ugr | Universidad de Granada

Desarrollo de Sistemas Distribuidos

Práctica 2:
APACHE THRIFT

José Manuel Navarro Cuartero

Índice

Introducción

La potencia de los IDLs (Interface Description Language) está en que podemos tener un servidor y un cliente escritos en lenguajes completamente distintos, y el IDL hará de intermediario para que puedan trabajas entre ellos.
En esta práctica, partiendo de un archivo base llamado “calculadora.thrift”, en el que definimos las operaciones que va a realizar el servidor, generamos una serie de archivos con “thrift -gen”.
Posteriormente, añadiremos los archivos principales, “cliente.py” y “servidor.py”, que serán los que utilizaremos.
Describiremos el proceso paso a paso.

Calculadora.thrift

```
1   service Calculadora{
2      void ping(),
3      double suma(1:double num1, 2:double num2),
4      double resta(1:double num1, 2:double num2),
5      double multiplica(1:double num1, 2:double num2),
6      double divide(1:double num1, 2:double num2),
7      i32 modulo(1:i32 num1, 2:i32 num2),
8      double potencia(1:double num1, 2:double num2),
9      double raiz(1:i32 num1, 2:double num2),
10  }
```

Como puede verse, hemos definido 7 operaciones distintas, además del ping que utilizaremos en el desarrollo para comprobar que la conexión funciona:

- Operaciones básicas: suma, resta, multiplicación y división.
- Operaciones con enteros: módulo, potencia y raíz.

A partir de este archivo, con la orden “thrift -gen py calculadora.thrift” se generarán el resto de archivos necesarios.
Además, nosotros crearemos manualmente “cliente.py” y “servidor.py”

Servidor

El archivo del servidor, “servidor.py” deberá contener todas las funciones que definimos en el archivo base.
Por una parte tendremos la parte en la que definimos el handler (la calculadora per se), y lanzaremos el 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("fin")
```

Por otro lado, en el handler definiremos las operaciones y daremos el resultado.

```
    def suma(self, n1, n2):
        print(str(n1) + " + " + str(n2))
        return n1 + n2

    def resta(self, n1, n2):
        print(str(n1) + " - " + str(n2))
        return n1 - n2
```

Cliente

En el archivo del cliente, “cliente.py” recogeremos el cálculo que nos piden desde el teclado y llamaremos al servidor.
Primero definimos el socket de transporte y lo abrimos.

```
transport = TSocket.TSocket("localhost", 9090)
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocol(transport)

client = Calculadora.Client(protocol)

transport.open()
```

Tomamos del teclado la operación a realizar.

```
print("Introduce la operacion:")
teclado = input()
teclado = teclado.split()
x_f = float(teclado[0])
y_f = float(teclado[2])
op = teclado[1]
```

Y llamamos al servidor para que nos dé el resultado.

```
    case 'x':
        resultado = client.multiplica(x_f,y_f)
    case '/':
        resultado = client.divide(x_f,y_f)
    case "mod":
```

Uso

Para utilizarlo en nuestro PC, como python3 es un lenguaje interpretado, no tendremos que compilar nada. Sólo tendremos que descomprimir el zip, ejecutaremos el servidor con “python3 servidor.py”, y el cliente con “python3 cliente.py”.