

分布式系统作业

第 3 次作业

姓名：郝裕玮

班级：计科 1 班

学号：18329015

一、问题描述

使用 protobuf 和 gRPC 远程过程调用的方法实现 Client-Server 系统，Server 提供简单的算数操作如加和等，Client 通过 RPC 向 server 发送请求，Server 返回计算结果。选做功能：Server 能够控制访问请求的数量，以及实现请求超时终止的。

二、解决方案

(1) 本次实验环境为 Windows10

(2) 在 cmd 窗口中执行以下代码，配置环境：

```
pip install grpcio
pip install protobuf
pip install grpcio-tools
```

(3) 桌面新建 proto 文件夹并在文件夹中新建 msg.proto，文件内容如下（具体分析已全部包含在代码注释中）：

```
syntax = "proto3";
//规定语法，这里使用的是 proto3 的语法
//使用 service 关键字定义服务

service MsgService {
    rpc GetMsg (MsgRequest) returns (MsgResponse){}
    //简单 RPC，即客户端发送一个请求给服务端，从服务端获取一个应答，就像一次普通的函数调用
}

//定义 message 内部需要传递的数据类型
message MsgRequest {
    float a = 1; //运算数 1
    string op = 2; //运算符号
    float b = 3; //运算数 2
    //消息定义中，每个字段都有唯一的一个数字标识符
    //这些标识符是用来在消息的二进制格式中识别各个字段的，一旦开始使用就不能够再改变
}

message MsgResponse {
```







```
float result = 1; //算数结果
}
```

(4) 在 proto 文件夹中打开 cmd 窗口，执行如下代码对 msg.proto 进行编译运行：

```
python -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=. msg.proto
```

```
C:\Users\93508\Desktop\proto>python -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=. msg.proto
C:\Users\93508\Desktop\proto>
```

(5) 编译运行后发现 proto 文件夹中多出两个文件：

	__pycache__	2021/11/10 15:34
	client	2021/11/10 17:05
	msg	2021/11/10 16:42
	msg_pb2	2021/11/10 17:03
	msg_pb2_grpc	2021/11/10 17:03
	server	2021/11/10 17:06

这两个文件是为后续的客户端和服务端所用（包含了许多可调用的变量，类和函数）。

(6) 在 proto 文件夹中新建 client.py 和 server.py，代码具体如下（具体思路 and 详细分析均已包含在代码注释中）：

client.py:

```
from __future__ import print_function
import grpc
import msg_pb2
import msg_pb2_grpc

def run():
    # 客户端很好理解,网络连接得到一个 channel,拿 channel 去实例化一个 stub,通过 stub 调用 RPC 函数
    channel = grpc.insecure_channel('localhost:50051')
    # 使用 grpc.insecure_channel('localhost:50051')进行连接服务端,接着在这个 channel 上创建 stub
    stub = msg_pb2_grpc.MsgServiceStub(channel)
```

```

    # 在 msg_pb2_grpc 里可以找到 MsgServiceStub 这个类相关信息。这个 stub 可以
    调用远程的 GetMsg 函数

    a1 = float(input());# 输入运算数 1, 并转为 float 型
    op1 = input();# 输入运算符类型
    b1 = float(input());# 输入运算数 2, 并转为 float 型

    response = stub.GetMsg(msg_pb2.MsgRequest(a=a1,op=op1,b=b1))#
    response 为服务器端发来的内容
    # MsgRequest 中的内容即 msg.proto 中定义的数据。在回应里可以得到
    msg.proto 中定义的 msg

    msg = "服务器端运算结果为: {}\n".format(response.result)#打印从服务器端
    接收到的数据, 即运算结果
    print(msg)

if __name__ == '__main__':
    run()

```

server.py:

```

from concurrent import futures
import time
import grpc
import msg_pb2
import msg_pb2_grpc
_ONE_DAY_IN_SECONDS = 60 * 60 * 24
# 导入 RPC 必备的包, 以及刚才生成的两个文件(grpc,msg_pb2,msg_pb2_grpc)
# 因为 RPC 应该长时间运行, 考虑到性能, 还需要用到并发的库(time,concurrent)

# 在服务器端代码中需要实现 proto 文件中定义的服务接口 (MsgService), 并重写处理
函数 (GetMsg)
# Python gRPC 的服务实现是写一个子类去继承 proto 编译生成的
userinfo_pb2_grpc.UserInfoServicer
# 并且在子类中实现 RPC 的具体服务处理方法, 同时将重写后的服务类实例化以后添加到
grpc 服务器中

class MsgService(msg_pb2_grpc.MsgServiceServicer):
    # 工作函数
    def GetMsg(self, request, context):
        # 在 GetMsg 中设计 msg.proto 中定义的 MsgResponse
        # 对收到的 request 的内容进行读取并根据 op 内容执行相关运算操作
        if(request.op == '+'):

```

```

        res=request.a+request.b
    if(request.op == '-'):
        res=request.a-request.b
    if(request.op == '*'):
        res=request.a*request.b
    if(request.op == '/'):
        res=request.a/request.b

    # 在服务器端打印从客户端收到的内容并打印结果，用于检验接收和计算的结果
    是否正确
    msg = "需要计算的式子为: {}{}{}, 结果为
    {}{}\n".format(request.a,request.op,request.b,res)
    print(msg)

    # 将结果返回给客户端
    return msg_pb2.MsgResponse(result = res)

# 通过并发库，将服务端放到多进程里运行
def serve():
    # gRPC 服务器
    # 定义服务器并设置最大连接数,concurrent.futures 是一个并发库，类似于线程
    池的概念
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=2))# 创
    建一个服务器 server
    msg_pb2_grpc.add_MsgServiceServicer_to_server(MsgService(), server)#
    在服务器中添加派生的接口服务（自己实现的处理函数）
    server.add_insecure_port('[::]:50051')# 添加监听端口
    print("服务器已打开，正在等待客户端连接...\n")
    server.start() # 启动服务器，同时 start()不会阻塞，如果运行时无事发生，则
    循环等待
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)# 关闭服务器
if __name__ == '__main__':
    serve()

```

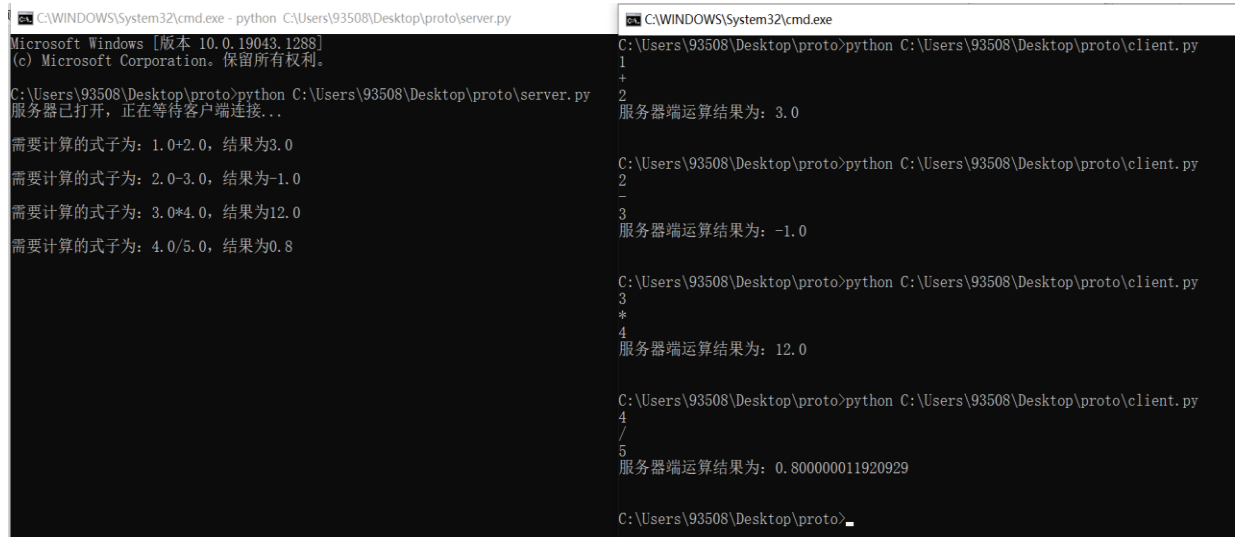
三、实验结果

在 proto 文件夹中打开 2 个 cmd 窗口，分别执行如下命令：

```
python C:\Users\93508\Desktop\proto\client.py
```

```
python C:\Users\93508\Desktop\proto\server.py
```

先运行服务器端，再运行客户端，运行结果如下：



```
C:\WINDOWS\System32\cmd.exe - python C:\Users\93508\Desktop\proto\server.py
Microsoft Windows [版本 10.0.19043.1288]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\93508\Desktop\proto>python C:\Users\93508\Desktop\proto\server.py
服务器已打开，正在等待客户端连接...

需要计算的式子为: 1.0+2.0, 结果为3.0
需要计算的式子为: 2.0-3.0, 结果为-1.0
需要计算的式子为: 3.0*4.0, 结果为12.0
需要计算的式子为: 4.0/5.0, 结果为0.8

C:\WINDOWS\System32\cmd.exe
C:\Users\93508\Desktop\proto>python C:\Users\93508\Desktop\proto\client.py
1
+
2
服务器端运算结果为: 3.0

C:\Users\93508\Desktop\proto>python C:\Users\93508\Desktop\proto\client.py
2
-
3
服务器端运算结果为: -1.0

C:\Users\93508\Desktop\proto>python C:\Users\93508\Desktop\proto\client.py
3
*
4
服务器端运算结果为: 12.0

C:\Users\93508\Desktop\proto>python C:\Users\93508\Desktop\proto\client.py
4
/
5
服务器端运算结果为: 0.800000011920929

C:\Users\93508\Desktop\proto>
```

由上图可知两边结果一致，本次实验圆满完成！

四、遇到的问题及解决方法

在编辑服务器端代码时，变量名未和 proto 文件里的变量保持一致，写成了如下的代码：

server.py

```
# 在服务器端打印从客户端收到的内容并打印结果，用于检验接收和计算的结果是否正确
msg = "需要计算的式子为: {}{}{}，结果为
{}\n".format(request.a,request.name,request.b,res)
print(msg)
```

msg.proto

```
//定义 message 内部需要传递的数据类型
message MsgRequest {
    float a = 1; //运算数 1
```

```

string op = 2; //运算符
float b = 3; //运算数 2
//消息定义中, 每个字段都有唯一的一个数字标识符
//这些标识符是用来在消息的二进制格式中识别各个字段的, 一旦开始使用就不能够
再改变
}

```

实际上 server 中的 request.name 应该修改为 request.op, 否则会出现如下错误:

```

Traceback (most recent call last):
  File "C:\Users\93508\Desktop\proto\greeter_client.py", line 22, in <module>
    run()
  File "C:\Users\93508\Desktop\proto\greeter_client.py", line 18, in run
    response = stub.GetMsg(msg_pb2.MsgRequest(a=aa, op=opp, b=bb))
  File "C:\Users\93508\AppData\Local\Programs\Python\Python37\lib\site-packages\grpc\_channel.py", line 946, in __call__
    return _end_unary_response_blocking(state, call, False, None)
  File "C:\Users\93508\AppData\Local\Programs\Python\Python37\lib\site-packages\grpc\_channel.py", line 849, in _end_unary_response_blocking
    raise _InactiveRpcError(state)
grpc._channel._InactiveRpcError: <InactiveRpcError of RPC that terminated with:
  status = StatusCode.UNKNOWN
  details = "Exception calling application: name"
  debug_error_string = "{"created": "@1636528085.056000000", "description": "Error received from peer ipv6;[:1]:50000", "file": "src/core/lib/surface/call.cc", "file_line": 1070, "grpc_message": "Exception calling application: name", "grpc_status": 2}"
>

```