

Memoria Práctica 2 - Thrift

Al igual que en la práctica anterior, el ejercicio consiste en implementar un programa distribuido que realice las operaciones básicas de una calculadora haciendo uso de la herramienta Thrift:

- El cliente realiza las llamadas al servidor dependiendo de la operación elegida.
- El servidor contiene las operaciones y realiza los cálculos.

El entorno de ejecución y desarrollo de la práctica es una máquina virtual con el sistema operativo Ubuntu 20.04 y el intérprete es Python3. Esta vez he extendido la funcionalidad de la calculadora. Tiene tres modos de ejecución, b (básico) | v (vectores) | m (matrices). Se debe pasar el modo por argumento al programa cliente, y dependiendo de este, el programa solicitará la entrada por teclado de unos datos y operaciones u otros.

Las operaciones implementadas en el lado del servidor son las siguientes (descritas en el archivo *calculadora.thrift*).

He realizado las operaciones básicas pedidas de la calculadora, es decir, la suma, resta, multiplicación y división de enteros. Además, he añadido raíces cuadradas

Los ejemplos están ejecutados en local con dos terminales abiertas simultáneamente (una emulando el cliente y otra el servidor).

En el archivo *calculadora.x* he definido el programa, la versión y las funciones así como una estructura auxiliar llamada *inputs* que almacena los dos operandos enteros para una mayor comodidad y limpieza en el código.

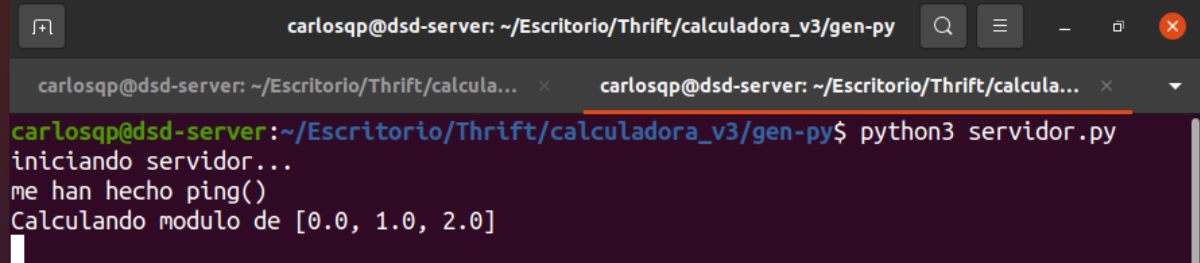
```
service Calculadora{
    void ping(),
    i32 suma(1:i32 num1, 2:i32 num2),
    i32 resta(1:i32 num1, 2:i32 num2),
    i32 multiplicacion(1:i32 num1, 2:i32 num2),
    double division(1:i32 num1, 2:i32 num2),
    double raiz(1:i32 num1),
    double modulo(1:list<double> v1),
    list<double> normalizar(1:list<double> v1),
    list<double> suma_v(1:list<double> v1, 2:list<double> v2),
    list<double> resta_v(1:list<double> v1, 2:list<double> v2),
    list<list<double>> suma_m(1:list<list<double>> m1, 2:list<list<double>> m2),
    list<list<double>> resta_m(1:list<list<double>> m1, 2:list<list<double>> m2),
    list<list<double>> multiplicacion_m(1:list<list<double>> m1, 2:list<list<double>> m2)
}
```

Código de *calculadora.thrift*

Tras definir el programa, generamos los archivos necesarios para su implementación y ejecución distribuida con la herramienta thrift (*thrift -gen py calculadora.thrift*) al igual que vimos en el ejemplo de la práctica.

Una vez hecho esto, se crea el directorio gen-py, donde implementaremos el código de los dos programas: *cliente.py* y *servidor.py*.

En el archivo *servidor.py* se encuentra la clase *CalculadoraHandler* que contiene la implementación de los métodos/operaciones de la calculadora (ver el código del archivo). Cabe destacar que en ciertas operaciones como la división, la raíz cuadrada o las operaciones con vectores/matrices se tratan los errores con excepciones, puesto que el lenguaje Python lo permite (este no era el caso en la práctica anterior). Para ejecutar el servidor, simplemente ejecutamos la orden *python3 servidor.py* desde la carpeta donde se encuentre el archivo.



```
carlosqp@dsd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dsd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 servidor.py
iniciando servidor...
me han hecho ping()
Calculando modulo de [0.0, 1.0, 2.0]
```

Ejemplo de salida del servidor

En el cliente (archivo *cliente.py*) implementamos el programa principal. Este requiere un argumento opcional (*--host*) que permite elegir el host, y un argumento obligatorio, el modo de ejecución (básico, vectores o matrices).



```
carlosqp@dsd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dsd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py -h
usage: cliente.py [-h] [--host HOST] mode

positional arguments:
  mode                modo: b(basico) | v(vectores) | m(matrices)

optional arguments:
  -h, --help          show this help message and exit
  --host HOST         IP del servidor
carlosqp@dsd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$
```

Con la orden *python3 cliente.py -h* podemos ver los argumentos necesarios

Este programa, al contrario que el cliente de Sun RPC, es un programa interactivo, por lo que los datos los tiene que introducir el usuario por teclado en tiempo de ejecución.

A continuación, mostraré algunos ejemplos de ejecución:

```

carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py b
hacemos ping al server
primer operando:      25
operacion + | - | * | / | r(raiz cuadrada):      r
raiz(25) = 5.0
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py b
hacemos ping al server
primer operando:      -1
operacion + | - | * | / | r(raiz cuadrada):      r
Math Error: negative square root
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$

```

Ejemplo de ejecución del modo básico 'b'

```

carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py b
hacemos ping al server
primer operando:      50
operacion + | - | * | / | r(raiz cuadrada):      /
segundo operando:      80
50/80 = 0.625
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py b
hacemos ping al server
primer operando:      1
operacion + | - | * | / | r(raiz cuadrada):      /
segundo operando:      0
Math Error: cannot divide by 0
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$

```

Ejemplo de ejecución del modo básico 'b'

```

carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x carlosqp@dssd-server: ~/Escritorio/Thrift/calcula... x
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py v
hacemos ping al server
tamaño vector(es):      3
v1 [0] = 0
v1 [1] = 1
v1 [2] = 2
operacion mod(modulo) | norm(normalizar) | + | - :      mod
modulo ([0.0, 1.0, 2.0]) = 2.23606797749979
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py v
hacemos ping al server
tamaño vector(es):      2
v1 [0] = 2
v1 [1] = 4
operacion mod(modulo) | norm(normalizar) | + | - :      -
v2 [0] = 1
v2 [1] = 1
[2.0, 4.0] - [1.0, 1.0] = [1.0, 3.0]
carlosqp@dssd-server:~/Escritorio/Thrift/calculadora_v3/gen-py$

```

Ejemplo de ejecución del modo de vectores 'v'

```

carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py
carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py m
hacemos ping al server
numero de filas:      2
numero de columnas:   2
m1 [0] [0] = 1
m1 [0] [1] = 0
m1 [1] [0] = 0
m1 [1] [1] = 1
operacion + | - | * : *
numero de filas:      2
numero de columnas:   2
m2 [0] [0] = 1
m2 [0] [1] = 3
m2 [1] [0] = 7
m2 [1] [1] = 8
Resultado de m1 * m2 es...
[1.0, 3.0]
[7.0, 8.0]
carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$

```

Ejemplo de ejecución del modo matrices 'm'

```

carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py m
hacemos ping al server
numero de filas:      2
numero de columnas:   1
m1 [0] [0] = 3
m1 [1] [0] = 4
operacion + | - | * : -
numero de filas:      1
numero de columnas:   2
m2 [0] [0] = 3
m2 [0] [1] = 4
Math error: matrix must have the same size!!
carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$ python3 cliente.py m
hacemos ping al server
numero de filas:      3
numero de columnas:   1
m1 [0] [0] = 4
m1 [1] [0] = 5
m1 [2] [0] = 6
operacion + | - | * : *
numero de filas:      1
numero de columnas:   4
m2 [0] [0] = 5
m2 [0] [1] = 6
m2 [0] [2] = 7
m2 [0] [3] = 8
Resultado de m1 * m2 es...
[20.0, 24.0, 28.0, 32.0]
[25.0, 30.0, 35.0, 40.0]
[30.0, 36.0, 42.0, 48.0]
carlosqp@dssd-server: ~/Escritorio/Thrift/calculadora_v3/gen-py$

```

Ejemplo de ejecución del modo matrices 'm'