



表單及檔案上傳



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: 表單發送資料
- 8-2: 使用 multer
- 8-3: 過濾上傳的檔案



8-1

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

1. 在這節課中會帶大家撰寫一個靜態的 HTML 文件
2. 利用 HTML 裡的 form 標籤， post 表單資料到網頁伺服器上，觸發我們撰寫的API。
3. 再來會以 Multer 套件來處理編碼類型為 multipart/form-data 的表單。
4. 以及如何在上傳的過程中過濾掉非法的檔案。
5. 過濾文件的方式會分別說明前端跟後端過濾該如何進行。

【Key Points】：

利用HTML 裡的 form 標籤post form data 到網頁伺服器
以 Multer 套件來處理編碼類型為 multipart/form-data 的表單
過濾文件的方式會分別說明前端跟後端過濾該如何進行

8-1:表單發送資料

- 前端開發環境準備
- 寫一個 **index.html**
- 發送**GET**表單
- 發送**POST**表單



designed by freepik



designed by freepik
Image by http://www.pngall.com/?p=2654

8-2

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

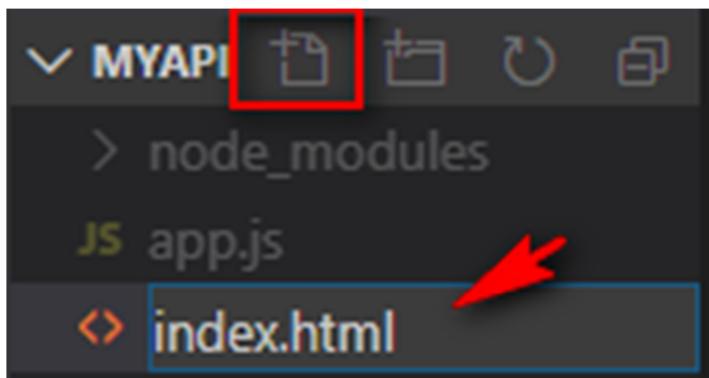
- 由於我們要利用 HTML 裡的 form 元件來發送資料，因此需要準備一個寫前端的環境，建議使用 VS Code
- 寫一個 index.html，並且透過 **sendfile** 的方式讀取並回應給瀏覽器
- 我們會先練習發送 GET 請求的表單
- 分析GET 表單送資料的方式，以及為何GET表單不適合哪種性質的資料
- 練習發送 POST 表單

【Key Points】：

寫一個 index.html，並且透過 **sendfile** 的方式讀取並回應給瀏覽器
先練習發送 GET 請求的表單
再練習發送 POST 表單

寫一個 index.html (一)

- 在 myapi 的開發目錄下用 code . 打開 VS Code
- 新增一個 index.html 文件



8-3

1. 在「命名提示字元」，輸入: mkdir myapi 建立名為 myapi 的資料夾作為專案名稱。
2. cd myapi 切換到該目錄
3. code . 啟動 VS Code 並且編輯 myapi 資料夾
4. 專案名稱也可改為你想要的名字
5. 在該目錄下點按「新增檔案」按鈕，新增一個 index.html 文件

<Note>

也可以在檔案總管，滑鼠右鍵點按特定資料夾，從快捷功能表選擇「以 Code 開啟」。

【Key Points】：

code . 啟動 VS Code 並且編輯 myapi 資料夾

可以在檔案總管，滑鼠右鍵點按特定資料夾，從快捷功能表選擇「以 Code 開啟」

點按「新增檔案」按鈕，新增一個 index.html 文件

寫一個 index.html (二)

- 編輯 index.html，這邊先簡單寫個一行標題，等等要測試從伺服器回傳靜態文件。
- 回到 app.js，把 express 引用進專案
- 利用 `sendfile(path)` 把檔案讀入，並回傳到瀏覽器上。

```
<!DOCTYPE html>
<html>

<body>
    <h1>Test Page</h1>
</body>

</html>
```

```
const express = require('express');
const app = express();

app.get("/", function(req, res) {
    res.sendfile(__dirname + '/index.html');
});

app.listen(3000);
```

8-4



1. 等等要測試如何在 express 上讀取靜態文件
2. 編輯 index.html
3. 簡單寫個有文字內容的頁面即可
4. 寫完之後我們回到 app.js 上
5. 把 express 模組引用進來
6. 寫個可以觸發 index.html 的路徑，因為是首頁用 "/" 作為網站根目錄即可
7. 最後在開啟 server 監聽 3000 port 即可
8. 打開瀏覽器，網址輸入: <http://127.0.0.1:3000/>，查看網頁結果
9. 沒意外應該可以看到一個顯示 Test Page 的頁面了

關鍵程式：

```
res.sendFile(__dirname + '/index.html');
```

`__dirname` 是當前文件目錄的保留變數，index.html 跟 app.js 放在同個目錄所以把路徑串起來即可讀取

【Key Points】：

測試如何在 express 上讀取靜態文件

```
res.sendFile()
```

`__dirname` 是當前文件目錄的保留變數

發送GET表單

- 在 HTML 中我們可以利用 `<form>` 標籤建立表單，以下是它的結構

```
<form action="目的地" method="傳遞方式">  
  
    表單內容、說明文字  
    ...  
    ...  
    表單輸入元素 <input> <option> <textarea>  
    ...  
    表單按鈕 <button>  
  
    送出表單 type='submit'  
  
</form>
```

8-5



先來了解 HTML Form 表單這個元素，當我們的靜態網頁希望可以向後端伺服器送資料請求時，就需要派表單上場。

表單的結構如簡報上所示，把該次請求所需要的 input 欄位全都放在同一個 `<form>` 裡面。

最後放置一個 `type="submit"` 的按鈕，就可以向目的 URL 發起對應的 HTTP 請求。

目的 URL 在 `action` 屬性中設定。

HTTP 請求類型則在 `method` 中設置。

【Key Points】：

目的 URL 在 `action` 屬性中設定

HTTP 請求類型則在 `method` 中設置

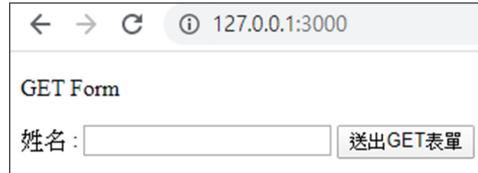
該次請求所需要的 input 欄位全都放在同一個 `<form>` 裡面，最後放置一個 `type="submit"` 的按鈕

發送GET表單

- 回到 index.html，我們在 <body> 裡面加入一個 GET 請求表單

```
<!DOCTYPE html>
<html>
<body>
    <p> GET Form </p>
    <form action="getdata" method="GET">
        姓名 : <input name="name" type="text">
        <input type="submit" value="送出GET表單">
    </form>
</body>
</html>
```

- 刷新瀏覽器可以看到結果



8-6

- 在大致了解 form 的結構之後我們回到 index.html 開始寫看看
- 在 <body> 標籤裡面加入一個 <form>
- action 設為 getdata · method 設為 GET
- 添加一個輸入姓名的輸入框 (input)
 - 姓名輸入框需要給定 name 屬性，該屬性的值會當作送出的 KEY
- 再加入一個 submit 按鈕
- 刷新瀏覽器 (F5) 應該就可以看到剛剛寫完的結果了

【Key Points】：

在 <body> 標籤裡面加入一個 <form>
action 設為 getdata · method 設為 GET
加入一個 submit 按鈕

發送GET表單

- 剛剛我們在 **action** 跟 **method** 分別填上 “**getdata**” 跟 “**GET**”
- 代表這份表單會用 **GET** 請求送到 **/getdata** 路由下，因此我們的 **app.js** 需要加入對應的程式碼

```
app.get('/getdata', function (req, res) {  
    console.log(req.query);  
    res.send('收到的資料 = ' + JSON.stringify(req.query))  
})
```

- 重新執行 **app.js**，在瀏覽器上表單填入姓名後點擊按鈕送出
- 可以發現 **URL** 被依照欄位內容格式化成對應的 **Query String** 了



8-7

- 剛剛我們在 **action** 跟 **method** 分別填上 “**getdata**” 跟 “**GET**”
- 這份表單會用 **GET** 請求送到 **/getdata** 路由下，因此我們的 **app.js** 需要加入對應的程式碼
- 回到 **app.js** 加入 ‘**/getdata**’ 的對應路由，並且把收到的資料簡單序列化成 **JSON** 回傳回去
- 重新執行 **app.js**，在瀏覽器上表單填入姓名後點擊按鈕送出
- 可以發現 **URL** 被依照欄位內容格式化成對應的 **Query String** 了

【Key Points】：

這份表單會用 **GET** 請求送到 **/getdata** 路由

因為 **GET** 表單會把內容明碼顯示在網址列上，因此**不適合傳送密碼等敏感資訊**。

等等會讓同學做看看 **POST** 表單，看看跟 **GET** 有沒有差別

發送POST表單

- 因為 GET 表單會把內容當成查詢字串送出去(明碼顯示在網址列上)，因此不適合傳送密碼等敏感資訊。
- 這時候我們就需要用 POST 表單了，回到 index.html，修改一下：

```
<p> POST Form </p>
<form action="postdata" method="POST">
    姓名 : <input name="name" type="text">
    密碼 : <input name="pwd" type="password">
    <input type="submit" value="送出POST表單">
</form>
```

- 刷新瀏覽器可以看到一個新的表單

A screenshot of a web browser displaying a POST form. The title bar says "POST Form". Inside the form, there are two input fields: one for "姓名" (Name) containing "我的名字" (My Name) and another for "密碼" (Password) containing a series of dots. To the right of these fields is a blue "送出POST表單" (Send POST Form) button.

8-8



- GET 表單會把內容當成查詢字串送出去(明碼顯示在網址列上)，因此不適合傳送密碼等敏感資訊
- 這時候我們就需要用 POST 表單了，回到 index.html 上面
- 改寫剛剛的 GET 表單，把 action 跟 method 都改寫成 post 的對應事件
- 除了剛剛的姓名輸入框之外我們在加入一個密碼欄位
- 把密碼欄位的 type 改成 password，可以讓密碼框在輸入時隱藏內容
- 重新刷新瀏覽器

<Note> 除了安全理由，POST可以傳送的資料量也遠比GET來得多。

【Key Points】：

GET 表單會把內容明碼顯示在網址列上

GET 不適合傳送密碼等敏感資訊

除了安全理由，POST可以傳送的資料量也遠比GET來得多

發送POST表單

- 回到 app.js 加入對應的 /postdata 事件。
- form data 預設用 UTF-8 編碼，因此使用 body-parser 套件解析

```
const bodyParser = require('body-parser');
var urlencodedParser = bodyParser.urlencoded() //解析 Form Data

app.post('/postdata', urlencodedParser, function (req, res) {
  console.log(req.body);
  res.send('收到的資料 = ' + JSON.stringify(req.body))
})
```

8-9



1. 回到 app.js 加入對應的 /postdata 事件
2. form data 在 HTTP body 裡面時預設用 UTF-8 編碼
3. 因此我們需要回去使用 body-parser 套件解析
4. 回到 app.js 把 body-parser 引用進來
5. 設置一個 urlencoded 的解析器
6. 在觸發 /postdata 路由時透過這個解析器把 form data 解析成物件
7. 最後回傳序列化的物件資料

【Key Points】：

把 body-parser 引用(require)進來

設置一個 urlencoded 的解析器

觸發 /postdata 路由時透過這個解析器把 form data 解析成物件

發送 POST 表單

- 重新執行 app.js，在瀏覽器上填入 POST 表單，並且點擊送出

POST Form

姓名: 我的名子 密碼: 送出POST表單

- 可以發現資料不會出現在網址列上面了，而且後端也能被正確解析成 Object。

收到的資料 = {"name": "我的名子", "pwd": "mytestpassword"}

```
body-parser deprecated undefined extended: p :6:35 { name: '我的名子', pwd: 'mytestpassword' }
```

8-10

- 重新執行 app.js
- 在瀏覽器上填入姓名跟密碼的 POST 表單，並且點擊送出
- 可以發現資料不會出現在網址列上面了
- 去終端機上面看也可以看到 form data 的資料被正確解析成 object 了
- 因此用 POST 來傳送表單資料可以具有部分的隱私性

【Key Points】：

重新執行 app.js

在瀏覽器上填入姓名跟密碼的 POST 表單，並且點擊送出

可以發現資料不會出現在網址列上面了

8-2：使用 multer

- 安裝 multer 套件
- 上傳文件表單
- 後端處理上傳文件



designed by freepik



designed by freepik

8-11

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

1. 上一單元了解 GET 跟 POST 表單
2. 接著我們要來看看如果想要實現檔案上傳的話在前後端應該怎麼做
3. 先安裝 multer
4. 接下來，學習如何使用 Multer 來接收上傳文件
5. 最後，觀察文件的 MIME 類型

【Key Points】：

安裝 multer

如何使用 Multer 來接收上傳文件

觀察文件的 MIME 類型

安裝 multer 套件

- 輸入：`npm install multer --save`
- multer 跟前面提過的 body-parser 一樣都是屬於中介軟體，不過不同的是 multer 只處理類型為 `multipart/form-data` 的資料
- 剛剛我們發送 POST 表單是用 `application/x-www-urlencoded` 類型編碼送出，但今天遇到要發送二進制文件時，效率就會很差。
- 這時候就需要 `multipart/form-data`

8-12



1. 輸入: `npm install multer --save`
2. multer 跟前面提過的 body-parser 一樣都是屬於中介軟體，不過不同的是 multer 只處理類型為 `multipart/form-data` 的資料
3. 剛剛我們發送 POST 表單是用 `application/x-www-urlencoded` 類型編碼送出
4. 發送二進制文件時如果用 UTF-8重新編碼的話，效率就會很差
5. 這時候就需要 `multipart/form-data` 的

HTML表單中的`enctype`屬性值有三種類型

- `application/x-www-urlencoded`
- `multipart/form-data`
- `text/plain`

rfc1867 在 Content-Type 的類型擴充了 `multipart/form-data` 用來支持向server發送二進制數據

【Key Points】：

安裝 multer 套件: `npm install multer --save`

multer 屬於中介軟體，只處理類型為 `multipart/form-data` 的資料

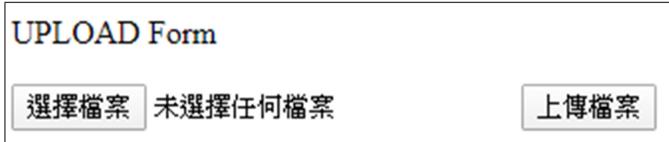
表單的`enctype`屬性要設定為"`multipart/form-data`"

上傳文件表單

- 回到 index.js 上，我們加入一個上傳文件的表單

```
<p> UPLOAD Form</p>
<form action="/upload_file" method="post" enctype="multipart/form-data">
    <input type="file" name="myfile">
    <input type="submit" value="上傳檔案">
</form>
```

- 在 <form> 標籤中加入 enctype 屬性，並且把值改成 **multipart/form-data**
- 刷新瀏覽器，可以看到表單已經加進去了



8-13



表單 <form> 屬性 enctype 主要有三個值可選，這個屬性管理的是表單的 MIME 編碼：

- application/x-www-form-urlencoded (預設值)
 - multipart/form-data
 - text/plain
-
- 因為我們要實作文件上傳，因此這邊要把 enctype 改成 multipart/form-data
 - 在 post 表單裡面新增一個文件選擇器
 - 文件選擇器的語法: <input type="file" name="myfile">
 - type 固定是"file"，name可以自己命名，以本例來說: "myfile"

【Key Points】：

表單 <form> 屬性 enctype 改成 multipart/form-data

文件選擇器的語法: <input type="file" name="myfile">

type 固定是"file"，name可以自己命名，以本例來說: "myfile"

上傳文件

- 接著在 app.js 中加入 multer 並且新增對應路由。

```
const express = require('express');
const multer = require('multer')
const app = express();

var upload = multer({ dest: 'upload/' }); // 設置檔案存放的路徑

app.post('/upload_file', upload.single('myfile'), function(req, res){
    res.send("上傳成功");
});

app.get("/", function(req, res) {
    res.sendFile(__dirname + '/index.html', function(err) {
        if (err) res.send(404);
    });
});

app.listen(3000);
```

8-14



- 如果還沒安裝 multer 請先安裝 (npm install multer –save)
- 接著在 app.js 中引用(require) multer 並且新增對應路由
- multer 用於處理檔案上傳到伺服器上，因此需先設置保存路徑
multer({ dest: 'upload/' })
- 接著在 post 路由中加進 upload.single('myfile')，這裡的 myfile 為剛剛我們HTML上面設的檔案選擇器名稱
- 因為只有上傳單一文件 所以用 single，假如要多文件的話要設定成 array

【Key Points】：

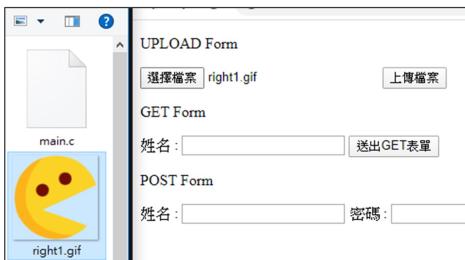
npm install multer

設置保存路徑 multer({ dest: 'upload/' })

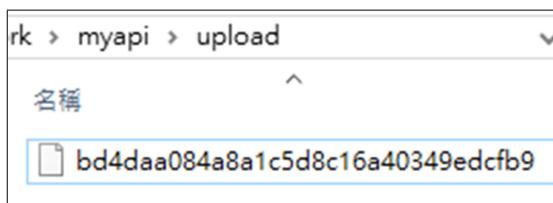
在 post 路由中加進 upload.single('myfile')，這裡的 myfile 為HTML設的檔案選擇器名稱

後端處理上傳文件

- 在瀏覽器上點擊“選擇檔案”按鈕，選取一個文件後上傳。



- 在專案目錄下，找到“upload”資料夾，裡面便會有我們剛剛上傳的檔案



8-15

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

- 當改寫完 app.js 後我們重新運行伺服器
- 之後在瀏覽器上刷新頁面
- 點擊“選擇檔案”按鈕，選取一個文件後上傳
- 回到專案目錄下，找到“upload”資料夾，裡面便會有我們剛剛上傳的檔案
- 這邊文件名稱被hash保存，我們等等會教到如何自定義文件保存規則

【Key Points】：

重新運行伺服器，瀏覽器上刷新頁面

選取一個文件後上傳

專案目錄下的“upload”資料夾，內含我們剛剛上傳的檔案

8-3：過濾上傳的檔案

- 顯示檔案資訊
- 自定義儲存模式
- 過濾檔案類型



designed by freepik



designed by freepik

8-16

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

- 同學們應該會發現剛剛上傳的檔案名稱已經被 hash 過而且無法明確從檔名判斷原檔案類型
- 這是因為 multer 可以讓我們自定義檔案上傳路徑、名稱、型別。
- 而剛剛的範例只有修改上傳路徑，其他選項維持預設沒做任何更動，因此才會有這個問題。
- 接下來我們會教學如何定義 multer 得上傳參數。
- 還有如何透過判斷 MIME 類型來過濾文件

【Key Points】：

剛剛上傳的檔案名稱已經被 hash 過而且無法明確從檔名判斷原檔案類型
multer 可以讓我們自定義檔案上傳路徑、名稱、型別
如何透過判斷 MIME 類型來過濾文件

顯示檔案資訊

- multer 把檔案的資訊跟儲存位置放在 `req.file` 裡面，上傳後可以讀取

```
app.post('/upload_file', upload.single('myfile'), function(req, res){  
    var file = req.file;  
    console.log('檔案類型 : %s', file.mimetype);  
    console.log('原始檔名 : %s', file.originalname);  
    console.log('檔案大小 : %s', file.size);  
    console.log('檔案存放路徑 : %s', file.path);  
    res.send("上傳成功" + file.path);  
});
```

- 在瀏覽器上重新上傳檔案，可以看到終端機上印出相關的資訊

```
檔案類型 : image/gif  
原始檔名 : right0.gif  
檔案大小 : 40169  
檔案存放路徑 : upload\cad43519f6dbe7b078fc6cac99128e8f
```

8-17



- multer 把檔案的資訊跟儲存位置放在 `req.file` 裡面
- 我們可以把它從 `request.file` 中抓出來，其中有些屬性值得我們看看
- 在 `upload_file` 裡面用 `console.log` 把各個屬性印出來
- 瀏覽器上重新上傳檔案，可以看到終端機上印出相關的資訊
- 觀察幾項重要資訊: `file.mimetype`, `file.originalname`, `file.size`, `file.path`

【Key Points】：

multer 把檔案的資訊跟儲存位置放在 `req.file` 裡面

幾項重要資訊: `file.mimetype`, `file.originalname`, `file.size`, `file.path`

讓同學們觀察不同類型的文件它們的 `mimetype` 是否有什麼不同

自定義儲存模式

- 當然預設的儲存模式不見得是我們想要的形式，multer 提供了 **storage** 參數來讓我們配置儲存設定。
- storage** 可以設置儲存路徑(**destination**)跟檔名(**filename**)
- 下面範例，把檔案名稱保存成「時間戳-原始檔名.xxx」格式

```
//自定義 storage
var myStorage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "upload"); // 保存的路徑 (需先自己創建)
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname); // 自定義檔案名稱
  }
});

var upload = multer({storage: myStorage}); // 設置 storage
```

8-18



- 預設的儲存模式不見得是我們想要的形式，multer 提供了 **storage** 這個參數來讓我們對檔案儲存的路徑、檔名進行自定義。
- storage** 可以設置儲存路徑(**destination**)跟檔名(**filename**)
 - destination**：設定資源的儲存路徑。注意，如果沒有這個配置項，預設會儲存在 `/tmp/uploads` 下。此外，路徑需要自己建立。
 - filename**：設定資源儲存在本地的檔名。
- 範例就是把檔案名稱保存成「時間戳記-原始檔名.xxx」格式
- 同學可以試試看其他的命名規則
- 如果我們想要回傳錯誤訊息可以用 `cb(null, false, {message: '具體錯誤'});` 的方式設置

【Key Points】：

multer 提供了 **storage** 這個參數來讓我們對檔案儲存的路徑、檔名進行自定義

storage 可以設置儲存路徑(**destination**)跟檔名(**filename**)

範例就是把檔案名稱保存成 時間戳記-原始檔名.xxx 得格式保存

過濾檔案類型

- 過濾檔案類型的方法主要可以分為在前端 (HTML、JS) 上處理，或是後端伺服器收到文件後再處理。
- 前端過濾的方式我們可以利用 `<input>` 控件裡面的 `accept` 屬性
- 如果要限制只要讀取 PNG、GIF、BMP 類型的圖片只需加上
`<input type="file" name="myfile" accept=".png,.gif,.bmp">`
- 如果想要全部的圖檔MIME類型
`<input type="file" name="myfile" accept="image/*">`
- 更多的 MIME Type 請看 <https://reurl.cc/mnd8AA>

8-19



1. 過濾檔案類型的方法主要可以分為在前端或後端處理
2. 前端過濾的方式我們可以利用 `<input>` 控件裡面的 `accept` 屬性
3. 如要限制只要讀取 PNG、GIF、BMP 類型的圖片可以在 `input` 裡面加上
 - `accept=".png,.gif,.bmp"`
4. MIME Type 請看 <https://reurl.cc/mnd8AA>
5. 當然在前端過濾是比較有安全疑慮的 (因為過濾規則放在別人的瀏覽器上而非我們自己的伺服器)

【Key Points】：

前端過濾的方式我們可以利用 `<input>` 控件裡面的 `accept` 屬性

`accept=".png,.gif,.bmp"`

讓同學試試看前端過濾是否有方法可以繞過規則 (提示：用瀏覽器開發人員介面 (F12))

過濾檔案類型

- 前端過濾的方式因為是在Client端做判斷，可能因為請求參數被惡意修改而變得不安全 (過濾失敗)。
- 因此我們可以利用 multer 的 fileFilter callback 去寫規則來自定義過濾方案，這個方法屬於在後端 (Server) 上多做一次檢查，相對安全。

```
var upload = multer({  
  storage: myStorage, // 設置 storage  
  fileFilter: function (req, file, cb) { // 檔案過濾  
    if (file.mimetype != 'image/gif') { // 檢查 MIME 類型是否不為 image/gif  
      return cb(new Error('Wrong file type'));  
    }  
    cb(null, true)  
  }  
});
```

8-20



- 前端過濾的方式因為是在Client端做判斷，可能因為請求參數被惡意修改而變得不安全 (過濾失敗)
- 後端上可以利用 multer 的 fileFilter callback 去寫規則來自定義過濾方案
- 我們在 multer 宣告時多加入一個 fileFilter 的引數，設置一個 callback function
- 在裡面我們去判斷當前檔案的 MIME 型態是否是我們想要的文件類型
- 當不符合類型時回傳錯誤訊息，符合就放行繼續保存文件

當檔案的 mimetype 不為指定類型時會返回一組錯誤訊息到瀏覽器上，這邊返回的類型可以自己決定，要返回 HTTP 狀態也可以，例如 403.3 伺服器禁止寫入，或是返回一個錯誤頁面...等

【Key Points】：

前端過濾的方式可能因為請求參數被惡意修改而變得不安全 (過濾失敗)
後端上可以利用 multer 的 fileFilter callback 去寫規則來自定義過濾方案
讓同學試看看上傳的時候可不可以上傳非 GIF 類型的檔案。

Summary < 精華回顧 >

- HTML 的 `<form>` 控件可以用 `action` 屬性控制目的路徑；`method` 控制請求方法。
- 送出表單的按鈕 `type` 需設為 `submit` 類型。
- GET 表單會把內容當成 `Query String` 送出去 (明碼顯示在網址列上)，因此不適合傳送密碼等敏感資訊。
- POST 表單預設是用 `application/x-www-urlencoded` 格式編碼資料。
- 當表單需要發送二進制文件時需要使用 `multipart/form-data` 格式。
- multer 提供 `storage` 參數可以設置儲存路徑(`destination`)跟檔名(`filename`)。
- 檔案過濾建議用 multer 的 `fileFilter callback` 去定義規則，會比前端判斷安全。



8-21

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

- 回顧一下上面我們教過的一些重點
1. HTML 的 `<form>` 控件可以用 `action` 屬性控制目的路徑；`method` 控制請求方法
 2. 送出表單的按鈕 `type` 需設為 `submit` 類型
 3. GET 表單會把內容當成 `Query String` 送出去 (明碼顯示在網址列上)
 - 因此不適合傳送密碼等敏感資訊
 4. POST 表單預設是用 `application/x-www-urlencoded` 格式編碼資料
 5. 當表單需要發送二進制文件時需要使用 `multipart/form-data` 格式
 - 二進制文件時如果用 UTF-8 重新編碼的話，效率就會很差
 6. multer 提供 `storage` 參數可以設置儲存路徑(`destination`)跟檔名(`filename`)
 - `destination`：設定資源的儲存路徑。注意，如果沒有這個配置項，預設會儲存在 `/tmp/uploads` 下。此外，路徑需要自己建立。
 - `filename`：設定資源儲存在本地的檔名。
 7. 檔案過濾建議用 multer 的 `fileFilter callback` 去定義規則，會比前端判斷安全

【Key Points】：

當表單需要發送二進制文件時需要使用 `multipart/form-data` 格式
multer 提供 `storage` 參數可以設置儲存路徑(`destination`)跟檔名(`filename`)
檔案過濾建議用 multer 的 `fileFilter callback` 去定義規則，會比前端判斷安全

線上程式題

- 8-1 如何限定只能上傳 PDF 檔

```
<input type="file" name="uploadedFile" multiple/>
```

- 請修改上述 HTML，使其只允許上傳pdf檔。

- 8-2 將上傳的檔案置於指定的位置

請修改程式碼，使其能夠將檔案存放在專案根目錄files資料夾底下，並且，檔名修改成「原始檔名-{當下ISO時間}」。

- 8-3 限定上傳的檔案數量

如何限定上傳的檔案數量？

8-22



題目名稱: 8-1 如何限定只能上傳 PDF 檔

內容說明:

```
<input type="file" name="uploadedFile" multiple/>
```

請修改上述 HTML，使其只允許上傳pdf檔。

題目名稱: 8-2 將上傳的檔案置於指定的位置

內容說明:

請修改程式碼，使其能夠將檔案存放在專案根目錄files資料夾底下，並且，檔名修改成「原始檔名-{當下ISO時間}」。

題目名稱: 8-3 限定上傳的檔案數量

內容說明:

如何限定上傳的檔案數量？

請閱讀教學系統的程式與說明，然後，到「// 作答區」填入答案。

答案有區分大寫小寫。

【Key Points】：

8-1 如何限定只能上傳 PDF 檔

8-2 將上傳的檔案置於指定的位置

8-3 限定上傳的檔案數量

課後練習題(Lab)

- **Module 8-1 – 表單發送資料**
 - Lab01: 安裝Node.js 環境
 - Lab02: 建立主要執行檔 app.js
 - Lab03: 發送 POST 表單
- **Module 8-2 – 使用 multer 上傳文件 + 過濾檔案**
 - Lab01: 安裝Node.js 環境
 - Lab02: 撰寫主程式與表單
 - Lab03: 過濾上傳檔案
- **完成後的程式與檔案，請參考 Example 資料夾的內容。**

Estimated time:
40 minutes

8-23



Module 08-1 – 表單發送資料

使用HTML 的 <input> 控件完成一份表單，並且在後端應用上接收請求並解析資料。

Lab01: 安裝Node.js 環境

Lab02: 建立主要執行檔 app.js

Lab03: 發送 POST 表單

Module 8-2 – 使用 multer 上傳文件 + 過濾檔案

藉助express、multer實現檔案上傳功能，並且實作前端與後端形式的文件過濾功能。

Lab01: 安裝Node.js 環境

Lab02: 撰寫主程式與表單

Lab03: 過濾上傳檔案

完成後的程式與檔案，請參考 Example 資料夾的內容。

【Key Points】：

發送 POST 表單

藉助express、multer實現檔案上傳功能

實作前端與後端形式的文件過濾功能

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的「8-1」或「8-2」
 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，輸入下列指令，安裝必要的模組套件:
`npm install express body-parser multer`
 6. 在終端機視窗，輸入 `node app.js`
 7. 啟動瀏覽器，連接 `http://localhost:3000` (8-2 測試時，僅限上傳 gif 圖檔)

8-24



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 開啟範例資料夾內的「8-1」或「8-2」
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，輸入下列指令，安裝必要的模組套件:
`npm install express body-parser multer`
6. 在終端機視窗，輸入 `node app.js`
7. 啟動瀏覽器，連接 `http://localhost:3000`

【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 主程式.js」