分布式系统实验报告（作业3）

18340166 王若琪

一、实验要求

使用protobuf和gRPC等远程过程调用的方法实现消息订阅（p ublish-subscribe）系统，该订阅系统能够实现简单的消息传输， 并能够控制访问请求的数量，还可以控制消息在服务器端存储 的时间。

二、实验过程

2.1 环境配置

选择使用 python 语言来完成作业，下载如下几个包即可：

```
pip install grpcio grpcio-tools protobuf -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

2.2 编写框架

接下来编写 protobuf 文件 PubSub.proto，来定义服务的框架：

```
syntax = "proto3";
package grpc;
message Empty {}
// I called it Note because message Message sounds complicated
message Note {
    string name = 1;
    string message = 2;
}
service PubSubServer{
    // This bi-directional stream makes it possible to send and receive
Notes between 2 persons
    rpc PubSubStream (Empty) returns (stream Note);
    rpc SendNote (Note) returns (Empty);
}
```

编译此文件，生成对应的 Python 文件 PubSub_pb2_grpc.py 和 PubSub_pb2.py：

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

2.3 编写 PubSubServer 类

- 在 PubSubServer 类中定义 PubSubStream(self, request_iterator, context) 函数，用来传送发布和订阅的消息。在此类中，服务器可以一直发送消息，客户端在一直等待接收消息，一旦服务器发布消息，所有客户端都会立刻接收到此消息：

```
def PubSubStream(self, request_iterator, context):
        last_timestamp = ''
        for item in self.listening.listen():
            if item['data'] != 1:
                data = json.loads(item['data'])
                if data["timestamp"] <= last_timestamp:
                    continue
                n = PubSub.Note()
                n.name = data['user']
                n.message = data['message']
                yield n
                last_timestamp = data["timestamp"]
```

- 在 PubSubServer 类中定义 SendNote(self, request: PubSub.Note, context) 函数，功能是可以让客户端传递消息给服务器：

```
    def SendNote(self, request: PubSub.Note, context):
        self.sending.publish('chat', json.dumps(
            {'message': request.message, 'user': request.name,
'timestamp': str(datetime.datetime.now())}))
        return PubSub.Empty()
```

2.4 编写 PubSubClass 类

- send(self) 函数用来发布消息，并在服务器端将发布过的消息储存进文件：

```
    def send(self):
        message = ''
        while message != 'exit':
            message = input()
            n = PubSub.Note()
            n.name = self.username  # set the username
            n.message = message
            self.conn.SendNote(n)
            with open(filename, 'a') as file_object:
                    file_object.write("{} : {}\n".format(n.name,
n.message))

```

- recive(self) 函数可以接收消息，并在服务器端将接收到的消息储存进文件：

```
    def recive(self):
        for note in self.conn.PubSubStream(PubSub.Empty()):
            if note.name and note.name != self.username:
                print("{} : {}\n".format(note.name, note.message))
                with open(filename, 'a') as file_object:
                    file_object.write("{} : {}\n".format(note.name,
note.message))
```

- delete(self) 函数可以定时删除客户端储存的消息：

```
    def delete(self):
        while True:
            time.sleep(20)
            deletefile = open(filename, 'w').close()
```

- 在服务器端，除了发布消息的线程，再多添两个线程，一个用来接收消息，一个用来给消息定时删除：

```
while True:
    ch.send()
```

```
        threading.Thread(target=self.recive, daemon=True).start()
        threading.Thread(target=self.delete, daemon=True).start()
```

客户端编程模式与上述内容大同小异，不做赘述，具体请参考源代码。

2.5 控制访问请求数目

在 redis-cli 中设置最大连接数目：

```
config set maxclients <num>
```

之后就可以控制访问数量，如果超出限制，会产生报错。

2.6 控制消息在服务器端存储时间

我在服务器端将消息储存在了文件 server_text.txt 中，在服务器端多开一个线程，执行 delete() 函数，每隔20秒清空一次文件内容，即可控制消息在服务器端的存储时间：

```
def delete(self):
    while True:
        time.sleep(20)
        deletefile = open(filename, 'w').close()
```

三、实验结果

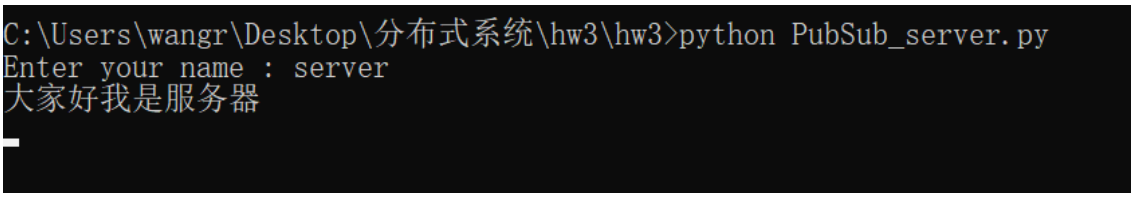
运行服务器：

```
python PubSub_server.py
```

运行客户端：

```
python PubSub_client.py
```

- 服务器发送消息时，订阅的客户端都立刻收到：



```
C:\Users\wangr\Desktop\分布式系统\hw3\hw3>python PubSub_client.py
Enter your name : client1
server : 大家好我是服务器
```

```
C:\Users\wangr\Desktop\分布式系统\hw3\hw3>python PubSub_client.py
Enter your name : client2
server : 大家好我是服务器
```

- 能够控制访问请求的数量：

  当数量大于设定值时，客户端会产生如下报错：

```
    (ex.args,))
redis.exceptions.ConnectionError: Error while reading from socket: (10054, '远程主机强迫关闭了一个现有的连接。', None, 10054, None)
```

  服务器会产生如下报错：

```
Traceback (most recent call last):
  File "PubSub_server.py", line 47, in <module>
    ch = PubSubClass(address, port)
  File "PubSub_server.py", line 18, in __init__
    self.send()
  File "PubSub_server.py", line 27, in send
    self.conn.SendNote(n)
  File "C:\Anaconda3\lib\site-packages\grpc\_channel.py", line 923, in __call__
    return _end_unary_response_blocking(state, call, False, None)
  File "C:\Anaconda3\lib\site-packages\grpc\_channel.py", line 826, in _end_unary_response_blocking
    raise _InactiveRpcError(state)
grpc._channel._InactiveRpcError: <_InactiveRpcError of RPC that terminated with:
        status = StatusCode.UNKNOWN
        details = "Exception calling application: max number of clients reached"
        debug_error_string = "{"created":"@1605838087.773000000","description":"Error received from peer ipv6:[::1]:11917","file":"src/core/lib/surface/call.cc","file_line":1062,"grpc_message":"Exception calling application: max number of clients reached","grpc_status":2}"
>
```

- 并且服务器端能够控制消息储存的时间，效果如下：

  定期删除消息前：

  定期删除消息后：

  每条消息最多在服务器端存储 20 秒。

四、总结与感想

- 本次实验走了很多弯路，一开始由于网络上关于 go 语言的参考资料比较多，所以我本来是想用 go 语言完成作业的，但是因为网络限制，go + grpc + protobuf 的环境配置了一整天都没配置好，所以才开始转战 python，python 环境很简单就配好了，但是怎么去编程又成了一大问题。
- 编程时参考了网页 https://github.com/pouria-farhadi/chat_server ，因为没有学过 python 编程，所以上手编程的时候有些困难，好在 python 语法很简单，所以再简单查阅 python 相关资料之后，就能够进行简单的应用了。

- 总而言之，本次实验走了很多弯路，而且因为这两种编程语言之前都没有接触过，几乎是从零开始，所以希望之后的作业能够提供更多的参考资料。