

Introducción a Apache Thrift

Desarrollo de Sistemas Distribuidos

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada



Curso 2021 – 2022

Indice


- 1 Introducción a IDLs
- 2 Tipos en Thrift
- 3 Servicios
- 4 Pasos para usar Thrift
- 5 Ejemplo completo en Python
- 6 Implementar servidor
- 7 Implementar cliente
- 8 Bibliografía

Sección 1 | Introducción a IDLs

Motivación

- Distintos lenguajes: con sus tipos y definiciones
- Es complicado el paso de mensajes por la red
- Solución: lenguaje intermedio para definir los datos (IDL: Interface Description Language)
- Con un compilador que genere los tipos para cada lenguaje

Apache Thrift



```

1 namespace java com.example.gen
2 namespace rb Com.Example.Gen
3
4 enum ServerInfoKey {
5     TIME = 1;
6     MEM = 2;
7 }
8
9 struct ReqServerInfo {
10     1:ServerInfoKey key;
11 }
12
13 struct ResServerInfo {
14     1:string value;
15 }
16
17 exception ServerInfoException {
18     1:list<string> traces;
19 }
20
21 service Environment
22 {
23     ResServerInfo getServerInfo(1:ReqServerInfo req) throws (1:ServerInfoException e);
24 }

```

```

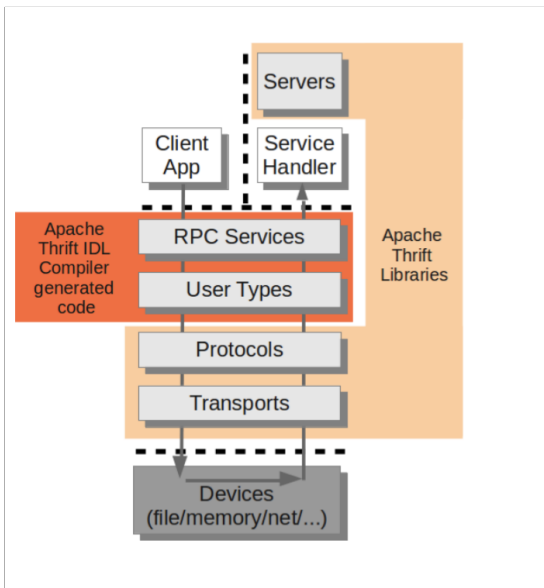
1 // Example generated code
2 import com.example.gen.ServerInfoException;
3
4 import org.apache.thrift.protocol.TBinaryProtocol;
5 import org.apache.thrift.server.TServlet;
6
7 public class EnvironmentServlet extends TServlet {
8     {name = "EnvironmentServlet", uriPatterns = { "/environment" }}
9     public EnvironmentServlet() {}
10
11     public ResServerInfo getServerInfo(ReqServerInfo req) throws ServerInfoException {
12         // ...
13         value = String.valueOf(Runtime.getRuntime().totalMem());
14         break;
15         case TIME:
16             value = String.valueOf(new Date());
17             break;
18         default:
19             throw new IllegalArgumentException("Not found: key = " + key);
20     }
21     catch (Exception e) {
22         StackTraceElement[] sts = e.getStackTrace();
23         List<String> traces = new ArrayList<>(sts.length);
24         for (StackTraceElement st : sts) {
25             // ...
26         }
27     }
28 }

```

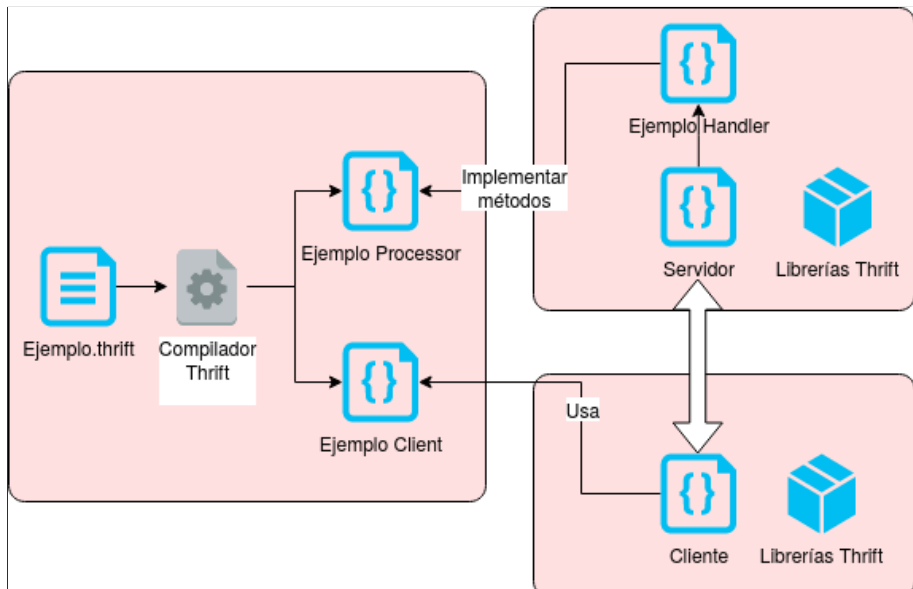
Apache Thrift

- Desarrollado por Facebook en 2007
- Útil para transmisión de datos binarios (más rápido que REST y similares)
- Multilenguaje
- Está compuesto de:
 - 1 Compilador que parte de un archivo .thrift y genera código para muchos lenguajes
 - 2 Librerías para ejecutar ese código en esos lenguajes (jars, paquetes python, ruby gems...)

Arquitectura de Apache Thrift



Apache Thrift: funcionamiento



Instalación del Compilador Thrift

Mac

- Instalar brew
- `brew install thrift`

Linux (Ubuntu, Debian)

- (Buscar el equivalente en vuestra distro)
- `sudo apt-get install thrift-compiler`

Windows

- <https://thrift.apache.org/docs/install/windows>

Sección 2 | Tipos en Thrift

Tipos en Thrift

- Básicos: bool, i16, i32, i64, double, string, binary.
- Contenedoras: list<tipo>, set<tipo>, map<tipo>.
- Enumerados, como C.
- Estructuras, con campos numerados

Ejemplo

Enumerado

```
enum TweetType {  
  TWEET,  
  RETWEET = 2,  
  REPLY}
```

Estructura

```
struct Tweet {  
  1: required i32 userId;  
  2: required string userName;  
  3: required string text;  
  4: optional Location loc;  
  5: optional TweetType tweetType = TweetType.TWEET  
  6: optional string language = "english"}
```

Sección 3 | Servicios

Servicios

- Define una lista de métodos
- Estos son los métodos a implementar por el servidor, y a utilizar desde el cliente
- Pueden lanzar excepciones
- Pueden ser síncronos o asíncronos

Ejemplo de Servicio

```
service Twitter {  
    void ping(),  
    bool postTweet(1:Tweet tweet)  
        throws (1:TwitterUnavailable unavailable),  
    TweetSearchResult searchTweets(1:string query);  
}
```

Sección 4 | Pasos para usar Thrift

Proceso para usar Thrift

- Generar el fichero IDL (por ejemplo calculadora.thrift)
- Compilar el fichero al lenguaje a utilizar Ejemplo para python:
`thrift -gen py calculadora.thrift`
- Se generarán muchos ficheros para usar por el cliente y el servidor
- Implementar Servidor
 - Instalar/importar paquete thrift en el lenguaje (distinto del compilador)
 - Importar clases generadas
 - Crear una clase handler e implementar los métodos del servicio
 - Crear el objeto server
 - Arrancarlo
- Implementar Cliente
 - Instalar/importar paquete thrift en el lenguaje (distinto del compilador)
 - Importar clases generadas
 - Crear un objeto cliente
 - Llamar a los métodos del cliente (llamará al servidor por dentro)

Sección 5 | Ejemplo completo en Python

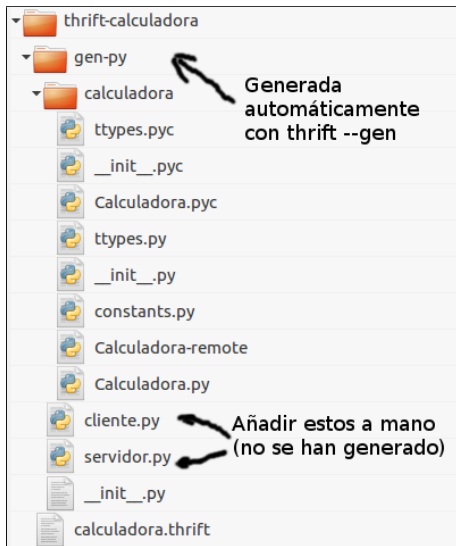
Paso 1: Escribir fichero calculadora.thrift

```
service Calculadora{  
    void ping(),  
    i32 suma(1:i32 num1, 2:i32 num2),  
    i32 resta(1:i32 num1, 2:i32 num2),  
}
```

Paso 2: Generar ficheros

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

Paso 2: Generar ficheros



Paso 3: Instalar paquetes de Thrift para Python

Dependiendo del lenguaje que useis habrá que hacer una cosa u otra. Por ejemplo en java habrá que añadir thrift.jar a las librerías del proyecto

- `pip install thrift`
- `python -m pip install thrift`
- O instalarlo desde Pycharm/virtualenv/conda...

(probar con sudo si no tirara)

Sección 6 | Implementar servidor

Paso 4.1: Implementar servidor.py (importar cosas)

```
import glob
import sys

from calculadora import Calculadora
#from calculadora.ttypes import Operation
#Lo de ttypes es si hubieramos anadido tipos en el fichero.thrift

#hay que instalar antes el paquete thrift de python
#(no confundir con el compilador thrift)
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from thrift.server import TServer

import logging
logging.basicConfig(level=logging.DEBUG)
#Esto es para imprimir cuando haya errores en el
#servidor y poder depurar!
```


Paso 4.2: Implementar servidor.py (implementar handler)

```
class CalculadoraHandler:
    def __init__(self):
        self.log = {}

    def ping(self):
        print("Me han hecho ping()")

    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
```

Paso 4.3: Implementar servidor.py (lanzar 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("done.")
```

¿Como sería el servidor en Java? Muy parecido

```
//Importar cosas
import org.apache.thrift.server.TServer;
import org.apache.thrift.server.TSimpleServer;
import org.apache.thrift.transport.TServerSocket;
import org.apache.thrift.transport.TServerTransport;

import tutorial.*;
...
//Implementar clase handler (en este u en otro fichero)
class CalculadoraHandler implements Calculadora.Iface{
    public void ping(){System.out.println("Me han hecho ping");}
    public int sumar(int a, int b){return a+b;}
    ...
}
//Lanzar el servidor en el static void main()
try {
    TServerTransport serverTransport = new TServerSocket(9090);
    TServer server = new TSimpleServer(new Args(serverTransport).
        processor(processor));

    System.out.println("Iniciando servidor...");
    server.serve();
} catch (Exception e) { e.printStackTrace(); }
```

Sección 7 | Implementar cliente

Paso 5.1: Implementar cliente.py (importar cosas)

```
from calculadora import Calculadora

from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
```

Paso 5.2: Implementar cliente.py (crear objeto cliente)

```
transport = TSocket.TSocket("localhost", 9090)
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
#creamos el cliente
client = Calculadora.Client(protocol)
```

Paso 5.3: Implementar cliente.py (usar objeto cliente)

```
transport.open()

print("Hacemos ping al server")
client.ping()

resultado = client.suma(1, 1)
print("1+1="+str(resultado))
resultado = client.resta(1, 1)
print("1-1="+str(resultado))

transport.close()
```

Ejecutar

```
pgarcia@evorq: ~/code/thrift-calculadora/gen-py
pgarcia@evorq:~/code/thrift-calculadora/gen-py$ python servidor.py
Iniciando servidor...
Me han hecho ping()
sumando 1 con 1
restando 1 con 1
```

```
pgarcia@evorq: ~/code/thrift-calculadora/gen-py
pgarcia@evorq:~/code/thrift-calculadora/gen-py$ python cliente.py
Hacemos ping al server
1+1=2
1-1=0
pgarcia@evorq:~/code/thrift-calculadora/gen-py$
```


Sección 8 | Bibliografía

Bibliografía

- Apuntes basados en el trabajo de Daniel Molina en la UCA
- <https://www.tutorialspoint.com/python/index.htm>
- <https://thrift.apache.org/>
- <https://www.practicepython.org/>