Práctica 2

Calculadora con Apache Thrift (RPC)

Miguel Muñoz Molina DSD2

Introducción

En esta práctica vamos a utilizar Apache Thrift para crear una calculadora utilizando un cliente y un servidor de distintos lenguajes de programación.

En mi caso voy a utilizar **java** para el cliente y **python** para el servidor.

Lo primero de todo es crear el archivo .thrift en el que definiremos el nombre de nuestro programa y las distintas operaciones que crearemos.

El archivo nos quedaría así:

```
service Calculadora {
        void ping () ,
        i32 suma (1: i32 num1 , 2: i32 num2 ) ,
        i32 resta (1: i32 num1 , 2: i32 num2 ) ,
        i32 multiplicacion(1: i32 num1 , 2: i32 num2) ,
        i32 division(1: i32 num1 , 2: i32 num2) ,
        i32 potencia(1: i32 num1 , 2: i32 num2) ,
}
```

Tras hacer este archivo, debemos ejecutar la orden que genero los archivos de los distintos lenguajes a usar. Por tanto, haremos:

> thrift -gen java -gen py calculadora.thrift

Esto nos generará dos carpetas, gen-java y gen-py, donde pondremos los distintos archivos.

```
try {
TTransport transport;

transport = new TSocket("localhost", 9090);
transport.open();

TProtocol protocol = new  TBinaryProtocol(transport);
Calculadora.Client client = new Calculadora.Client(protocol);

for (int i=0; i<args.length; i++){
    if (i%2 == 0)
        num.add(Integer.parseInt(args[i]));
    else
        op.add(args[i]);
}

perform(client, op, num);

transport.close();
} catch (TException x) {
x.printStackTrace(); }
```

La idea sigue siendo la misma, le paso una cadena de operaciones de la longitud que quiera y se guarda en dos vectores.

Para el servidor de python, definimos todas las operaciones de la calculadora, además de definir la dirección y puerto del servidor.

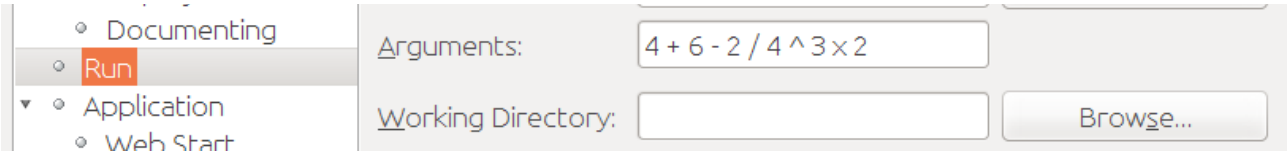
```
if __name__ == '__main__':
    handler = CalculadoraHandler()
    processor = Calculadora.Processor( handler )
    transport = TSocket.TServerSocket( host = '127.0.0.1' , port =9090)
    tfactory = TTransport.TBufferedTransportFactory()
    pfactory = TBinaryProtocol.TBinaryProtocolFactory()

    server = TServer.TSimpleServer( processor , transport , tfactory , pfactory )

    print ( 'Iniciando servidor ...')

    server.serve()

    print ( 'done .')
```

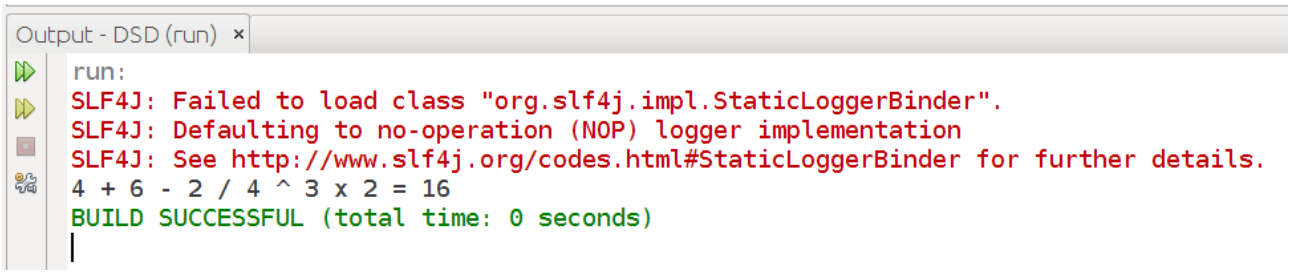
Tras esto ya solo nos queda ejecutar el cliente y el servidor en terminales distintas.

Un problema que tuve es al compilar java desde la terminal me daba un error, así que decidí añadirlo a un proyecto de netbeans (añadiendo las librerías respectivas) en el que no daba ese problema e iba todo con normalidad.

Por ello, mi idea es pasar la carpeta del proyecto a gen-java, pero no funciona al compilar los archivos con los paquetes, da warnings y en teoría hace bien la compilación pero dice no encontrar el main en el JavaClient, el cual sí está y puede que sea algún problema con mi ordenador. Es por ello que es recomendable pasar la carpeta dsd a un proyecto de netbeans para que funcione añadiendo en el apartado de run del proyecto los argumentos a dar (siento las molestias).

De todas formas, voy a apoyar con capturas de pantalla de varios ejemplos para justificar su correcto funcionamiento.





Como se ve, da un “error” a la hora de ejecutar que no influye en el funcionamiento del programa. Probablemente mi versión del paquete slf4j no es el correcto pero no parece interferir con el resultado.

Mas ejemplos de ejecución: