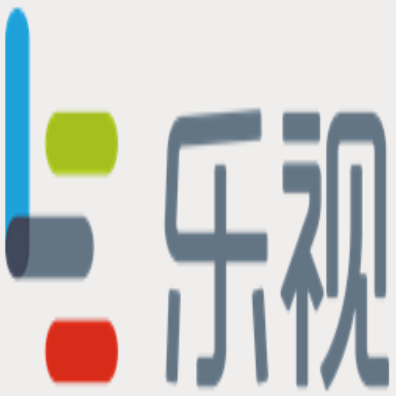
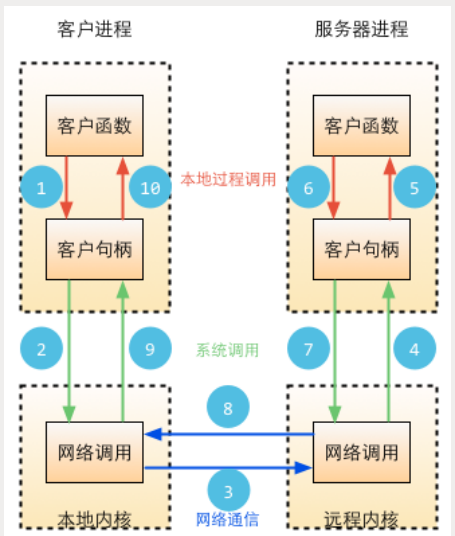


thrift框架讲解

演讲者：白晓琦

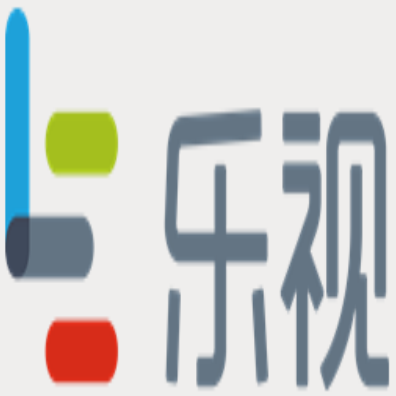
什么是RPC?

- 远程过程调用(Remote Procedure Call)是一个计算机通信协议
- 允许运行于一台计算机的程序调用另一台计算机的子程序，而程序员无需额外地为这个交互作用编程
- 通俗一点就是：调用远程计算机上的程序，就像调用本地程序一样。



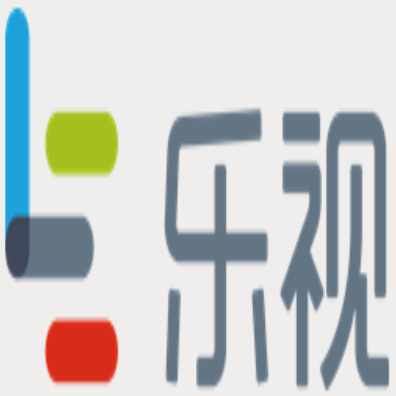
远程过程调用流程图

- 流程：
- 常见的RPC框架: Apache的 **Thrift**、google的gRPC、阿里的Dubbo、Apache的Avro、微信的phxrpc等



thrift定义

- 由Fackbook开发, 贡献给Apache并已经开源
- 支持多种编程语言的RPC(Remote Procedure Call远程过程调用)框架
- 利用自身的协议栈和代码生成工具，根据接口定义语言(IDL Interface definition lanuage)构建可以在C++,Java,Python等程序语言之间进行无缝结合的高效服务
- github地址:
  - https://github.com/apache/thrift
  - https://github.com/facebook/fbthrift



数据类型

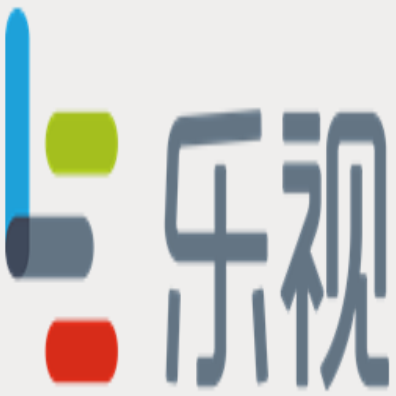
- 基本类型

  bool、byte、i16、i32、i64、double、string、binarry
- 枚举(enum)
- 容器

  list<t> set<t> map<t1, t2>
- 结构体(struct)
- 异常(exception)
- 服务(service)
- 类型定义(Typedefs)

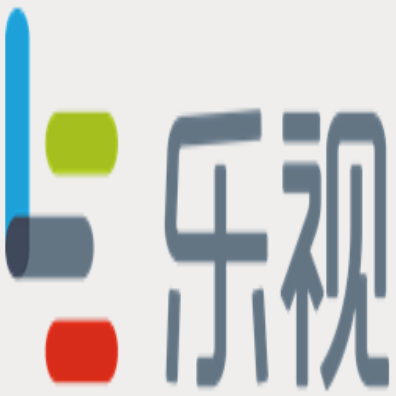
  Thrift supports C/C++ style typedefs.

```
typedef i32 MyInteger   // 1
typedef Tweet ReTweet   // 2
```



字段约束

- required: 必须赋值字段禁止为空
- optional: 可选字段可以为空
- oneway: 对client请求后不返回响应



Sample

pushservice.thrift

```
namespace py pushservice
enum AfterOpen { GOAPP = 1, GOURL = 2, GOACTIVITY = 3, GOCUSTOM = 4}
enum DisplayType {MESSAGE = 1, NOTIFICATION = 2}
struct AndroidMsgBody {
    1: optional string ticker,  ...
    7: optional AfterOpen after_open = AfterOpen.GOAPP,
    8: optional string open_content
}
struct IosMsgAps {
    1: string alert,  ...
}
struct Policy {
    1: optional string start_time,
    2: optional string expire_time,  ...
}
exception ParamError {
    1: string message,
}
exception ServerError {
    1: i16 code,
    2: string message,
}
service PushService {
    void ping(),
    oneway void zip(),  //client发送请求后不返回响应，无返回值
    bool unicast_notification_android(1:string app, 2:DisplayType display_type, 3:str
    bool unicast_notification_ios(1:string app, 2:string device_token, 3:IosMsgAps ap
    ...
}
```

thrift --gen py pushservice.thrift

thrift架构

需要实现的业务逻辑

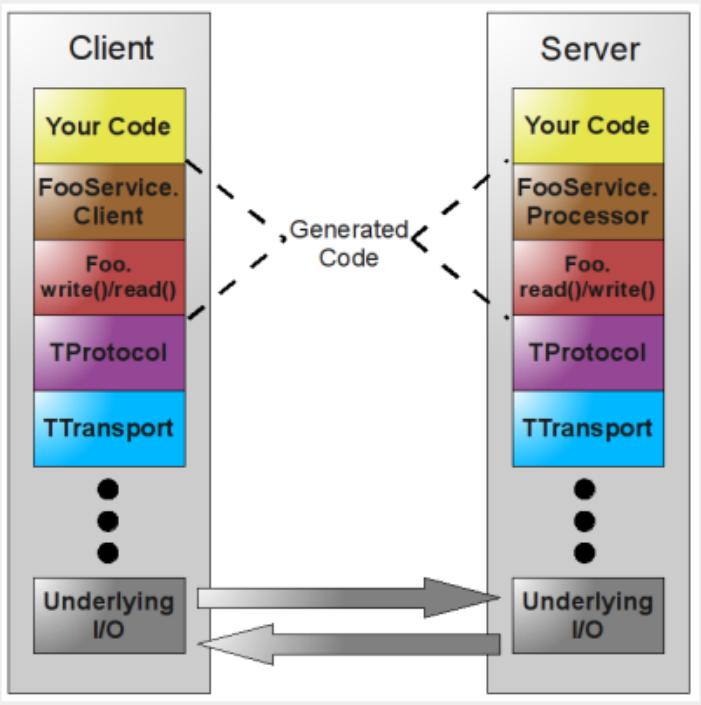
根据 Thrift 的接口定义文件生成的客户端和服务器端代码

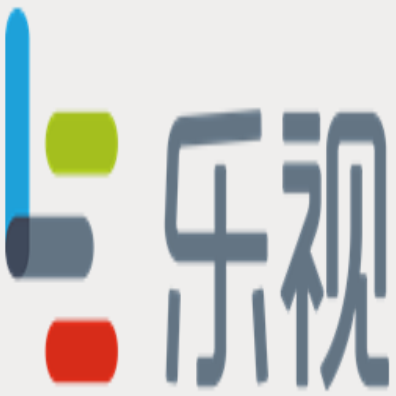
根据 Thrift 文件生成代码实现数据的读写操作

数据的序列化和反序列化

字节流(Byte Stream)数据在IO模块上的传输

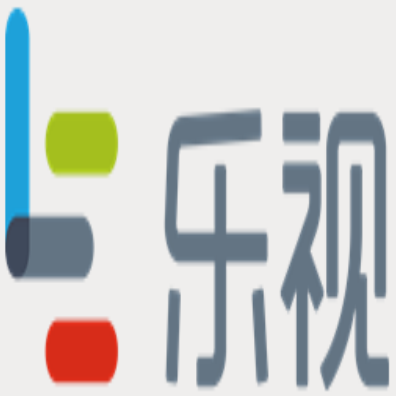
底层 I/O 通信





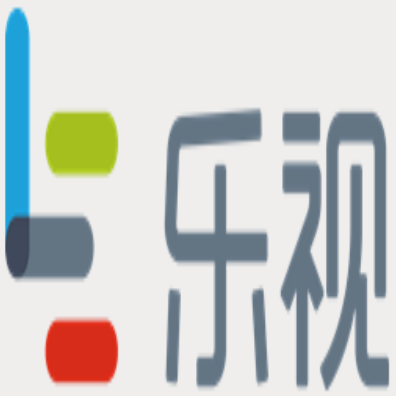
Ttransport 传输层

- 负责Thrift的字节流(Byte Stream)数据在IO模块上的传输
- 主要传输类型:
  - TSocket 使用阻塞的Socket进行IO传输
  - TFramedTransport 使用一个带Buffer的Socket进行IO传输，使用NoblockingServer的时候会需要使用TFramedTransport
  - TFileTransport 使用类文件对象进行IO传输
  - TZlibTransport 使用zlib对数据的压缩传输



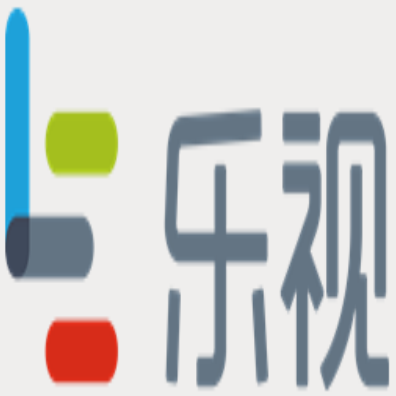
TProtocol 协议

- 客户端与服务端之间传输的通信协议，不同的传输协议对应不同的数据编码
- 根据需求常见的协议有:
  - TBinaryProtocol 二进制编码格式进行数据传输
  - TCompactProtocol 高效率的、密集的二进制编码格式进行数据传输
  - TJSONProtocol 使用 JSON 的数据编码协议进行数据传输
  - TSimpleJSONProtocol 只提供 JSON 只写的协议，适用于通过脚本语言解析



TServer

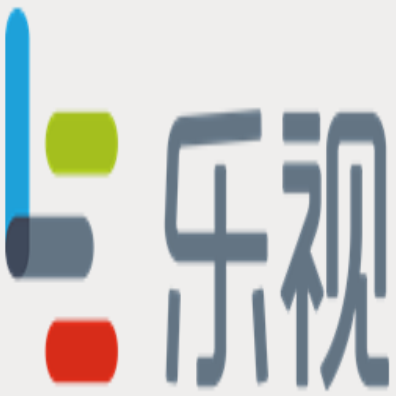
- 负责接收Client的请求，并将请求转发到Processor进行处理
- 常见的服务端类型有以下几种：
  - TSimpleServer 单进程服务器端 使用标准的阻塞式 I/O
  - TForkingServer forks a new process for each request
  - TThreadedServer spawns a new thread per each connection
  - TThreadPoolServer 多线程服务器端 使用标准的阻塞式 I/O
  - TProcessPoolServer 多进程服务器端 使用标准的阻塞式 I/O
  - TNonblockingServer 多线程服务器端 使用非阻塞式 I/O



Server Code Sample

server.py

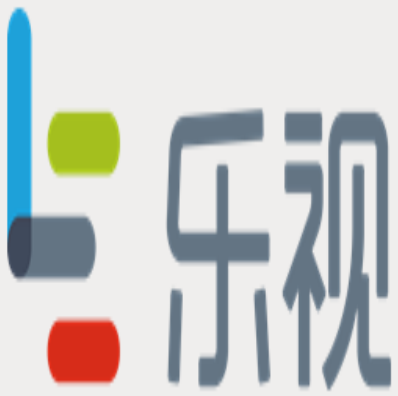
```
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from thrift.server import TServer
from pushservice import PushService
from pushservice.ttypes import ParamError, ServerError

class PushServiceHandler:
    def __init__(self):
        self.log = {}
    def ping(self):
        print 'ping pang pong'
    def unicast_notification_ios(self, app, device_tokens, aps, extra, policy, descri
        print '业务逻辑实现部分'

if __name__ == '__main__':
    handler = PushServiceHandler()
    processor = PushService.Processor(handler)
    ts = TSocket.TServerSocket(host="0.0.0.0" ,port=9090)
    tfactory = TTransport.TBufferedTransportFactory()
    pfactory = TBinaryProtocol.TBinaryProtocolFactory()

    server = TServer.TSimpleServer(processor, ts, tfactory, pfactory)
    server.serve()
```



Client Code Sample

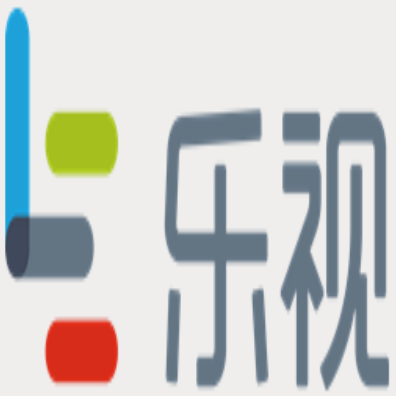
client.py

```
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from pushservice import PushService
from pushservice.ttypes import IosMsgAps, AndroidMsgBody

def main():
    ts = TSocket.TSocket('localhost', 9090)
    transport = TTransport.TBufferedTransport(ts)
    protocol = TBinaryProtocol.TBinaryProtocol(transport)
    client = PushService.Client(protocol)
    transport.open()
    client.ping()
    aps = IosMsgAps("server测试")
    flag = client.unicast_notification_ios('leying', '851807bf98181e22cd1e6f7db7d43ff
    transport.close()

if __name__ == '__main__':
    try:
        main()
    except Thrift.TException as tx:
        print '%s' % tx
```



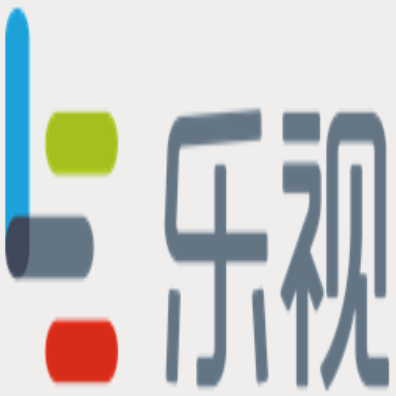
优点和缺点

- 优点
  - 通过thrift文件生成目标代码，简单易用
  - 支持多语言 支持多种序列化方式 支持多种通讯协议 支持多进程和非阻塞式服务端
  - 客户端调用更直观，初步的类型和结构校验
  - 性能比较好
- 缺点：
  - 文档少文档少
  - 只能由客户端发起请求，不支持双向通信
  - 不具备动态特性(可以通过动态定义生成消息或者动态编译)



适用场景

- 对内部提供的Service&API
- 传输的数据比较大，对性能要求比较高的Service&API



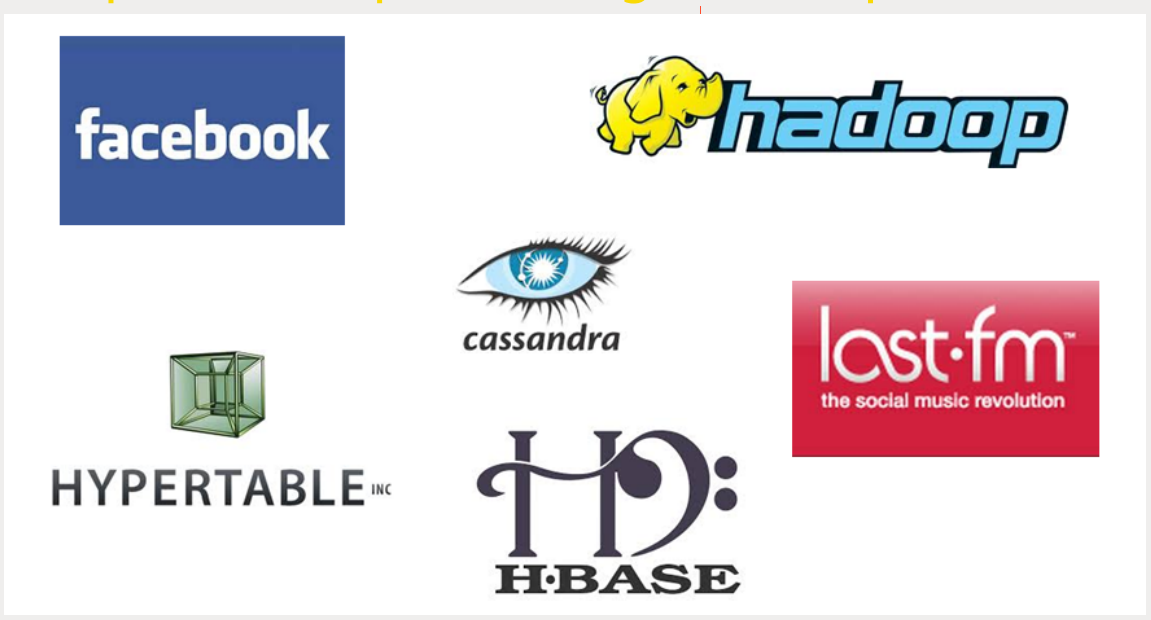
应用

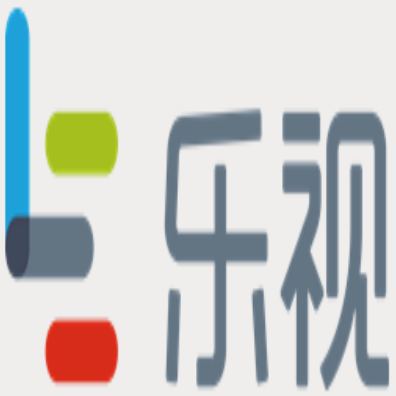
- Facebook的开源的日志收集系统scribe

  https://github.com/facebook/scribe
- Evernote开放接口

  https://github.com/evernote/evernote-thrift
- HBase

  http://wiki.apache.org/hadoop/Hbase/ThriftApi



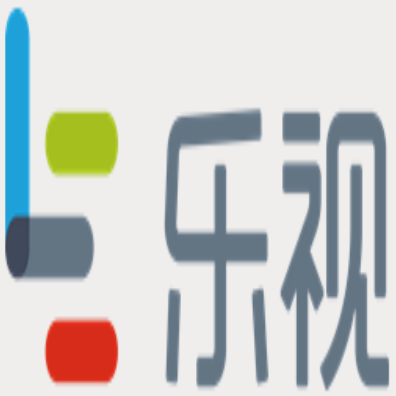


Protobuf(Protocol Bufffers)

- Google公司开发并开源的
- 一种轻便高效的结构化数据存储格式，可以用于结构化数据(序列化)，类似json和XML
- 根据接口定义语言(IDL Interface definition lanuage)构建
- 与语言和平台无关
- 适合做数据存储或 RPC 数据交换格式
- 消息数据序列化后的大小，比json、XML小很多
- github地址： https://github.com/google/protobuf

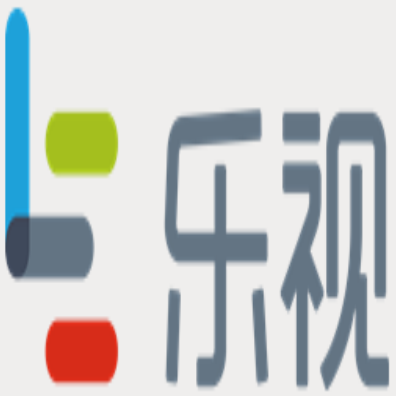
`helloworld.proto`

```
package lm;
message helloworld
{
  required int32     id = 1;  // ID
  required string    str = 2;  // str
  optional int32     opt = 3;  //optional field
}
```



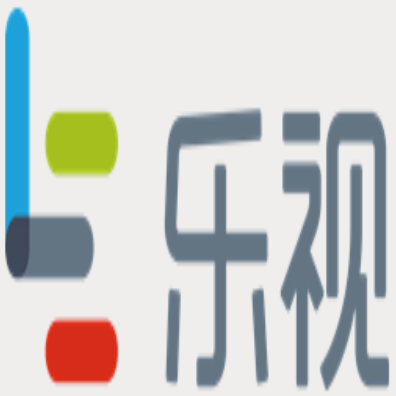
Thrift vs Protobuf

	Thrift	Protobuf
语言支持	Java C++ Python C++ D Dart Go javascript Node.js Cocoa Erlang Haskell OCaml Perl PHP Ruby Smalltalk	C++ Java Python Objective-C C# JavaNano JavaScript Ruby Go PHP(TBD)
基本类型	bool byte integers(16/32/64) double string map list set	bool integers(32/64) float double string bytes repeated
枚举	yes	yes
常量	yes	no
结构化类型	struct	message
异常处理	yes	no
编译语言	C++	C++



Thrift vs Protobuf

	Thrift	Protobuf
RPC实现	yes	no (gRPC)
结构化类型扩展	no	yes
文档	lacking	good
优点	更多的语言支持，丰富的数据结构(map，set)，包括了RPC服务的实现	数据序列化更小，文档丰富
缺点	文档匮乏	没有RPC服务实现



End

Thanks

leeco.com