Apache Thrift

Apache Thrift是Facebook的一个开源项目，主要是一个跨语言的服务开发框架。它有一个代码生成器来对它所定义的IDL定义文件自动生成服务代码框架。用户只要在其之前进行开发就行，对于底层的RPC通信等都是透明的。目前支持的语言有C++、Java、Python、Ruby、Erlang等。

# 1.下载和安装

*#wget* [*http://mirrors.shuosc.org/apache/thrift/0.11.0/thrift-0.11.0.tar.gz*](http://mirrors.shuosc.org/apache/thrift/0.11.0/thrift-0.11.0.tar.gz)

*#tar –xzvf thrift-0.11.0.tar.gz*

//编译和安装

*#./configure*

*#./make*

*#./make install*

验证：

*# thrift -version*

*Thrift version 0.11.0*

# 2.使用

1)定义简单服务hello.thrift

*service HelloService*

*{*

*void hello(1: string name)*

*}*

2)通过IDL工具生成源代码

*#thrift --gen java hello.thrift*

查看生成的代码目录：gen-java

*#ls gen-java*

*HelloService.java*

# 3.Thrift Server及实例

Thrift服务器包含用于绑定协议和传输层的基础架构，它提供阻塞、非阻塞、单线程和多线程的模式运行在服务器上，可以配合服务器/容器一起运行，也可以和现有的J2EE服务器/Web容器无缝结合，下图是Thrift的架构图：



Thrift包含一个完整的堆栈结构用于构建客户端和服务端，如上图所示：

1. 用户实现的业务逻辑，IDL(接口定义语言)
2. 根据Thrift定义的服务接口描述文件生成的客户端和服务端代码框架
3. 根据Thrift文件生成的代码实现数据的读写操作
4. 以上是Thrift的传输体系，协议以及底层I/O通信

## 3.1 用户定义服务（IDL）

根据Thrift语法规范编写脚本文件Hello.thrift，代码如下：

*namespace java com.fys.thrift.demo*

*service Hello{*

*string helloString(1:string para)*

*i32 helloInt(1:i32 para)*

*bool helloBoolean(1:bool para)*

*void helloVoid()*

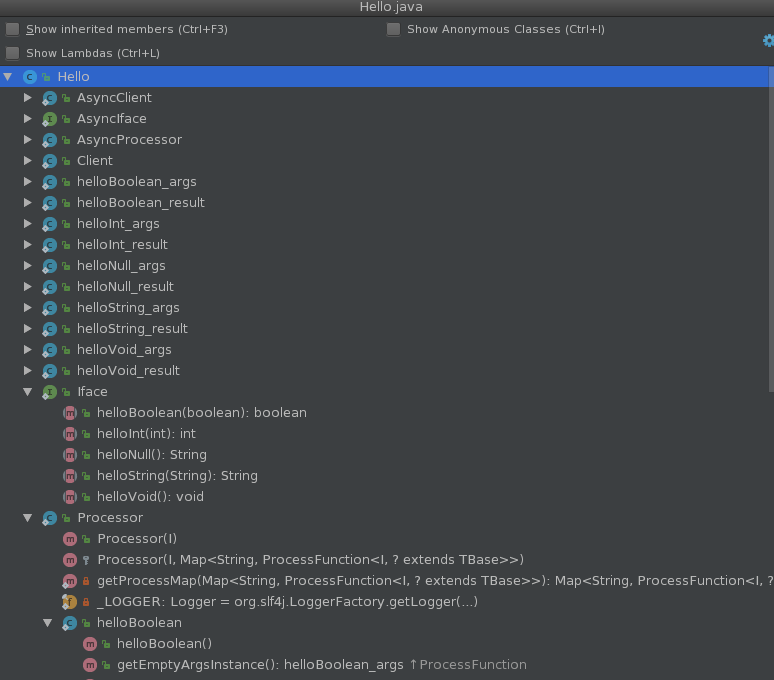
*string helloNull()*

*}*

上面定义了服务的Hello的五个方法，每个方法包含一个方法名、参数列表和返回类型，每个参数包括参数序号、参数类型以及参数名。使用thrift工具编译Hello.thrift，命令如下：

*thrift --gen java Hello.thrift*

生成的代码Hello.java如下：



如上图，该文件包含了在Hello.thrift文件中描述的服务Hello的接口定义，即Hello.Iface接口，以及服务调用的底层通信细节，包括客户端的调用逻辑Hello.Client以及服务器端的处理逻辑Hello.Processor，用于构建客户端和服务器端的功能。

## 3.2 服务器端

1）实现服务器端的服务实现代码

*public class HelloServiceImpl implements Hello.Iface {*

*@Override*

*public String helloString(String para) throws TException {*

*return para;*

*}*

*@Override*

*public int helloInt(int para) throws TException {*

*try {*

*Thread.sleep(20000) ;*

*} catch(InterruptedException e) {*

*e.printStackTrace();*

*}*

*return para;*

*}*

*@Override*

*public boolean helloBoolean(boolean para) throws TException {*

*return para;*

*}*

*@Override*

*public void helloVoid() throws TException {*

*System.out.println("Hello World");*

*}*

*@Override*

*public String helloNull() throws TException {*

*return null;*

*}*

*}*

服务处理逻辑实现Hello.Iface接口，在服务器启动时将HelloServiceImpl作为具体的处理器传递给Thrift服务器，服务器代码如下：

*import org.apache.thrift.TProcessor;*

*import org.apache.thrift.protocol.TBinaryProtocol;*

*import org.apache.thrift.server.TServer;*

*import org.apache.thrift.server.TThreadPoolServer;*

*import org.apache.thrift.transport.TServerSocket;*

*public class HelloServiceServer {*

*public static void main(String[] args) {*

*try {*

*TServerSocket serverTransport = new TServerSocket(7911);*

*TBinaryProtocol.Factory proFactory = new TBinaryProtocol.Factory();*

*TProcessor processor = new Hello.Processor(new HelloServiceImpl());*

*TServer server = new TThreadPoolServer(new TThreadPoolServer.Args(serverTransport).processor(processor));*

*System.out.println("Start server on port 7911 ...");*

*server.serve();*

*} catch(Exception e ){*

*e.printStackTrace();*

*}*

*}*

*}*

## 3.3 客户端

客户端调用Hello.Client访问服务端的逻辑实现，代码如下：

*import org.apache.thrift.protocol.TBinaryProtocol;*

*import org.apache.thrift.protocol.TProtocol;*

*import org.apache.thrift.transport.TSocket;*

*import org.apache.thrift.transport.TTransport;*

*public class HelloServiceClient {*

*public static void main(String[] args) {*

*try {*

*TTransport transport = new TSocket("localhost", 7911);*

*transport.open();*

*TProtocol protocol = new TBinaryProtocol(transport);*

*Hello.Client client = new Hello.Client(protocol);*

*client.helloVoid();*

*transport.close();*

*} catch(Exception e) {*

*e.printStackTrace();*

*}*

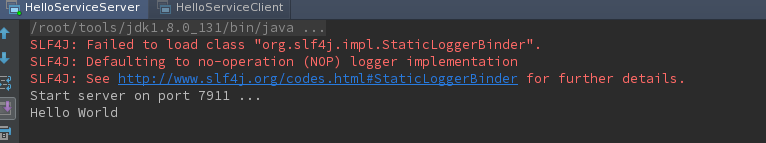
*}*

*}*

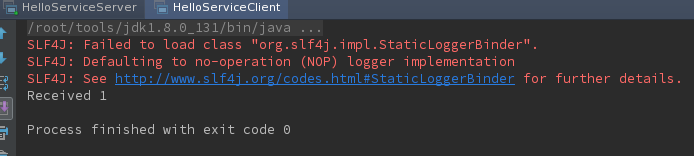
## 3.4 运行

启动服务端后，再启动客户端调用服务Hello的方法helloVoid，在服务器端的控制台输出Hello World，如下所示：

Server端：

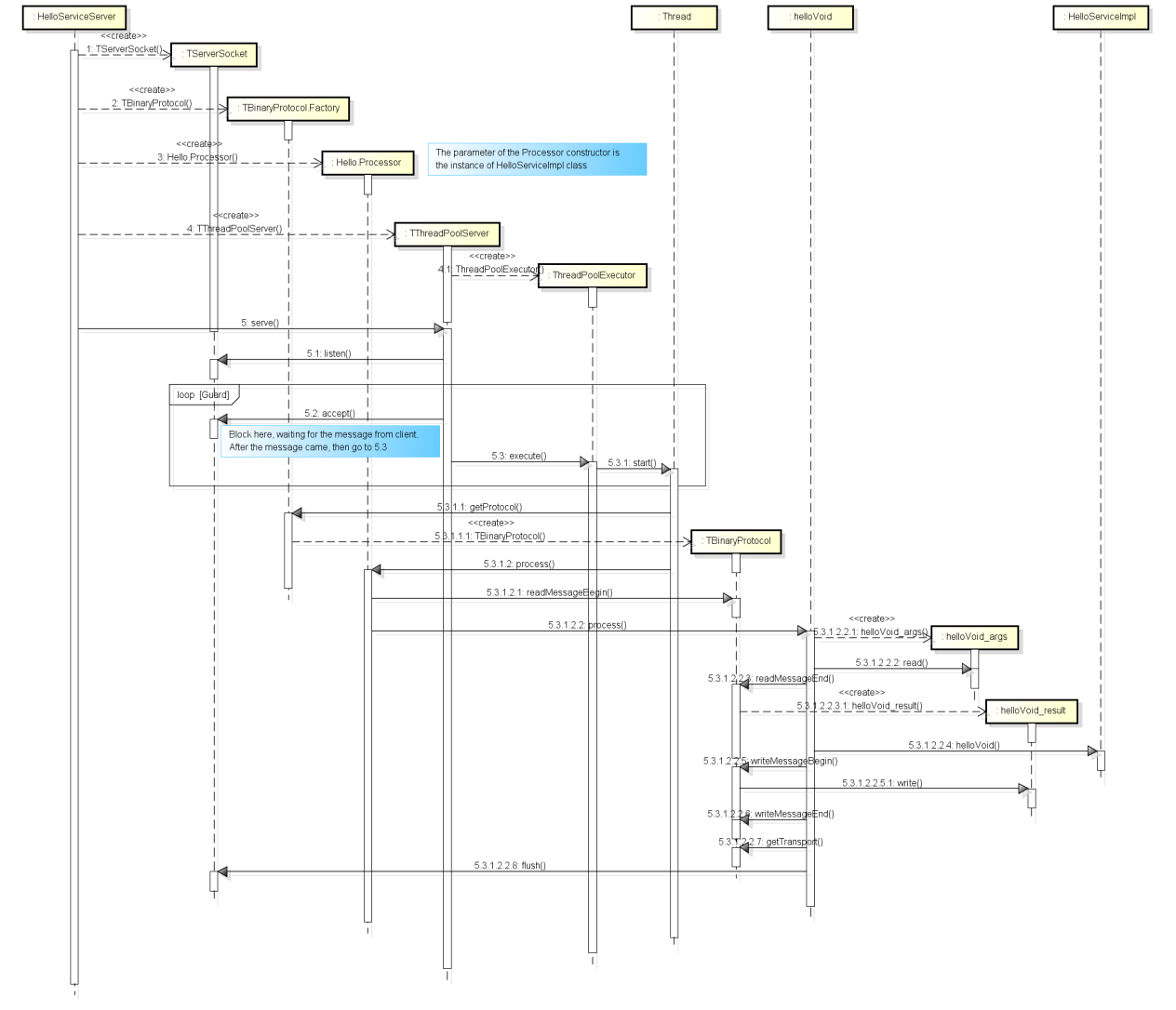


客户端：



## 3.5 Thrift调用序列

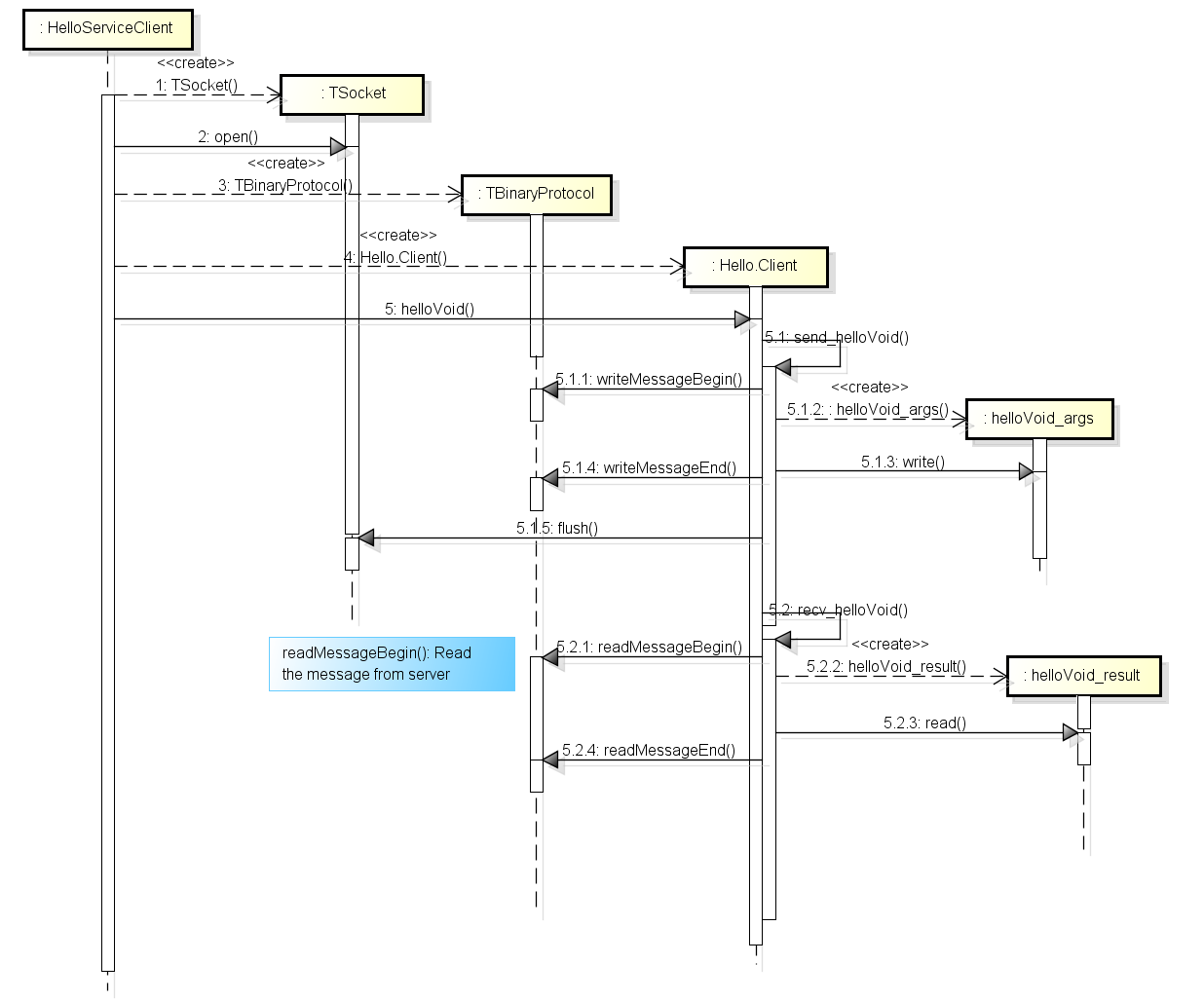
服务端和客户端的调用流程如下：



该图所示是HelloServiceServer启动的过程以及服务被客户端调用时，服务器的响应过程，在图中程序调用TthreadPoolServer的serve方法后，servr进入阻塞监听状态，阻塞在TserverSocket的accept（5.2）方法上。

当接收到来自客户端的消息后，服务器发起一个新线程处理这个消息请求，原线程再次进入阻塞状态。在新线程中，服务器通过TbinaryProtocol协议读取消息内容，调用HelloServiceImpl的helloVoid方法，并将结果写入helloVoid\_result中传回客户端。

客户端的的执行流程如下所示：



HelloServiceClient调用服务的过程以及接收服务器端返回值后处理结果过程。在图中，程序调用Hello.Client的helloVoid方法，在方法中，通过send\_helloVoid方法发送对服务的调用请求，通过recv\_helloVoid方法接收服务处理请求后返回的结果。