**分散式系統  
Lab: Remoting**

請務必填寫學號系級姓名，以免成績登錄錯誤。

學號系級姓名: **108703017 資科三 邱彥翔**

請依問題與提示在指定區域回答問題，並依規定時間內上傳至moodle。

操作一: SOAP-based Web Services開發 (平台: Node.js)

1. 建立一個新的資料夾「lab-remoting」，在此目錄下，新建一個soap目錄
2. 在此lab-remoting目錄中建立一個新的package.json檔案，內容如下:

{

"name": "dslab-remoting",

"version": "1.0.0",

"dependencies": {

"soap": "^0.36.0",

"@grpc/grpc-js": "^1.2.2",

"@grpc/proto-loader": "\*"

}

}

1. 在和package.json同一個目錄下，於命令列執行npm install，安裝所需模組
2. 確認Adder.wsdl、AddMu.wsdl、soapClient.js與soapServer.js等檔案存在lab-remoting/soap目錄中。
3. 開啟並了解soapServer.js程式碼的功能與意義:
4. 請將soapServer.js中，含「讀入wsdl檔」功能的敘述 (請貼上整個statement，也就是分號前的所有程式碼)，貼在下面「答」之後

答:

const xml = require('fs').readFileSync('Adder.wsdl', 'utf8');

1. 請將soapServer.js中，含「實作add並回傳x和y之和的實作」功能的敘述(請貼上整個statement)，貼在下面「答」之後

答:

const service = {

CalculatorImplService: {

CalculatorImplPort: {

add: function (args) {

return {result: args.x + args.y};

}

}

}

};

1. 在程式中，建立http server後，指派給一個變數，該變數的名稱為何? 這個http server傾聽的通訊埠號(port number)為何?

答:

server

8192

1. soap.listen(…)中傳入了四個參數，包含WSDL、http server、服務的實作與一個此服務的掛載網址，請寫出此網址為何?

答: <http://localhost:8192/Adder> (請填入正確答案)

1. 開啟並了解soapClient.js程式碼的功能與意義:
2. 引入soap函式庫後，程式呼叫了soap的createClient的方法，這個方法傳入二個參數，其中一個是SOAP Server的WSDL的位址。請問此位址為何?

答: <http://localhost:8192/Adder?wsdl> (請填入正確答案)

1. 由createClient方法所傳入的回呼函式中有二個參數，分別為err與client，由client我們可以直接呼叫client.add來呼叫SOAP Server上的加法函式。其中，args指的就是傳入遠端add呼叫的參數x與y，請問x與y的值各為何?

答：

(x, y) = (3, 2)

1. 切換目錄到/soap
2. 執行node soapServer.js，在console中應出現server initialized
3. 執行node soapClient.js，觀察console所印出的執行結果。
4. 修改soapClient.js中的args，試著藉由呼叫SOAP Server計算x=10, y=20的結果。將soapClient.js所印出在console中的SOAP訊息貼在下面。

答:

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="http://soap.advsd.nccu/"><soap:Body><tns:add><x>10</x><y>20</y></tns:add></soap:Body></soap:Envelope>

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://soap.advsd.nccu/"><soap:Body><tns:addResponse><tns:result>30</tns:result></tns:addResponse></soap:Body></soap:Envelope>

操作二: 寫作新的SOAP 乘法(multiply)服務

1. 請根據操作一中的觀察，修改soapServer.js，將引入的wsdl檔案由Adder.wsdl改為AddMul.wsdl。
2. 根據AddMul.wsdl中的註解，參考add服務的定義，定義乘法(multiply)服務的相關wsdl宣告。將修改後的AddMul.wsdl貼在答的下方 (提示: 可參考AddMul.wsdl中的註解)

答:

const service = {

CalculatorImplService: {

CalculatorImplPort: {

add: function (args) {

return {result: args.x + args.y};

},

multiply: function(args) {

return {result: args.x \* args.y};

}

}

}

};

1. 修改soapServer.js，在service中新增multiply服務與實作

提示:

const service = {

CalculatorImplService: {

CalculatorImplPort: {

add: function (args) {

return {result: args.x + args.y};

},

*multiply: function(args) {*

*….*

*}*

}

}

};

1. 修改soapServer.js，在修改存取網址為「AddMul」:

soap.listen(server, '/AddMul', service, xml, function () {

console.log('server initialized');

});

1. 關掉並重新執行soapServer.js，在console中應出現server initialized
2. 修改soapClient.js，將url改為<http://localhost:8192/AddMul?wsdl>

const url = 'http://localhost:8192/AddMul?wsdl';

1. 修改soapClient.js，將client.add改為client.multiply

提示: client.multiply(args, function (err, result, rawResponse, soapHeader, rawRequest) {

if (err) console.log(err);

console.log(rawRequest);

console.log('');

console.log(rawResponse);

});

1. 修改soapClient.js中的args，試著藉由呼叫SOAP Server計算x=10, y=20的結果。將soapClient.js所印出在console中的SOAP訊息貼在下面。

答:

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="http://soap.advsd.nccu/"><soap:Body><tns:multiply><x>10</x><y>20</y></tns:multiply></soap:Body></soap:Envelope>

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://soap.advsd.nccu/"><soap:Body><tns:multiplyResponse><tns:result>200</tns:result></tns:multiplyResponse></soap:Body></soap:Envelope>

1. 結束後記得關閉soapServer.js

操作三: gRPC開發 (平台: Node.js)

1. 在「lab-remoting/rpc」目錄下，應該看到client.js, helloworld.proto及server.js等三個檔案
2. 開啟並了解helloworld.proto與server.js程式碼的功能與意義:
3. rpc SayHello (HelloRequest) returns (HelloReply) {}中用到二個訊息 HelloRequest和HelloReply，

message HelloRequest {

string name = 1;

}

message HelloReply {

string message = 1;

}

請問裡面的name=1、message=1，是什麼意思?

答: 代表順序，proto buffer會按照該順序進行編碼

1. 找出程式從那裡讀入helloworld.proto定義檔?

(請整個敘述貼在下方)

答:

var hello\_proto = grpc.loadPackageDefinition(packageDefinition).helloworld;

1. 觀察sayHello函式中如何處理傳入訊息(如何取得參數值name)之後回傳(本題不需作答)。
2. 觀察server.addService()中，sayHello函式是如何登錄到服務中
3. 依序執行server.js、client.js觀察執行結果。
4. 請修改helloworld.proto、server.js與client.js，加入一個新的遠端gRPC函式。(請參考程式中的註解與sayHello的範例)
5. 功能:傳入2個值x、y，回傳results為x+y的結果
6. 名稱: Add (Helloworld.proto), add(server.js和client.js):
7. 訊息與參數: AddRequest，參數依序為int32 x與int32 y
8. 回傳訊息與參數: AddReply，參數為int32 result
9. 修改server.js模仿 function sayHello加入新的函式function add
10. 修改server.js，在server.addService中登錄add函式
11. 修改client.js，模仿client.sayHello新增client.add
12. 測試程式執行結果 (記得重開server.js, 3+2應等於5)
13. 請將修改後的helloworld.proto、server.js與client.js貼下面。

答:

// ========================= helloworld.proto =========================

syntax = "proto3";

package helloworld;

// The greeting service definition.

service Greeter {

// Sends a greeting

rpc SayHello (HelloRequest) returns (HelloReply) {}

// step 5: write a definition for Add here

// ex:

// rpc Add (AddRequest)... (AddReply)

rpc Add (AddRequest) returns (AddReply){}

}

// The request message containing the user's name.

message HelloRequest {

string name = 1;

}

// The response message containing the greetings

message HelloReply {

string message = 1;

}

// step 5-(3) and 5-(4): message AddRequest and message AddReply

// The request message containing the user's name.

message AddRequest {

int32 x = 1;

int32 y = 2;

}

// The response message containing the greetings

message AddReply {

int32 result = 1;

}

// ============================= server.js =============================

var PROTO\_PATH = \_\_dirname + '/helloworld.proto';

var grpc = require('@grpc/grpc-js');

var protoLoader = require('@grpc/proto-loader');

var packageDefinition = protoLoader.loadSync(

PROTO\_PATH,

{

keepCase: true,

longs: String,

enums: String,

defaults: true,

oneofs: true

});

var hello\_proto = grpc.loadPackageDefinition(packageDefinition).helloworld;

/\*\*

\* Implements the SayHello RPC method.

\*/

function sayHello(call, callback) {

callback(null, {message: 'Hello ' + call.request.name});

// first param: if no err send null

}

// add function here: sum x and y and return as {result: ...}

function add(call, callback) {

// you can use call.request.x and call.request.y to obtain x and y

callback(null, {result: call.request.x + call.request.y})

}

/\*\*

\* Starts an RPC server that receives requests for the Greeter service at the

\* sample server port

\*/

function main() {

var server = new grpc.Server();

// step 5-(6): change the following statment to :

// server.addService(hello\_proto.Greeter.service, {sayHello: sayHello, add:add});

server.addService(hello\_proto.Greeter.service, {sayHello: sayHello, add: add});

server.bindAsync('0.0.0.0:50051', grpc.ServerCredentials.createInsecure(), () => {

server.start();

});

//server.bind('0.0.0.0:50051', grpc.ServerCredentials.createInsecure());

}

main();

// ============================= client.js =============================

var PROTO\_PATH = \_\_dirname + '/helloworld.proto';

var grpc = require('@grpc/grpc-js');

var protoLoader = require('@grpc/proto-loader');

var packageDefinition = protoLoader.loadSync(

PROTO\_PATH,

{

keepCase: true,

longs: String,

enums: String,

defaults: true,

oneofs: true

});

var hello\_proto = grpc.loadPackageDefinition(packageDefinition).helloworld;

function main() {

var client = new hello\_proto.Greeter('localhost:50051',

grpc.credentials.createInsecure());

client.sayHello({name: 'Tom'}, function (err, response) {

console.log('Greeting Response:', response.message);

});

// step 5-(2): client.add({x: 3, y: 2}, function (err, response) {...

// });

// note that you should use response.result to get the outcome

client.add({x: 3, y: 2}, function (err, response){

console.log('result: ', response.result);

})

}

main();