Thrift 那点事

Thrift 的概念

Apache 的 Thrift 软件框架，是用来进行可伸缩的、跨语言的服务开发，它通过一个代码生成引擎来构建高效、无缝的服务，这些服务能够实现跨语言调度，目前支持的语言有： C++，Java、Python 等。

Thrift 的用途

跟 dubbo 这些做比较，其实 Thrift 自身在分布式上没有什么优势，它没有像 dubbo 那样有注册中心、服务发现和一些监控。Thrift 就是通过定义一个用自己的语言描述的公用的服务（Thrift 文件），然后又能提供-gen 生成各种语言去生成服务和调用服务。

Thrift 的五大金刚

Transport 传输层

主要分两个维度：**服务端**和**客户端**

- 服务端的父类 **TServerTransport**

分为阻塞 TServerSocket 和非阻塞 TNonblockingTransport（这是个抽象类，实现是 TNonblockingServerSocket）。 所以从实现上来看都是 XXServerSocket，这样好记点。

- 客户端的父类是 **TTransport**，同样也会分阻塞和非阻塞，

更有分不同传输形式的，比如 TSocket 面向 socket 编程的，而 THttpClient 就是 http 协议传输，TFileTransport 则是文件传输的。

这里的 TSocket 是同步阻塞的，也就是传统的 IO，而 TFrameTransport 是同步非阻塞的，相当于 NIO，还有 TNonblockingTransport 是异步非阻塞的，相当于 AIO。后两者都是非阻塞的，所以都要对应服务端的 TNonblockingServerTransport

Protocol 协议

传输的协议，在 thrift 中提供的协议不多，主要分为两种：**文本传输**（如 json、compact 压缩）和**二进制传输**（binary）。无论是文本传输还是二进制传输，最终他们都会根据设定的编码方式（如 UTF-8）转换成相应的二进制字节，然后再传输！文本传输和二进制传输的唯一区别在于换行符的处理。

Processor

在 thrift 自动生成 java 文件中，IFace，Client 和 Processor 是最重要的静态内部类。这个 Processor 就是服务端在实现具体接口逻辑的，然后在服务端启动的时候告诉服务器，这个接口要这么处理！客户端根本不需要用到。

Client

这个是在客户端调用服务时候用的，内部的方法都包含了我们在接口里面定义的所有 method，所以客户端传好相应的参数后，直接 client 上调用。

TServer

跟 TServerTransport 对应的，其实就是 processor+Transport+protocol 的一个组合，就是强行把这三者联合在一起使用，对外提供服务 TServer.serve();主要的有三个 TServer：

- TSimpleServer 单线程阻塞 IO，这个也在测试那里玩玩
- TThreadServer 多线程阻塞 IO
- TNonblockingServer 这个当然是多线程非阻塞 IO 了

Thrift 的其它缺点

1. 不支持双向通道，如果要支持双向通道比较麻烦；
2. rpc 方法非线程安全，这就是为何很多时候服务器会被挂死，是因为客户端的并发 rpc 调用导致的，只需要客户端对 rpc 的调用进行串行化即可。统一服务器应答的时候，也需要串行化，否则有可能会把对方给挂死。特别是在多线程情况下

Thrift 与 Thwift

swift 是一个用于创建 thrift 序列化类型和服务的 java 工具库，使用 swfit 可以生成非常简洁的 java 代码。并且更重要的是可以通过 java 代码生成接口描述文件(Thrift interface description language,IDL).

GRPC 的崛起

GRPC 是 google 开源的一个高性能、跨语言的 RPC 框架，基于 HTTP2 协议，基于 protobuf 3.x，基于 Netty 4.x +。GRPC 与 thrift、avro-rpc 等其实在总体原理上并没有太大的区别。

优点：

1、支持 Protocol Buffer 已被证明是一个很高效的序列化技术；

2、支持 HTTP 2.0 标准化的协议，,因为添加了头信息,可以方便在**框架层面**对调用做拦截和控制(比如说限流,调用链分析,安全认证等)。

3、基于 HTTP2，具有流双向流传输等特性。

缺点：

1、截止到今日，GRPC 仍然处于开发阶段，尚没有 release 版本，而且特性也很多需要补充；

2、虽然 HTTP2 协议已成定局，但尚未被主流 web 容器包括代理服务器支持（绝大部多数 HTTP Server、Nginx 都尚不支持，即 Nginx 不能将 GRPC 请求作为 HTTP 请求来负载均衡，而是作为普通的 TCP 请求），这意味着 GRPC 在 HTTP 负载均衡方面尚有欠缺；

3、GRPC 尚未提供连接池，尚未提供“服务发现”、“负载均衡”机制；

4、Spring 容器尚未提供整合；

5、GRPC 生成的接口，调用方式实在是不太便捷。