



JavaScript 開發實戰

核心概念篇



多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：<http://blog.miniasp.com/>

課程大綱

- 物件、變數與型別
- JavaScript 基礎物件概念
- JavaScript 物件導向基礎



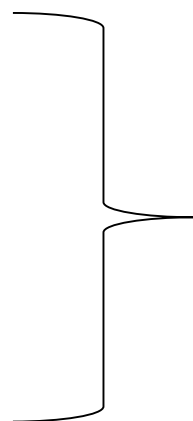
JavaScript 物件可以指派給一個變數並會在執行時期擁有型別

物件、變數與型別

JavaScript 是個物件導向程式語言

- 所有東西都是**物件型別**，除了：

- 數值 (number)
- 字串 (string)
- 布林 (boolean)
- null
- undefined



原始型別 (Primitive Type)

- 原始型別包裹 (primitive wrappers)

- number = Number
- string = String
- boolean = Boolean

JavaScript 物件是個容器 (Container)

- 每個 JavaScript 物件**僅**包含

- 屬性 (Property)

- 物件 (原始型別、物件型別)
 - 函式 (Function)

- 範例

- `var car = {};` // 物件 (Object)
 - `car.name = "BMW";` // 屬性 (Property)
 - `car.start = function () {` // 函式 (Function)
 `return "OK";`
 `};`

JavaScript 物件是個雜湊陣列(HashMap)

- JavaScript 取得物件內容的方法

- `var car = {
 'name': 'BMW',
 'start': function () { return "OK"; },
 '001': 'LogEntry#1'
};`

- `car.name`

- `car['name']`

- `car.001` ??

- `car['001']`

- `window.document.forms[0]`

- <http://utf-8.jp/public/aaencode.html>

JavaScript 是個動態型別語言

- JavaScript 使用 var 宣告變數

- var x = 5;

- var x;

- typeof(x)

- x = "Will";

- 無型別 (Untyped)
 - 無法在**開發時期**宣告型別
- 弱型別 (Weak-typed)
 - 只能在**執行時期**檢查型別

物件、變數與型別之間的關係

- 物件
 - 記憶體中的**資料**
 - 僅存在於**執行時期**
- 變數
 - 用來儲存**物件**的記憶體位址 (指標)
 - 在**開發時期**進行宣告 (使用 var 關鍵字)
- 型別
 - 用來標示**物件**的種類
 - 不同型別可能會有不同的預設**屬性與方法**

物件、變數與型別之間的關係 (範例)

- 以下 4 行程式碼在執行的過程中，請問：
出現過幾個**記憶體物件**？
出現過幾個**變數**？
出現過幾種**型別**？

```
var a;
```

```
a = 1;
```

```
a = "a";
```

```
a = "a" + a;
```

變數與屬性之間的關係

- 屬性
 - 物件的屬性可以是任意物件 (含原始型別)
 - 兩種指派方法
 - `window.myKey = 1;`
 - `window['myKey'] = 1;`
- 變數
 - 用來儲存物件的記憶體位址 (指標)
 - 在開發時期進行宣告 (使用 `var` 關鍵字)
 - 預設會變成物件容器的屬性 (但無法刪除)

變數與屬性之間的關係

- 變數

- `var a = 1;`
- `var b = a;`

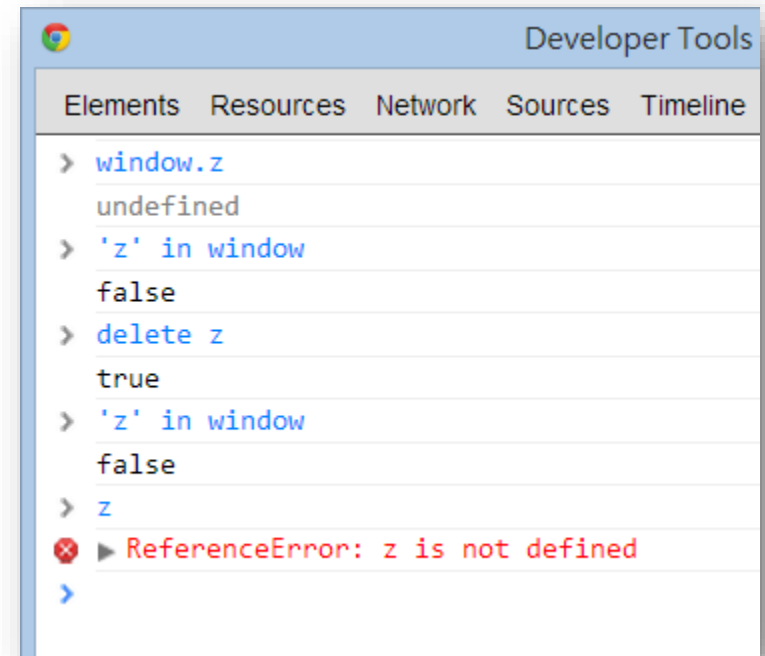
- 屬性

- `var a = b;`
- `var a = window.b;`
- `b = 1;`
- `window.b = 1;`
- `delete b;`
- `var a = b;`

變數與屬性之間的關係 (範例)

- 以下程式碼片段，請問輸出為何？
 - `var a = 1;`
 - `window['a'] = 2;`
 - `delete window.a;`
 - `console.log(a);`

 - `b = 2;`
 - `delete b;`
 - `console.log(b);`

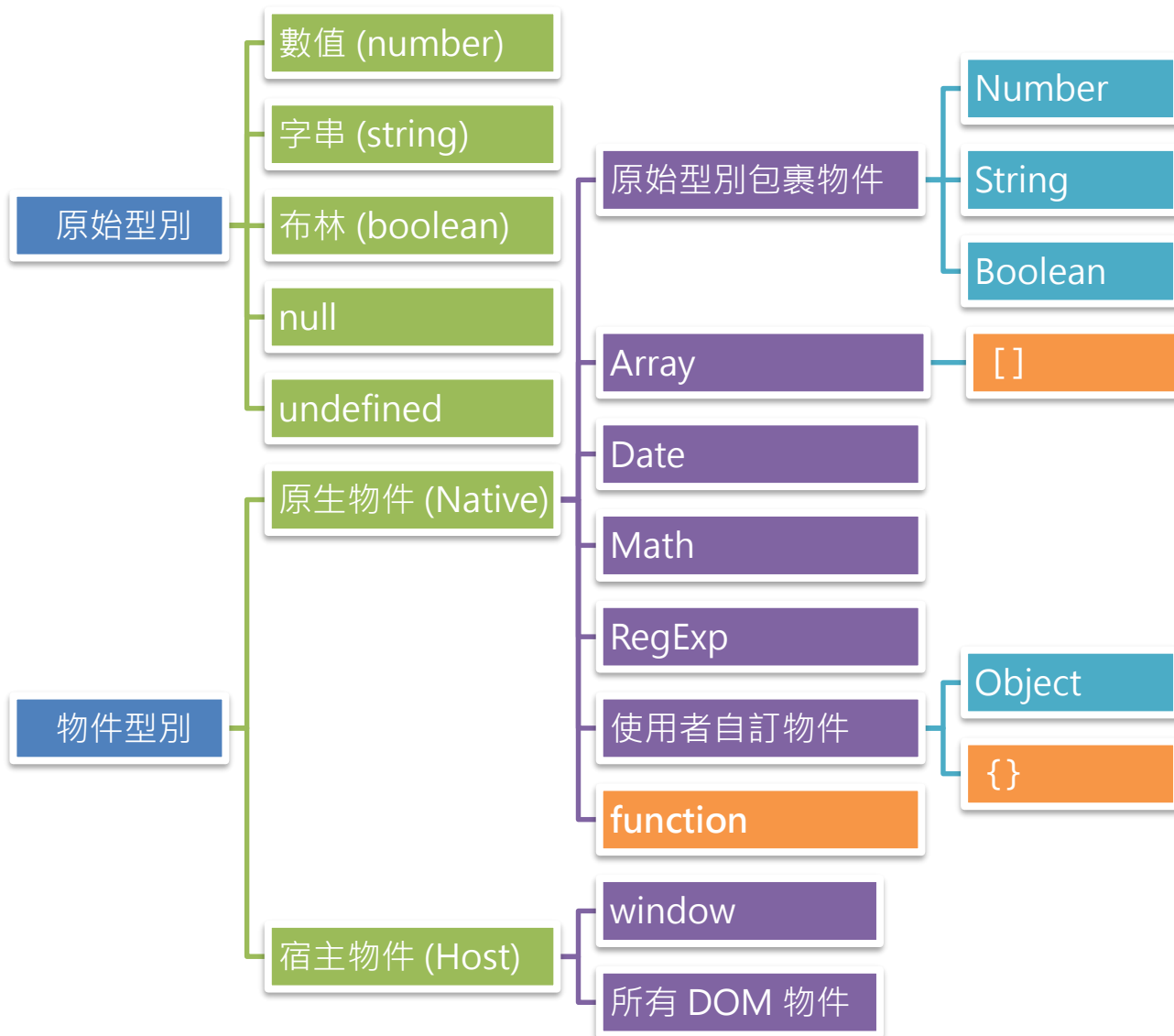


了解不同型別之間的特性

型別系統



JavaScript 有哪些**型別**可用？



JavaScript 型別有兩大分類

- 原始型別 (Primitive Type)

- 數值 (number)
- 字串 (string)
- 布林 (boolean)
- null
- undefined

無法「自由擴增屬性」

- 物件型別 (Object Type)

- 原生物件
 - 函式物件算原生物件
- 宿主物件

可以「自由擴增屬性」

何謂【自由擴增屬性】？

- 物件範例

- `var obj = { 'a': 1, 'b': 2 };`

- 擴增屬性

- `obj.c = 3;`

- 刪除屬性

- `delete obj.c;`

- 判斷屬性是否存在

- `if ('c' in obj) { }`

- `typeof(obj.c) == 'undefined'`

➔ 錯誤用法

- 須注意「繼承屬性」的判斷！

原始型別包裹物件

- 主要用途
 - 由於**原始型別**無法自由擴增屬性
 - 透過包裹物件後
 - 可以讓**原始型別**擁有包裹物件的**屬性與方法**
 - 可以透過**物件型別**的特性，自由擴增**屬性與方法**
- 共通方法
 - valueOf()
 - 取得**物件**內部的**原始值** (Primitive Value)
 - toString()
 - 將**物件**內部的**原始值** (Primitive Value)轉換成**字串**

原始型別

JavaScript 原始型別：數值 (number)

- 使用方法

- **var a = 100;**

- var a = 10e5;

- var a = 0100;

- var a = 0xFFFF;

- var a = new Number(100);

- var a = Number('100'); → 轉型

- var a = Number('100a'); → 轉型失敗 → **NaN**

- **var a = parseInt('100 aa', 10); → 字串解析**

Not a number is a number!



- 原始型別包裹物件

- Number 內建屬性與方法說明

- http://www.w3schools.com/jsref/jsref_obj_number.asp

數值型別使用技巧

- 轉型方法

- `+'070'`

- `Number('070')`

- `parseInt('070')` → IE8 以下會變成 8 進制

- `parseInt('070', 10)` → 建議寫法！

- 最佳實務

- 使用 `parseInt` / `parseFloat`

- 一律加上第 2 個參數，明確指定基數！

- 使用 `+` 強迫引發自動轉型

- `var a = '7'; var b = +(a)`

- 使用 `(N).toString(baseN)` 轉換進制

- `(0xAF).toString(10)`

- `65535..toString(16)`

JavaScript 原始型別：字串 (string)

- 使用方法

- `var a = 'Will';`
- `var a = new String('Will');`
 - `String {0: "W", 1: "i", 2: "l", 3: "l"}`
- `var a = String(100);`
- `var b = a[0];` ➔ 取得第一個字元

- 原始型別包裹物件

- String 內建屬性與方法說明
 - http://www.w3schools.com/jsref/jsref_obj_string.asp
- 其他重點
 - `length` 屬性
 - `String` 包裹物件提供一組 HTML 相關的操作方法

JavaScript 原始型別：布林 (boolean)

- 使用方法

- `var a = true;`
- `var a = false;`
- `var a = new Boolean(false);`
- `var a = Boolean('false');`
- `var a = Boolean('0');`
- `var a = Boolean('');`
- `var a = Boolean(0);`
- `var a = !!('');`

- 原始型別包裹物件

- Boolean 內建屬性與方法說明
 - http://www.w3schools.com/jsref/jsref_obj_boolean.asp

認識 Truthy 與 Falsy 值

- `false` , `0` , `""`

- `var a = (false == 0);` // `true`
 - `var b = (false == "");` // `true`
 - `var c = (0 == "");` // `true`

- `null` , `undefined`

※ `null` 只能跟 `null` 與 `undefined` 比較

- `var d = (null == false);` // `false`
 - `var e = (null == true);` // `false`
 - `var d = (undefined == false);` // `false`
 - `var e = (undefined == true);` // `false`
 - `var f = (null == null);` // `true`
 - `var g = (undefined == undefined);` // `true`
 - `var h = (undefined == null);` // `true`

- `NaN`

※ `NaN` 跟任何物件比較都是 `false`

- `var i = (NaN == null);` // `false`
 - `var j = (NaN == NaN);` // `false`

布林型別使用技巧

- 隱含比對 vs. 明確比對
 - == 或 != (隱含比對，會引發自動轉型)
 - === 或 !== (明確比對)
- 最佳實務
 - 一律使用 === 或 !== 比對
 - 避免 falsy value 判斷!
 - 使用 !! 強迫引發自動轉型
 - var a = !! (0);
 - var b = !! ("0");
 - JS Comparison Table
<http://dorey.github.io/JavaScript-Equality-Table/>

Falsy Value 的使用技巧

- 判斷物件是否有初始值
 - `if(myVar) {`

`}`
- 給予變數預設值
 - `var arr = arr || [];`
 - `var num = num || 99;`
 - `var str = str || "";`
 - `var bool = bool || true;`
 - `var obj = obj || {};`

JavaScript 原始型別：空值 (null)

- 使用方法
 - `var a = null;`
- 重點觀念
 - null 跟 NaN 有點類似
 - NaN 不是一個**數字**，但型別卻是個**數字** ("number")
 - null 不是一個**物件**，但型別卻是個**物件** ("object")
- 最佳實務
 - 盡量不要用 null
 - 判斷物件是否為 null 的正確方法
 - `var a = null;`
 - `if (a === null) {`
 }

JavaScript 原始型別：未定義 (undefined)

- 使用方法

- `var a; // undefined`
- `a = 1; a = undefined;`

- 重點觀念

- `undefined` (型別) 是一個內建型別 (原始型別)
- `undefined` (物件) 是 `undefined` (型別) 的值
- `undefined` (變數) 是一個全域變數 (也是個屬性)
 - `window.undefined === undefined`
 - 在 IE8 以下 (含) `undefined` 可以被重新指派成其他物件！
- 無論 `null` 或是 `undefined` 都會隱含轉型成 `false`
 - `undefined == null // true`

物件型別

JavaScript 物件型別

- 原生物件 (Native Objects)
 - 於 ECMAScript 標準中定義的物件型別
 - 使用者定義物件型別
 - `var o = {};`
 - 內建物件型別
 - Array, Date, Math, RegExp
 - Number, String, Boolean, Object
- 宿主物件 (Host Objects)
 - 由 JavaScript **執行環境**額外提供的物件
 - [window](#) (Browser), 所有 DOM 物件 (Browser)
 - [global](#) (Node.js) , [process](#) (Node.js)

JS 原生物件：使用者定義物件

- 如何建立一個物件？

- `var obj = new Object();`
 `obj.name = 'Will';`
 `obj.company = '多奇數位創意有限公司';`

- 透過 **物件實字** (Object Literal) 建立物件

- `var obj = {};`
 `obj.name = 'Will';`
 `obj.company = '多奇數位創意有限公司';`

- `var obj = {`
 `0: 'test',` // 數字
 `name: 'Will',` // 識別子
 `'company': '多奇數位創意有限公司'` // 字串
 `};`

JS 原生物件：Array

- 使用方法 1

- `var mycars = new Array();`
- `mycars[0] = "Will";`
- `mycars[1] = "Web";`
- `mycars.push("The");`

- 使用方法 2

- `var mycars = [];`
- `mycars[0] = "Will";`
- `mycars[1] = "Web";`
- `mycars.push('The');`

- Array 內建屬性與方法說明

- http://www.w3schools.com/jsref/jsref_obj_array.asp

Elements	Resources	Network	Elements	Resources	Network
> var a = [5];	undefined		> var a = [5,10];	undefined	
> var b = new Array(5);	undefined		> var b = new Array(5,10);	undefined	
> a	[5]		> a	[5, 10]	
> b	[undefined × 5]		> b	[5, 10]	
> a.length	1		> a.length	2	
> b.length	5		> b.length	2	
>			>		

JS 原生物件：Date

- 使用方法

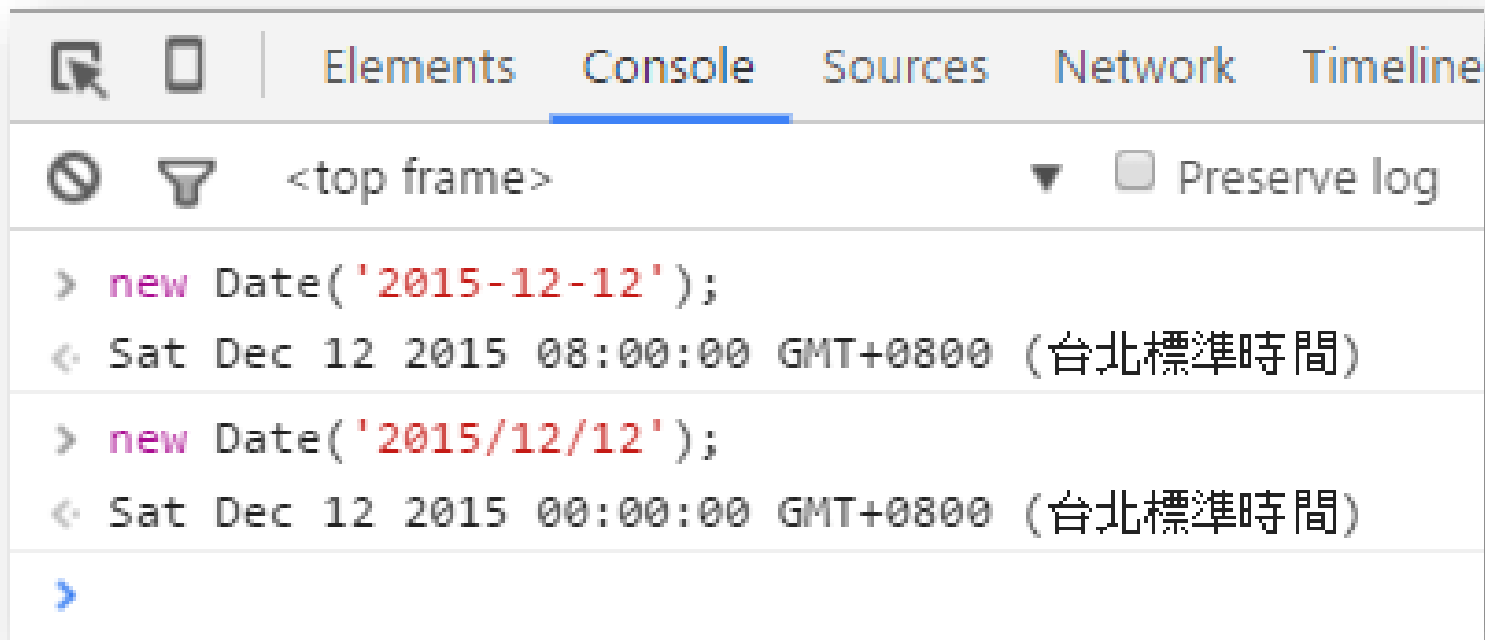
- `var d = new Date();`
- `var d = new Date(milliseconds);`
 - 這是 GMT 時區的 Timestamp
- `var d = new Date(dateString);`
- `var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);`
 - 請注意 `month` 的數字範圍是從 0 ~ 11

- Date 內建屬性與方法說明

- http://www.w3schools.com/jsref/jsref_obj_date.asp
- [前端工程研究：關於 JavaScript 中 Date 型別的常見地雷與建議作法](#)

關於 Date 字串格式與時區的關係

- `new Date('2015-12-12');`
- `new Date('2015/12/12');`
- `new Date('2015-12-12 00:00:00');`
- `new Date('2015/12/12 00:00:00');`



JS 原生物件：Math

- 使用方法

- `var a = Math.PI;` // PI 常數
 - `var a = Math.sqrt(16);` // 開根號(4)
 - `var a = Math.round(2.5);` // 四捨五入(3)
 - `var a = Math.floor(1.8);` // 無條件捨去(1)
 - `var a = Math.ceil(1.1);` // 無條件進位(2)

- Math 內建屬性與方法說明

- http://www.w3schools.com/jsref/jsref_obj_math.asp

JS 原生物件：RegExp

- 使用方法 1

- `var re = new RegExp(pattern, modifiers);`
- `var re = new RegExp("[A-Z][12]\\d{8}", "igm");`

- 使用方法 2

- `var re = /pattern/igm;`
- `var re = /[A-Z][12]\\d{8}/igm;`

- RegExp 內建屬性與方法說明

- http://www.w3schools.com/jsref/jsref_obj_regexp.asp

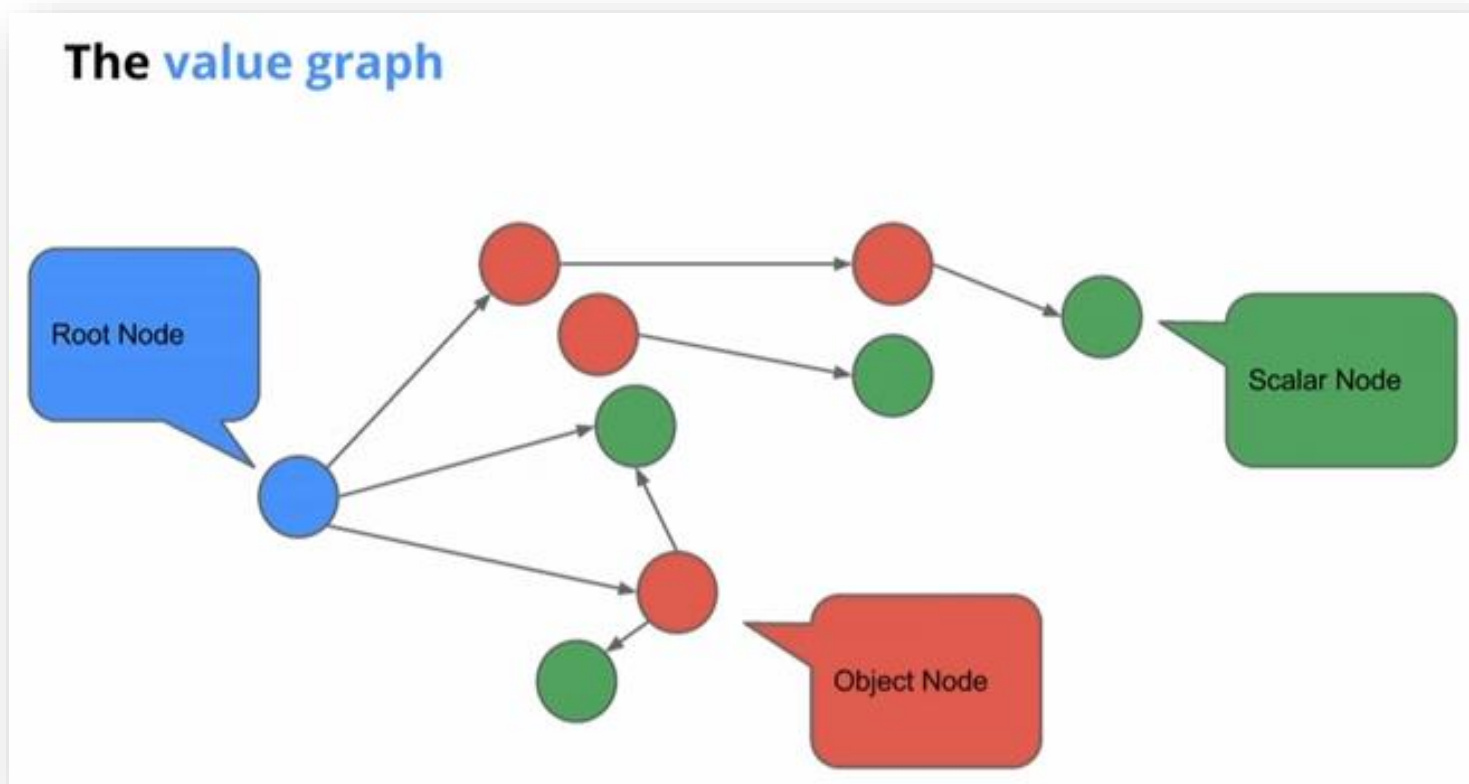


你所不知道的 JavaScript 程式語言特性

JAVASCRIPT 基礎物件概念

JavaScript 物件資料結構

- 所有物件資料都從**根物件**開始**連結**(chain)



探討變數與物件之間的連結 (1)

- 物件

```
window.document.myobj = { 'num': 1 };
```

- 變數

```
var o = window.document.myobj;
```

- 型別

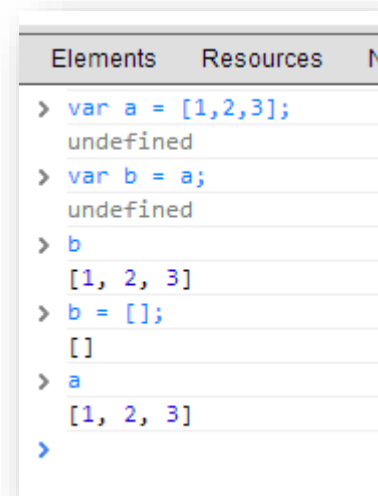
```
typeof (o.num)
```

- 重新指派變數值

```
o.num = 2;
```

- 請問以下陳述式是否為真？

```
o == window.document.myobj
```



Elements	Resources	N
> var a = [1,2,3];		undefined
> var b = a;		undefined
> b		[1, 2, 3]
> b = [];		[]
> a		[1, 2, 3]
>		

探討變數與物件之間的連結 (2)

- 物件

`window.document.forms[0]`

- 變數

`var o = window.document.forms[0];`

- 型別

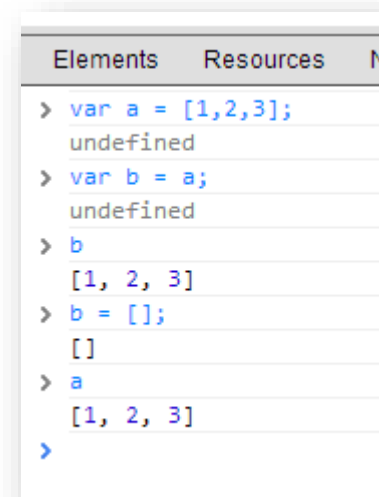
`typeof (o)`

- 重新指派變數

`o = "123";`

- 請問以下陳述式是否為真？

`o == window.document.forms[0]`



A screenshot of a JavaScript console window with three tabs: 'Elements', 'Resources', and 'N'. The console shows the following sequence of commands and their outputs:

- `> var a = [1,2,3];` followed by `undefined`
- `> var b = a;` followed by `undefined`
- `> b` followed by `[1, 2, 3]`
- `> b = [];` followed by `[]`
- `> a` followed by `[1, 2, 3]`
- `>` followed by a blue prompt character.

任何 JavaScript 執行環境都有個根物件

- 瀏覽器
 - [window](#)
- Node.js
 - [global](#)
- AngularJS (非JS執行環境，但沿用其概念)
 - [\\$rootScope](#)

瀏覽器的 根物件 (window) 有什麼？

- 屬性

- Infinity
- NaN

- undefined

- 方法

- decodeURI()
- decodeURIComponent()
- encodeURI()
- encodeURIComponent()
- escape()
- unescape()
- eval()

- isFinite()
- isNaN()
- parseFloat()
- parseInt()
- String()
- Number()
- Boolean()

- 保留字 (JavaScript Reserved Words)

- <http://msdn.microsoft.com/en-us/library/ie/0779sbks.aspx>

函式物件 (function)

- function 是個特殊的物件
 - `var a = function () { }`
 - `typeof (a)`
 - `a.x = 1;`
 - `a.x`
- function 的兩大特色
 - JS 的**一級物件** (first-class object)
 - 可以被動態建立
 - 可以指定給變數，也可以複製給其他變數
 - 可以擁有自己的**屬性**或**方法** (物件特性)
 - 提供了**變數的作用域** (scope)
 - 不以區塊 `{ }` 建立作用域，有別於其他程式語言！

函式的表示法

	函式表示式 (function expression)	函式宣告式 (function declaration)
具名函式	<pre>var add = function add(a, b) { return a+b; }; add.name</pre>	<pre>function add(a, b) { return a+b; } add.name</pre>
匿名函式	<pre>var add = function (a, b) { return a+b; }; add.name</pre>	<pre>function (a, b) { return a+b; }</pre> <p>Uncaught SyntaxError: Unexpected token (</p>

全域變數 vs. 區域變數

- 宣告變數時
 - 請一律透過 **var** 宣告變數
 - 變數宣告會讓物件自動成為物件容器的屬性
- 區域變數
 - 區域變數的範圍是靠 function 區隔的
(並非以大括號當作範圍)
 - 宣告變數時，會成為內部物件的屬性/變數
- 全域變數
 - 意即 根物件 (Root Object) 的屬性/變數

使用 var 關鍵字宣告變數

- 忘記使用 var 關鍵字~~宣告變數~~
 - 根本不是個變數，而是**全域物件**的一個**屬性**
 - 有機會透過 **delete** 運算子刪除該~~變數~~ (屬性)
- 使用 var 關鍵字宣告變數
 - **標準做法**
 - 無法使用 **delete** 運算子刪除該變數 (屬性)
 - 注意 JS 的 **Hoisting** (提升) 特性
 - 所有 **var** 命令將會自動提升順序到函式最前面!
 - 所有 **函式宣告式** (function declaration) 也會!!

分散 var 變數宣告的問題: Hoisting

- 範例

```
bookname = "MVC4";  
function MyOldBook() {  
    alert(bookname);  
    var bookname = "MVC2";  
    alert(bookname);  
}
```

- 特性

- JS 允許函式在任何位置使用 var 宣告變數
- 執行時期，所有 var 變數會自動提升到函數開始執行的地方

立即函式 (Immediate Function)

- 函式宣告式 + 匿名函式
 - 執行範例

```
(function (a, b) {  
    var c = 10;  
    return a+b+c;  
})(10, 20);
```
 - 也可以接到執行結果。
- 主要用途
 - 限制變數存在的作用域！
 - 讓變數不輕易成為「全域變數」的方法！

回呼模式 (Callback Pattern)

- 將函式當成參數傳遞給其他函式
 - ```
function get(url, callback) {
 var html = GetSomething(url);
 callback(html);
}
```
- 以具名函式方式傳入
  - ```
function callback(data) {  
    $('result').html(data);  
    alert('Load was performed.');
```
 - ```
get('ajax/test.html', callback);
```



# 閉包 (Closure)

- 觀念回顧：Function 的兩大特色
  - JS 的一級物件 (first-class object)
  - 提供了變數的作用域 (scope)
- 閉包的特性
  - 在特定函式中存取另一個函式的變數
  - 外層函式宣告的變數可以在內層函式中取用
  - 重點觀念：作用域鏈結 (Scope Chain)
- 閉包的程式碼外觀
  - 運用在巢狀函式的定義中



Prototype-based Object-Oriented

# JAVASCRIPT 物件導向基礎

# 函式 (function) 與 建構式 (constructor)

- JavaScript 沒有 class
  - 代表你不用事先建立藍圖，就能建立物件
- JavaScript 透過「建構式」建立物件藍圖
  - **建構式**就是**函式**，又稱**建構式函式**
  - 建構式範例：
    - ```
var Car = function (name) {  
    this.name = name;  
    this.slogan = function () {  
        return 'Driven by passion. ' + this.name + '!';  
    }  
}
```

透過 建構式 (constructor) 建立物件

- 使用 new 關鍵字

- 建構式定義

```
var Car = function (name) {  
    this.name = name;  
    this.slogan = function () {  
        return 'Driven by passion. ' + this.name + '.';  
    }  
}
```

- 建立物件

```
var mycar = new Car('FIAT');
```

建構式函式在建立物件時的執行過程

```
var Car = function (name) {  
    // 此時的 this 等於本次要被建立的物件實體 (obj)  
    // var this = {};  
    this.name = name;  
    this.slogan = function () {  
        return 'Yo! ' + this.name + '!';  
    }  
    // return this;  
}  
  
var obj = new Car();
```

函式 與 建構式 的 this

- 建構式 (函式)
 - this 代表「建立物件時物件實體」
 - `var obj = new Lesson();`
- 函式
 - this 代表「根物件」(window)
 - `var obj = Lesson();`
 - 會汙染根物件！(例如: `window.name` 會被竄改)
- 測試
 - `this === window`

原型物件 (Prototype Object)

- 原型物件的特性
 - 只有函式物件才擁有**公開的**原型物件
 - 所有**其他物件**僅擁有**私有的**原型物件
 - 代表物件實體的上層物件 (父物件)
 - 所有物件實體會自動**繼承**原型物件 (一種說法)
 - **原型物件**的目的在於建立**物件之間的鏈結關係!**
- 取得上層物件的方法
 - `function Car(name) { this.name = name; }`
 - `var o = new Car();`
 - `Object.getPrototypeOf(o) === Car.prototype`

建立物件並繼承其他物件

- 定義建構式(函式)
 - `var Lesson = function () {
 this.name = 'Will';
}`
- 設定原型物件屬性 (上層物件預設為空的 Object 物件)
 - `Lesson.prototype.name = 'Huang';`
- 建立物件實體
 - `var obj = new Lesson();` // 從建構式複製this物件
 - `obj`
 - `obj.name`
 - `delete obj.name`
 - `obj.name`

建構式函式在建立物件時的流程 (簡單)

```
var Car = function (name) {  
    // 建立物件實體時，this 已經繼承自原型物件!  
    // var this = {};  
    this.name = name;  
    this.slogan = function () {  
        return 'Yo! ' + this.name + '!';  
    }  
    // return this;  
}  
Car.prototype.name = 'Huang';  
  
var obj = new Car();
```

原型鏈結 (prototype chain)

- 鏈結順序
 - 0: 物件本身擁有的屬性
 - 1: 物件內部 `__proto__` 物件的屬性
 - 繼承自 prototype 的屬性
 - 接著會一直不斷的透過 `__proto__` 物件找下去，直到最後遇到 Object 物件為止
 - 註: 根物件 `!= Object`
 - scope chain 與 prototype chain 是完全不同的東西

檢查物件屬性是否來自原型物件

- 範例程式

- var Lesson = function () {
- this.name = 'Will';
- }
- Lesson.prototype.name = 'Huang';

- 檢查是否從特定建構式繼承該屬性

- var obj = new Lesson();
- obj.hasOwnProperty('name')
- delete obj.name
- obj.hasOwnProperty('name')

物件繼承的寫法

- 宣告上層物件藍圖
 - ```
function 哺乳類動物() {
 this.name = "哺乳類動物";
}
```

```
哺乳類動物.prototype.name = '動物界';
```
- 宣告下層物件藍圖
  - ```
function 貓咪() {  
    this.name = "貓咪";  
}
```
- 建立物件繼承關係
 - ```
貓咪.prototype = new 哺乳類動物();
```

 //鏈結上層物件
  - ```
貓咪.prototype.constructor = 貓咪;
```

 //重設建構式為自己
- 建立物件
 - ```
var obj = new 貓咪();
```

# 原型鏈結 (prototype chain) 範例解說

- 使用「原始型別」當屬性的**風險**
  - <http://jsbin.com/willh-prototype-chain-demo/1/edit?javascript>
  - 原型鏈結的潛在風險
- 使用「物件型別」當屬性的**優點**
  - <http://jsbin.com/willh-prototype-chain-demo/7/edit>
  - 確保原型鏈結的查找行為變的一致
  - **密技**：只要多一個 "." 就可以確保物件一致!

# 檢查物件繼承的方法

- 檢查是否從特定建構式繼承
  - `obj instanceof Lesson`
  - `obj instanceof Lesson.prototype.constructor`
  - `obj instanceof Object`
  - `obj instanceof Number`

- 常用技巧

```
function Person(name){
 if (!(this instanceof Person))
 return new Person(name);
 this.name = name;
}
```

# apply & call

- 建構式
  - `var Lesson = function (p1, p2) {  
    this.name = 'Will';   return this;  
}`
- `apply`
  - 建構式.`apply`(物件, [參數1, 參數2]);
- `call`
  - 建構式.`call`(物件, 參數1, 參數2, ...);
- 目的
  - 把「傳入物件」當成建構式的 **this**

# 聯絡資訊

- The Will Will Web

記載著 Will 在網路世界的學習心得與技術分享

- <http://blog.miniasp.com/>

- Will 保哥的技術交流中心 (臉書粉絲專頁)

- <http://www.facebook.com/will.fans>



- Will 保哥的噗浪

- <http://www.plurk.com/willh/invite>

- Will 保哥的推特

- [https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)