Socket.IO Programming using Swift for iOS/OSX  Environment

**結報**

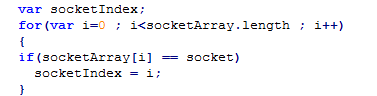
**0310795 黃昱銘**

1. 在 Socket.io 的資料傳輸中，程式要如何分辨不同目的的資料傳入，並針對不同目的的資料傳入做出不同的動作（不同 function call）？

使用server內socket.on(event,function)這個函式，就可以從event分辨不同目的的資料傳入，並執行相對於這個事件的function(動作)，例如：基本的傳送訊息目的

1. 請問在你的聊天室client-server架構裡：
   1. 你如何讓client端收到的訊息裡顯示傳送者名稱？

在server中，我們將每個加入的client加入一個socketArray，當我們想知道資料是從哪個client傳進來時，只要使用一個for迴圈掃整個socketArray判斷哪一個等於現在呼叫函式的socket。以下舉例：



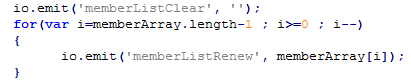
然後再創建一個memberArray，在socket connect時，即把其名稱存入memberArray內相對socketIndex的位置。接著透過以下在chat message from client內的函式，廣播給除了自己的人，在訊息前加入自己的member name



* 1. 你如何讓member list保持在最新的狀態？

我讓每次有人加入或退出時，就發送兩個命令給所有socket，一個是清空原client 內的memberTextView，另一個則是一個一個member加入新的List。

Server端：



* 1. 呈上題，除了你用得方法之外，有沒有其他做法？

我一開始的想法是直接加入或刪除位於memberTextView內相對index的文字，然而由於對TextView的使用方法不熟悉，不知道如何插入或指定刪除，因此就換了目前的方法。

* 1. 呈上題，所有做法中(包含你程式裡寫的)請選出你覺得最好的做法，並說明理由

應該是我一開始的想法會比較快比較省時間。因為我現在用的方法是每次只要有人加入或離開，就把整個list重傳給所有client，用想的就知道比較耗時跟效能，但卻也很容易就寫出來；而我一開始的想法則是只需要傳哪個index有變動，就可以更新名單。雖然client端比較複雜，然而卻可以省下傳輸的頻寬。

1. 心得與建議

這個階段的課程真的相當有趣，很可惜的是只有三次的課程可以讓我接觸IOS，如果有機會真的很想用自己的雙手打造一個可以給很多人用的APP，覺得可以專門開一門寫手機APP的課程，像JAVA一樣弄一個自己發想的期末專題，感覺可以看到很多有趣的APP問世。自己也很想再深入接觸swift語言。

1. 請貼上Project的程式碼 (將程式貼上並加上註解)

index.js(server端)：

var portNumber = 3000;

var app = require('express')();

var http = require('http').Server(app);

var io = require('socket.io')(http);

var socketArray = [];

var memberArray = [];

//設定從網頁連進Server時，回傳index.html給瀏覽器 (瀏覽器端為Client，index.html裡面有client的code)

app.get('/',

function(req, res)

{

res.sendFile(\_\_dirname + '/index.html');

});

//設定 Server 監聽 3000 這個 port

http.listen(portNumber,

function()

{

console.log('listening on \*:' + portNumber);

});

//可將 io 視為 Server 上管理所有 Socket 的 Manager

io.on('connection',

function(socket)

{

//將新創造的 Socket加到 socketArray 裡

socketArray[socketArray.length] = socket;

//在 server 的 console 中輸出現有的 socket 數

console.log('socket count: '+socketArray.length);

//加入新member，存名字在server端，刷新所有client名單

socket.on('addmember',

function(msg)

{

//存新member在memberArray最底端

memberArray[memberArray.length] = msg;

//顯示在console上

console.log(msg + " has join")

//清空所有socket的memberTextView

io.emit('memberListClear', '');

//將新List一行一行刷新給所有socket，先進的在list上面

for(var i=memberArray.length-1 ; i>=0 ; i--)

{

io.emit('memberListRenew', memberArray[i]);

}

});

//處理從client傳送的公開訊息

socket.on('chat message from client',

function(msg)

{

//只回傳給當前的socket自己輸入的訊息

socket.emit('chat message from server', 'me:'+ msg)

//找尋當前傳訊者socket的index

var socketIndex;

for(var i=0 ; i<socketArray.length ; i++)

{

if(socketArray[i] == socket)

socketIndex = i;

}

//傳訊息給除了當前socket的其他人(包括發信者名字)

socket.broadcast.emit('chat message from server', memberArray[socketIndex]+":"+ msg);

});

//處理從client傳送的私人訊息(加分題一)

socket.on('chat message to private',

function(msg)

{

//將msg以","分裂成splitArray(自訂格式為：”id,message”)

var splitArray = msg.split(",");

//找尋當前傳訊者socket的index

var socketIndex;

for(var i=0 ; i<socketArray.length ; i++)

{

if(socketArray[i] == socket)

socketIndex = i;

}

//宣告index為id，並從字串轉integer

var index = parseInt( splitArray[0] ,10);

//回傳給當前傳訊者(包含目標名字)

socket.emit('chat message from server', '(private to)'+ memberArray[index-1] +':'+ splitArray[1] );

//傳送給目標(包含傳訊者名字)

socketArray[index-1].emit('chat message from server', '(private)'+ memberArray[socketIndex]+":"+splitArray[1] );

});

//處理當使用者disconnect

socket.on('disconnect',

function()

{

//尋找要斷線的 socket 在 socketArray 中的 index

var socketIndex;

for(var i=0 ; i<socketArray.length ; i++)

{

if(socketArray[i] == socket)

socketIndex = i;

}

//清空所有client memberTextView(一定要在斷線前)

io.emit('memberListClear', '');

//從 socketArray 和memberArray中把斷線的 socket

socketArray.splice(socketIndex, 1);

memberArray.splice(socketIndex, 1);

//刷新所有client memberTextView

for(var i=memberArray.length-1 ; i>=0 ; i--)

{

io.emit('memberListRenew', memberArray[i]);

}

//在 server 的 console 中輸出現有的 socket 數

console.log('socket count: '+socketArray.length);

});

});

ViewController.swift(client端)

import UIKit

class ViewController: UIViewController {

let hostString: String = "http://localhost:3000"

var socket : SocketIOClient? = nil

var name : String = "" //儲存本client 名字

//連接所有物件給storyboard

@IBOutlet weak var messageTextField: UITextField!

@IBOutlet weak var indexTextField: UITextField!

@IBOutlet weak var nameTextField: UITextField!

@IBOutlet weak var memberTextView: UITextView!

@IBOutlet weak var chatContentTextView: UITextView!

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view, typically from a nib.

//創造連線 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function 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CallBack這個函數

self.socket!.on("chat message from server", callback: serverMsgCallBack)

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

//把clearListCallBack這個function設定成 "memberListClear" event的handler,告訴socket接到 " memberListClear " event時要CallBack這個函數

self.socket!.on("memberListClear",callback: clearListCallBack)

//叫socket連線到前 已指定的Server

self.socket!.connect()

self.socket!.on("memberListRenew",callback: renewListCallBack)

}

override func didReceiveMemoryWarning() {

super.didReceiveMemoryWarning()

// Dispose of any resources that can be recreated.

}

//設定sendButton 的Press事件處理方法

@IBAction func sendButtonPressed(sender: AnyObject) {

let message = self.messageTextField.text!

let index = self.indexTextField.text!

//若使用者沒有輸入index，則發送全頻訊息

if (index == ""){

self.socket!.emit( "chat message from client" , message )

}

//若有，則發送私訊給指定index

else{

self.socket!.emit( "chat message to private" , index+","+message)

}

self.messageTextField.text = "" //清空輸入訊息欄位

}

func anyEventCallBack( anyEvent: SocketAnyEvent)

{

// 印出所有收到的event跟event附帶的data,debug時可

//print("--- Got event: \(anyEvent.event), with items: \(anyEvent.items) ---")

}

@IBAction func connectPressed(sender: UIButton) {

self.socket!.connect()

}

@IBAction func disconnectPressed(sender: UIButton) {

self.socket!.disconnect()

let newChatContent = ""

self.memberTextView.text = newChatContent //清空自己的memberList

}

//connect一成功即觸發

func connectCallBack( data:[AnyObject], ack:SocketAckEmitter)

{

print("--- socket connected ---")

name = self.nameTextField.text!

self.socket!.emit("addmember", name) //傳送自己的名字給server

// 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)

//刷新訊息欄

let newChatContent = "\(message)\n\(self.chatContentTextView.text)"

self.chatContentTextView.text = newChatContent

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

}

func clearListCallBack(data:[AnyObject], ack:SocketAckEmitter)

{

//清空memberTextView

let newChatContent = ""

self.memberTextView.text = newChatContent

}

func renewListCallBack(data:[AnyObject], ack:SocketAckEmitter)

{

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

var newChatContent : String

//若收到的renew name 為自己的name 則在顯示時加上>>(加分題二)

if (name == message){

newChatContent = ">>\(message)\n\(self.memberTextView.text)"

}

else{

newChatContent = "\(message)\n\(self.memberTextView.text)"

}

self.memberTextView.text = newChatContent

}

}