Socket

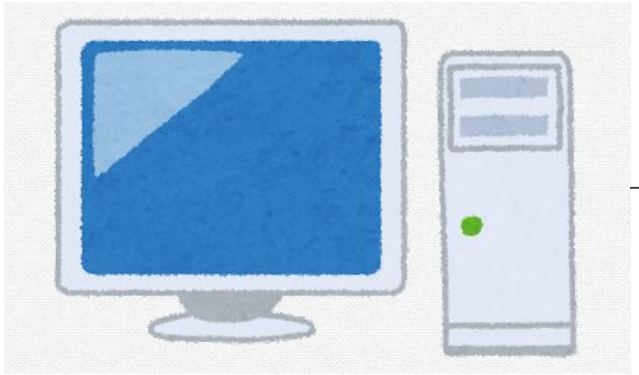
大綱

為什麼要學網路傳輸
名詞解釋
開始寫第一個TCP程式
黏包分解
開始寫第一個UPD程式
多個TCP連線的處理

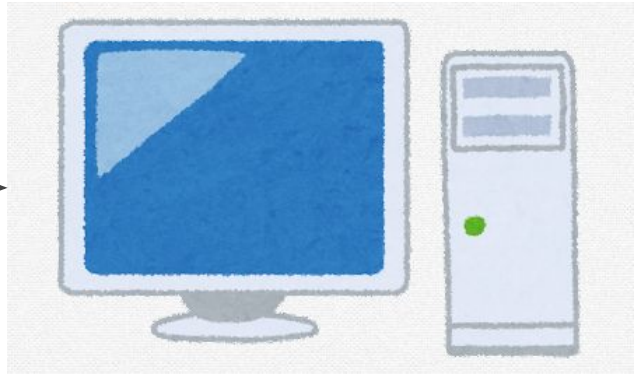
為什麼要學網路傳輸

工廠有很多設備

視覺檢測設備



回報



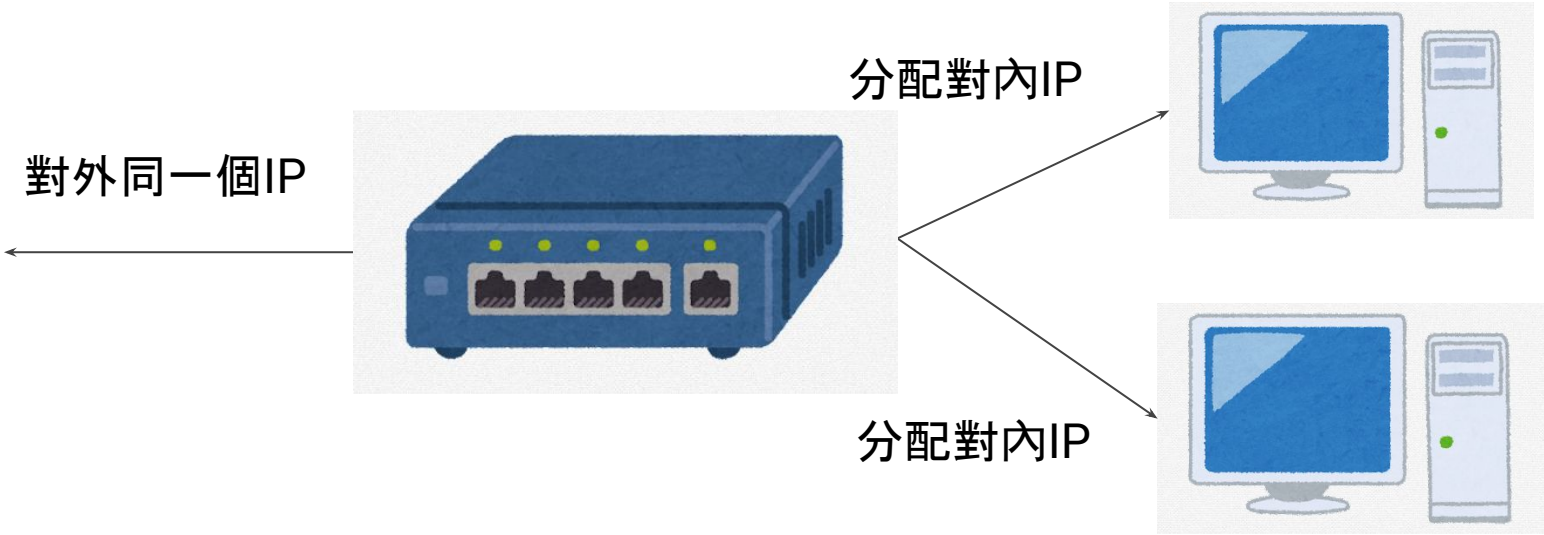
中控設備

檢測到問題
影像處理

統計&執行命令

常見的名詞介紹 IP

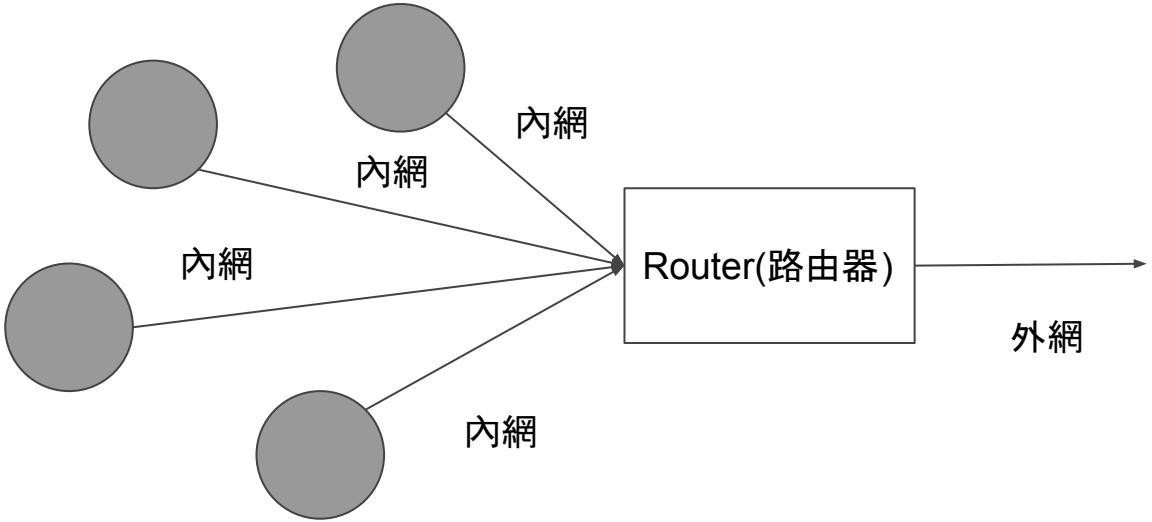
IP 就是每一台電腦的地址
但是世界上太多裝置，所以地址已經幾乎沒了



IP 的小知識

通常網路被分為內部網路和外部網路

- 同一群內部網路的對外IP都一樣
- IP為四個區域a.b.c.d 其中 a b c d 為 0~255
- 內部網路的IP通常是192.168.xxx.xxx
- 設備互聯，要輸入內網對方的IP



IP 的小知識 - 子遮罩

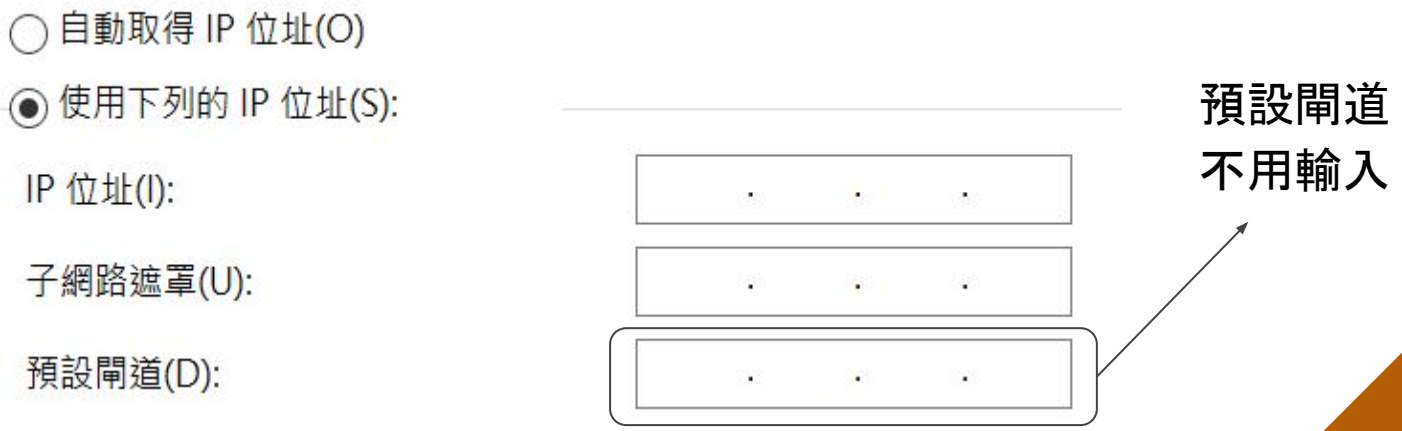
ip a.b.c.d, submask 255.255.0.0 表示只有ip a.b開頭的電腦才能連線

可以連線	不能連線
IP: 192.168.11.1 IP: 192.168.10.2	IP: 192.168.11.1 IP: 192.168.10.2
子遮罩 255.255.0.0	子遮罩 255.255.255.0

Router自動設定的子遮罩 + IP位置

```
連線特定 DNS 尾碼 . . . . . :
連結-本機 IPv6 位址 . . . . . : fe80::892f:4158:53e8:880b%10
IPv4 位址 . . . . . : 192.168.56.1
子網路遮罩 . . . . . .: 255.255.255.0
預設閘道 . . . . . .:
```

也可自行設定子遮罩 + IP位置



IP 的小知識

- 看自己的IP可以用ifconfig or ipconfig
- 127.0.0.1 本機的IP
- 要讓別人連線近來，記得**防火牆要關起來**

很容易忘記，然後就無法連線

常見的名詞介紹 - port

一台電腦會有很多個port(港口)可以串連
在寫TCP的時候必須要設定port的數值

就像從商船從美國到台灣，必須要告訴商船停在哪一個港口

client server架構

	Client	Server
IP	輸入Server的IP	輸入Server的IP
可否拒絕連線	否	是
概念	去連線對方	等待被連線
常用於	選端的接收器、處理器	中控台、資料庫
數量	通常比較多	通常比較少

判斷client server的標準：
server-> 可以拒絕被連線
server-> 不能拒絕被連線

Client 和 Server弄錯
會讓產品很難用

OSI七層模型

越下面越接近硬體

層數	名稱	說明	舉例
7	應用層	通訊協定和資料操作	HTTP
6	展示層	轉譯、加密和壓縮資料	
5	會議層	數據設定和維護電腦之間的通訊連接	SQL
4	傳輸層	協助OSI前三層與OSI後三層進行溝通 處理流量控制和錯誤控制	TCP
3	網路層	形成網路封包	IP
2	資料連結層	負責網路尋址、錯誤偵測和改錯	Wi-Fi、乙太網
1	實體層	物理上的東西	電路、電線

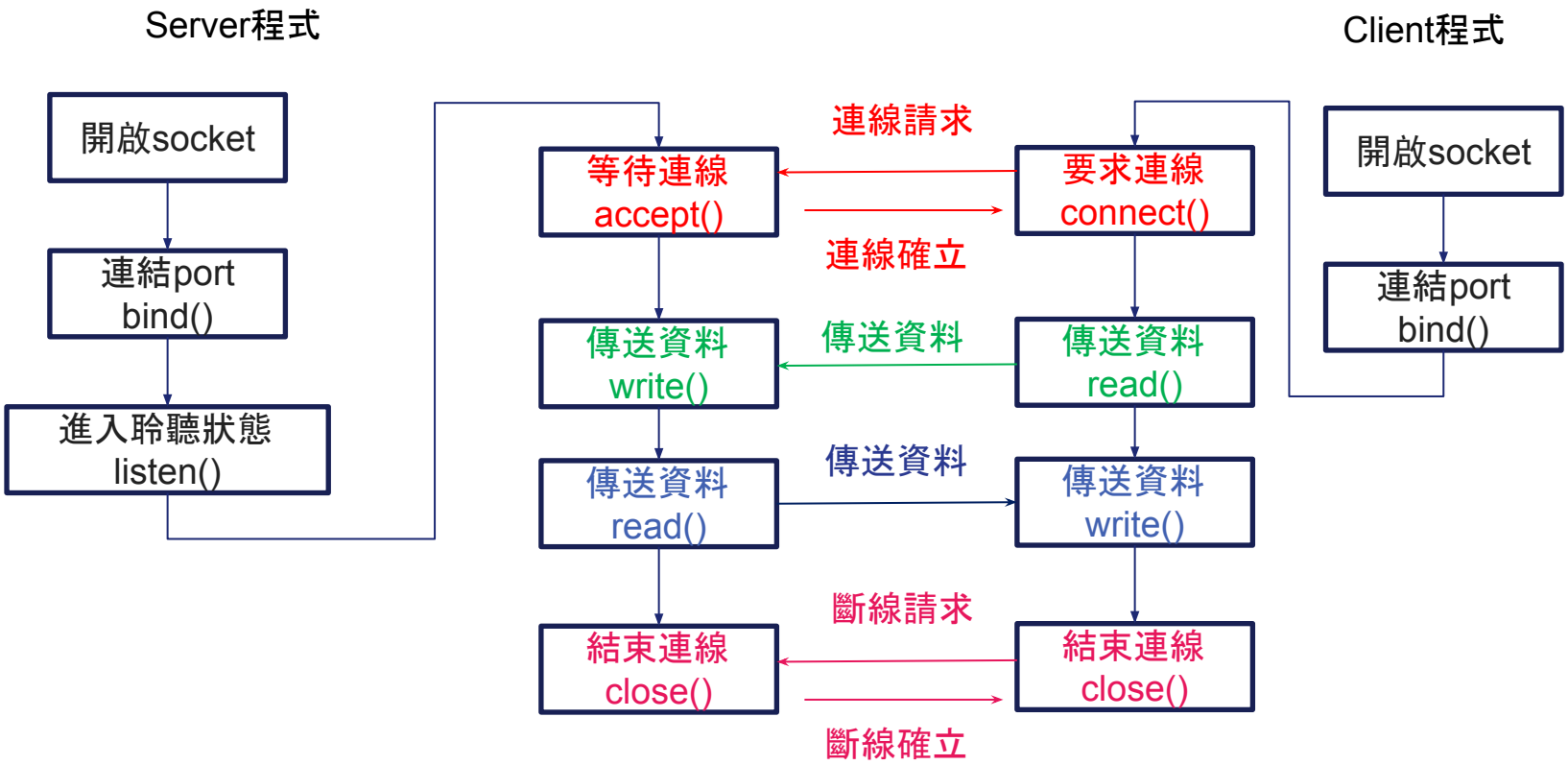
有概念就好，有一些考試喜歡考這一個

常見的名詞介紹 - TCP

TCP: Transmission Control Protocol

利用三次握手建立連線
可以確保對方一定收的到資料(只是封包可能會斷)
這是設備連線常常會聽到的名詞

Tcp/Ip的流程


Server程式
Client程式
開啟socket
連結port
bind()
進入聆聽狀態
listen()
等待連線
accept()
連線請求
要求連線
connect()
連線確立
開啟socket
連結port
bind()
傳送資料
write()
傳送資料
傳送資料
read()
傳送資料
read()
傳送資料
傳送資料
write()
斷線請求
結束連線
close()
結束連線
close()
斷線確立

開始第一個tcp/ip的程式

Server程式

```python
import socket

HOST = '127.0.0.1'
PORT = 7000
MAX_CONNECT_NUMBER = 5

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(MAX_CONNECT_NUMBER)

print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')

while True:
    conn, addr = s.accept()
    print('connected by ' + str(addr))

    while True:
        indata = conn.recv(1024)
        if len(indata) == 0: # connection closed
            conn.close()
            print('client closed connection.')
            break
        print('recv: ' + indata.decode())

        outdata = 'echo ' + indata.decode()
        conn.send(outdata.encode())
```

通常是自己電腦的IP
可以嘗試看看別人的IP會怎樣

https://github.com/JuFengWu/socket_tutorial/blob/master/simple_socket/socket_server.py

開始第一個tcp/ip的程式

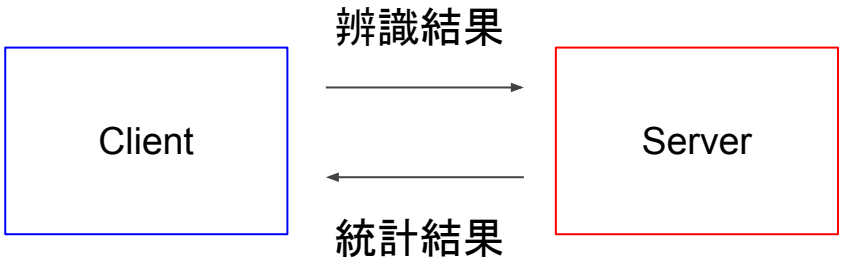
Client程式

```
import socket

HOST = '127.0.0.1'
PORT = 7000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

while True:
    outdata = input('please input message: ')
    print('send: ' + outdata)
    s.send(outdata.encode())

    indata = s.recv(1024)
    if len(indata) == 0: # connection closed
        s.close()
        print('server closed connection.')
        break
    print('recv: ' + indata.decode())
```

https://github.com/JuFengWu/socket_tutorial/blob/master/simple_socket/socket_client.py

大展身手的時間到了

案例：系統做影像辨識之後，可以知道人是否在看廣告
計算看廣告的人/所有經過的人



大展身手的時間到了

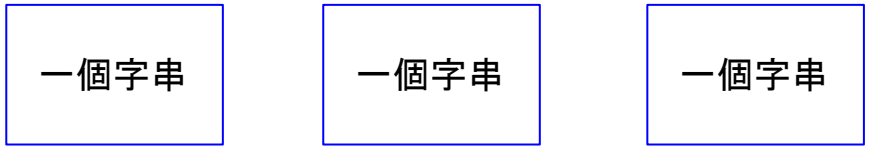
命令表

指令	指令意思	Client	Server
look	看廣告的人	送look字串	收到字串 看廣告的人+1
pass	經過的人	送pass字串	收到字串 經過的人+1
clear	清除結果	送clear字串	清除看廣告和經過的人
show	顯示看廣告的人數 和經過的人數 以及比例	送show字串	顯示結果

所有命令都是Client送給Server

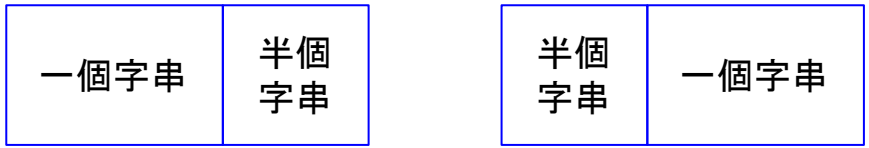
黏包(sticky bag)

正常



例如:
正常你會收到如下字串
look
look
look

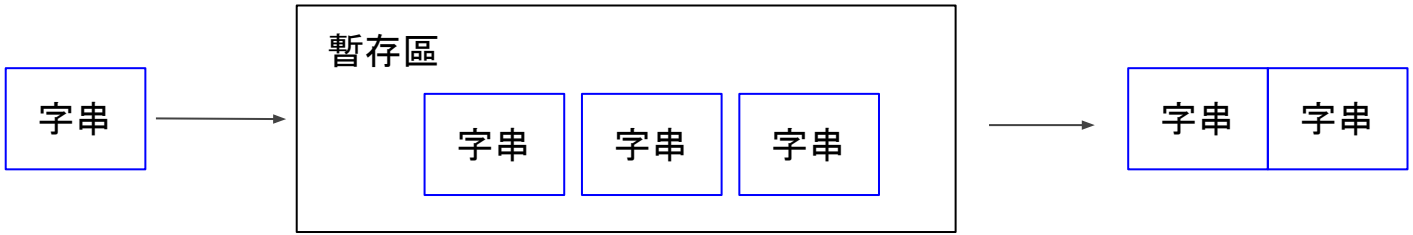
黏包



例如:
黏包你會收到如下字串
looklo
oklook

為什麼會黏包？

一次收或是送太多資料



不只TCP

arduino用USB串接也會

常見的黏包拆解

1.在字串前面或是後面加上特殊符號
2.使用檢查碼ex 使用xor或是總字符數量 (底層居多)

我們來使用第一種吧

XOR是什麼?

黏包拆解

來的封包有幾種可能

1. 正確的 look#
2. 兩個黏在一起 look#look#
3. 1.5個封包+0.5封包 look#lo + ok#
4. 全部散掉 l+oo+k+#

使用string的
split函式 +
find函式

黏包拆解

因為黏包是一個功能
所以會寫成一個function

```
testString = ['look#','look#lo','ok#','l','oo','k#','look#look#']

lastString = ''

sortString = []

def get_string(inputString):
  global lastString
  global sortString
  finalString = lastString + inputString
  if finalString.find('#')!=-1:
    allOutput = finalString.split('#')
    print(allOutput)
    for i in range(len(allOutput)-1):
      sortString.append(allOutput[i])
    lastString = allOutput[len(allOutput)-1]
  else:
    print("put it into lastString")
    lastString = finalString

for t in testString:

  get_string(t)

print("ready to print sorting string")
for s in sortString:
  print(s)
```

https://github.com/JuFengWu/socket_tutorial/blob/master/stcky_bag/seperate_sticky_bag.py

動手的時間到了

把解黏包的程式和剛剛的實做程式結合

測試方法:

1.pass

2. pa + ss

3. lookpa + ss

看看是否OK

使用UDP溝通

特色

- 相對TCP, UDP會一直傳送資料出去, 不會黏包
- UDP不會確定到底對方有沒有收到, 一直丟就對了
- 封包比TCP小, 傳送的速度會更快

使用情境

- 狂送資料, 而且不care舊資料
- 例如新聞台的直播, 傳送馬達現在的狀況

UDP的Client

- UDP少了connect的指令(因為不用握手)
- UPD在送的時候要加上ip位置

UDP Client

```python
import socket

HOST = '127.0.0.1'
PORT = 7000
server_addr = (HOST, PORT)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    outdata = input('please input message: ')
    print('sendto ' + str(server_addr) + ': ' + outdata)
    s.sendto(outdata.encode(), server_addr)

    indata, addr = s.recvfrom(1024)
    print('recvfrom ' + str(addr) + ': ' + indata.decode())
```

TCP Client

```python
import socket

HOST = '127.0.0.1'
PORT = 7000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

while True:
    outdata = input('please input message: ')
    print('send: ' + outdata)
    s.send(outdata.encode())

    indata = s.recv(1024)
    if len(indata) == 0: # connection closed
        s.close()
        print('server closed connection.')
        break
    print('recv: ' + indata.decode())
```

https://github.com/JuFengWu/socket_tutorial/blob/master/udp/udp_client.py

UDP的Server

- UDP少了listen(因為來者不拒) 以及recv (因為不用握手)
- UDP只有receive會有data和address

UDP Server

```python
import socket

HOST = '127.0.0.1'
PORT = 7000

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))

print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')

while True:
    indata, addr = s.recvfrom(1024)
    print('recvfrom ' + str(addr) + ': ' + indata.decode())

    outdata = 'echo ' + indata.decode()
    s.sendto(outdata.encode(), addr)
```

TCP Server

```python
import socket

HOST = '127.0.0.1'
PORT = 7000
MAX_CONNECT_NUMBER = 5

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(MAX_CONNECT_NUMBER)

print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')

while True:
    conn, addr = s.accept()
    print('connected by ' + str(addr))

    while True:
        indata = conn.recv(1024)
        if len(indata) == 0: # connection closed
            conn.close()
            print('client closed connection.')
            break
        print('recv: ' + indata.decode())

        outdata = 'echo ' + indata.decode()
        conn.send(outdata.encode())
```

https://github.com/JuFengWu/socket_tutorial/blob/master/udp/udp_server.py

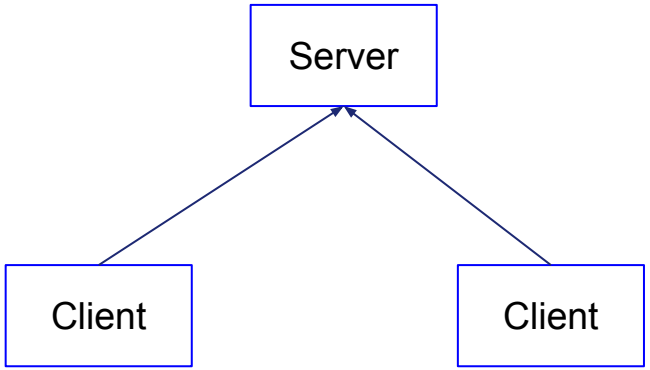
動手的時間到了

把剛剛的TCP/IP的程式改成用UDP去傳送

這時候如果有善用function
寫起來的速度就會快很多

一個Server聽多個Client

一個Server可能會聽多個Client
這會需要改程式嗎?



舉例:

很多主機和攝影機做影像處理
最後回報給一個中控電腦上

一個Server聽多個Client

沿用舊的程式會發生什麼事情?

client1

```
please input message: first
send: first
```

沒連線上

```
server start at: 127.0.0.1:7000
wait for connection...
connected by ('127.0.0.1', 59842)
recv: second
```

server

client2

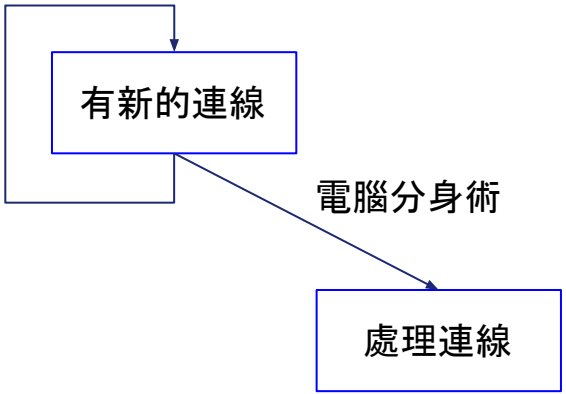
```
server start at: 127.0.0.1:7000
wait for connection...
connected by ('127.0.0.1', 59842)
recv: second
```

有連線上

為什麼會錯誤？

一個Server聽多個Client

有一個while卡著，其他連線進不來
這時候要用Thread的概念去跑



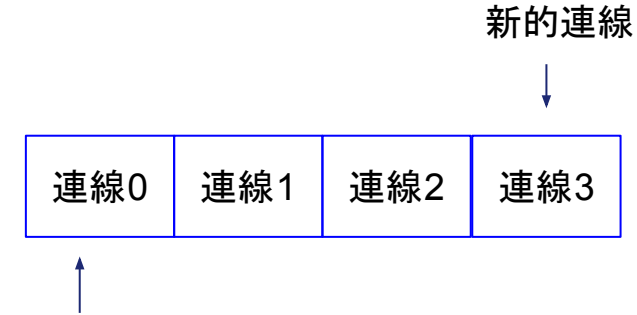

```python
import socket
import threading

HOST = '127.0.0.1'
PORT = 7000
MAX_CONNECT_NUMBER = 5

def new_client(connect, addr):
    print('connected by ' + str(addr))

    while True:
        indata = connect.recv(1024)
        if len(indata) == 0: # connection closed
            connect.close()
            print('client closed connection.')
            break
        print('recv: ' + indata.decode())

        outdata = 'echo ' + indata.decode()
        connect.send(outdata.encode())
    connect.close()
    print("close it")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(MAX_CONNECT_NUMBER)

print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')


while True:
    conn, addr = s.accept()
    threading.Thread(target = new_client, args=(conn,addr,)).start()
```

還記得Thread怎麼用嗎？　　好像還是失敗？

https://github.com/JuFengWu/socket_tutorial/blob/master/multi_connect/socket_server_fail.py

一個Server聽多個Client

- 因為連線的時候，變數被覆蓋
- Call by reference的狀況，其他參數被修改了
- 可以改成用array塞入connect



每一個thread處理各自的連線

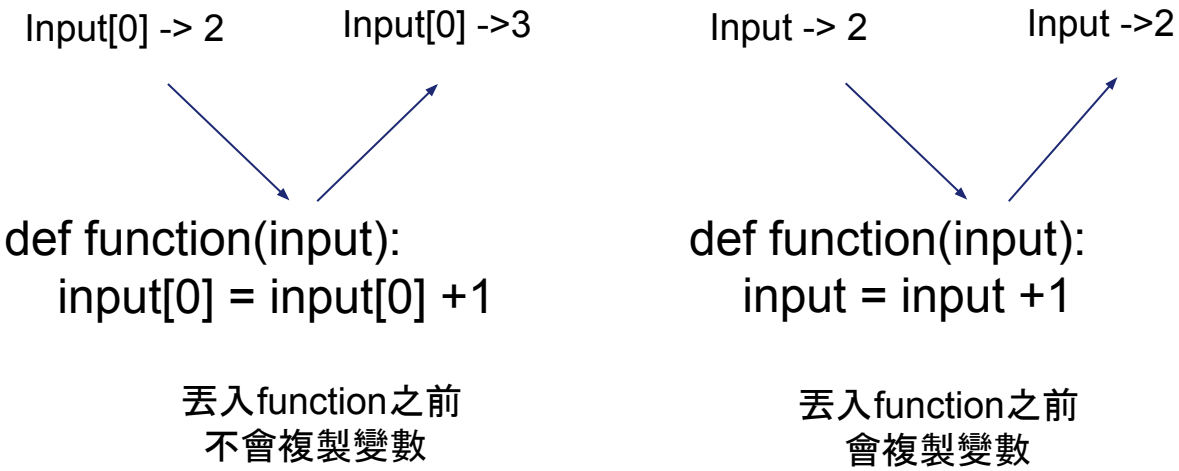
還記得 call by reference嗎?

```python
def new_client(connect, addr,index):
    print('connected by ' + str(addr))

    while True:
        indata = connect[index].recv(1024)
        if len(indata) == 0: # connection closed
            connect[index].close()
            print('client closed connection.')
            break
        print('recv: ' + indata.decode())

        outdata = 'echo ' + indata.decode()
        connect[index].send(outdata.encode())
    connect[index].close()
    print("close it")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(MAX_CONNECT_NUMBER)

print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')

connect = []
index = 0
while True:
    conn, addr = s.accept()
    connect.append(conn)
    threading.Thread(target = new_client, args=(connect,addr,index,)).start()
    index = index+1
```

https://github.com/JuFengWu/socket_tutorial/blob/master/multi_connect/socket_server.py

補充 - Call by reference

call by reference　　call by value



實作的時間到了

把剛剛的程式改回TCP，並且加入多重連線的保護

不要問我為什麼一直改來改去
因為職場上客戶、老闆、PM就是常常改來改去

一個緊急按鈕的反應
按下斷電->按下不斷電->按下徹底斷電

最後

1. socket是一個跨電腦 or 跨程式之間常用的交換資料的方式
2. 常使用字串的方式進行資訊的交換
3. TCP保證收到而且有黏包的問題
4. UDP不保證收到而且沒有黏包的問題
5. 有一些人會把寫socket當作會這一個程式語言的指標

回家作業:
把今天的程式上寫上 說明, 告訴人家怎麼使用
之後會上傳到github

補充 file IO

檔案輸出入的幾個指令

r ： 讀

r+ : 讀寫

a ：增加文字

a+ : 增加文字和讀寫

w : 新建檔案只寫

w+ : 新建檔案讀寫

```
f = open('write_something.txt',"a")
f.write("hi")
f.write("this is python")
f.close
```

https://github.com/JuFengWu/file_io