

計算機網路

暑修版

LAB 2

- Socket Programming
- Simple Web Server
- Testing Simple Web Server
- Final Project Requirements

Socket Programming(1/3)

- Socket Programming採用Server/Client（以下簡稱S/C架構）的架構來實作，所以下面列出幾個簡單觀念先了解後會更快上手
- S/C架構下的程式進行通訊，Server端需要提供一个固定位置（一個IP:port），Client要連接就要先知道這位置
- 因為port是用來辨識電腦上每個獨立的服務，每個不同的服務所使用的port就會不同
- 可用的port範圍從0~65536，前1024個port是TCP/IP欲留著給一些特定協定使用（例如：HTTP=80, FTP=21），所以程式的port只能使用大於1024之後的整數

Socket Programming(2/3)

- 在python中 S/C 是使用socket物件來運作

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 宣告socket 物件 此處為使用TCP
```

- Server端使用bind() 來選定監聽的IP address 與 port number 並用listen() 設定一次可連線的client數量

```
server.bind(('127.0.0.1', 8080)) # 綁定要監聽的IP address以及 port number, s.bind()內的變數型態為tuple  
server.listen(5) # 同時可以連線client的數量( 可與thread搭配使用 )
```

- Client端用Socket和Server端做連接，Server端用 .accept() 來取得和Client端連線的Socket物件

- Client :

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 參數內代表用TCP or UDP 範例是使用TCP  
server.connect(('127.0.0.1', 8080))
```

- Server:

```
client, addr = server.accept() # 等待一台client連線  
print('connected by ' + str(addr)) # 成功連線後才會往下執行
```

Socket Programming(3/3)

- 當Server和Client連接後，就可以用recv()和send()來做資料的讀寫傳輸

```
indata = client.recv(1024) # 等待client傳入資料 資料型態為byte 參數內表示一次能接收到的資料長度 太大有可能遺失
print("recv: " + indata.decode()) # 如若要轉換成string 需加上 .decode()
outdata = "recv: " + indata.decode()
client.send( outdata.encode ) # string to byte 須加上 .encode()
```

- 最後資料傳遞完後，記得要呼叫close()來關閉所有使用到的Sockets

```
server.close() #關閉socket
```

HTTP Protocol

HTTP protocol組成是由client發送request與server 回傳response 來完成
request 包含三部分: request, request header, request body (如下圖)

```
<METHOD> <URL> <VERSION> \r\n
<Header1>: <HeaderValue1>\r\n
<Header2>: <HeaderValue2>\r\n
<HeaderN>: <HeaderValueN>\r\n
\r\n
<Body Data>
```

隔開的部分為空格！

換行使用'\r\n'！

Final project不要求處理Body data
但要可以做parse header

Response 也包含三部分 status code, response header, Entity Body

```
<VERSION> <STATUS CODE> \r\n
<Header1>: <HeaderValue1>\r\n
<Header2>: <HeaderValue2>\r\n
<HeaderN>: <HeaderValueN>\r\n
\r\n
<Body Data>
```

<https://www.rfc-editor.org/rfc/rfc2616>

Simple Web Server (1/5)

```
if __name__ == '__main__':  
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 宣告socket 物件 此處為使用TCP  
  
    server.bind((HOST,PORT)) # 綁定要監聽的IP address以及 port number, s.bind()內的變數型態為tuple  
    server.listen(5) # 同時可以連線client的數量( 可與thread搭配使用 )  
  
    while True :  
        client, addr = server.accept() # 等待一台client連線  
        http_request = client.recv(1024) # 等待client傳入資料 資料型態為byte 參數內表示一次能接收到的資料長度 太大有可能遺失  
        http_request = http_request.decode() #如若需用string 請加上decode  
        print( http_request )  
        http_request = http_request.split('\r\n')  
        response = HTTP_request( http_request )  
        client.send(response.encode())  
        client.close()
```

Simple Web Server (2/5)

```
def HTTP_request_GET( request ) :  
    path = request[0].split(' ')[1]  
    if path == '/' :  
        path = './index.html'  
    else :  
        path = '.' + path  
    try :  
        html_file = open(path, 'r')  
        html_data = html_file.read()  
        response = request[0].split(' ')[2] + STATUS200 # HTTP狀態  
        index = 0  
        for header in request :  
            if index != 0 and index != 1 :  
                response = response + '\r\n' + header # HTTP header  
            index += 1  
        response = response + '\r\n' + html_data # 加上.html 內容 請記得用'\r\n'  
    return response
```


Simple Web Server (3/5)

```
except FileNotFoundError as e :
    html_file = open( './pageNotFound.html' ) # not found page
    html_data = html_file.read()
    response = request[0].split(' ')[2] + STATUS404 # error Message
    index = 0
    for header in request :
        if index != 0 and index != 1 :
            response = response + '\r\n' + header
        index += 1
    response = response + '\r\n' + html_data
    return response
```

Simple Web Server (4/5)

```
def HTTP_Bad_Request( request ) :  
    response = request[0].split(' ')[2] + STATUS400 # error Message  
    response = response + '\r\n' + 'Content-Type: text/html'  
    response = response + '\r\n\r\n' + '<h1>400 Bad Request</h1>'  
    return response  
  
def HTTP_request( request ) :  
    if ( request[0].split(' ')[0] == 'GET' ) :  
        return HTTP_request_GET( request )  
    else :  
        return HTTP_Bad_Request( request )
```

Simple Web Server (5/5)

- 這個 Simple Web Server
 - 在 localhost:port 80 監聽 Client 的連線要求
 - Client 要求 GET 會回傳 200 OK
 - 要求不存在的頁面會回傳404
 - GET 以外的回傳 400 Bad Request
- 200 OK 或 400 Bad Request 都屬於 HTTP 状态码
- HTTP状态码
 - 1XX 信息
 - 2XX 成功
 - 3XX 重定向
 - 4XX 客戶端錯誤
 - 5XX 服務端錯誤

Testing Simple Web Server (1/2)

- 打開瀏覽器輸入網址 127.0.0.1



Testing Simple Web Server (2/2)

- Google Chrome 「postman」

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize

312 hello here is WebServer

hello here is 312 web

🌐 Status: 200 OK Time: 8 ms Size: 420 B [Save Response](#) ▼

Final Project : Building a Simple Web Server

- Sends response to client.
- Handles (Get, Post, Head, Put, Delete) HTTP requests.(5pt)
- Use Cookie to Authenticate a user.(5pt)
- After Authentication, accepts the request.(5pt)
- Parses header.(5pt)
- Obtains requested file from server's file system(upload and download).(5pt)
- Creates HTTP response message : Header line + file.(5pt)
- Need to care Status code(200, 301, 400, 404, 505).(5pt)
 - 301: Return the current page.
 - 301: Return a Hyperlink.
- After creating server, you can request file using a browser(e.g. Chrome, FireFox) or postman.
- Send your code to i-learning assignments.
- Reserve the TA to test your web server.