Socket.IO Programming using Swift for iOS/OSX  Environment

**結報**

**0410835 鄭余哲**

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）？

Socket.on 的一個參數就是event的string，第二個參數是event handler要執行的function。

Socket.emit的的一個參數也是event的string，第二個參數是要傳輸的資料。

所以在socket程式中，只要在兩端設定好相同的event的string，在sender設定好傳輸的資料，reciever寫好對應的handler function。

如此一來就可以對不同目的的資料做出不同的動作，基於event的string來分辨。

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？

我在server端先查好傳送者的名稱，再將傳送者的名稱傳送給所有的client

如此一來，傳輸者一傳輸出訊息之後，server會先在SocketnameArray查詢該傳輸者名稱，在由broadcast傳送給每個client，client端就會顯示傳送者的名稱

* 1. 你如何讓member list保持在最新的狀態？

在新的socket連線時，更新member list，並在connectcallback中向server要求更新member list，server端收到要求後，broadcast給所有client，如此一來，所有client都會知道有新的client加入聊天室

在有socket要求斷線時，更新member list並broadcast給剩下的所有client，如此一來，剩下的client都會有最新的member list

* 1. 呈上題，除了你用得方法之外，有沒有其他做法？
     1. 當有新的socket連線或socket要求斷線時，僅更新server端的member list，而定時由server端傳輸給所有client member list。
     2. 當有新的socket連線或socket要求斷線時，僅更新server端的member list，而定時由client端向server端要求更新 member list
     3. 由新的連線socket或即將斷線的socket內先更新好member list，再將更新完的member list 傳輸給server，在由server broadcast 給剩餘的socket
  2. 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

我認為我原本的方法比較好，在實作上比較直覺，更新資訊也比較即時。如果用定時更新的方法可能就無法及時獲得所有人的資訊。而方法三又不夠直覺，如果時做起來可能很容易寫錯。

1. 心得與建議

因為大部分的人使用的電腦仍是windows，所以使用macOS仍是卡卡的，我認為多一點時間在教學簡單的快捷鍵或是使用操作說明，將會大幅提升大家coding 的效率。

不過助教人超好，願意讓我一個問題一個問題問，我覺得上課的互動感覺很棒。

我覺得打swift的機會難得，所以我很珍惜每次的上機機會。不過虛擬機開啟的速度真的時常令人崩潰。不過這堂課整體來說算蠻好玩的。期望往後能對自己的程式開發能有所幫助。

1. 請貼上Project的程式碼 (將程式貼上並加上註解)

Client端:

//

// ViewController.swift

// MyChatRoom

//

// Created by ESD 24 on 2018/11/23.

//

//lab２新增

import SocketIO

import UIKit

class ViewController: UIViewController {

@IBOutlet weak var messageTextField: UITextField! // 要傳送訊息的地方

@IBOutlet weak var chatContentTextView: UITextView! //聊天室內容顯示的地方

@IBOutlet weak var memberlist: UITextView! //顯示聊天室成員的地方

var i = 0

@IBOutlet weak var nameTextField: UITextField! //connect時的名稱或是私訊的對象

//lab2新增

let hostString: String = "http://localhost:3000" //連線server端的位置

var socket : SocketIOClient? = nil

var manager: SocketManager? = nil

var name = ""

override func viewDidLoad() {

super.viewDidLoad()

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

//lab2 修改

//創造連線 的URL物件

let hostUrl = URL(string: self.hostString)

//創造Socket Manager，同時設定socket將連線到的URL

self.manager = SocketManager(socketURL: hostUrl!)

//從Socket Manager取得SocketIOClient物件

self.socket = self.manager?.defaultSocket

//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 "show message on screen" event的handler，告訴 socket接到 "show message on screen" event時要call serverMsgCallBack這個函數

self.socket!.on("show message on screen", callback: serverMsgCallBack)

self.socket!.on("show\_name",callback: showNameCallBack)

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

}

override func didReceiveMemoryWarning() {

super.didReceiveMemoryWarning()

// Dispose of any resources that can be recreated.

}

@IBAction func ConnectButton(\_ sender: AnyObject) {

//叫socket連線到前 已指定的Server

self.socket!.connect()

}//連線用的button

@IBAction func DisconnectButton(\_ sender: AnyObject) {

self.socket!.disconnect()

self.memberlist.text=""

}//斷線用的button

@IBAction func sendButtonPressed(\_ sender: AnyObject) {

let message = self.messageTextField.text! //傳輸訊息內容

self.i = self.i + 1

//let newChatContent = "[\(self.i)] \(message)\n\(self.chatContentTextView.text!)"

self.socket!.emit("user send out message", message)

//self.chatContentTextView.text = newChatContent

self.messageTextField.text = ""

}//傳輸訊息用的button

@IBAction func PersonalSend(\_ sender: AnyObject) {

let person = self.nameTextField.text!//私訊對象

let message = self.messageTextField.text!//私訊內容

self.socket!.emit("personal message",person+"::"+message)//傳給server

self.messageTextField.text = ""

}//私訊用的button

//lab2新增function

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) ---")

}//任意事件的callback function

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

{

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

self.name=self.nameTextField.text!

//因為剛連上，Delay 0.5秒，再傳送message，讓server有時間註冊好event

let deadLine = DispatchTime.now() + .milliseconds(500)

DispatchQueue.main.asyncAfter(deadline: deadLine) {

// socket傳送 event + message 給server

//self.socket!.emit("user send out message", "Hello! I've connected!")

self.socket!.emit("send\_name", self.name)

}

}//連線時的callback function

func serverMsgCallBack( data:[Any], ack:SocketAckEmitter)

{

print("--- receive \"show message on screen\" event ---")

//找出message string

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

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

self.chatContentTextView.text = "\(message)\n\(self.chatContentTextView.text!)"

}//收到server端的內容並印出來

func showNameCallBack( data:[Any], ack:SocketAckEmitter)

{

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

let newString = message.replacingOccurrences(of: self.name, with: self.name+"<--")

self.memberlist.text = "\(newString)"

}//更新member list

}

Server端：

var portNumber = 3000;//server 採用的port

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

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

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

var socketArray = [];

var nameArray = [];

//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) /\*[1]\*/

{

socket.on('user send out message', /\*[2]\*/

function(msg)

{

var socketIndex;

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

if(socketArray[i] == socket)

socketIndex = i;

} //only send message back to the client of this socket ( with event string 'show message on screen' )

//socket.emit('show message on screen', '== server got your message == : '+ msg);

socket.emit('show message on screen', nameArray[socketIndex]+" : "+ msg); //broadcast to everyone include the client of this socket ( with event string 'event string' ), you can define 'event string' your self

//io.emit('event string', '== server to every client == ' + msg);

//broadcast to everyone except for the client of this socket ( with event string 'event string' ), you can define 'event string' your self

//socket.broadcast.emit('show message on screen', '== server to other client == ' + msg);

socket.broadcast.emit('show message on screen', nameArray[socketIndex]+" : " + msg); });

socket.on('personal message',

function(msg)//控制私訊

{

var splitArray = msg.split("::");//以::切割訊息

var sendIndex;

var receiveIndex=-1;

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

if(socketArray[i] == socket)

sendIndex = i;

}//尋找傳輸者

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

if(nameArray[i] == splitArray[0])

receiveIndex = i;

}//尋找接收者

socket.emit('show message on screen', nameArray[sendIndex]+" : "+splitArray[1]);//發送給傳輸者

if(receiveIndex!=-1){

socketArray[receiveIndex].emit('show message on screen', nameArray[sendIndex]+" : "+splitArray[1]);

}//發送給接收者

});

socket.on('send\_name',function(name){//更新member list

nameArray[nameArray.length] = name;

var nameString = nameArray.join('\n');

io.emit('show\_name',nameString);//給所有人更新member list

});

socketArray[socketArray.length] = socket;//每當新的連線者 增加socketArray 內容

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

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;

}

//從socketArray 中把斷線的socket移除

socketArray.splice(socketIndex,1);

nameArray.splice(socketIndex,1);

var nameString = nameArray.join('\n');

io.emit('show\_name',nameString);

//在server的console中輸出現有的socket數量

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

});

});

/\*

[1] when a new clinet connect to server, server will deploy a new socket to handle the connection to this client and call this function with the new created socket as parameter.

[2] tells socket to handle 'chat message from clinet' event. when socket get 'chat message frome clinet' event, server will call following function and the event message as the parameter

\*/