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: 108703050 資科三 吳尚恩

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:

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

   (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:

```
const service = {
    CalculatorImplService: {
        CalculatorImplPort: {
            add: function (args) {
                return {result: args.x + args.y};}}}};
```

(3) 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: 該http server的變數名稱為 server , 而他傾聽的埠為 8192

(4) 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)

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 (請填入正確答案)

   (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 : 3 , y : 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:

soap request :

```
<?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>
```

soap response :

```
<?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:

```xml
<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="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>
    <!-- insert "multiply" and "multiplyResponse" message tags
here-->
    <wsdl:portType name="Calculator">
        <wsdl:operation name="multiply">
            <wsdl:input message="tns:multiply" name="multiply"> </
wsdl:input>
            <wsdl:output message="tns:multiplyResponse"
name="multiplyResponse"> </wsdl:output>
        </wsdl:operation>
        <!-- insert "multiply" operation here-->
    </wsdl:portType>
    <wsdl:binding name="CalculatorImplServiceSoapBinding"
type="tns:Calculator">
        <soap:binding style="rpc" transport="http://
schemas.xmlsoap.org/soap/http"/>
        <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>
        <!-- insert "multiply" operation here-->
    </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:

   soap request :
   ```
   <?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>
   ```
   soap response :
   ```
   <?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: name=1 : name's field number = 1

   message=1 : greeting's field number = 1

   (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 (Helloworld.proto), add(server.js和client.js):
   (3) 訊息與參數: AddRequest，參數依序為int32 x與int32 y

(4) 回傳訊息與參數: 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:

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) {
    callback( null,{ result : call.request.x + call.request.y } )
 // you can use call.request.x and call.request.y to obtain x and 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});

    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);
  });
  client.add({ x: 3, y: 2 }, function (err, response) {
    console.log("Add : ", response.result);
  });
  // step 5-(2): client.add({x: 3, y: 2}, function (err, 
response) {...
  //                                      });
  // note that you should use response.result to get the 
outcome
}
main();
```

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;
}
```