

# JavaScript | WebSocket 讓前後端沒有距離



神Q超人 · [Follow](#)

Published in Enjoy life enjoy coding · 8 min read · Feb 23, 2019



2.4K



9



## 前言

最近因為工作的關係接觸到 WebSocket ， WebSocket 是網路協定的一種， Client 可以透過此協定與 Server 做溝通，而他和一般 http 或 https 不同的是， WebSocket 協定只需透過一次連結便能保持連線，不必再透過一直發送 Request 來與 Server 互動！

## WebSocket

如前言所說，WebSocket 只需開啟連結便能和 Server 做溝通，且 websocket 傳送資料的方式是雙向的，Client 端可以像 Ajax 一樣做請求，Server 端也能主動發送 Client 所需要的資料。

一般的 WebSocket 請求網址會長這個樣子：

```
ws://example.com
```

```
//經過 SSL 加密後，前方的 ws 會變成 wss  
wss://example.com
```

### Server 端 - 搭建 WebSocket 環境

WebSocket 的 Server 部分，本文會以 Node.js 建置，如果電腦上還沒有安裝 Node.js 可以先參考「[\[筆記\]\[node.js\]第一次建置node.js開發環境和安裝npm就上手！](#)」。

處理好安裝環境後還需要下載兩個套件，分別是用來開發 Web 框架的 express 和負責處理 WebSocket 協定的 ws：

```
npm install express  
npm install ws
```

安裝完後可以到專案中的 `package.json` 中確認是否成功：

```
1  {  
2    "name": "websocketserver",  
3    "version": "1.0.0",  
4    "description": "",  
5    "main": "server.js",  
6    "dependencies": {  
7      "express": "^4.16.4",  
8      "ws": "^6.1.4"
```

Medium

Search

Write

Sign up

Sign in



```
11    "scripts": {  
12      "test": "echo \"Error: no test specified\" && exit 1"  
13    },  
14    "author": "",  
15    "license": "ISC"  
16  }
```

下載完後的套件會被記錄到 `package.json` 中

接著在專案裡新增一個 JavaScript 檔案 `server.js` 當作專案的進入點：

```
1 //import express 和 ws 套件
2 const express = require('express')
3 const SocketServer = require('ws').Server
4
5 //指定開啟的 port
6 const PORT = 3000
7
8 //創建 express 的物件，並綁定及監聽 3000 port，且設定開啟後在 console 中提示
9 const server = express()
10   .listen(PORT, () => console.log(`Listening on ${PORT}`))
11
12 //將 express 交給 SocketServer 開啟 WebSocket 的服務
13 const wss = new SocketServer({ server })
14
15 //當 WebSocket 從外部連結時執行
16 wss.on('connection', ws => {
17
18   //連結時執行此 console 提示
19   console.log('Client connected')
20
21   //當 WebSocket 的連線關閉時執行
22   ws.on('close', () => {
23     console.log('Close connected')
24   })
25 })
```

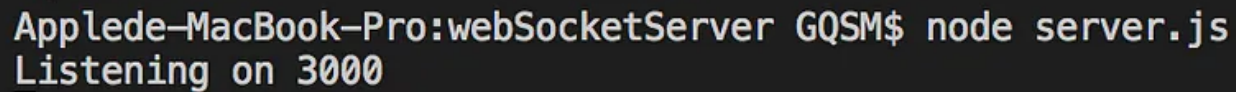
WebSocket server(1).js hosted with ❤ by GitHub

[view raw](#)

沒問題後便可以輸入以下指令執行 `server.js`：

```
node server.js
```

當本機 Server 的指定 Port 被打開時，會先執行我們監聽指定的事件：



```
Applede-MacBook-Pro:websocketServer GQSM$ node server.js  
Listening on 3000
```

開啟成功後會在 console 中打印提示

## Client 端 - 連接 WebSocket Server

Server 端處理完後，就換到 Client 端來連結剛剛開啟的 WebSocket 服務，這裡另外建一個專案，在專案裡只需一個 index.html 及 index.js：

index.html 的部分先簡單處理，只需引用 index.js 就可以了：

```
<html>  
  <body>  
    <script src='./index.js'></script>  
  </body>  
</html>
```

index.js 的部分會用來處理與 WebSocket 的連結：



上方的 url 為剛剛使用 node.js 在本機上執行的 Server，另外的 onopen 及 onclose 分別為他們指定一個 Function，在開啟和關閉連線時執行，執行結果：

```
open connection
> ws
< ▾ WebSocket {url: "ws://localhost:3000/", readyState: 1, bufferedAmount: 0, onopen: f, onerror: null, ...} ⓘ
  binaryType: "blob"
  bufferedAmount: 0
  extensions: ""
  ▶ onclose: () => { console.log('close connection') }
  onerror: null
  onmessage: null
  ▶ onopen: () => { console.log('open connection') }
  protocol: ""
  readyState: 1
  url: "ws://localhost:3000/"
  ▶ __proto__: WebSocket
```

開啟連結後會執行 onopen 的事件

執行結果中可以看到 onopen 中在 console 打印的提示，除此之外，也可以從剛剛執行 Server 的地方觀察開啟連結後的訊息：

```
Applede-MacBook-Pro:websocketServer GQSM$ node server.js
Listening on 3000
Client connected
```

在 Server 上執行時記錄的訊息



上方也列出 WebSocket 的物件有哪些屬性，比較重要的還有 `onmessage`，**Client** 就是靠它在接收由 **Server** 發送的資料，但在提到它之前，得先回到 **Server** 了解如何和 **Client** 做溝通。

## 回到 **Server** 端 - 處理接收發送訊息

提到溝通，過程一定是有來有往，在開啟 `WebSocket` 後，**Server** 端會使用 `send` 發送訊息，接收則是如同在 `connection` 內監聽 `close` 一樣，只是換成對 `message` 設定監聽，並接收一個參數 `data`，捕獲 **Client** 端發送的訊息：

在 WebSocket 的連結開啟後，設定 message 的監聽

## 回到 **Client** 端 - 處理接收發送訊息

剛剛處理完 Server ，要換回 Client 端使用 `onmessage` 處理接收及 `send` 送出訊息：

為 `onmessage` 指定要執行的函示，接收一個參數 `event` 為 Server 發送的物件

onmessage 指定的函式中多了一個參數 event，裡面會有這次溝通的詳細訊息，從 Server 回傳的資料會在 event 的 data 屬性中。

但是上方的程式碼還沒有增加在 Client 中發送訊息的 send，因此下方在連接到 WebSocket 後直接在 console 中發送，並確認回傳訊息：

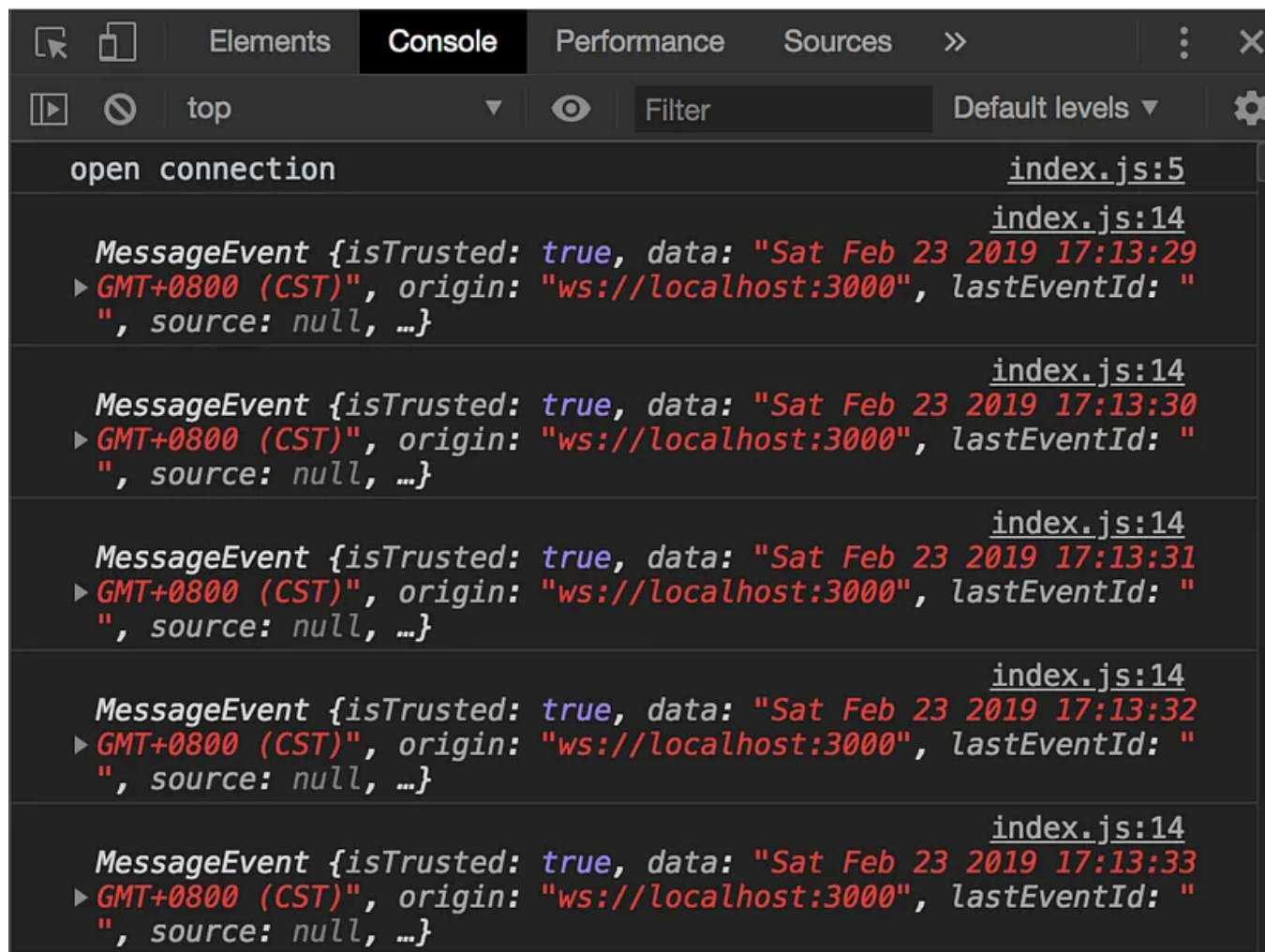
```
> ws.send('Hello')
< undefined
  ▶ MessageEvent {isTrusted: true, data: "Hello", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} index.js:14
> ws.send('I am GQSM')
< undefined
  ▶ MessageEvent {isTrusted: true, data: "I am GQSM", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} index.js:14
    bubbles: false
    cancelBubble: false
    cancelable: false
    composed: false
    ▶ currentTarget: WebSocket {url: "ws://localhost:3000/", readyState: 1, bufferedAmount: 0, onopen: f, onerror: null, ...}
      data: "I am GQSM"
      defaultPrevented: false
      eventPhase: 0
      isTrusted: true
      lastEventId: ""
      origin: "ws://localhost:3000"
      path: []
      ports: []
      returnValue: true
      source: null
      ▶ srcElement: WebSocket {url: "ws://localhost:3000/", readyState: 1, bufferedAmount: 0, onopen: f, onerror: null, ...}
      ▶ target: WebSocket {url: "ws://localhost:3000/", readyState: 1, bufferedAmount: 0, onopen: f, onerror: null, ...}
      timeStamp: 42721.504999964964
      type: "message"
      userActivation: null
      ▶ __proto__: MessageEvent
```

Server 接收 Client 發送的訊息後再回傳

不過上面看起來還是以 Client 做 send 發送訊息給 Server 處理過才得到回傳資料，該怎麼從 Server 上直接發送呢？很簡單，只需要透過 setInterval 就能讓 Server 在固定時間發送資料給 Client，例如下方的例子：

使用 `setInterval` 不斷發送最新訊息給 Client 端

Client 連結後的結果如下：



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays a series of messages received from a WebSocket server. The messages are as follows:

- open connection** (index.js:5)
- MessageEvent** {isTrusted: true, data: "Sat Feb 23 2019 17:13:29 GMT+0800 (CST)", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} (index.js:14)
- MessageEvent** {isTrusted: true, data: "Sat Feb 23 2019 17:13:30 GMT+0800 (CST)", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} (index.js:14)
- MessageEvent** {isTrusted: true, data: "Sat Feb 23 2019 17:13:31 GMT+0800 (CST)", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} (index.js:14)
- MessageEvent** {isTrusted: true, data: "Sat Feb 23 2019 17:13:32 GMT+0800 (CST)", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} (index.js:14)
- MessageEvent** {isTrusted: true, data: "Sat Feb 23 2019 17:13:33 GMT+0800 (CST)", origin: "ws://localhost:3000", lastEventId: "", source: null, ...} (index.js:14)

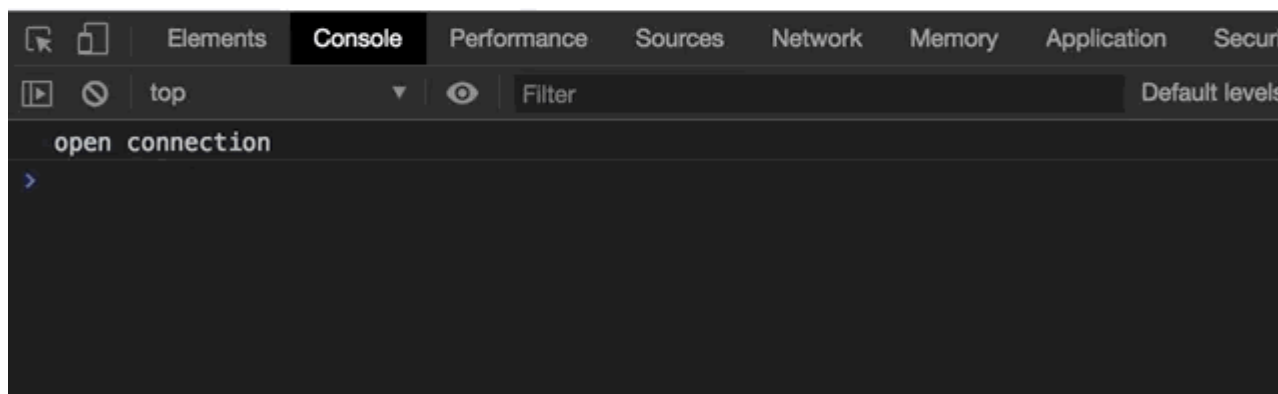
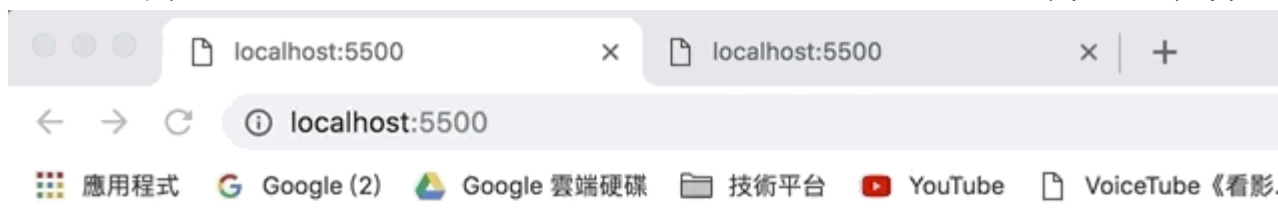
不斷接收 Server 主動傳遞的訊息

最後一次回到 **Server** - 多人連線

通常 WebSocket 都會運用在聊天室，但是就剛剛的使用方式來說，今天 ClientA 和 ClientB 都連結同一個 Server，而他們各自在 Client 使用 `send` 發送資料給 Server，在這個情況下 Server 只會依據兩個 Client 各自發送的内容，再分別回傳給 ClientA 和 ClientB，並無法讓 ClientB 能夠在 ClientA 發送訊息時也收到回傳的資料。

例如以下例子，用兩個視窗開啟 Client 並各自發送請求：





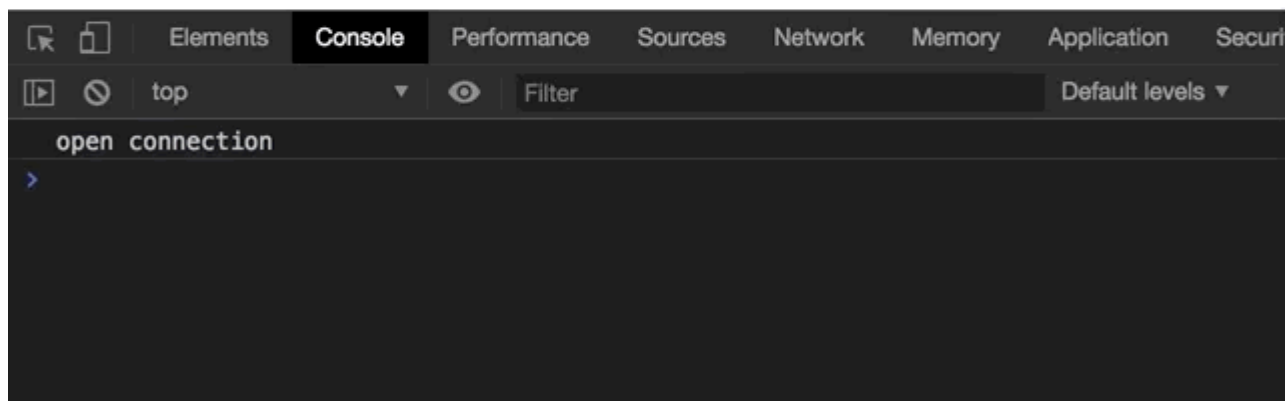
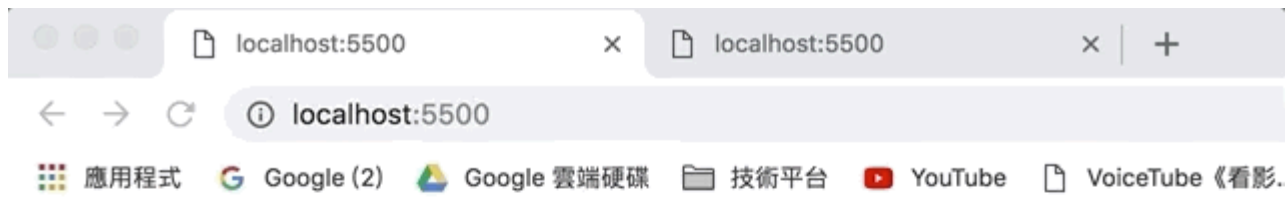
客戶端無法共享 Server 發送的訊息

那該怎麼像廣播一樣，當我在某一個 Client 發送訊息時，讓 Server 告知所有其他同時連接中的 Client 都知道我對 Server 發送這個訊息，也同時接收到 Server 回傳的資料呢？

答案就在一開始下載的套件 `ws` 中，它可以使用 `clients` 找出當前所有連結中的 Client 資訊，並透過迴圈將訊息發送至每一個 Client 中：

使用 `clients` 取得所有連接中的 `client`

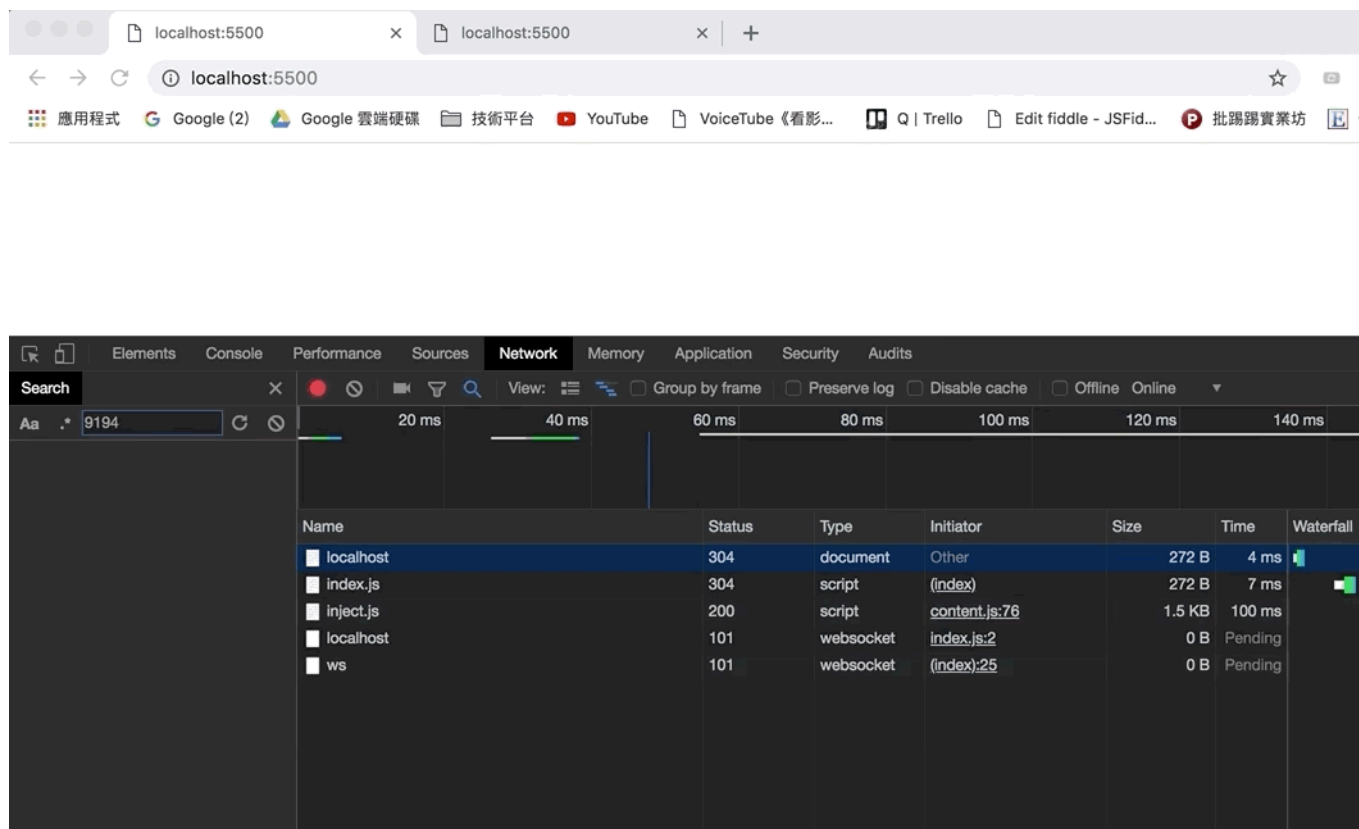
這麼一來，不論是哪個 Client 端發送訊息，Server 都會將訊息回覆給所有連接中的 Client：



## 補充內容

雖然在 WebSocket 的協定上 Client 和 Server 不需再通過 Request，因此在開發人員工具中的 Network 中就看不到 Request 的資料，但是取而代之的是，那

些傳遞過程可以透過第一次要求連接時的 Request 中觀察：



觀察在 WebSocket 協定的傳輸方式

關於 WebSocket 從 Client 或是 Server 在 send 資料時，除了字串外還可以使用 USVString 、 ArrayBuffer 、 Blod 和 ArrayBufferView 等型態（這部分感謝 Hank Hsiao 留言提醒）。

另外，要傳送 JSON 的資料的時，記得在 send 中做 JSON.stringify，接收到時再用 JSON.parse 轉成物件處理即可！

如果想找個伺服器部署 WebSocket，可以參考「[Heroku | 搭配 Git 在 Heroku 上部署網站的手把手教學](#)」

本文介紹了 WebSocket 的基本使用方式及一些例子，希望能夠減少各位研究的時間，這幾天還會接著研究如何搭配 React。

如果文章中有任何問題，或是不理解的地方，都可以留言告訴我！謝謝大家！

### 參考文章

1. <https://devcenter.heroku.com/articles/node-websockets>
2. <http://www.ruanyifeng.com/blog/2017/05/websocket.html>

JavaScript

Websocket



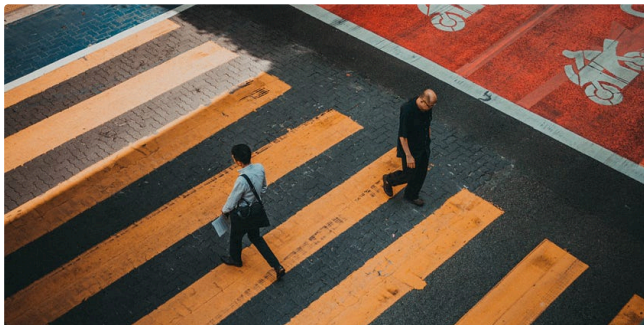
Written by 神Q超人

2.5K Followers · Editor for Enjoy life enjoy coding

82 年次 · 單純相信努力不會騙人

Follow

More from 神Q超人 and Enjoy life enjoy coding





神Q超人 in Starbugs Weekly 星巴哥技術專欄

## Git | 我以為的 Git Rebase 與和 Git Merge 做合併分支的差異

Hi！大家好，我是神Q超人！最近和朋友聊天的時候遇到了這個問題：「欸，你知道 Git ...

Apr 26, 2022



813



5



```
{
  (name) {
    = name;
  }

  hello(name) {
    !!(name)!;
  }

  _age !== undefined) {
    `${this.name} age is ${this._age}.`;
  }

  on't know ${this.name}'s age.';
}

{
  = age;
}

21   getFrom() {
22     const state = 'Taiwan';
23     return `${this.name} from ${state}.`;
24   }
25 }
26
27   class Employee extends Person {
28     constructor(name, position) {
29       super(name);
30       this.position = position;
31     }
32   }
33
34   getPosition() {
35     return `${this.name}'s position is the ${this.p
36   }
37
38   superCallGetForm() {
39     return super.getFrom();
40   }
41 }
```



神Q超人 in Enjoy life enjoy coding

## JavaScript | ES6 中最容易誤會的語法糖 Class - 基本用法

這些差別都是取決於物件導向是基於 Class 或 Prototype，因此就算 ES6 新增了一個 Class...

May 26, 2019



1.3K



4



神Q超人 in Enjoy life enjoy coding

## CSS | 所以我說那個版能不能好切一點？ - Grid 基本用法

Grid 也是新增在 display 的新屬性，在排版上的應用類似於表格，可以為一整個區塊創建 ...

Apr 21, 2019



122



1



```
{
  Status = (status: number): void => {
    tus.success:
    請求成功！');
    tus.error:
    請求失敗！');
    or('No have status code!'));
  }

  21   interface responseContent {
  22     status: number,
  23   }
  24
  25   const submitFetch = (responseStatusCod
  26     fetch('http://httpbin.org/status/$fr
  27     method: 'POST',
  28   }).then((response: responseContent)
  29     handleResponseStatus(response.stat
  30   ));
  31
  32
  33   // 請求成功！
  34   submitFetch('200');
  35
  36   // 請求失敗！
  37   submitFetch('400');
```



神Q超人 in Enjoy life enjoy coding

## TypeScript | 善用 Enum 提高程式的可讀性 - 基本用法 feat. JavaScript

Enum 是在 TypeScript 中增加的新語法，也被稱做「列舉」或「枚舉」，實務面會用它來...

Jul 7, 2019



940



3



See all from 神Q超人


See all from Enjoy life enjoy coding

---

## Recommended from Medium

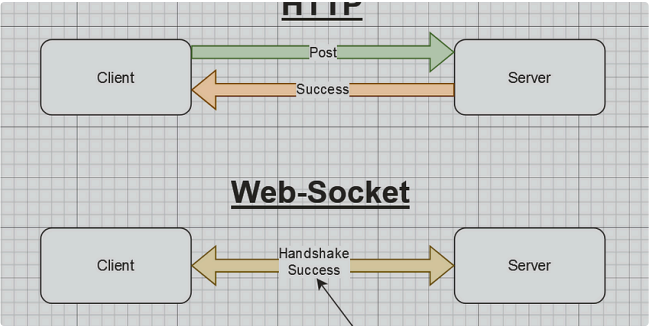




 Romulo Gatto

## Introduction to WebSocket Programming with Node.js

🌟 May 22 🖱️ 3 💬 1 



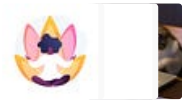
 Schangm

## Options For Building Real-time Apps

Normally when we establish a client-to-server connection, the most common usage would...

Aug 26 🖱️ 4 

### Lists



#### Stories to Help You Grow as a Software Developer

19 stories · 1398 saves



#### Generative AI Recommended Reading

52 stories · 1417 saves



#### General Coding Knowledge

20 stories · 1626 saves



#### Visual Storytellers Playlist

61 stories · 505 saves




 TEJESWAR REDDY

## A Comprehensive Guide to Server-Sent Events (SSE) for Real-Time...


 Sep 29  6  1 



 Yunuscan Bartık

## WEB SOCKET

Web socket is a computer communication protocol that provides a two-way...

Sep 5  52 




 Koby

## Understanding WebSockets: A Comprehensive Guide

Introduction



 Waqar Amin

## WebSocket Programming? How to Approach Real-time socket...

May 30  1



If you are interested in knowing about web sockets you probably know REST Api's....

Sep 8  50



See more recommendations