**[Apache Thrift学习之一（入门及Java实例演示）](https://www.cnblogs.com/duanxz/p/5516558.html)**

**目录：**

* 概述
* 下载配置
* 基本概念
  1. 数据类型
  2. 服务端编码基本步骤
  3. 客户端编码基本步骤
  4. 数据传输协议
* 实例演示（java）
  1. thrift生成代码
  2. 实现接口Iface
  3. TSimpleServer服务模型
  4. TThreadPoolServer 服务模型
  5. TNonblockingServer 服务模型
  6. THsHaServer服务模型
  7. 异步客户端

**[一]、概述**

Apache Thrift 是 Facebook 实现的一种高效的、支持多种编程语言的远程服务调用的框架。Thrift是由Facebook开发的，并在2008年捐给了Apache基金会，成为了一个孵化器项目。

Thrift是一个软件框架，用来进行可扩展且跨语言的服务的开发。它结合了功能强大的软件堆栈和代码生成引擎，

Thrift是一个驱动层接口，它提供了用于客户端使用多种语言实现的API。  
Thrift是个代码生成库，支持的客户端语言包括C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml 。它的目标是为了各种流行的语言提供便利的RPC调用机制，而不需要使用那些开销巨大的方式，比如SOAP。

要使用Thrift，就要使用一个语言中立的服务定义文件，描述数据类型和服务接口。这个文件会被用作引擎的输入，编译器生成代码为每种支持的语言生成RPC客户端代码库。这种静态生成的设计让它非常容易被开发者所使用，而且因为类型验证都发生在编译期而非运行期，所以代码可以很有效率地运行。

Thrift的设计提供了以下这些特性：  
1、语言无关的类型  
因为类型是使用定义文件按照语言中立的方式规定的，所以它们可以被不同的语言分析。比如，C++的结构可以和Python的字典类型相互交换数据。  
2、通用传输接口  
不论你使用的是磁盘文件、内存数据还是socket流，都可以使用同一段应用代码。  
3、协议无关  
Thrift会对数据类型进行编码和解码，可以跨协议使用。  
4、支持版本  
数据类型可以加入版本信息，来支持客户端API的更新。

官网地址：[thrift.apache.org](http://thrift.apache.org/" \t "_blank)

推荐值得一看的文章：

* <http://jnb.ociweb.com/jnb/jnbJun2009.html>
* <http://wiki.apache.org/thrift>
* http://thrift.apache.org/static/files/thrift-20070401.pdf

**[二]、下载配置**

1）安装thrift：到thrift官网下载exe文件，然后将文件重命名为thrift.exe,拷贝到D:\EBOOK\thrift目录下(或者任何目录下)，然后就可以在dos环境下使用了

　　D:\EBOOK\thrift>thrift -gen java D:\work\workspace\thriftworkspace\demo1\demoHello.thrift ，输出的java文件默认输出到当前目录下D:\EBOOK\thrift\gen-java，也可以使用-o参数指定输出路径;

  2）下载相关依赖包

  2.1）libthrift.jar ，下载地址：http://repo1.maven.org/maven2/org/apache/thrift/libthrift/0.9.0/

  2.2）slf4j-api.jar

  2.3）slf4j-simple.jar

到官网http://thrift.apache.org/download 下载最新版本，截止今日（2016-05-23）最新版本为0.9.3

1. 如果是Maven构建项目的，直接在pom.xml 中添加如下内容：

<dependency>

<groupId>org.apache.thrift</groupId>

<artifactId>libthrift</artifactId>

<version>0.8.0</version>

</dependency>

2.如果自己编译lib包，把下载的压缩包解压到X:盘，然后在X:\thrift-0.8.0\lib\java 目录下运行ant进行自动编译，会在X:\thrift-0.8.0\lib\java\build\ 目录下看到编译好的lib包：libthrift-0.8.0.jar

**[三]、基本概念**

1.数据类型

* 基本类型：
  + bool：布尔值，true 或 false，对应 Java 的 boolean
  + byte：8 位有符号整数，对应 Java 的 byte
  + i16：16 位有符号整数，对应 Java 的 short
  + i32：32 位有符号整数，对应 Java 的 int
  + i64：64 位有符号整数，对应 Java 的 long
  + double：64 位浮点数，对应 Java 的 double
  + string：utf-8编码的字符串，对应 Java 的 String
* 结构体类型：
  + struct：定义公共的对象，类似于 C 语言中的结构体定义，在 Java 中是一个 JavaBean
* 容器类型：
  + list：对应 Java 的 ArrayList
  + set：对应 Java 的 HashSet
  + map：对应 Java 的 HashMap
* 异常类型：
  + exception：对应 Java 的 Exception
* 服务类型：
  + service：对应服务的类

2.服务端编码基本步骤：

* 实现服务处理接口impl
* 创建TProcessor
* 创建TServerTransport
* 创建TProtocol
* 创建TServer
* 启动Server

3.客户端编码基本步骤：

* 创建Transport
* 创建TProtocol
* 基于TTransport和TProtocol创建 Client
* 调用Client的相应方法

4.数据传输协议

* TBinaryProtocol : 二进制格式.
* TCompactProtocol : 压缩格式
* TJSONProtocol : JSON格式
* TSimpleJSONProtocol : 提供JSON只写协议, 生成的文件很容易通过脚本语言解析

tips:客户端和服务端的协议要一致

**[四]、实例演示**

1. thrift生成代码

创建Thrift文件：D:\work\workspace\thriftworkspace\demo1\demoHello.thrift ,内容如下：

namespace java com.dxz.thrift.demo

service HelloWorldService {

string sayHello(1:string username)

}

thrift-0.8.0.exe 是官网提供的windows下编译工具，运用这个工具生成相关代码：

D:\EBOOK\thrift>thrift-0.9.3.exe -r -gen java D:\work\workspace\thriftworkspace\demo1\demoHello.thrift

生成后的目录结构如下：

将生成的HelloWorldService.java 文件copy到自己测试的工程中，我的工程是用maven构建的，故在pom.xml中增加如下内容：

<dependency>

<groupId>org.apache.thrift</groupId>

<artifactId>libthrift</artifactId>

<version>0.8.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.5.8</version>

</dependency>

如果是ant构建的工程，将libthrift-0.9.3.jar加入到工程中

2. 实现接口Iface

java代码：HelloWorldImpl.java

package com.dxz.thrift.demo;

import org.apache.thrift.TException;

public class HelloWorldImpl implements HelloWorldService.Iface {

public HelloWorldImpl() {

}

@Override

public String sayHello(String username) throws TException {

return "Hi," + username + " welcome to thrift world";

}

}

3.TSimpleServer服务端

简单的单线程服务模型，一般用于测试。

编写服务端server代码：HelloServerDemo.java

package com.dxz.thrift.demo;

import org.apache.thrift.TProcessor;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TJSONProtocol;

import org.apache.thrift.protocol.TSimpleJSONProtocol;

import org.apache.thrift.server.TServer;

import org.apache.thrift.server.TSimpleServer;

import org.apache.thrift.transport.TServerSocket;

public class HelloServerDemo {

public static final int SERVER\_PORT = 8090;

public void startServer() {

try {

System.out.println("HelloWorld TSimpleServer start ....");

TProcessor tprocessor = new HelloWorldService.Processor<HelloWorldService.Iface>(new HelloWorldImpl());

// HelloWorldService.Processor&lt;HelloWorldService.Iface&gt;

// tprocessor =

// new HelloWorldService.Processor&lt;HelloWorldService.Iface&gt;(

// new HelloWorldImpl());

// 简单的单线程服务模型，一般用于测试

TServerSocket serverTransport = new TServerSocket(SERVER\_PORT);

TServer.Args tArgs = new TServer.Args(serverTransport);

tArgs.processor(tprocessor);

tArgs.protocolFactory(new TBinaryProtocol.Factory());

// tArgs.protocolFactory(new TCompactProtocol.Factory());

// tArgs.protocolFactory(new TJSONProtocol.Factory());

TServer server = new TSimpleServer(tArgs);

server.serve();

} catch (Exception e) {

System.out.println("Server start error!!!");

e.printStackTrace();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloServerDemo server = new HelloServerDemo();

server.startServer();

}

}

编写客户端Client代码：HelloClientDemo.java

package com.dxz.thrift.demo;

import org.apache.thrift.TException;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TJSONProtocol;

import org.apache.thrift.protocol.TProtocol;

import org.apache.thrift.transport.TSocket;

import org.apache.thrift.transport.TTransport;

import org.apache.thrift.transport.TTransportException;

public class HelloClientDemo {

public static final String SERVER\_IP = "localhost";

public static final int SERVER\_PORT = 8090;

public static final int TIMEOUT = 30000;

/\*\*

\*

\* @param userName

\*/

public void startClient(String userName) {

TTransport transport = null;

try {

transport = new TSocket(SERVER\_IP, SERVER\_PORT, TIMEOUT);

// 协议要和服务端一致

TProtocol protocol = new TBinaryProtocol(transport);

// TProtocol protocol = new TCompactProtocol(transport);

// TProtocol protocol = new TJSONProtocol(transport);

HelloWorldService.Client client = new HelloWorldService.Client(protocol);

transport.open();

String result = client.sayHello(userName);

System.out.println("Thrify client result =: " + result);

} catch (TTransportException e) {

e.printStackTrace();

} catch (TException e) {

e.printStackTrace();

} finally {

if (null != transport) {

transport.close();

}

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloClientDemo client = new HelloClientDemo();

client.startClient("china");

}

}

先运行服务端程序，日志如下：

HelloWorld TSimpleServer start ....

再运行客户端调用程序，日志如下：

Thrify client result =: Hi,china welcome to thrift world.

测试成功，和预期的返回信息一致。

4.TThreadPoolServer 服务模型

线程池服务模型，使用标准的阻塞式IO，预先创建一组线程处理请求。

编写服务端代码：HelloServerDemo2.java

package com.dxz.thrift.demo;

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 HelloServerDemo2 {

public static final int SERVER\_PORT = 8090;

public void startServer() {

try {

System.out.println("HelloWorld TThreadPoolServer start ....");

TProcessor tprocessor = new HelloWorldService.Processor<HelloWorldService.Iface>(new HelloWorldImpl());

TServerSocket serverTransport = new TServerSocket(SERVER\_PORT);

TThreadPoolServer.Args ttpsArgs = new TThreadPoolServer.Args(serverTransport);

ttpsArgs.processor(tprocessor);

ttpsArgs.protocolFactory(new TBinaryProtocol.Factory());

// 线程池服务模型，使用标准的阻塞式IO，预先创建一组线程处理请求。

TServer server = new TThreadPoolServer(ttpsArgs);

server.serve();

} catch (Exception e) {

System.out.println("Server start error!!!");

e.printStackTrace();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloServerDemo2 server = new HelloServerDemo2();

server.startServer();

}

}

客户端Client代码和之前的一样，只要数据传输的协议一致即可，客户端测试成功，结果如下：

Thrify client result =: Hi,china welcome to thrift world.

5.TNonblockingServer 服务模型

使用非阻塞式IO，服务端和客户端需要指定 TFramedTransport 数据传输的方式。

编写服务端代码：HelloServerDemo3.java

package com.dxz.thrift.demo;

import org.apache.thrift.TProcessor;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.server.TNonblockingServer;

import org.apache.thrift.server.TServer;

import org.apache.thrift.server.TThreadPoolServer;

import org.apache.thrift.transport.TFramedTransport;

import org.apache.thrift.transport.TNonblockingServerSocket;

import org.apache.thrift.transport.TServerSocket;

public class HelloServerDemo3 {

public static final int SERVER\_PORT = 8090;

public void startServer() {

try {

System.out.println("HelloWorld TNonblockingServer start ....");

TProcessor tprocessor = new HelloWorldService.Processor<HelloWorldService.Iface>(new HelloWorldImpl());

TNonblockingServerSocket tnbSocketTransport = new TNonblockingServerSocket(SERVER\_PORT);

TNonblockingServer.Args tnbArgs = new TNonblockingServer.Args(tnbSocketTransport);

tnbArgs.processor(tprocessor);

tnbArgs.transportFactory(new TFramedTransport.Factory());

tnbArgs.protocolFactory(new TCompactProtocol.Factory());

// 使用非阻塞式IO，服务端和客户端需要指定TFramedTransport数据传输的方式

TServer server = new TNonblockingServer(tnbArgs);

server.serve();

} catch (Exception e) {

System.out.println("Server start error!!!");

e.printStackTrace();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloServerDemo3 server = new HelloServerDemo3();

server.startServer();

}

}

编写客户端代码：HelloClientDemo3.java

package com.dxz.thrift.demo;

import org.apache.thrift.TException;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TJSONProtocol;

import org.apache.thrift.protocol.TProtocol;

import org.apache.thrift.transport.TSocket;

import org.apache.thrift.transport.TTransport;

import org.apache.thrift.transport.TTransportException;

import org.apache.thrift.TException;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TProtocol;

import org.apache.thrift.transport.TFramedTransport;

import org.apache.thrift.transport.TSocket;

import org.apache.thrift.transport.TTransport;

import org.apache.thrift.transport.TTransportException;

public class HelloClientDemo3 {

public static final String SERVER\_IP = "localhost";

public static final int SERVER\_PORT = 8090;

public static final int TIMEOUT = 30000;

public void startClient(String userName) {

TTransport transport = null;

try {

transport = new TFramedTransport(new TSocket(SERVER\_IP, SERVER\_PORT, TIMEOUT));

// 协议要和服务端一致

TProtocol protocol = new TCompactProtocol(transport);

HelloWorldService.Client client = new HelloWorldService.Client(protocol);

transport.open();

String result = client.sayHello(userName);

System.out.println("Thrify client result =: " + result);

} catch (TTransportException e) {

e.printStackTrace();

} catch (TException e) {

e.printStackTrace();

} finally {

if (null != transport) {

transport.close();

}

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloClientDemo3 client = new HelloClientDemo3();

client.startClient("HelloClientDemo3");

}

}

客户端的测试成功，结果如下：

Thrify client result =: Hi,HelloClientDemo3 welcome to thrift world.

6.THsHaServer服务模型

半同步半异步的服务端模型，需要指定为： TFramedTransport 数据传输的方式。

编写服务端代码：HelloServerDemo4.java

package com.dxz.thrift.demo;

import org.apache.thrift.TProcessor;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.server.THsHaServer;

import org.apache.thrift.server.TNonblockingServer;

import org.apache.thrift.server.TServer;

import org.apache.thrift.server.TSimpleServer;

import org.apache.thrift.server.TThreadPoolServer;

import org.apache.thrift.transport.TFramedTransport;

import org.apache.thrift.transport.TNonblockingServerSocket;

import org.apache.thrift.transport.TServerSocket;

public class HelloServerDemo4 {

public static final int SERVER\_PORT = 8090;

public void startServer() {

try {

System.out.println("HelloWorld THsHaServer start ....");

TProcessor tprocessor = new HelloWorldService.Processor<HelloWorldService.Iface>(new HelloWorldImpl());

TNonblockingServerSocket tnbSocketTransport = new TNonblockingServerSocket(SERVER\_PORT);

THsHaServer.Args thhsArgs = new THsHaServer.Args(tnbSocketTransport);

thhsArgs.processor(tprocessor);

thhsArgs.transportFactory(new TFramedTransport.Factory());

thhsArgs.protocolFactory(new TBinaryProtocol.Factory());

// 半同步半异步的服务模型

TServer server = new THsHaServer(thhsArgs);

server.serve();

} catch (Exception e) {

System.out.println("Server start error!!!");

e.printStackTrace();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloServerDemo4 server = new HelloServerDemo4();

server.startServer();

}

}

[复制代码](javascript:void(0);)

客户端代码HelloClientDemo4.java

[复制代码](javascript:void(0);)

package com.dxz.thrift.demo;

import java.io.IOException;

import org.apache.thrift.TException;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TProtocol;

import org.apache.thrift.transport.TSocket;

import org.apache.thrift.transport.TTransport;

import org.apache.thrift.transport.TTransportException;

public class HelloClientDemo4 {

public static final String SERVER\_IP = "localhost";

public static final int SERVER\_PORT = 8090;

public static final int TIMEOUT = 30000;

public void startClient(String userName) {

TTransport transport = null;

try {

transport = new TFramedTransport(new TSocket(SERVER\_IP, SERVER\_PORT, TIMEOUT));

// 协议要和服务端一致

TProtocol protocol = new TBinaryProtocol(transport);

// TProtocol protocol = new TCompactProtocol(transport);

// TProtocol protocol = new TJSONProtocol(transport);

HelloWorldService.Client client = new HelloWorldService.Client(protocol);

transport.open();

String result = client.sayHello(userName);

System.out.println("Thrify client result =: " + result);

} catch (TTransportException e) {

e.printStackTrace();

} catch (TException e) {

e.printStackTrace();

} finally {

if (null != transport) {

transport.close();

}

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloClientDemo4 client = new HelloClientDemo4();

client.startClient("HelloClientDemo4");

}

}

[复制代码](javascript:void(0);)

结果：Thrify client result =: Hi,HelloClientDemo4 welcome to thrift world.

7.异步客户端

编写服务端代码：HelloServerDemo5.java

[复制代码](javascript:void(0);)

package com.dxz.thrift.demo;

import org.apache.thrift.TProcessor;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.server.TNonblockingServer;

import org.apache.thrift.server.TServer;

import org.apache.thrift.transport.TFramedTransport;

import org.apache.thrift.transport.TNonblockingServerSocket;

public class HelloServerDemo5 {

public static final int SERVER\_PORT = 8090;

public void startServer() {

try {

System.out.println("HelloWorld TNonblockingServer start ....");

TProcessor tprocessor = new HelloWorldService.Processor<HelloWorldService.Iface>(new HelloWorldImpl());

TNonblockingServerSocket tnbSocketTransport = new TNonblockingServerSocket(SERVER\_PORT);

TNonblockingServer.Args tnbArgs = new TNonblockingServer.Args(tnbSocketTransport);

tnbArgs.processor(tprocessor);

tnbArgs.transportFactory(new TFramedTransport.Factory());

tnbArgs.protocolFactory(new TCompactProtocol.Factory());

// 使用非阻塞式IO，服务端和客户端需要指定TFramedTransport数据传输的方式

TServer server = new TNonblockingServer(tnbArgs);

server.serve();

} catch (Exception e) {

System.out.println("Server start error!!!");

e.printStackTrace();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloServerDemo5 server = new HelloServerDemo5();

server.startServer();

}

}

[复制代码](javascript:void(0);)

编写客户端Client代码：HelloAsynClientDemo.java

[复制代码](javascript:void(0);)

package com.dxz.thrift.demo;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.TimeUnit;

import org.apache.thrift.TException;

import org.apache.thrift.async.AsyncMethodCallback;

import org.apache.thrift.async.TAsyncClientManager;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TProtocolFactory;

import org.apache.thrift.transport.TNonblockingSocket;

import org.apache.thrift.transport.TNonblockingTransport;

import com.dxz.thrift.demo.HelloWorldService.AsyncClient.sayHello\_call;

public class HelloAsynClientDemo {

public static final String SERVER\_IP = "localhost";

public static final int SERVER\_PORT = 8090;

public static final int TIMEOUT = 30000;

public void startClient(String userName) {

try {

TAsyncClientManager clientManager = new TAsyncClientManager();

TNonblockingTransport transport = new TNonblockingSocket(SERVER\_IP, SERVER\_PORT, TIMEOUT);

TProtocolFactory tprotocol = new TCompactProtocol.Factory();

HelloWorldService.AsyncClient asyncClient = new HelloWorldService.AsyncClient(tprotocol, clientManager,

transport);

System.out.println("Client start .....");

CountDownLatch latch = new CountDownLatch(1);

AsynCallback callBack = new AsynCallback(latch);

System.out.println("call method sayHello start ...");

asyncClient.sayHello(userName, callBack);

System.out.println("call method sayHello .... end");

boolean wait = latch.await(30, TimeUnit.SECONDS);

System.out.println("latch.await =:" + wait);

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("startClient end.");

}

public class AsynCallback implements AsyncMethodCallback<sayHello\_call> {

private CountDownLatch latch;

public AsynCallback(CountDownLatch latch) {

this.latch = latch;

}

@Override

public void onComplete(sayHello\_call response) {

System.out.println("onComplete");

try {

// Thread.sleep(1000L \* 1);

System.out.println("AsynCall result =:" + response.getResult().toString());

} catch (TException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

} finally {

latch.countDown();

}

}

@Override

public void onError(Exception exception) {

System.out.println("onError :" + exception.getMessage());

latch.countDown();

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

HelloAsynClientDemo client = new HelloAsynClientDemo();

client.startClient("HelloAsynClientDemo");

}

}

[复制代码](javascript:void(0);)

先运行服务程序，再运行客户端程序，测试结果如下：

[复制代码](javascript:void(0);)

Client start .....

call method sayHello start ...

call method sayHello .... end

onComplete

AsynCall result =:Hi,HelloAsynClientDemo welcome to thrift world.

latch.await =:true

startClient end.

[复制代码](javascript:void(0);)