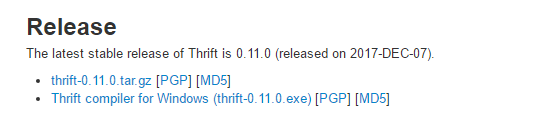
本文的例子是定义好thrift文件，生成java和python的服务端与客户端，实现两种语言的简单相互调用。

# 一、下载代码生产工具

访问官网： <http://thrift.apache.org/download>

下载对于的版本，都是开箱即用的，没有其他的依赖。



Linux的需要编译安装到thrift的解压目录，执行命令：

#chmod +x \* //设置执行权限

#./bootstrap.sh

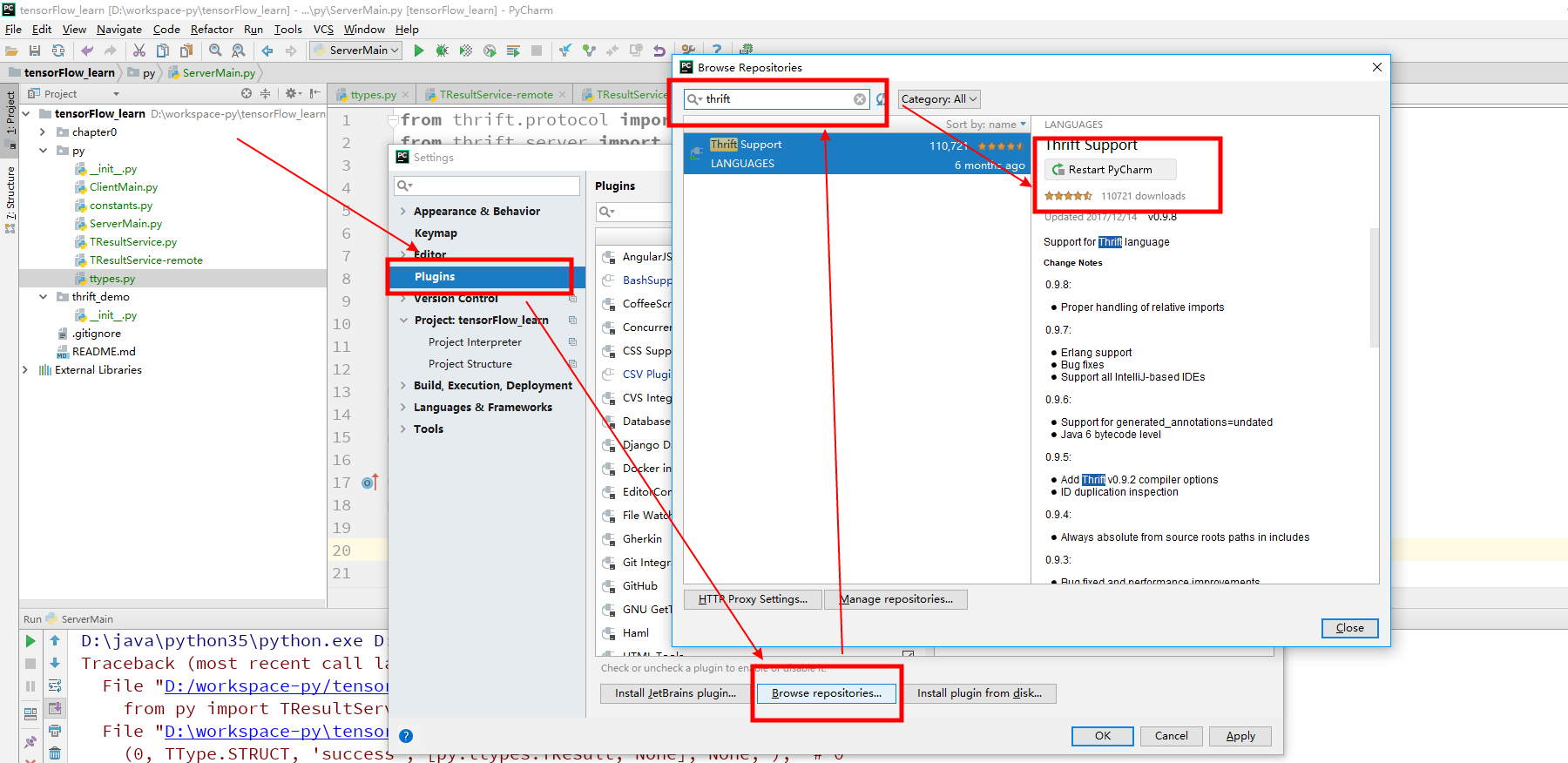
#./configure

#make

#make install

# 二、编写thrift文件

这里推荐使用pycharm的thrift插件，安装好之后，编写 .thrift 文件能够自动补全提示，安装方式如下。



我们定义两个文件，一个是结果类型的thrift文件，另一个是服务的thrift文件。

## 1. result.thrift

**namespace java** com.zx.dto *#用于生成java代码的package*

**namespace py py***# 用于存放python代码的目录*

*# 结果集，类型中包括两个值*

**struct** TResult{

1: **optional string** code;

2: **optional string** describe;

}

## 2. server.thrift

**namespace java** com.zx.service*#用于生成java代码的package*

**namespace py py***# 用于存放python代码的目录*

**include "result.thrift"***# 引入结构体*

**typedef** result.TResult Result*# 将结构体定义为一种数据类型*

*#定义一个服务，服务中有一个方法，调用方法后返回一个结果类型*

**service** TResultService{

Result getResponse(1:**string** code);

}

# 三、生成代码

生成代码的时候需要注意， 推荐使用先生成服务的代码，再生成定义类型的代码。经过试验发现有的时候，生成的代码名称会重复，意思就是后生成的会覆盖掉原来的代码。

Java： thrift-0.11.0.exe --gen java service.thrift thrift-0.11.0.exe --gen java result.thrift

python: thrift-0.11.0.exe --gen py service.thrift thrift-0.11.0.exe --gen py result.thrift

生成后的代码如下：



# 四、java代码

1. 创建maven项目，添加依赖（保证版本的一致性）

<dependency>

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

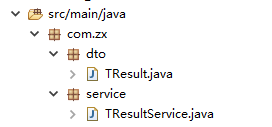
<artifactId>libthrift</artifactId>

<version>0.10.0</version>

</dependency>

2.将生成的代码复制到项目中

注意：很多时候，生成的代码使有问题的，复制后要注意查看代码是否出现异常，并即时修复。



3. 编写服务的实现方法

实现生成的TResultService中定义的Iface接口即可。

public class ServerMain implements TResultService.Iface {

@Override

public TResult getResponse(String code) throws TException {

System.out.println("java服务端,参数：" + code);

TResult result=new TResult();

result.setCode(code);

result.setDescribe("成功");

return result;

}

}

4. 编写server启动程序

public static void main(String[] args) {

try {

TServerTransport serverTransport = new TServerSocket(9090);

//定义一个TProcess处理类，创建TProcess对象

TProcessor processor = new TResultService.Processor<ServerMain>(new ServerMain());

//定义一个TServer传输对象，用于tcp的socket通信

TServer server = new TSimpleServer(new Args(serverTransport).processor(processor));

System.out.println("Starting the simple server...");

//启动服务

server.serve();

} catch (Exception e) {

e.printStackTrace();

}

}

5. 编写client的启动程序

public static void main(String[] args) {

try {

// 创建一个传输层对象（TTransport），设置调用的服务地址为本地，端口为 9090,TSocket传输方式

TTransport transport = new TSocket("localhost", 9090);

//打开socket，建立与服务器直接的socket连接

transport.open();

// 创建通信协议对象（TProtocol），设置传输协议为 TBinaryProtocol

TProtocol protocol = new TBinaryProtocol(transport);

//创建一个Thrift客户端对象

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

// 调用服务的 getResponse 方法

TResult result=client.getResponse("来自java的调用");

System.out.println(result.toString());

transport.close();

} catch (TTransportException e) {

e.printStackTrace();

} catch (TException e) {

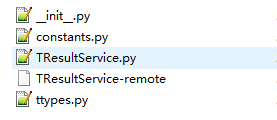
e.printStackTrace();

}

}

# 五、python代码

python生成后的代码结构如下。所有的类型都被定义在ttype类中，所有，当生成的服务代码中定义多以文件的时候，这个文件可能被覆盖。



1. 安装thrift

使用命令： pip install thrift。即可安装最新的thrift python库

2.推荐使用pycharm创建一个python项目，将生成的代码复制到项目中

3. 编写服务的实现方法

实现生成的TResultService中定义的Iface接口即可。

4. 编写server启动程序

|  |
| --- |
| **from** thrift.protocol **import** TBinaryProtocol  **from** thrift.server **import** TServer  **from** thrift.transport **import** TSocket, TTransport  **from** py **import** TResultService  **from** py **import** ttypes  \_\_HOST = **'127.0.0.1'**  \_\_PORT = 9090  **class** TResultServiceHandler(TResultService.Iface):  **def** \_\_init\_\_(self):  self.log = {}  **def** getResponse(self, code):  print(**'服务端被调用,请求参数:'**, code)  **return** ttypes.TResult(code, **'成功'**)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  handler = TResultServiceHandler()  processor = TResultService.Processor(handler)  transport = TSocket.TServerSocket(host=\_\_HOST, port=\_\_PORT)  tfactory = TTransport.TBufferedTransportFactory()  pfactory = TBinaryProtocol.TBinaryProtocolFactory()  server = TServer.TThreadPoolServer(processor, transport, tfactory, pfactory)  print(**'Starting the server'**)  server.serve()  print(**'done'**) |

5. 编写client的启动程序

|  |
| --- |
| **from** thrift.protocol **import** TBinaryProtocol  **from** thrift.server **import** TServer  **from** thrift.transport **import** TSocket, TTransport  **from** thrift.Thrift **import** TException  **from** py **import** TResultService  **from** py **import** ttypes  \_\_HOST = **'127.0.0.1'**  \_\_PORT = 9090  **try**:  *#创建一个传输层对象（TTransport），设置调用的服务地址为本地，端口为 9090,TSocket传输方式*  transport = TSocket.TSocket(\_\_HOST, \_\_PORT)  transport = TTransport.TBufferedTransport(transport)  *# 打开socket，建立与服务器直接的socket连接*  transport.open()  *# 创建通信协议对象（TProtocol），设置传输协议为*  protocol = TBinaryProtocol.TBinaryProtocol(transport)  *# 创建一个Thrift客户端对象*  client = TResultService.Client(protocol)  *# 调用服务的 getResponse 方法*  result = client.getResponse(**'来自python的调用'**)  print(result)  transport.close()  **except** TException **as** e:  print(**'exceptino'**, e) |

# 六、启动服务

注意我们代码中的地址都是localhost，IP都是9090

为了验证生成的代码的正确性，每次使用时，现将之前启动的服务关闭

1. java client 调用java server
2. python client 调用python server
3. java client 调用python server
4. python client 调用java server

1. java client 调用java server

先启动server，显示Starting the simple server...，启动成功

再启动client，client显示： Received 1 TResult(code:2333, describe:来自java的调用)，server显示： java服务端,参数：来自java的调用

2. python client 调用python server

先启动server，显示Starting the server，启动成功

再启动client，client显示： TResult(code='来自python的调用', describe='成功')，server显示：服务端被调用,请求参数: 来自python的调用

3. java client 调用python server

先启动python server，显示Starting the server，启动成功

再启动java client，client显示： TResult(code='来自java的调用', describe='成功')，server显示： java服务端,参数：来自java的调用

4. python client 调用java server

先启动java server，显示Starting the simple server...，启动成功

再启动python client，client显示： TResult(code='来自java的调用', describe='成功')，server显示： java服务端,参数：来自java的调用

# 总结

①Handler

服务端业务处理逻辑。这里就是业务代码，比如 计算两个字符串 相似度

②Processor

从Thrift框架 转移到 业务处理逻辑。因此是RPC调用，客户端要把 参数发送给服务端，而这一切由Thrift封装起来了，由Processor将收到的“数据”转交给业务逻辑去处理

③Protocol

数据的序列化与反序列化。客户端提供的是“字符串”，而数据传输是一个个的字节，因此会用到序列化与反序列化。

④Transport

传输层的数据传输。

⑤TServer

服务端的类型。服务器以何种方式来处理客户端请求，比如，一次Client请求创建一个新线程呢？还是使用线程池？……可参考：阻塞通信之Socket编程

TSimpleServer —— 单线程服务器端使用标准的阻塞式 I/O

TThreadPoolServer —— 多线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