Práctica 2: Calculadora con Apache-Thrift

Leire Requena Garcia

Introducción

Para esta parte se sigue con el planteamiento de la anterior, se ha de desarrollar una calculadora que se llame en un cliente y ejecute las funciones en el servidor; esta vez se usará Apache-Thrift para el desarrollo.

Instalación

He usado como sistema operativo Arch Linux con el kernel 5.10.32-1-lts.

Los paquetes que hay que instalar son `thrift 0.14.0-1` y `python 3.9.3-1`. Además se necesitan como módulos de python `thrift` y `numpy` que se pueden instalar con `pip`.

Calculadora

Lo primero que hay que hacer es defirnir las operaciones y las estructuras de datos en el archivo `calculadora.thrift`:

```
typedef list<i32> vec

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),

    vec sumavector(1:vec v1, 2:vec v2),
    vec restavector(1:vec v1, 2:vec v2),
    vec multiplicacionvector(1:vec v1, 2:vec v2),

    vec multiplicacionvectorescalar(1:vec v, 2:i32 escalar),
}
```

Y en la terminal deberemos ejecutar la siguiente instrucción para que genere la carpeta con los archivos necesarios:

```
> thrift -gen py calculadora.thrift
```

Servidor

Se crea un archivo `servidor.py` dentro de la carpeta que se ha generado. Las operaciones se definirán dentro de una clase `CalculadoraHandler`; este archivo debe importar todos los paquetes de thrift para poder actuar como servidor.

Las operaciones simples se hacen con enteros y son las siguientes:

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

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

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

def division(self, n1, n2):
    print("dividiendo " + str(n1) + " con " + str(n2))
    return int(n1 / n2)
```

El casting a entero en la división es necesario porque si no el programa da un error en el controlador de la conexión.

Para las operaciones con vectores se ha usado numpy para facilitar el cálculo:

```
def sumavector(self, v1, v2):
    print("sumando vector A con vector B")
    vA = np.array(v1)
    vB = np.array(v2)
    return vA + vB

def restavector(self, v1, v2):
    print("restando vector A con vector B")
    vA = np.array(v1)
    vB = np.array(v2)
    return vA - vB

def multiplicacionvector(self, v1, v2):
    print("multiplicando vector A con vector B")
    vA = np.array(v1)
    vB = np.array(v2)
    return vA * vB

def multiplicacionvectorescalar(self, v, escalar):
    print("multiplicando un vector por un escalar")
    return np.array(v) * escalar
```

Al final se inicializa la conexión 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("fin")
```

Cliente

En el cliente tenemos que iniciar la conexión al servidor:

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

client = Calculadora.Client(protocol)

transport.open()
```

Si se han recibido menos de 4 argumentos o el primero de ellos es `-h` se imprime una función de ayuda, en otro caso se continúa con el programa.

El primer argumento es el operando 1, el segundo la operación y el tercero el operando 2. Si ambos operandos son números se harán operaciones básicas, si son una 'v' se harán operaciones con vectores, y si es una 'v' y un número se multiplicará el vector por el escalar.

En el caso de que ambos sean números se llamará a la siguiente función:

```
def opBasicas(cliente, operando1, operacion, operando2):
    resultado = None

    if operacion == '+':
        resultado = cliente.suma(operando1, operando2)
    elif operacion == '-':
        resultado = cliente.resta(operando1, operando2)
    elif operacion == 'x':
        resultado = cliente.multiplicacion(operando1, operando2)
    elif operacion == '/':
        resultado = cliente.division(operando1, operando2)
    else:
        print("Operación con enteros no implementada")

    return resultado
```

Si los dos son vectores se pedirá el tamaño y los elementos de cada vector:

```
vecA = []
vecB = []
tam  = int(input("Introduzca el tamaño de los vectores:"))

for i in range(tam):
    num = int(input("Introduzca un elemento para el primer vector: "))
    vecA.append(num)

for i in range(tam):
    num = int(input("Introduzca un elemento para el segundo vector: "))
    vecB.append(num)
```

Despues se llama a la función que seleccione la operación que manda al servidor:

```
def opVectores(cliente, operando1, operacion, operando2):
    resultado = None
    if operacion == '+':
        resultado = cliente.sumavector(operando1, operando2)
    elif operacion == '-':
        resultado = cliente.restavector(operando1, operando2)
    elif operacion == 'x':
        resultado = cliente.multiplicacionvector(operando1, operando2)
    else:
        print("Operación con vectores no implementada")

    return resultado
```

Si es un vector por un escalar se piden los datos del vector y se llama a la función correspondiente del servidor:

```
v = []
tam  = int(input("Introduzca el tamaño del vector:"))

for i in range(tam):
    num = int(input("Introduzca un elemento para el vector: "))
    v.append(num)

resultado = client.multiplicacionvectorescalar(v, int(operando2))
```

En cualquier otro caso no se hace nada y se imprime 0 como resultado.

Al final del programa debemos cerrar la conexión con `transport.close()`.

Ejecución

Para la ejecución, debemos iniciar una instancia del servidor en una terminal:

```
> python servidor.py
```

Y ejecutar el cliente en otra terminal con las opciones que queramos, dejo algunos ejemplos en imágenes:

```
DSD/P2/calculadora-apache/gen-py on  main [?]
➜ python cliente.py -h
Ejecución: python operando1 operacion operando2
Para operaciones básicas: python num1 operacion num2, siendo num1 y num2 enteros
Para operaciones con vectores: python v operacion v
Para multiplicar un vector con un escalar: python v x num, siendo num un entero
Para ver este mensaje de ayuda: python -h

DSD/P2/calculadora-apache/gen-py on  main [?]
➜ python cliente.py 50 + 12
hacemos ping al server
El resultado es: 62

DSD/P2/calculadora-apache/gen-py on  main [?]
➜ python cliente.py 50 / 12
hacemos ping al server
El resultado es: 4
```

```
DSD/P2/calculadora-apache/gen-py on  main [?]
➜ python cliente.py v x v
hacemos ping al server
Introduzca el tamaño de los vectores:2
Introduzca un elemento para el primer vector: 5
Introduzca un elemento para el primer vector: 4
Introduzca un elemento para el segundo vector: 8
Introduzca un elemento para el segundo vector: 10
El resultado es: [40, 40]
```

```
DSD/P2/calculadora-apache/gen-py on  main [?] took 11s
➜ python cliente.py v x 4
hacemos ping al server
Introduzca el tamaño del vector:3
Introduzca un elemento para el vector: 10
Introduzca un elemento para el vector: 9
Introduzca un elemento para el vector: 56
El resultado es: [40, 36, 224]
```