Node.js & HTML5

開發思惟與入門學習

Thinking and Getting Started

Jollen Chen

Node.js & HTML5 開發思惟與入門學習

Thinking and Getting Started

Jollen Chen

This book is for sale at
http://leanpub.com/html5-javascript-thinking

This version was published on 2013-10-22

Contents

1 JavaScript 設計模式

JavaScript 或許不是物件導向式語言（OOP），因為目前流行的 JavaScript 語法並沒有很明顯的 OO 特徵。但是，JavaScript 編程時，到處充滿物件導向的觀念。本章針對 JavaScript 的入門觀念，進行簡要的整理。

對初學者而言，程式語法是重要的課程，但是要入門 JavaScript 編程，學習語法還不夠。幾個重要的入門觀念，不但能幫助初學者寫好程式，更可以說是 JavaScript 真正的入門課程：

- Object & Function
- Instantiable Function
- Callback & Lambda
- Inheritance & Prototype
- MVP 設計模式
- MVVM 架構模式

1.1 Object

JavaScript 第一個觀念，就是「物件生成」。利用 function 關鍵字來宣告類別，並且利用 new 關鍵字來實例化，是生成物件最簡單的寫法。另外一個做法是利用 Object.create() 方法。

在 JavaScript 裡，也可以直接宣告物件。利用一對大括號所宣告出來的變數，都是物件。

JavaScript 不是強型別（Strong data type）的程式語言，任何的變數宣告，只要使用 var 關鍵字即可。要生成（Create）物件時也一樣，以下是一個範例：

```
var person = {
        name: "Jollen",
        job: "Software Developer",

        queryJob: function() {
                alert(this.job);
        }
};
```

上述的表示式執行後，可以得到 person 物件。如同典型的物件導向觀念，在物件裡會有 attribute 與 method。

在 person 物件裡，有二個 attribute 與一個 method。例如，要呼叫 person 物件的 queryJob() method：

```
var person = {
        name: "Jollen",
        job: "Software Developer",

        queryJob: function() {
                alert(this.job);
        }
};

person.queryJob();
```

直接宣告物件是一種寫法，另外一種寫法是稱為 Instantiable Function，說明如下。

1.2 宣告 Class

同樣地，JavaScript 沒有類似 Class 這樣的語法，所以要宣告 Class 的話，以 function 關鍵字來實作即可，等價於函數宣告：

```
function Person(name, job) {
        this.name = name;
        this.job = job;
        this.queryJob = function() {
                alert(this.job);
        };
}
```

將 Function 關鍵字做為 Class 的宣告，自然就要討論是否能以 new 關鍵字將 Class 實例化成物件。在 JavaScript 裡，可以支援這樣的寫法。以下是一個實例化（Instantiate）的例子：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>MokoCrush</title>
</head>

<body>
<script>
function Person(name, job) {
        this.name = name;
        this.job = job;
        this.queryJob = function() {
                alert(name + "'s job is " + job);
        };
}

var person = new Person("Jollen", "Software Developer");

person.queryJob();
</script>
</body>
</html>
```

在這個例子裡，person 是 Person class 的實例化。所以，調用 person.queryJob() 方法時，所看到的畫面如下：

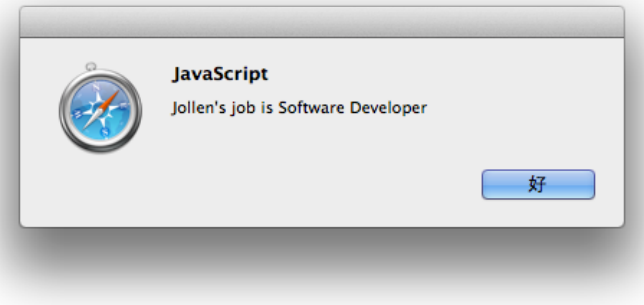

圖 **1.1 example-1-1.html** 執行結果

JavaScript 裡生成物件的做法：

- 使用 var 關鍵字宣告物件
- Instantiable Function

1.3 使用 Factory Pattern

Instantiable Function 可以利用 new 關鍵字生成它的物件，我們可以進一步將生成物件的過程封裝起來。這就是 Factory Pattern 的用途。

Factory Pattern 是很常用的一種物件生成設計模式，在軟體工程的領域裡，它用來將建立物件的過程抽象化。例如，將上述的例子重構，改以 factory pattern 來實作：

```
function personFactory(name, job) {
        var o = new Object();

        o.name = name;
        o.job = job;
        o.queryJob = function() {
                alert(name + "'s job is " + job);
        };

        return o;
}

var person1 = personFactory("Jollen", "Software Develop\
er");
var person2 = personFactory("Paul", "Product Manager");

person2.queryJob();
```

範例中，person1 以及 person2 物件的生成被抽象化了，也就是被封裝了起來。在主程式裡，我們看不到 new 關鍵字的使用，利用了 personFactory() 函數將真正的物件生成封裝起來。不過，使用 factory pattern 的話，就沒有辦法判斷物件原本的 class type 了。例如：

```
alert(person1 instanceof Object); // true
```

在 personFactory() 裡生成的物件，實際上是 Object 的實例化。實作時，如果需要明確地生成 Person 類別的物件，還是要以先前的做法為主：

```
function Person(name, job) {
        this.name = name;
        this.job = job;
        this.queryJob = function() {
                alert(this.job);
        };
}

// 將 Person() 視為 constructor
var person = new Person("Jollen", "Software Developer");

alert(person instanceof Person); // true
```

在這個例子裡，物件 person 確實是 Person 類別的實例化。將 Person() 視為 constructor，而不是函數，觀念上就能達成上述的目的（判斷物件原本的 class type）。

以上的觀念，可以用一個觀念來總結：函數（functions）就是物件（objects）。這是 JavaScript 極為重要的觀念，實務上，大量被應用在 Callback function 的實作上。上述的例子，將 Person() 視為 constructor 來使用；這個觀念亦稱為 Constructor Pattern。

1.4 Design Pattern for Front-End

用白話文來解釋設計模式（Design Pattern）：撰寫代碼與解決特定問題的「固定作法」。由於 HTML5+JavaScript 的廣泛流行，有一些專門針對 HTML+JavaScript 的設計模式被提出，其中最重要的二個設計模式為：Module Pattern 與 jQuery Plugin Pattern。

要學好 HTML5+JavaScript 應用程式開發，上述二個設計模式不能不學。此外，大家要有一個認知，就是：只懂 JavaScript 語法是做不出好高級品的，意思是撰寫不出良好品質的程式碼。在 HTML5+JavaScript 的世界裡，設計模式才是重點。要

强調一點，並不是不需要學習 JavaScript 語法，而是只懂得語法並不足以應付未來的研發工作。

1.5 Module Pattern

如同此設計模式的名字所述，module pattern 的目的是把程式碼「模組化」；將 JavaScript 的程式碼模組化，有其固定的做法。要將程式碼模組化，前提是將程式碼 Closure，即封閉性。接下來，以一個連續的範例，來說明 Module Pattern。在繼續進行下去前，請務必熟讀並了解 JavaScript 物件的觀念。

使用 **Private/Public** 觀念

請不要再使用 Local variable 與 Global variable 的寫法了，這在 HTML5+JavaScript 的世界裡行不通。以下是一個標準的錯誤示範：

```
var count = 0;                    //GLOBAL

function incrementCounter() {
   count++;                       //GLOBAL
   return count;
}

function resetCounter() {
   var orig;                      //LOCAL

   orig = count;
   count = 0;
}
```

這個例子只是讓大家瞧瞧 Local variable（區域變數）與 Global variable（全域變數）的寫法。現在，我們更上一層樓，把 Local variable 重構為 Private attribute，並把 Global variable 重構為 Public attribute。如下：

```
var testModule = (function () {

        var counter = 0;                                        // Priva

        return {
                incrementCounter: function() {          // Global
                        return ++counter;
                },

                resetCounter: function() {                  // Global
                        counter = 0;
                }
        };

}) ();
```

原本只是利用 function 關鍵字來定義一個類別（Class），現在更進一步以 Module 的方式將類別「封閉」。結果是，原本的區域變數 counter 變成了 Private attribute；原本的二個函數，現在成為了 Global method。將原本的程式碼，製作成 JavaScript module，這就是 Module pattern 的基本精神。

Closure 的目的，在避免全域變數的污染。變數污染，指的是自己的全域變數，被外部的程式碼做修改。以上述例子來看，如果 counter 沒有在 Closure 裡的話，其它地方的 JavaScript 就可以任意修改其值：因為 counter 是全域變數。為了避免這個問題，將程式碼 Closure 起來：只有 Closure 裡的程式碼，能修改 counter 變數。

外部程式碼，無法修改「封閉」程式碼裡的變數。概念上，以 module 將 attribute 與 method 進行封裝，這個關念就是「closure」。簡單說，將封閉的程式碼，放進 testModule 變數：testModule 成為一個模組。模組可被使用。

JavaScript 沒有明顯的物件導向語法，所以上面的一切都是觀念問題，而不是語法問題。軟體工程領域，很多時候都是

在處理這樣的哲學思想；技術面只是整個軟體工程的一小部份。

所以，我們要把軟體開發當做一個創作過程，而不是寫程式（Coding）的過程。

Import Modules

jQuery 是很好用的程式庫，它也被製作成 module。不過，由於 jQuery 强的擴充性，讓 jQuery 擁有為數可觀的「plugins」。所以，jQuery 本身就是一個平臺（Platform），或是一個開發框架（Framework），再加上有如空氣般，jQuery 在 Web 相關領域真是無所不在，所以就有 jQuery pattern 的出現，後續將會介紹 jQuery pattern。

jQuery 在 JavaScript 領域已經自成一格。

jQuery 本身是一個 module，module 可以匯入（Import）使用。以下是一個範例：

```
var testModule = (function (jQ) {

        var counter = 0;

        // Private
        function showHTML() {
                jQ(".header").html("<h1>" + counter + "</h1>");
        };

        return {
                incrementCounter: function() {
                        return ++counter;
                },

                resetCounter: function() {
                        counter = 0;
```

```
                },

                setCount: function(val) {
                        counter = val;
                },

                showCount: function() {
                        showHTML();
                }
        };

}) ($);
```

1.6 jQuery Pattern

jQuery pattern 就是開發 jQuery 插件（Plugin）的方式，所以技術上倒也沒有什麼學問。不過，jQuery pattern 有很高深的哲學道理，意思是說，在軟體工程領域裡，它創造了一個獨特的觀念。這個觀念就是 jQuery 知名的 “$”（Dollar sign），也就是「Selector」。

以下的例子，就是 jQuery pattern：

```
$( “div#news” ).html( “<h2>News Today</h2>” );
```

從 jQuery 設計模式的角度思考，上述的寫法似乎不太好。從 jQuery 設計模式的角度思考，如果今天我們想要透過 WebSocket 與伺服器溝通，並且在一個 “div” 裡來顯示結果，應該怎麼設計呢？想法如下：

- 將 WebSocket 的功能寫成一個 function
- 將 JavaScript function 封裝成 module
- 在 jQuery 裡擴充新的函數，簡單說，就是製作一個 jQuery 插件（Plugin）

以下是一段程式碼樣板：

```
(function($) {
$.fn.createWebSocket = function () {
  if ("WebSocket" in window)
  {
     alert("WebSocket is supported by your Browser!");
     var ws = new WebSocket("ws://<you-ip-adderess>:888\
8/start");
     ws.onopen = function()
     {
     };
     ws.onmessage = function (evt)
     {
     };
     ws.onclose = function()
     {
     };
     ws.onerror = function()
     {
     };
  }
  else
  {
     alert("WebSocket NOT supported by your Browser!");
  }
};

})($);
```

上述的寫法，採用暱名模組來實作。接者，再將程式碼儲存為 jquery.websocket.js。使用方法如下：

```
<!DOCTYPE html>
<head>
<script type='text/javascript' src="./jquery.min.js">
<script type='text/javascript' src="./jquery.websocket.\
js">
</head>
<body>
<div id="message"></div>

<script type="text/javascript">
$("#message").createWebSocket();
</script>
</body>
</html>
```

這是採用 jQuery pattern 的寫法。這種做法可以良好地組織 HTML5 與 JavaScript 程式碼。此外，JavaScript 的 module 具備「Closure」的特性，即封閉性，可以避免一些衍生問題。

由於 HTML5+JavaScript 的設計思想，和 Natvie App 的作法有很大的不同，所以了解 HTML5+Javascript 的應用程式「如何設計」，會是重要的一門課。了解設計模式，除了能有效組織 HTML5+JavaScript 程式碼外，也能做出正確的設計。

1.7 選擇器模式

我們在實作 Web Socket 連線生成時，利用了 jQuery pattern，這是一種選擇器模式。為什麼要使用選擇器模式，除了程式碼的組織較好外，另一個原因就是效能。事實上，讓程式碼組織更良好是次要的理由，真正的、主要的、最重要的原因是：使用選擇器方式可以讓 JavaScript 程式碼效能更好。

根據不同瀏覽器的實作，選擇器模式可以達到超過十倍以上的效能。再回顧一次我們的寫去：

```
<div id="message"></div>
<script type="text/javascript">
$("#message").createWebSocket();
</script>
```

總計利用了三個模式:

- 以 Closure 模式將類別封閉，這與 static class 有關係，在這裡先不做討論
- 使用選擇器模式，範例採用目前最流行的 jQuery selector “$”
- Read/Write Div Pattern

選擇器模式的效率取決於瀏覽器本身的實作，不過，以選擇器模式來代替直接存取 DOM，一般相信是最好的做法。因此，現代的 JavaScript 程式庫，幾乎都利用選擇器模式來實作（jQuery 一直都是最佳例子）；當我們實作自己的 JavaScript 程式庫時，也該善用選擇器模式。

典型的選擇器模式，是直接呼叫 DOM 的 API:

```
document.querySelector( “#header” );
```

不過，使用 jQuery 的選擇器「$」是目前的主流做法。

1.8 jQuery Pattern 實作 (jQuery 插件開發)

簡單來說，jQuery pattern 就是撰寫 jQuery Plugins。要開發 jQuery 的插件，是相當輕鬆愉快的工作，這都歸功於 jQuery 的優良架構。

Step 1：加入新增函數

在 $.fn 物件裡，加入新的函數屬性。範例:

```
$.fn.hello = function() {
        // your code here
};
```

Step 2：將程式碼 Closure

```
(function($) {
        $.fn.hello = function() {
                // your code here
        };
}) ($);
```

為什麼要將程式碼 Closure（封閉）的原因，前文已做過說明。

Step 3：儲存為獨立檔案使用

將上述程式碼，儲存為獨立檔案，例如：jquery.foo.js，並在 HTML5 裡使用。例如：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>

    <script type='text/javascript' src="./jquery.min.js\
"></script>
    <script type='text/javascript' src="./jquery.foo.js\
"></script>
</head>

<body>
        <div class="content">
        </div>
```

```
<script>
$(".content").hello();
</script>
</body>
</html>
```

上述的做法，是將自已的實作，設計成 jQuery Plugin 的形式。制作 jQuery 插件是非常簡單的，只要以上三個步驟即可完成。

1.9 其它模式

本章整理了 JavaScript 實務上的入門觀念，當然還有許多主題等著我們去探索。例如：MVC。MVC（Model-View-Controller）是一個年代久遠的設計模式。在軟體工程領域，也是一個使用率非常高的模式。在開發 HTML5+JavaScript 應用程式時，可以使用 MVC 模式將程式碼組織得更完善。做法如下：

- model/ 目錄
- controller/ 目錄
- view/ 目錄
- media/ 目錄

上面的做法，其實就是 Diango 組織一個專案的方式。Diango 是一個知名的 Web 開發框架。關於其它更進階的主題，將在後文繼續介紹。