[**[zz]使用thrift做c++,java和python的相互调用**](https://www.cnblogs.com/zhangzhang/archive/2012/03/18/2404641.html)

linux上安装thrift见   
<http://jinghong.iteye.com/blog/1102535>   
thrift做为跨语言调用的方案有高效，支持语言较多，成熟等优点；代码侵入较强是其弱点。   
下面记录以C++做服务器，C++,java和python做客户端的示例，这个和本人现在工作环境吻合，使用多线程长连接的socket来建立高效分布式系统的跨语言调用平台。   
遗憾的是目前版本(0.7.0)的C语言还不支持Compact协议，导致在现在的环境中nginx c module调用thrift要使用binary协议。thrift开发团队似乎对C语言不太感冒。   
1.定义idl文件acsuser.thrift

Idl代码  收藏代码

1. struct User{
2. 1: string uid,
3. 2: string uname,
4. 3: bool usex,
5. 4: i16 uage,
6. }
7. service UserService{
8. void add(1: User u),
9. User get(1: string uid),
10. }

2.生成c++,java和python代码框架

Shell代码  收藏代码

1. thrift -r --gen cpp acsuser.thrift
2. thrift -r --gen java acsuser.thrift
3. thrift -r --gen py acsuser.thrift

这时生成子目录gen-cpp,gen-java,gen-py   
  
3.生成C++服务端代码

Shell代码  收藏代码

1. cp gen-cpp/UserService\_server.skeleton.cpp UserServer.cpp

修改UserServer.cpp

C++代码  收藏代码

1. #include "UserService.h"
2. #include <config.h>
3. //#include <protocol/TBinaryProtocol.h>
4. #include <protocol/TCompactProtocol.h>
5. #include <server/TSimpleServer.h>
6. #include <transport/TServerSocket.h>
7. #include <transport/TBufferTransports.h>
8. #include <concurrency/ThreadManager.h>
9. #include <concurrency/PosixThreadFactory.h>
10. #include <server/TThreadPoolServer.h>
11. #include <server/TThreadedServer.h>
13. **using** **namespace** ::apache::thrift;
14. **using** **namespace** ::apache::thrift::protocol;
15. **using** **namespace** ::apache::thrift::transport;
16. **using** **namespace** ::apache::thrift::server;
17. **using** **namespace** ::apache::thrift::concurrency;
19. **using** boost::shared\_ptr;
21. **class** UserServiceHandler : **virtual** **public** UserServiceIf {
22. **public**:
23. UserServiceHandler() {
24. // Your initialization goes here
25. }
27. **void** add(**const** User& u) {
28. // Your implementation goes here
29. printf("uid=%s uname=%s usex=%d uage=%d\n", u.uid.c\_str(), u.uname.c\_str(), u.usex, u.uage);
30. }
32. **void** get(User& \_return, **const** std::string& uid) {
33. // Your implementation goes here
34. \_return.uid = "leo1";
35. \_return.uname = "yueyue";
36. \_return.usex = 1;
37. \_return.uage = 3;
38. printf("uid=%s uname=%s usex=%d uage=%d\n", \_return.uid.c\_str(), \_return.uname.c\_str(), \_return.usex, \_return.uage);
39. }
41. };
43. int main(int argc, char \*\*argv) {
44. shared\_ptr<UserServiceHandler> handler(**new** UserServiceHandler());
45. shared\_ptr<TProcessor> processor(**new** UserServiceProcessor(handler));
46. shared\_ptr<TProtocolFactory> protocolFactory(**new** TCompactProtocolFactory());
47. shared\_ptr<TTransportFactory> transportFactory(**new** TBufferedTransportFactory());
48. shared\_ptr<TServerTransport> serverTransport(**new** TServerSocket(9090));
50. shared\_ptr<ThreadManager> threadManager = ThreadManager::newSimpleThreadManager(10);
51. shared\_ptr<PosixThreadFactory> threadFactory = shared\_ptr<PosixThreadFactory>(**new** PosixThreadFactory());
52. threadManager->threadFactory(threadFactory);
53. threadManager->start();
54. printf("start user server...\n");
56. TThreadPoolServer server(processor, serverTransport, transportFactory, protocolFactory, threadManager);
57. server.serve();
58. **return** 0;
59. }

注意这段代码使用TCompactProtocol，需要#include <config.h>   
另外这个是Blocking的多线程服务器   
  
4.生成C++的client文件UserClient.cpp

C++代码  收藏代码

1. #include "UserService.h"
2. #include <config.h>
3. #include <transport/TSocket.h>
4. #include <transport/TBufferTransports.h>
5. #include <protocol/TCompactProtocol.h>
7. **using** **namespace** apache::thrift;
8. **using** **namespace** apache::thrift::protocol;
9. **using** **namespace** apache::thrift::transport;
11. **using** boost::shared\_ptr;
13. int main(int argc, char \*\*argv) {
14. boost::shared\_ptr<TSocket> socket(**new** TSocket("localhost", 9090));
15. boost::shared\_ptr<TTransport> transport(**new** TBufferedTransport(socket));
16. boost::shared\_ptr<TProtocol> protocol(**new** TCompactProtocol(transport));
18. transport->open();
20. User u;
21. u.uid = "leo";
22. u.uname = "yueyue";
23. u.usex = 1;
24. u.uage = 3;
26. UserServiceClient client(protocol);
27. client.add(u);
29. User u1;
30. client.get(u1,"lll");
32. transport->close();
33. printf("uid=%s uname=%s usex=%d uage=%d\n", u1.uid.c\_str(), u1.uname.c\_str(), u1.usex, u1.uage);
34. **return** 0;
35. }

5.生成Makefile

Makefile代码  收藏代码

1. BOOST\_DIR = /usr/local/include/boost/
2. THRIFT\_DIR = /usr/local/include/thrift
3. LIB\_DIR = /usr/local/lib
4. GEN\_SRC = ./gen-cpp/acsuser\_types.cpp ./gen-cpp/acsuser\_constants.cpp ./gen-cpp/UserService.cpp
5. default: server client
6. server: UserServer.cpp
7. g++ -g -o UserServer -I${THRIFT\_DIR} -I${BOOST\_DIR}  -I./gen-cpp -L${LIB\_DIR} -lthrift UserServer.cpp ${GEN\_SRC}
8. client: UserClient.cpp
9. g++ -g -o UserClient -lm -pthread -lz -lrt -lssl -I${THRIFT\_DIR} -I${BOOST\_DIR}  -I./gen-cpp -L${LIB\_DIR} -lthrift UserClient.cpp ${GEN\_SRC}
10. clean:
11. $(RM) -r UserServer UserClient

6.启动c++ server

Shell代码  收藏代码

1. ./UserServer

7.测试c++ client

Shell代码  收藏代码

1. ./UserClient

8.写java client文件UserClient.java

Java代码  收藏代码

1. **import** org.apache.thrift.TException;
2. **import** org.apache.thrift.protocol.TCompactProtocol;
3. **import** org.apache.thrift.protocol.TProtocol;
4. **import** org.apache.thrift.transport.TFramedTransport;
5. **import** org.apache.thrift.transport.TNonblockingSocket;
6. **import** org.apache.thrift.transport.TSocket;
7. **import** org.apache.thrift.transport.TTransport;
8. **import** org.apache.thrift.transport.TTransportException;
10. //import UserService.Client;
12. **public** **class** UserClient {
13. **private** **void** start() {
14. **try** {
15. TTransport socket = **new** TSocket("localhost", 9090);
16. //TTransport transport = new TFramedTransport(socket);
17. TProtocol protocol = **new** TCompactProtocol(socket);
19. UserService.Client client = **new** UserService.Client(protocol);
20. socket.open();
21. System.out.println(client.get("lll"));
23. User u = **new** User();
24. u.uid="leojava";
25. u.uname="yueyue";
26. u.usex=**true**;
27. u.uage=3;
28. client.add(u);
29. socket.close();
31. } **catch** (TTransportException e) {
32. e.printStackTrace();
33. } **catch** (TException e) {
34. e.printStackTrace();
35. }
36. }
38. **public** **static** **void** main(String[] args) {
39. UserClient c = **new** UserClient();
40. c.start();
42. }
43. }

编译和运行java client

Shell代码  收藏代码

1. javac -classpath /usr/local/lib/libthrift-0.7.0.jar:/usr/local/lib/log4j-1.2.14.jar:/usr/local/lib/commons-logging-1.1.1.jar:/usr/local/lib/slf4j-api-1.5.8.jar UserClient.java ./gen-java/\*.java
2. java -classpath .:./gen-java:/usr/local/lib/libthrift-0.7.0.jar:/usr/local/lib/log4j-1.2.14.jar:/usr/local/lib/commons-logging-1.1.1.jar:/usr/local/lib/slf4j-api-1.5.8.jar:/usr/local/lib/slf4j-log4j12-1.5.8.jar UserClient

9.写Python client文件PythonClient.py

Python代码  收藏代码

1. #!/usr/bin/env python
2. **import** sys
3. sys.path.append('./gen-py')
4. **from** acsuser **import** UserService
5. **from** acsuser.ttypes **import** \*
6. **from** thrift **import** Thrift
7. **from** thrift.transport **import** TSocket
8. **from** thrift.transport **import** TTransport
9. **from** thrift.protocol **import** TCompactProtocol
11. # Make socket
12. transport = TSocket.TSocket('localhost', 9090)
13. # Buffering is critical. Raw sockets are very slow
14. transport = TTransport.TBufferedTransport(transport)
15. # Wrap in a protocol
16. protocol = TCompactProtocol.TCompactProtocol(transport)
17. # Create a client to use the protocol encoder
18. client = UserService.Client(protocol)
19. # Connect!
20. transport.open()
21. # Call Server services
22. u = client.get('lll')
23. **print** 'uid=%s uname=%s usex=%d u.uage=%d' %(u.uid,u.uname,u.usex,u.uage)
25. u1 = User()
26. u1.uid='leo'
27. u1.uname='yueyue'
28. u1.usex=1
29. u1.uage=3
30. client.add(u1)

执行python client代码

Shell代码  收藏代码

1. chmod 777 PythonClient.py
2. ./PythonClient.py

* [sample.tar.gz](http://dl.iteye.com/topics/download/acdaf443-019d-325d-b341-69958e407d40) (2.1 KB)