分布式系统第三次作业

何泽 18340052

> 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， 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。
>
> 编程语言不限，但是推荐使用python和Go。
>
> 参考：
>
> - https://github.com/vardius/pubsub
> - https://github.com/GoogleCloudPlatform/cloud-pubsub-samples-python/blob/master/grpc/pubsub_sample.py

I 实验目标

- 实现消息订阅系统，客户端可以查看服务器端主题并订阅，之后可以向服务器端发送消息；同时服务器端收到订阅请求后首先将历史消息发送给客户端，同时收到新消息时转发给所有订阅的客户端
- 可以控制客户端访问请求的最大数量
- 可以控制历史消息在服务器端保存的时间

II 代码实现

1. protobuf

三个 `rpc` 函数，分别代表发布消息、查看主题和订阅某主题

```
syntax = "proto3";
service Pubsub {
    // 发布主题消息
```

```
    rpc publish(publishRequest) returns (reply) {}
    // 浏览主题
    rpc browse(browseRequest) returns (stream reply) {}
    // 订阅主题
    rpc subcribe(subRequest) returns (stream reply) {}
}
message publishRequest {
    string topic = 1;
    string context = 2;
}
message reply {
    string message = 1;
}
message browseRequest {
    string topic = 1;
}
message subRequest {
    string topic = 1;
    string clientId = 2;
    int32 TTL = 3;
}
```

2. 服务器端

- 发布某主题的消息

```
def publish(self, topic, message):
    msg = ""
    if topic not in self.storage:
        self.storage[topic] = [{'createTime': time.time(), 'message': message}]
        msg += "create topic: {}\n".format(topic)
    else:
        self.storage[topic].append({'createTime': time.time(), 'message': message})
    if topic in self.event:
        for client in self.event[topic]:
            self.event[topic][client].set()

    msg += "publish successful"
    return msg
```

- 控制历史消息存储的时间，每隔一段时间就删除超时历史消息

```
def refresh(self, TTL=10):
    ddl = time.time() - 10
    for topic in self.storage:
        while len(self.storage[topic]) and self.storage[topic][0]['createTime'] <= ddl:
            del self.storage[topic][0]
```

- 返回所有可以订阅的主题

```
def browse(self, topic):
    if topic not in self.storage:
        return ["topic not created"]
    for msg in self.storage[topic]:
        yield self.gen_msg(msg)
```

- 订阅某个主题

```
def subcribe(self, topic, clientId, TTL=20):
    if topic not in self.event:
        self.event[topic] = {}
    self.event[topic][clientId] = Event()
    createTime = time.time()
    remainTime = TTL
    while True:
        self.event[topic][clientId].wait(remainTime)
        remainTime = TTL - (time.time() - createTime)
        if remainTime <= 0:
            break
        yield self.gen_msg(self.storage[topic][-1])
        self.event[topic][clientId].clear()
```

- 通过 `grpc` 通信过程

```
class PubsubService(pubsub_pb2_grpc.Pubsub):
    def __init__(self):
        self.pubsub = Pubsub()
    def publish(self, request, context):
        msg = self.pubsub.publish(request.topic,
request.context)
        return pubsub_pb2.reply(message = msg)
    def browse(self, request, context):
        for msg in self.pubsub.browse(request.topic):
            yield pubsub_pb2.reply(message=msg)
    def subcribe(self, request, context):
        for msg in self.pubsub.subcribe(request.topic,
request.clientId, request.TTL):
            yield pubsub_pb2.reply(message=msg)
```

- 服务器运行

```
server =
grpc.server(futures.ThreadPoolExecutor(max_workers=10))
pubsubServe = PubsubService()
pubsub_pb2_grpc.add_PubsubServicer_to_server(pubsubServe,
server)

server.add_insecure_port('[::]:50051')
server.start()
try:
    while True:
        time.sleep(1)
        pubsubServe.pubsub.refresh()
except KeyboardInterrupt:
    server.stop(0)
```

3. 客户端

客户端可以查看服务器端主题并订阅，之后可以向服务器端发送消息

```
import grpc
import time
import threading as trd
import pubsub_pb2
import pubsub_pb2_grpc

clientId = input("Id: ")
channel = grpc.insecure_channel('localhost:50051')
stub = pubsub_pb2_grpc.PubsubStub(channel)
def publish(topic, context):
    print("Publish message in {}:".format(topic, context))
    response = stub.publish(pubsub_pb2.publishRequest(topic=topic,
context=context))
    print(response.message)
def browse(topic):
    print("Browse topic {}".format(topic))
    response = stub.browse(pubsub_pb2.browseRequest(topic=topic))
    for msg in response:
        print(msg.message)
```

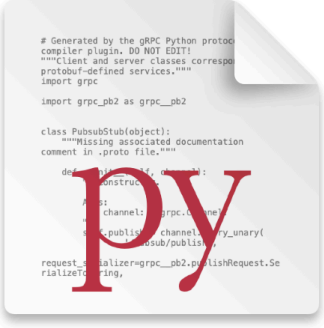
```
def _subcribe(topic, TTL):
    for msg in stub.subcribe(pubsub_pb2.subRequest(topic=topic, clientId=clientId, TTL=TTL)):
        print("Receive message from {}:".format(topic, msg.message))

def subcribe(topic, TTL=20):
    print("Subscribed {} successfully.".format(topic))
    thrd = trd.Thread(target=_subcribe, args=(topic,TTL))
    thrd.start()

publish('test_topic', 'message1')
browse('test_topic')
time.sleep(5)
publish('test_topic', 'message2')
subcribe('test_topic', 20)
publish('test_topic', 'message3')
time.sleep(6)
browse('test_topic')
```

III 运行结果

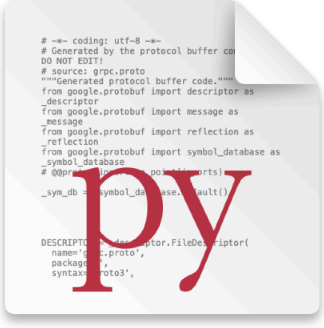
- 首先运行 `protobuf` 文件

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

之后便会生成如下两个文件：



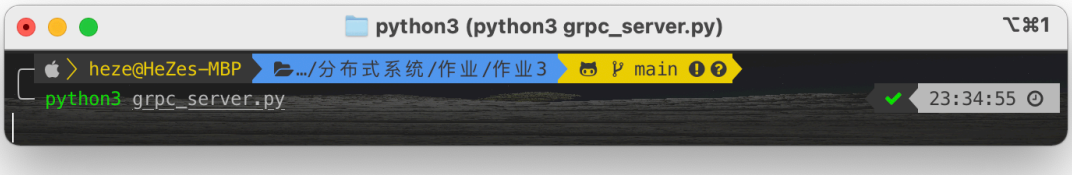
grpc_pb2_grpc.py



grpc_pb2.py

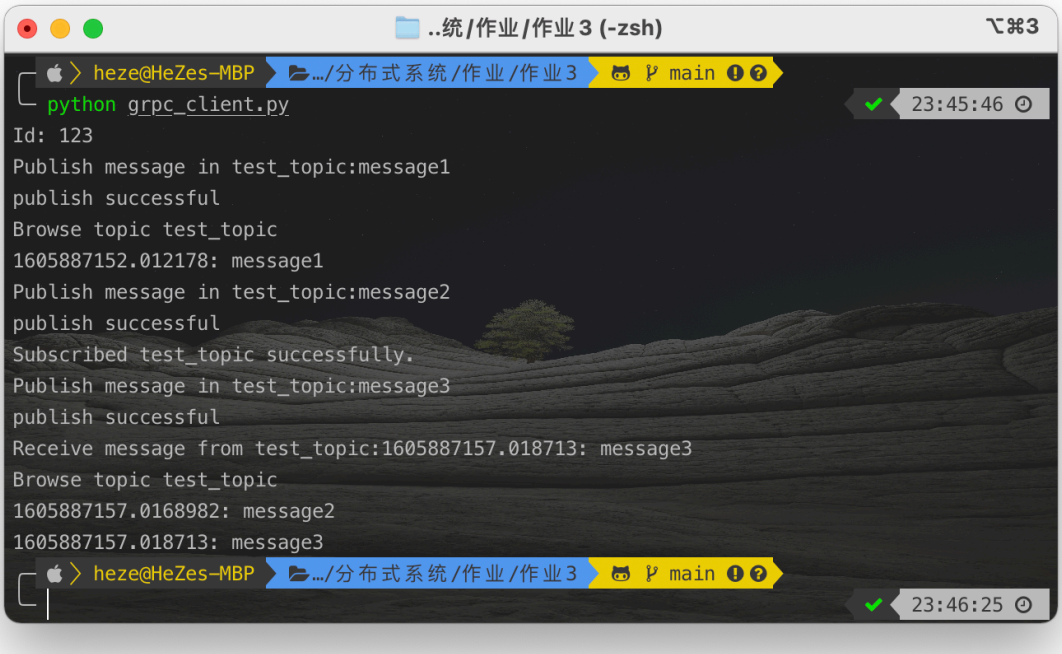
- 运行服务器端：

```
python3 grpc_server.py
```



- 运行客户端：

```
python grpc_client.py
```



可以看到，发布订阅消息、浏览主题历史消息、控制消息保存时间均已完成，同时控制最大连接数量可更改服务器端 `server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))` 中的 `max_workers` 变量，至此，实验完成。