thrift框架使用C++

[收藏](javascript:void(0);)

[zmldndx](https://my.oschina.net/zmlblog/home)

* 发表于 3年前

* 阅读 11093

* 收藏 13

* 点赞 2

* [评论 1](https://my.oschina.net/zmlblog/blog/177245#comment-list)

1. 编写thrift接口文件student.thrift 

struct Student{

1: i32 sno,

2: string sname,

3: bool ssex,

4: i16 sage,

}

service Serv{

i32 put(1: Student s),

}

2. 用“thrift -r --gen cpp student.thrift”在gen-cpp文件夹中生成cpp及头文件，其中自动生成了Serv\_server.skeleton.cpp文件，它是简单的server端代码，可以修改（一般都重新参照来写server端代码）   
可以用g++ -g -DHAVE\_NETINET\_IN\_H -I. -I/usr/local/include/thrift -L/usr/local/lib Serv.cpp student\_types.cpp student\_constants.cpp Serv\_server.skeleton.cpp -o server -lthrift   
生成server可执行程序；   
  
3. 可以修改Serv\_server.skeleton.cpp文件，做成非阻塞server:   
    新建server.cpp文件（参照Serv\_server.skeleton.cpp并参照nonblockingServer的写法）： 

#include <concurrency/ThreadManager.h> //zml

#include <concurrency/PosixThreadFactory.h> //zml

#include "Serv.h"

#include <protocol/TBinaryProtocol.h>

#include <server/TSimpleServer.h>

#include <transport/TServerSocket.h>

#include <transport/TBufferTransports.h>

#include <server/TNonblockingServer.h> //zml

using namespace ::apache::thrift;

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

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

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

using namespace::apache::thrift::concurrency; //zml

using boost::shared\_ptr;

#define THREAD\_NUM 2

const int g\_port = 9090;

class ServHandler : virtual public ServIf {

public:

ServHandler() {

// Your initialization goes here

}

int32\_t put(const Student& s) {

// Your implementation goes here

printf("put student.sno=%d\n", s.sno);

return s.sno;

}

};

int thrift\_server\_run()

{

//创建thrift server

shared\_ptr<ServHandler> handler(new ServHandler());

shared\_ptr<TProcessor> processor(new ServProcessor(handler));

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

shared\_ptr<ThreadManager> threadManager = ThreadManager::newSimpleThreadManager(THREAD\_NUM);

shared\_ptr<PosixThreadFactory> threadFactory = shared\_ptr<PosixThreadFactory> (new PosixThreadFactory()); //PosixThreadFactory可以自定义（继承于ThreadFactory）

threadManager->threadFactory(threadFactory);

threadManager->start();

TNonblockingServer server(processor, protocolFactory, g\_port, threadManager);

try {

server.serve();

}

catch(TException e) {

printf("Server.serve() failed\n");

exit(-1);

}

return 0;

}

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

thrift\_server\_run();

while(1) {

sleep(10);

}

return 0;

}

    生成server可执行程序：   
    g++ -g -DHAVE\_NETINET\_IN\_H -I. -I/usr/local/include/thrift -L/usr/local/lib Serv.cpp student\_types.cpp student\_constants.cpp server.cpp -o server -lthriftnb -levent -lthrift -lrt   
  
4. 编写对接nonblockingServer的client端代码： 

#include "Serv.h" // 替换成你的.h

#include <transport/TSocket.h>

#include <transport/TBufferTransports.h>

#include <protocol/TBinaryProtocol.h>

using namespace apache::thrift;

using namespace apache::thrift::protocol;

using namespace apache::thrift::transport;

using boost::shared\_ptr;

int main()

{

boost::shared\_ptr<TSocket> socket(new TSocket("localhost", 9090));

//对接nonblockingServer时必须的，对普通server端时用boost::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));

boost::shared\_ptr<TTransport> transport(new TFramedTransport(socket));

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

ServClient client(protocol);

//设置发送、接收、连接超时

socket->setConnTimeout(2000);

socket->setRecvTimeout(2000);

socket->setSendTimeout(2000);

transport->open();

//insert your code here

Student stu;

stu.sno = 1;

stu.sname = "zml";

stu.ssex = 0;

stu.sage = 25;

int ret = client.put(stu);

printf("client put ret=%d\n", ret);

transport->close();

return 0;

}

    编译生成client可执行程序：   
    g++ -g -DHAVE\_NETINET\_IN\_H -I/usr/local/include/thrift -L/usr/local/lib/ Serv.cpp student\_types.cpp student\_constants.cpp client.cpp -o client -lpthread -lthrift -lrt