

# 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: 112753138 資碩工一 周宸宇

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": "*"
  }
}
```

3. 在和 package.json 同一個目錄下，於命令列執行 npm install，安裝所需模組
4. 確認 Adder.wsdl、AddMu.wsdl、soapClient.js 與 soapServer.js 等檔案存在 lab-remoting/soap 目錄中。
5. 開啟並了解 soapServer.js 程式碼的功能與意義:

(1) 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')
```

(2) 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:

```
add: function (args) {
  return {result: args.x + args.y};
}
```

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

(3) soap.listen(...)中傳入了四個參數，包含 http server、此服務的掛載網址、WSDL、及服務實作，請寫出此網址為何？

答: http://localhost:8192 /Adder

6. 開啟並了解 soapClient.js 程式碼的功能與意義:

(1) 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`

(2) 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 為 3, y 為 2

7. 切換目錄到/soap

8. 執行 node soapServer.js，在 console 中應出現 server initialized

9. 執行 node soapClient.js，觀察 console 所印出的執行結果。

10. 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:ad
d></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:r
esult></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:

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://soap.advsd.nccu/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
name="CalculatorImplService" targetNamespace="http://soap.advsd.nccu/">
  <wsdl:message name="add">
    <wsdl:part name="x" type="xsd:int"> </wsdl:part>
    <wsdl:part name="y" type="xsd:int"> </wsdl:part>
  </wsdl:message>
  <wsdl:message name="addResponse">
    <wsdl:part name="return" type="xsd:int"> </wsdl:part>
  </wsdl:message>

  <!-- insert "multiply" and "multiplyResponse" message tags here-->
  <wsdl:message name="multiply">
    <wsdl:part name="x" type="xsd:int"> </wsdl:part>
    <wsdl:part name="y" type="xsd:int"> </wsdl:part>
  </wsdl:message>

  <wsdl:message name="multiplyResponse">
    <wsdl:part name="return" type="xsd:int"> </wsdl:part>
  </wsdl:message>

  <wsdl:portType name="Calculator">
    <wsdl:operation name="add">
      <wsdl:input message="tns:add" name="add"> </wsdl:input>
      <wsdl:output message="tns:addResponse" name="addResponse">
</wsdl:output>
    </wsdl:operation>
```

```

        <!-- insert "multiply" operation here-->
        <wsdl:operation name="multiply">
            <wsdl:input message="tns:multiply" name="multiply"> </wsdl:input>
            <wsdl:output message="tns:multiplyResponse"
name="multiplyResponse"> </wsdl:output>
        </wsdl:operation>

    </wsdl:portType>
    <wsdl:binding name="CalculatorImplServiceSoapBinding"
type="tns:Calculator">
        <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="add">
            <soap:operation soapAction="" style="rpc"/>
            <wsdl:input name="add">
                <soap:body namespace="http://soap.advsd.nccu/" use="literal"/>
            </wsdl:input>
            <wsdl:output name="addResponse">
                <soap:body namespace="http://soap.advsd.nccu/" use="literal"/>
            </wsdl:output>
        </wsdl:operation>

        <!-- insert "multiply" operation here-->
        <wsdl:operation name="multiply">
            <soap:operation soapAction="" style="rpc"/>
            <wsdl:input name="multiply">
                <soap:body namespace="http://soap.advsd.nccu/" use="literal"/>
            </wsdl:input>
            <wsdl:output name="multiplyResponse">
                <soap:body namespace="http://soap.advsd.nccu/" use="literal"/>
            </wsdl:output>
        </wsdl:operation>

    </wsdl:binding>
    <wsdl:service name="CalculatorImplService">
        <wsdl:port binding="tns:CalculatorImplServiceSoapBinding"
name="CalculatorImplPort">

```

```

        <!-- modify the following url to be "http://localhost:8192/AddMul"
-->
        <soap:address location="http://localhost:8192/AddMul"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

3. 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) {
        ....
      }
    }
  }
};

```

4. 修改 soapServer.js，在修改存取網址為「AddMul」:
 

```

      soap.listen(server, '/AddMul', service, xml, function () {
        console.log('server initialized');
      });

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

```

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

```
7. 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);
      });

```
8. 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:

```
node soapClient.js
```

```
<?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></t
ns: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>
```

## 9. 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:

(1) 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：用來唯一識別變數的號碼，其中一個用途如下

例如：定義 string name = 1, string school = 2，未來在序列化或反序列化的時候，順序被交換或是資訊被省略時，最後解析時也可以透過識別號碼來正確處理資料

(2) 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 packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
```

```
{
    keepCase: true,
    longs: String,
    enums: String,
    defaults: true,
    oneofs: true
});
```

- (3) 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)。
- (4) 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)
  - (1) 功能:傳入 2 個值 x、y，回傳 results 為 x+y 的結果
  - (2) 名稱: Add (在 Helloworld.proto 中), add(在 server.js 和 client.js 中):
  - (3) 修改 helloworld.proto，新增一個 message 名為 AddRequest，參數依序為 int32 x 與 int32 y
  - (4) 修改 helloworld.proto，新增一個回傳 message 名為 AddReply，參數為 int32 result
  - (5) 修改 server.js 模仿 function sayHello 加入新的函式 function add
  - (6) 修改 server.js，在 server.addService 中登錄 add 函式
  - (7) 修改 client.js，模仿 client.sayHello 新增 client.add
  - (8) 測試程式執行結果 (記得重開 server.js, 3+2 應等於 5)
5. 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) 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
message AddRequest {
    int32 x = 1;
    int32 y = 2;
}
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});
    //call.request.name 為 client 端傳來的參數 message 為 server 端回傳的
    訊息
    // 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
    ans = call.request.x + call.request.y
    console.log('x+y 的結果為' + (ans));
    callback(null, {result: ans}); //result 是 proto 定義好的
}

/**
 * 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}); //Greeter.services 為 proto 檔案中的
    service 名稱,{sayHello: sayHello}為 server 端的 function 名稱

```

```

    server.bindAsync('0.0.0.0:50051',
    grpc.ServerCredentials.createInsecure(), () => {
        console.log('server started');
    }); //server.bindAsync 用途是綁定 server 端的 ip 和 port,第一個參數
為 ip 和 port,第二個參數為 server 端的認證方式,第三個參數為 callback
function
    //=> {console.log('server started');}表示 server 端啟動成功後會
印出'server started'

    //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());

```

```
    //var client 是 client 端的物件,hello_proto.Greeter('localhost:50051',grpc.credentials.createInsecure())是 client 端的 ip 和 port

    client.sayHello({name: 'Tom'}, function (err, response) {
        //sayHello 為 proto 檔案中的 rpc 名稱,{name: 'Tom'}為 client 端傳給 server 端的參數 function (err, response)為 callback function
        console.log('Greeting Response:', response.message);
    });
    client.add({x: 3, y: 2}, function (err, response) {
        console.log('Add Response:', response.result);
    //response.result 是 proto 定義好的
    });
    // note that you should use response.result to get the outcome
}
//callback function 用途是當 server 端回傳訊息後,client 端可以透過 callback function 來處理 server 端回傳的訊息

main();
```