- Geocoder 類別
  - 建立地理編碼，將地址轉換為經緯度
- geocode(GeocoderRequest, callback(GeocoderResult, GeocoderStatus))方法
  - GeocoderRequest – 設定建立地理編碼的相關資訊， 如address
  - GeocoderResult – 建立地理編碼回傳值
    - results[0].geometry.location –建立地理編碼的經緯度
  - GeocoderStatus –建立地理編碼回傳狀態

# 附錄二：Canvas 的使用

- <canvas>元素為一塊繪圖的區域，透過getContext()方法取得CanvasRenderingContext2D物件
- 繪圖的屬性及方法
  - 繪製線條
  - 繪製矩形
  - 繪製弧線
  - 處理圖像
  - 繪製文字

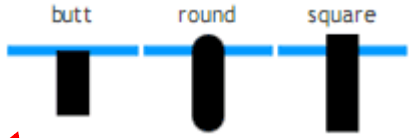
```
var canvas = document.getElementById(“myCanvas”);
var context = canvas.getContext("2d");
```

# 繪製線條方法與屬性

- beginPath() : 開始一條新路徑
- moveTo(*x*,*y*) : 設定一條新的子路徑的開始位置
- lineTo(*x*, *y*) : 在目前位置新增一條直線
- strokeStyle : 設定路徑的顏色、模式和漸變
- fillStyle : 填滿路徑的顏色、模式和漸變
- lineWidth : 設定線條寬度
- lineCap : 設定線條端點
- lineJoin : 設定線條連接點
- stroke() : 沿著當前目前路徑繪製一條直線
- fill() :填滿路徑內部
- closePath():如果目前路徑開啟，則連接到路徑開始位置

# 繪製線條

```
context.beginPath();                    //開始繪製路徑
context.moveTo(100,50);                 //移到某一點上(起始點)
context.lineTo(50,100);                 //設定線條的位置(終點)
context.lineTo(150,100);
context.lineTo(100,50);
context.strokeStyle = '#ff0000';        //線條樣式設定(顏色)
context.lineWidth   = 10;               //線條寬度
context.lineCap = "round";          //端點 round,butt(default),square
Context.lineJoin="round" ;              //連接點
round,bevel,miter(default)
context.fillStyle   = '#0000ff';        //填滿顏色
context.fill();                         //填滿
context.stroke();                       //開始畫線
context.closePath();                    //結束繪製路徑
```



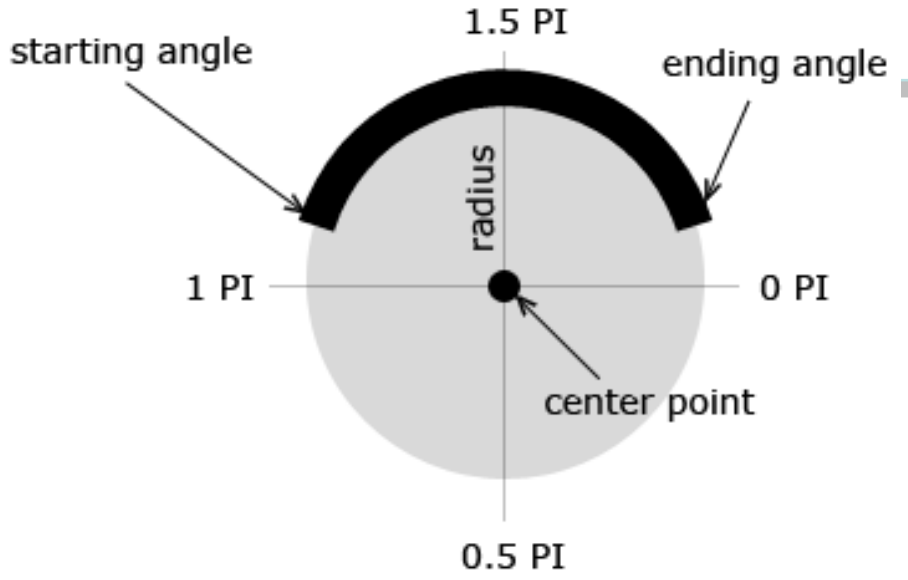

# 繪製矩形

- 畫矩形的兩個方法
  - fillRect(x,y,width.height)
  - strokeRect(x,y.width,height)
- 清除矩形
  - clearRect(x,y.width,height)

```
context.fillStyle = "#FF0000";
context.strokeStyle = '#0000ff';
context.fillRect(0,0,150,50);
context.strokeRect(0,60,150,50);
context.clearRect(30,25,90,60);
```

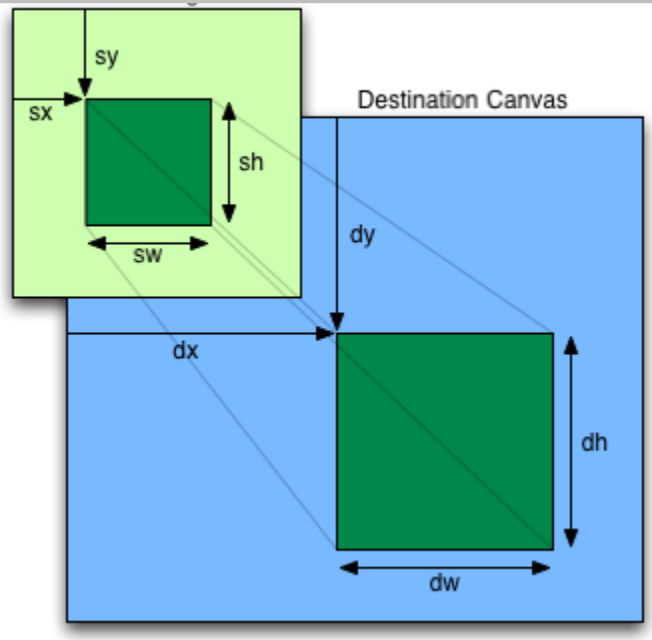
# 繪製弧線

```
var centerX = 150;
var centerY = 100;
var radius = 75;
var startingAngle = 1 * Math.PI;
var endingAngle = 0 * Math.PI;
var counterclockwise = false;
context.arc(centerX, centerY, radius, startingAngle,endingAngle, counterclockwise);
context.lineWidth = 15;
context.strokeStyle = "black"; // line color
context.stroke();
```



- Scale and crop images

```
drawImage(image, dx, dy)
drawImage(image, x, y, width, height)
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth,
dHeight)
```



# 處理圖像範例

```
var imageObj = new Image();
    imageObj.onload = function(){
        var sourceX = 150;
        var sourceY = 150;
        var sourceWidth = 150;
        var sourceHeight = 150;
        var destWidth = sourceWidth;
        var destHeight = sourceHeight;
        var destX = 20;
        var destY = 20;

 //context.drawImage(imageObj, destX, destY);
 //context.drawImage(imageObj, destX, destY, destWidth, destHeight);
context.drawImage(imageObj, sourceX, sourceY, sourceWidth, sourceHeight, destX,
destY, destWidth, destHeight);

    };
    imageObj.src = "cars2_logo.jpg";
```

# 繪製文字方法與屬性

- font:設定文字的 *font-style*、*font-weight*、*font-size*、*font-family*
- textAlign : 設定文字內容的水平對齊(start 、left、center 、end 、right)
- textBaseline :設定文字內容的垂直對齊(top、hanging、middle、bottom、alphabetic)
- fillText(*text,x,y*) : 依據座標位置填滿文字內部
- strokeText(*text,x,y*): 依據座標位置繪製文字外框

# 繪製文字

```
var x = canvas.width / 2;
var y = canvas.height / 2;
context.font = "30pt Calibri";
context.textAlign = "center";
context.textBaseline = "middle";
context.fillStyle = "#0000ff"; // text color
context.fillText("Hello World!", x, y);

context.strokeStyle = "blue"; // stroke color
context.strokeText("Hello World!", x, y);
```