HTTP server (httpd)

喬逸偉 (Yiwei Chiao)

1 switched 檔案服務

目前 index.js 利用 switch 指令來依據使用者要求，傳回不同的檔案內容到瀏覽器端。程式碼片段類似下面的型態：

```
19. request.on('end', () => {
20.   switch (request.url) {
21.     case '/':
22.       fs.readFile('../htdocs/index.html', (err, data) => {
23.         if (err) {
24.           console.log(' 檔案讀取錯誤 ');
25.         }
26.         else {
27.           response.writeHead(200, {
28.             'Content-Type': 'text/html'
29.           });
30.
31.           // 傳送回應內容。
32.           response.write(data);
33.           response.end();
34.         }
35.       });
36.
37.       break;
38.
39.     case '/assets/css/styles.css':
40.       fs.readFile('../htdocs/assets/css/styles.css', (err, data) => {
41.         if (err) {
42.           console.log(' 檔案讀取錯誤 ');
43.         }
44.         else {
45.           response.writeHead(200, {
46.             'Content-Type': 'text/css'
```

```
47.             });
48.
49.             // 傳送回應內容。
50.             response.write(data);
51.             response.end();
52.           }
53.         });
54.
55.         break;
56.
57.       case '/js/index.js':
58.         fs.readFile('../htdocs/js/index.js', (err, data) => {
59.           if (err) {
60.             console.log(' 檔案讀取錯誤 ');
61.           }
62.           else {
63.             response.writeHead(200, {
64.               'Content-Type': 'application/javascript'
65.             });
66.
67.             // 傳送回應內容。
68.             response.write(data);
69.             response.end();
70.           }
71.         });
72.
73.         break;
74.
75.       default:
76.         console.log(` 未定義的存取 : ${request.url}`);
77.
78.         response.end();
79.
80.         break;
81.     }
82. });
```

原則上就是依第 76 行回報的**未定義的存取**，增加 switch 陳述裡的 case 分支，回應相對應的**檔案內容**。看起來不差，可是如果，多一個要求，就多一段 case 陳述，對複雜一點的網站來說，可能很快就會耗盡我們的腦容量來追踪。

我們需要一個**聰明點**的方法。

1.1 抽離獨立函式

首先注意到，第 22~37, 40~55 和 58~73 的程式碼**基本上**完全相同，只有 fs.readFile(...) 的檔名參數和 response.writeHead(....) 的 Content-Type 參數不同。這並不意外，因為這些程式碼本來就是 22~37 行程式碼的複製品 (copy)。可以想見，如果再增加 case，可能也只是再增加一份 22~37 的 copy 而已。

所以，第一步可以將 22~37 行程式碼抽離成一個獨立的函式 (function)，如下：

```
/**
  * 利用 http.ServerResponse 物件回傳檔案內容
  *
  * @name serve
  * @function
  * @param response - http.ServerResponse 物件
  * @param fname - 要回傳的檔案名
  * @param mime - 回傳檔案內容的 MIME_type
  * @returns {undefined}
  */
let serve = (response, fname, mime) => {
  let fs = require('fs');

  fs.readFile(fname, (err, data) => {
    if (err) {
      console.log(' 檔案讀取錯誤');
    }
    else {
      response.writeHead(200, {
        'Content-Type': mime
      });

      response.write(data);
      response.end();
    }
  });
};
```

上面的程式碼定義了一個新的函數，稱為 serve，它需要三 (3) 個參數，分別是 http.ServerResponse (response), 檔案名稱 (fname) 和資料的 MIME_type (mime)。而函式的內容就是原始 switch case 裡的內容。

有了這個函數，原來的 switch 可以改寫成：

```
    switch (request.url) {
```

```
      case '/':
        serve(response, '../htdocs/index.html', 'text/html');

        break;

      case '/assets/css/styles.css':
        serve(response, '../htdocs/assets/css/styles.css', 'text/css');

        break;

      case '/js/index.js':
        serve(response, '../htdocs/js/index.js', 'application/javascript');

        break;

      default:
        console.log(' 未定義的存取: ' + request.url);

        response.end();

        break;
    }
```

如此，大幅簡化了 switch 陳述。

完整的 index.js 如下：

```
'use strict';

let http = require('http');

/**
  * 利用 http.ServerResponse 物件回傳檔案內容
  *
  * @name serve
  * @function
  * @param response - http.ServerResponse 物件
  * @param fname - 要回傳的檔案名
  * @param mime - 回傳檔案內容的 MIME_type
  * @returns {undefined}
  */
let serve = (response, fname, mime) => {
  let fs = require('fs');
```

```
  fs.readFile(fname, (err, data) => {
    if (err) {
      console.log(' 檔案讀取錯誤');
    }
    else {
      response.writeHead(200, {
        'Content-Type': mime
      });

      response.write(data);
      response.end();
    }
  });
};

http.createServer((request, response) => {
  let fs = require('fs');
  let postData = '';

  // 利用 'data' event 消耗掉 data chunk;
  // 'end' event 才會被 fired
  request.on('data', (chunk) => {
    postData += chunk;

    console.log(
      ' 接收的 POST data 片段: [' + chunk + '].'
    );
  });

  request.on('end', () => {
    switch (request.url) {
      case '/':
        serve(response, '../htdocs/index.html', 'text/html');

        break;

      case '/assets/css/styles.css':
        serve(response, '../htdocs/assets/css/styles.css', 'text/css');

        break;
```

```
      case '/js/index.js':
        serve(response, '../htdocs/js/index.js', 'application/javascript');

        break;

      default:
        console.log(' 未定義的存取: ' + request.url);

        response.end();

        break;
    }
  });
}).listen(8088);

// log message to Console
console.log(' 伺服器啓動，連線 url:  http://127.0.0.1:8088/');

// index.js
```

1.2 建立路由表 (routing table)

將 serve() 函數獨立出去後，index.js 是有了大幅改善，但還是不能滿意。因為，還是要為新的 request.url 增加新的 switch case；還是在增加腦子的負荷。

重新審視 switch case 陳述，可以注意到，switch case 的用途不過是用來將 request.url 對應到真正的**檔案內容**和檔案內容的 MIME_type，沒有其它的用途。而如果只是為了作**對應** (mapping)，有個更古老，好用的方法來處理，**查表** (table lookup)。這裡，借用電腦網路的名詞，就稱這個將要建立的**表**是一個**路由表** (routing table)。

下面是建立出來的路由表：

```
const routingTable = {
  '/': {
    url: '../htdocs/index.html',
    mime: 'text/html'
  },

  '/assets/css/styles.css': {
    url: '../htdocs/assets/css/styles.css',
    mime: 'text/css'
  },
```

```
  '/js/index.js': {
    url: '../htdocs/js/index.js',
    mime: 'application/javascript'
  },
};
```

如上表顯示的，所謂的 **routing table** 在程式裡其實就是一個普通的 JavaScript 物件 (object)；只不過這個**物件**的**屬性** (property) 設計過，每一個**屬性**的名稱都對應一個不同的 request.url，而它的**屬性值** (value) 則是另一個簡單的物件，記錄了真實的檔案位置和對應的 MIME_type。

有了 routingTable 的協助，request.on('end') 的程式片段可以改寫如下：

```
  request.on('end', () => {
    if (request.url in routingTable) {
      let obj = routingTable[request.url];

      serve(response, obj.url, obj.mime);
    }
    else {
      console.log(' 未定義的存取: ' + request.url);

      response.end();
    }
  });
```

原來的 switch 陳述不見了，而且更棒的是，如果有新的 request.url 出現，這裡不用作任何事情，只需要去修改 routingTable 的定義就行了。

1.3 簡化路由表

最後再回頭看一眼剛定義的 routingTable。注意到除了 '/' 對應到 '../htdocs/index.html'，使得 request.url 和**真實**的 url 不同之外，另外兩組的 request.url 和**真實**的 url 幾乎完全相同。這有個重大的缺點：

- 將網站的內部結構以 url 的型式暴露在外。

request.url 基本上就是使用者在瀏覽器網址列上輸入的網址，或 .html 裡記錄的 url link。讓它和網站上的目錄結構作完整的對應，等於告訴使用者網站的架構安排是什麼樣子。讓網站暴露在不必要的風險中。

還好，routingTable 本身就是個對應表，修改這個表就可以解決問題。而修改的目標，除了斷離 request.url 和**真實** url 的字面聯繫外，當然也希望簡化使用者的麻煩。所以，簡化的方向在 request.url 上。

先看修改過的 routingTable。

```
const routingTable = {
  '/': {
    url: '../htdocs/index.html',
    mime: 'text/html'
  },

  '/styles.css': {
    url: '../htdocs/assets/css/styles.css',
    mime: 'text/css'
  },

  '/breakit.js': {
    url: '../htdocs/js/index.js',
    mime: 'application/javascript'
  },
};
```

簡單說，就是將 request.url 裡的路徑資訊**移除**。而相對應的，需要修改 index.html。而這一部份就留作練習。

最後，附上到目前為止，完整的 index.js 檔。

```
 1. 'use strict';
 2.
 3. let http = require('http');
 4.
 5. const routingTable = {
 6.   '/': {
 7.     url: '../htdocs/index.html',
 8.     mime: 'text/html'
 9.   },
10.
11.   '/styles.css': {
12.     url: '../htdocs/assets/css/styles.css',
13.     mime: 'text/css'
14.   },
15.
16.   '/breakit.js': {
17.     url: '../htdocs/js/index.js',
18.     mime: 'application/javascript'
19.   },
20. };
```

```
21.
22. /**
23.    * 利用 http.ServerResponse 物件回傳檔案內容
24.    *
25.    * @name serve
26.    * @function
27.    * @param response - http.ServerResponse 物件
28.    * @param fname - 要回傳的檔案名
29.    * @param mime - 回傳檔案內容的 MIME_type
30.    * @returns {undefined}
31.    */
32. let serve = (response, fname, mime) => {
33.   let fs = require('fs');
34.
35.   fs.readFile(fname, (err, data) => {
36.     if (err) {
37.       console.log(' 檔案讀取錯誤');
38.     }
39.     else {
40.       response.writeHead(200, {
41.         'Content-Type': mime
42.       });
43.
44.       response.write(data);
45.       response.end();
46.     }
47.   });
48. };
49.
50. http.createServer((request, response) => {
51.   let fs = require('fs');
52.   let postData = '';
53.
54.   // 利用 'data' event 消耗掉 data chunk;
55.   // 'end' event 才會被 fired
56.   request.on('data', (chunk) => {
57.     postData += chunk;
58.
59.     console.log(
60.       ' 接收的 POST data 片段 k: [' + chunk + '].'
61.     );
62.   });
```

```
63.
64.   request.on('end', () => {
65.     if (request.url in routingTable) {
66.       let obj = routingTable[request.url];
67.
68.       serve(response, obj.url, obj.mime);
69.     }
70.     else {
71.       console.log(' 未定義的存取: ' + request.url);
72.
73.       response.end();
74.     }
75.   });
76. }).listen(8088);
77.
78. // log message to Console
79. console.log(' 伺服器啓動，連線 url:  http://127.0.0.1:8088/');
80.
81. // index.js
```

1.4 問題與練習

修改 htdocs/index.html 使得目前版本的 index.js 還是可以正常顯示內容。