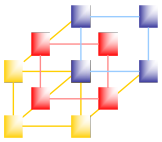
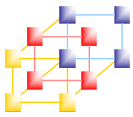


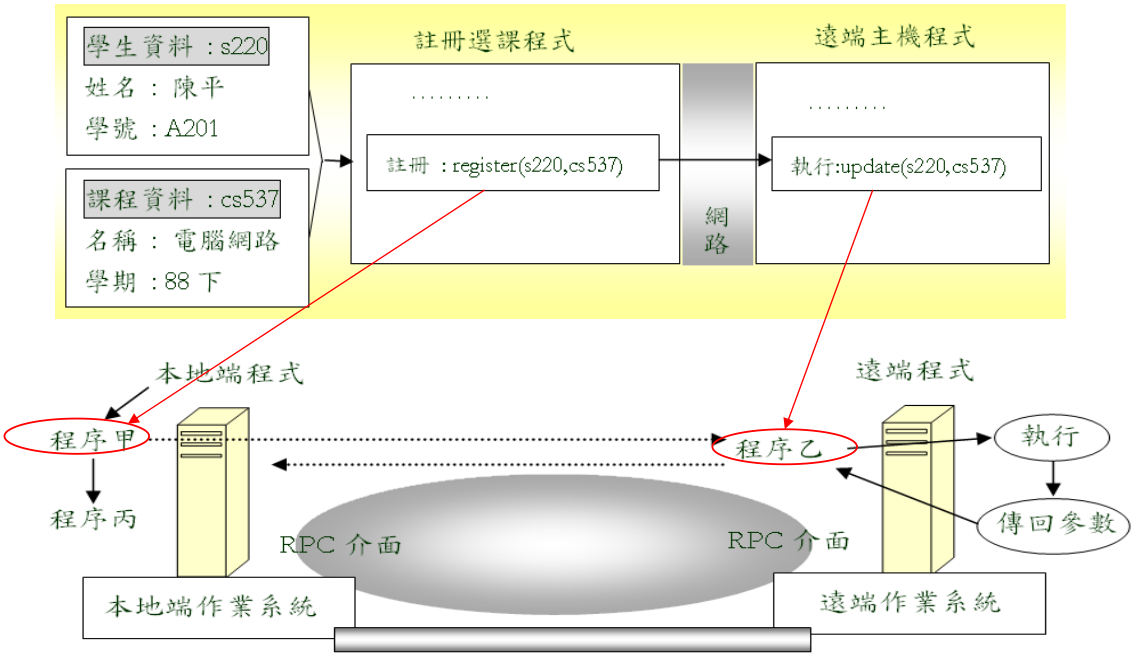
Unit 2
RPC and REST API



RPC (Remote Procedure Calls)

- RPC 為開發網路應用程式常用的方法
  - 使用 Socket 介面撰寫網路應用程式時常須考慮各種通訊的細節（例如資料型式與結構的轉換、連線的管理等）
  - RPC 將這些細節簡化，讓程式設計者可以把時間轉移到應用系統的需求上
- RPC 將網路通訊看成是程式中的程序(procedure)
  - 當程式需要遠端程序提供某種服務或接受訊息時，直接呼叫一個程序，而這個程序有相對應的遠端程序，可以在呼叫後於遠端的電腦上被執行

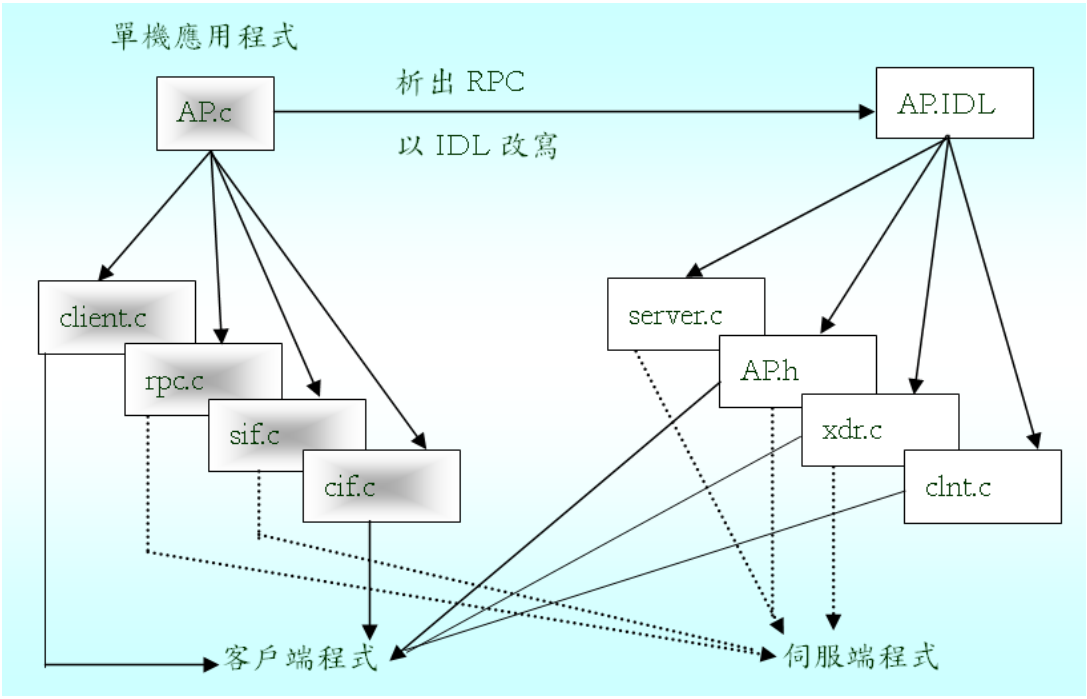
RPC 呼叫的原理



RPC 程式設計

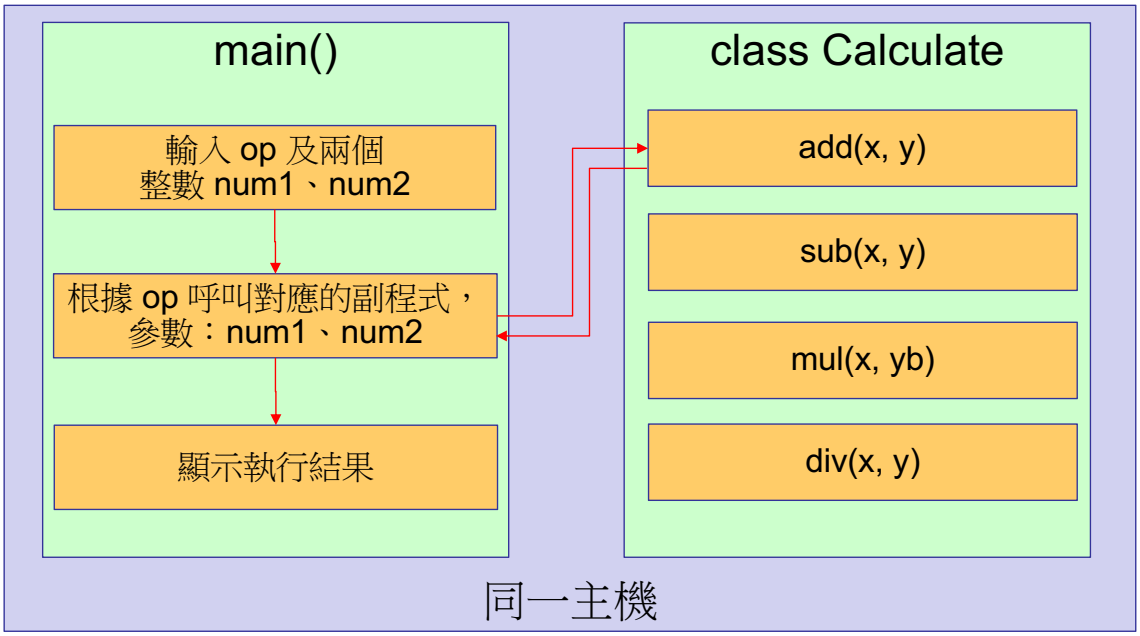
- 先把 RPC 程式寫成一般單機的程式，然後再把通訊的部分換成 RPC
  - 介面定義語言(IDL, Interface Definition Language) 用來定義遠端程序以及一些共同的資料結構
  - 伺服端程式（即 Server Program）執行被呼叫的程序
  - 客戶端程式（即 Client Program）進行遠端程序的呼叫

遠端程序呼叫的程式設計過程



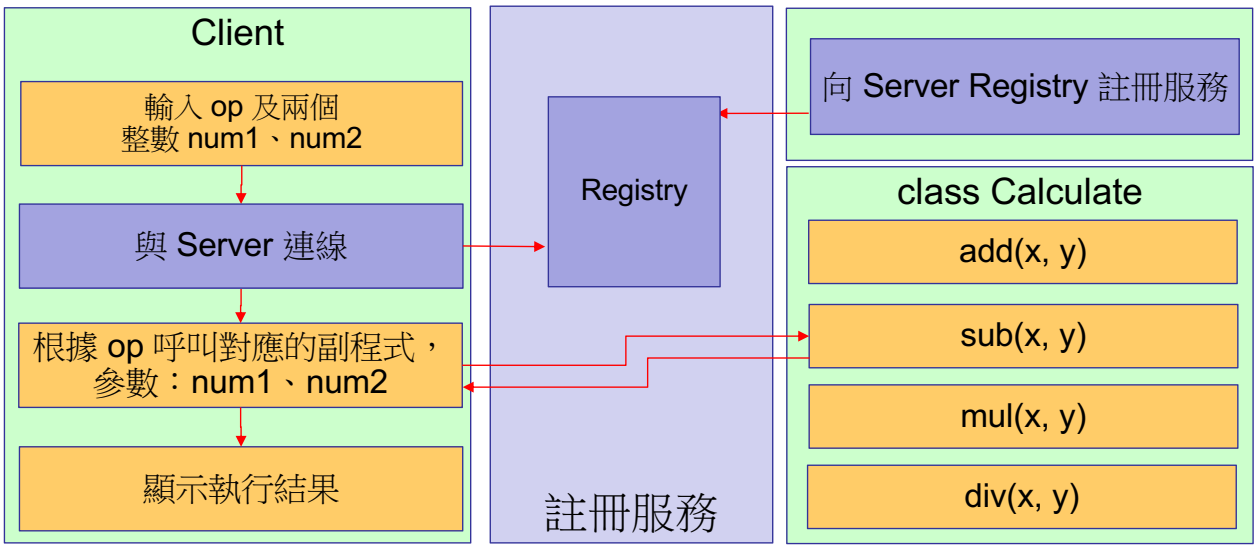
RPC實例 (1/2)

- 1-Calculator.py 為單機版之程式



Python RPC 實例 (2/2)

SimpleXMLRPCServer



2-RPCServer.py
2-RPCClient.py

Install an instance

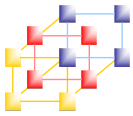
- Server

```
from xmlrpc.server import SimpleXMLRPCServer
Class RPCService:
 # define services

...
obj = RPCService()
server = SimpleXMLRPCServer(('localhost', PORT))
server.register_instance(obj)
…
server.serve_forever()
```

- Client

```
import xmlrpc.server
server = xmlrpc.client.ServerProxy('http://localhost:port')
```

3-RPCServer.py
2-RPCClient.py

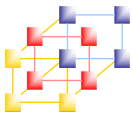


Install functions

- Server

```
def RPCfunction(param1, …):
   …
server = SimpleXMLRPCServer(('localhost', PORT))
server.register_function(RPCfunction, 'method')
```


4-RPCServer.py
4-RPCClient.py



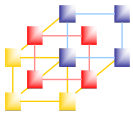
MultiCall object

- Server

```
server.register_multicall_functions()
server.register_function(fun1, 'method1')
server.register_function(fun2, 'method2')
…
```

- Client

```
proxy = xmlrpc.client.ServerProxy('http://localhost:port' )
multicall = xmlrpc.client.MultiCall(proxy)
multicall.fun1(params)
multicall.fun2(params)
…
result = multicall()
```

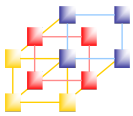


Install an instance with custom dispatch method

5-RPCServer.py
2-RPCClient.py

- Server

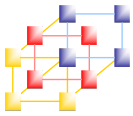
```
class RPCService:
   def _dispatch(self, method, params):
        if(method ==  'fun1'):
               return fun1(params)
        elif(method ==  'fun2'):
               return fun2(params)
        elif(method == …):
               …
        else:
               raise 'Bad method'
```

Multi-thread RPC Server

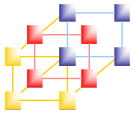
6-MultiThreadRPCServer.py
2-RPCClient.py
6-MultiThreadRPCClient.py

```
from xmlrpc.server import SimpleXMLRPCServer
from socketserver import ThreadingMixIn

class ThreadXMLRPCServer(ThreadingMixIn, SimpleXMLRPCServer):
      pass

server = ThreadXMLRPCServer(('localhost', PORT))
```



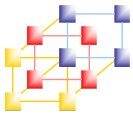
REST APIs

- REST (REpresentational State Transfer) is a software architecture style that defines a pattern for client and server communications over a network
- REST APIs provide access to web service data through public web URLs
- HTTP methods
  - GET Retrieve an existing resource
  - POST Create a new resource
  - PUT Update an existing resource
  - PATCH Partially update an existing resource
  - DELETE Delete a resource



Status Codes

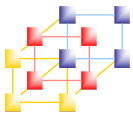
Code	Meaning	Description
200	OK	The requested action was successful.
201	Created	A new resource was created.
202	Accepted	The request was received, but no modification has been made yet.
204	No Content	The request was successful, but the response has no content.
400	Bad Request	The request was malformed.
401	Unauthorized	The client is not authorized to perform the requested action.
404	Not Found	The requested resource was not found.
415	Unsupported Media Type	The request data format is not supported by the server.
422	Unprocessable Entity	The request data was properly formatted but contained invalid or missing data.
500	Internal Server Error	The server threw an error when processing the request.



Packages

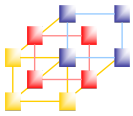
- To write code (client) that interacts with REST APIs, most Python developers turn to requests to send HTTP requests
  - `pip3 install requests`
- Flask is a Python microframework used to build web applications and REST APIs (server)
  - `pip3 install flask`

7-RESTServer.py



REST Server

```
from flask import Flask, json, request, jsonify

API = Flask(__name__)

@API.get("/dir")
def get_func():
      ….

@API.post("/dir")
def add_func():


API.run(host='0.0.0.0', port=PORT, debug=True)
```



REST Client

```
import requests

response = requests.get(URL, params = my_params)

response = requests.post(URL, json=new_dict)

response = requests.put(URL, json=new_dict)
```