



使用 BlueBird



designed by freepik

Estimated time:

50 min.

資訊工業策進會 Institute for Information Industry

【Key Points】：

學習目標

- 17-1: 安裝設定 Bluebird
- 17-2: 使用依順序執行多個 SQL 敘述
- 17-3: Promise 錯誤處理



17-1

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

本模組的學習目標：

1. 了解 Promise 類型物件的使用方式。
2. 會安裝 Bluebird。
3. 能使用 Bluebird 在 fs (File System) 套件上。
4. 能使用 Bluebird 在 mysql 套件上，並在一個路由內依序執行數個 SQL。
5. 了解 Promise 的錯誤處理。

【Key Points】：

- 從 Bluebird 實作 Promise 的功能。
- 說明 Promise 的特性。
- 將 Promise 應用在 mysql 的 SQL 執行上。

17-1: 安裝設定 BlueBird

- Promise 的優點
- 為什麼選擇 Bluebird
- 套用在 fs 上
- 套用在 mysql 上



designed by freepik



designed by freepik

17-2

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

本知識點的內容涵蓋：

1. 了解 Promise 的優點。
2. 了解 Bluebird 的特色。
3. 能安裝 Bluebird。
4. 能使用 Bluebird 在 fs 套件上。
5. 能使用 Bluebird 在 mysql 套件上。

【Key Points】：

- 安裝及引用 Bluebird。
- 使用 `bluebird.promisifyAll()` 來包裝 fs 物件的方法。
- 使用 `bluebird.promisifyAll()` 來包裝 mysql 的連線物件。

Promise 的優點

- 改善回呼地獄 (Callback hell)
- 較佳的錯誤處理方式 (錯誤捕獲)
- 方便依序執行任務

```
new Promise((resolve, reject)=>{
    setTimeout(function(){
        resolve('Hello');
    }, Math.random()*1000);
})
.then(result=>{
    info.innerHTML += `${result} <br>`;
    return 'hi';
})
.then(a=>{
    info.innerHTML += `a: ${a} <br>`;
})
```

17-3



1. 改善回呼地獄 (Callback hell) · 程式碼變成扁平 (可讀性和維護性較高) 。
2. 較佳的錯誤處理方式 (錯誤捕捉) · 使用 `catch()` 方法呼叫 。
3. 方便依序執行任務 · 使用 `then()` 方法的鏈式呼叫 。
4. `Promise.all()` 並行執行多個非同步任務 · 全部執行後回呼 。
5. `Promise.race()` 並行多個競爭任務 · 取得最先返回的結果 。
6. 程式片斷演示了 `Promise` 使用方式 · `resolve()` 為非同步執行後取得結果 · 將結果傳至下一個 `then()` 內的 `callback function` 的參數 (範例中為 `result` 參數) 。第二個 `then()` 內的 `callback function` 會在第一個完成後才會被呼叫 。

【Key Points】：

- 1. 了解 `callback function hell` 難維護的問題 。
- 2. 熟悉 `Promise` 的使用方式 。
- 3. 可以使用之前教過的 `fetch()` 複習 `Promise` 類型物件的用法 。

為什麼選擇 Bluebird

- Bluebird 實作 Promise 的功能，且更易於使用
- 早期 Promise 效能較差
- 比原生 Promise 有更多功能特色
- 前後端皆可使用

安裝：
\$ npm install bluebird

引用：
const Promise = require("bluebird");
或：
const bluebird = require("bluebird");

17-4



1. Bluebird 官方網站：<http://bluebirdjs.com>。
2. Nodejs 8 之前的原生 Promise 效能較差，甚至到 12 的版本，Bluebird 的效能都比原生的要好。
3. Bluebird 的包裝器 `promisify()` 和 `promisifyAll()` 在搭配 `fs` 和 `mysql` 套件時更容易使用。
4. Bluebird 可以使用在前端的 JavaScript，也可以使用在後端的 Nodejs 上，尤其後端使用上更為普遍。
5. Bluebird 有比原生 Promise 更多的功能，尤其是集合操作的 `map()`, `each()` 等方法。

【Key Points】：

- Bluebird 實作 Promise 所有的標準。
- Bluebird 效能比原生 Promise 效能要好。
- Bluebird 功能比原生 Promise 要多。

使用在 fs 套件上

- **套用前**

```
const fs = require('fs');
fs.readFile(__dirname + '/sample.txt', (err, data) => {
  if (err) {
    console.error(err);
  } else {
    console.log(data);
  }
});
```

- **套用 Bluebird 後**

```
const Promise = require('bluebird'),
      fs = require('fs');
Promise.promisifyAll(fs);
fs.readFileAsync(__dirname + '/sample.txt')
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

17-5



1. FileSystem (fs) 的 readFile() 方法功能為讀取檔案，為非同步操作。其第一個參數為檔案路徑，第二個參數（最後一個）為 callback function (cb)。
2. cb 的第一個參數為錯誤訊息，第二個參數為讀取後的檔案內容資料。
3. 若沒有錯誤發生，程式中的 err 會得到 null。
4. 使用 promisifyAll() 會將物件內，非同步的方法，加入新的對應方法而包裝成 Promise 功能。
5. 以 readFile() 為例，經由 promisifyAll() 之後，增加了對應的 readFileAsync() 方法。除了 callback function 外，其它參數和順序是相同的。

【Key Points】：

以 readFile() 為例說明，一般非同步方法的用法。

Callback function 通常第一個參數為 error，若其值為 null 表示正常執行，沒有錯誤。
promisifyAll() 所增加的方法是對應到原有的非同步方法，其名稱在其結尾為 Async。

使用在 mysql 套件上

- 在連線物件上使用 `promisifyAll()` 包裝器

```
const mysql = require('mysql');
const bluebird = require('bluebird');
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'root',
    database: 'test',
});
db.on('error', (event)=>{
    console.log(event);
});
db.connect();
bluebird.promisifyAll(db);
```

17-6



1. 使用 `mysql` 物件的 `createConnection()` 方法建立 MySQL 的連線物件 `db`。
2. 建立連線時，要給 MySQL 所在的主機名稱、用戶帳號、用戶密碼和使用的資料庫。
3. `on('error')` 為 連線時的錯誤處理器。
4. 呼叫 `db.connect()` 後，才會進行資料庫連線。
5. `promisifyAll()` 主要是把 `db.query()` 另外包裝成 `db.queryAsync()` 方法。

【Key Points】：

- 建立連線物件時，注意主機、帳號、密碼和資料庫名稱都必須相符，才能連線成功。
- 為了方便除錯，`on('error')` 事件處理器請勿省略。
- 若 `db.connect()` 沒有出現錯誤，經過 `promisifyAll()` 即可正確包裝 `db` 物件。

17-2:依順序執行多個 SQL 敘述

- 執行兩個以上的 SQL (各別執行)
- 沒有使用 Promise (使用 callback function)
- 使用 Bluebird
- 確保執行順序



designed by freepik



designed by freepik

17-7

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

本知識點的內容涵蓋：

1. 了解要執行多個 SQL 時，不能各自獨立。
2. 能實作在不使用 Promise 的情況下，以 callback functions 來依序執行多個 SQL。
3. 了解 `promisify()` 和 `promisifyAll()` 的功能。
4. 能以 Bluebird 使用在 mysql 上。
5. 能使用 Bluebird 的輔助，依序執行多個 SQL。

【Key Points】：

- 使用 mysql 執行多個 SQL 時，不應該各自獨立。
- 不使用 Bluebird 時，如何使用 callback functions 依序執行 SQL。
- 使用 Bluebird 時，讓 mysql 依序執行 SQL。

執行兩個以上的 SQL

- 各別執行時（不好的寫法）

```
router.post('/my-route', (req, res)=>{
  const sql1 = "SELECT * FROM `address_book` WHERE `sid`=2";
  const sql2 = "SELECT * FROM `address_book` WHERE `mobile`='0918111222'";
  db.query(sql1, (error, result1)=>{
    // 取得第一個結果
  });
  db.query(sql2, (error, result2)=>{
    // 取得第二個結果
  });
});
```

17-8



1. 在一個路由內可以執行多個 SQL。
2. 如範例程式，若兩個 SQL 執行並不相關，將會有兩個回呼函式。
3. 無法確定哪個回呼函式先執行，哪個後執行。
4. 在回應給前端時，亦無法決定 `res.json()`（或其它回應方式）要放在哪裡。
5. 這種寫法雖然是合法的寫法，但是錯誤的方式。

【Key Points】：

- 在一個路由內可以執行多個 SQL。
- 每次的 SQL 執行都是非同步的。
- SQL 執行應該要有順序。

沒有使用 Promise

- 使用巢狀 callback functions

```
router.post('/my-route', (req, res)=>{
    const sql1 = "SELECT * FROM `address_book` WHERE `sid`=2";
    const sql2 = "SELECT * FROM `address_book` WHERE `mobile`='0918111222'";
    db.query(sql1, (error, result1)=>{
        // 取得第一個結果
        db.query(sql2, (error, result2)=>{
            // 取得第二個結果
        })
    });
});
```

17-9



1. 如範例程式，同樣在一個路由內執行兩個 SQL 敘述。
2. 在執行 SQL 的第一個 callback function，可以取得第一個結果，然後再執行第二個 SQL 敘述。
3. 如果在沒有發生錯誤的情況下，第二個 callback function 內，已經取得兩次的結果。
4. 這個範例碼雖然沒有使用 Promise，但依然可以確保兩個 SQL 的執行順序。
5. 當有多個 SQL 要執行時，將需要多次的程式縮排，容易發生不容易維護的 callback function hell 問題。

【Key Points】：

- 在一個路由內可以執行多個 SQL。
- 要依照順序執行多個 SQL 時，必須使用巢狀的 callback functions。
- 回應給前端時，必須在取得第二個結果之後。

使用 Bluebird

```
router.post('/my-route', (req, res)=>{
  const sql1 = "SELECT * FROM `address_book` WHERE `sid`=2";
  const sql2 = "SELECT * FROM `address_book` WHERE `mobile`='0918111222'";
  db.queryAsync(sql1)
    .then(result1=>{
      // 取得第一個結果
      return db.queryAsync(sql2);
    })
    .then(result2=>{
      // 取得第二個結果
    })
});
```

17-10



1. 經過 Bluebird 包裝過的連線物件，會有 `queryAsync()` 方法。
2. `queryAsync()` 回傳 `Promise` 物件，所以可以在 `then()` 的 `callback function` 取得 SQL 執行的結果。
3. 在第一個 `callback function` 中，回傳另一次的 `queryAsync()` 呼叫，可以在下一個 `then()` 的 `callback function` 取得結果。
4. 如此整個程式結構不會因為 `callback function` 層層包裹而往右縮排。
5. 程式結構是一個鏈狀的方法呼叫，左側的程式碼是對齊的。

【Key Points】：

- 連線物件 `db` 必須經由 Bluebird 的 `promisify()` 或 `promisifyAll()` 包裝過。
- SQL 執行的結果，可以使用外層變數（陣列或物件）存放。
- SQL 有使用 ? 時，要放入 SQL 的變數值可以放在 `queryAsync()` 的第二個參數（陣列）。

確保執行順序

```
router.post('/my-route', (req, res)=>{
  const sql1 = "SELECT * FROM `address_book` WHERE `sid`=2";
  const sql2 = "SELECT * FROM `address_book` WHERE `mobile`='0918111222'";
  const sql3 = "SELECT * FROM `address_book` WHERE `name` LIKE 'bill%'";
  db.queryAsync(sql1)
    .then(result1=>{
      // 取得第一個結果
      JS index.js > ...
      1   const express = ...
      2   const app = express();
      3
      4   app.get('/A*', function (re
      5     |   res.end(`name start wit
    } 6   });

```

17-11



1. 當有超過兩個 SQL 敘述要執行時，同樣使用 `then()` 鏈狀呼叫。
2. `callback function` 內都是回傳 `queryAsync()` 的回傳結果。
3. 如範例程式碼，要依序執行 SQL 就以 `queryAsync()` 的呼叫順序即可。
4. 所以 `sql1` 先執行，接著是 `sql2` 執行，最後是 `sql3` 執行。
5. 要將資料傳給前端時，在第後一個 `then()` 的 `callback function` 內，使用 `res.json()` 之類的方法回應。

【Key Points】：

- 要確保執行順序，可以如範例程式的做法。
- 若沒有順序的問題，可以使用 `Promise.all()`。
- 確保執行順序的另一種做法是使用 `Promise.each()`。

17-3: Promise 錯誤處理

- 使用 `catch()` 方法
- 不可使用 `try/catch` 敘述
- `catch()` 的位置
- 多個 `catch()`



designed by freepik



designed by freepik

17-12

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

本知識點的內容涵蓋：

1. 了解 `catch()` 的功能。
2. 了解 `try/catch` 敘述的使用時機。
3. 明白 `then()` 和 `catch()` 使用的時機。
4. 了解 `catch()` 常設置的位置
5. 知道使用多個 `catch()` 捕捉不同部位的 `rejection` 或 `Error` 物件。

【Key Points】：

- `try/catch/finally` 的使用時機。
- `Promise` 的 `catch()` 使用時機。
- 多個 `catch()` 用來捕捉不同部位的 `rejection`。

使用 catch() 方法

- **catch() 和 finally()**

```
const myFunc = (doReject=false, errorMsg='test')=>{
    return new Promise((resolve, reject)=>{
        setTimeout(function(){
            doReject ? reject(errorMsg) : resolve(Math.random()*100);
        }, 100);
    });
}

myFunc(true, 'hello')
    .then(r=>console.log('OK: ', r))
    .catch(err=>console.log('Error: ', err))
    .finally(()=>console.log('Finally'))
```

17-13



1. **Promise** 在使用時，通常會包裹成 **function**，範例程式碼中的 **myFunc** 就是典型的寫法，回傳一個 **Promise** 的實體（實例，**instance**）。
2. 為了模擬非同步的延遲，使用 **setTimeout()** 間隔時間觸發匿名函式。
3. **myFunc** 的第一個參數用來決定是否發生回決（例外），第二個參數為錯誤訊息（可以在呼叫時決定）。
4. 第二段呼叫 **myFunc** 時，讓它發生 **reject**。所以 **then()** 裡面的 **callback function** 不會被呼叫，而是 **catch()** 裡的被呼叫。
5. **finally()** 裡的 **callback function** 是一定會被呼叫，不論前面走的是 **then()** 或者 **catch()**。

【Key Points】：

- **then()**、**catch()** 和 **finally()** 的角色關係和 **try/catch/finally** 敘述類似。
- **Promise** 建立時傳入的 **callback function**，呼叫 **resolve()** 表示正常執行。
- 範例中的 **then()** 和 **catch()** 內的 **callback function** 視狀況擇一執行，而 **finally()** 是一定會執行。

不可使用 try/catch 敘述

```
try {
    myFunc(true, 'bad')
        .then(r => console.log('OK:', r));
} catch(ex){
    console.log('ex:', ex);
}
```

- 錯誤訊息

```
(node:1368) UnhandledPromiseRejectionWarning: bad
(node:1368) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error
originated either by throwing inside of an async function without a catch block, or by rejecting a
promise which was not handled with .catch(). (rejection id: 1)
(node:1368) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In
the future, promise rejections that are not handled will terminate the Node.js process with a non-
zero exit code.
```

17-14



1. 雖然 Promise 的 then() · catch() 和 finally() 的角色關係和 try/catch/finally 敘述類似，但兩者使用的時機不同。
2. try/catch/finally 只用在同步的程式片段。
3. Promise 的 then() · catch() 和 finally() 使用在非同步的情況。
4. 如範例程式，try/catch 敘述無法捕捉Promise 物件所丟出的 Rejection，只能使用 catch() 方法去處理。
5. 當沒有被捕捉的 Rejection 發生時，會丟出 UnhandledPromiseRejection 例外。

【Key Points】：

- 了解 try/catch/finally 的使用時機。
- 了解 Promise 的 then() · catch() 和 finally() 的 使用時機。
- try/catch 敘述用在 Promise 上，是無法捕捉 Exception 或 Error 的。

catch() 的位置

- Promise 發生 rejection

```
myFunc(true, 'step1')
  .then(r=>{
    console.log(r);
  })
  .catch(ex=>{
    console.log('Error: ', ex);
  });
});
```

- then() 內部發生 rejection

```
myFunc()
  .then(r=>{
    console.log(r);
    return myFunc(true, 'step2')
  })
  .catch(ex=>{
    console.log('Error: ', ex);
  });
});
```

17-15



- 一般使用 Promise 時，會把 then() 放在 catch() 的前面。
- Promise 發生 rejection 或錯誤時，錯誤會被 catch() 獲捉（如第一段程式碼）。
- then() 內部發生 rejection 時，同樣會被後面的 catch() 所補捉。
- catch() 也可以捕捉「throw new **Error**('bad~')」的狀況。
- 所以 catch() 通常放在最後面，用來處理前面所發生的任何狀況。

【Key Points】：

- 了解 then() 和 catch() 順序的關係。
- catch() 通常放在後面。
- catch() 可以捕捉 rejection，也可以捕捉 Error。

多個 catch()

```
myFunc()  
  .then(r=>{  
    console.log(r);  
    return myFunc(true, 'first');  
  })  
  .catch(ex=>{  
    console.log('Error 1: ', ex);  
  })  
  .then(r=>{  
    console.log(r);  
    throw new Error('Bad!!')  
  })  
  .catch(ex=>{  
    console.log('Error 2: ', ex);  
  })
```

- 執行結果

```
27.31091700315598  
Error 1: first  
undefined  
Error 2: Error: Bad!!
```

17-16



1. 範例中的 Promise 物件後面依序呼叫 then()、catch()、then() 和 catch()。
2. 第一個 then() 裡面的 callback function 會丟出 rejection，而被第一個 catch() 所捕捉。
3. 第二個 then() 會執行，但不會收到參數，r 為 undefined。
4. 第二個 then() 所丟出的 Error 物件會被後面的 catch() 捕捉。
5. 如果沒有第一個 catch() 的情況，則第二個 then() 的 callback function 不會執行，rejection 會被後面的 catch() 捕捉。

【Key Points】：

- then() 會依據順序執行。
- catch() 可以捕捉之前未被捕捉的 rejection 或 Error 物件。
- 一般 catch() 放置在最後面，用來處理所有的 rejection 和 Error 物件。

Summary 〈 精華回顧 〉

- 安裝 Bluebird
- 使用 Bluebird 的 `promisifyAll()` 包裝 `fs` 物件
- 使用 Bluebird 的 `promisifyAll()` 包裝 `mysql` 的連線物件
- `mysql` 的連線物件的 `queryAsync()` 的使用方式
- 使用 `queryAsync()` 依序執行多個 SQL 敘述。
- `Promise` 的除錯方式



17-17

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

回顧一下我們剛學到的內容：

1. 安裝 Bluebird
2. 使用 Bluebird 的 `promisifyAll()` 包裝 `fs` 物件
3. 使用 Bluebird 的 `promisifyAll()` 包裝 `mysql` 的連線物件
4. `mysql` 的連線物件的 `queryAsync()` 的使用方式
5. 使用 `queryAsync()` 依序執行多個 SQL 敘述。
6. `Promise` 的除錯方式

【Key Points】：

- 重點放在，如何引用 Bluebird 以包裝 `mysql` 連線物件。
- 依序執行多個 SQL 敘述。
- `Promise` 的除錯方式。

程式練習題

- **題目1:** 如何安裝及引用 Bluebird
- **題目2:** 承上，使用 Bluebird 的 `promisifyAll()` 包裝 mysql
- **題目3:** 使用 mysql 連線物件的 `queryAsync()` 方法，執行兩個以上的 SQL 敘述，並取得結果，以 JSON 格式傳給前端。

Estimated time:

20 min.

17-18



題目: 如何安裝及引用 Bluebird

內容說明:

使用 npm 安裝到 Node/Express 專案內

題目: 使用 Bluebird 的 `promisifyAll()` 包裝 mysql

內容說明:

使用 Bluebird 的 `promisifyAll()` 包裝 mysql 連線的物件。

若 mysql 連線的物件是放在模組內做匯出，請直接在模組內使用 Bluebird 。

題目: 使用 mysql 連線物件的 `queryAsync()` 方法，執行兩個以上的 SQL 敘述，並取得結果，以 JSON 格式傳給前端。

內容說明:

例如資料表 products

第一個 SQL 取得資料的總筆數。

第二個 SQL 取得第一個分頁的資料 6 筆（每個分頁 6 筆）。

【Key Points】：

如何安裝及引用 Bluebird

使用 Bluebird 的 `promisifyAll()` 包裝 mysql

使用 mysql 連線物件的 `queryAsync()` 方法，執行兩個以上的 SQL 敘述

課後練習題 (Lab)

- 使用 '/products/:page' 的路由，用來取得不同分頁的資料。
- 輸出的格式為 JSON。

```
{  
    "page": "目前頁次",  
    "perPage": "每頁幾筆，數值可固定 6",  
    "totalRows": "總筆數",  
    "totalPages": "總頁數",  
    "rows": "產品分頁的資料，陣列"  
}
```

Estimated time:

20 min.

17-19



財團法人資訊工業策進會

INSTITUTE FOR INFORMATION INDUSTRY

情節描述：

本 Lab 是假設在已經有 node/express 專案，並有安裝 mysql 套件，並有 MySQL 資料庫，亦有 products 資料表。

預設目標：

<http://localhost:3000/products/1>，可取得第一頁的產品資料，格式如範例。

【Key Points】：

使用變數代稱

如何讀取變數代稱的內容

以 email 格式練習Regular Expression

線上程式題

- **17-1 Promise 的常見句型**

請教 Promise 的常見句型，通常都怎麼寫？

- **17-2 引用 bluebird 處理資料庫**

想要使用 bluebird 模組協助處理資料庫，請問程式該怎麼寫呢？

- **17-3 bluebird 如何確保 SQL 執行順序？**

已使用 bluebird 模組協助處理資料庫，請問 bluebird 如何確保 SQL 執行順序？

17-20



題目名稱: 17-1 Promise 的常見句型

內容說明:

請教 Promise 的常見句型，通常都怎麼寫？

題目名稱: 17-2 引用 bluebird 處理資料庫

內容說明:

想要使用 bluebird 模組協助處理資料庫，請問程式該怎麼寫呢？

題目名稱: 17-3 bluebird 如何確保 SQL 執行順序？

內容說明:

已使用 bluebird 模組協助處理資料庫，請問 bluebird 如何確保 SQL 執行順序？

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】：

17-1 Promise 的常見句型

17-2 引用 bluebird 處理資料庫

17-3 bluebird 如何確保 SQL 執行順序？

課後練習題(Lab)

- **情節描述:**
本Lab 假設環境已安裝MySQL (或MariaDB) 資料庫，並已經有Node.js/Express套件模組，可透過mysql套件連線資料庫並讀取資料。接下來，學員要接著安裝Bluebird套件，並達成分頁資料API的建立。
- **預設目標:**
利用Bluebird完成產品資料分頁API。路由使用“/products/:page?”，產生指定格式的JSON資料。
- **Lab01: 安裝Bluebird模組**
- **Lab02: 在index.js內建立有功能的路由**
- **完成後的程式與檔案，請參考 Example 資料夾的內容**

Estimated time:
20 minutes

17-21



【情節描述】

本Lab 假設環境已安裝MySQL (或MariaDB) 資料庫，並已經有Node.js/Express套件模組，可透過mysql套件連線資料庫並讀取資料。接下來，學員要接著安裝Bluebird套件，並達成分頁資料API的建立。

【預設目標】

利用Bluebird完成產品資料分頁API。路由使用“/products/:page?”，取得的JSON格式如下：

```
{  
  "page": "目前頁數",  
  "perPage": "每頁幾筆，數值可固定 6",  
  "totalRows": "總筆數",  
  "totalPages": "總頁數",  
  "rows": "產品分頁的資料，陣列"  
}
```

Lab01: 安裝Bluebird模組

Lab02: 在index.js內建立有功能的路由

完成後的程式與檔案，請參考 Example 資料夾的內容。

【Key Points】：

Lab01: 安裝Bluebird模組

Lab02: 在index.js內建立有功能的路由

bluebird.promisifyAll(db);

範例程式使用說明

- 範例程式資料夾: Module_17_example
- 使用步驟:
 1. 安裝 Node.js
 2. 安裝 Visual Studio Code
 3. 安裝 XAMPP, 執行 products.sql 指令檔
 4. 以 Visual Studio Code 開啟本模組的範例資料夾
 5. 在 Visual Studio Code 按下「Ctrl + 滑鼠」開啟 terminal 終端機視窗。
 6. 在終端機視窗，輸入下列指令，安裝必要的模組套件:
npm install
 7. 在終端機視窗，輸入 npm start
 8. 啟動瀏覽器，連接 http://localhost:3000/products/1

17-22



使用步驟:

1. 安裝 Node.js (<https://nodejs.org/en/>)
2. 安裝 Visual Studio Code (<https://code.visualstudio.com/>)
3. 安裝 XAMPP (https://www.apachefriends.org/zh_tw/index.html)
4. 在檔案總管點兩下c:\xampp\xampp-control.exe，啟動 XAMPP Control Panel
5. 點按 Apache 與MySQL的「Start」按鈕，啟動兩套伺服器
6. 點按MySQL那列的「Admin」按鈕，啟動phpMyAdmin 管理程式，
7. 先點一下 test 資料庫，再切換到 SQL 頁籤
8. 複製貼入 products.sql 內容到SQL 頁籤，然後按下「執行」按鈕。
9. 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」
10. 在 Visual Studio Code 按下「Ctrl + 滑鼠」開啟 terminal 終端機視窗。
11. 在終端機視窗，輸入下列指令，安裝必要的模組套件: npm install
12. 在終端機視窗，輸入 npm start
13. 啟動瀏覽器，連接 http://localhost:3000/products 或 http://localhost:3000/products/1

【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」開啟 terminal 終端機視窗，輸入: 「node 主程式.js」