

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

6-4-2022

Práctica 2.2: Apache Thrift

Calculadora con operaciones
básicas y complejas

Several thin, curved lines in dark blue and light grey that sweep upwards from the bottom left corner.

Alberto Llamas González
3º GII, IS

1. Introducción

Esta segunda parte de la práctica 2 consiste, al igual que la primera, en desarrollar una calculadora utilizando esa misma filosofía de cliente-servidor utilizando la tecnología de Apache Thrift. El cliente envía una serie de datos al servidor quien realiza las operaciones correspondientes y devuelve el resultado al cliente, que lo imprime por pantalla.

En principio, se exigían unas operaciones mínimas básicas (suma, resta, multiplicación y división) y se han añadido algunas operaciones más complejas.

2. Calculadora básica

En primer lugar, se ha implementado una calculadora con las operaciones básicas, suma, resta, multiplicación y división. Para ello, siguiendo la filosofía de la primera parte de la práctica, se introducirá por la terminal los parámetros a operar. En el archivo de definiciones de las operaciones, tendríamos las siguientes:

```
void ping(),  
i32 suma(1:i32 num1, 2:i32 num2),  
i32 resta(1:i32 num1, 2:i32 num2)  
i32 producto(1:i32 num1, 2:i32 num2)  
double division(1:i32 num1, 2:i32 num2)
```

Al ejecutar `thrift -gen` nos generaría la carpeta `gen-py` y añadiendo los archivos `cliente.py` y `servidor.py` tendríamos la base de nuestra calculadora lista. En el archivo `servidor.py`, añadimos las funciones con las operaciones previamente mencionadas. A continuación, mostramos un ejemplo de la implementación de la multiplicación (ya que la suma y resta venían implementadas en el código de ejemplo):

```
def producto(self, n1, n2):  
    print("multiplicando " + str(n1) + " con " + str(n2))  
    return n1 * n2
```

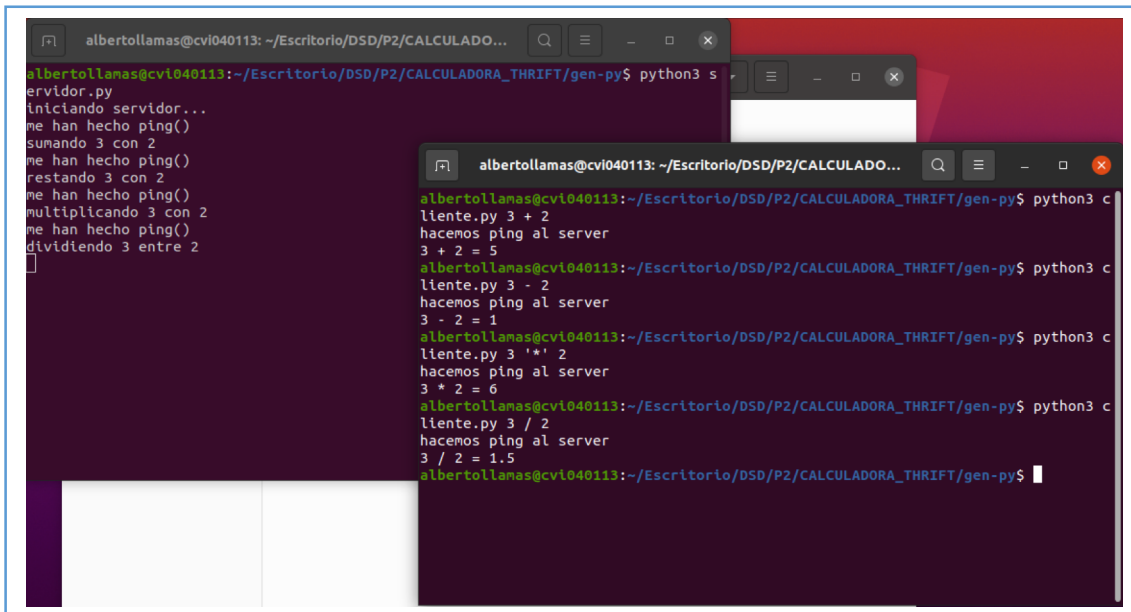
En el cliente, se usa una función denominada `basica(num1, operacion, num2)` donde se llama a una función u otra del servidor, dependiendo de la operación introducida por el usuario.

```
def basico (num1, operacion, num2):  
    if (operacion == '+'):  
        resultado = client.suma(num1,num2)  
    elif(operacion == '-'):  
        resultado = client.resta(num1, num2)  
    elif (operacion == '*'):  
        resultado = client.producto(num1, num2)  
    elif (operacion == '/'):  
        resultado = client.division(num1, num2)  
    else:  
        resultado = None  
  
    return resultado
```

Ejemplo de uso:

python3 servidor.py (En una terminal)

python3 cliente.py <num1> <operación (+,-,*,/)> <num2> (En otra terminal)



```

albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 s
servidor.py
iniciando servidor...
me han hecho ping()
sumando 3 con 2
me han hecho ping()
restando 3 con 2
me han hecho ping()
multiplicando 3 con 2
me han hecho ping()
dividiendo 3 entre 2
me han hecho ping()

albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 c
cliente.py 3 + 2
hacemos ping al server
3 + 2 = 5
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 c
cliente.py 3 - 2
hacemos ping al server
3 - 2 = 1
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 c
cliente.py 3 '*' 2
hacemos ping al server
3 * 2 = 6
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 c
cliente.py 3 / 2
hacemos ping al server
3 / 2 = 1.5
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$

```

3. Aumentando la complejidad

Una vez realizadas las operaciones básicas, se decidió realizar operaciones algo más complejas utilizando vectores y matrices. Se añadió en el archivo de definiciones (.thrift) las siguientes operaciones:

```

list<i32> sumaVectores(1: list<i32> vector1, 2: list<i32> vector2),
list<i32> restaVectores(1: list<i32> vector1, 2: list<i32> vector2),
list<i32> productoVectores(1: list<i32> vector1, 2: list<i32> vector2),
list<double> divisionVectores(1: list<i32> vector1, 2: list<i32> vector2),
i32 productoEscalar(1: list<i32> vector1, 2: list<i32> vector2),
list<list<i32>> sumaMatrices (1: list<list<i32>> matriz1, 2:
list<list<i32>> matriz2),

list<list<i32>> restaMatrices (1: list<list<i32>> matriz1, 2:
list<list<i32>> matriz2),

list<list<i32>> productoMatrices (1: list<list<i32>> matriz1, 2:
list<list<i32>> matriz2)

```

Serían operaciones básicas entre vectores y entre matrices. Para ello, se ha utilizado la biblioteca de Python *numpy* realizando al inicio del archivo

servidor.py el siguiente import -> **import numpy as np** (es necesario instalar la biblioteca).

También se ha implementado una función que calcula el producto escalar de 2 vectores. Mostraré una operación básica con matrices y otra con vectores.

```
def productoVectores(self, vector1, vector2):
    print(f"multiplicando {str(vector1)} * {str(vector2)} ")
    return np.array(vector1) * np.array(vector2)

def productoMatrices(self, matriz1, matriz2):
    print(f"multiplicando {str(matriz1)} y {str(matriz2)} ")
    return np.array(np.matrix(matriz1) * np.matrix(matriz2))
```

En el cliente dependiendo de si el segundo argumento pasado por parámetros es "vector" o "matriz", realizamos una operación u otra. Se muestra a continuación, dos ejemplos de uso, uno para cada tipo (vectores y matrices)

```
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADO...
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 s
servidor.py
iniciando servidor...
me han hecho ping()
sumando 3 con 2
me han hecho ping()
restando 3 con 2
me han hecho ping()
multiplicando 3 con 2
me han hecho ping()
dividiendo 3 entre 2
me han hecho ping()
sumando [1, 2] + [2, 3]
me han hecho ping()
haciendo el producto escalar de [1, 2] y [3, 4]
[1, 2] * [3, 4] = [3, 8]

51 def matrices(matriz1, matriz2, operacion):
52     if (len(matriz1) == len(matriz2)):
53         if (operacion == '+'):
54             resultado = client.sumaMatriz(matriz1, matriz2)
55         elif (operacion == '-'):
56             resultado = client.restaMatriz(matriz1, matriz2)
57         elif (operacion == '*'):
58             resultado = client.productoMatrices(matriz1, matriz2)
59         elif (operacion == '/'):
60             resultado = client.divisionMatriz(matriz1, matriz2)
61         else:
62             print("Operación no válida")
63     else:
64         print("Matrices no compatibles")
65     return resultado

albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 c
cliente.py
hacemos ping al server
Introduzca el numero de elementos de los vectores: 2
Introduzca los elementos del primer vector:
1
2
vector 1: [1, 2]
Introduzca la operacion a realizar: escalar
Introduzca los elementos del segundo vector:
3
4
vector 2: [3, 4]
[1, 2] * [3, 4] = [3, 8]
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$
```

```

albertollamas@cvl040113: ~/Escritorio/DSD/P2/CALCULADO...
albertollamas@cvl040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3 s
servidor.py
Iniciando servidor...
me han hecho ping()
sumando 3 con 2
me han hecho ping()
restando 3 con 2
me han hecho ping()
multiplicando 3 con 2
me han hecho ping()
dividiendo 3 entre 2
me han hecho ping()
sumando [1, 2] + [2, 3]
me han hecho ping()
haciendo el producto escalar de [1, 2] y [3, 4]
me han hecho ping()
multiplicando [[1, 2], [3, 4]] y [[5, 6], [7, 8]]
[]

51 def matrices(matriz1, matriz2, operacion):
52     if (len(matriz1) == len(matriz2)):
53         if (operacion == '+'):
54             resultado = client.sumaMatrices(m
55         elif (operacion == '-'):
56             resultado = client.restaMatrices(
57         elif (operacion == '*'):

```

```

albertollamas@cvl040113:~/Escritorio/DSD/P2/CALCULADO...
albertollamas@cvl040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$ python3
cliente.py matriz
hacemos ping al server
Introduzca el numero de filas de las matrices: 2
Introduzca el numero de columnas de las matrices: 2
Introduzca los elementos de la primera matriz:
1
2
3
4
5
6
7
8
Introduzca la operacion a realizar: *
Introduzca los elementos de la segunda matriz:
5 6
7 8
[[1, 2], [3, 4]] * [[5, 6], [7, 8]] = [[19, 22], [43, 50]]
albertollamas@cvl040113:~/Escritorio/DSD/P2/CALCULADORA_THRIFT/gen-py$

```

4. Conclusiones

Concluyendo, podemos observar que con Apache Thrift se realiza todo de una manera más sencilla. Nos permite, aunque no se haya implementado en otro lenguaje, utilizar Ruby por ejemplo para programar el servidor. Además, el hecho de que se permita el uso de Python, facilita las cosas ya que no hay problemas de reserva de memoria como con Sun RPC (debíamos usar C) y tenemos bibliotecas que nos facilitan cálculos más complejos como el de vectores o matrices.

5. Bibliografía

- <https://www.oreilly.com/library/view/machine-learning-with/9781491989371/ch01.html>
- <https://docs.hektorprofe.net/python/entradas-y-salidas/entrada-por-script/>
- Apuntes proporcionados por los profesores de la asignatura

