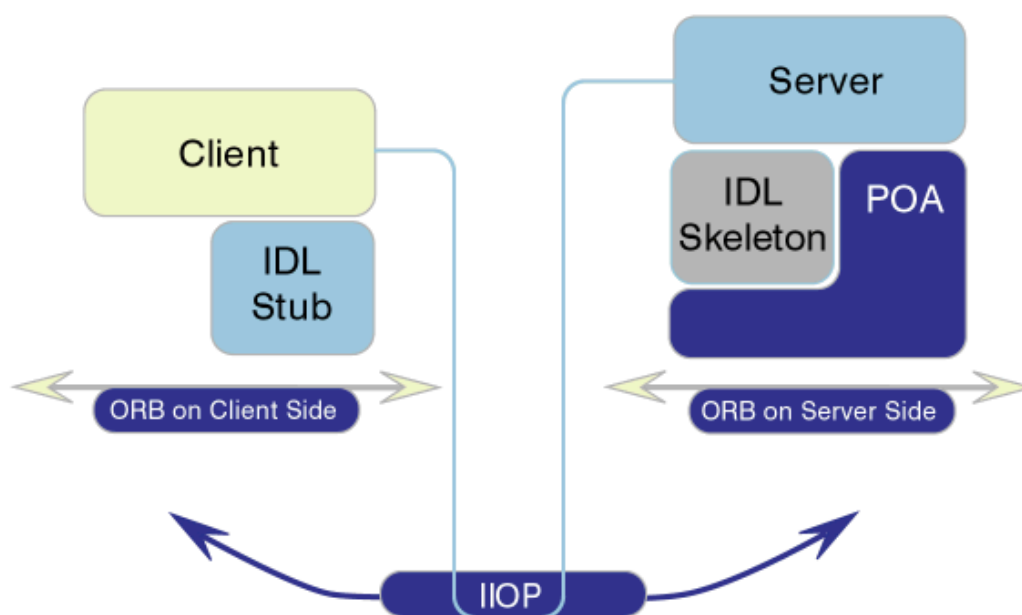


Thrift 学习

Thrift 简介

出自于 Facebook，后来提交 Apache 基金将其作为一个开源项目，对于 Facebook 来说，创造 thrift 是为了解决 Facebook 系统中各系统间大数据量的传输通信以及系统之间语言环境不同需要跨平台的特性。在多种不同语言之间通信，thrift 可以作为二进制的高性能的通信中间件，支持数据（对象）序列化和多种类型的 RPC 服务。Thrift 适用于程序对程序静态的数据交换，需要先确定好他的数据结构，他是完全静态化的，当数据结构发生变化时，必须重新编辑 IDL 文件，代码生成，再编译载入的流程，跟其他 IDL 工具相比较可以视为是 Thrift 的弱项，Thrift 适用于搭建大型数据交换及存储的通用工具，对于大型系统中的内部数据传输相对于 JSON 和 xml 无论在性能、传输大小上有明显的优势。Thrift 是 IDL (Interface Definition Language) 描述性语言的一个具体实现。

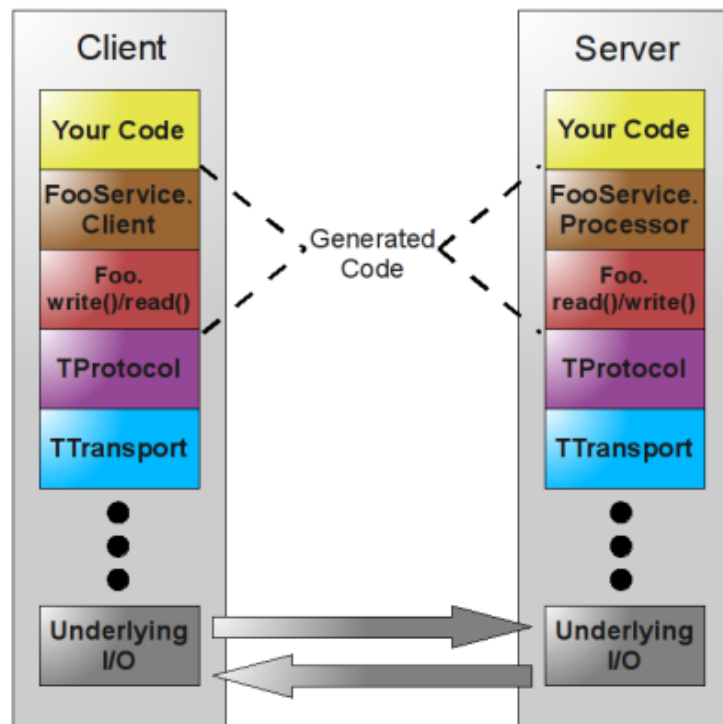
传统的 CORBA (Common Object Request Broker Architecture)



Thrift 基础架构

Thrift 是一个服务端和客户端的架构体系，类似于 XML-RPC+JAVA-to-IDL+Serialization Tools=Thrift。具有自己定义的传输协议规范 TProtocol 和传输数据标准 TTransports，通过 IDL 脚本对传输数据的数据结构 (Struct) 和传输数据的业务逻辑 (Service) 根据不同的运行环境快速构建响应的代码，并通过自己内部的序列化机制堆传输的数据进行简化和压缩提高高

并发，减少大型系统中数据交互的成本。



- 1, 业务逻辑实现：通过 Thrift 脚本生成的代码
- 2, 客户端和服务端对应的 Service：根据生成代码构建的客户端和处理器的代码
- 3, 读写操作的计算结果：client 和 Server 端的计算结果
- 4, TProtocol:
- 5, TTransport
- 6, 底层 IO 通信

从 TProtocol 下面 3 个部分是 Thrift 的传输体系和传输协议以及底层 I/O 通信，Thrift 并且提供 堵塞、非阻塞，单线程、多线程的模式运行在服务器上，还可以配合服务器/容器一起运行，可以和现有 JEE 服务器/Web 容器无缝的结合。

数据类型

- Base Type : 基本类型
- Struct: 结构体类型
- Container: 容器类型, List, Set, Map
- Exception: 异常类型
- Service: 定义对象的接口, 和一系列方法

协议

Thrift 可以选择客户端和服务端端的传输通信协议的类别，在传输协议上总体划分为文本和二进制传输协议。

- TBinaryProtocol: 二进制编码格式进行数据传输。
- TCompactProtocol: 这种协议是非常有效的, 使用 Variable-Length Quantity (VLQ) 编码对数据进行压缩。
- TJSONProtocol: 使用 JSON 的数据编码协议进行数据传输。
- TSimpleJSONProtocol: 适用于通过脚本语言解析。
- TDebugProtocol: 在开发的过程中帮助开发人员调试, 以文本的形式展现方便阅读。

传输层

- TSocket: 使用阻塞式 I/O 进行传输, 也是常见的模式。
- TFramedTransport: 使用非阻塞方式, 安块的大小, 进行传输, 类似于 Java 的 NIO。
- TFileTransport: 按照文件的方式进行传输。
- TMemoryTransport: 使用内存 I/O, 就好比 Java 中的 ByteArrayOutputStream 实现
- TZlibTransport: 使用执行 zlib 压缩, 不提供 Java 的实现

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, 并实现 Java 中的 NIO。

Thrift 与其他传输方式的比较

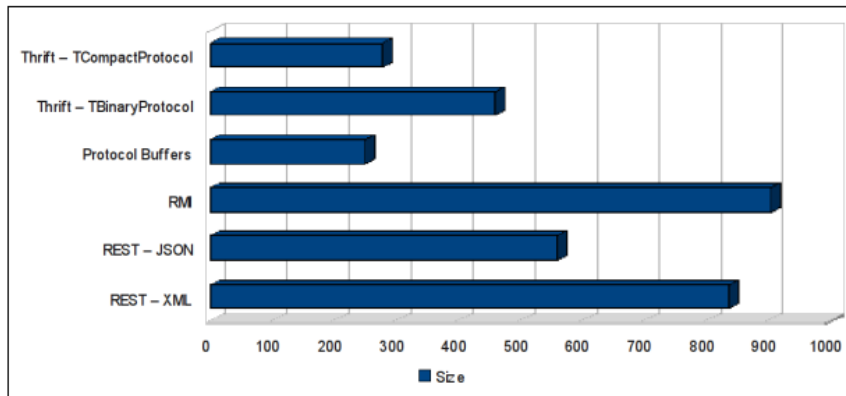
XML 与 JSON 相比体积比较大

JSON 体积小, 新颖, 不够完善。

Thrift 体积小, 使用起来比较麻烦, 不如前两者轻便, 但是对于 1 并发高, 2, 数据传输量大, 3, 多语言环境

对比:

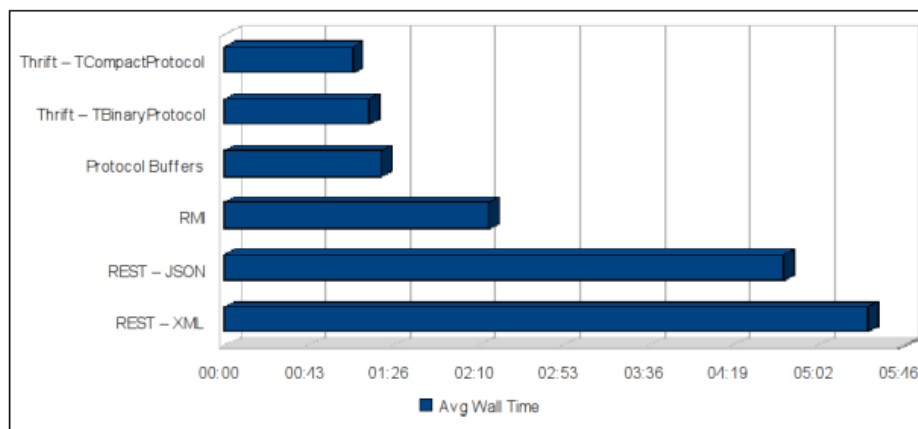
- 1, 传输内容所产生的大小
- 2, 传输过程中服务端和客户端所产生的开销



Method	Size*	% Larger than TCompactProtocol
Thrift — TCompactProtocol	278	N/A
Thrift — TBinaryProtocol	460	65.47%
Protocol Buffers**	250	-10.07%
RMI (using Object Serialization for estimate)**	905	225.54%
REST — JSON	559	101.08%
REST — XML	836	200.72%

最大是 RMI 其次是 XML，使用 Thrift 的 TCompactProtocol 协议和 Google 的 Protocol Buffers 相差的不算太多。

用 Thrift 中的协议和其他方式所产生的运行开销比较结果如下：



	Server CPU %	Avg Client CPU %	Avg Wall Time
REST — XML	12.00%	80.75%	05:27.45
REST — JSON	20.00%	75.00%	04:44.83
RMI	16.00%	46.50%	02:14.54
Protocol Buffers	30.00%	37.75%	01:19.48
Thrift — TBinaryProtocol	33.00%	21.00%	01:13.65
Thrift — TCompactProtocol	30.00%	22.50%	01:05.12

Thrift JAVA 开发详解

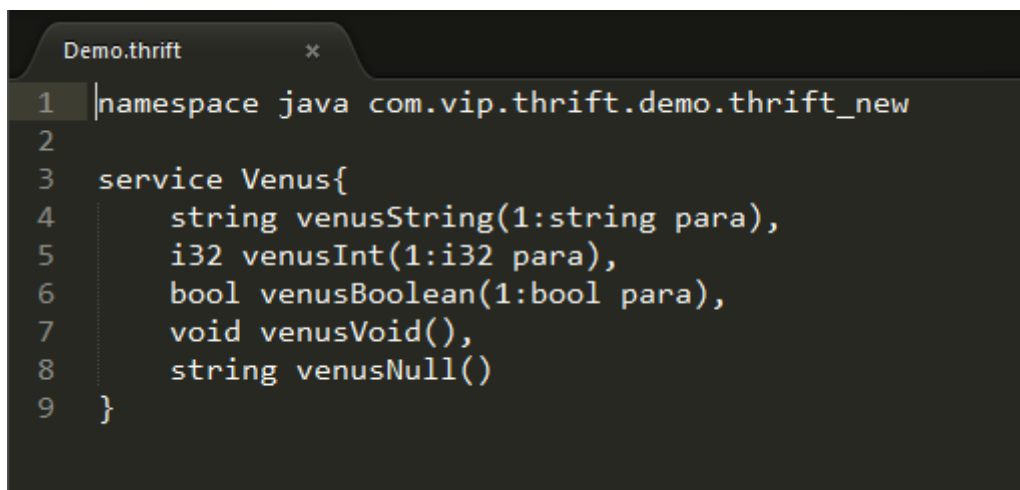
代码地址: [git@github.com:zhaohuizhang/thrift_demo.git](https://github.com:zhaohuizhang/thrift_demo.git)

创建 Maven 工程

添加相关的 Jar 包

```
<dependency>
  <groupId>org.apache.thrift</groupId>
  <artifactId>libthrift</artifactId>
  <version>0.9.2</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.8</version>
</dependency>
```

编写 IDL 文件



```
Demo.thrift
1 namespace java com.vip.thrift.demo.thrift_new
2
3 service Venus{
4     string venusString(1:string para),
5     i32 venusInt(1:i32 para),
6     bool venusBoolean(1:bool para),
7     void venusVoid(),
8     string venusNull()
9 }
```

生成代码

```
thrift --gen java Demo.thrift
```

```
2* * Autogenerated by Thrift Compiler (0.9.2)
7 package com.vip.thrift.demo.thrift_new;
8
9* import java.util.ArrayList;
31
32 @SuppressWarnings({ "cast", "rawtypes", "serial", "unchecked" })
33 @Generated(value = "Autogenerated by Thrift Compiler (0.9.2)", date = "2015-9-28")
34 public class Venus {
35
36*   public interface Iface {
49
50*   public interface AsyncIface {
68
69*   public static class Client extends org.apache.thrift.TServiceClient implements Iface {
192
193*   public static class AsyncClient extends org.apache.thrift.async.TAsyncClient implements AsyncIface {
405
406*   public static class Processor<I extends Iface> extends org.apache.thrift.TBaseProcessor<I> implements
535
536*   public static class AsyncProcessor<I extends AsyncIface> extends org.apache.thrift.TBaseAsyncProcessor<I> {
837
838*   public static class venusString_args implements
1200
1201*   public static class venusString_result implements
1563
1564*   public static class venusInt_args implements org.apache.thrift.TBase<venusInt_args, venusInt_args._Fields>,
1924
1925*   public static class venusInt_result implements org.apache.thrift.TBase<venusInt_result, venusInt_result._Fields>,
2287
```

编写实现类，实现 Venus.Iface

```
VenusImpl.java Venus.java VenusServer... VenusClient... AsyncClient... OrderModel.java
1 package com.vip.thrift.demo.thrift_new;
2
3 import org.apache.thrift.TException;
4
5 public class VenusImpl implements Venus.Iface {
6
7*   public String venusString(String para) throws TException {
11
12*   public int venusInt(int para) throws TException {
16
17*   public boolean venusBoolean(boolean para) throws TException {
21
22*   public void venusVoid() throws TException {
26
27*   public String venusNull() throws TException {
31
32 }
33
```

编写服务端，发布服务

（阻塞式 IO/非阻塞式 IO+多线程处理）服务

```
13
14 public class VenusServer {
15
16     * 阻塞式、多线程处理
17     public void startServer() {
18
19         try {
20             // 设置传输通道，普通通道
21             // TServerTransport serverTransport = new TServerSocket(7911);
22
23             // 传输通道 - 非阻塞方式
24             TNonblockingServerTransport serverTransport = new TNonblockingServerSocket(7911);
25
26             // 异步IO，需要使用TFramedTransport，它将分块缓存读取
27             TTransportFactory transportFactory = new TFramedTransport.Factory();
28
29             // 使用高密度二进制协议
30             TProtocolFactory proFactory = new TCompactProtocol.Factory();
31
32             // TProtocolFactory proFactory = new TBinaryProtocol.Factory();
33             // 设置处理器VenusImpl
34             TProcessor processor = new Venus.Processor(new VenusImpl());
35
36             Args args = new Args(serverTransport);
37             args.processor(processor);
38             args.protocolFactory(proFactory);
39             args.transportFactory(transportFactory);
40
41             // 创建服务器
42             TServer server = new TThreadedSelectorServer(args);
```

编写客户端

调用（阻塞式 IO/NIO+多线程处理）服务

```

10
11 public class VenusClient {
12     * 调用阻塞式、多线程处理服务
13     public void startClient() {
14         try {
15             // 设置传输通道-普通IO流通道
16             // TTransport transport = new TSocket("localhost", 7911);
17
18             // 调用非阻塞IO (NIO) 服务的客户端
19             // 设置传输通道, 对于非阻塞服务, 需要使用TFramedTransport, 它将数据分块发送
20             TTransport transport = new TFramedTransport(new TSocket("localhost", 7911)
21
22             // 使用高密度二进制协议
23             TProtocol protocol = new TCompactProtocol(transport);
24
25             // TProtocol protocol = new TBinaryProtocol(transport);
26             // 创建Client
27             Venus.Client client = new Venus.Client(protocol);
28
29             transport.open();
30             long start = System.currentTimeMillis();
31             Random random = new Random();
32             for (int i = 0; i < 10000; i++) {
33                 System.out.println(client.venusBoolean(true));
34                 System.out.println(client.venusInt(random.nextInt(10000)));
35                 System.out.println(client.venusNull());
36                 System.out.println(client.venusString("Do Something!"));
37                 client.venusVoid();
38             }
39             System.out.println("Run time:" + (System.currentTimeMillis() - start));
40
41
42
43

```