

NodeJS ExpressJS 網站設計



林新德

shinder.lin@gmail.com

參考專案：<https://bitbucket.org/lsd0125/mfee62-node.git>

- 1. Node.js 簡介
- 2. CJS 和 ESM 模組操作
- 3. http 模組 createServer
- 4. process 物件
- 5. Express 框架
- 6. EJS 樣板引擎
- 7. Query String 參數
- 8. 表單資料與 middlewares
- 9. 動態路由 (路徑參數)
- 10. 路由模組化
- 11. 前端發送表單資料的格式
- 12. Session
- 13. 時間格式
- 14. 連線 MySQL
- 15. 將 Session 資料存入 MySQL
- 16. 通訊錄列表
- 17. 通訊錄新增資料
- 18. 使用 Zod 驗證資料
- 19. 完成 CRUD 操作
- 20. CORS 跨來源資源共用

1. Node.js 簡介

- 2009年 Ryan Dahl 使用 Chrome 的 JavaScript 引擎（代號：V8），包裝成 JavaScript 執行環境（Runtime）Node.js。
- 可以在瀏覽器以外執行 JavaScript（像 Python 或 Ruby），讀寫檔案、寫服務程式、做資料庫連線等。
- 官網：<https://nodejs.org/>
- 安裝：至官網下載安裝檔。
- 安裝後，開啟命令提示列（command prompt、terminal）。
- 查看版本：`> node --version`
- 查看 npm 版本：`> npm -v`
- 進入 Node 互動模式：`> node`

1.1 使用全域套件

- 全域安裝 `es-checker` 模組（套件）用以測試 Node 支援 ES6 程度：
- `> npm install -g es-checker`
- `> sudo npm install -g es-checker` # mac 全域安裝需要權限
- 測試環境（測試用，通常只使用一次）：`> es-checker`

- 查看所有全域套件：`> npm ls -g`
- 查看 `npm` 的 `install` 指令簡短說明：`> npm install --help`
- 查看 `npm` 的 `install` 指令詳細說明：`> npm help install`
- 套件官網：<https://www.npmjs.com/>

1.2 建立專案

- 查看工作目錄內容 Windows：`> dir`
- 查看工作目錄內容 Mac：`$ ls -al`
- 建立資料夾：`> mkdir 資料夾名稱`
- 切換資料夾：`> cd 資料夾名稱`
- 到檔案管理員，點擊上方路徑，可以考備完整路徑。
- 使用 `npm` 初始化專案（建立專案套件管理檔案 `package.json`）：
- `> npm init -y`
- 從 `github` 或 `bitbucket` 網站 `clone` 下來的專案，可使用下式安裝 `package.json` 裡記錄的模組：
- `> npm install`

1.3 執行 JS 檔

1. 專案目錄內建立 `src/` 資料夾。
2. 建立 `src/func01.js` 內容如右。

```
const f1 = (a) => a * a;  
const f2 = () => {  
  let sum = 0;  
  for (let i = 1; i <= 10; i++) {  
    sum += i;  
  }  
  return sum;  
};  
console.log(f1(6));  
console.log(f2());
```

執行方式：> `node ./src/func01.js`

2. CJS 和 ESM 模組操作

- CJS 為 CommonJS，在預設的 Nodejs 專案為 CJS，使用 `require()` 和 `module.exports` 進行模組的匯入和匯出。
- <https://nodejs.org/dist/latest-v18.x/docs/api/module.html>
- ESM 為使用 ES6 module，可以使用 `import` 和 `export` 關鍵字。
- 從 Node12 開始，加入的 ESM 的功能。
- 在 `package.json` 中加入屬性 **"type": "module"** 可以使整個專案為 ESM。
- 也可以使用 `.mjs` 副檔名，指示檔案為 ES6 modules
- <https://nodejs.org/dist/latest-v18.x/docs/api/esm.html>

2.1 CommonJS 模組操作

```
// src/person.js
class Person {
  constructor(name='noname', age=20) {
    this.name = name;
    this.age = age;
  }
  toJSON(){
    const obj = {
      name: this.name,
      age: this.age,
    };
    return JSON.stringify(obj);
  }
}
module.exports = Person; // node 匯出類別
```

```
// src/person-test.js
const Person = require('./person');
const p1 = new Person('Bill', 26);
const p2 = new Person;
console.log(p1.toJSON());
console.log(p2.toJSON());
```

執行方式：> node src/person-test.js

2.2 ESM 模組操作

```
// func03.mjs
export const f1 = (a) => a * a;
const f3 = (a) => a * a * a;
export default f3;
```

****** 匯入模組檔案時，使用相對路徑，以「./」開頭，並需包含副檔名

```
// test-func03.mjs
import f3, {f1 as newName} from './func03.mjs';

console.log(newName(6));
console.log(f3(5));
```

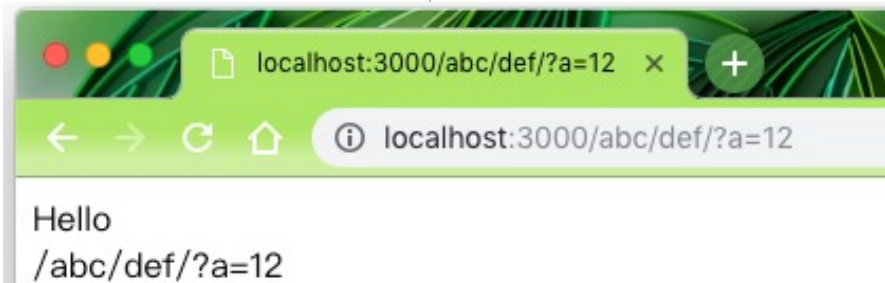
****** 本講義之後將以 ESM 為主

3. http 模組 createServer()

```
// src/http-server01.js
import http from "node:http";

const server = http.createServer((req, res) => {
  res.writeHead(200, {
    "Content-Type": "text/html; charset=utf-8",
  });
  res.end(`<h2>泥好</h2>
<p>${req.url}</p>
`);
});

server.listen(3000);
```



執行方式：> **node src/http-server01.js**

Ctrl-C 停止 server

3.1 使用 Nodemon 開發測試

- Nodemon 會監看專案裡的檔案，有任何檔案變更，會重新啟動。
- Nodemon 通常用在「服務」的程式（一直在待命的程式）。
- 全域安裝 nodemon
- `> npm i -g nodemon`
- `$ sudo npm i -g nodemon`
- nodemon 的功能：專案中相關檔案修改時，會重新啟動程式（Server 程式）。
- 正式環境建議使用的行程管理器：PM2 (<http://pm2.keymetrics.io/>)

3.2 HTTP 檔頭寫入檔案

```
// src/http-server02.js 寫入檔案
import http from "node:http";
import fs from "node:fs/promises";

const server = http.createServer(async (req, res) => {

  const jsonStr = JSON.stringify(req.headers, null, 4);
  await fs.writeFile("./headers.txt", jsonStr);

  res
    .writeHead(200, {
      "Content-Type": "application/json; charset=utf-8",
    })
    .end(jsonStr);
});

server.listen(3000);
```

3.3 注意非同步問題

** 這是個錯誤的作法

```
// src/http-server03.js
import http from "node:http";
import fs from "node:fs";
http
  .createServer((req, res) => {
    fs.writeFile("headers.txt", JSON.stringify(req.headers), (error) => {
      if (error) return console.log(error);
      console.log("HTTP檔頭儲存");
    });
    fs.readFile("src/http-server03.js", (error, data) => {
      if (error) {
        res.writeHead(500, { "Content-Type": "text/plain" });
        res.end("500 - src/http-server03.js");
      } else {
        res.writeHead(200, { "Content-Type": "text/plain" });
        res.end(data);
      }
    });
  })
  .listen(3000);
```


4. process 物件

- `process` 代表整個 `node` 執行的行程，其中包含重要的屬性。

4.1 透過 `process.argv` 取得命令列參數

- `process.argv` 執行程式時，取得命令列參數。

```
// src/argv.js  
console.log(process.argv);
```

```
// 執行  
$ node src/argv.js aaa bbb --c
```

4.2 透過 process.env 取得環境變數

- process.env 可以取得作業系統的環境變數。

```
// src/env01.js
process.env.MY_PARAM = "HELLO ENV"; // 直接設定
console.log(process.env);           // 讀取所有環境變數值
```

```
# 在 terminal 中設定環境變數
SET my_var=my_value      # Windows 使用 SET 指令
export my_var=my_value   # Mac 使用 export 指令
```

**** NodeJS 20.6 之後的作法**

- 直接使用 **--env-file** 參數設定欲載入的環境變數檔。
- **node** 和 **nodemon** 都可以使用
- 可以直接設定在 **package.json** 裡的 **scripts**。

```
"scripts": {  
  "dev": "nodemon --env-file=dev.env index.js",  
  "start": "node --env-file=production.env index.js"  
},
```

```
# dev.env  
WEB_PORT=3001  
DB_HOST=127.0.0.1  
DB_USER=root  
DB_PASS=root  
DB_NAME=proj57  
DB_PORT=3306
```

**** NodeJS 20.6 之前的作法**

- 安裝 **dotenv** 套件，以載入 **.env** 檔案裡的設定。
- `> npm i dotenv`
- **.env** 檔不應該加入 **git**（版本控制）。
- **.env** 可以放在專案以外的路徑。

```
import "dotenv/config"; // 使用專案的 .env
```

```
// 使用特定的 env 檔案  
import dotenv from "dotenv";  
dotenv.config({  
  path: "./sample.env",  
});
```


5. Express 框架

- 專案安裝 Express
- `> npm install --save express`
- `> npm i express`
- 查看 `package.json` 內容
- 建立主程式：`index.js`（沒有限定檔名）

5.1 Express初體驗

```
// 1. 引入 express
import express from "express";

// 2. 建立 web server 物件
const app = express();

// 3. 路由
app.get('/', function (req, res) {
  res.send('Hello World!');
});

// 4. Server 偵聽
app.listen(3000, function () {
  console.log('啟動 server 偵聽埠號 3000');
});
```

執行方式：> `nodemon index.js`

測試

`http://localhost:3000/`

5.2 自訂 404 頁面

測試

<http://localhost:3000/abc>

```
// *** 此段放在所有路由設定的後面 ***
```

```
app.use((req, res) => {  
  res.type("text/plain");  
  res.status(404);  
  res.send("404 - 找不到網頁");  
});
```

```
app.use((req, res) => {  
  res.status(404).send(`<h1>找不到頁面</h1>`);  
});
```

5.3 設定「靜態內容」資料夾

- 專案內建立資料夾 `public/`
- 在裡面放 `a.html`
- 將下列程式，放在其他路由設定的後面，404 路由的前面

```
app.use(express.static("public"));
```

- 使用瀏覽器查看 <http://localhost:3000/a.html>
- 靜態內容的資料夾可以設定多個，但「**請注意順序**」

5.4 使用 Bootstrap

- 使用 **npm** 安裝 **Bootstrap**，並設定靜態內容資料夾。

```
app.use("/", express.static("public"));  
app.use("/bootstrap", express.static("node_modules/bootstrap/dist"));
```

- 在 **a.html** 裡引入所需的 **css** 檔和 **js** 檔：

```
<link rel="stylesheet" href="/bootstrap/css/bootstrap.min.css" />  
<script src="/bootstrap/js/bootstrap.bundle.min.js"></script>
```

- 在 **a.html** 裡，放入 **bootstrap** 官網的 **navbar** 範例。

6. EJS 樣板引擎

- 官網：<https://ejs.co/>
- EJS (Embedded JavaScript templating) 套件位址：
- <https://www.npmjs.com/package/ejs>
- 使用樣版引擎的優點：可以把「呈現」和「邏輯處理」分開，易於管理。
- 安裝：`> npm i ejs`
- 並在專案中建立 `/views` 資料夾，做為存放樣版檔案的位置

6.1 設定 EJS

```
// 註冊樣版引擎  
app.set("view engine", "ejs");  
  
// 設定 views 路徑 (選擇性設定)  
// app.set("views", "我的路徑/views");
```

6.2 EJS 標籤

說明網址：<https://ejs.co/#docs>

<code><%</code>	'Scriptlet' tag, for control-flow, no output
<code><%=</code>	'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
<code><%=</code>	Outputs the value into the template (HTML escaped)
<code><%-</code>	Outputs the unescaped value into the template
<code><%=</code>	Comment tag, no execution, no output
<code><%%</code>	Outputs a literal '<%'
<code>%></code>	Plain ending tag
<code>-%></code>	Trim-mode ('newline slurp') tag, trims following newline
<code>_%></code>	'Whitespace Slurping' ending tag, removes all whitespace after it

6.3 測試 EJS

修改 `src/index.js` :

```
app.get("/", function (req, res) {  
  res.render("home", {name: "Shinder"});  
});
```

`views/home.ejs` 的內容 :

```
<h2><%= name %></h2>
```


6.4 使用 EJS 的 include()

建立 `views/parts/` 資料夾。將 `a.html` 檔案切割成 4 個部份。

```
<!DOCTYPE html>
<html lang="zh">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>a.html</title>
    <link rel="stylesheet" href="/bootstrap/css/bootstrap.min.css" />
  </head>
  <body>
    <div class="container">
      <!-- 使用 bootstrap navbar 的範例碼 -->
      <nav class="navbar navbar-expand-lg bg-body-tertiary">...</nav>
    </div>
    <div class="container">
      <h2>a.html</h2>
    </div>
    <script src="/bootstrap/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

1. 建立 **views/parts/** 資料夾。
2. 將 **a.html** 檔案切割成 4 個部份。
3. 修改 **home.ejs** 如下式。

```
<!-- views/home.ejs -->
<%- include('parts/html-head') %>
<%- include('parts/html-navbar') %>
<div class="container">
  <h2><%= name %></h2>
</div>
<%- include('parts/html-scripts') %>
<%- include('parts/html-tail') %>
```

<%- 用來避免 HTML 跳脫

路徑為相對路徑

.ejs 副檔名可以省略

6.5 以表格呈現陣列裡的資料

```
app.get("/json-sales", (req, res) => {  
  
  const sales = [  
    { name: "Bill", age: 28, id: "A001" },  
    { name: "Peter", age: 32, id: "A002" },  
    {  
      name: "Carl",  
      age: 29,  
      id: "A003",  
    },  
  ],  
  
  res.render("json-sales", { sales });  
});
```

views/json-sales.ejs

```
<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <th>編號</th>
      <th>姓名</th>
      <th>年齡</th>
    </tr>
  </thead>
  <tbody>
    <% for(let s of sales){ %>
      <tr>
        <td><%= s.id %></td>
        <td><%= s.name %></td>
        <td><%= s.age %></td>
      </tr>
    <% } %>
  </tbody>
</table>
```


7. Query String 參數

可以透過 `req.query.名稱` 取得，例如：`req.query.a`

```
http://localhost:3000/try-qs?a=1&b=3
```

```
http://localhost:3000/try-qs?a[]=2&a[]=bill
```

```
http://localhost:3000/try-qs?a=2&a=bill
```

```
http://localhost:3000/try-qs?a[age]=20&a[name]=bill
```


8. 表單資料與 middlewares

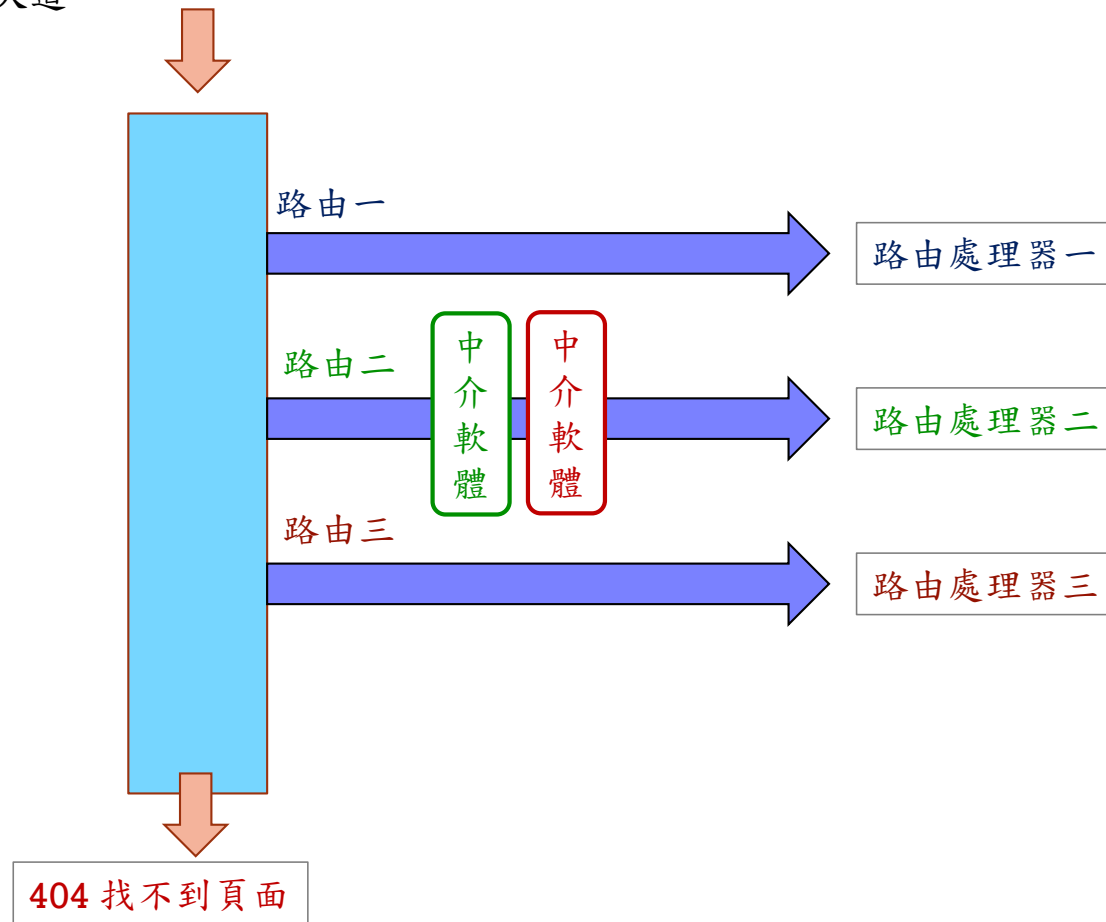
8.1 使用 body-parser 提供的 middlewares

```
// 取得 urlencoded parser, 使用 qs lib, 而不使用內建的 querystring lib
const urlencodedParser = express.urlencoded({ extended: true });

app.get("/try-post-form", (req, res) => {
  res.render("try-post-form");
});

// 把 urlencodedParser 當 middleware
app.post("/try-post-form", urlencodedParser, (req, res) => {
  res.render("try-post-form", req.body);
});
```


路由大道



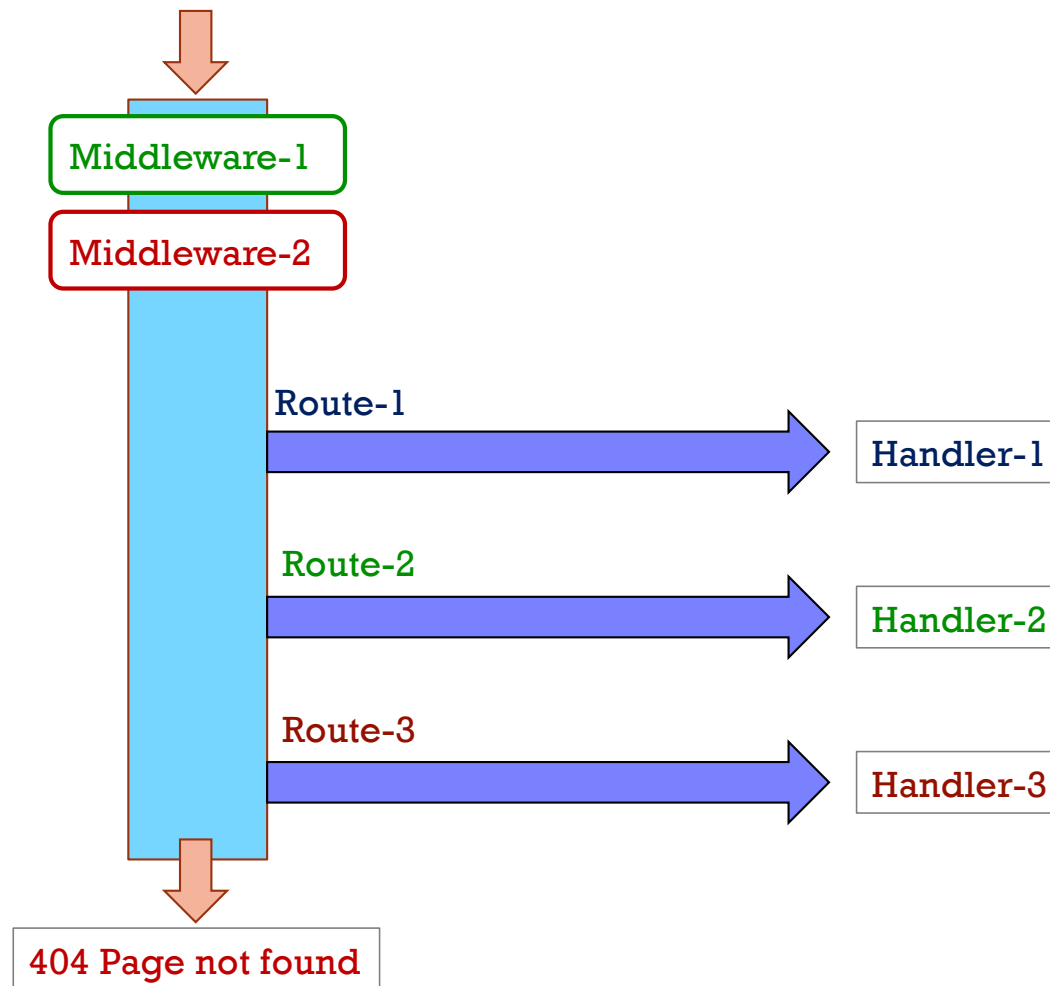
```
<%# views/try-post-form.ejs %>
<div class="col-lg-6">
  <form method="post" enctype="application/x-www-form-urlencoded">
    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" class="form-control" name="email">
      <% if(typeof email !== 'undefined'){ %>
        <small>上次輸入: <%= email %></small>
      <% } %>
    </div>
    <div class="form-group">
      <label for="password">Password</label>
      <input type="text" class="form-control" name="password">
      <% if(locals.password){ %>
        <small>上次輸入: <%= password %></small>
      <% } %>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
</div>
```

8.2 使用 Top-level Middleware

- 將 **body-parser** 設定成頂層 **middleware**，放在所有路由之前。
- 其包含兩種解析功能：**urlencoded** 和 **json**。

```
// parse application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// parse application/json
app.use(express.json());
```



8.3 使用 Postman 測試上傳 JSON 格式

The screenshot displays the Postman interface for a POST request. The URL bar shows `http://localhost:3000/try-post-form`. The method is set to `POST`. The `Headers` tab is active, showing a single header: `Content-Type: application/json`. The `Body` tab is also visible, showing the JSON payload: `{ "email": "shinder.lin@gmail.com", "password": "123456" }`.

Request URL: `http://localhost:3000/try-post-form`

Method: `POST`

Headers (1):

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Content-Type	application/json	

Body (JSON (application/json)):

```
1 {  
2   "email": "shinder.lin@gmail.com",  
3   "password": "123456"  
4 }
```

8.4 使用 REST Client

- 使用 VSCode 外掛 REST Client 測試後端功能。
- REST Client 使用 *.rest 文字檔，做為測試設定。

```
GET HTTP://localhost:3000/
```

```
### 分隔線
```

```
POST HTTP://localhost:3000/try-post
```

```
Content-Type: application/json
```

```
{  
  "name": "shinder"  
}
```

8.5 使用 Multer 處理檔案上傳

- 使用 `multer`
- 安裝:> `npm i multer`
- 說明可參考 `multer` 的 `npmjs` 主頁
- <https://www.npmjs.com/package/multer>
- 建立 `tmp_uploads` 做為檔案上傳的暫存資料夾（名稱可自訂）
- 建立 `public/img` 做為存放圖檔的資料夾（名稱可自訂）

```
// 主程式 index.js
```

```
import multer from "multer";  
const upload = multer({ dest: "tmp_uploads/" });
```

```
// 一個欄位上傳單一個檔案
```

```
app.post("/try-upload", upload.single("avatar"), (req, res) => {  
  res.json({  
    file: req.file,  
    body: req.body,  
  });  
});
```

```
// 一個欄位上傳多個檔案
```

```
app.post("/try-uploads", upload.array("photos"), (req, res) => {  
  res.json(req.files);  
});
```



```
console.log(req.file); //結果
```

```
{  
  "fieldname": "avatar",  
  "originalname": "test00.png",  
  "encoding": "7bit",  
  "mimetype": "image/png",  
  "destination": "tmp_uploads/",  
  "filename": "25e3c4de203391f7dc8bfce9360002b0",  
  "path": "tmp_uploads/25e3c4de203391f7dc8bfce9360002b0",  
  "size": 21634  
}
```

8.6 Multer 的 storage 和 fileFilter

```
import multer from "multer";
import { v4 as uuidv4 } from "uuid";
```

▪ modules/upload-imgs.js

```
// 篩選檔案和決定副檔名
const extMap = {
  "image/jpeg": ".jpg",
  "image/png": ".png",
  "image/webp": ".webp",
};
```

```
const fileFilter = (req, file, callback) => {
  callback(null, !!extMap[file.mimetype]);
};
const storage = multer.diskStorage({
  destination: (req, file, callback) => {
    callback(null, "public/img");
  },
  filename: (req, file, callback) => {
    const f = uuidv4() + extMap[file.mimetype];
    callback(null, f);
  },
});
export default multer({ fileFilter, storage });
```


9. 動態路由 (路徑參數)

- 使用變數代稱設定路由
- 使用 **regular expression** 設定路由

9.1 使用變數代稱設定路由

- : 冒號之後為代稱名
- ? 為選擇性的路徑區段
- * 為 wildcard

```
app.get('/my-params1/:action/:id', (req, res)=>{  
  res.json(req.params);  
});  
  
app.get('/my-params2/:action?/:id?', (req, res)=>{  
  res.json(req.params);  
});  
  
app.get('/my-params3/*//*?', (req, res)=>{  
  res.json(req.params);  
});
```

9.2 使用 Regular Expression 設定路由

```
app.get(/^\/hi\/?/, (req, res)=>{  
  let result = {  
    url : req.url  
  };  
  result.split = req.url.split('/');  
  res.json(result);  
});
```

測試以下的 url:

```
http://localhost:3000/hi  
http://localhost:3000/hi/  
http://localhost:3000/hi/123  
http://localhost:3000/hi123
```

```
// 手機號碼  
app.get(/^\/m\/09\d{2}-?\d{3}-?\d{3}$/i, (req, res)=>{  
  let u = req.url.slice(3);  
  u = u.split('?')[0];  
  u = u.split('-').join('');  
  res.json({ u });  
});
```


10. 路由模組化

- 通常將路由的設定分散放置 `/routes` 資料夾內的檔案
- 避免主程式過長，以方便管理
- 一個路由檔案相當於數個路由的群組，方便設定該群組的 **Top-level Middlewares**
- 方便設定前置路徑給路由群組

- 路由模組化 (方式一)

```
// routes/admin2.js
import express from "express";
const router = express.Router();

router.get("/admin2/:p1?/:p2?", (req, res) => {
  res.json({
    params: req.params,
    url: req.url,
    baseUrl: req.baseUrl,
    originalUrl: req.originalUrl,
  });
});

export default router;
```

```
// 在 index.js 內加入
import admin2Router from "../routes/admin2.js";
app.use(admin2Router); //當成 middleware 使用
```

- 路由模組化 (方式二)

```
// 在 index.js 內加入  
import admin3Router from "../routes/admin3.js";  
  
app.use("/admins", admin3Router); // 前段路由為 /admins
```

```
// routes/admin3.js
import express from "express";
const router = express.Router();
router
  .route("/member/edit/:id")
  .all((req, res, next) => {
    res.locals.memberData = {
      name: "shinder",
      id: "A002",
    };
    next();
  })
  .get((req, res) => {
    const obj = {
      data: res.locals.memberData,
    };
    res.send("get edit:" + JSON.stringify(obj));
  })
  .post((req, res) => {
    res.send("post edit:" + JSON.stringify(res.locals.memberData));
  });
export default router;
```


11. 前端發送表單資料的格式

- 1. application/x-www-form-urlencoded
- 2. application/json
- 3. multipart/form-data

11.1 使用 application/x-www-form-urlencoded

```
const fd = new FormData(document.form1);

const usp = new URLSearchParams(fd);

const r = await fetch("/login", {
  method: "POST",
  body: usp.toString(),
  headers: {
    "Content-Type": "application/x-www-form-urlencoded",
  },
});
const data = await r.json();
console.log({ data });
```

11.2 使用 application/json

```
const fd = new FormData(document.form1);

const dataObj = {};
for (let [k, v] of fd.entries()) {
  // console.log({ k, v });
  dataObj[k] = v;    // 將資料組成 Object
}

const r = await fetch("/login", {
  method: "POST",
  body: JSON.stringify(dataObj),
  headers: {
    "Content-Type": "application/json",
  },
});
const data = await r.json();
```

11.3 使用 multipart/form-data

```
// *** 後端需要 upload.none() 處理

const fd = new FormData(document.form1);
const r = await fetch("/login", {
  method: "POST",
  body: fd
});

const data = await r.json();
console.log({ data });
```


12. Session

- 若 **Client** 的瀏覽器停在某個網頁，使用者可能某些原因久久未再拜訪該網站，或者根本就已離開該站。此時會依 **Session** 的存活時間，決定 **Session** 是否有效。
 - **Server** 是以 **Client** 最後一次拜訪開始重新計時的，若 **Client** 在 **Session** 存活時間內，持續訪問該站，**Session** 就會一直有效。
 - 利用 **Cookie** 存放「**Session ID**」，在 **Client** 第一次拜訪時將 **Session ID** 存入 **Cookie**。
 - 有了 **Session ID** 之後，**Server** 會在主機（記憶體、檔案或資料庫）為每個 **Session ID** 建立一個對應的 **Session** 物件，資料就存在 **Session** 物件裡。
-
- 安裝 **express-session**
 - `> npm i express-session`

- 在 index.js 裡設定 session

```
import session from "express-session";
```

```
app.use(  
  session({  
    // 新用戶沒有使用到 session 物件時不會建立 session 和發送 cookie  
    saveUninitialized: false,  
    resave: false, // 沒變更內容是否強制回存  
    secret: "雜湊 session id 的字串",  
    // cookie: {  
    //   maxAge: 1200_000, // 20分鐘，單位毫秒  
    // },  
  })  
);
```

- 範例：顯示頁面刷新次數

```
app.get("/try-sess", (req, res) => {  
  // req.session.my_num = req.session.my_num || 0;  
  req.session.my_num ||= 0;  
  req.session.my_num++;  
  res.json(req.session);  
});
```

12.1 登入功能

- 登入表單 `views/login.ejs`
- 表單路由 `app.get("/login", (req, res) => {})`
- 處理表單的路由 `app.post("/login", async (req, res) => {})`
- 登出路由 `app.get("/logout", (req, res) => {})`

■ 登入表單 views/login.ejs

```
<form name="form1" onsubmit="sendData(event)">
  <div class="mb-3">
    <label for="email" class="form-label">email</label>
    <input type="text" class="form-control" id="email" name="email" />
    <div class="form-text"></div>
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">password</label>
    <input
      type="password"
      class="form-control"
      id="password"
      name="password"
    />
    <div class="form-text"></div>
  </div>

  <button type="submit" class="btn btn-primary">登入</button>
</form>
```

▪ index.js 登入的路由-1

```
app.get("/login", (req, res) => {
  res.render("login");
});

app.post("/login", async (req, res) => {
  let output = {
    success: false,
    postData: req.body,
    code: 0,
  };

  const sql = `SELECT * FROM members WHERE email=?`;

  const [rows] = await db.query(sql, [req.body.email]);
  if (!rows.length) {
    output.code = 400; // 帳號是錯的
    return res.json(output);
  }
  const member = rows[0];
  // 接下頁...
```

▪ index.js 登入的路由-2

```
// 接上頁...
const result = await bcrypt.compare(req.body.password, member.password);
if (!result) {
  output.code = 420; // 密碼是錯的
} else {
  output.success = true;
  output.code = 200;

  // 記錄到 session
  req.session.admin = {
    id: member.id,
    email: member.email,
    nickname: member.nickname,
  };
}

res.json(output);
});
```


■ 登出 : index.js

```
app.get("/logout", (req, res) => {  
  delete req.session.admin;  
  res.redirect("/");  
});
```


13. 時間格式

- 使用 `moment.js` 或輕量版 `dayjs`
 - 官網：<https://momentjs.com>
 - 說明文件：<https://momentjs.com/docs>
 - 安裝：`> npm i moment`
-
- 若需要時區的功能，請使用 `moment-timezone`
 - 安裝：`> npm i moment-timezone`

時間格式化輸出

```
// index.js
import moment from "moment-timezone";
```

```
app.get("/try-moment", (req, res) => {
  const fm = "YYYY-MM-DD HH:mm:ss";
  const m1 = moment(); // 取得當下時間的 moment 物件
  const m2 = moment("2024-02-29");
  const m3 = moment("2023-02-29");
  res.json({
    m1: m1.format(fm),
    m2: m2.format(fm),
    m3: m3.format(fm),
    m1v: m1.isValid(), // 是不是有效的日期
    m2v: m2.isValid(),
    m3v: m3.isValid(),
    m1z: m1.tz("Europe/London").format(fm),
    m2z: m2.tz("Europe/London").format(fm),
  });
});
```


14. 連線 MySQL

- 預先安裝 MySQL 資料庫管理系統
- 可安裝 MAMP (Apache, MySQL, PHP) 開發環境
- 使用 node 的 mysql2 套件連線
- 安裝：`> npm i mysql2`

- 使用範例資料庫

- <https://github.com/shinder/mmmh57-php/blob/master/db-training/shin01pma.sql>

```
CREATE TABLE `address_book` (  
  `sid` int(11) NOT NULL,  
  `name` varchar(255) NOT NULL,  
  `email` varchar(255) NOT NULL,  
  `mobile` varchar(255) NOT NULL,  
  `birthday` date NOT NULL,  
  `address` varchar(255) NOT NULL,  
  `created_at` datetime NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
INSERT INTO `address_book`  
(`sid`, `name`, `email`, `mobile`, `birthday`, `address`, `created_at`) VALUES  
(1, '李小明', 'ming01@gmail.com', '0918555666', '1995-10-02', '台南市', '2020-03-24 09:30:37');  
  
ALTER TABLE `address_book`  
  ADD PRIMARY KEY (`sid`);  
ALTER TABLE `address_book`  
  MODIFY `sid` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
```

14.1 建立連線模組 modules/connect-mysql.js

```
import mysql from "mysql2/promise";
const { DB_HOST, DB_USER, DB_PASS, DB_NAME, DB_PORT } = process.env;
console.log({ DB_HOST, DB_USER, DB_PASS, DB_NAME });

const db = mysql.createPool({
  host: DB_HOST,
  user: DB_USER,
  password: DB_PASS,
  database: DB_NAME,
  // port: DB_PORT,    // 如果使用 3306 以外的通訊埠需要設定
  waitForConnections: true,
  connectionLimit: 5,
  queueLimit: 0,
});

export default db;
```


14.2 讀取資料表

- `db.query()` 處理好之後，會取得一陣列。
- 陣列裡的第一個值為查詢的結果，通常我們只要這個項目。
- 陣列裡的第二個值為資料表欄位定義的相關資料，通常不需要。

```
app.get("/try-db", async (req, res) => {  
  const sql = "SELECT * FROM address_book LIMIT 3";  
  
  // 第一個值為查詢的結果，第二個值為資料表欄位定義的相關資料  
  const [rows, fields] = await db.query(sql);  
  res.json(rows);  
});
```

14.3 新增、修改、刪除

- `db.query()` 處理好之後，會取得一陣列，裡面只有一個元素值。
- 陣列裡的第一個值為結果物件，其屬性如下註解說明。

```
{  
  "fieldCount": 0,  
  "affectedRows": 1,    // 影響的列數  
  "insertId": 1000,     // 新增資料時，新增的 PK  
  "info": "",  
  "serverStatus": 2,  
  "warningStatus": 1,  
  "changedRows": 0      // 變更的列數  
}
```


15. 將 Session 資料存入 MySQL

- 安裝 `express-mysql-session` 套件

```
import session from "express-session";
import mysql_session from "express-mysql-session";
import db from "../modules/connect-mysql.js";

const MysqlStore = mysql_session(session);
const sessionStore = new MysqlStore({}, db);
```

```
app.use(
  session({
    saveUninitialized: false, resave: false,
    secret: "用以生成 session id 雜湊的字串",
    store: sessionStore,
  })
);
```

- 使用資料庫存放 **session** 資料時，可能會有資料不同步的問題，解決的方式：
 - 使用 **Session.destroy(callback)**
 - 使用 **Session.save(callback)**

```
// 使用 session.destroy()

app.get("/logout", (req, res) => {
  req.session.destroy((error) => {
    if (req.query.u) {
      // 如果有指示登出後要前往的頁面
      res.redirect(req.query.u);
    } else {
      res.redirect("/");
    }
  });
});
```

```
// 使用 session.save()

app.get("/logout", (req, res) => {
  delete req.session.admin;
  req.session.save((error) => {
    if (req.query.u) {
      // 如果有指示登出後要前往的頁面
      res.redirect(req.query.u);
    } else {
      res.redirect("/");
    }
  });
});
```


16. 通訊錄列表

■ 讀取資料的函式-1: routes/address-book.js

```
const getListData = async (req) => {
  const perPage = 20; // 每頁最多有幾筆
  let output = {
    success: false,    // 有沒有成功取得資料
    redirect: "",      // 有沒有要轉向
    info: "",
    page: 1,
    perPage,
    totalRows: 0,      // 總筆數
    totalPages: 0,     // 總頁數
    rows: [],          // 該頁資料
  };
  let page = +req.query.page || 1;
  if (page < 1) {
    output.redirect = "?page=1";
    output.info = "page 值不能小於 1";
    return output;
  }
  const [[{ totalRows }]] = await db.query(
    "SELECT COUNT(1) totalRows FROM `address_book`"
  );
};
```

■ 讀取資料的函式-2: routes/address-book.js

```
let totalPages = 0, rows = [];  
if (totalRows > 0) {  
  totalPages = Math.ceil(totalRows / perPage);  
  if (page > totalPages) {  
    output.redirect = "?page=" + totalPages;  
    output.info = "page 值不能大於總頁數";  
    return output;  
  }  
  const sql = `SELECT * FROM address_book ORDER BY sid DESC LIMIT ${  
    (page - 1) * perPage  
  }, ${perPage} `;  
  [rows] = await db.query(sql);  
  for (let r of rows) {  
    if (r.birthday) r.birthday = dayjs(r.birthday).format("YYYY-MM-DD");  
  }  
  output.success = true;  
}  
output = { ...output, perPage, page, totalRows, totalPages, rows };  
return output;  
};
```

- 呈現的頁面 `views/address-book/list.ejs`

```
<% for(let r of rows){ %>
<tr>
  <td>
    <a href="javascript: deleteItem(<%= r.sid %>)">
      <i class="fa-solid fa-trash-can"></i></a>
    </td>
    <td><%= r.sid %></td>
    <td><a href="/address-book/api/<%= r.sid %>"><%= r.name %></a></td>
    <td><%= r.mobile %></td>
    <td><%= r.email %></td>
    <td><%= r.birthday %></td>
    <td><%= r.address %></td>
    <td>
      <a href="address-book/edit/<%= r.sid %>">
        <i class="fa-solid fa-pen-to-square"></i></a>
      </td>
    </tr>
  <% } %>
```


17. 通訊錄新增資料

```
<%# views/address-book/add.ejs 中的表單 (部份) %>
<h5 class="card-title">新增通訊錄資料</h5>
<form name="form1" onsubmit="sendData(event)">
  <div class="mb-3">
    <label for="name" class="form-label">name</label>
    <input type="text" class="form-control" id="name" name="name" />
    <div class="form-text"></div>
  </div>
  <div class="mb-3">
    <label for="email" class="form-label">email</label>
    <input type="text" class="form-control" id="email" name="email" />
    <div class="form-text"></div>
  </div>
  <div class="mb-3">
    ...
  </div>
</form>
```

// views/address-book/add.ejs 中的發 AJAX 片段

```
if (isPass) {
  const fd = new FormData(document.form1);
  const usp = new URLSearchParams(fd);
  fetch("/address-book/api", {
    method: "POST",
    body: usp.toString(),
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
  }).then((r) => r.json())
    .then((obj) => {
      console.log(obj);
      if (obj.success) {
        alert("資料新增成功");
      } else {
        for(let s in obj.errors){
          if(document.form1[s] && (document.form1[s] instanceof Element)){
            const el = document.form1[s];
            el.closest("div").classList.add("warning");
            el.nextElementSibling.innerHTML = obj.errors[s];
          }
        }
      }
    })
    .catch((ex) => console.log(ex));
}
```

■ 新增資料路由-1: routes/address-book.js

```
router.get("/add", async (req, res) => {  
  res.locals.pageName = "ab-add";  
  res.render("address-book/add");  
});
```

// 新增資料的功能

```
router.post("/api", async (req, res) => {
```

```
  const output = {  
    success: false,  
    errors: {},  
    result: {},  
    postData: req.body, // 除錯檢查用  
  };  
  // TODO: 欄位格式檢查
```

```
  let isPass = true; // 有沒有通常檢查
```

```
  if (req.body.name) {  
    let { name, email, mobile, birthday, address } = req.body;
```

■ 新增資料路由-2: routes/address-book.js

```
// 檢查姓名欄位
if (name.length < 2) {
  output.errors.name = "姓名字串長度請大於 2 個字元";
  isPass = false;
}
// 檢查 email
if (!email_re.test(email)) {
  output.errors.email = "Email 格式不正確";
  isPass = false;
}
birthday = dayjs(birthday);
if (!birthday.isValid()) {
  birthday = null;
} else {
  birthday = birthday.format("YYYY-MM-DD");
}

let result;
if (isPass) {
  try {
```


■ 新增資料路由-3: routes/address-book.js

```
    const sql = `INSERT INTO address_book
    ( name, email, mobile, birthday, address, created_at)
    VALUES (?, ?, ?, ?, ?, NOW() )`;
    [result] = await db.query(sql, [
      name,
      email,
      mobile,
      birthday,
      address,
    ]);
    output.success = !!result.affectedRows;
    output.result = result;
  } catch (ex) {
    output.error = "SQL 錯誤";
    output.ex = ex;
  }
}
}
res.json(output);
});
```


18. 使用 Zod 驗證資料

- 1. 安裝 zod

```
npm install zod
```

- 2. 引入 zod

```
import { z } from "zod";
```

- 3. 撰寫 schema

- 4. 驗證

```
const mySchema = z.object({
  name: z
    .string({ message: "姓名欄為必填" })
    .min(3, { message: "長度要三個字以上" }),
  email: z.string().email({ message: "請填寫正確的電郵格式" }),
  birthday: z
    .string()
    .date("生日的日期格式為: YYYY-MM-DD")
    .optional() // 選擇性的欄位
    .or(z.literal("")), // 值可以是空字串
});
```

```
const { name, email, mobile, birthday, address } = req.body;
const data = { name, email, mobile, birthday, address };

const checkResult = mySchema.safeParse(data);
if (!checkResult.success) {
  // 沒有通過資料檢查
  output.errors = checkResult.error.issues;
  return res.json(output);
}
```


19. 完成 CRUD 操作

- 試著完成修改資料及刪除資料的功能
- 依相同的方式完成管理者資料的新增
- 管理者登入功能
- 登入後才能編輯員工資料

20. CORS 跨來源資源共用

- 跨來源資源共用（**Cross-Origin Resource Sharing (CORS)**）是一種使用額外 HTTP 標頭令目前瀏覽網站的使用者代理取得存取其他來源（網域）伺服器特定資源權限的機制。
- 何謂相同的來源（**same origin**）？**協定**、**網域**、**通訊埠**，三者皆相同。
- <https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS>
- 當使用 **fetch()** 或傳統 **AJAX** 跨源（**cross origin**）去呼叫 **API** 時，需要對方主機允許。
- **npmjs**主頁：<https://www.npmjs.com/package/cors>
- 安裝：`> npm i cors`

- 一般的使用方式（不需使用 **cookies** 和 **session** 時）

```
import cors from "cors";
```

```
app.use(cors());
```

- 跨來源需要使用 **cookies** 和 **session** 時，必須是相同的網域或 IP（使用白名單）

```
import cors from "cors";
```

```
const whitelist = [undefined, "http://localhost:3030"];
const corsOptions = {
  credentials: true,
  origin: function (origin, callback) {
    console.log("origin: " + origin);
    if (whitelist.indexOf(origin) !== -1) {
      callback(null, true);
    } else {
      callback(null, false);
    }
  },
};
app.use(cors(corsOptions));
```

■ 準備後端服務

```
// 有使用到 req.session 的 api 都可以  
  
app.get("/try-sess", (req, res) => {  
  req.session.my_num ||= 0;  
  req.session.my_num++;  
  res.json(req.session);  
});
```

■ 前端 JS

```
function doSend() {  
  fetch("//localhost:3030/try-sess", {  
    credentials: "include", // cross origin 傳送 cookie  
  })  
  .then((r) => r.json())  
  .then((obj) => {  
    document.querySelector("#info").innerHTML = JSON.stringify(obj);  
  });  
}
```


補充一 使用 Node/Express 服務 React 發佈後的專案

- 1. 在特定資料夾（或家目錄）複製專案
- **git clone** <https://github.com/shinder/shinder-react-hooks.git>
- 2. terminal 進入專案目錄
- **cd ./** [shinder-react-hooks](#)
- 3. 安裝套件
- **npm i**
- 4. 測試開發環境
- **npm start**
- 5. 按「**Ctrl + C**」停止開發執行

- 6. 發佈專案
- `npm run build`
- 7. 將發佈產生的 `/build` 資料夾，整個複製到 **Node/Express** 專案的根目錄
- 8. 將下列程式碼，加入主程式 `index.js`

```
// 服務 react 發佈後的專案
app.use("/", express.static("build"));
app.get("*", (req, res) => {
  res.send(`<!doctype html><html>index.html 的內容，略...</html>`);
});
```

- 9. 若有其他後端的路由，應該放置在上列程式碼之前。

附錄一

Request 接收的資料	
req.query	網址上 Query String 參數
req.body	表單資料 (body-parser, multer 套件)
req.file	上傳單一檔案 (multer 套件)
req.files	上傳多個檔案 (multer 套件)
req.params	路徑變數
req.session	Session 物件 (express-session 套件)

附錄二

Response 的回應方法	
<code>res.end()</code>	預設回應一般文字內容 (<code>text/plain</code>)
<code>res.send()</code>	依資料回應不同的內容 1. <code>String</code> : 回應 <code>text/html</code> 2. <code>Object</code> : 回應 <code>application/json</code>
<code>res.json()</code>	回應 <code>application/json</code>
<code>res.render()</code>	使用樣版檔 (<code>.ejs</code>) 回應 <code>text/html</code>
<code>res.redirect()</code>	轉向到別的網址

附錄三

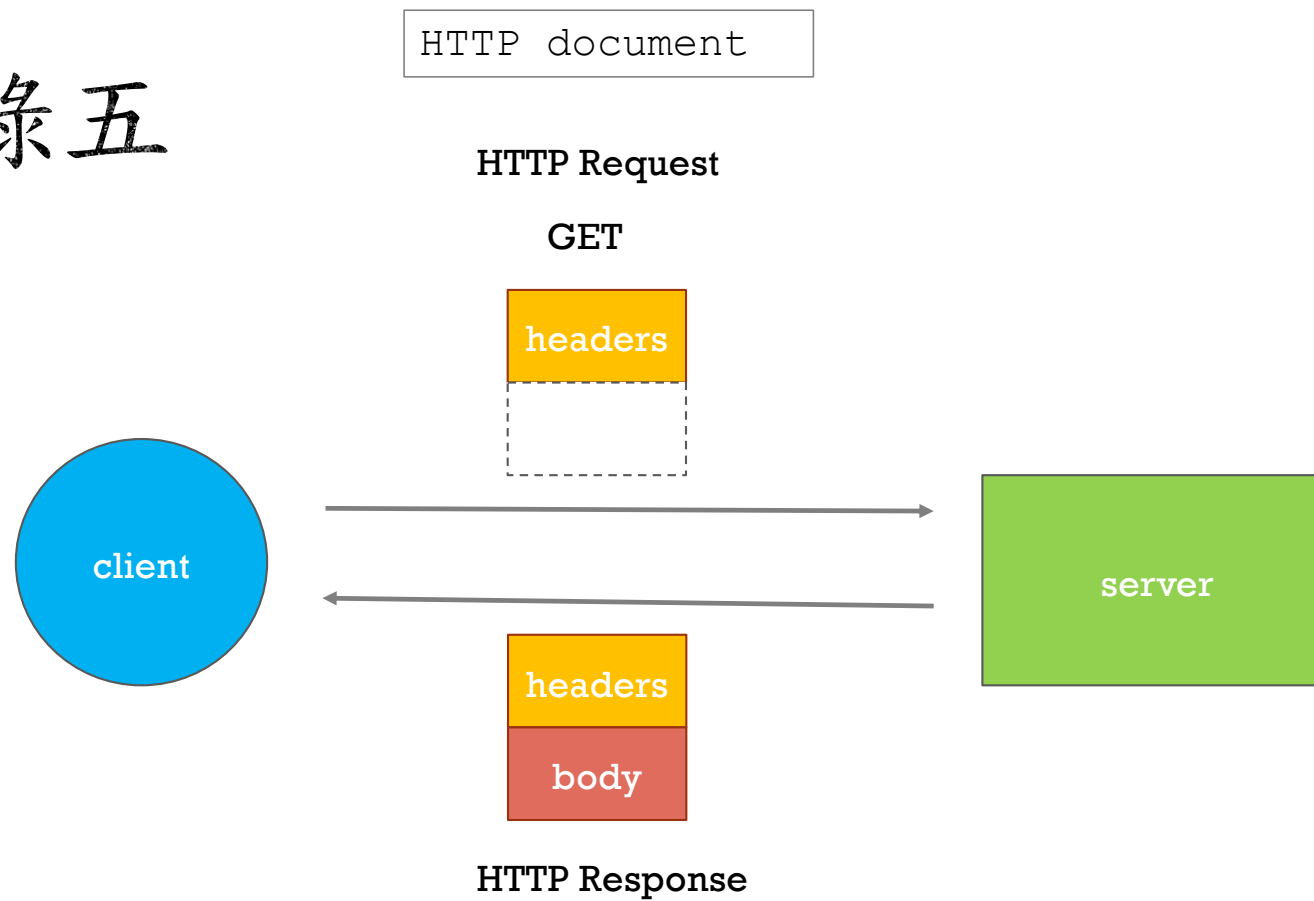
RESTful API 概念

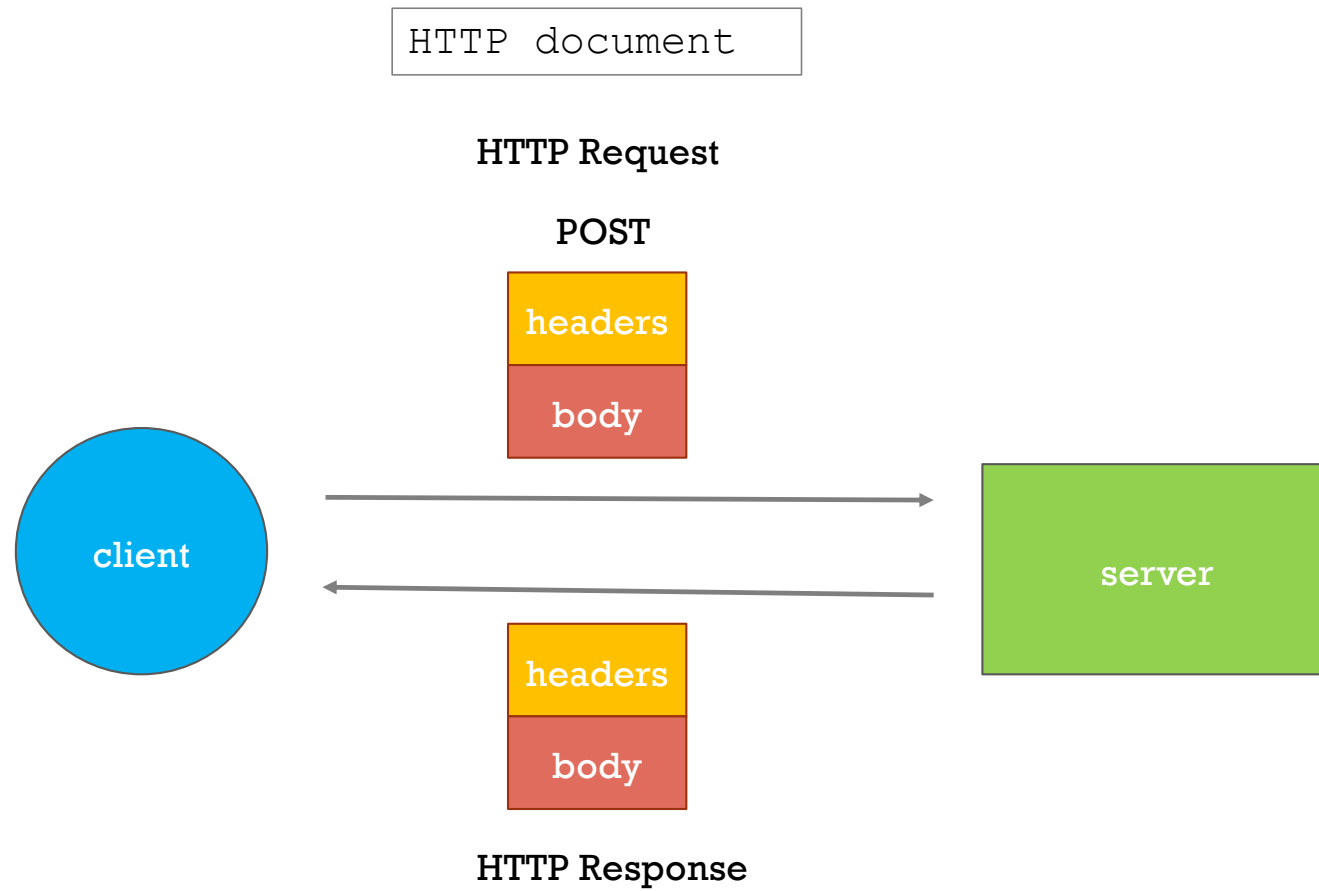
HTTP 方法	範例路徑	功能
GET	/products	Read, 讀取列表資料
GET	/products/12	Read, 讀取 id 為 12 的單筆資料
POST	/products	Create, 新增資料
PUT	/products/12	Update, 修改 id 為 12 的單筆資料
DELETE	/products/12	Delete, 刪除 id 為 12 的單筆資料

附錄四

URL 連結省略寫法說明	
https://stackoverflow.com/questions/39458201?a=1	完整 URL
//stackoverflow.com/ questions/39458201 ?a=1	省略協定
/questions/39458201 ?a=1	省略網域
?a=1	省略路徑
#my_hash	頁面內的連結
空字串	省略整個 URL
abc/def	相對路徑
./abc/def	相對路徑

附錄五







Thank you !