thriftpy

张汝家
rujiazhang@foxmail.com

thriftpy

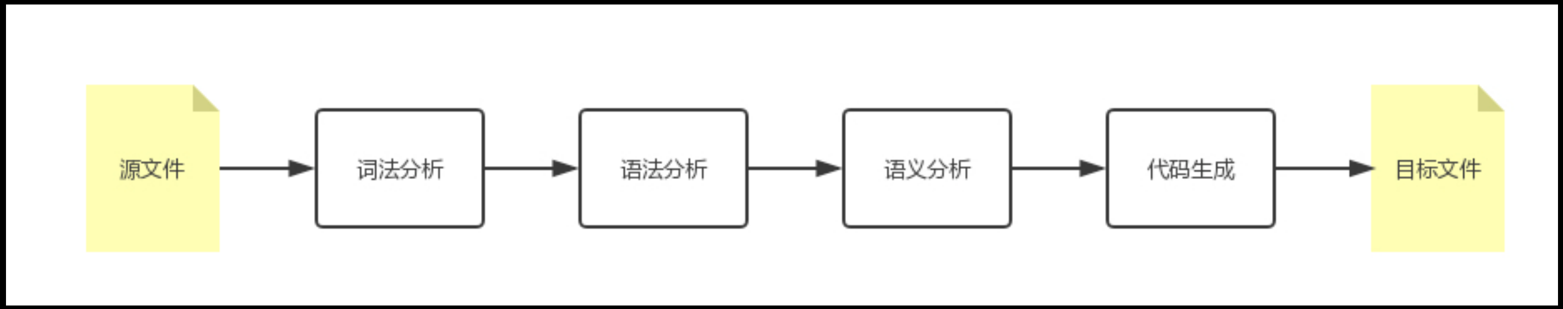
- 0. 微服务的RPC
- 1. thrift作为IDL

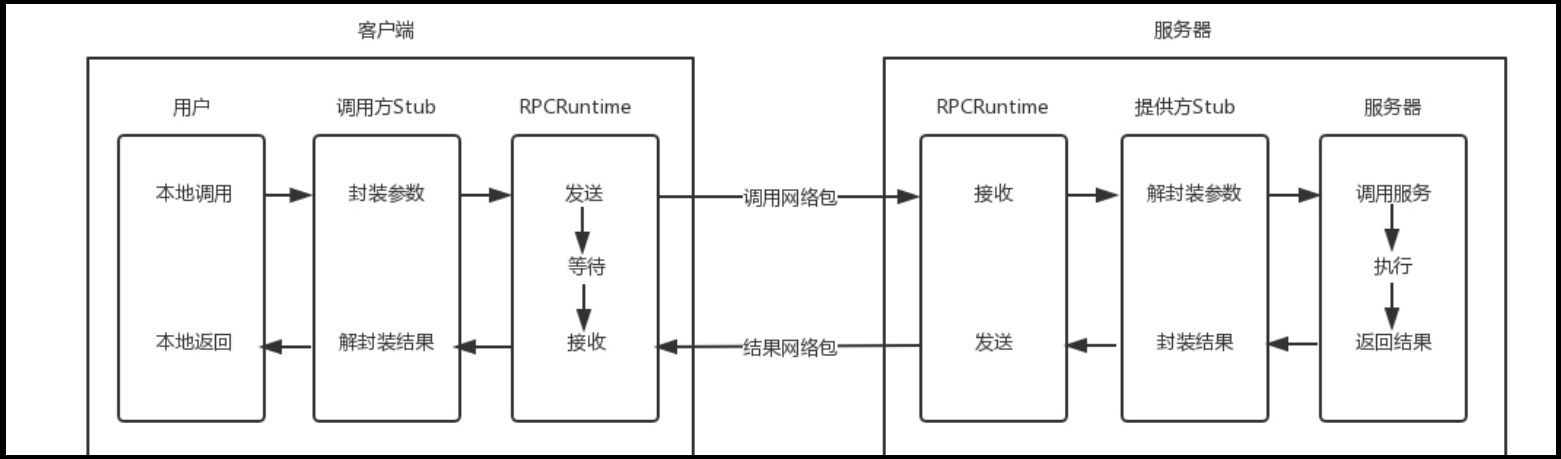
微服务

- microservice
- 轻量级的服务粒度、统一的通信协议、交付自动化、快速迭代且无历史包袱的互联网企业

RPC

- 微服务间通信：RPC（Remote Procedure Call）
- 本地调用->远程调用





RPC

- 协议约定问题
  - 语法
  - 数据表示
- 传输问题
  - 通信性能
  - 通信质量
- 服务发现问题

thrift

- thrift：节约；节俭
- 由facebook开发，转交Apache，后fb又再次开源
- 接口描述语言(Interface description language, IDL)
- RPC协议
- 跨语言
- 论文：http://thrift.apache.org/static/files/thrift-20070401.pdf

thriftpy

- 0. 微服务的RPC
- 1. thrift作为IDL

thrift IDL file demo

```
# person.thrift

const i16 DEFAULT_LIST_SIZE = 10

typedef i32 timestamp

enum PhoneType {
    MOBILE = 0,
    HOME,
    WORK,
}

struct PhoneNumber {
    1: optional PhoneType type = PhoneType.MOBILE,
    2: optional string number,
}

struct Person {
    1: required string name,
    2: optional list<PhoneNumber> phones,
    3: optional timestamp created_at,
}

exception PersonNotExistsError {
    1: optional string message = "Person Not Exists!",
}

service PersonService {
    bool add(1: required Person person);
    bool remove(1: string name) throws (1: PersonNotExistsError not_exists);
    Person get(1: string name) throws (1: PersonNotExistsError not_exists);
}
```

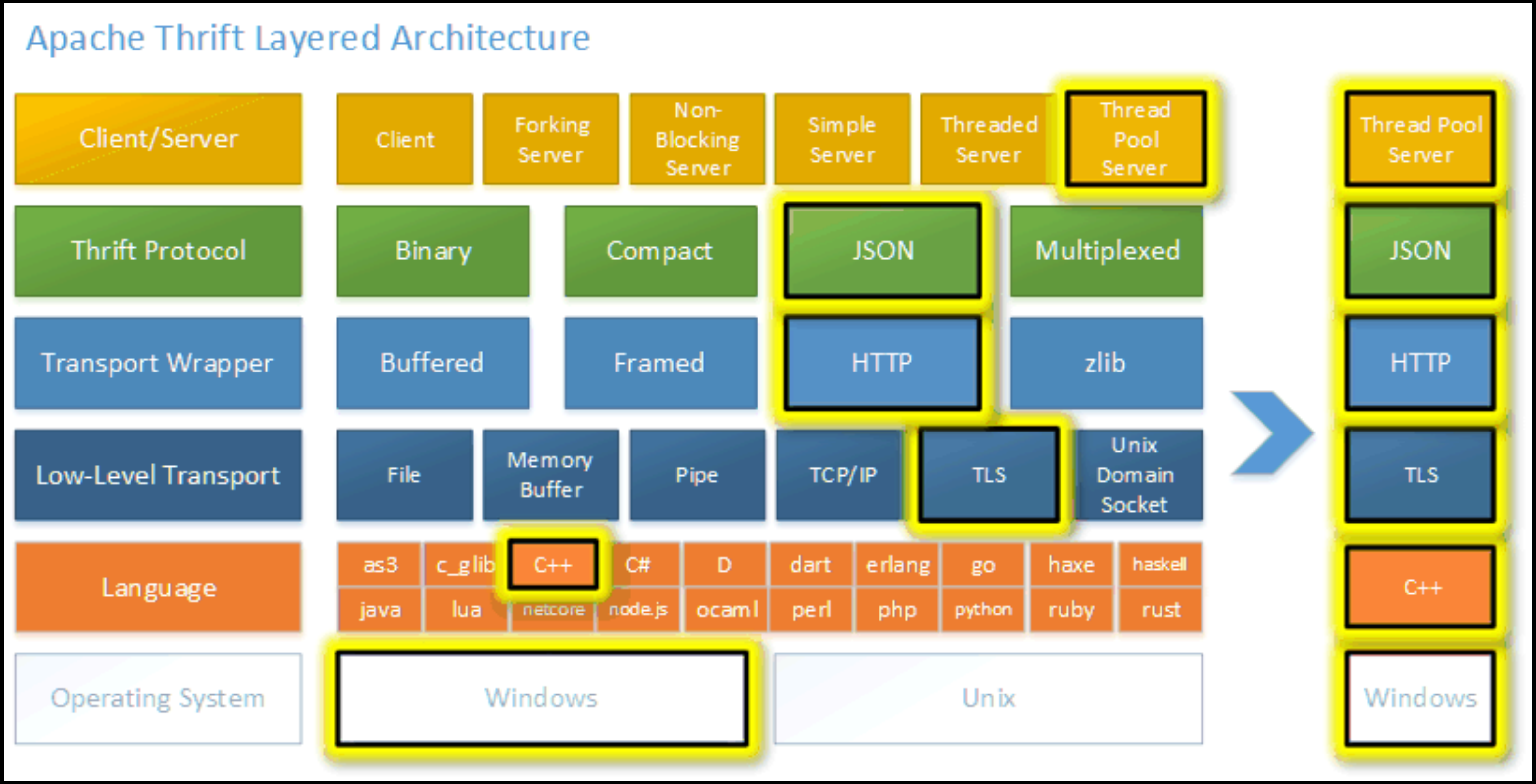
thrift IDL Types

- Services
- Base Types
    - bool
    - byte
    - i16/i32/i64
    - double
    - string
- Structs
- Containers
- Exceptions

thriftpy

- 0. 微服务的RPC
- 1. thrift作为IDL
- 2. thrift作为RPC协议

thrift RPC协议



thrift RPC协议

- Server: single-threaded, event-driven etc
- Processor: compiler generated
- Protocol: binary, JSON, compact etc
- Transport: raw TCP, HTTP etc

thrift transport

- 提供I/O抽象
  - 典型的基于TCP栈的使用流式socket
  - 非典型情况：file, disk…
- 处理读写数据
  - interface: open, close, read, write, flush
- raw transport & transport wrapper

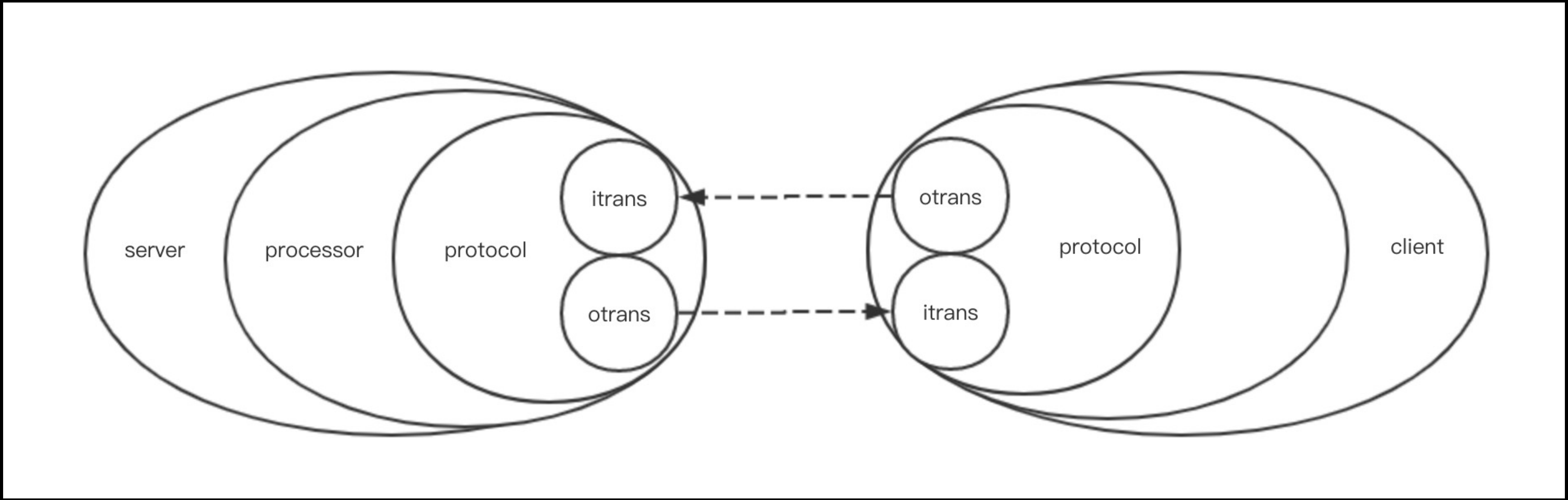
thrift protocol

- 基于transport提供的I/O
- 提供内存中的数据结构与网络流的相互转化
  - interface: read*DataStruct*Begin/write*DataStruct*End
  - *DataStruct*: message, struct, field, map, list, string, i16…
- 编码/解码、序列号/反序列化…
- JSON、XML、Plain Text、Binary、Compact Binary…
- version

thrift processor

- Processor
  - 基于protocol提供的I/O
  - 概括读写能力
  - 组合IDL的生成代码
- Server
  - 最上层

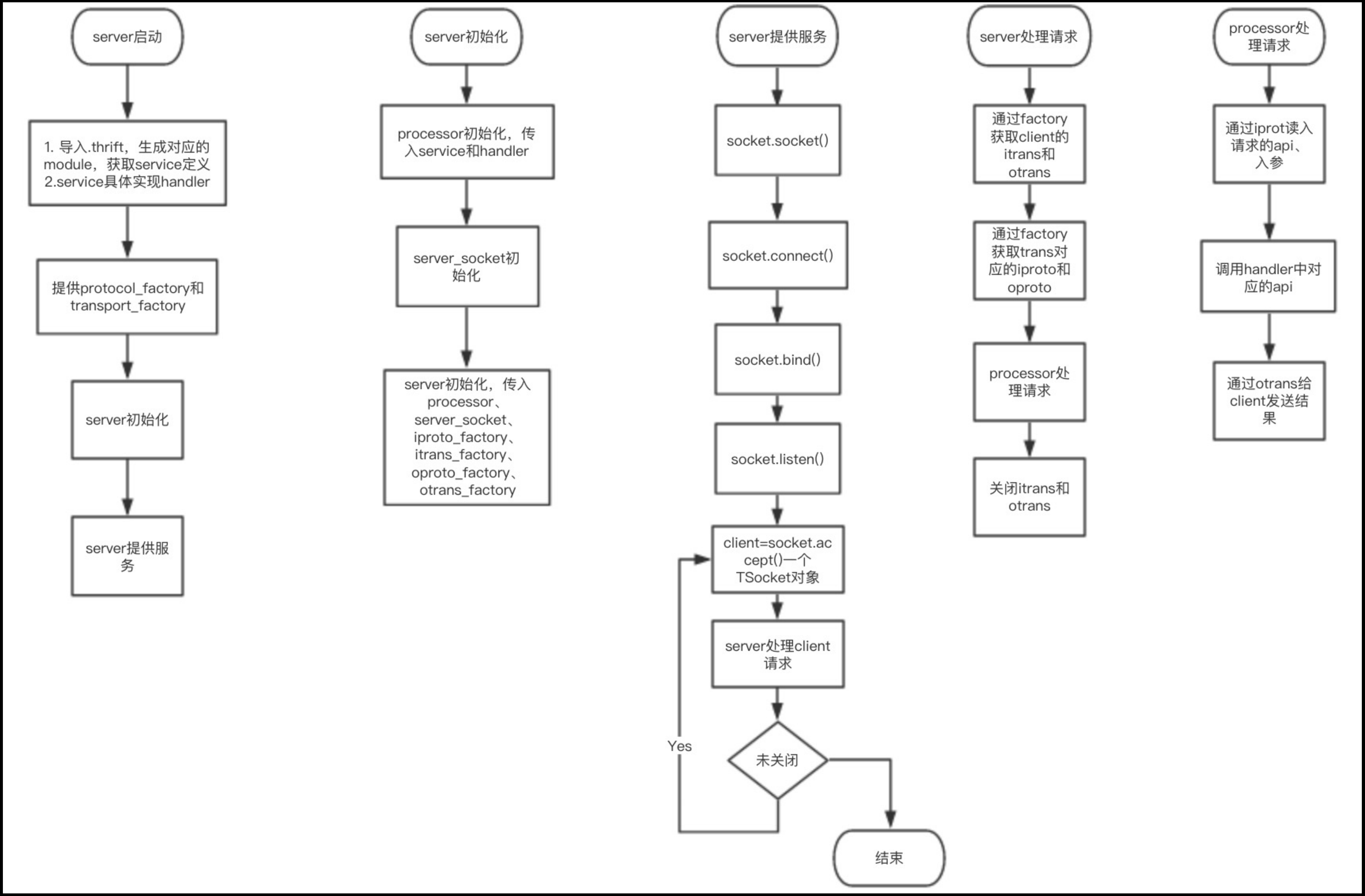
thrift RPC协议框架



thriftpy

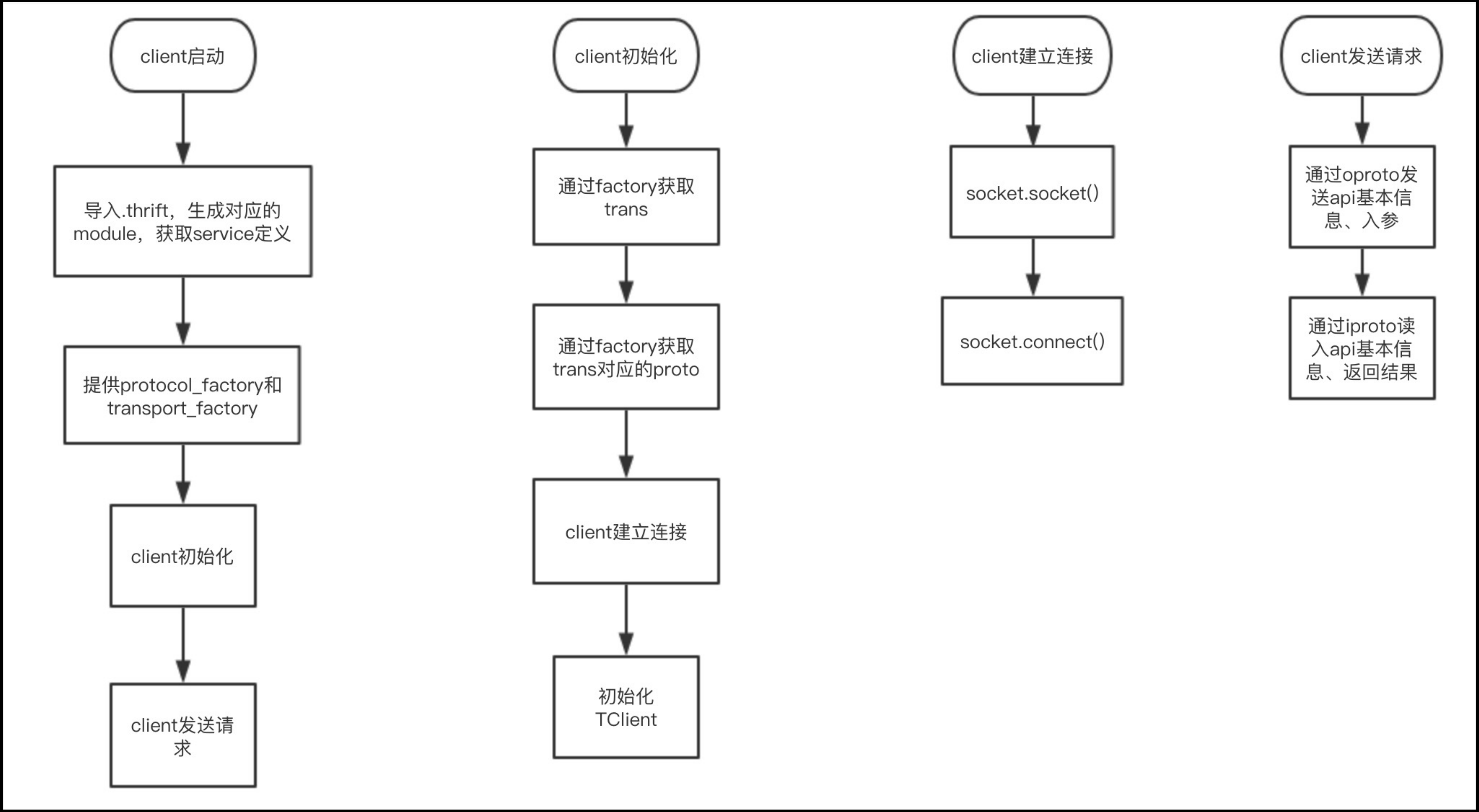
- 0. 微服务的RPC
- 1. thrift作为IDL
- 2. thrift作为RPC协议
- 3. thriftpy的实现

thriftpy socket+binary实现server

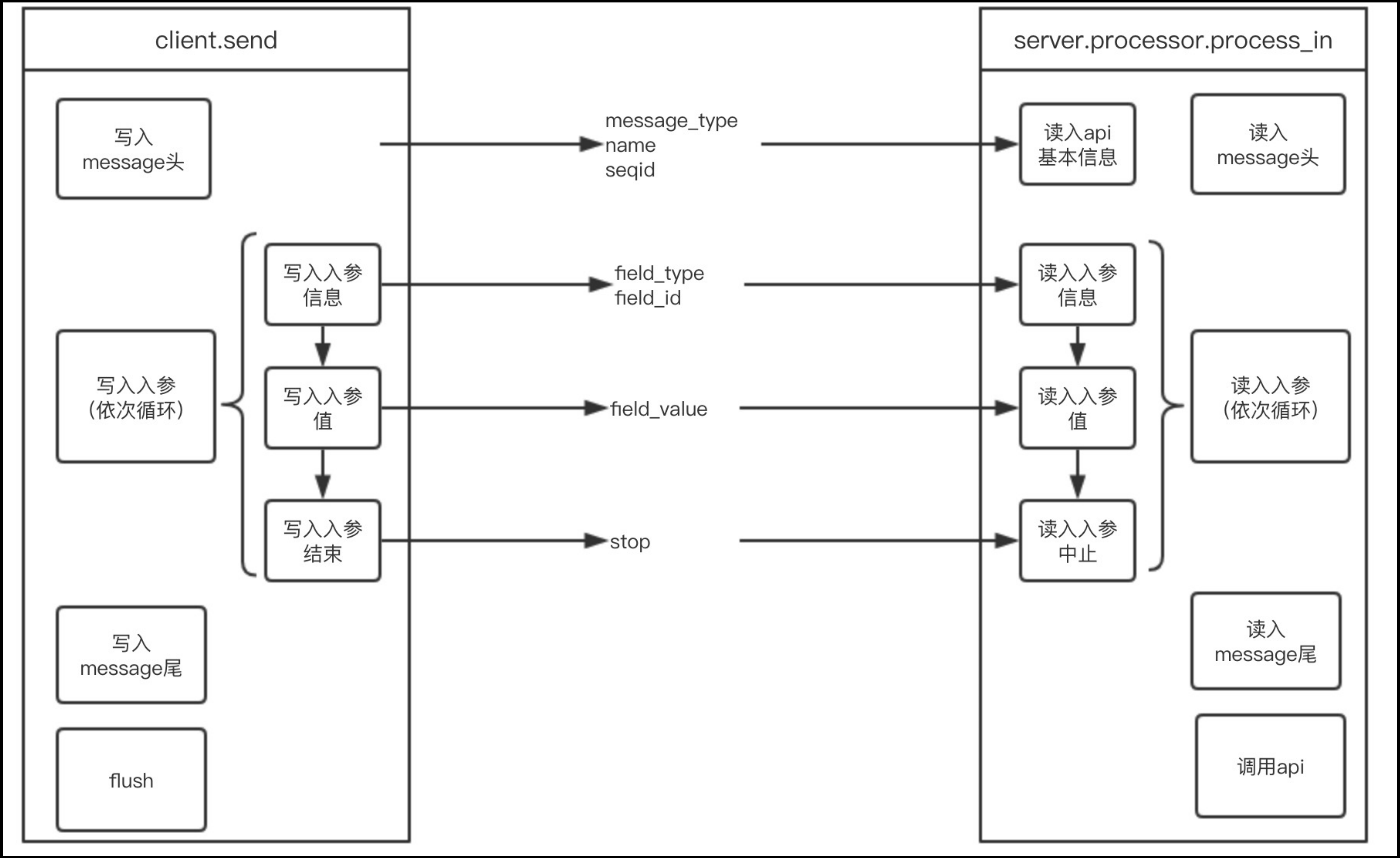


server启动
1. 导入.thrift，生成对应的module，获取service定义
2.service具体实现handler
提供protocol_factory和transport_factory
server初始化
server提供服务
server初始化
processor初始化，传入service和handler
server_socket初始化
server初始化，传入processor、server_socket、iproto_factory、itrans_factory、oproto_factory、otrans_factory
server提供服务
socket.socket()
socket.connect()
socket.bind()
socket.listen()
client=socket.accept()一个TSocket对象
server处理client请求
未关闭
Yes
结束
server处理请求
通过factory获取client的itrans和otrans
通过factory获取trans对应的iproto和oproto
processor处理请求
关闭itrans和otrans
processor处理请求
通过iprot读入请求的api、入参
调用handler中对应的api
通过otrans给client发送结果

thriftpy socket+binary实现client


client启动
导入.thrift，生成对应的module，获取service定义
提供protocol_factory和transport_factory
client初始化
client发送请求
client初始化
通过factory获取trans
通过factory获取trans对应的proto
client建立连接
初始化TClient
client建立连接
socket.socket()
socket.connect()
client发送请求
通过oproto发送api基本信息、入参
通过iproto读入api基本信息、返回结果

thriftpy socket+binary protocol的读写



thriftpy

- 0. 微服务的RPC
- 1. thrift作为IDL
- 2. thrift作为RPC协议
- 3. thriftpy的实现
- 4. 一个socket+binary实现示例

thriftpy generate code

- Person.thrift —> Person.py

thriftpy generate code

- TPayload
  - thrift_spec
  - default_spec
- struct
- Service.api_args
- Service.api_result

thriftpy socket+binary 实现示例

```python
# person_client.py
# -*- coding: utf-8 -*-

import time
import thriftpy2

from thriftpy2.rpc import client_context

person_thrift = thriftpy2.load("person.thrift", module_name="person_thrift")


def main():
    with client_context(person_thrift.PersonService, '127.0.0.1', 6000) as c:
        name = u'张汝家'
        number = '156****7119'
        phone_number = person_thrift.PhoneNumber(person_thrift.PhoneType.MOBILE, number)
        person = person_thrift.Person(name, [phone_number], int(time.time()))
        print c.add(person)
        print c.get(name)
        try:
            print c.get(name[1:])
        except person_thrift.PersonNotExistsError as e:
            print e.message
        print c.remove(name)

if __name__ == '__main__':
    main()
```

thriftpy socket+binary 实现示例

```python
# person_server.py
# -*- coding: utf-8 -*-

import thriftpy2

from thriftpy2.rpc import make_server

person_thrift = thriftpy2.load("person.thrift", module_name="person_thrift")


class Dispatcher(object):
    def __init__(self):
        self.persons = dict()

    def add(self, person):
        self.persons[person.name] = person
        return True

    def remove(self, name):
        if name in self.persons:
            self.persons.pop(name)
            return True
        raise person_thrift.PersonNotExistsError(u"{} not exists".format(name))

    def get(self, name):
        if name in self.persons:
            return self.persons[name]
        raise person_thrift.PersonNotExistsError(u"{} not exists".format(name))


def main():
    server = make_server(person_thrift.PersonService, Dispatcher(),
                         '127.0.0.1', 6000)
    print("serving...")
    server.serve()


if __name__ == '__main__':
    main()
```

thriftpy socket+binary 实现示例

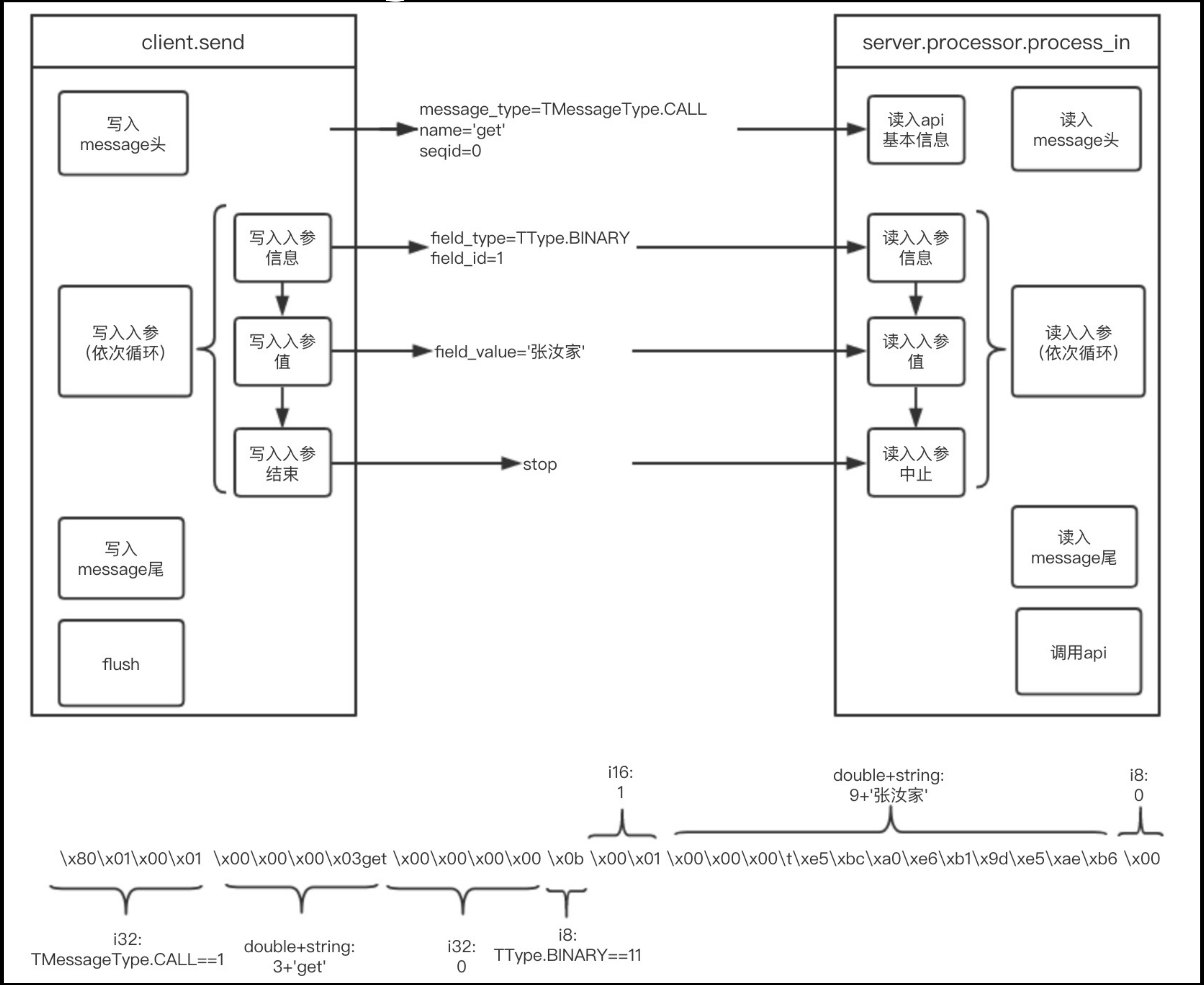
- run person_server.py
- run person_client.py

thriftpy socket+binary 实现示例

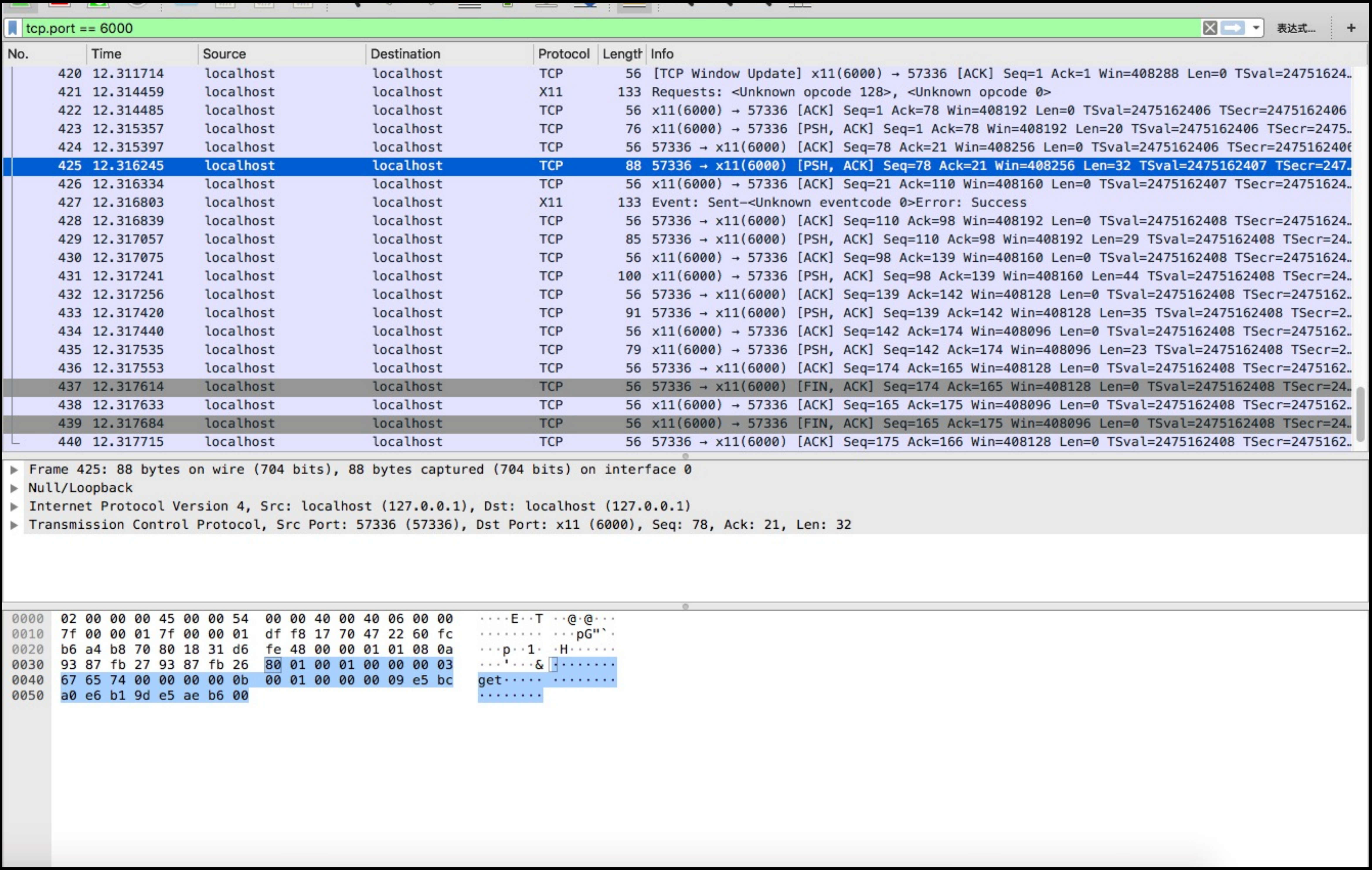
```
def pack_i8(byte):
    return struct.pack("!b", byte)


def pack_string(string):
    return struct.pack("!i%ds" % len(string), len(string), string)

def unpack_i32(buf):
    return struct.unpack("!i", buf)[0]

def unpack_double(buf):
    return struct.unpack("!d", buf)[0]
```

- 最基本的转换
- String的转换
- Structs, Containers, Exceptions, Services的转换

thriftpy socket+binary 实现示例



thriftpy socket+binary 实现示例



tcp.port == 6000

表达式…

```
No.     Time       Source     Destination  Protocol Length Info
    420 12.311714  localhost  localhost    TCP      56  [TCP Window Update] x11(6000) → 57336 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSval=24751624…
    421 12.314459  localhost  localhost    X11      133 Requests: <Unknown opcode 128>, <Unknown opcode 0>
    422 12.314485  localhost  localhost    TCP      56  x11(6000) → 57336 [ACK] Seq=1 Ack=78 Win=408192 Len=0 TSval=2475162406 TSecr=2475162406
    423 12.315357  localhost  localhost    TCP      76  x11(6000) → 57336 [PSH, ACK] Seq=1 Ack=78 Win=408192 Len=20 TSval=2475162406 TSecr=2475…
    424 12.315397  localhost  localhost    TCP      56  57336 → x11(6000) [ACK] Seq=78 Ack=21 Win=408256 Len=0 TSval=2475162406 TSecr=2475162406
    425 12.316245  localhost  localhost    TCP      88  57336 → x11(6000) [PSH, ACK] Seq=78 Ack=21 Win=408256 Len=32 TSval=2475162407 TSecr=247…
    426 12.316334  localhost  localhost    TCP      56  x11(6000) → 57336 [ACK] Seq=21 Ack=110 Win=408160 Len=0 TSval=2475162407 TSecr=24751624…
    427 12.316803  localhost  localhost    X11      133 Event: Sent-<Unknown eventcode 0>Error: Success
    428 12.316839  localhost  localhost    TCP      56  57336 → x11(6000) [ACK] Seq=110 Ack=98 Win=408192 Len=0 TSval=2475162408 TSecr=24751624…
    429 12.317057  localhost  localhost    TCP      85  57336 → x11(6000) [PSH, ACK] Seq=110 Ack=98 Win=408192 Len=29 TSval=2475162408 TSecr=24…
    430 12.317075  localhost  localhost    TCP      56  x11(6000) → 57336 [ACK] Seq=98 Ack=139 Win=408160 Len=0 TSval=2475162408 TSecr=24751624…
    431 12.317241  localhost  localhost    TCP      100 x11(6000) → 57336 [PSH, ACK] Seq=98 Ack=139 Win=408160 Len=44 TSval=2475162408 TSecr=24…
    432 12.317256  localhost  localhost    TCP      56  57336 → x11(6000) [ACK] Seq=139 Ack=142 Win=408128 Len=0 TSval=2475162408 TSecr=2475162…
    433 12.317420  localhost  localhost    TCP      91  57336 → x11(6000) [PSH, ACK] Seq=139 Ack=142 Win=408128 Len=35 TSval=2475162408 TSecr=2…
    434 12.317440  localhost  localhost    TCP      56  x11(6000) → 57336 [ACK] Seq=142 Ack=174 Win=408096 Len=0 TSval=2475162408 TSecr=2475162…
    435 12.317535  localhost  localhost    TCP      79  x11(6000) → 57336 [PSH, ACK] Seq=142 Ack=174 Win=408096 Len=23 TSval=2475162408 TSecr=2…
    436 12.317553  localhost  localhost    TCP      56  57336 → x11(6000) [ACK] Seq=174 Ack=165 Win=408128 Len=0 TSval=2475162408 TSecr=2475162…
    437 12.317614  localhost  localhost    TCP      56  57336 → x11(6000) [FIN, ACK] Seq=174 Ack=165 Win=408128 Len=0 TSval=2475162408 TSecr=24…
    438 12.317633  localhost  localhost    TCP      56  x11(6000) → 57336 [ACK] Seq=165 Ack=175 Win=408096 Len=0 TSval=2475162408 TSecr=2475162…
    439 12.317684  localhost  localhost    TCP      56  x11(6000) → 57336 [FIN, ACK] Seq=165 Ack=175 Win=408096 Len=0 TSval=2475162408 TSecr=24…
    440 12.317715  localhost  localhost    TCP      56  57336 → x11(6000) [ACK] Seq=175 Ack=166 Win=408128 Len=0 TSval=2475162408 TSecr=2475162…
```

```
▶ Frame 425: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: localhost (127.0.0.1), Dst: localhost (127.0.0.1)
▶ Transmission Control Protocol, Src Port: 57336 (57336), Dst Port: x11 (6000), Seq: 78, Ack: 21, Len: 32
```

```
0000  02 00 00 00 45 00 00 54  00 00 40 00 40 06 00 00   ····E··T ··@·@···
0010  7f 00 00 01 7f 00 00 01  df f8 17 70 47 22 60 fc   ········ ···pG"`·
0020  b6 a4 b8 70 80 18 31 d6  fe 48 00 00 01 01 08 0a   ···p··1· ·H······
0030  93 87 fb 27 93 87 fb 26  80 01 00 01 00 00 00 03   ···'···& ········
0040  67 65 74 00 00 00 00 0b  00 01 00 00 00 09 e5 bc   get····· ········
0050  a0 e6 b1 9d e5 ae b6 00                           ········
```

thriftpy more

- transport Wrap
  - buffered, framed, memory
- http client & http server
- version
- tracking
  - header