|  |  |
| --- | --- |
| Thrift使用笔记 | [☝](http://lyarbean.github.io/%E7%A8%8B%E5%BA%8F%E8%A8%AD%E8%A8%88/Thrift%E4%BD%BF%E7%94%A8%E7%AD%86%E8%A8%98/) |

发布于31 Jul 2016

标签

* (1) [thrift](http://lyarbean.github.io/tags.html#thrift-ref)

Thrift

Thrift是一个知名的RPC框架，由Facebook贡献给Apache，Facebook也在维护其独立的分支，并且提供了一个新的生成器选项Cpp2，但由此生成的代码依赖于Facebook其它开源的组件以及C+ +11甚至是C++14的支持，不适合于较保守的项目，例如VS2010+Qt 4.8.5。

Thrift通过生成器选项cpp可以生成C++同步的接口代码，而通过加入cob\_style选项，还能生成异步的接口代码；无论哪种情况，可并发的同步接口也一并生成。由于这个可并发的同步接口，我们将不得不链接到boost的线程库。

当接口实现代码完成后，我们就能配合Thrift提供的协议、传输器、处理器和服务器进行工作。Thfit提供了多种实用的协议，如TBinaryProtocol、TCompactProtocol和TJSONProtocol等；传输器方面，Thrfit提供了许多实现，包括TBufferedTransport、TFramedTransport和TMemoryBuffer，TFDTransport、TFileTransport、THttpTransport、 TPipe、TSocket等；而处理器方面，Thrift提供了同步和异步的选择，取决我们是否使用cob类型；Thrift在服务器方面也提供了多种选择，单线程的TSimpleServer，非阻塞的TNonblockingServer，多线程的TThreadedServer以及使用线程池的TThreadPoolServer。

同步和异步

虽然Thrift在每个层面多彩多样，但是并不能任意搭配。Thrift提供的传输器基本上是同步的，意味着我们无法使用异步接口。对于下面的IDL：

service Test {

bool has(1: string item),

}

就客户端，对比Thrift生成的同步和异步的代码：

class TestClient : virtual public TestIf {

public:

TestClient(boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> prot) {

setProtocol(prot);

}

TestClient(boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> iprot, boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> oprot) {

setProtocol(iprot,oprot);

}

// 此處略去

protected:

boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> piprot\_;

boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> poprot\_;

::apache::thrift::protocol::TProtocol\* iprot\_;

::apache::thrift::protocol::TProtocol\* oprot\_;

};

class TestCobClient : virtual public TestCobClIf {

public:

TestCobClient(boost::shared\_ptr< ::apache::thrift::async::TAsyncChannel> channel, ::apache::thrift::protocol::TProtocolFactory\* protocolFactory) :

channel\_(channel),

itrans\_(new ::apache::thrift::transport::TMemoryBuffer()),

otrans\_(new ::apache::thrift::transport::TMemoryBuffer()),

piprot\_(protocolFactory->getProtocol(itrans\_)),

poprot\_(protocolFactory->getProtocol(otrans\_)) {

iprot\_ = piprot\_.get();

oprot\_ = poprot\_.get();

}

// 此處略去

protected:

boost::shared\_ptr< ::apache::thrift::async::TAsyncChannel> channel\_;

boost::shared\_ptr< ::apache::thrift::transport::TMemoryBuffer> itrans\_;

boost::shared\_ptr< ::apache::thrift::transport::TMemoryBuffer> otrans\_;

boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> piprot\_;

boost::shared\_ptr< ::apache::thrift::protocol::TProtocol> poprot\_;

::apache::thrift::protocol::TProtocol\* iprot\_;

::apache::thrift::protocol::TProtocol\* oprot\_;

};

显而易见，Thrift的异步客户端的传输器需要使用TAsyncChannel，而非TTransport！这是个本质的问题。Thrift提供的异步方案，在不分协议下，只有一种， 即使用TEvhttpClientChannel和TEvhttpServer。通过阅读这两个类的实现，我们不难理解Thrift异步架构的工作原理。

实现自己的异步方案

在必须实现异步而且不使用http的情况下，可以走的路只有自己实现一套。首先，先看TAsyncChannel在客户端充当的角色。

virtual void TAsyncChannel::sendAndRecvMessage(const VoidCallback& cob,

apache::thrift::transport::TMemoryBuffer\* sendBuf,

apache::thrift::transport::TMemoryBuffer\* recvBuf)

这个虚接口在客户端发起请求时调用，而且就在协议把请求写入otrans后调用的。它的作用之一是将otrans的数据发往服务器端；另一个责任则是接收服务器端发回的结果，并将之写入itrans；最后一项责任则是紧接着调用cob。这个cob由客户端传入，一般是调用recv\_XXX() 来取出结果。例如以下的lambda：

try {

client->has([](TestCobClient\* client) {

bool has = client->recv\_has();

} , a);

} catch (apache::thrift::TApplicationException& x){

//...

}

在服务器端， 其异步接口的实现方式和同步接口的实现方式在处理器和传输器的逻辑上是恰好相反的。同步模式下，由处理器通过传输器去（阻塞地）读取整份数据，然后交给handler去处理，之后随即通过传输器发回客户端。而在异步模式下，由传输层读取整份数据后，转交异步处理器处理；异步处理器将数据交由handler处理， handler的处理过程可以是异步的，例如不是直接调用，而是塞入事件循环；等到完成时，再调用cob，此时cob会将结果发回客户端。

对于使用Qt的项目，利用事件循环，要实现异步方案是很简单的。例如，采用QLocalSocket和QLocalServer即可实现基于管道或者UNIX domain socket（取决于平台）的方案。

坑

说起坑，只有更坑。

话说我们项目的大计划是向异步迈步的，但是不想使用Http协议，在Windows上管道在项目中有应用场景，因此TPipe被优先考虑。当时我对Thrift所知甚少，尽管Thrift没有提供现成和TPipe协作的异步方案，但还是想着桥到船头自然直，后面再慢慢改为异步的。不过坑的事情还没到异步问题上。TPipe本身实现存在问题，TPipe在close后再连上并读取数据会发生TPipe: GetOverlappedResult failed异常！详情可见[Thrift-3238](https://issues.apache.org/jira/browse/THRIFT-3238)。

就TPipe这个问题，我直接就用QLocalSocket写了个异步的方案。然而还是有坑！如果有客户端的QLocalSocket close后再连上，之后读取数据，发现QLocalSocket会signal error，错误内容为连接错误，详情可参见[QtBug-13646](https://bugreports.qt.io/browse/QTBUG-13646)。在Qt 4.8.5上，这个坑还一直存在，可能Qt5已经修复了，我还没验证。规避的方法则是由服务器端断开连接，这要求有客户端主动向服务器请求关闭。

现在想想TPipe和QLocalSocket的坑很有可能是同一个问题：开发文档不详细！