

Práctica 2-Apache Thrift

Santiago Muñoz Castro

April 2021

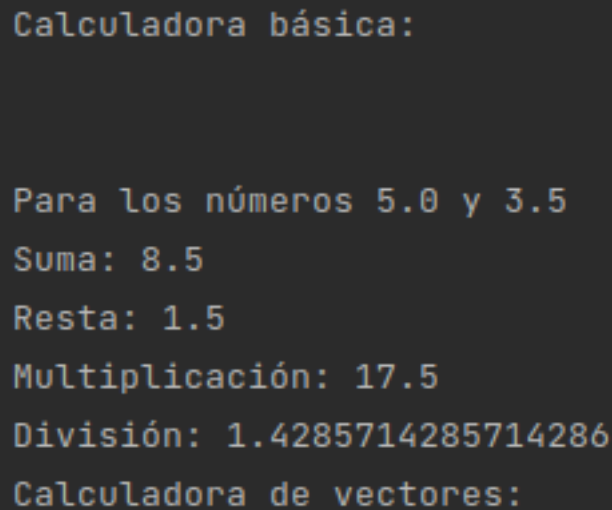
1 Introducción

En esta práctica he implementado la calculadora realizada en la práctica 1 de RPC, pero utilizando esta vez Apache Thrift que nos ha ayudado a partir de un archivo .thrift a generar código y librerías para python y java.

2 Funciones implementadas

Las funciones implementadas en la calculadora son las siguientes:

- Calculadora básica: suma, resta, multiplicación y división:



```
Calculadora básica:  
  
Para los números 5.0 y 3.5  
Suma: 8.5  
Resta: 1.5  
Multiplicación: 17.5  
División: 1.4285714285714286  
Calculadora de vectores:
```

Figure 1: Calculadora básica

- Calculadora de vectores: suma, resta, multiplicación y división de vectores.

```

Calculadora de vectores:

El vector 1 es:
1.0  2.0  3.0  4.0
El vector 2 es:
0.0  5.0  6.0  7.0
Suma:
1.0  7.0  9.0  11.0
Resta:
1.0  -3.0  -3.0  -3.0
Multiplicacion:
0.0  10.0  18.0  28.0
Error division: No se puede dividir un número entre 0, es una indeterminación

```

Figure 2: Calculadora de vectores

- Calculadora de ecuaciones cuadráticas: del tipo ax^2+bx+c .

```

Calculadora ecuacion cuadratica:

La solucion a la ecuacion de segundo grado 3.0x^2 5.0x -2.0 es:
0.3333333333333333  -2.0
Calculadora operacion combinada:

```

Figure 3: Calculadora de ecuaciones cuadráticas

- Calculadora de operaciones combinadas.

```

La soluciona a la operacion combinada 5 + 8 - 7 * 3 + 5 / 5 - 7 * 10 es -77.0

```

Figure 4: Calculadora de operaciones combinadas

3 Implementación de la práctica

Para la realización de la práctica he decidido que el servidor este programado en python y el cliente este programado en java. Para empezar generamos un archivo .thrift en el que declararemos todas las estructuras y funciones que queremos que thrift transforme a esos códigos. Por ejemplo, para la calculadora básica he declarado un struct (operacionBasica), la cual consta de dos variables (a y b) de tipo double.

```
1 struct operacionBasica{
2 1: double a =0 ,
3 2: double b =0,
4 }
```

Figure 5: Declaración de struct en thrift

Luego al declarar las funciones le pasamos simplemente como parámetro ese struct.

```
29 double suma(1:operacionBasica oB),
30 double resta(1:operacionBasica oB),
31 double multiplicacion(1:operacionBasica oB),
```

Figure 6: Declaración de funciones en thrift

Para que thrift genere el código en java hay que poner el siguiente comando **thrift -gen java calculadora.thrift** y para que lo genere en python **thrift -gen py calculadora.thrift**

Ya solo faltaría complementar la función en nuestro server de python y realizar la llamada a la función en nuestro cliente de java (tienen que estar ambos conectados al mismo host-localhost y puerto 9090).

```

def suma(self, oB):
    return oB.a + oB.b

def resta(self, oB):
    return oB.a - oB.b

def multiplicacion(self, oB):
    return oB.a * oB.b

```

Figure 7: Desarrollo de funciones en el servidor python

```

operacionBasica oB= new operacionBasica();
oB.a=5.0;
oB.b=3.5;

System.out.println("Para los números "+oB.a+" y "+oB.b);
double sum = client.suma(oB);
double rest= client.resta(oB);
double mult= client.multiplicacion(oB);
System.out.println("Suma: "+ sum);
System.out.println("Resta: "+ rest);
System.out.println("Multiplicación: "+ mult);

```

Figure 8: Llamada a las funciones desde el cliente java

Para la ejecución en python he tenido que añadir el interpretador de thrift y para java he tenido que añadir la librería jar libthrift y slf4j-api.

4 Gestión de errores

Por último también he implementado gestión de errores para el resto de métodos, por ejemplo en la división se nos avisara de un error en caso de que dividamos un número entre 0, en los vectores avisamos en caso de que el tamaño de ambos vectores no sea igual y en las ecuaciones cuadráticas avisamos de error en caso de que esta no tenga solución.

Para la implementación en thrift he creado un exception con dos variables (tipo de error y el por qué ambos un string).

```
11 exception operacionInvalida {
12 1: string tipo
13 2: string why
14 }
15
```

Figure 9: Declaración exception en thrift

Luego cuando definamos los métodos añadimos un throws de esa exception para que thrift lo tenga en cuenta.

```
32 double division(1:operacionBasica oB) throws (1:operacionInvalida oI),
33 list<double> sumaVectores(1:operacionVectores oV) throws (1:operacionInvalida oI),
34 list<double> restaVectores(1:operacionVectores oV) throws (1:operacionInvalida oI),
35 list<double> multiplicacionVectores(1:operacionVectores oV) throws (1:operacionInvalida oI),
36 list<double> divisionVectores(1:operacionVectores oV) throws (1:operacionInvalida oI),
37 list<double> ecuacionCuadratica(1:segundoGrado sg) throws (1:operacionInvalida oI),
```

Figure 10: Declaración de funciones con gestión de fallos en thrift

Por último en el desarrollo de la función tenemos en cuenta el fallo lanzando un raise (python) tal como en la siguiente imagen.

```
def division(self, oB):
    if oB.b==0:
        raise operacionInvalida('division', 'No se puede dividir un número entre 0, es una indeterminación')
    return oB.a / oB.b
```

Figure 11: Ejemplo de gestión de fallos en python

En el cliente java usamos try-catch de toda la vida para que nos lance el error en caso de que ocurra.

```
try {
    double div = client.division(oB);
    System.out.println("División: " + div);
} catch (operacionInvalida oi) {
    System.out.println("Error " + oi.tipo + ": " + oi.why);
}
```

Figure 12: Try-catch en el cliente java

El ejemplo de ejecución lo podemos observar en la figura 2 que vemos que no se puede realizar la división de ambos vectores ya que no se puede dividir $1/0$, nuestro server es consciente y se notifica al cliente

5 Cosas a tener en cuenta

los parámetros introducidos en el cliente son predefinidos y no por entrada en terminal. Las operaciones combinadas al igual que en la p1 no soportan don operando seguidos, es decir, no se puede poner $1 / - 5$. También he implementado un cliente más mediocre en python por si se quiere probar.