分布式系统作业3

18340013 陈琮昊

一、作业要求：

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。

二、准备工作：

注：本次实验完成在`Windows10+Python3.6(Anaconda3)`环境下。

首先就是要安装环境，在Python下输入两条命令即可：

```
pip install grpcio
pip install grpcio-tools googleapis-common-protos
```

安装好以后先跑一个`demo`来试一下！

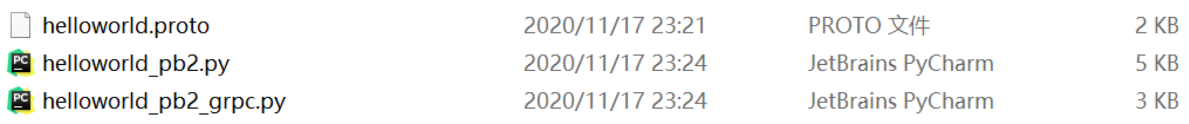
创建一个`proto`文件夹，编写一个文件命名为`helloworld.proto`并放在该文件夹下，文件内容如下：

```
// Copyright 2015 gRPC authors.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
option objc_class_prefix = "HLW";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

然后通过`cd`进入该目录下，输入如下命令：

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

可以看到文件夹内多了两个`Python`文件：



helloworld.proto	2020/11/17 23:21	PROTO 文件	2 KB
helloworld_pb2.py	2020/11/17 23:24	JetBrains PyCharm	5 KB
helloworld_pb2_grpc.py	2020/11/17 23:24	JetBrains PyCharm	3 KB

其中`helloworld_pb2.py`文件包含生成的`request(HelloRequest)` 和`response(HelloReply)`类。`hello_pb2_grpc.py` 文件包含生成的客户端`(GreeterStub)`和服务端`(GreeterServicer)`的类。

然后编写客户端和服务端的代码：

Server(`greeter_server.py`):

```
from concurrent import futures
import time
import grpc
import helloworld_pb2
import helloworld_pb2_grpc

_ONE_DAY_IN_SECONDS = 60 * 60 * 24


class Greeter(helloworld_pb2_grpc.GreeterServicer):
    # 工作函数
    def SayHello(self, request, context):
        print(request.name)
        message = "This message is from Server.And what i want to say is hello
\" " + request.name + " \"";
        return helloworld_pb2.HelloReply(message = message)


def serve():
    # gRPC 服务器
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    helloworld_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
    server.add_insecure_port('[::]:50051')
    print("sever is opening ,waiting for message...")
    server.start()  # start() 不会阻塞，如果运行时你的代码没有其它的事情可做，你可能需要循
环等待。
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)
```

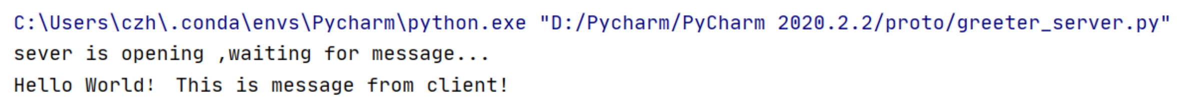
```
if __name__ == '__main__':
    serve()
```

Client(`greeter_client.py`):

```
from __future__ import print_function

import grpc

import helloworld_pb2
import helloworld_pb2_grpc


def run():
    channel = grpc.insecure_channel('localhost:50051')
    stub = helloworld_pb2_grpc.GreeterStub(channel)
    response = stub.SayHello(helloworld_pb2.HelloRequest(name='Hello World!  This
is message from client!'))
    print("Greeter client received: " + response.message)


if __name__ == '__main__':
    run()
```

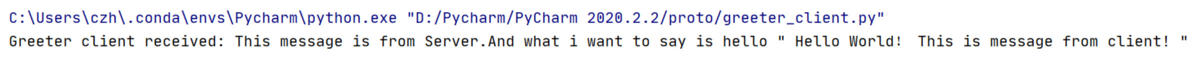
然后打开两个Terminal运行，结果如下：

Server：



```
C:\Users\czh\.conda\envs\Pycharm\python.exe "D:/Pycharm/PyCharm 2020.2.2/proto/greeter_server.py"
sever is opening ,waiting for message...
Hello World!  This is message from client!
```

Client：



```
C:\Users\czh\.conda\envs\Pycharm\python.exe "D:/Pycharm/PyCharm 2020.2.2/proto/greeter_client.py"
Greeter client received: This message is from Server.And what i want to say is hello " Hello World!  This is message from client! "
```

至此一个demo完成。

三、实验任务：

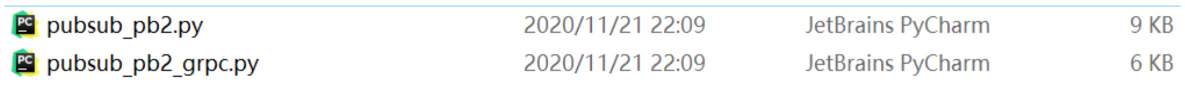
根据上面的demo可以知道本次实验的流程，首先写一个 `pubsub.proto` 文件，内容如下：

```
syntax = "proto3";

service Pubsub {
    rpc publish(publishRequest) returns (reply) {}                //发布主题
    rpc browse(browseRequest) returns (stream reply) {}           //浏览主题
    rpc subcribe(subRequest) returns (stream reply) {}            //订阅主题
}

message publishRequest {
    string topic = 1;               //主题
    string context = 2;             //上下文
}

message reply {
    string message = 1;           //消息
}

message browseRequest {
    string topic = 1;             //主题
}

message subRequest {
    string topic = 1;  // 主题
    string clientId = 2;  // 客户端ID
    int32 TTL = 3;  // 存储时间
}
```

然后通过`cd`进入该目录下，输入如下命令：

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

得到了两个`Python`文件：



pubsub_pb2.py	2020/11/21 22:09	JetBrains PyCharm	9 KB
pubsub_pb2_grpc.py	2020/11/21 22:09	JetBrains PyCharm	6 KB

然后编写客户端和服务器端代码，文件分别为`server.py`和`client.py`。

`server.py`:

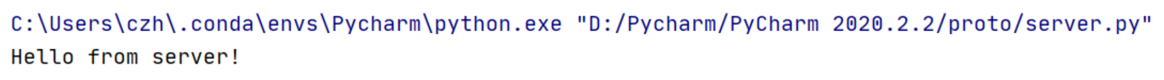
```
from threading import Event
from concurrent import futures
import time
import grpc


import pubsub_pb2
import pubsub_pb2_grpc

class Pubsub(object):
    def __init__(self):
        self.storage = {}
        self.event = {}
    def publish(self, topic, message):
        # 发布
        newmessage = ""
        if topic not in self.storage:
            self.storage[topic] = [{'create time': time.time(), 'message':
message}]
            newmessage += "create topic: {}\n".format(topic)
        else:
            self.storage[topic].append({'create time': time.time(), 'message':
message})
        if topic in self.event:
            for client in self.event[topic]:
                self.event[topic][client].set()
        newmessage += "Successfully publish!"
        return newmessage
    def generatemessage(self, newmessage):
        # 转为可见消息
        return str(newmessage['create time']) + ": " + newmessage['message']
    def refresh(self, TTL=5):
        # 控制消息在服务器的存储时间，超时则删除
        ttl = time.time() - 5
        for topic in self.storage:
            while len(self.storage[topic]) and self.storage[topic][0]['create
time'] <= ttl:
                del self.storage[topic][0]
    def browse(self, topic):
        # 浏览
        if topic not in self.storage:
            return ["topic not created!"]
        for newmessage in self.storage[topic]:
            yield self.generatemessage(newmessage)
    def subcribe(self, topic, clientId, TTL=10):
        # 订阅
        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.generatemessage(self.storage[topic][-1])
            self.event[topic][clientId].clear()

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

if __name__ == '__main__':
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=20))
    pubsub_server = PubsubService()
    pubsub_pb2_grpc.add_PubsubServicer_to_server(pubsub_server, server)
    server.add_insecure_port('[::]:50051')
    server.start()
    print('Hello from server!')
    try:
        while True:
            time.sleep(1)
            pubsub_server.pubsub.refresh()
    except KeyboardInterrupt:
        server.stop(0)
```

`client.py`:

```python
import grpc
import time
import threading
import pubsub_pb2
import pubsub_pb2_grpc

clientId = input("Please 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 i in response:
        print(i.message)
def receive(topic, TTL):
    for j in stub.subcribe(pubsub_pb2.subRequest(topic=topic, clientId=clientId,
TTL=TTL)):
        print("Receive message from {}:{}".format(topic, j.message))
def subcribe(topic, TTL=10):
    print("Successfully subscribed {}!".format(topic))
    xc = threading.Thread(target=receive, args=(topic,TTL))
    xc.start()

publish('testtopic1', 'message1')
browse('testtopic1')
time.sleep(5)
publish('testtopic2', 'message2')
subcribe('testtopic2', 10)
publish('testtopic3', 'message3')
time.sleep(5)
browse('testtopic3')
```

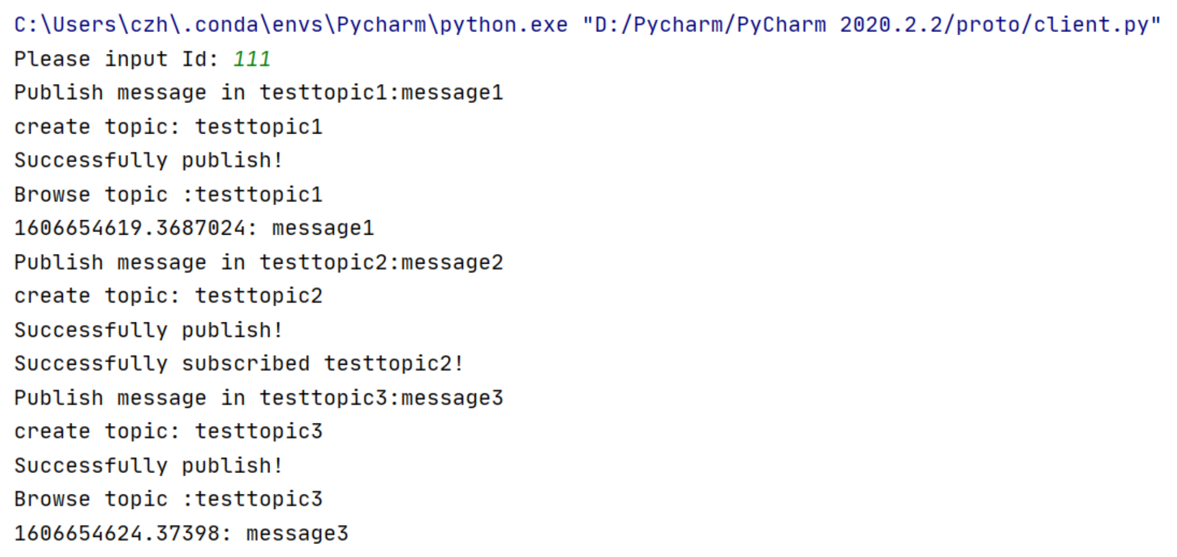
实验结果如下：

服务器端：



```
C:\Users\czh\.conda\envs\Pycharm\python.exe "D:/Pycharm/PyCharm 2020.2.2/proto/server.py"
Hello from server!
```

客户端：



```
C:\Users\czh\.conda\envs\Pycharm\python.exe "D:/Pycharm/PyCharm 2020.2.2/proto/client.py"
Please input Id: 111
Publish message in testtopic1:message1
create topic: testtopic1
Successfully publish!
Browse topic :testtopic1
1606654619.3687024: message1
Publish message in testtopic2:message2
create topic: testtopic2
Successfully publish!
Successfully subscribed testtopic2!
Publish message in testtopic3:message3
create topic: testtopic3
Successfully publish!
Browse topic :testtopic3
1606654624.37398: message3
```

可以看到，发布、浏览、订阅消息均已完成，并且也能够同时控制消息在服务器端存储的时间(TTL)。至于控制访问请求的数量，只需修改`server.py`内的`server = grpc.server(futures.ThreadPoolExecutor(max_workers=20))`中的`max_workers`即可。

四、实验感想：

本次实验在开始之前先是选择了编程语言：在网上查找`go`的`gRPC`安装过程发现相当复杂，而且`go`语言之前没有接触过；而`Python`的`gRPC`配置则只需两条命令，而且还很熟悉`Python`的语法，所以自然而然的选择了`Python`来完成本次任务。

五、参考资料：

https://blog.csdn.net/u013992365/article/details/81704459?utm_source=blogxgwz20

https://github.com/pouria-farhadi/chat_server/blob/master