[**关于thrift的一些探索——thrift序列化技术**](https://www.dplord.com/2016/05/12/thrift-serialize-technology-of-some-research-about-thrift/)

作者: dplord, 访问量 1704 

thrift的IDL，相当于一个钥匙。而thrift传输过程，相当于从两个房间之间的传输数据。

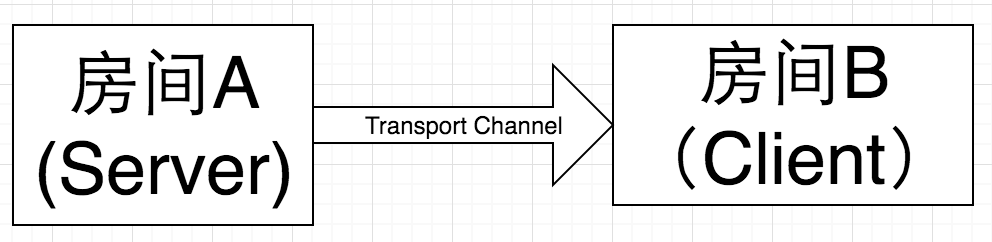
[](https://www.dplord.com/wp-content/uploads/2016/04/QQ20160426-3@2x.png)

图-1

（因为Thrift采用了C/S模型，不支持双向通信：client只能远程调用server端的RPC接口，但client端则没有RPC供server端调用，这意味着，client端能够主动与server端通信，但server端不能主动与client端通信而只能被动地对client端的请求作出应答。所以把上图-1中的箭头，画为单向箭头更为直观）基于上图，Thrift的IDL文件的作用可以描述为，从房间A要传递一批数据给房间B，把数据装在箱子里，锁上锁，然后通过Transport Channel把带锁的箱子给房间B，而Thrift IDL就是一套锁跟钥匙。房间A跟房间B都有同样的一个thrift IDL文件，有了这个文件就可以生成序列化跟反序列化的代码，就像锁跟钥匙一样。而一旦没有了Thrift IDL文件，房间A无法将数据序列化好锁紧箱子，房间B没有钥匙打开用箱子锁好传输过来的数据。因此，IDL文件的作用就在此。

**一、为什么要用Thrift序列化, 不用纯文本的协议**

我能想到的用thrift提供的thrift binary protocol而不用json、xml等纯文本序列化协议的几个好处如下:

* 序列化后的体积小, 省流量
* 序列化、反序列化的速度更快，提高性能
* 兼容性好，一些接口，涉及多种语言的int32、int64等等跟语言、机器、平台有关, 用纯文本可能有兼容性的问题
* 结合thrift transport技术，可以RPC精准地传递对象,而纯文本协议大多只能传递数据，而不能完完全全传递对象

关于以上几点我个人认为的好处，下面作一下简单解答

**1.1 序列化后的体积小, 省流量**

给出测试的IDL Services\_A.thrift内容如下，定义了一些数据格式，这个结构体数据复杂度一般，有list、也有list对象、也有一些基本的struct等等。

namespace php Services.test.A

namespace java Services.tets.A

struct student{

1:required string studentName, #学生姓名

2:required string sex, #性别

3:required i64 age, #学生年龄

}

struct banji{

1:required string banjiName, #班级名称

2:required list<student> allStudents, #所有学生

}

struct school {

1:required string schoolName,

2:required i64 age,

3:required list<string> zhuanye, #所有专业

4:required list<banji> allBanji, #所有班级

}

分别把Services\_A.thrift 序列化为json、跟thrift.bin文件，序列化到文件中。对比文件大小。

这里用php写了一个例子, 代码如下:

<?php

/\*\*

\* Created by PhpStorm.

\* User: dengpan

\* Date: 16/4/21

\* Time: 12:39

\*/

ini\_set('memory\_limit','10240M');

require\_once \_\_DIR\_\_ . "/Thrift/ClassLoader/ThriftClassLoader.php";

use Thrift\ClassLoader\ThriftClassLoader;

use Thrift\Protocol\TBinaryProtocol;

use Thrift\Transport\TSocket;

use Thrift\Transport\TFramedTransport;

use Thrift\Transport\TPhpStream;

use Thrift\Transport\TPhpStreamMy;

use Thrift\Transport\TBufferedTransport;

$loader = new ThriftClassLoader();

$loader->registerNamespace('Thrift', \_\_DIR\_\_);

$loader->registerNamespace('Services', \_\_DIR\_\_);

$loader->registerDefinition('Services', \_\_DIR\_\_);

$loader->register();

require "Types.php";

$school = [];

$school['schoolName'] = "hahhaha";

$school['age'] = 60;

$school['zhuanye'] = [

'专业1',

'专业2',

'专业3',

];

$nameArr = ["张三", "李四", "王五", "王菲", "张韶涵", "王祖贤", "范冰冰", "新垣结衣", "詹姆斯", "诺维茨基"];

$sexArr = ["男", "女"];

$allBanji = [];

for($i = 0; $i < 1000; $i ++)

{

$banji = [];

$banji['banjiName'] = "计算机" + $i + "班";

for($j = 0; $j < 1000; $j ++) {

$banji['allStudents'][] = new student(

[

'studentName' => $nameArr[rand(0, count($nameArr) - 1)],

'sex' => $sexArr[rand(0, count($sexArr) - 1)],

'age' => rand(0, 6) + 18,

]

);

}

$allBanji[] = new banji($banji);

}

$school['allBanji'] = $allBanji;

$sc = new school($school);

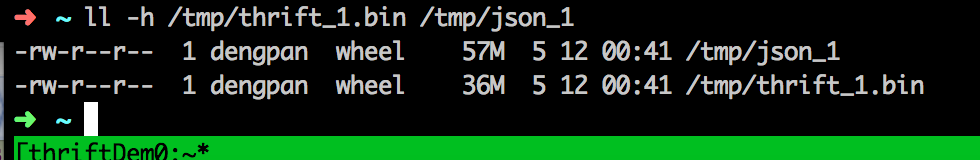
$str = json\_encode($school);

file\_put\_contents("/tmp/json\_1", $str);

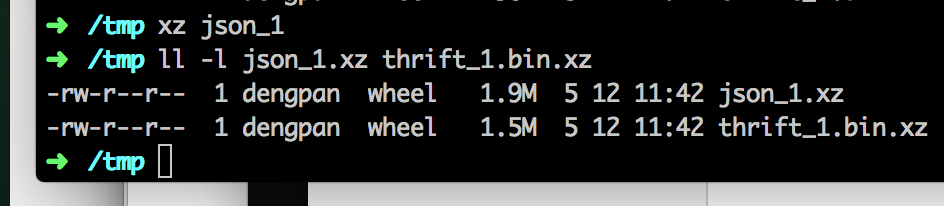
$transport = new TBufferedTransport(new TPhpStreamMy(null, '/tmp/thrift\_1.bin'), 1024, 1024);

$sc->write(new TBinaryProtocol($transport, true, true));

最终的结果， json文件跟thrift binary文件大小如下图:

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-0@2x.png)

thrift binary file是json文件大小的 0.63。为什么文件小这么多，我认为有以下几个原因，① json是文本的，而thrift是二进制的，文本文件跟二进制文件的区别，可以参考我的另一篇文章, [二进制文件跟普通文本文件的区别](https://www.dplord.com/2016/03/14/diffenerce-between-binary-file-and-text-file/)。② thrift序列化过程中，由于IDL在client、servo端都有一套，所以没有传输一些字段比如field name等等，只需要传递field id, 这样也是一些节省字节的方法。另外，提醒一下，序列化跟压缩不是一回事，比如上面的文件压缩为xz看看大小如下图:

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-5@2x.png)

压缩后体积变为原来的3.3%、4.4%。可以先序列化再压缩传输，压缩的compress-level要指定合理，不然压缩跟解压缩占据系统资源而且耗时大。

**1.2序列化、反序列化的速度更快，提高性能**

下面给出一个序列化反序列化测试

分别用① java Thrift binary protocol跟java fastjson库去序列以上对应格式为json, ②用C++ Thrift binary protocol与c++ jsoncpp序列化，对比速度,

java 测试代码如下:

**public** **class** **ThriftDataWrite3** **{**

**private** **static** **final** Random sRandom **=** **new** Random**();**

**public** **static** **void** main**(**String**[]** args**)** **throws** IOException**,** TException**{**

*//构造school对象*

String**[]** nameArr **=** **{**"张三"**,** "李四"**,** "王五"**,** "赵6"**,** "王祖贤"**,** "赵敏"**,** "漩涡鸣人"**,** "诺维茨基"**,** "邓肯"**,** "克莱尔丹尼斯"**,** "长门"**,** "弥彦"**,** "威少"**};**

**int** nameArrLength **=** nameArr**.**length**;**

school sc **=** **new** school**();**

sc**.**setSchoolName**(**"哈哈哈哈哈哈"**);**

sc**.**setAge**(**12**);**

List**<**String**>** l **=** **new** ArrayList**<>();**

**for** **(int** i **=** 0**;** i **<** 100**;** i**++)** **{**

l**.**add**(**"专业" **+** i**);**

**}**

sc**.**setZhuanye**(**l**);**

List**<**banji**>** allBanji **=** **new** ArrayList**<**banji**>();**

**for** **(int** i **=** 0**;** i **<** 1000**;** i**++)** **{**

banji bj **=** **new** banji**();**

bj**.**setBanjiName**(**"班级" **+** i**);**

List allStuents **=** **new** ArrayList**<**student**>();**

**for** **(int** j **=** 0**;** j **<** 1000**;** j **++)** **{**

allStuents**.**add**(**

**new** student**(**

nameArr**[**sRandom**.**nextInt**(**nameArrLength**)],**

**((**sRandom**.**nextInt**(**2**)** **==** 0**)** **?** "男" **:** "女"**),**

**(**sRandom**.**nextInt**(**10**)** **+** 18**)**

**)**

**);**

**}**

bj**.**setAllStudents**(**allStuents**);**

allBanji**.**add**(**bj**);**

**}**

sc**.**setAllBanji**(**allBanji**);**

*//①序列化为thrift binary protocol*

**final** **long** startTime **=** System**.**currentTimeMillis**();**

TSerializer serializer **=** **new** TSerializer**(new** TBinaryProtocol**.**Factory**());**

**for** **(int** i **=** 0**;** i **<** 200**;** i**++)** **{**

**byte[]** bytes **=** serializer**.**serialize**(**sc**);**

*//serializer.toString(sc);*

**}**

**final** **long** endTime **=** System**.**currentTimeMillis**();**

System**.**out**.**println**(**"thrift序列化200次时间为" **+** **(**endTime **-** startTime**)** **+** "ms"**);**

*//②序列化为json*

**final** **long** endTime7 **=** System**.**currentTimeMillis**();**

**for** **(int** i **=** 0**;** i **<** 200**;** i**++)** **{**

JSON**.**toJSONString**(**sc**);**

**}**

**final** **long** endTime2 **=** System**.**currentTimeMillis**();**

System**.**out**.**println**(**"json序列化200次时间为" **+** **(**endTime2 **-** endTime7**)** **+** "ms"**);**

*//准备待序列化的数据*

**byte[]** bytes **=** serializer**.**serialize**(**sc**);**

String jsonStr **=** JSON**.**toJSONString**(**sc**);**

*//③反序列thrift binary data*

**final** **long** endTime3 **=** System**.**currentTimeMillis**();**

TDeserializer tDeserializer **=** **new** TDeserializer**();**

**for** **(int** i **=** 0**;** i **<** 200**;** i**++)** **{**

school sc1 **=** **new** school**();**

tDeserializer**.**deserialize**(**sc1**,** bytes**);**

*// System.out.println(sc1.toString());*

**}**

**final** **long** endTime4 **=** System**.**currentTimeMillis**();**

System**.**out**.**println**(**"thrift反序列化200次时间为" **+** **(**endTime4 **-** endTime3**)** **+** "ms"**);**

*//④反序列化json*

**final** **long** endTime5 **=** System**.**currentTimeMillis**();**

**for** **(int** i **=** 0**;** i **<** 200**;** i**++)** **{**

JSON**.**parseObject**(**jsonStr**,** sc**.**getClass**());**

*// System.out.println(JSON.parseObject(jsonStr, sc.getClass()).toString());*

**}**

**final** **long** endTime6 **=** System**.**currentTimeMillis**();**

System**.**out**.**println**(**"json反序列化200次时间为" **+** **(**endTime6 **-** endTime5**)** **+** "ms"**);**

**}**

**}**

java的序列化thrift、json，反序列化thrift跟json的结果为:

thrift序列化200次时间为82482ms  
json序列化200次时间为167954ms  
thrift反序列化200次时间为42919ms  
json反序列化200次时间为207896ms

其中可以看出java中thrift的序列化速度是fastjson 序列化json的2倍多，thrift反序列化的速度是fastjson反序列化json的接近5倍。

再看C++测试同样的Case， C++代码如下:

①  头文件thriftSerializeTest.h

//

// Created by 邓攀邓攀 on 16/5/11.

//

#ifndef UNTITLED\_THRIFTSERIALIZETEST\_H

#define UNTITLED\_THRIFTSERIALIZETEST\_H

//my own add header start

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

#include "service\_a\_types.h"

//my own add header end

using namespace std;

class thriftSerializeTest {

public:

static school initializeSchool();

static void jsonSerialize(school sc, string\* str, int\* time);

static void thriftBinarySerialize(school sc,string\* str, int\* time);

static void jsonDeserialize(string str, int\* time);

static void thriftBinaryDeserialize(string str, int\* time);

};

#endif //UNTITLED\_THRIFTSERIALIZETEST\_H

② cpp 文件

//

// Created by 邓攀邓攀 on 16/5/11.

//

#include "thriftSerializeTest.h"

#include "service\_a\_types.h"

#include <sstream>

#include <sys/time.h>

#include "thrift/transport/TBufferTransports.h"

#include "thrift/protocol/TBinaryProtocol.h"

#include "json/json.h"

#define MAX\_NUM 200

using apache::thrift::protocol::TBinaryProtocol;

using apache::thrift::transport::TMemoryBuffer;

school thriftSerializeTest::initializeSchool()

{

school sc;

vector<string> nameArr{"张三", "李四", "王五", "赵6", "王祖贤", "赵敏", "漩涡鸣人", "诺维茨基", "邓肯", "克莱尔丹尼斯", "长门", "弥彦", "威少"};

int nameArrLength = nameArr.size();

sc.\_\_set\_age(12);

sc.\_\_set\_schoolName("哈哈哈哈哈哈");

vector<string> v;

for (int i = 0; i < 100; ++i) {

std::ostringstream oss;

oss << i;

v.push\_back("专业" + oss.str());

}

sc.\_\_set\_zhuanye(v);

vector<banji> allBanji;

for (int j = 0; j < 1000; ++j) {

banji bj;

std::ostringstream oss;

oss << j;

bj.\_\_set\_banjiName("班级" + oss.str());

vector<student> allStudents;

for (int i = 0; i < 1000; ++i) {

student student1;

student1.\_\_set\_age(18 + (rand() % 10));

student1.\_\_set\_sex((rand() % 2) ? "男" : "女");

student1.\_\_set\_studentName(nameArr.at(rand() % nameArr.size()));

allStudents.push\_back(student1);

}

bj.\_\_set\_allStudents(allStudents);

allBanji.push\_back(bj);

}

sc.\_\_set\_allBanji(allBanji);

return sc;

}

void thriftSerializeTest::jsonSerialize(school sc, string \*str, int \*time) {

Json::Value root;

root["studentName"] = Json::Value(sc.schoolName);

root["age"] = Json::Value((int)sc.age);

for(auto v : sc.zhuanye)

{

root["zhuanye"].append(v);

}

for(auto v: sc.allBanji)

{

Json::Value banji;

banji["banjiName"] = Json::Value(v.banjiName);

for(auto k : v.allStudents)

{

Json::Value student;

student["studentName"] = Json::Value(k.studentName);

student["age"] = Json::Value((int)k.age);

student["sex"] = Json::Value(k.sex);

banji["allStudents"].append(student);

}

root["allBanji"].append(banji);

}

Json::FastWriter fw;

struct timeval start, end;

gettimeofday(&start, NULL);

for (int i = 0; i < MAX\_NUM; ++i) {

fw.write(root);

}

gettimeofday(&end, NULL);

int t = 1000\*(end.tv\_sec - start.tv\_sec);

\*time = t;

\*str = fw.write(root);

}

void thriftSerializeTest::thriftBinarySerialize(school sc,string \*str, int \*time) {

struct timeval start, end;

gettimeofday(&start, NULL);

for (int i = 0; i < MAX\_NUM; ++i) {

boost::shared\_ptr<TMemoryBuffer> buffer(new TMemoryBuffer());

boost::shared\_ptr<TBinaryProtocol> binaryProtcol(new TBinaryProtocol(buffer));

sc.write((binaryProtcol.get()));

buffer->getBufferAsString();

}

gettimeofday(&end, NULL);

int t = 1000\*(end.tv\_sec - start.tv\_sec);

\*time = t;

boost::shared\_ptr<TMemoryBuffer> buffer(new TMemoryBuffer());

boost::shared\_ptr<TBinaryProtocol> binaryProtcol(new TBinaryProtocol(buffer));

sc.write((binaryProtcol.get()));

\*str = buffer->getBufferAsString();

}

void thriftSerializeTest::jsonDeserialize(string str, int \*time) {

Json::Reader reader;

Json::Value root;

int num = MAX\_NUM;

struct timeval start, end;

gettimeofday(&start, NULL);

while (num > 0) {

//确保成功解析

if (reader.parse(str.c\_str(), root)) {

num --;

}

}

gettimeofday(&end, NULL);

int t = 1000\*(end.tv\_sec - start.tv\_sec);

\*time = t;

}

void thriftSerializeTest::thriftBinaryDeserialize(string str, int \*time) {

struct timeval start, end;

gettimeofday(&start, NULL);

boost::shared\_ptr<TMemoryBuffer> buffer(new TMemoryBuffer());

boost::shared\_ptr<TBinaryProtocol> binaryProtcol(new TBinaryProtocol(buffer));

school sc;

for (int i = 0; i < MAX\_NUM; ++i) {

buffer->resetBuffer((uint8\_t\*)str.data(), str.length());

sc.read(binaryProtcol.get());

}

gettimeofday(&end, NULL);

int t = 1000\*(end.tv\_sec - start.tv\_sec);

\*time = t;

}

int main() {

//获取school对象

school sc = thriftSerializeTest::initializeSchool();

string thriftStr, jsonStr;

int time;

//①序列化为thrift binary protocol

thriftSerializeTest::thriftBinarySerialize(sc, &thriftStr, &time);

cout << "thrift序列化" << MAX\_NUM << "次时间为" << time << "ms\n";

//②反序列thrift binary data

thriftSerializeTest::thriftBinaryDeserialize(thriftStr, &time);

cout << "thrift反序列化" << MAX\_NUM << "次时间为" << time << "ms\n";

//③jsoncpp序列化为json

thriftSerializeTest::jsonSerialize(sc, &jsonStr, &time);

cout << "jsoncpp序列化json" << MAX\_NUM << "次时间为" << time << "ms\n";

//④jsoncpp反序列化json

thriftSerializeTest::jsonDeserialize(jsonStr, &time);

cout << "jsoncpp反序列化json" << MAX\_NUM << "次时间为" << time << "ms\n";

return 0;

}

编译cpp的命令为:

g++ service\_a\_constants.cpp service\_a\_types.cpp  thriftSerializeTest.cpp -std=c++11 -I/home/dengpan/opt/thrift-0.9.3/include -L/home/dengpan/opt/thrift-0.9.3/lib64 -lthrift -ljsoncpp -O3

C++代码测试结果为:

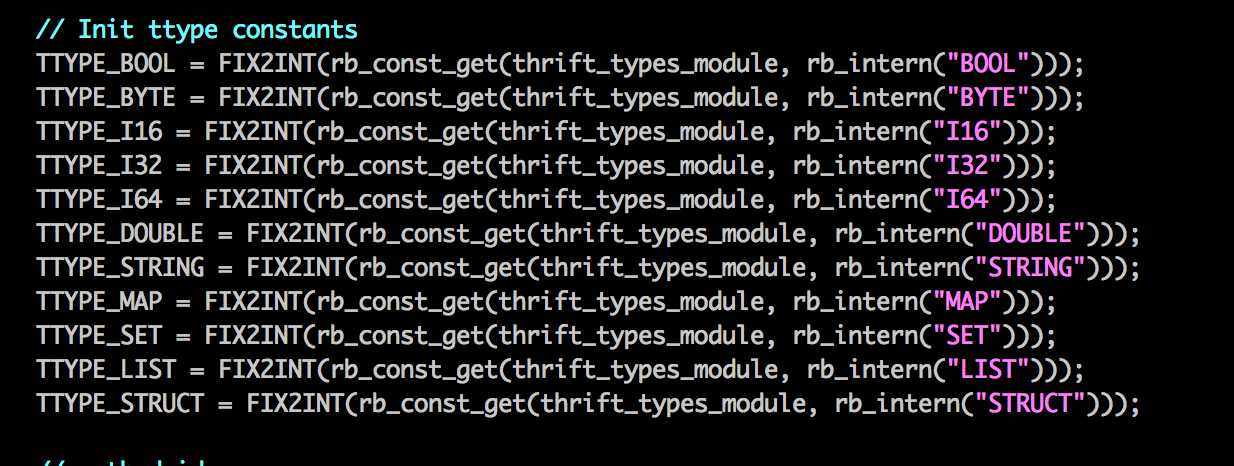
thrift序列化200次时间为23000ms  
thrift反序列化200次时间为48000ms  
jsoncpp序列化json200次时间为330000ms  
jsoncpp反序列化json200次时间为494000ms

其中可以看出c++中thrift的序列化速度是jsoncpp 序列化json的14.3倍，thrift反序列化速度是jsoncpp反序列化json的10.2倍。其中c++序列thrift比java快，c++ jsoncpp严重不如java的fastjson，可能选取更好的c++ json库，可能结果好看点，但是java的fastjson还是很优秀的，用了大量的ASM写法，应该比java、c++绝大多数的json库要好。具体的，有空看看其他的c++ json库来测试一下。读者有更好的库，也可以推荐一下。

thrift对比json，流量更小、序列化反序列化更快，甚至快上5-10倍，这两个特点，在移动端跟实时性要求很高的游戏上，显得非常重要，也吸引人们在这些场景去使用像thrift这样的RPC、序列化的工具。

**1.3兼容性好**

Thrift Types定义了一些基本的Base Type，分别在源代码各个语言中都有一一映射。但是每个语言中，不是所有的定义的类型都支持，Thrift的一些绝大多数语言都支持的base Type有void、bool、i16、i32、i64、double、string。然后thrift通过一一映射，将IDL里面的base Type与各种编程语言里面做了对应，这样可以保证兼容性。而且不会存在java提供的一个接口，一个json字段是i64的，到了C++调用http接口的时候，拿int去取这个字段，丢失了数字的高位。这种情况在跨语言调用，由于每个语言的int、long、short的位数与范围都不同，开发者涉及多语言需要对每个语言的类型范围、位数很清楚才可以避免这样的极端情况的丢失数据，而用了Thrift就不用担心了，它已经帮我们映射好了一些基准类型。后面的thrift code generator也是生成基于TBase类型的对象，用到的也都是thrift base types。关于thrift base type可以参考官方文档: [Thrift Types](http://thrift.apache.org/docs/types)。具体的每个语言支持哪些Types定义，可以看源代码，比如thrift-src中，看到lib/rb/ext/thrift\_native.c 中TType constants相关的为:

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-1@2x.png)看到thrift跟ruby的类型支持情况，没看到binary, 说明ruby不支持binary类型。c++支持thrift的binary类型。其实基准类型，已经足够使用了，建议不要使用不属于i16、i32、i64、double、string、list、map之外的，非通用的TType，不然可能没事找事，碰到一些兼容问题。

**1.4可传递对象**

由于IDL的存在，可以在IDL里面定义一些表示层级关系，数据结构，把我们要传递的对象，翻译成IDL里面的struct, 然后再把IDL编译成对应文件，这样就可以传递对象了。json等文本协议，会丢失部分信息。比如php里面$a =[“name” => 123], json\_encode后，跟 $a= new stdClass();$a->name = 123; json\_encode之后，是一样的。一个是数组，一个是对象。类似的这种序列化中类型丢失或者改变的，其实还有其他的例子。要我们小心的去使用。(可能这个例子举得并不充分，因为php里面数组太灵活了，可以干绝大多数事情)。而Thrift，只要我们定义好IDL，就可以放心的去传递对象了。

**二、序列化方法**

任何一个Struct，Thrift code generator都为它生成了一个对应的class,该类都包含write和read方法，write就是serialie过程, read方法就是unserialize过程。由于Thrift是连带Client调用Service的代码整套生成的，因此想单独拿Thrift序列化一个对象官方没给什么例子，不过各种语言把struct序列化为binary的方法大同小异。我这里研究了下各种语言怎么把对象单独序列化为string，这里一并贴出来。

①   C++ 序列thrift对象为string

boost::shared\_ptr<TMemoryBuffer> buffer(new TMemoryBuffer());  
boost::shared\_ptr<TBinaryProtocol> binaryProtcol(new TBinaryProtocol(buffer));

youThriftObj.write((binaryProtcol.get()));  
buffer->getBufferAsString();

② java序列化thrift对象为string

TSerializer serializer = new TSerializer(new TBinaryProtocol.Factory());  
serializer.toString(bb); //就是序列后之后的string

③ php序列化thrift对象为string

$memBuffer = new \Thrift\Transport\TMemoryBuffer();  
$protocol = new TBinaryProtocol($memBuffer);  
$bbb->write($protocol);  
$str = $memBuffer->getBuffer();

④ php序列化thrift对象到File

由于php中，没有TFileTransport, 因此改写了一下TPhpStream， 改成TPhpStreamMy，这样可以比较方便序列化到文件跟从文件中反序列化，这里也一并给出，[TPhpStreamMy的github gist地址](https://gist.github.com/dplord/7c498b5beb1129e688d16592cd968160)。放到php Thrift lib的Thrift/Transport/TPhpStreamMy.php位置。

序列化thrift对象到file

$transport = new TBufferedTransport(new TPhpStreamMy(null, ‘your-thrift-binary-file-path’), 1024, 1024);  
$yourThriftObj->write(new TBinaryProtocol($transport, true, true));

从文件中读取thrift对象

$transport = new TBufferedTransport(new TPhpStreamMy(‘your-thrift-binary-file-path’, null), 1024, 1024);  
$yourThriftObj->read(new TBinaryProtocol($transport));

其他语言的thrift序列化过程，也是类似步骤。

**三、反序列化方法**

反序列化方法跟序列化方法步骤类似，主要是把write操作改为read操作即可，下面给出一些语言的反序列化方法:

① C++ 反序列string为Object

boost::shared\_ptr<TMemoryBuffer> buffer(new TMemoryBuffer());  
boost::shared\_ptr<TBinaryProtocol> binaryProtcol(new TBinaryProtocol(buffer));  
buffer->resetBuffer((uint8\_t\*)str.data(), str.length()); //str为thrift对象序列化之后的string  
yourThriftObj.read(binaryProtcol.get());

②  java反序列byte[]为Object

TDeserializer tDeserializer = new TDeserializer();  
tDeserializer.deserialize(yourThriftObj, bytes);  //bytes等于thrift对象序列化之后的byte[]

③ php反序列化string为Object

$memBuffer = new \Thrift\Transport\TMemoryBuffer($str); //str为thrift对象序列化之后的string  
$protocol = new TBinaryProtocol($memBuffer);  
$yourThriftObj->read($protocol);

其他语言的thrift反序列化过程，也是类似步骤。

**四、序列化代码解读-字节探究**

这里用一个更简单的thrift service完整例子，用c++实现thrift的THttpServer, php实现thrift的THttpClient, 用wireshark抓包，看看整个通信过程，与字节发包的情况。

示例thrift/service\_b.thrift IDL如下:

struct student {

1:optional string name,

2:optional i64 age,

}

service My\_Services {

student getStudent(1:student st);

}

C++实现thrift THttpServer代码如下:

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

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

**#include "My\_Services.h"**

**#include <thrift/protocol/TBinaryProtocol.h>**

**#include <thrift/server/TSimpleServer.h>**

**#include <thrift/transport/TServerSocket.h>**

**#include <thrift/transport/TBufferTransports.h>**

**#include <thrift/transport/THttpServer.h>**

**#include <thrift/server/TNonblockingServer.h>**

**using** **namespace** **::**apache**::**thrift;

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

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

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

**using** boost**::**shared\_ptr;

**class** **My**\_ServicesHandler **:** **virtual** **public** My\_ServicesIf {

**public:**

My\_ServicesHandler() {

*// Your initialization goes here*

}

**void** getStudent(student**&** \_return, **const** student**&** st) {

*// Your implementation goes here*

\_return.\_\_set\_age(st.age **\*** 2);

\_return.\_\_set\_name(st.name **+** st.name **+** "哈哈哈哈");

printf("getStudents \n");

}

};

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

**int** port **=** 9090;

shared\_ptr**<**My\_ServicesHandler**>** handler(**new** My\_ServicesHandler());

shared\_ptr**<**TProcessor**>** processor(**new** My\_ServicesProcessor(handler));

shared\_ptr**<**TServerTransport**>** serverTransport(**new** TServerSocket("127.0.0.1", port));

shared\_ptr**<**TTransportFactory**>** transportFactory(**new** THttpServerTransportFactory());

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

TSimpleServer server(processor, serverTransport, transportFactory, protocolFactory);

server.serve();

**return** 0;

}

server实现很简单，就是接受student对象，然后把student对象中的student.age = student.age \* 2, student.name = student.name + student.name + “哈哈哈哈”，然后返回这个student对象。

php实现Php THttpClient如下:

<?php

// 引入客户端文件

require\_once \_\_DIR\_\_ . "/../Thrift/ClassLoader/ThriftClassLoader.php";

require "My\_Services.php";

require "Types.php";

use Thrift\ClassLoader\ThriftClassLoader;

use Thrift\Protocol\TBinaryProtocol;

use Thrift\Transport\TSocket;

use Thrift\Transport\TFramedTransport;

$loader = new ThriftClassLoader();

$loader->registerNamespace('Thrift', \_\_DIR\_\_ . "/../");

$loader->registerNamespace('Swoole', \_\_DIR\_\_ . "/../");

$loader->registerNamespace('Services', \_\_DIR\_\_ . "/../");

$loader->registerDefinition('Services', \_\_DIR\_\_ . "/../");

$loader->register();

$transport = new Thrift\Transport\THttpClient("127.0.0.1", 9090);

$protocol = new TBinaryProtocol($transport);

$client = new My\_ServicesClient($protocol);

$transport->open();

$res = $client->getStudent(new student(

[

'name' => '邓攀',

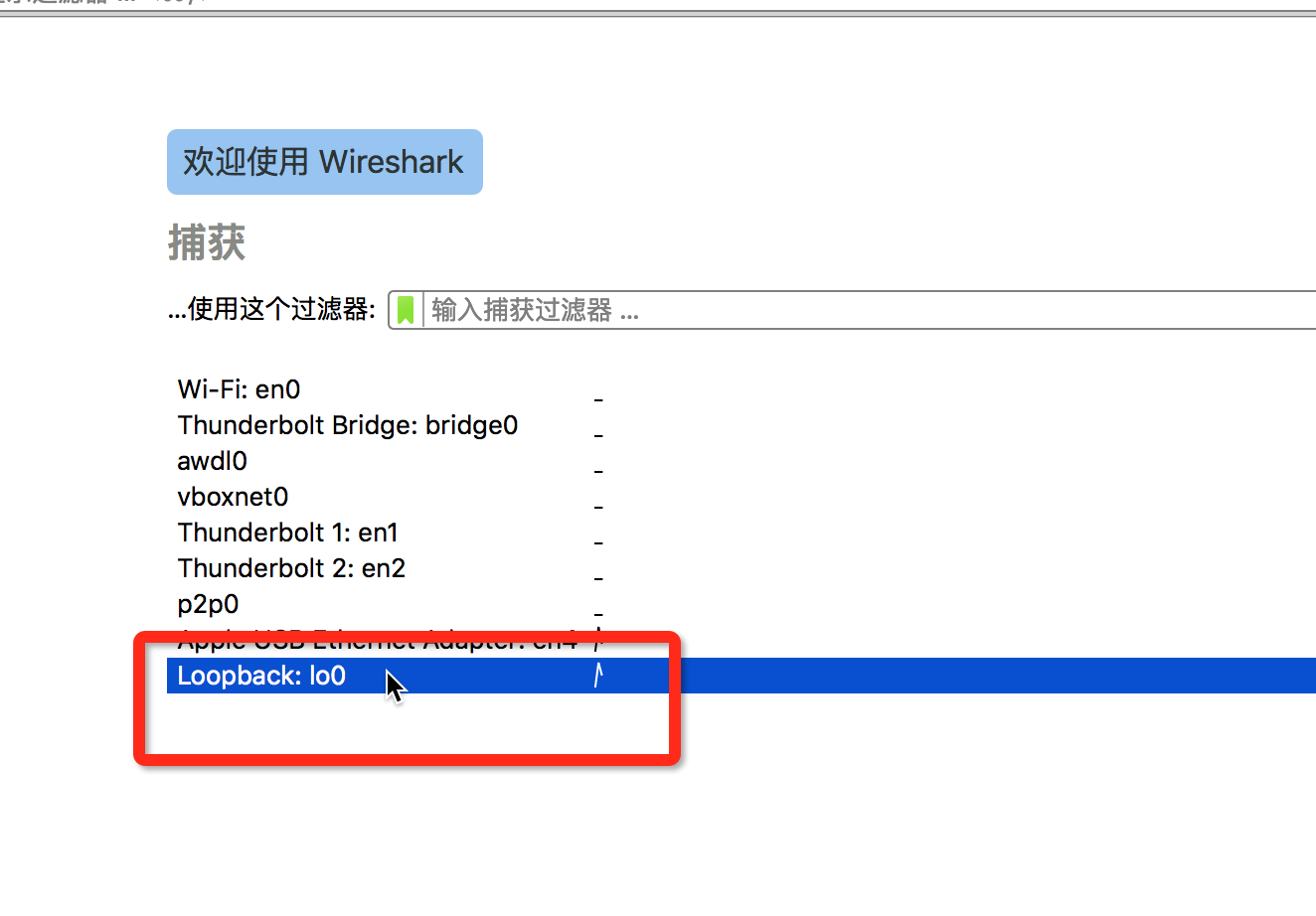
'age' => 1222,

]

));

var\_dump($res);

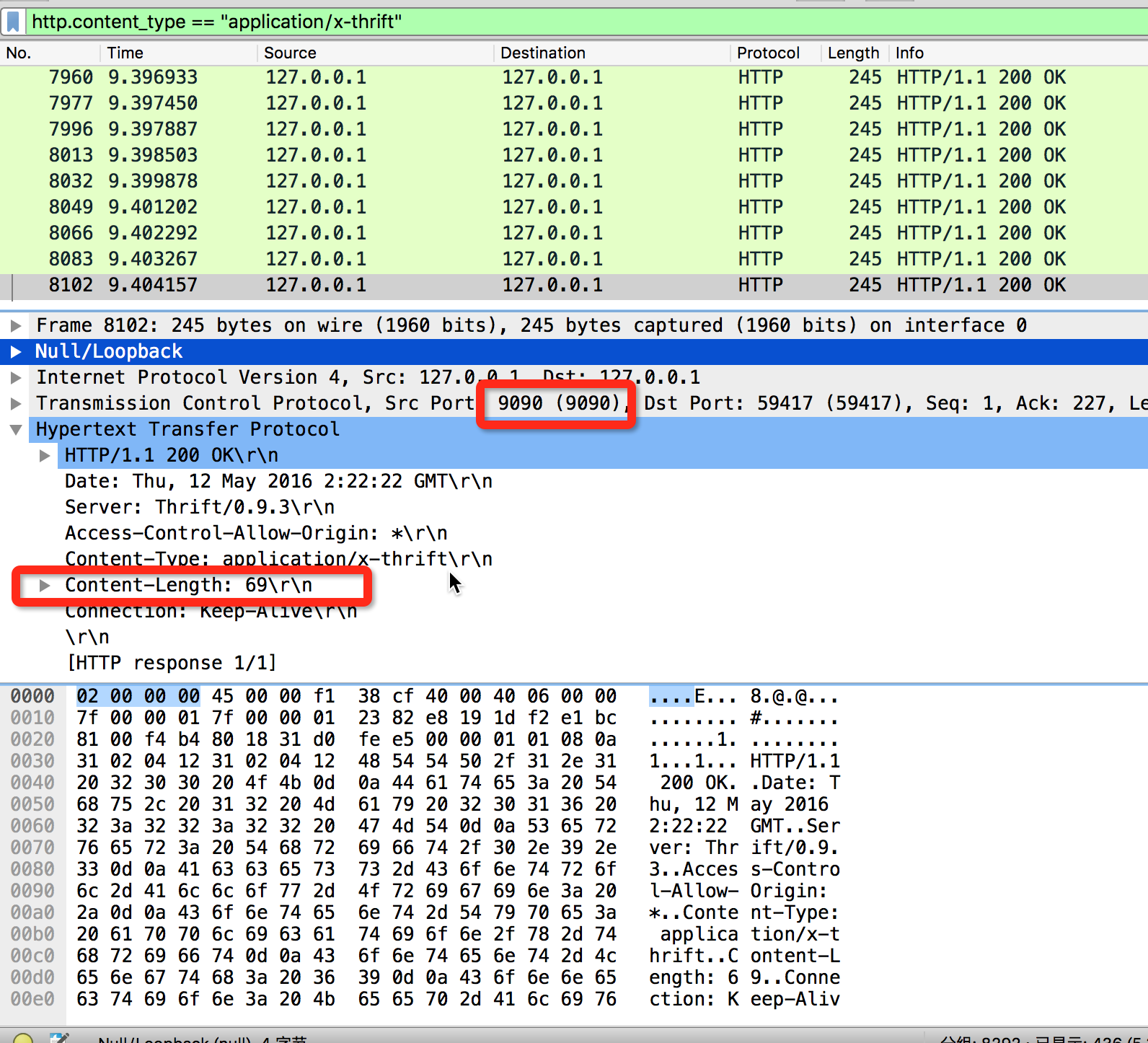
$transport->close();

用wireshark抓包查看http通信过程，注意这个是本地127.0.0.1通信的，要在wireshark抓包指定loopback监听[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-2@2x.png)

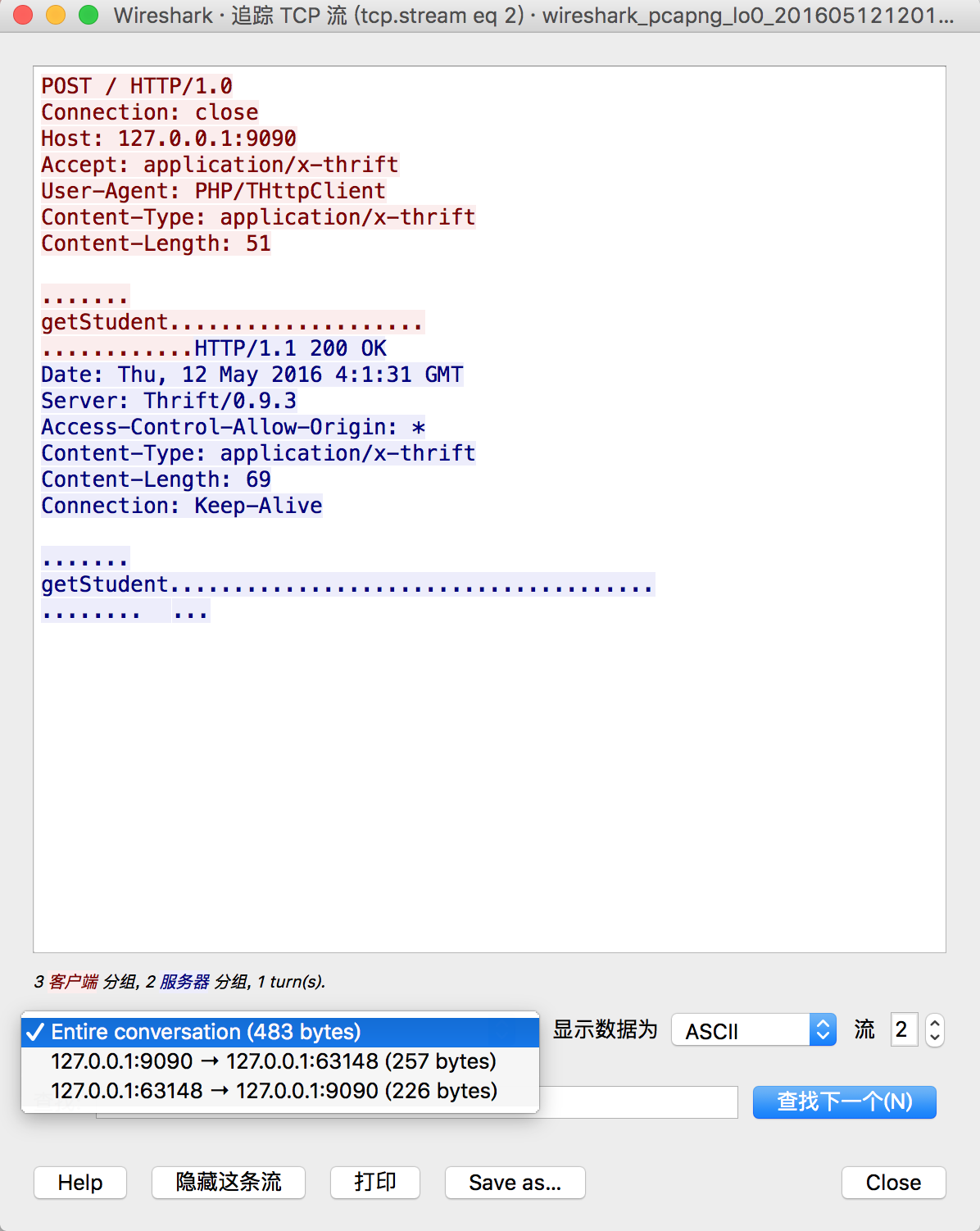
然后在wireshark中指定过滤器

http.content\_type == “application/x-thrift” && http.request.method==”POST”

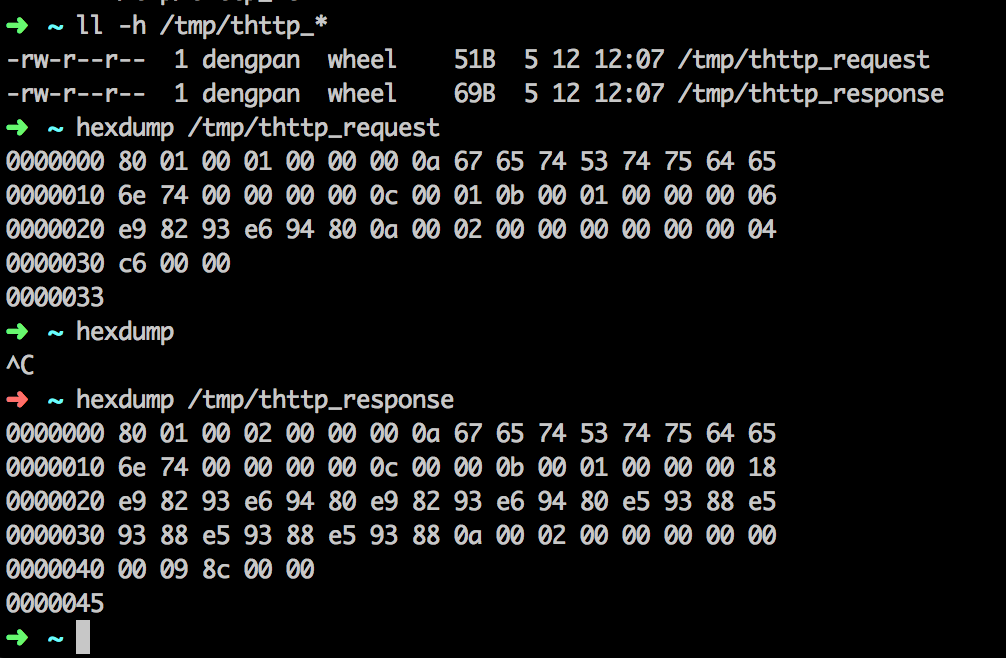
监听到请求，

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-3@2x.png)

可以看到监听到的请求，Content-Length是69B。为了方便跟踪，follow 这个tcp流，

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-6@2x.png)

可以看到http ressquest发送了226字节，http response收到了257字节。我把发送的数据、接收的数据，去除http头，就是纯二进制数据，写进了文件/tmp/thttp\_request、/tmp/thttp\_response中。文件大小与hexdump的文件值如下:

[](https://www.dplord.com/wp-content/uploads/2016/05/QQ20160512-8@2x.png)下面来分析下这个文件的字节内容。以thttp\_response文件为例，我们收到了69个字节，解析出来了name:邓攀邓攀哈哈哈哈,age:2444的数据。

**五、提高一些脚本语言中的序列化性能**

一些脚本语言的序列化性能不太好，再特别注意性能的场景下，建议可以用C/C++实现thrift的序列化，然后写成语言的一个模块、module的形式，给脚本语言去调用，这样可以极大的提高性能。也能利用到thrift的优点。

参考文章:

1. [让Thrift支持双向通信－－董的博客](http://dongxicheng.org/search-engine/thrift-bidirectional-async-rpc/)
2. [Thrift: The Missing Guide](https://diwakergupta.github.io/thrift-missing-guide/#_c)
3. [Thrift 官方文档](http://thrift.apache.org/docs/)