Socket.IO Programming using Swift for iOS/OSX Environment (2)

Socket.IO with iOS Swift Client

一. 實驗目的

了解基本的 Socket 架構，利用 Socket.IO 實作簡單的 JavaScript Socket Server 及 iOS Swift Socket Client。

二. Socket簡介

1. Socket
   - Socket 為兩個互相溝通的程序(process)連結的點
   - 一個連線中，兩個程序一邊各有一個Socket
   - Socket 利用 IP, Port pair 來判斷要怎麼傳輸
   - 網路應用程式設計者不需處理網路下層(傳輸層、網路層)的工作，而專注於其本身(應用層)的程式設計
   - 兩端的 Socket 一旦建立好連線，即是點對點傳輸，與水管概念相似

2. Server side
   - 監聽一個 Port
   - 當有人連線到這個 port 時，創立一個對應於這個 client 的 socket，所有對於這個 client 的傳輸都經由這個 socket 傳輸
   - Server 只能透過 Socket 與其對應的 Client 傳輸

3. Client side
   - 創立一個用於連線到 Server 的 Socket
   - 設定 Socket 要連線到的 Server 之 IP 和 Port
   - 利用 Socket 連線後，所有對於這個 Server 的傳輸還是透過此 Socket 傳送/接收

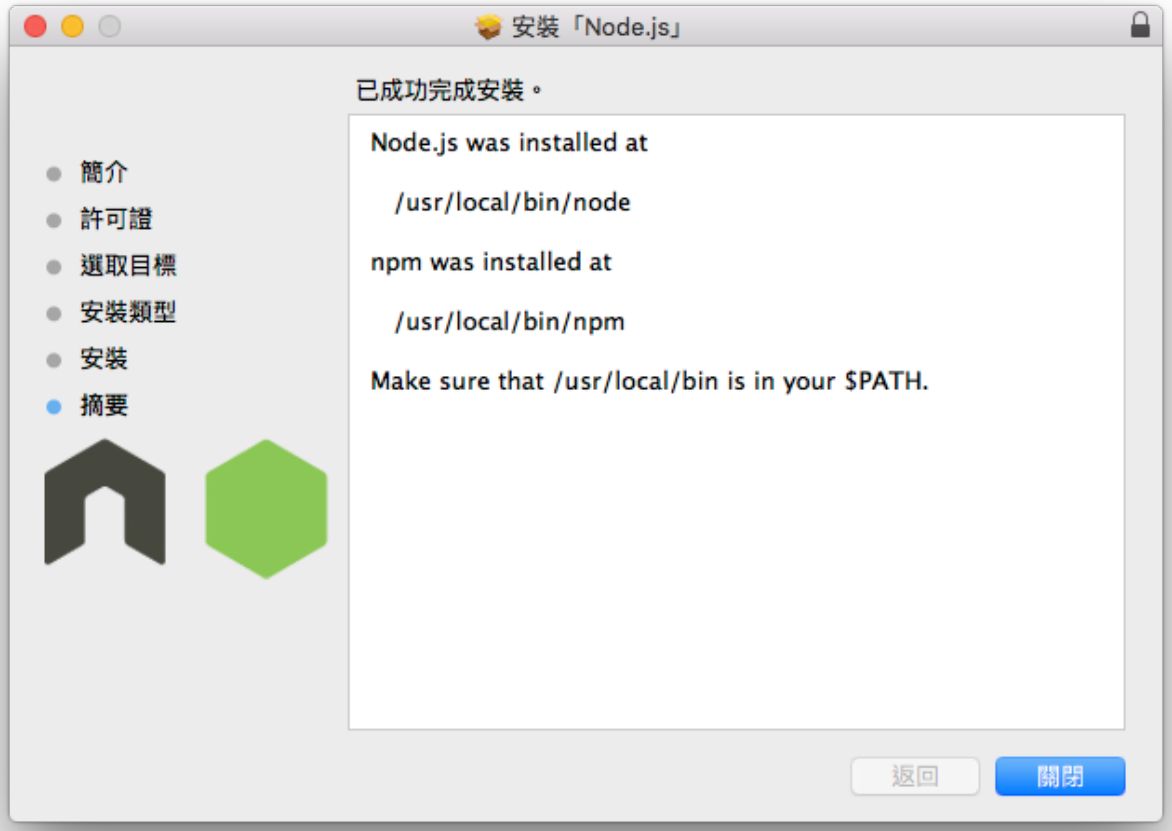
4. Lab 架構
   - 此次Lab所練習的Socket.IO實作，並非常見的BSD Socket，而是利用 HTML 5 中 WebSocket 為基礎的建立的可實時雙向溝通傳輸API。比起傳統的BSD Socket，概念相同，但在設定的流程上更為簡化，支援的平台也較廣泛。
   - Socket.IO 之 Server 端為利用 Node.js 架設的 JavaScript web server

三. 實驗步驟

1. 安裝Node.js

(1) 至Node.js官網或e3平台下載 Node.js 安裝檔

(2) 打開安裝檔，點選 繼續 -> 同意 -> 安裝 直到出現安裝完成介面

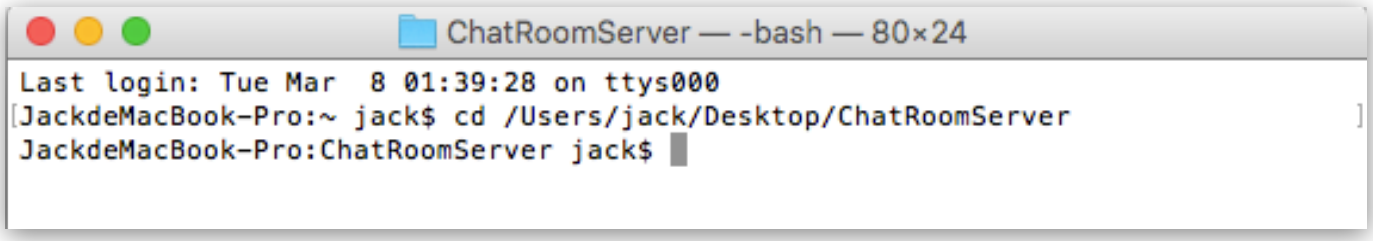


2. 架設 Socket.io Server

(1) 將e3上提供之 ChatRoomServer.zip 下載至桌面，解壓縮出 ChatRoomServer 資料夾

(2) 從應用程式資料夾或利用 ctrl+space 找出 終端機(terminal.app) 程式，執行

(3) 利用 cd 指令，將目錄移動到 ChatRoomServer 資料夾

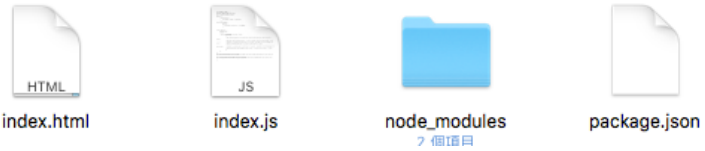


- Tips: 路徑獲得方式可在 ChatRoomServer 資料夾內任意檔案按右鍵->簡介，從彈出視窗的 “位置” 欄位反白->右鍵->複製，到終端機貼上

(4) 依序輸入以下指令

```
npm install --save express@4.10.2
npm install --save socket.io
```

(5) 若指令執行成功，ChatRoomServer 資料夾中會多出一個 node_modules 資料夾，且裡面會有 express 及 socket.io 兩個子資料夾

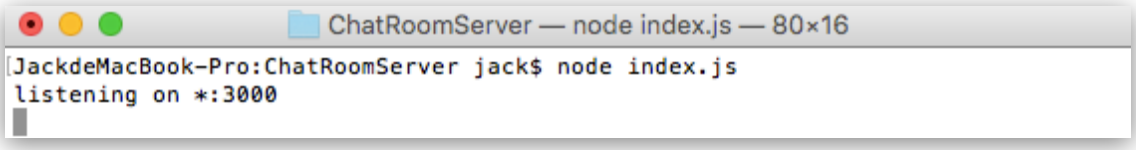


3. 執行 Socket.IO Server

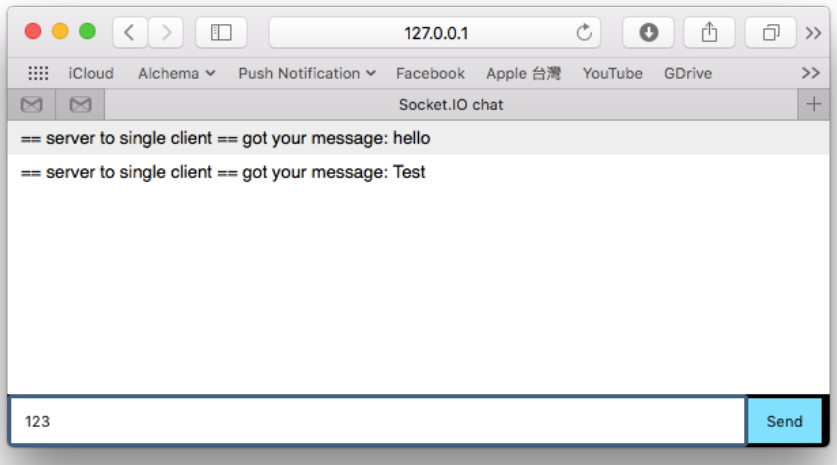
(1) 終端機目錄在 ChatRoomServer 資料夾內時，輸入

```
node index.js
```

可看到 “ listening on *:3000 ” 字樣，表示 Server 執行成功。



(2) 從螢幕下方 Dock 開啟 Safari 瀏覽器，網址輸入 “127.0.0.1:3000” 或 “localhost:3000”，即可用網頁版 client 測試 Socket.IO Server

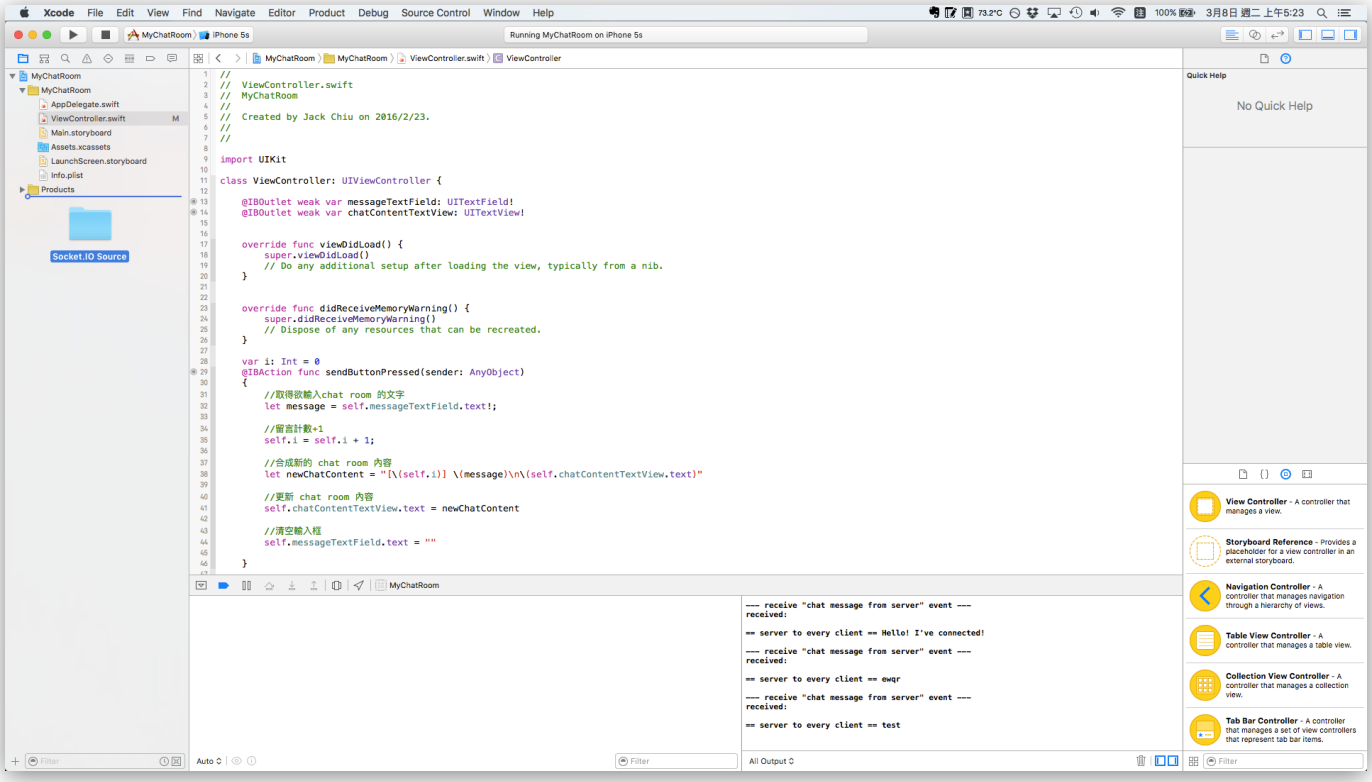


4. 將 MyChatRoom 專案 加入Socket.IO Library

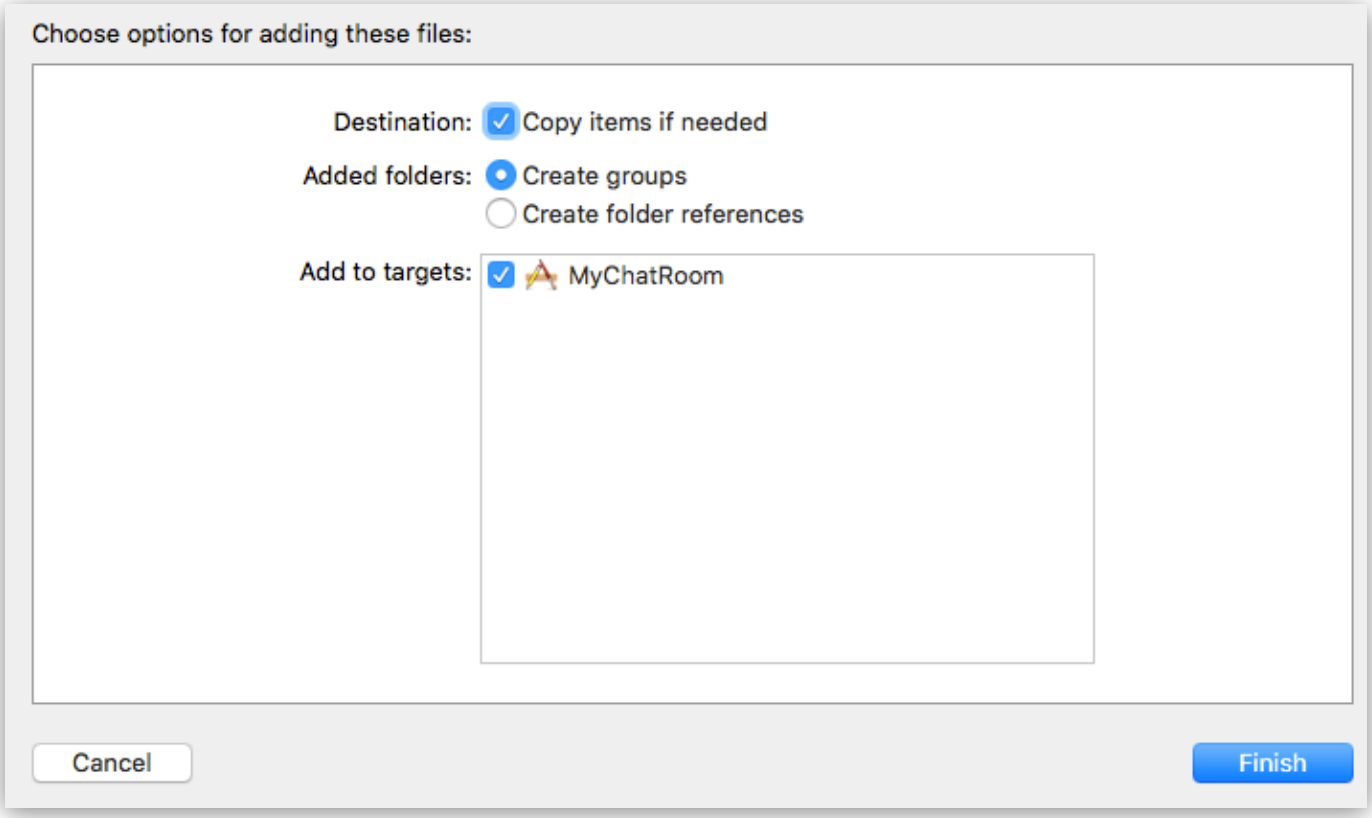
(1) 下載e3上提供之 Socket.IO Source.zip 解壓縮出 Socket.IO Source 資料夾

(2) 開啟 Lab 1 的 MyChatRoom 專案檔 MyChatRoom.xcodeproj

(3) 將 Socket.IO Source 資料夾拉入專案左方 Project Navigator 中



(4) 勾選 Copy items if needed ，確認 Add to targets: 中的 MyChatRoom 打勾

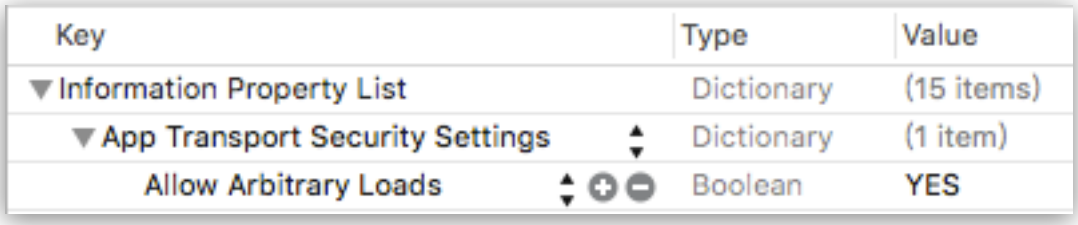


5. 設定App允許對外連線

(1) 在左方 Project Navigator 中找到 Info.plist

(2) 在 Information Property List 左方點選 ＋ ，在其下層加入新的 key：
“App Transport Security Settings”

(3) 在 “App Transport Security Settings” 下方加入新的 key：
“Allow Arbitrary Loads”
並將其 Value 改成 YES



6. 加入 Socket.IO 程式碼

(1) 從左方 Project Navigator 中點選 ViewController.swift ， 編輯 ViewController Class

(2) 在 ViewController Class 中加入以下兩個 property

```
        let hostString: String = "http://localhost:3000"
        var socket : SocketIOClient? = nil
```

(3) 再加入以下三個 method

```
func anyEventCallBack( anyEvent: SocketAnyEvent)
{
    //列印出所有收到的event跟event附帶的data，debug時可用
    //print("--- Got event: \(anyEvent.event), with items: \(anyEvent.items) ---")
}

func connectCallBack( data:[AnyObject], ack:SocketAckEmitter)
{
    print("--- socket connected ---")

    //利用socket傳送 event + message 給server
    self.socket!.emit("chat message from client", "Hello! I've connected!")
}

func serverMsgCallBack( data:[AnyObject], ack:SocketAckEmitter)
{
    print("--- receive \"chat message from server\" event ---")

    //找出message string
    let message: String = (data[0] as! String)

    print("received:\n\n" + "\(message)" + "\n")
}
```

(4) 在iOS中，一個介面 (ViewController) 被創造好後，在螢幕還沒顯示前，會自動呼叫 viewDidLoad( ) 這個method，可將此介面出現時需要先設定的參數或預先執行的動作放在這裡。我們在 viewDidLoad( ) method 中，加入以下設定Socket.IO的程式碼：

```
//創造連線用的URL物件
let hostUrl = NSURL(string: self.hostString)

//創造SocketIOClient函數，同時設定socket將連線到的URL
self.socket = SocketIOClient(socketURL: hostUrl!)

//把anyEventCallBack這個function設定成socket中所有event的handler，告訴socket接到所有的event時都要call anyEventCallBack這個函數
self.socket!.onAny(anyEventCallBack)

//把connectCallBack這個functio 設定成socket中 "connect" event的handler，告訴socket接到"connect" event時要call connectCallBack這個函數
self.socket!.on("connect",callback: connectCallBack)

//把serverMsgCallBack這個function設定成 "chat message from server" event的handler，告訴socket接到 "chat message from server" event時要call gCallBack這個函數
self.socket!.on("chat message from server", callback: serverMsgCallBack)

print("--- connecting to \(self.hostString) —")

//叫socket連線到前面已指定的Server
self.socket!.connect()
```

(5) 結果如下：

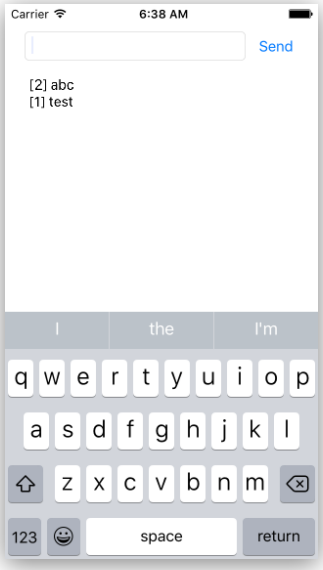
```
let hostString: String = "http://localhost:3000"
var socket : SocketIOClient? = nil

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.

    //創造連線用的URL物件
    let hostUrl = NSURL(string: self.hostString)

    //創造SocketIOClient函數，同時設定socket將連線到的URL
    self.socket = SocketIOClient(socketURL: hostUrl!)

    //把anyEventCallBack這個function設定成socket中所有event的handler，告訴socket接到所有的e
    self.socket!.onAny(anyEventCallBack)

    //把connectCallBack這個functio 設定成socket中 "connect" event的handler，告訴socket接到
    self.socket!.on("connect",callback: connectCallBack)

    //把serverMsgCallBack這個function設定成 "chat message from server" event的handler，'
    self.socket!.on("chat message from server", callback: serverMsgCallBack)

    print("--- connecting to \(self.hostString) —")

    //叫socket連線到前面已指定的Server
    self.socket!.connect()

}

func anyEventCallBack( anyEvent: SocketAnyEvent)
{
    //列印出所有收到的event跟event附帶的data，debug時可用
    //print("--- Got event: \(anyEvent.event), with items: \(anyEvent.items) ---")
}

func connectCallBack( data:[AnyObject], ack:SocketAckEmitter)
{
    print("--- socket connected ---")

    //利用socket傳送 event + message 給server
    self.socket!.emit("chat message from client", "Hello! I've connected!")
}

func serverMsgCallBack( data:[AnyObject], ack:SocketAckEmitter)
{
    print("--- receive \"chat message from server\" event ---")

    //找出message string
    let message: String = (data[0] as! String)

    print("received:\n\n" + "\(message)" + "\n")
}
```

(6) 執行專案，若看到 Console 有以下結果，表示 Swift 上的 Socket.IO Client 連線 Server 成功

```
--- connecting to http://localhost:3000 ---
--- socket connected ---
--- receive "chat message from server" event ---
received:

== server to single client == got your message: Hello! I've connected!
```

7. 將 Socket.IO 與 GUI 介面結合

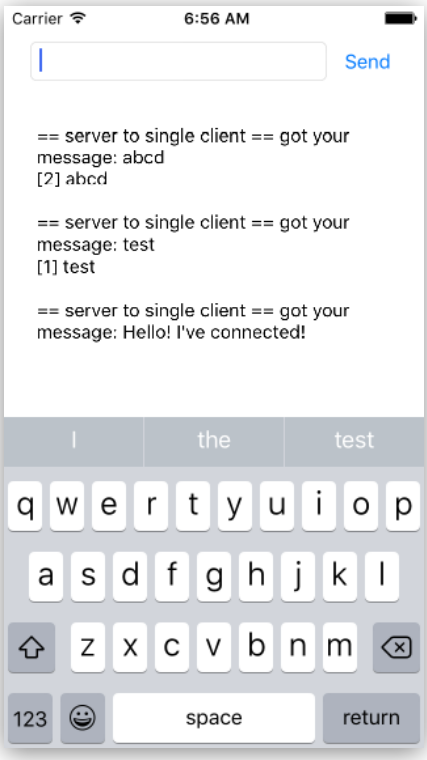
(1) 我們可利用 self.socket!.emit(event string , message) 來傳送 message 到 server，請試著自行增加一行程式碼，讓 MyChatRoom 程式介面上的 Text Field 文字，再按下 Send 按鈕後傳送到 Server 端。成功後 Console 可得到類似以下結果：




```
--- connecting to http://localhost:3000 ---
--- socket connected ---
--- receive "chat message from server" event ---
received:

== server to single client == got your message: Hello! I've connected!

--- receive "chat message from server" event ---
received:

== server to single client == got your message: test

--- receive "chat message from server" event ---
received:

== server to single client == got your message: abc
```

All Output

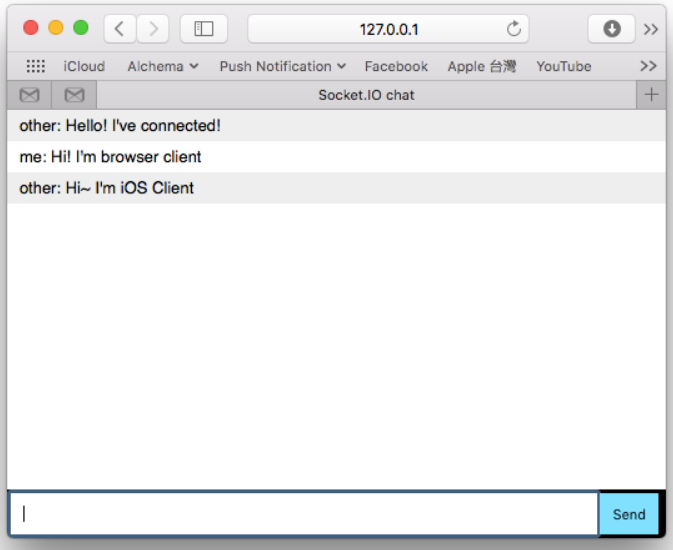
(2) 如 viewDidLoad 函數裡程式碼的註解所寫，Client 接到 Server 傳送的 “chat message from server” 這種 event 的時後，會呼叫 serverMsgCallBack( ) 這個函數。請試著自行增加一行程式碼，讓 Server 回傳回來的訊息可以顯示在介面上的 Text View 裡面。成功後App介面可得到類似以下結果：

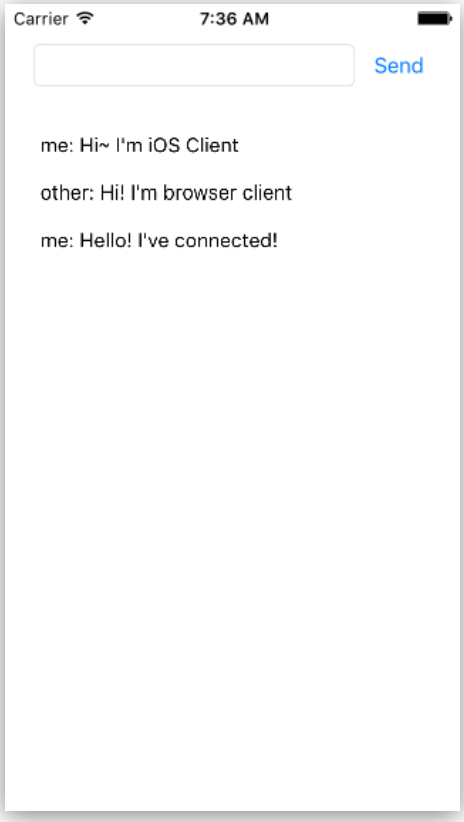


8. Client -> Server -> Client 溝通

(1) 現在 Client 只能與 Server 單獨對話。需要做到 Client 與 Client 聊天，需要讓 Server 幫忙把 message 傳遞給其他也連上線的 Client。此時需要修改 Server 的程式碼。

(2) 打開 ChatRoomServer 資料夾，在 index.js 檔案上按右鍵->打開檔案的應用程式 -> Xcode.app。詳細閱讀程式碼和註解，試著修改 Server 端的 JavaScript 程式碼，讓 client 之間的訊息可以互相傳遞，並且標示出自己傳的訊息(me)及其他人傳的訊息(other)。Server 程式碼修改完後，會需要重新執行，請至 終端機.app 利用 ctrl+c 結束 server 程式，再利用 “node index.js” 指令重新啟動 server。成功後App可與網頁互傳訊息：





Tips:

- `Server 端所用的語言 JavaScript 與 Swift 的語法很像，幾乎可以互通`
- `注意 event string 的內容，有註冊call back 的event，Server/Clinet端在收到這個event時才會去執行相對應的call back function。`
- `可把Lab1 sendButtonPressed( )裡面寫的動作刪掉`

四. Demo

請 Demo 實驗步驟 7. 與 8. 的執行結果。