分布式系统作业

消息订阅系统实现

第三次作业

姓名：曾慧蕾

班级：系统结构班

学号：21307358

1. 问题描述

使用protobuf和gRPC等远程过程调用的方法实现消息订阅(publish-subscribe)系统，该订阅系统能够实现简单的消息传输，还可以控制消息在服务器端存储的时间

编程语言不限，但是推荐使用python和C

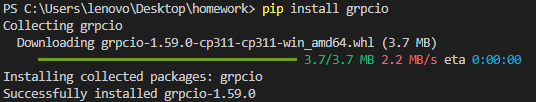
1. 解决方案
2. 准备工作

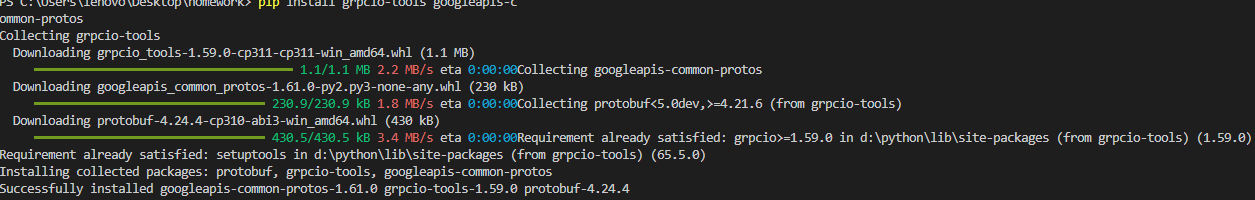
本次实验在window10环境下，使用python语言完成。因此先考虑安装环境。在python终端下输入如下命令，安装好grpc环境。

pip install grpcio

pip install grpcio-tools googleapis-common-protos

配置好后显示如下：





1. protobuf文件编写与处理

根据参考文件中的proto文件格式以及本实验要求的功能，编辑文件pubsub.proto如下所示：

syntax = "proto3";

service Pubsub {

    // 发布消息

    rpc publish(PubRequest) returns (reply) {}

    // 订阅消息 使用响应流 可一次处理多个消息

    rpc subcribe(SubRequest) returns (stream reply) {}

}

message PubRequest{

    string topic = 1; //消息主题

    string context = 2; //消息内容

}

message SubRequest{

    string topic = 1; //消息主题

    string clientID = 2; // 客户端ID

    int32 TTL = 3; // 存储时间

}

message reply{

    string message = 1; //消息 响应

}

随后，为获得所需的grpc\_pb2\_grpc.py以及grpc\_pb2.py文件，使用命令行进入该文件目录下（C:\Users\lenovo\Desktop\homework\proto），并输入如下命令：

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

其中，各参数含义如下：

-m grpc\_tools.protoc 使用grpc\_tools.protoc的库模块

-I= <proto文件所在目录>

--python\_out=<生成代码的目录>

--grpc\_python\_out=<生成代码的目录>

最后为<proto文件路径>

最终生成文件如下所示：



1. 服务器程序编写

服务器需要从第二步中生成的pubsub\_pb2.py和pubsub\_pb2\_grpc.py中获得消息的定义以及发送信息封装好的函数，因此首先导入这两个文件。

先确定好pubsub类的框架：

class Pubsub(object):

    def \_\_init\_\_(self): #初始化

        self.storage = {}

        self.event = {}

    def publish(self, topic, message): #发布消息

        ……

    def refresh(self, TTL = 10): #更新时间 超时则删除

        ……

    def subcribe(self, topic, clientID, TTL = 10): #订阅某消息

        ……

publish：

    def publish(self, topic, message): #发布消息

        new\_message = ''

        add\_message = {'create time':time.time(), 'message': message}

        if topic not in self.storage: #存储发布消息

            self.storage[topic] = [add\_message]

            new\_message += f"create topic:{topic}\n"

        else:

            self.storage[topic].append(add\_message)

        new\_message += 'successfully publish'

        if topic in self.event: #检查是否有客户端订阅消息

            for client in self.event[topic]:

                self.event[topic][client].set() #将该事件设为已设置（线程通信）

        return new\_message

refresh：

    def refresh(self, TTL = 10): #更新时间 超时则删除

        t = time.time() - TTL

        for topic in self.storage: # 检查消息库里的所有消息

            while len(self.storage[topic]) and self.storage[topic][0]['create time'] <= t:

                del self.storage[topic][0] #在该主题下有消息且该消息时间过久时 删除

subcribe：

    def subcribe(self, topic, clientID, TTL = 10): #订阅某消息

        T1 = time.time()

        T2 = TTL

        if topic not in self.event: #该订阅行为未被创建过 现创建

            self.event[topic] = {}

        self.event[topic][clientID] = Event() #创建事件

        while T2 > 0:

            if topic in self.storage and len(self.storage[topic]) > 0:

                m = self.storage[topic][-1] #最新的一条消息

                msg = str(m['create time']) + ":" + m['message']

                yield msg

                self.event[topic][clientID].clear() # 确保循环结束时状态为unset

            else:

                self.event[topic][clientID].wait(timeout=1) #等待1秒

                T2 = TTL - (time.time() - T1) #更新剩余时间

        return None #超时仍未收到消息，则返回none

接下来实现PubsubServer服务器，用以实现gRPC服务端。

PubsubServer类框架如下所示：

class PubsubServer(proto.pubsub\_pb2\_grpc.Pubsub): #实现pubsub.proto中的功能

    def \_\_init\_\_(self):

        self.pubsub = Pubsub()

    def publish(self):

        ……

    def subcribe(self):

        ……

Publish函数：

    def publish(self, request):

        result = self.pubsub.publish(request.topic, request.context)

        response = proto.pubsub\_pb2.reply(message = result)

        return response

subcribe函数：

    def subcribe(self, request):

        for result in self.pubsub.subcribe(request.topic, request.clientID, request.TTL):

            response = proto.pubsub\_pb2.reply(message = result)

            yield response

最后实现serve（）函数，函数中要创建一个gRPC服务器，在指定端口（50051）上监听客户端连接请求，同时通过PubsubServer实例来实现消息订阅系统的功能。

\_PORT = '50051'

def serve():

    pubsubserver = PubsubServer()

    server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10)) # 创建grpc服务器，最大工作线程数为10

    proto.pubsub\_pb2\_grpc.add\_PubsubServicer\_to\_server(pubsubserver, server) #将pubsubserver加入到服务器中

    server.add\_insecure\_port('[::]:'+\_PORT) #服务器上添加端口 监听

    server.start()

    try:

        while True:

            time.sleep(1)

            pubsubserver.pubsub.refresh()

    except KeyboardInterrupt:

        server.stop(0)

1. 客户端程序编写

客户端需要从第二步中生成的pubsub\_pb2.py和pubsub\_pb2\_grpc.py中获得消息的定义以及发送信息封装好的函数，因此首先导入这两个文件。

然后，定义好服务器主机名和端口号，创建gRPC通道与存根，用于调用服务器上定义的 gRPC 服务。

\_HOST = 'localhost'

\_PORT = '50051'

clientID = input("please enter an ID:")

channel = grpc.insecure\_channel(\_HOST + ':' + \_PORT)

stub = proto.pubsub\_pb2\_grpc.PubsubStub(channel) #存根 封装

client端代码框架如下：

def publish(topic, context):

    pass

def subcribe(topic, clientID, TTL = 10):

    pass

def receive(topic, clientID, TTL):

    pass

publish函数：

def publish(topic, context):

    msg1 = f"Publish message in {topic}:{context}"

    rsp = proto.pubsub\_pb2.PubRequest(topic = topic, context = context)

    response = stub.publish(rsp)

    print(msg1,"\n",response.message)

subcribe函数：

def subcribe(topic, clientID, TTL = 10):

    msg1 = f"Successfully subcribe from {topic}"

    thread = threading.Thread(target=receive, args=(topic, clientID, TTL))

    print(msg1)

    thread.start()

receive函数：

def receive(topic, clientID, TTL):

    msg1=''

    msg2=''

    for msg in stub.subcribe(proto.pubsub\_pb2.SubRequest(topic = topic, clientID = clientID, TTL = TTL)):

        msg1 = f"Receive message from {topic}:{msg.message}"

        if msg1 == msg2: #防止重复信息无限输出

            break

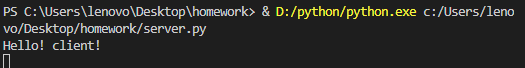
        print(msg1)

        msg2=msg1

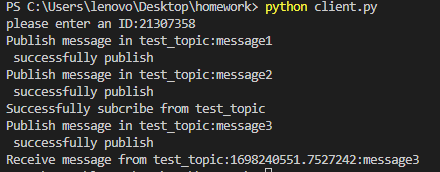
1. 实验结果

打开两个终端，先运行服务器端后运行客户端，分别得到结果如下：

服务器端：

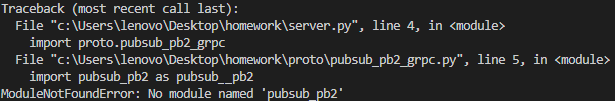


客户端：

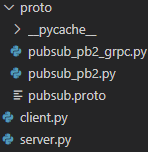


符合预期结果

1. 遇到的问题及解决方法
2. 在刚开始运行server时，一直报错pubsub\_pb2\_grpc.py文件无法找到pubsub\_pb2 模块。具体报错信息如下：



文件结构为：



在这个文件结构下出现导入错误的原因可能是因为 Python 解释器在搜索模块时，无法正确地找到模块的路径。在该文件结构中，proto 文件夹是一个自定义的模块，而 pubsub\_pb2\_grpc.py 和 pubsub\_pb2.py 是该模块中的子模块。为了正确导入子模块，Python 解释器需要能够找到模块的路径。

解决方案：

1. 将 proto 文件夹的路径添加到搜索路径中

import sys

sys.path.append('path/to/proto')

1. 在pubsub\_pb2\_grpc.py中在pubsub\_pb2前加proto.
2. 把文件全堆到同一个文件夹内

2、参数数量不匹配，具体报错信息如下：  
Traceback (most recent call last):

File "D:\python\Lib\site-packages\grpc\\_server.py", line 552, in \_call\_behavior

response\_or\_iterator = behavior(argument, context)

^^^^^^^^^^^^^^^^^^^^^^^^^^^

TypeError: PubsubServer.publish() takes 2 positional argume

解决方法：虽然压根没有用到该参数，但是在PubsubServer.publish和PubsubServer.subcribe中再添加一个context参数即可。

3、列表与索引不匹配

在receive方法中，持续报错：

for msg in stub.subcribe(proto.pubsub\_pb2.SubRequest(topic = topic, clientID = clientID, TTL = TTL)):

^^^^^^^^^^^^

……

that terminated with:

status = StatusCode.UNKNOWN

details = "Exception iterating responses: list indices must be integers or slices, not str"

debug\_error\_string = "UNKNOWN:Error received from peer {grpc\_message:"Exception iterating responses: list indices must be integers or slices, not str", grpc\_status:2, created\_time:"2023-10-25T12:58:25.6189222+00:00"}"

>

后经检查是存放消息的表使用clientID作为索引，但在写subcribe函数时对消息表的定义是列表，而用字符串索引列表是不被允许的。

将列表改为字典后即可解决

理论部分：

1. 为什么传输层通信服务往往不适合构建分布式应用程序？

传输层通信服务通常面向点对点的通信，难以扩展到大规模的分布式系统，难以移植或者和其他应用程序交互。并且传输层通信仅提供低级别的网络通信抽象，缺乏高级功能，许多功能以及数据的处理都需要开发人员来实现，导致分布式应用程序开发难度进一步增大。

1. 描述一下客户端和服务器端之间使用套接字的无连接通信是如何进行的?
2. 服务器端创建套接字：服务器端首先创建一个UDP套接字，并绑定到一个特定的IP地址和端口号上。
3. 客户端创建套接字：客户端也创建一个UDP套接字，用于发送数据给服务器端（无需绑定）。
4. 客户端发送数据：客户端将要发送的数据打包成UDP数据报，并指定服务器的IP地址和端口号。然后客户端通过套接字将数据报发送给服务器端。
5. 服务器端接收数据：服务器端通过套接字监听指定的端口，如执行一个阻塞的read调用。当接收到客户端发送的数据报时，服务器端从套接字中读取数据。
6. 服务器端处理请求：服务器端根据接收到的数据进行相应的处理。
7. 服务器端发送响应：服务器端将处理结果打包成UDP数据报，并指定客户端的IP地址和端口号。然后服务器端通过套接字将数据报发送给客户端。
8. 客户端接收响应：客户端通过套接字监听指定的端口，当接收到服务器端发送的数据报时，客户端从套接字中读取数据。
9. 客户端处理响应：客户端根据接收到的响应进行相应的处理。
10. 假定客户通过异步RPC对服务器进行调用，随后等待服务器使用另一异步RPC返回结果。这种方法与客户执行常规的RPC有没有什么不同？如果使用的是同步RPC而不是异步RPC，情况如何？
    1. 、会有不同。在调用异步RPC后客户端不会立刻阻塞等待结果返回，而是可以继续执行别的任务。在服务器返回结果后客户端再通过回调函数或异步通知机制来处理结果。这两次异步RPC在通信可靠的情况下调用是一致的。
    2. 、若是使用同步RPC，客户端会阻塞并一直等待结果返回，只有在服务器返回结果后客户端才会继续按照顺序依次执行RPC调用。
11. 假定只能使用瞬时同步通信原语，如何实现用于瞬时异步通信的原语

可以使用回调机制来实现用于瞬时异步通信的原语。客户端在发起请求时提供一个回调函数：在服务器在处理完请求后，将响应传递给回调函数然后传给客户端。此后客户端可以继续执行其他任务，而不需要阻塞等待响应。等到服务器在处理完请求后，调用回调函数将响应传递给客户端。