Thrift

C-2 创建: 张林伟, 最后修改: 张林伟 2018-12-10 17:52

目录

简介

官方介绍：The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

由 Facebook 开发的远程服务调用框架 Apache Thrift，它采用接口描述语言定义并创建服务，支持可扩展的跨语言服务开发，所包含的代码生成引擎可以在多种语言中，如 C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk 等创建高效的、无缝的服务，其传输数据采用二进制格式，相对 XML 和 JSON 体积更小，对于高并发、大数据量和多语言的环境更有优势。

RPC：远程过程调用。两个应用A和B部署在不同机器上，应用A需要调用应用B的方法/函数，由于无法在同一内存空间，无法直接调用，需要通过网络来表达调用的语义和传达调用的数据。

问题：

1.通讯问题：两台机器之间主要使用TCP连接进行通讯。可以长连接，可以通讯完毕立即断掉连接，可以多个共享同一个TCP连接；

2.寻址：通讯需要知道地址（如ip）、端口号、方法名等信息；

3.序列化和反序列化：由于底层TCP连接使用二进制来传输，因此需要序列化和反序列化。

需要RPC的原因：项目过大，多个服务或应用无法在一个进程上运行，甚至无法在同一机器完成本地调用，因此需要将应用部署到不同机器，方法/函数内部写好网络通讯过程，但调用方感受不到。

谁能用通俗的语言解释一下什么是 RPC 框架？ - 知乎

安装

Thrift 安装（macos 10.13.1 thrift 0.8.0）

Demo

1.首先编写Hello.thrift

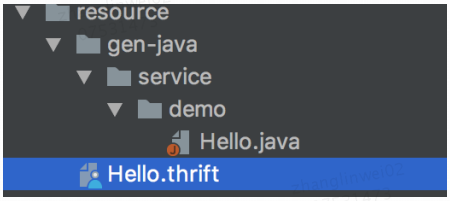
代码块 Plain Text

```
namespace java service.demo
service Hello{
    string helloString(1: string para)
    i32 helloInt(1: i32 para)
    bool helloBoolean(1: bool para)
    void helloVoid()
    string helloNull()
}
```

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。

2.使用Thrift工具编译Hello.thrift生成Hello.java

```
thrift -r --gen java Hello.thrift
```



Hello.java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。

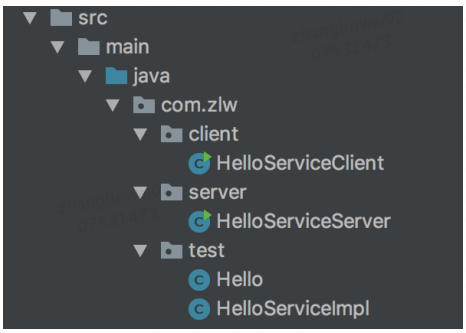
3.创建maven项目

在pom.xml中加上如下依赖

代码块 XML

```xml
    <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>1.7.5</version>
    </dependency>
```

项目结构



4.编写HelloServiceImpl.java

先将Hello.java复制到maven项目中

编写HelloServiceImpl.java实现Hello.Iface接口

代码块 Java

```java
package com.zlw.test;

/**
 * Desc:
 * ------------------------------------
 * Author:zhanglinwei02@meituan.com
 * Date:2018/7/16
 * Time:19:20
 */

import org.apache.thrift.TException;

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;
    }
}
```

5.编写服务端代码HelloServiceServer.java

代码块 Java

```
package com.zlw.server;

import com.zlw.test.Hello;
import com.zlw.test.HelloServiceImpl;
import org.apache.thrift.TProcessor;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.server.TServer;
import org.apache.thrift.server.TSimpleServer;
import org.apache.thrift.transport.TServerSocket;
import org.apache.thrift.transport.TTransportException;

/**
 * Desc: 创建服务端实现代码HelloServiceServer, 把HelloServiceImpl作为一个具体的处理器传递给Thrift服务器
 * ---------------------------------
 * Author:zhanglinwei02@meituan.com
 * Date:2018/7/16
 * Time:19:20
 */
public class HelloServiceServer {
    /**
     * 启动thrift服务器
     * @param args
     */
    public static void main(String[] args) {
        try {
            // 关联处理器与 Hello 服务的实现
            TProcessor tprocessor = new Hello.Processor<Hello.Iface>(new HelloServiceImpl());
            // 设置服务端口为9898
            TServerSocket serverTransport = new TServerSocket(9898);
            TServer.Args tArgs = new TServer.Args(serverTransport);
            tArgs.processor(tprocessor);
            tArgs.protocolFactory(new TBinaryProtocol.Factory());
            TServer server = new TSimpleServer(tArgs);
            System.out.println("Start server on port 9898...");
            server.serve();
        }catch (TTransportException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

6.编写客户端代码HelloServiceClient.java

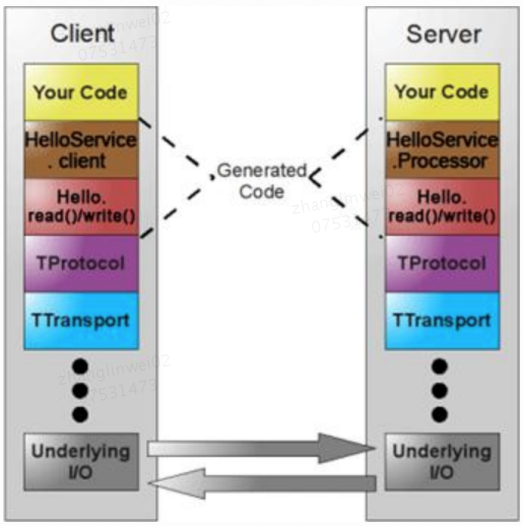
代码块 Java

```java
package com.zlw.client;

import com.zlw.test.Hello;
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.protocol.TProtocol;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TTransport;
import org.apache.thrift.transport.TTransportException;

/**
 * Desc: 创建客户端实现代码HelloServiceClient,调用Hello.client访问服务端的逻辑实现
 * ----------------------------------
 * Author:zhanglinwei02@meituan.com
 * Date:2018/7/17
 * Time:10:47
 */
public class HelloServiceClient {
    public static void main(String[] args) {
        System.out.println("客户端启动....");
        TTransport transport = null;
        try {
            // 设置调用的服务地址为本地，端口为 7911
            transport = new TSocket("localhost", 9898, 30000);
            // 协议要和服务端一致
            TProtocol protocol = new TBinaryProtocol(transport);
            Hello.Client client = new Hello.Client(protocol);
            transport.open();
            // 调用服务端helloString()方法
            String result = client.helloString("哈哈");
            System.out.println(result);
        } catch (TTransportException e) {
            e.printStackTrace();
        } catch (TException e) {
            e.printStackTrace();
        } finally {
            if (null != transport) {
                transport.close();
            }
        }
    }
}
```

7.运行

分别运行HelloServiceServer.java和HelloServiceClient.java

Thrift架构



黄色部分是用户实现的业务逻辑，褐色部分是根据 Thrift 定义的服务接口描述文件生成的客户端和服务器端代码框架，红色部分是根据 Thrift 文件生成代码实现数据的读写操作。红色部分以下是 Thrift 的传输体系、协议以及底层 I/O 通信，使用 Thrift 可以很方便的定义一个服务并且选择不同的传输协议和传输层而不用重新生成代码。

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。

server端调用细节：server端调用了 serve 方法后，进入阻塞监听状态，其阻塞在 TServerSocket 的 accept 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。

client端调用细节：client端调用了 Hello.Client 的 helloVoid 方法，在 helloVoid 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。

数据类型

Thrift 脚本可定义的数据类型包括以下几种类型：

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：未知编码文本或二进制字符串，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

协议

传输协议上总体划分为文本 (text) 和二进制 (binary) 传输协议。

常用协议：

- 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

传输层

常见传输层：

- TSocket —— 使用阻塞式 I/O 进行传输，是最常见的模式
- TFramedTransport —— 使用非阻塞方式，按块的大小进行传输，类似于 Java 中的 NIO
- TNonblockingTransport —— 使用非阻塞方式，用于构建异步客户端

服务端类型

常见的服务端类型：

- 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

Facebook/Swift

使用 Thrift 的 IDL 语言生成的 Thrift 接口文件复杂，导致接口增减方法不便。swift 是 Facebook 开发的用于简化 Thrift 文件的工具。主要采用注解形式来声明 Thrift 的接口和 POJO 类。

1.定义 POJO 类

代码块 Java

@ThriftField(1) 注解用于成员变量的 get 方法，括号内 1 代表字段 ID， set 方法不需要 ID。

2.定义 Thrift 接口

代码块 Java

常见问题

1.thrift的service方法名不能重复

因为thrift支持多语言，而其他语言并不是向Java语言一样支持重载。

2.方法新增参数，旧版本client会无法调用rpc服务

兼容方案：保留旧版本接口，标注废弃，然后重新写一个新方法，但新版client方法名可以跟旧版一样，但实际调用service写的新方法

参考资料

http://thrift.apache.org/tutorial/java

https://www.ibm.com/developerworks/cn/java/j-lo-apachethrift/index.html

https://www.cnblogs.com/fingerboy/p/6424248.html