



# 讀寫檔案



designed by © freepik

Estimated time:

50 min.

 資訊工業策進會 Institute for Information Industry

【Key Points】：

# 學習目標

- **8-1: 讀取文字檔**
- **8-2: 寫入JSON檔**
- **8-3: http request 檔頭寫入檔案**



8-1

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

我們即將一起學習：

- fs模組簡介
- 同步 fs.readFileSync()
- 非同步 fs.readFile()
- 如何使用Stream
- 什麼是JSON
- Object與JSON
- 以 fs.writeFileSync() 寫入 JSON
- 非同步 fs.writeFile()
- Request Header
- Response Header
- 使用express製作簡單的伺服器
- 將 Request Header寫入檔案

【Key Points】：

- 8-1: 讀取文字檔
- 8-2: 寫入JSON檔
- 8-3: http request 檔頭寫入檔案

## 8-1: 讀取文字檔

- **fs**模組簡介
- 同步 **fs.readFileSync( )**
- 非同步 **fs.readFile( )**
- 如何使用**Stream**



designed by freepik



designed by freepik

8-2

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

- Node.js有提供原生API-fs，可以讀寫檔案。
- **fs.readFileSync( )** 會使整體程式阻塞。
- **fs.readFile( )** 透過callback避免程式阻塞。
- Stream分成**Readable**、**Writable**、**Duplex**、**Transform**。
- Stream適合大型檔案讀寫。

### 【Key Points】：

Node.js有提供原生API-fs，可以讀寫檔案  
**fs.readFile( )** 透過callback避免程式阻塞  
Stream適合大型檔案讀寫

## fs 模組簡介

- Node.js File System API
- 所有操作都有同步與非同步的API
- 非同步操作是使用Callback將結果回傳
- 同步操作的exception可以用try catch捕捉

8-3



- const fs = require('fs'); 相當於引用了Node.js File System module。
- 檔案較大，操作需要耗時較久，建議用非同步操作。
- Callback被大量使用在非同步API。
- 同步操作如fs.readFileSync( )，Sync就意味著同步。
- 非同步API的exception都是在Callback function的裡的第一個參數。

【Key Points】：

const fs = require('fs'); 相當於引用了Node.js File System module

所有操作都有同步與非同步的API

檔案較大，操作需要耗時較久，建議用非同步操作

# 同步 fs.readFileSync( )

- **readFileSync( )**第一個參數是檔案路徑，  
檔案路徑可以分為相對路徑與絕對路徑
- **readFileSync( )**第二個參數物件可以填入encoding值、flag值
- 同步API可以透過try catch捕捉exception

```
const fs=require('fs')

;(()=>{
  try {
    const data= fs.readFileSync('./learn_fs.txt',{encoding:'utf-8'});
    console.log(data)// hello world
  } catch (e) {
    console.error(e.message)
  }
})()
```

8-4



- **readFileSync( )**第一個參數是檔案路徑。
- 檔案路徑可以分為相對路徑與絕對路徑。
- **readFileSync( )**第二個參數物件可以填入encoding值、flag值。
- encoding為編碼方式，常見如utf-8，flag則能使用一些如操作檔案的相關指令。
- 同步API可以透過try catch捕捉exception。

## 【Key Points】：

readFileSync( )第一個參數是檔案路徑

readFileSync( )第二個參數物件可以填入encoding值，常見如utf-8

同步API可以透過try catch捕捉exception

# 非同步 fs.readFile( )

- 呼叫格式: fs.readFile("檔名", 選項, callback函式)
- Callback function參數第一個是exception，第二個是資料

```
const fs=require('fs')

;(()=>{
  fs.readFile('./learn_fs.txt',{encoding:'utf-8'},(e,data)=>{
    if(e){
      console.error(e.message)
    }
    console.log(data)// hello world
  })
})()
```

8-5



- 呼叫格式: fs.readFile("檔名", 選項, callback函式)
- Callback function參數第一個是exception，第二個是資料。
- 非同步讀取檔案不會讓程式阻塞。
- 與同步讀取檔案不同的地方是透過Callback傳遞資料。
- 錯誤優先處理是Node.js核心觀念。
- 稍大的檔案適合用非同步處理，但大型檔案建議用stream方式。

## 【Key Points】：

呼叫格式: fs.readFile("檔名", 選項, callback函式)

Callback function參數第一個是exception，第二個是資料。

非同步讀取檔案不會讓程式阻塞

# 如何使用 Stream

- **stream** 以監聽事件的方式運作
- **open**事件：做些開啟檔案後的需要做的事
- **close**事件：關閉檔案後需要做的事
- **data** 事件：傳遞資料

```
const fs=require('fs')
;(()=>{
  fs.createReadStream('./learn_fs.txt',{encoding:'utf-8'})
  .on('open',()=>{
    console.log('open file')
  })
  .on('data',(data)=>{
    console.log(data)
  })
  .on('close',()=>{
    console.log('close')
  })
})()
```

8-6



- Stream 以監聽事件的方式運作。
- open事件的callback function可以做些開啟檔案後需要做的事。
- data事件的callback function會傳遞資料。
- close事件的callback function可以做些關閉檔案後需要做的事。
- stream還有很多監聽事件與API，使用者可以有更大彈性，也減輕伺服器的負荷。

## 【Key Points】：

open事件：做些開啟檔案後的需要做的事

close事件：關閉檔案後需要做的事

data 事件：傳遞資料

## 8-2: 寫入JSON檔

- 什麼是JSON
- Object與JSON
- 以 `fs.writeFileSync()` 寫入 JSON
- 非同步 `fs.writeFile()`



designed by freepik



designed by freepik

8-7

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

- JSON 的官方 [MIME 類型](#)是 application/json，副檔名是 .json，數據結構是key value結構。
- JSON基本數據格式有number、boolean、""、null、array、object。
- JavaScript可以透過JSON.parse()將字串轉換成JSON物件。
- 透過JSON.stringify()將JSON物件轉換成字串。
- `fs.writeFileSync()`會使整體程式阻塞，建議使用 `fs.writeFile()`。

### 【Key Points】：

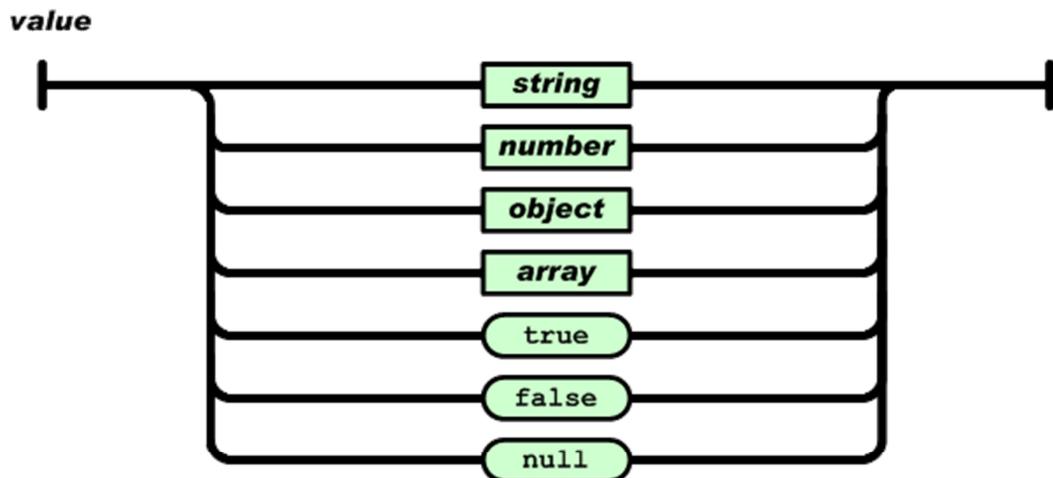
透過JSON.parse()將字串轉換成JSON物件。

透過JSON.stringify()將JSON物件轉換成字串

`fs.writeFileSync()`會使整體程式阻塞，建議使用 `fs.writeFile()`

## 什麼是JSON

- MIME類型
- 數據結構
- 使用雙引號
- 沒有函式



1-8

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

- MIME是一種標準，常見如:text/plain、text/html、image/jpeg、application/json、application/javascript等。
- JSON取代老舊的XML格式。
- 數據結構支援: number、boolean、null、string、array、object。
- JSON 內部結構的key value都必須使用雙引號。
- JSON中沒有函式。

【Key Points】：

JSON取代老舊的XML格式

數據結構支援: number、boolean、null、string、array、object

JSON 內部結構的key value都必須使用雙引號

# Object與JSON

- **JavaScript 的 Object**

- **JSON Object**

- **JSON.parse( )**

可以將一段格式正確的plain text，轉換成JSON Object

- **JSON.stringify( )**

可以將合法物件轉換成 JSON 格式的字串

8-9



- 在JavaScript世界裡，函式也是物件，物件可以有屬性與方法，也可以繼承某物件。
- JSON Object沒有方法定義，只能針對屬性作定義，也不能繼承自某物件。
- JSON.parse( )可以將一段格式正確的plain text，轉換成JSON Object。
- JSON.stringify( )可以將合法物件轉換成JSON 格式的字串。
- JSON 常被當作前端與後端相互溝通傳遞的資料格式，明顯比XML輕量。

## 【Key Points】：

在JavaScript世界裡，函式也是物件

JSON Object沒有方法定義，只能針對屬性作定義，不能繼承自某物件

JSON 常被當作前端與後端相互溝通傳遞的資料格式

# 以 `fs.writeFileSync()` 寫入 JSON

- `writeFileSync`第一個參數為路徑  
路徑非為相對路徑與絕對路徑。
- 第二個參數則是要寫入的資料
- 第三個參數可以填入encoding格式

```
const fs=require('fs')
;(()=>{
  const str='{"key":"value"}';
  try {
    fs.writeFileSync('./learn.json',str,{encoding:'utf-8'})
  } catch (e) {
    console.error(e.message)
  }
})()
```

8-10



- `writeFileSync`第一個參數為路徑。
- 路徑非為相對路徑與絕對路徑。
- 第二個參數則是要寫入的資料。
- 第三個參數可以填入encoding格式。
- 可以透過try catch捕捉exception。

## 【Key Points】：

`writeFileSync`第一個參數為路徑，路徑非為相對路徑與絕對路徑。

第二個參數則是要寫入的資料。

第三個參數可以填入encoding格式。

# 非同步 fs.writeFile( )

- **writeFile**第一個參數是檔案路徑
- 第二個參數是要寫入的資料，
- 第三個參數可以放encoding格式。
- 第四個參數是 **Callback function**

```
const fs=require('fs')
;(()=>{
  const str='{"key":"value"}';
  fs.writeFile('./learn.json',str,{encoding:'utf-8'},(e)=>{
    if(e){
      console.error(e.message)
    }
  })
})()
```

8-11



- **writeFile()** 第一個參數是檔案路徑，
- 第二個參數是要寫入的資料，
- 第三個參數可以放encoding格式。
- 第四個參數是 **Callback function**，**Callback function**的參數值若為 null 表示無錯誤，順利寫入資料到檔案。
- 錯誤優先處理是Node.js核心觀念。
- 非同步寫入檔案，會讓程式阻塞。
- 稍大的檔案適合用非同步處理，但大型檔案建議用**stream**方式。

## 【Key Points】：

非同步讀取檔案不會讓程式阻塞。

**writeFile**第一個參數是檔案路徑，第二個參數是要寫入的資料，第三個參數可以放encoding格式。第四個參數是 **Callback function**，**Callback function**的參數值若為 null 表示無錯誤，順利寫入資料到檔案

## 8-3: http request 檔頭寫入檔案

- Request Header
- Response Header
- 使用express製作簡單的伺服器
- 將 Request Header寫入檔案



designed by freepik



designed by freepik

8-12

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

- Accept、Accept-Charset、Accept-Encoding、Accept-Language常見被用來告知伺服器，瀏覽器可以接收哪種資訊。
- Authorization常被用來做client端與server端的身分驗證。
- Access-Control-Allow-Origin指定哪些網站可以共享資源，可以參考同源網域政策。
- Cache-Control可以設定cache時間長度(max-age)，也可以設定不用cache(no-cache)。
- 使用非同步寫入檔案避免網站請求阻塞。

【Key Points】：

Request Header

Response Header

Request Header寫入檔案

# Request Header

- 告知伺服器：瀏覽器可以接收哪種資訊：  
Accept、Accept-Charset、Accept-Encoding、Accept-Language
- client端與server端的身分驗證  
Authorization
- 設定cache時間長度(max-age)  
Cache-Control
- Cookie：瀏覽器端存放的小段資料
- Content-Length：此次通訊body區的資料長度
- User-Agent：由哪個載具(瀏覽器)發出請求

8-13



- Accept、Accept-Charset、Accept-Encoding、Accept-Language常見被用來告知伺服器：瀏覽器可以接收哪種資訊。
- Authorization常被用來做client端與server端的身分驗證。
- Cache-Control可以設定cache時間長度(max-age)，也可以設定不用cache(no-cache)。
- Cookie，蠻多網站都還使用Cookie，使用者登入後存放登入憑證於瀏覽器端，方便登入後的所有操作通過伺服器端的驗證。
- Content-Length會記錄此次通訊body區的資料長度
- User-Agent則可以看出是由哪個載具(瀏覽器)發出請求。

## 【Key Points】：

Accept系列的標頭，用來告知伺服器：瀏覽器可以接收哪種資訊

Cookie：瀏覽器端存放的小段資料

Cache-Control可以設定cache時間長度(max-age)

# Response Header

- Access-Control-Allow-Origin: 控管可否跨域取得資源
- Cache-Control: 告知瀏覽器cache相關設定
- 告訴瀏覽器此次回應內容的相關資訊:  
Content-Encoding、Content-Language、Content-Length、Content-Type，例如：  
Content-Type: application/json 表示這次傳送的是JSON資料
- Set-Cookie: 告訴瀏覽器cookie要設定哪些內容
- Status: 告訴瀏覽器此次回應的狀態碼

8-14



- Access-Control-Allow-Origin: 控管可否跨域取得資源。詳情請查詢: 同源網域政策(Same Origin Policy) 與 CORS。
- Cache-Control: 告知瀏覽器cache相關設定為何。
- Content-Encoding、Content-Language、Content-Length、Content-Type告訴瀏覽器此次回應內容的相關資訊。例如：  
Content-Type: application/json 表示這次傳送的是JSON資料
- Set-Cookie: 告訴瀏覽器cookie要設定哪些內容。
- Status告訴瀏覽器此次回應的狀態碼。

## 【Key Points】：

Status告訴瀏覽器此次回應的狀態碼

Set-Cookie: 告訴瀏覽器cookie要設定哪些內容

Access-Control-Allow-Origin: 控管可否跨域取得資源

# 使用express製作簡單的伺服器

- 引用 express 模組
- 監聽埠號 3000，接受用戶端連線。
- App.get("/header", ...) 設定一個/header 路由，請求方式為 get。
- 每次有用戶端連線「/header」，以 res.send() 傳出內容給用戶端
- 以 console.log() 出 req.header 所有的標頭內容

```
const express = require('express');
const app = express();
app.get('/header',(req,res,next)=>{
    console.log(req.headers)
    res.send('ok')
})
app.listen(3000)
```

8-15



- 引用 express 模組
- 監聽埠號 3000，接受用戶端連線。
- App.get("/header", ...) 設定一個/header 路由，請求方式為 get。
- 每次有用戶端連線「/header」，以 res.send() 傳出內容給用戶端
- 以 console.log() 出 req.header 所有的標頭內容，請嘗試用不同瀏覽器連連看，觀察 User-Agent 的結果 (IE 或 Edge 或 Chrome...)。

## 【Key Points】：

引用 express 模組App.get("/header", ...) 設定一個/header 路由，請求方式為 get  
每次有用戶端連線「/header」，以 res.send() 傳出內容給用戶端  
以 console.log() 出 req.header 所有的標頭內容

# 將 Request Header 寫入檔案

- 將 req.headers 轉成 JSON 字串

```
const dataStr = JSON.stringify( req.headers )
```

- 將 JSON 字串以 fs.writeFile()  
寫到檔案

```
fs.writeFile("...", dataStr, ...)
```

```
const express = require('express');
const app = express();
const fs=require('fs')
app.get('/header',(req,res,next)=>{
    console.log(req.headers)
    console.log(typeof req.headers)
    const dataStr=JSON.stringify(req.headers)
    fs.writeFile('./header.txt',dataStr,(e)=>{
        if(e){
            console.error(e)
        }
        console.log('finish write ',new Date().toISOString())
    })
    res.send('ok '+new Date().toISOString())
})
app.listen[3000]
```

8-16



財團法人資訊工業策進會

INSTITUTE FOR INFORMATION INDUSTRY

- console.log(typeof req.headers) 可以看出 headers 是物件。
- 因為 req.headers 是物件，所以可以使用 JSON.stringify() 將物件轉換成字串。
- 將 JSON 字串以 fs.writeFile() 寫到檔案
- 使用非同步寫入檔案避免網站請求阻塞。
- console.log('finish write') 並且帶上ISO時間
- res.send('ok')也帶上ISO時間。
- 比對兩個時間差，會發現：完成寫入檔案的時間較晚，伺服器回應較早，由此可看到非同步運作的證據。

## 【Key Points】：

將 req.headers 轉成 JSON 字串

將 JSON 字串以 fs.writeFile() 寫到檔案

比對兩個時間差，看到非同步運作的證據

# Summary 〈精華回顧〉

- 了解fs同步與非同步的用法。
- 採用同步的方式讀寫檔案
- 採用非同步方式讀寫檔案
- 了解JSON與Object的關係。
- 了解Request Header。
- 了解Response Header。
- 了解在網站請求回應過程中，要善用非同步處理。



8-17

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

本模組的重點:

- fs模組簡介
- 同步 fs.readFileSync()
- 非同步 fs.readFile()
- 如何使用Stream
- 什麼是JSON
- Object與JSON
- 以 fs.writeFileSync() 寫入 JSON
- 非同步 fs.writeFile()
- Request Header
- Response Header
- 使用express製作簡單的伺服器
- 將 Request Header寫入檔案

【Key Points】：

讀取文字檔

寫入JSON檔

http request 檔頭寫入檔案

# 線上程式題

- 8-1 物件轉換成JSON字串將以下物件轉換成JSON字串並console.log出來。

```
const shoppingCart={  
    Count:100,  
    perPrice:10  
}
```

- 8-2 使用同步方式寫資料到檔案

想要使用同步方式寫資料到檔案，程式怎麼寫？

```
fs.writeFileSync('./shopping_cart.json',fileContent);
```

- 8-3 使用非同步方式寫資料到檔案

想要使用非同步方式寫資料到檔案，程式怎麼寫？

8-18



## 8-1 物件轉換成JSON字串

將以下物件轉換成JSON字串並console.log出來。

```
const shoppingCart={  
    Count:100,  
    perPrice:10  
}
```

## 8-2 使用同步方式寫資料到檔案

想要使用同步方式寫資料到檔案，程式怎麼寫？

```
fs.writeFileSync('./shopping_cart.json',fileContent);
```

## 8-3 使用非同步方式寫資料到檔案

想要使用非同步方式寫資料到檔案，程式怎麼寫？

```
fs.writeFile('./shopping_cart.json',fileContent,{encoding:'utf-8'},(e)=>{  
    if(e){  
        console.error(e)  
    }  
    console.log('finish write json file')  
})
```

【Key Points】：

8-1 物件轉換成JSON字串

8-2 使用同步方式寫資料到檔案

8-3 使用非同步方式寫資料到檔案

# 課後練習題(Lab)

- **情節描述:**  
讀寫檔案是常見的系統操作，透過Node.js一樣可以做到，使用Node.js API fs模組讀寫檔案，並試著寫入JSON格式的字串。
- **預設目標:**
  - 理解fs模組的同步讀寫
  - 理解fs模組的非同步讀寫
  - 理解JSON字串的寫入
- **Lab01:理解fs模組的同步讀寫**
- **Lab02:理解fs模組的非同步讀寫**
- **Lab03:理解JSON字串的寫入**

Estimated time:  
**20 minutes**

8-19



## 【情節描述】

讀寫檔案是常見的系統操作，透過Node.js一樣可以做到，使用Node.js API fs模組讀寫檔案，並試著寫入JSON格式的字串。

## 【預設目標】

- 理解fs模組的同步讀寫
- 理解fs模組的非同步讀寫
- 理解JSON字串的寫入

Lab01:理解fs模組的同步讀寫

Lab02:理解fs模組的非同步讀寫

Lab03:理解JSON字串的寫入

完成後的程式與檔案，請參考 Example 資料夾的內容。

## 【Key Points】：

Lab01:理解fs模組的同步讀寫

Lab02:理解fs模組的非同步讀寫

Lab03:理解JSON字串的寫入

# 範例程式使用說明

- 範例程式資料夾: Module\_08\_example
- 使用步驟:
  1. 安裝 Node.js
  2. 安裝 Visual Studio Code
  3. 以 Visual Studio Code 開啟本模組的範例資料夾
  4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
  5. 在終端機視窗，輸入 node lab01.js

8-20



使用步驟:

1. 安裝 Node.js  
<https://nodejs.org/en/>
2. 安裝 Visual Studio Code  
<https://code.visualstudio.com/>
3. 以 Visual Studio Code 開啟範例資料夾  
在檔案總管，滑鼠右鍵點按「本範例資料夾」，從快捷功能表點選「以Code開啟」  
或者，啟動 Visual Studio Code 之後，功能表 File | Open Folder，選擇「本範例資料夾」
4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
5. 在終端機視窗，輸入 node lab01.js

【Key Points】：

程式執行環境 Node.js

程式開發編輯環境: Visual Studio Code

在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗，輸入：「node 程式檔名.js」