Memoria explicativa Apache Thrift

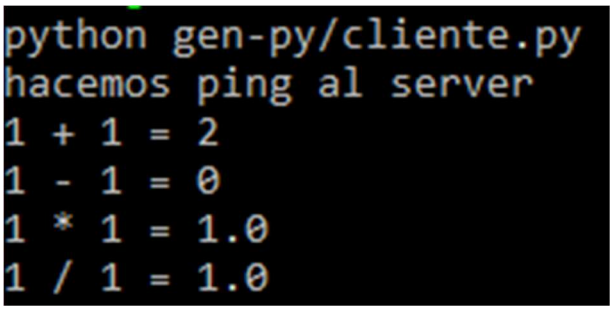
Se nos encargó desarrollar una calculadora distribuida usando la herramienta “Apache Thrift”, a continuación, describimos en este fichero nuestra solución y comentamos algunos detalles. En primer lugar, las funcionalidades más importantes que hemos implementado son:

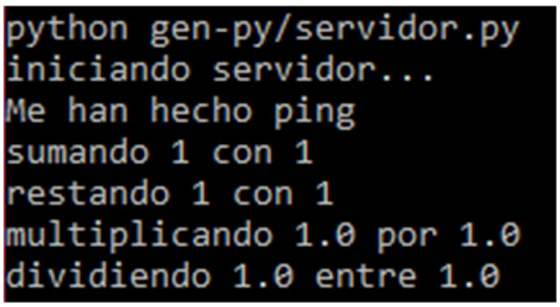
- Operaciones aritméticas básicas
- Operaciones aritméticas con datos complejos
- Operaciones con datos complejos compuestos de otros datos definidos por nosotros
- Comunicación entre cliente y varios servidores
- Comunicación entre distintos lenguajes (Java y Python)

Resumiendo, en esta práctica hemos implementado una calculadora con clientes y servidores en diferentes lenguajes. Llevamos a cabo operaciones básicas y complejas con datos simples y otros más complejos. Hemos desarrollado clientes y servidores en el mismo lenguaje y también en diferentes, es decir, podemos usar un cliente en Python que se conecte a un servidor en Python o en Java y viceversa, de hecho, permitimos todas las combinaciones (Python -> Python, Python -> Java, Java -> Java, Java -> Python). Como funcionalidad añadida, permitimos que un cliente se conecte a varios servidores diferentes a la vez, es decir, podemos ejecutar la operación “suma” en un servidor Python y la operación “resta” en un servidor Java, por ejemplo, algo ideal dependiendo de que operaciones queramos ejecutar y de qué lenguajes nos ofrezcan mayores rendimientos (cálculo intensivo en C++, Complejos scripts en Python, etc).

Operaciones básicas

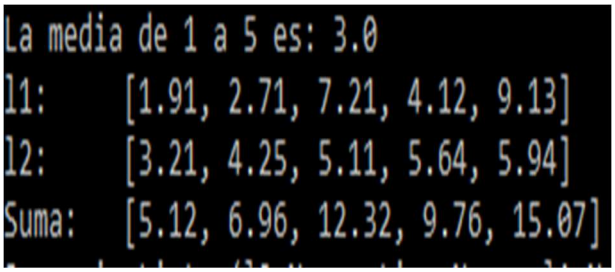
Suma, resta, multiplicación y división de elementos. A la derecha el servidor en Python, a la izquierda el cliente en Python.

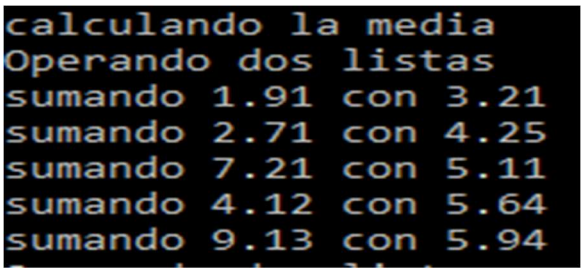




Operaciones con datos complejos

Media de una lista de valores, sumar dos listas. Se usan funciones que reciben como parámetros una lista o dos respectivamente.





```
typedef list<double> MiLista
typedef list<MiLista> MiMatriz

enum TipoOperacion {
    SUMA            = 0,
    RESTA           = 1,
    MULTIPLICACION  = 2,
    DIVISION        = 3,
    MEDIA           = 4
}

struct OperacionListas {
    1: MiLista l1,
    2: MiLista l2,
    3: TipoOperacion tipo,
}
```

Operaciones con datos complejos compuestos de otros datos definidos por nosotros

Usamos la operación OperaListas la cual recibe un dato de tipo OperacionListas. Dicho dato se encuentra formado por dos listas de tipo MiLista (definido por nosotros) así como el tipo de operación (también definido por nosotros).

A continuación, mostramos un caso de ejecución en el que cambiando solamente el atributo tipo del dato OperacionListas, podemos llamar a una sola función para realizar, sobre dos listas, diferentes operaciones en un servidor remoto.

Cliente arriba y servidor abajo.

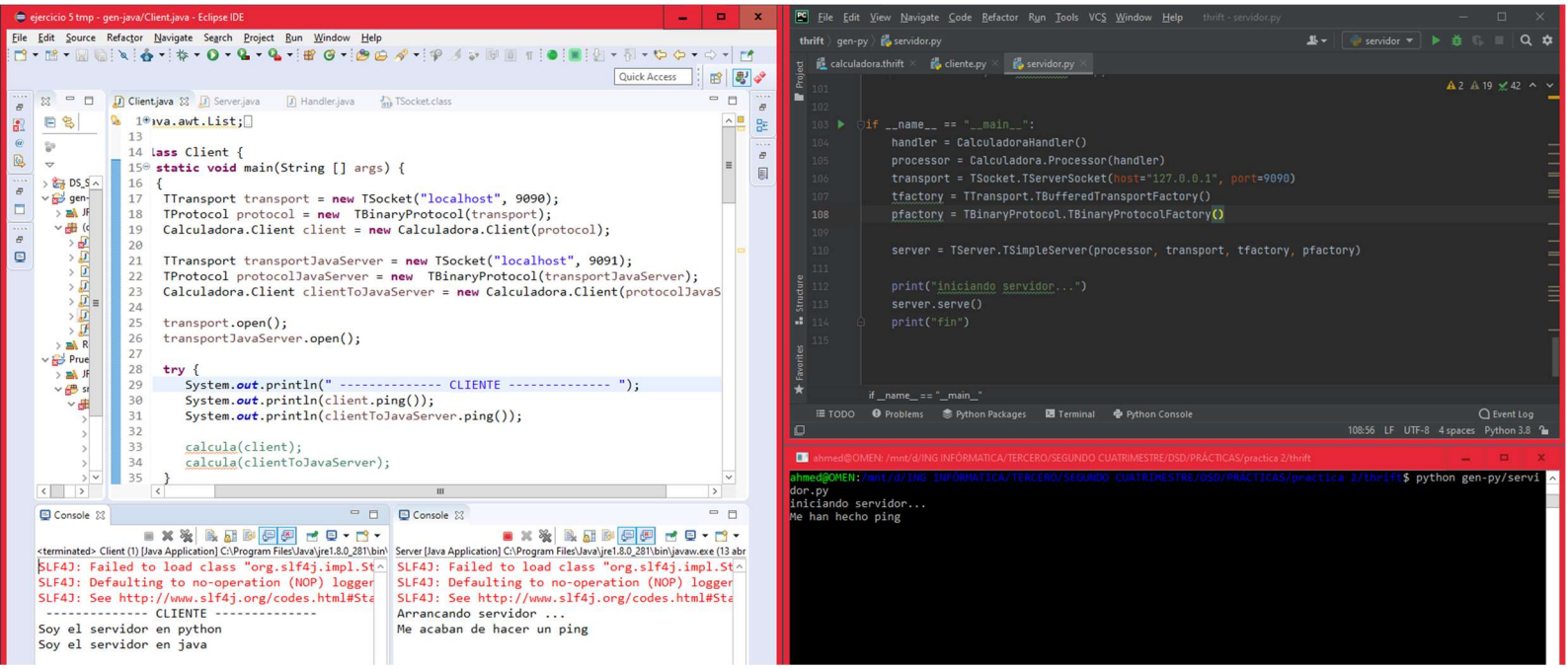
```
Operacion suma: [5.12, 6.96, 12.32, 9.76, 15.07]
Operacion resta: [-1.3, -1.54, 2.0999999999999996, -1.5199999999999996, 3.1900000000000004]
Operacion multiplicacion: [6.1311, 11.5175, 36.8431, 23.2368, 54.232200000000006]
Operacion division: [0.5950155763239875, 0.6376470588235295, 1.410958904109589, 0.7304964539007093, 1.537037037037037]
```

Como podemos apreciar en la captura de la derecha, al operar dos listas las operaciones que internamente usamos elemento a elemento son las operaciones básicas, esto es un ejemplo de reutilización de código y también de que el servidor puede llamar a sus funciones desde otras como si fuese una clase más.

```
Operando dos listas
sumando 1.91 con 3.21
sumando 2.71 con 4.25
sumando 7.21 con 5.11
sumando 4.12 con 5.64
sumando 9.13 con 5.94
Operando dos listas
restando 1.91 con 3.21
restando 2.71 con 4.25
restando 7.21 con 5.11
restando 4.12 con 5.64
restando 9.13 con 5.94
Operando dos listas
multiplicando 1.91 por 3.21
multiplicando 2.71 por 4.25
multiplicando 7.21 por 5.11
multiplicando 4.12 por 5.64
multiplicando 9.13 por 5.94
Operando dos listas
dividiendo 1.91 entre 3.21
dividiendo 2.71 entre 4.25
dividiendo 7.21 entre 5.11
dividiendo 4.12 entre 5.64
dividiendo 9.13 entre 5.94
```

Comunicación entre cliente y varios servidores

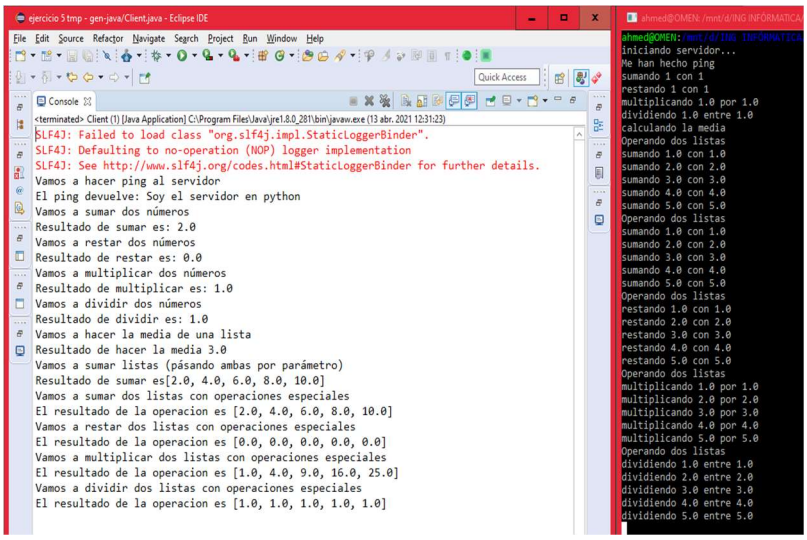
Para llevar a cabo esta comunicación basta con definir varios servidores en diferentes puertos (podríamos cambiar también la ip, aunque en nuestro caso será localhost porque no disponemos de otra PC). En el fichero cliente definimos dos client, uno que se conecte al puerto 9090 y otro al 9091 (como ejemplo, cualquier otro puerto mayor a 1024 valdría).



A la izquierda vemos eclipse ide para java, en él se ejecutan el cliente en java y el servidor en java (puerto 9091). A la derecha podemos ver el IDE pycharm, en él se aprecia el código de la función main del servidor en Python. Justamente debajo de Pycharm, tenemos una terminal en la que se ejecuta el servidor en Python. Desde el cliente en java abrimos dos conexiones, y llamamos a la función ping con los clientes de dichas conexiones. Dichos métodos ping nos devuelven un mensaje del servidor si todo salió correcto y podemos ver en el ejemplo como desde un cliente en java hemos ejecutado una función en dos servidores a la vez y sin importar el lenguaje en el que estuviesen programados.

Comunicación entre distintos lenguajes (Java y Python)

Del ejemplo anterior se deriva este apartado, es decir, se puede comprobar en el apartado anterior que independientemente del lenguaje del cliente y del servidor la comunicación es exitosa. Parece un suceso trivial tener esta abstracción de lenguajes, pero es algo realmente potente y útil ya que podemos aprovechar las características de diferentes lenguajes, que además se ejecutan en servidores remotos, para nuestro programa cliente.



Observaciones finales

Hemos implementado mecanismos de control de errores, se puede echar un vistazo a su funcionamiento en el archivo con extensión “Thrift”. Además, en el cliente en Python disponemos de secciones comentadas en las que podemos forzar el error y ver cómo lo resuelven los mecanismos descritos previamente.

Uso de IDEs como Pycharm ha sido bastante interesante ya que es capaz de detectarnos errores en tiempo real que de otra manera tal vez no sería fácil detectar (sobre todo si no se conoce Python en profundidad o es la primera vez que lo usamos) por lo que se debería incentivar a los alumnos a hacer uso de esta herramienta.

La realización de esta práctica no ha sido muy complicada, la parte que más ha costado ha sido entender el funcionamiento de Thrift como tal y sobre todo resolver las dependencias de bibliotecas y otros ficheros necesarios. En mi caso he optado por descargar las bibliotecas manualmente y añadirlas al path del sistema para poder usarlas sin problemas.

Algunos enlaces que me han sido de mucha utilidad han sido los siguientes:

- https://gitbox.apache.org/repos/asf?p=thrift.git;a=tree;f=tutorial;hb=HEAD
- https://gitbox.apache.org/repos/asf?p=thrift.git;a=tree;f=tutorial/py;h=fd2ea290f807556c85905805267f3469688ed1aa;hb=HEAD
- https://gitbox.apache.org/repos/asf?p=thrift.git;a=tree;f=tutorial/java/src;h=d56632301e9f9ff6c4ae5a5fd0f09bbb1c481ad3;hb=HEAD
- https://download.jar-download.com/cache_jars/org.apache.thrift/libthrift/0.14.1/jar_files.zip
- https://pypi.org/project/thrift/#files