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

1. 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 {

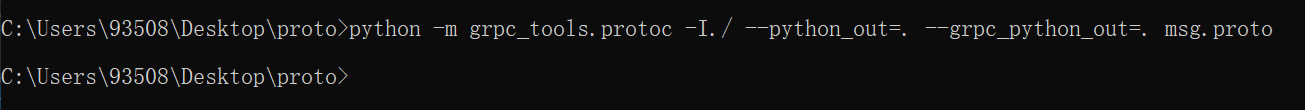
    float result = 1;//算数结果

}

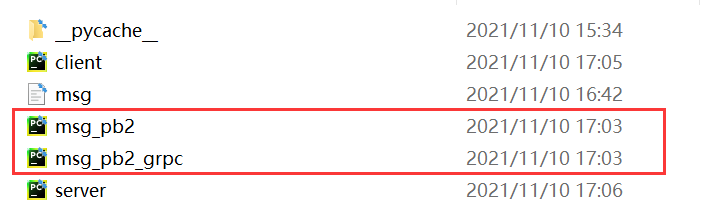
（4）在proto文件夹中打开cmd窗口，执行如下代码对msg.proto进行编译运行：

python -m grpc\_tools.protoc -I./ --python\_out=. --grpc\_python\_out=.

msg.proto



（5）编译运行后发现proto文件夹中多出两个文件：



这两个文件是为后续的客户端和服务器端所用（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（具体思路和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();# 输入运算符号类型

    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,corcurrent.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()

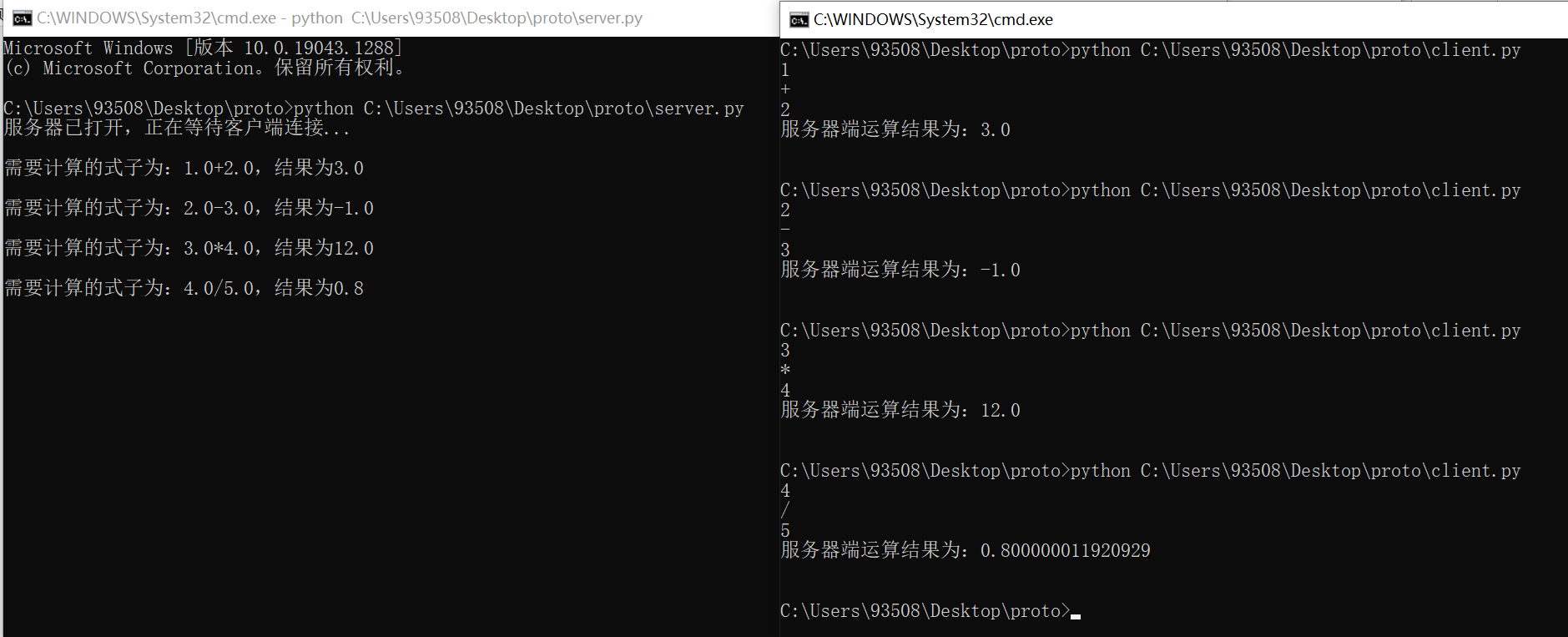
三、实验结果

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：



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

}

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

