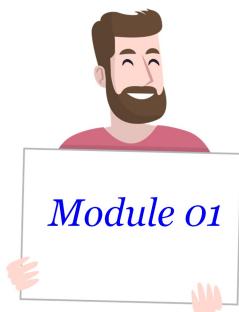




什麼是Node.js



designed by freepik

Estimated time:

50 min.

III 資訊工業策進會 Institute for Information Industry

【Key Points】：

學習目標

- **1-1: V8 JavaScript引擎**
- **1-2: JavaScript執行環境**
- **1-3: NodeJS還包含了什麼**



1-1

 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

一般程式語言可分成兩種，編譯型與解釋型。

編譯型語言與解釋型語言在執行過程中的最大差異是編譯型語言只需要編譯一次就可以執行。

但解釋型語言是每次執行前都要解釋成機器碼才能執行。

本模組將依序學習：

V8 JavaScript引擎

JavaScript執行環境

NodeJS還包含了什麼

最後會有「自我評量」與「課後練習步驟(Lab)」，請各位依照說明進行練習

【Key Points】：

Node.js是許多新創公司選擇的開發語言，跟它的開發生態有關。

因為JavaScript在網頁開發上幾乎很難短時間被取代。

作為一個後端開發它又有豐富的套件生態，甚至前端套件也可以運用在後端，所以只要了解套件使用，就可以快速開發。

1-1: V8 JavaScript引擎

- 如何Compile V8引擎？
- V8引擎如何執行JavaScript程式碼？
- V8引擎與Node.js的關係？
- 另一款JavaScript引擎-Chakra



designed by freepik



designed by freepik

1-2



JavaScript,V8,Node.js間的三角關係究竟為何？

JavaScript僅僅是語言，

V8引擎的角色類似橋梁，

Node.js則是封裝好許多原生模組。

Node.js透過V8讓JavaScript可以跟系統底層互動，這樣就可以做後端開發。

當然不只V8引擎是JavaScript引擎，還有微軟陣營推出的Chakra。

【Key Points】：

V8的誕生也跟網頁大量使用JavaScript有關
Google不得不推出JavaScript的優化解決方案
所以就自主開發出V8引擎

Node.js特性

- Node.js 是一個架構在Chrome V8引擎上的JavaScript執行環境。
- Node.js 使用事件驅動、非阻塞 I/O 模型，輕量且效能卓著。
- Node.js 能讓開發人員以 JavaScript 打造網頁伺服器程式。



1-3



2008年，Google 推出 Chrome 瀏覽器，並且使用V8引擎來解析 JavaScript。V8引擎滿足了 Ryan Dahl 對高性能Web伺服器的要求。

2009 年，Ryan Dahl 把V8引擎移植到伺服器端，並且發表了 Node.js。

在 Node.js 出現之前，JavaScript 通常只用於用戶端瀏覽器內的網頁程式設計。

Node.js 出現之後，JavaScript 可寫作伺服器端程式。剛開始，Node.js 只能在 Linux 系統執行。

在 v0.6.0 版本之後，Node.js 可跨平台在 Windows、Linux 及 Mac OS X 系統運行。

Node.js 具有下列特性：

- ✓ 得益於V8引擎，效能卓著。
- ✓ Node.js 採用異步(非阻塞) I/O，不會有因為同步(阻塞) I/O 導致效能急遽下降的問題。
- ✓ JavaScript 固有的事件驅動、閉包等語言特性，使用起來非常方便。

【Key Points】：

- ✓ Node.js 採用異步(非阻塞) I/O，不會有因為同步(阻塞) I/O 導致效能急遽下降的問題。
- ✓ JavaScript 固有的事件驅動、閉包等語言特性，使用起來非常方便。
- ✓ Node.js 可跨平台在 Windows、Linux 及 Mac OS X 系統運行

如何Compile V8引擎？

- **GYP(Generate Your Projects)**
- **GN(Generate Ninja)**

1. V8這類大型軟體架構就會使用**compile system** 工具，早期V8使用GYP，但現已轉使用GN，而Node.js還是繼續使用GYP。
2. GYP是用Python所寫，GN是用C++所寫，GN compile效率較快。
3. GYP與GN都可以跨平台compile，這也奠定了Node.js跨平台的能力。

1-4



GYP = Generate Your Projects，是一個構建自動化工具。

其實仔細看Node.js安裝一些加解密的套件的過程，會看到GYP compile 的過程。

因為Node.js底層是GYP 做整個專案編譯，所以，跟Node.js核心有關的套件幾乎都可以看見GYP的影子。

可以看看這些套件有沒有使用到**node-gyp**，就可以窺知這些套件是否會跟Node.js的核心作互動。

GYP與GN都可以跨平台compile，這也奠定了Node.js跨平台的能力。

【Key Points】：

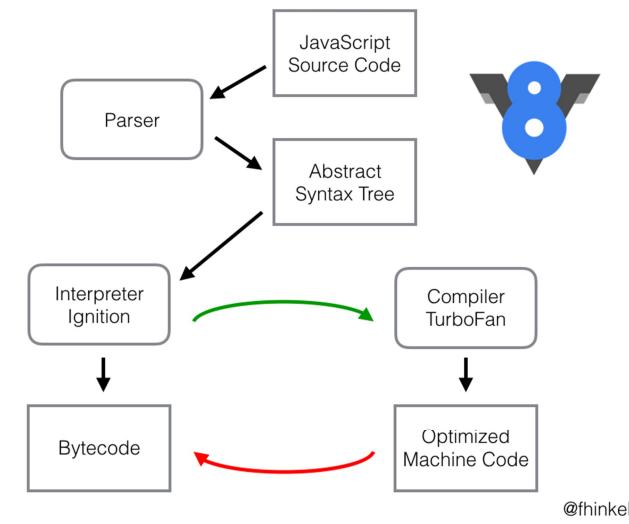
GYP = Generate Your Projects

GYP與GN都可以跨平台compile，這也奠定了Node.js跨平台的能力。

node-gyp是為了能夠compile成Node.js原生套件，所以fork GYP開源專案開發出的一套跨平台的compile工具。

V8引擎如何執行JavaScript程式碼？

- **JavaScript是高階語言，電腦無法直接讀取執行。**
- **V8採用JIT編譯將JavaScript即時編譯成機器碼。**



圖片來源:

<https://medium.com/@duartekevin91/basics-of-understanding-chromes-v8-engine-c5c8ec61fa6b>

@fhinkel

1-5

 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

程式碼讓電腦讀取執行的過程:

1. 程式碼經過Parser，生成AST(抽象語法樹)。
2. Ignition解釋器會解析AST(抽象語法樹)並生成Bytecode，並收集函式資訊。
3. 如果函式被呼叫多次，就會透過TurboFan JIT Compiler把Bytecode編譯成優化過的機器碼。
4. 機器碼交由電腦執行。
5. 如果Ignition收集的函式資訊與TurboFan生成的優化機器碼，無法在參數型別匹配或其他資訊有誤的話，就會進行反優化Deoptimization，恢復函式狀態。

為何Google會選擇開發解釋型的V8引擎讓Chrome使用？

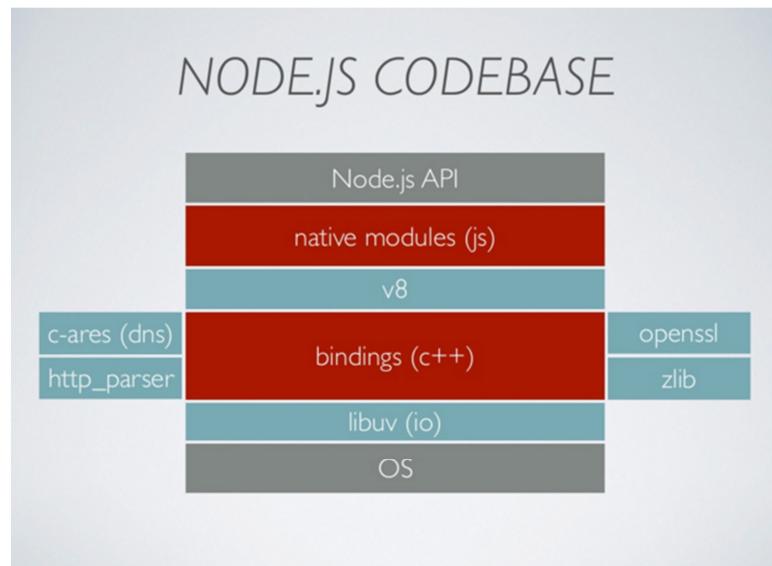
因為JavaScript是動態語言，不會先做型別檢查，如此就需要一個效率更高的編譯方式，網頁體驗追求的是那一兩秒內處理完成，使用者受不了程式編譯時間過長，所以Google就開發V8引擎。

【Key Points】：

1. AST(抽象語法樹)
2. Ignition解釋器會解析AST(抽象語法樹)並生成Bytecode
3. Bytecode編譯成優化過的機器碼

V8引擎與Node.js的關係？

- C++ 語言所寫，高效且可提供JavaScript與電腦底層互動的介面。
- 可做為C++與JavaScript的橋接層，提供對應的類別介面。



圖片來源: <http://blog.gitdns.org/2016/02/04/js-v8/>

1-6



V8是JavaScript運行的引擎

透過V8引擎，C++程式也是可以呼叫js。

V8引擎就是一個橋樑，讓js可以與C++互動。

透過FunctionTemplate與ObjectTemplate擴充套件就能呼叫C++的函式和類別。

Node.js選擇了V8引擎就能與作業系統底層或網路應用的系統底層互動，只要封裝成js的native modules提供上層API，Node.js也就可以作為一個伺服器的語言了。

【Key Points】：

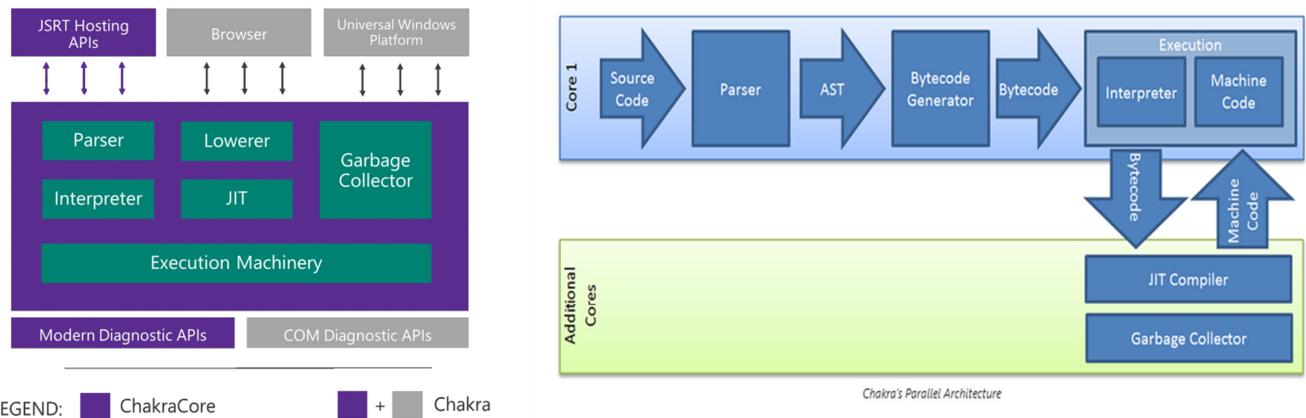
V8是JavaScript運行的引擎

C++可以呼叫js嗎？透過V8引擎，C++程式也是可以呼叫js，V8引擎就是一個橋樑，讓js可以與C++互動。

Node.js可以作為一個伺服器的語言。

另一款JavaScript引擎-Chakra

- Chakra引擎由Microsoft開發，應用在Edge瀏覽器。
- Chakra引擎使用解釋器搭配JIT編譯器產生機器碼。



圖片來源:

<https://www.eyerys.com/articles/news/microsoft-open-sources-chakracore-windows-10s-edge-web-browser-javascript-engine>

1-7

Chakra引擎是Microsoft開發，應用在Edge瀏覽器

ChakraCore可以平行進行JIT編譯又同時進行優化程式碼，

同時又可進行記憶體垃圾回收(GC)

綜合上述兩點，使得整個解釋編譯過程的效率極高。

另外，ChakraCore也會盡量延遲解釋編譯那些尚未使用到的function，使整個解釋編譯過程節省更多資源。

為何Edge瀏覽器跟Chrome瀏覽器的記憶體消耗量差異這麼大？

因為Edge採平行編譯且改進GC，可以直接開Edge與Chrome看看工作管理員的記憶體部分。

Chakra引擎是Microsoft開發，應用在Edge瀏覽器

編譯的效率很高

會盡量延遲解釋編譯那些尚未使用到的function

1-2: JavaScript執行環境

- 什麼是全域執行環境(Global Execution Context)?
- 什麼是呼叫堆疊(Call Stack)?
- 什麼是事件迴圈(Event Loop)?



designed by freepik



designed by freepik

1-8

 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

作用域分全域作用域與區域作用域：

- 全域作用域在程式執行一開始就會建立
- 區域作用域則是指函式內部的範圍

一般來說全域作用的變數，區域作用域裡頭都能引用。

作用域跟變數的生命週期有關，在後面會介紹到的閉包可以充分感受到作用域的影響。

此外，本知識點還會談到：

呼叫堆疊(Call Stack)

事件迴圈(Event Loop)

【Key Points】：

全域作用域與區域作用域

呼叫堆疊(Call Stack)

事件迴圈(Event Loop)

課堂示範說明

- 先至<https://code.visualstudio.com/> 下載VS Code並安裝。
- 新增try_js資料夾。
- 新增context.html檔案。
- 在context.html 貼入下一張投影片的程式碼。

1-9



先至 <https://code.visualstudio.com/> 下載VS Code
安裝 VS Code
新增 try_js 資 料夾
在 try_js 資料夾新增 context.html 檔案
在context.html 貼入下一張投影片的程式碼

【Key Points】：

安裝 VS Code
在 try_js 資料夾新增 context.html 檔案
在context.html 貼入下一張投影片的程式碼

什麼是全域執行環境？

```
<script>↵
    console.log(this) // window↵
</script>↵
```

```
<script>↵
    console.log(car) // undefined↵
    var car='Benz'↵
</script>↵
```

```
<script>↵
    window.name='Benz'↵
    function callMe(){↵
        console.log(this.name) // Benz↵
    }↵
    callMe()↵
</script>↵
```

```
<script>↵
    window.name='Benz'↵
    var car={↵
        name:'Lexus',↵
        callMe:function(){↵
            console.log(this.name) // Lexus↵
        }↵
    }↵
    car.callMe()↵
</script>↵
```

1-10



1. 只有一個global context。
2. 變數hoisting，JavaScript都會把變數挪到最上面做宣告，所以才沒出現錯誤，而console.log會顯示undefined，意思是尚未給予任何值。
3. 執行順序由上而下逐行執行。
4. 呼叫一個function會立即建立起執行環境。
5. this指向跟由哪個物件呼叫function有關。

在函式前添加「use strict」，就可以避免this指向全域物件，但就必須先宣告好this物件為何。

【Key Points】：

只有一個global context

呼叫一個function會立即建立起執行環境

this指向跟由哪個物件呼叫function有關

什麼是呼叫堆疊？

```
1  function foo(b) {  
2    var a = 5;  
3    return a * b + 10;  
4  }  
5  
6  function bar(x) {  
7    var y = 3;  
8    return foo(x * y);  
9  }  
10 console.log(bar(6));
```

Output:-

Call Stack

console.log(bar(6))

main()

1-11

III 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

當執行到某個函式時...

JS引擎就會將該函式丟到堆疊，

該函式執行到return時，將該函式從堆疊中移除。

堆疊的運作是「後進先出」。

「後進先出」這種資料結構的運作方式是：最後放進去的，最先取出來。

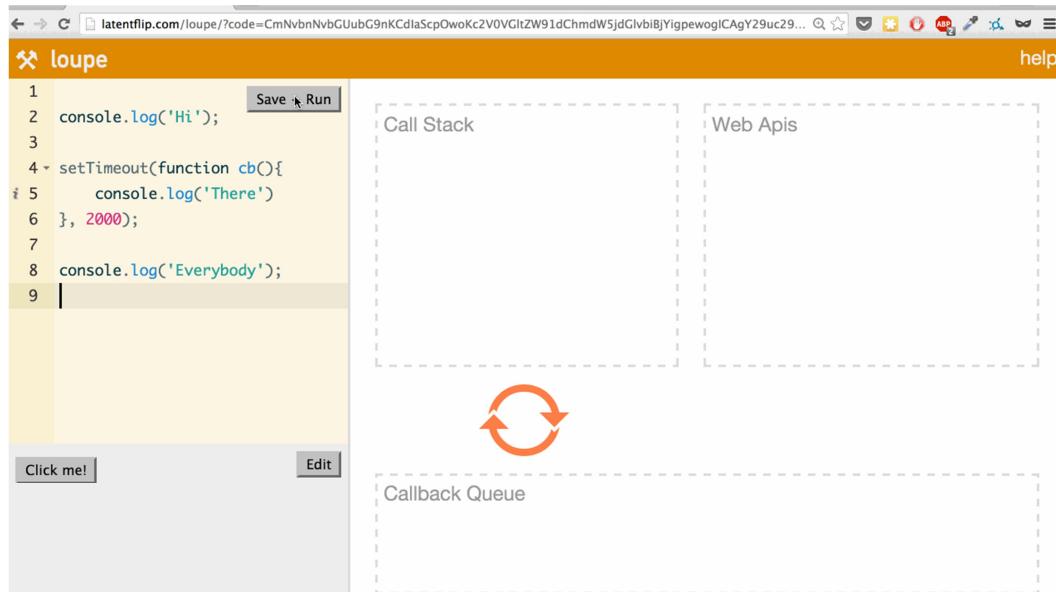
【Key Points】：

函式呼叫

堆疊

堆疊的執行順序是後進先出。

什麼是事件迴圈？



1-12



程式執行時，依序將函式放入呼叫堆疊(Call Stack)。

如呼叫到Web APIs相關函式則將其回呼函式放入回呼佇列(Callback Queue)等待叫用。

回呼時機來臨時，才將回呼函式放入呼叫堆疊(Call Stack)執行。

`setTimeout` 是JS引擎提供的函式，可預約若干時間後，執行特定函式。

`setTimeout` 非常重要，JavaScript在 ES6 提供的 Promise 的底層就是有使用到 `setTimeout`。

【Key Points】：

程式執行時，依序將函式放入呼叫堆疊(Call Stack)

回呼函式放入回呼佇列(Callback Queue)等待叫用

`setTimeout` 非常重要

1-3: NodeJS還包含了什麼？

- 垃圾回收機制(GC)
- 閉包(Closure)
- 非同步非堵塞IO
- Promise Async / Await



designed by freepik



designed by freepik

1-13

 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

這邊主要是在補充更多Node.js的重要機制與特性。包括：

- 垃圾回收機制(GC)
- 閉包(Closure)
- 非同步非堵塞IO
- Promise Async/Await

你將了解到 Node.js 如何靠著一條執行緒完成多項任務而不需要等待太多時間。最主要的是以非同步 callback 作法搭配 event loop，讓事件不斷滾動，創造出一種可以負荷多事件但一條執行緒的情形，以滿足後端伺服器的請求。

Node.js 如果遇到IO密集的任務，將會使Node.js耗費大量記憶體，這點在開發跟規劃伺服器要多留意。

【Key Points】：

- 垃圾回收機制(GC)
- 閉包(Closure)
- 非同步非堵塞IO、Promise Async/Await

垃圾回收機制(GC)

- 在呼叫堆疊(Call Stack)執行過的函式會銷毀
- V8引擎會不斷確認函式狀態
- V8引擎會遍歷Heap中的所有值，其對應的記憶體地址如查無引用，就會回收記憶體。
- V8引擎會不斷確認值的引用是否還存在

1-14



這邊有承先啟後的作用，先前介紹過Call Stack，這邊補充執行完的 function 會被怎樣處理？

V8會不斷確認函式狀態，也會不斷確認值的引用是否還存在。

Node.js跟C語言不太一樣，因為這邊可以看到垃圾回收機制都是Node.js處理掉，我們開發者無須特別動手。

而C語言就需要自己手動回收，相當麻煩。

雖然Node.js會主動做垃圾回收，但開發者仍須注意記憶體洩漏的問題，因為很可能記憶體使用量的增長速度遠超過記憶體回收的速度，例如無窮迴圈或是閉包的使用失當，另外能不宣告全域變數就不要宣告全域變數。

【Key Points】：

Node.js會主動做垃圾回收

開發者仍須注意記憶體洩漏

例如無窮迴圈或是閉包的使用失當

閉包(Closure)

- 在一個function裡頭return另一個function，就形成了一個閉包。
- 傳出來的函式，可存取到出生地的函式變數。
- 閉包的作用是延長內部函式變數的生命週期，小心可能造成記憶體洩漏。

```
<script>
    function getCount(i) {
        var count = i;
        var calculate = function() {
            console.log(count);
            count++;
        };
        return calculate;
    }
    var displayCount = getCount(0);
    displayCount(); // 0
    displayCount(); // 1
    displayCount(); // 2
</script>
```

1-15



上述程式的執行結果會看到 0,1,2 被 console.log() 印出來。

理論上，每一行執行完，Node.js就會做記憶體回收。

但 dispalyCount() 裡的變數卻沒有被回收（看到0, 1, 2愈來愈大，而非每次都是零）。

因為變數count還有被另一個內部函式引用，而且整個getCount function 最後是回傳內部函式calculate function。

所以變數count的生命週期會在每次getCount被呼叫而延長，不會被回收。

【Key Points】：

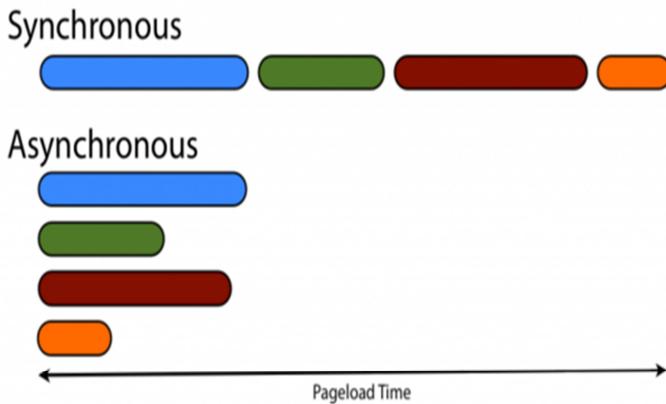
上述程式的執行結果會看到 0,1,2 被 console.log() 印出來

因為變數count還有被另一個內部函式引用

所以變數count的生命週期會在每次getCount被呼叫而延長，不會被回收

非同步非堵塞IO

- Node.js 借助 Callback function 來達到非同步非阻塞IO。



```
<script>
  var readFile=function(cb){←
    setTimeout(function () {←
      cb('file content:hello async function')←
    }, 5000)←
  }←
  console.log('dosomethingAAA')←
  readFile(function(file){←
    console.log(file)←
  })←
  console.log('dosomethingBBB')←
</script>
```

圖片來源: <https://itw01.com/5KQ5EXJ.html>

1-16



依序印出

dosomethingAAA

dosomethingBBB

file content:hello async function

解釋:

setTimeout function 本身會 delay 執行，

在這邊是模擬執行一個需要一段時間才會回傳的任務，

Node.js 相當多 API 都有提供 callback function 以避免阻塞。

你可以將 Synchronous (同步) 理解成一步接一步; Asynchronous (非同步) 理解成可多線同時進行。

非同步處理是 Node.js 一大特色，但 callback 寫法很容易造成 callback hell。

【Key Points】：

將 Synchronous (同步) 理解成一步接一步; Asynchronous (非同步)

Node.js 相當多 API 都有提供 callback function 以避免阻塞

非同步處理是 Node.js 一大特色

Promise Async/Await

- ES6 Promise
- ES6 Async / Await

```
<script>
  var threeSecLater = function () {
    setTimeout(function () {
      console.log('3 sec later')
    }, 3000)
  }

  var fourSecLater = function () {
    setTimeout(function () {
      console.log('4 sec later')
    }, 4000)
  }

  threeSecLater()
  fourSecLater()
</script>
```

```
<script>
  var threeSecLater = function () {
    return new Promise(function(resolve, reject) {
      setTimeout(function () {
        console.log('3 sec later')
        resolve()
      }, 3000)
    })
  }

  var fourSecLater = function () {
    return new Promise(function(resolve, reject) {
      setTimeout(function () {
        console.log('4 sec later')
        resolve()
      }, 4000)
    })
  }

  ;(async()=>{
    await threeSecLater()
    await fourSecLater()
  })()
</script>
```

1-17



如果想讓非同步function同步執行，可以使用Promise。

讓函式 return Promise 物件，再藉由 `async/await` 的語法，讓非同步 function 變成同步執行。

Promise 是ES6版本提出來的。

Node.js 第6版開始都原生支持。

並且每次Node.js升級都有針對Promise做效能提升，可見Node.js多重視Promise。

【Key Points】：

讓函式 return Promise 物件

藉由 `async/await` 的語法，讓非同步 function 變成同步執行

Node.js 第6版開始都原生支持

Summary 〈 精華回顧 〉

- 了解JavaScript與V8的關係
- 了解Node.js的基本架構與其運作原理
- 了解JavaScript的作用域與hoisting,closure等
- 了解Call Stack,Event Loop等
- 了解同步非同步的差異，
如何讓非同步函式同步執行。



1-18

 財團法人資訊工業策進會
INSTITUTE FOR INFORMATION INDUSTRY

本模組很深入地說明了許多 Node.js 的特性與內部運作，包括

- JavaScript與V8的關係
- Node.js的基本架構與其運作原理
- JavaScript的作用域與hoisting,closure等
- Call Stack,Event Loop等
- 同步非同步的差異，甚至怎讓非同步函式同步執行

【Key Points】：

JavaScript與V8的關係

Node.js的基本架構與其運作原理

同步非同步的差異

線上程式題

- **題目一：使用callback function**

使用callback function對以下的logDay function進行改寫，console.log依序打印出Monday,Tuesday

- **題目二：使用setTimeout 函式**

使用setTimeout function間隔1秒，console.log依序輸出Monday,Tuesday

- **題目三：使用Promise then語法**

使用Promise then語法對問題1的logDay function再加以改寫，console.log依序輸出Monday,Tuesday

1-19



題目一：使用callback function

使用callback function對以下的logDay function進行改寫，console.log依序打印出Monday,Tuesday

題目二：使用setTimeout 函式

使用setTimeout function間隔1秒，console.log依序輸出Monday,Tuesday

題目三：使用Promise then語法

使用Promise then語法對問題1的logDay function再加以改寫，console.log依序輸出Monday,Tuesday

閱讀題目說明與現有程式

在作答區修改程式

【Key Points】：

callback function

setTimeout

Promise

課後練習題(Lab)

- 練習題的程式都未完全實作完，請學員按照題目要求接續完成。
- Lab01:理解this**
請在不刪除現有程式碼的條件下，新增程式碼，使其能夠console.log出body的屬性handsCount的值
- Lab02:理解Callback**
請在不刪除現有程式碼的條件下，添加程式碼並使用callback function，讓程式可以依序console.log出first line,second line,third line,last line。
- Lab03:理解Promise**
請使用Promise改寫getData function，其餘請勿更改，執行程式後預期等待三秒才會console.log出This is data

Estimated time:

20 min.

1-20



情節描述：

這次練習會更了解物件的操作、Callback與Promise的使用，以下練習題都未完全實作完，請同學接續完成並滿足題目要求。

Lab01:理解this

請在不刪除現有程式碼的條件下，新增程式碼，使其能夠console.log出body的屬性handsCount的值

Lab02:理解Callback

請在不刪除現有程式碼的條件下，添加程式碼並使用callback function，讓程式可以依序console.log出first line,second line,third line,last line。

Lab03:理解Promise

請使用Promise改寫getData function，其餘請勿更改，執行程式後預期等待三秒才會console.log出This is data

【Key Points】：

This

Callback function

Promise

範例程式使用說明

- 範例程式資料夾: Module_01_example
- 使用步驟:
 1. 啟動檔案總管
 2. 瀏覽到 Module_01_example 資料夾
 3. 點兩下 lab01.html
 4. 以 Chrome 瀏覽器為例，按下 F12 或「Ctrl + Shift + I」，開啟「開發人員工具」
 5. 切換到 Console 頁籤，F5重新整理，觀看程式執行效果。
 6. 針對 lab02.html 與 lab03.html，重複相同的動作。

1-21



使用步驟：

1. 啟動檔案總管
2. 瀏覽到 Module_01_example 資料夾
3. 點兩下 lab01.html
4. 以 Chrome 瀏覽器為例，按下 F12 或「Ctrl + Shift + I」，開啟「開發人員工具」
5. 切換到 Console 頁籤，F5重新整理，觀看程式執行效果。
6. 針對 lab02.html 與 lab03.html，重複相同的動作。

【Key Points】：

瀏覽到 Module_01_example 資料夾

以 Chrome 瀏覽器開啟 lab01.html

在「開發人員工具」的 Console 頁籤，觀看程式執行效果