# Thrift

## thrift教程

thrift官网: <https://thrift.apache.org/>

上课代码地址：https://git.acwing.com/yxc/thrift\_lesson

## 2. 创建作业 & 测试作业的正确性

homework 6 create # 可以重新创建所有lesson\_6的作业

homework 6 test # 可以评测lesson\_6的所有作业

## 3. 作业

本次作业为复现课上最后一个版本的内容，课程视频地址：https://www.acwing.com/video/3479/

注意：本次作业的2个题目采用整体评测，即如果两个作业同时正确，则得100分；否则如果至少有一个作业错误，则得0分。

创建好作业后，先进入文件夹/home/acs/homework/lesson\_6/，当前目录的文件结构如下：

`-- thrift\_lesson

|-- game

| `-- src

|-- match\_system

| `-- src

|-- readme.md

`-- thrift

|-- match.thrift

`-- save.thrift

(0) 进入thrift\_lesson/match\_system/src/目录，用cpp实现课上的match-server和save-client逻辑。

接口文件在thrift\_lesson/thrift/中。

实现后启动server，监听端口9090。

(1) 进入thrift\_lesson/game/src/目录，用python3实现课上的match-client逻辑。

文件名和输入格式与课上内容相同。

作业代码：

Homework\_0

// This autogenerated skeleton file illustrates how to build a server.

// You should copy it to another filename to avoid overwriting it.

#include "match\_server/Match.h"//服务器端头文件

#include "save\_client/Save.h"//客户端头文件

#include <thrift/protocol/TBinaryProtocol.h>

#include <thrift/server/TSimpleServer.h>

#include <thrift/transport/TServerSocket.h>

#include <thrift/transport/TBufferTransports.h>

#include <thrift/transport/TSocket.h>

#include <thrift/transport/TTransportUtils.h>//服务器端特殊头文件

#include <thrift/concurrency/ThreadManager.h>//多线程服务器

#include <thrift/concurrency/ThreadFactory.h>//多线程服务器

#include <thrift/TToString.h> //多线程服务器

#include <thrift/server/TThreadedServer.h>//多线程服务器

#include <iostream>

#include <thread>//用于开线程的头文件

#include <mutex>//锁 防止多个线程相互干扰队列

#include <condition\_variable>//条件变量对锁做了一个封装

#include <queue>

#include <string>

#include <vector>

#include <unistd.h>

using namespace ::apache::thrift;

using namespace ::apache::thrift::protocol;

using namespace ::apache::thrift::transport;

using namespace ::apache::thrift::server;

using namespace ::match\_service;//服务器端

using namespace ::save\_service; //客户端

using namespace std;

struct Task

{

User user;

std::string type;

};

struct MessageQueue //生产者-消费者模型 消息队列

{

queue<Task> q;

mutex m;

condition\_variable cv;

}message\_queue;

class Pool //定义一个玩家池

{

public:

void save\_result(int a, int b)

{

printf("Match Result: %d %d\n",a,b);

std::shared\_ptr<TTransport> socket(new TSocket("123.57.47.211", 9090));

std::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));

std::shared\_ptr<TProtocol> protocol(new TBinaryProtocol(transport));

SaveClient client(protocol);

try {

transport->open();

client.save\_data("acs\_6502","4d487d0d",a,b);

transport->close();

} catch (TException& tx) {

cout << "ERROR: " << tx.what() << endl;

}

}

/\*void match()

{

while(users.size()>1)

{

auto a = users[0],b=users[1];

users.erase(users.begin());

users.erase(users.begin());

save\_result(a.id,b.id);

}

}\*/

/\*void match(){

while (users.size()>1)

{

sort(users.begin(),users.end(),[&](User& a, User& b){return a.score < b.score;});

bool flag = true;

for(uint32\_t i=1; i<users.size(); i++)

{

auto a = users[i-1], b = users[i];

if(b.score - a.score <= 50)

{

users.erase(users.begin()+i-1,users.begin()+i+1);

save\_result(a.id,b.id);

flag = false;

break;

}

}

if(flag) break;

}

}\*/

bool check\_match(uint32\_t i,uint32\_t j)

{

auto a = users[i],b = users[j];

int dt = abs(a.score-b.score);

int a\_max\_dif = wt[i] \* 50;

int b\_max\_dif = wt[j] \* 50;

return dt <= a\_max\_dif && dt<= b\_max\_dif;

}

void match()

{

for(uint32\_t i=0;i<wt.size();i++)

wt[i]++;//所有匹配池的玩家秒数+1

while(users.size()>1)

{

bool flag = true;

for(uint32\_t i=0; i < users.size();i++)

{

for(uint32\_t j=i+1;j<users.size();j++)

{

if(check\_match(i,j))

{

auto a = users[i],b = users[j];

users.erase(users.begin()+j);//先删后面，否则下标会变

users.erase(users.begin()+i);

wt.erase(wt.begin()+j);//先删后面，否则下标会变

wt.erase(wt.begin()+i);

save\_result(a.id,b.id);

flag = false;

break;

}

}

if(!flag) break;

}

if(flag) break;

}

}

void add(User user) //添加一个玩家

{

users.push\_back(user);

wt.push\_back(0);

}

void remove(User user) //删除一个玩家

{

for(uint32\_t i=0; i<users.size() ; i++)

if( users[i].id == user.id)

{

users.erase(users.begin()+i);

wt.erase(wt.begin()+i);

break;

}

}

private: //存储所有的玩家

vector<User> users;

vector<int> wt; //wating-time等待时间/s

}pool;

class MatchHandler : virtual public MatchIf {

public:

MatchHandler() {

// Your initialization goes here

}

int32\_t add\_user(const User& user, const std::string& info) {

// Your implementation goes here 生产者

printf("add\_user\n");

unique\_lock<mutex> lck(message\_queue.m);

//把锁定义在函数体内可/以不用显式解锁

message\_queue.q.push({user,"add"});

//多个线程访问队列容易覆盖应该有的操作

message\_queue.cv.notify\_all();//唤醒条件变量

return 0;

}

int32\_t remove\_user(const User& user, const std::string& info) {

// Your implementation goes here 生产者

printf("remove\_user\n");

unique\_lock<mutex> lck(message\_queue.m);

message\_queue.q.push({user,"remove"});

message\_queue.cv.notify\_all();//唤醒所有线程

return 0;

}

};

/\*void match(){

while (users.size()>1)

{

sort(users.begin(),users.end(),[&](User& a, User& b){return a.score < b.score;});

bool flag = true;

for(uint32\_t i=1; i<users.size(); i++)

{

auto a = users[i-1], b = users[i];

if(b.score - a.score <= 50)

{

users.erase(users.begin()+i-1,users.begin()+i+1);

save\_result(a.id,b.id);

flag = false;

break;

}

}

if(flag) break;

}

}

\*/

void consume\_task(){

while(true)

{

std::unique\_lock<mutex> lck(message\_queue.m);

if(message\_queue.q.empty())

{

//message\_queue.cv.wait(lck);

//先把锁释放掉，然后将函数卡死在这里，直到唤醒条件变量。

lck.unlock();

pool.match();

sleep(1);

}

else

{

auto task = message\_queue.q.front();

message\_queue.q.pop();

lck.unlock();//一定要解锁，不解锁会导致执行task时间太久

//do task

if (task.type == "add") pool.add(task.user);

else if(task.type == "remove") pool.remove(task.user);

//pool.match();

}

}

}

class MatchCloneFactory : virtual public MatchIfFactory {

public:

~MatchCloneFactory() override = default;

MatchIf\* getHandler(const ::apache::thrift::TConnectionInfo& connInfo) override

{

std::shared\_ptr<TSocket> sock = std::dynamic\_pointer\_cast<TSocket>(connInfo.transport);

cout << "Incoming connection\n";

cout << "\tSocketInfo: " << sock->getSocketInfo() << "\n";

cout << "\tPeerHost: " << sock->getPeerHost() << "\n";

cout << "\tPeerAddress: " << sock->getPeerAddress() << "\n";

cout << "\tPeerPort: " << sock->getPeerPort() << "\n";

return new MatchHandler;

}

void releaseHandler( MatchIf\* handler) override {

delete handler;

}

};

int main(int argc, char \*\*argv) {

/\*

int port = 9090;

::std::shared\_ptr<MatchHandler> handler(new MatchHandler());

::std::shared\_ptr<TProcessor> processor(new MatchProcessor(handler));

::std::shared\_ptr<TServerTransport> serverTransport(new TServerSocket(port));

::std::shared\_ptr<TTransportFactory> transportFactory(new TBufferedTransportFactory());

::std::shared\_ptr<TProtocolFactory> protocolFactory(new TBinaryProtocolFactory());

TSimpleServer server(processor, serverTransport, transportFactory, protocolFactory);\*/

TThreadedServer server(

std::make\_shared<MatchProcessorFactory>(std::make\_shared<MatchCloneFactory>()),

std::make\_shared<TServerSocket>(9090), //port

std::make\_shared<TBufferedTransportFactory>(),

std::make\_shared<TBinaryProtocolFactory>());

cout << "Start Match Server" << std::endl;

thread matching\_thread(consume\_task);

//函数指针传入，单开一个消费者生产者线程,如果不单开的话，serve不了

server.serve();

return 0;

}

创建match-server：

cd thrift\_lesson/match\_system/src

thrift -r --gen cpp ../../thrift/match.thrift

mv gen-cpp match\_server

rm match\_server/Match\_server.skeleton.cpp

创建save-client：

cd thrift\_lesson/match\_system/src

thrift -r --gen cpp ../../thrift/save.thrift

mv gen-cpp save\_client

rm save\_client/Save\_server.skeleton.cpp

创建main.cpp，源代码在这里:

编译：

g++ -c main.cpp match\_server/\*.cpp save\_client/\*.cpp

链接：

g++ \*.o -o main -lthrift -pthread

启动服务：

./main

Homework\_1

from match\_client.match import Match

from match\_client.match.ttypes import User

from sys import stdin

from thrift import Thrift

from thrift.transport import TSocket

from thrift.transport import TTransport

from thrift.protocol import TBinaryProtocol

def operate(op,user\_id,username,score):

# Make socket

transport = TSocket.TSocket('localhost', 9090)

# Buffering is critical. Raw sockets are very slow

transport = TTransport.TBufferedTransport(transport)

# Wrap in a protocol

protocol = TBinaryProtocol.TBinaryProtocol(transport)

# Create a client to use the protocol encoder

client = Match.Client(protocol)

# Connect!

transport.open()

user = User(user\_id,username,score)

if op == "add":

client.add\_user(user,"")

elif op == "remove":

client.remove\_user(user,"")

# Close!

transport.close()

def main():

for line in stdin:

op,user\_id,username,score = line.split(' ')

operate(op,int(user\_id),username,int(score))

if \_\_name\_\_ == "\_\_main\_\_":

main()

创建match-client：

cd thrift\_lesson/game/src

thrift -r --gen py ../../thrift/match.thrift

mv gen-py match\_client

创建client.py，内容在这里。

执行client.py：

python3 client.py