14 | 开源RPC框架如何选型？

2018-09-22 胡忠想

14 | 开源RPC框架如何选型？
朗读人：胡忠想　12′51″ | 5.90M

专栏第 6 期我给你讲解了 RPC 远程调用的原理，简单回顾一下一个完整的 RPC 框架主要有三部分组成：通信框架、通信协议、序列化和反序列化格式。根据我的经验，想要开发一个完整的 RPC 框架，并且应用到线上生产环境，至少需要投入三个人力半年以上的时间。这对于大部分中小团队来说，人力成本和时间成本都是不可接受的，所以我建议还是选择开源的 RPC 框架比较合适。

那么业界应用比较广泛的开源 RPC 框架有哪些呢？

简单划分的话，主要分为两类：一类是跟某种特定语言平台绑定的，另一类是与语言无关即跨语言平台的。

跟语言平台绑定的开源 RPC 框架主要有下面几种。

- Dubbo：国内最早开源的 RPC 框架，由阿里巴巴公司开发并于 2011 年末对外开源，仅支持 Java 语言。
- Motan：微博内部使用的 RPC 框架，于 2016 年对外开源，仅支持 Java 语言。

- Tars：腾讯内部使用的 RPC 框架，于 2017 年对外开源，仅支持 C++ 语言。
- Spring Cloud：国外 Pivotal 公司 2014 年对外开源的 RPC 框架，仅支持 Java 语言，最近几年生态发展得比较好，是比较火的 RPC 框架。

而跨语言平台的开源 RPC 框架主要有以下几种。

- gRPC：Google 于 2015 年对外开源的跨语言 RPC 框架，支持常用的 C++、Java、Python、Go、Ruby、PHP、Android Java、Objective-C 等多种语言。
- Thrift：最初是由 Facebook 开发的内部系统跨语言的 RPC 框架，2007 年贡献给了 Apache 基金，成为 Apache 开源项目之一，支持常用的 C++、Java、PHP、Python、Ruby、Erlang 等多种语言。

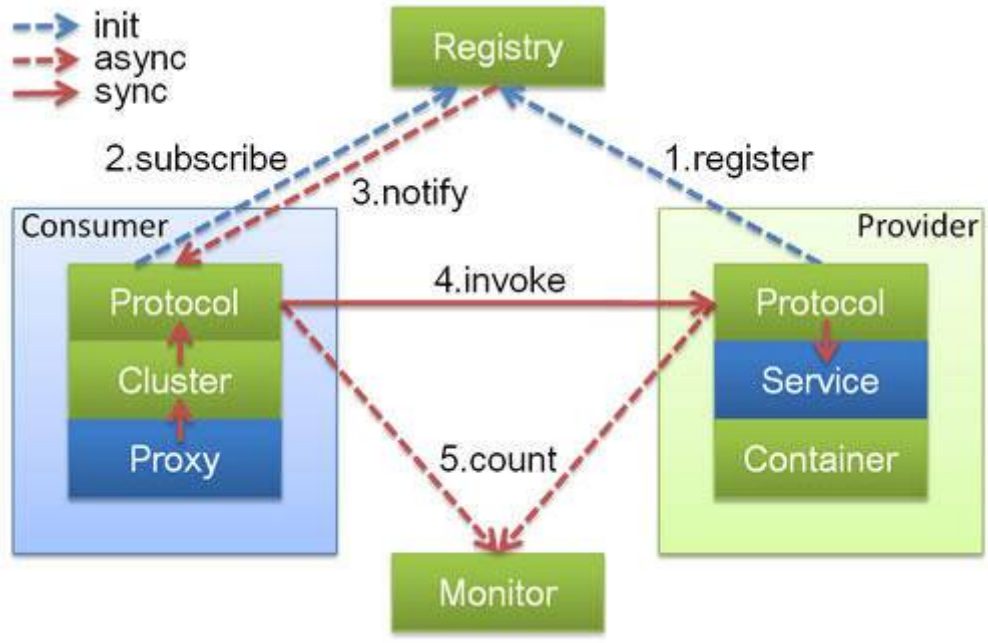
所以很明显，如果你的业务场景仅仅局限于一种语言的话，可以选择跟语言绑定的 RPC 框架中的一种；如果涉及多个语言平台之间的相互调用，就应该选择跨语言平台的 RPC 框架。

针对每一种 RPC 框架，它们具体有何区别？该如何选择呢？接下来，我就从每个框架的实现角度来具体给你讲解。当你知道了他们的具体实现，也就能知道他们的优缺点以及适用场景了。

限定语言平台的开源 RPC 框架

1. Dubbo

先来聊聊 Dubbo，Dubbo 可以说是国内开源最早的 RPC 框架了，目前只支持 Java 语言，它的架构可以用下面这张图展示。



（图片来源：https://dubbo.incubator.apache.org/docs/zh-cn/dev/sources/images/dubbo-relation.jpg）

从图中你能看到，Dubbo 的架构主要包含四个角色，其中 Consumer 是服务消费者，Provider 是服务提供者，Registry 是注册中心，Monitor 是监控系统。

具体的交互流程是 Consumer 一端通过注册中心获取到 Provider 节点后，通过 Dubbo 的客户端 SDK 与 Provider 建立连接，并发起调用。Provider 一端通过 Dubbo 的服务端 SDK 接收到 Consumer 的请求，处理后再把结果返回给 Consumer。

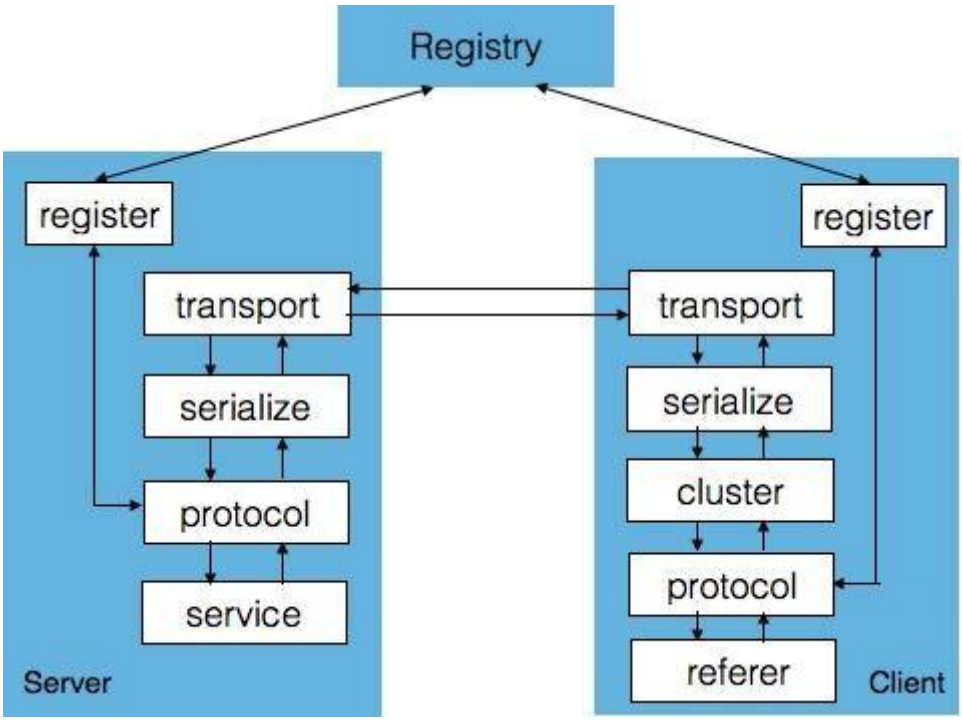
可以看出服务消费者和服务提供者都需要引入 Dubbo 的 SDK 才来完成 RPC 调用，因为 Dubbo 本身是采用 Java 语言实现的，所以要求服务消费者和服务提供者也都必须采用 Java 语言实现才可以应用。

我们再来看下 Dubbo 的调用框架是如何实现的。

- 通信框架方面，Dubbo 默认采用了 Netty 作为通信框架。
- 通信协议方面，Dubbo 除了支持私有的 Dubbo 协议外，还支持 RMI 协议、Hession 协议、HTTP 协议、Thrift 协议等。
- 序列化格式方面，Dubbo 支持多种序列化格式，比如 Dubbo、Hession、JSON、Kryo、FST 等。

2. Motan

Motan 是国内另外一个比较有名的开源的 RPC 框架，同样也只支持 Java 语言实现，它的架构可以用下面这张图描述。



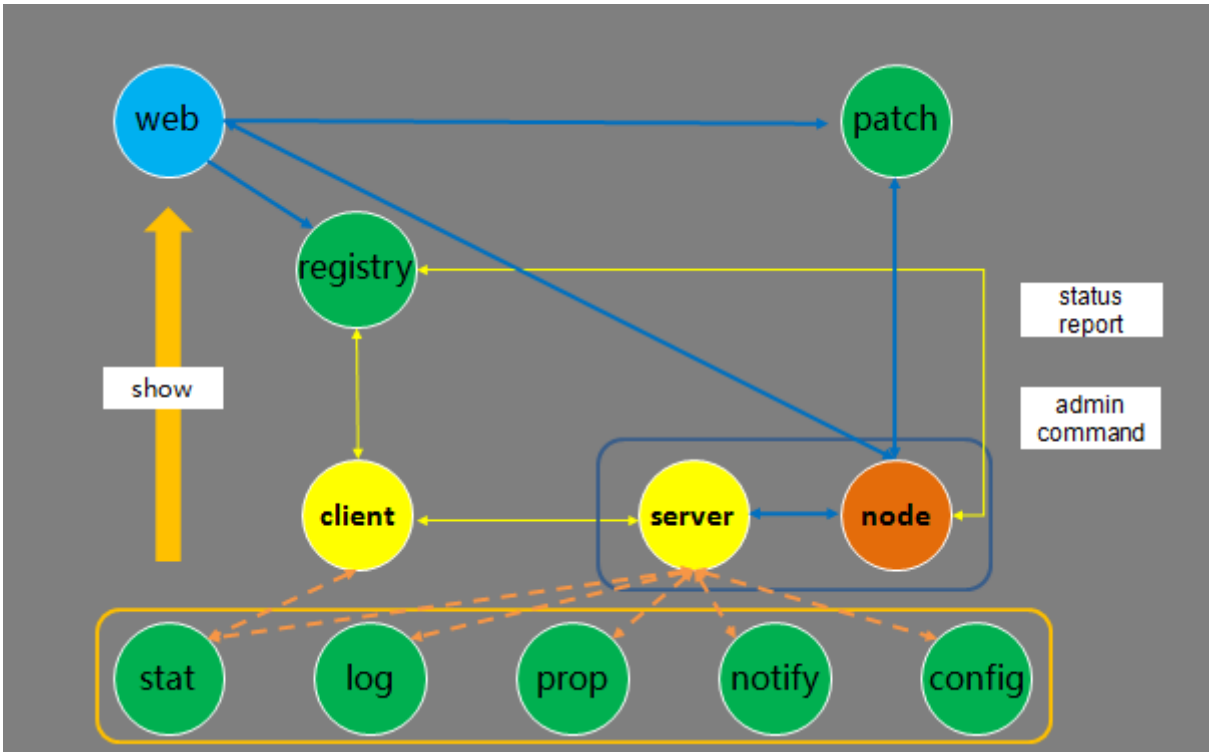
（图片来源：https://github.com/weibocom/motan/wiki/media/14612352579675.jpg）

Motan 与 Dubbo 的架构类似，都需要在 Client 端（服务消费者）和 Server 端（服务提供者）引入 SDK，其中 Motan 框架主要包含下面几个功能模块。

- register：用来和注册中心交互，包括注册服务、订阅服务、服务变更通知、服务心跳发送等功能。Server 端会在系统初始化时通过 register 模块注册服务，Client 端会在系统初始化时通过 register 模块订阅到具体提供服务的 Server 列表，当 Server 列表发生变更时也由 register 模块通知 Client。
- protocol：用来进行 RPC 服务的描述和 RPC 服务的配置管理，这一层还可以添加不同功能的 filter 用来完成统计、并发限制等功能。
- serialize：将 RPC 请求中的参数、结果等对象进行序列化与反序列化，即进行对象与字节流的互相转换，默认使用对 Java 更友好的 Hessian 2 进行序列化。
- transport：用来进行远程通信，默认使用 Netty NIO 的 TCP 长链接方式。
- cluster：Client 端使用的模块，cluster 是一组可用的 Server 在逻辑上的封装，包含若干可以提供 RPC 服务的 Server，实际请求时会根据不同的高可用与负载均衡策略选择一个可用的 Server 发起远程调用。

3. Tars

Tars 是腾讯根据内部多年使用微服务架构的实践，总结而成的开源项目，仅支持 C++ 语言，它的架构图如下。



（图片来源：

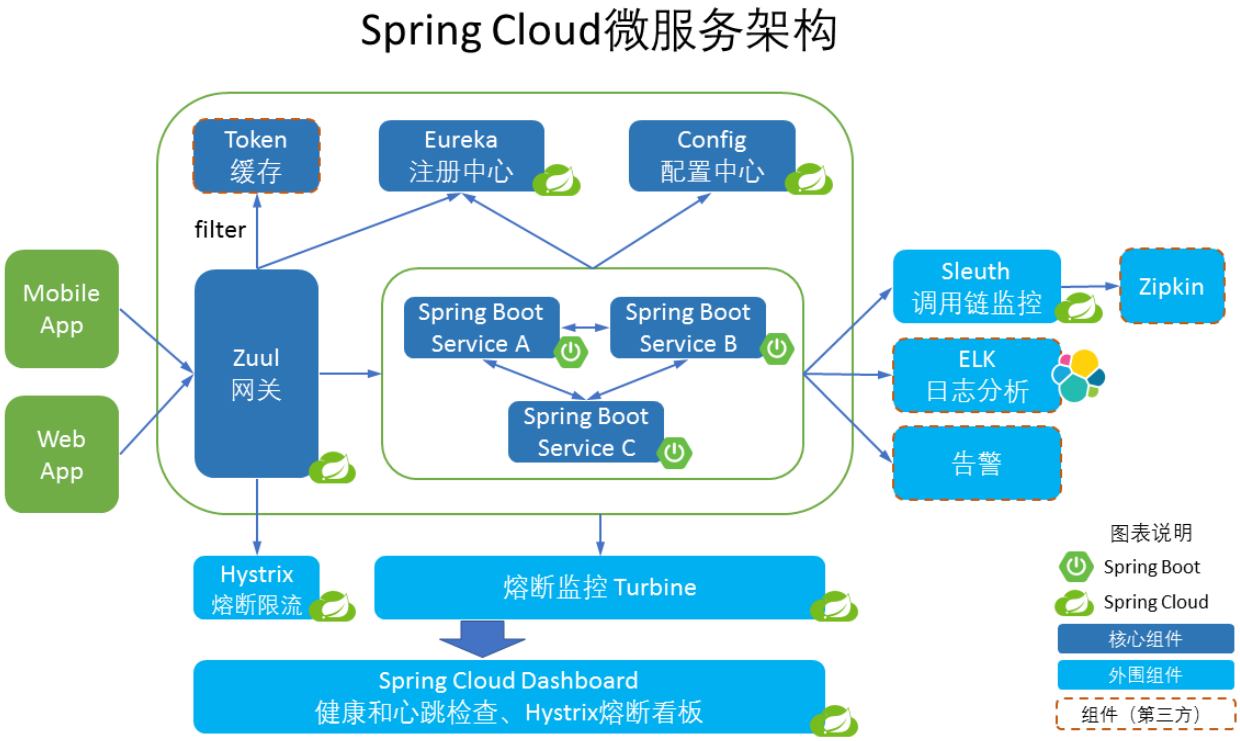
https://github.com/TarsCloud/Tars/blob/master/docs/images/tars_jiaohu.png ）

Tars 的架构交互主要包括以下几个流程：

- 服务发布流程：在 web 系统上传 server 的发布包到 patch，上传成功后，在 web 上提交发布 server 请求，由 registry 服务传达到 node，然后 node 拉取 server 的发布包到本地，拉起 server 服务。
- 管理命令流程：web 系统上的可以提交管理 server 服务命令请求，由 registry 服务传达到 node 服务，然后由 node 向 server 发送管理命令。
- 心跳上报流程：server 服务运行后，会定期上报心跳到 node，node 然后把服务心跳信息上报到 registry 服务，由 registry 进行统一管理。
- 信息上报流程：server 服务运行后，会定期上报统计信息到 stat，打印远程日志到 log，定期上报属性信息到 prop、上报异常信息到 notify、从 config 拉取服务配置信息。
- client 访问 server 流程：client 可以通过 server 的对象名 Obj 间接访问 server，client 会从 registry 上拉取 server 的路由信息（如 IP、Port 信息），然后根据具体的业务特性（同步或者异步，TCP 或者 UDP 方式）访问 server（当然 client 也可以通过 IP/Port 直接访问 server）。

4. Spring Cloud

Spring Cloud 是为了解决微服务架构中服务治理而提供的一系列功能的开发框架，它是完全基于 Spring Boot 进行开发的，Spring Cloud 利用 Spring Boot 特性整合了开源行业中优秀的组件，整体对外提供了一套在微服务架构中服务治理的解决方案。因为 Spring Boot 是用 Java 语言编写的，所以目前 Spring Cloud 也只支持 Java 语言平台，它的架构图可以用下面这张图来描述。



（图片来源：http://www.hyhblog.cn/wp-content/uploads/2018/07/Arch-Design-Spring-Cloud-1024x576.png）

由此可见，Spring Cloud 微服务架构是由多个组件一起组成的，各个组件的交互流程如下。

- 请求统一通过 API 网关 Zuul 来访问内部服务，先经过 Token 进行安全认证。
- 通过安全认证后，网关 Zuul 从注册中心 Eureka 获取可用服务节点列表。
- 从可用服务节点中选取一个可用节点，然后把请求分发到这个节点。
- 整个请求过程中，Hystrix 组件负责处理服务超时熔断，Turbine 组件负责监控服务间的调用和熔断相关指标，Sleuth 组件负责调用链监控，ELK 负责日志分析。

5. 对比选型

介绍完这 4 种限定语言的开源 RPC 框架后，我们该如何选择呢？

很显然，如果你的语言平台是 C++，那么只能选择 Tars；而如果是 Java 的话，可以选择 Dubbo、Motan 或者 Spring Cloud。这时你又要问了，它们三个又该如何抉择呢？

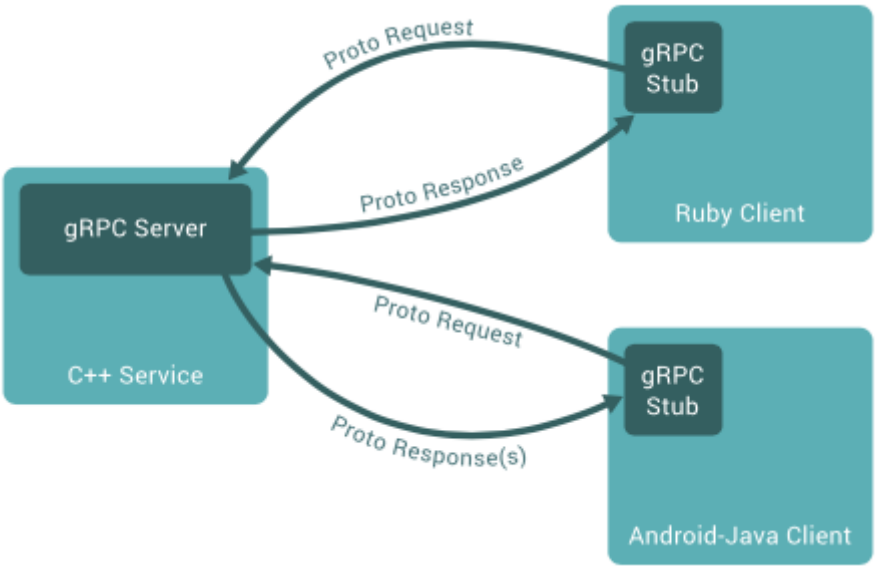
仔细分析，可以看出 Spring Cloud 不仅提供了基本的 RPC 框架功能，还提供了服务注册组件、配置中心组件、负载均衡组件、断路器组件、分布式消息追踪组件等一系列组件，也难怪被技术圈的人称之为“Spring Cloud 全家桶”。如果你不想自己实现以上这些功能，那么 Spring Cloud 基本可以满足你的全部需求。而 Dubbo、Motan 基本上只提供了最基础的 RPC 框架的功能，其他微服务组件都需要自己去实现。

不过由于 Spring Cloud 的 RPC 通信采用了 HTTP 协议，相比 Dubbo 和 Motan 所采用的私有协议来说，在高并发的通信场景下，性能相对要差一些，所以对性能有苛刻要求的情况下，可以考虑 Dubbo 和 Motan。

跨语言平台的开源 RPC 框架

1. gRPC

先来看下 gRPC，它的原理是通过 IDL（Interface Definition Language）文件定义服务接口的参数和返回值类型，然后通过代码生成程序生成服务端和客户端的具体实现代码，这样在 gRPC 里，客户端应用可以像调用本地对象一样调用另一台服务器上对应的方法。



（图片来源：https://grpc.io/img/landing-2.svg）

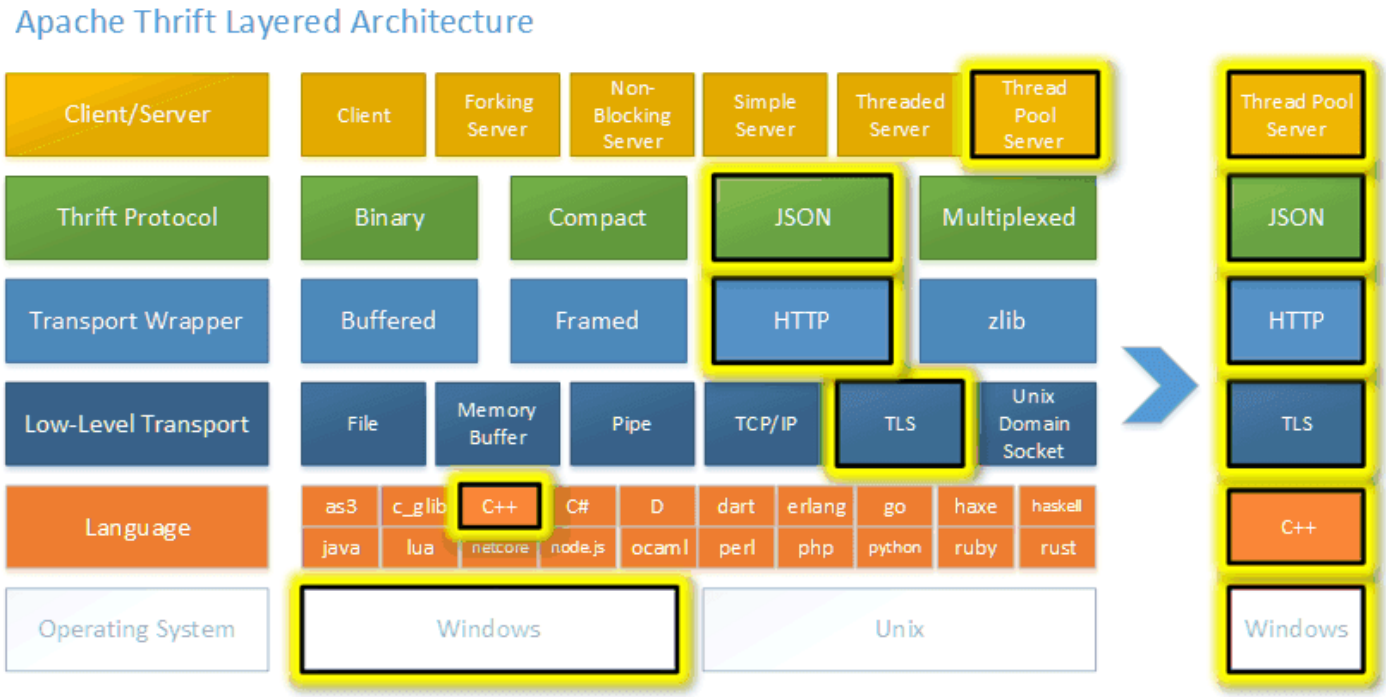
它的主要特性包括三个方面。

- 通信协议采用了 HTTP/2，因为 HTTP/2 提供了连接复用、双向流、服务器推送、请求优先级、首部压缩等机制，所以在通信过程中可以节省带宽、降低 TCP 连接次数、节省 CPU，尤其对于移动端应用来说，可以帮助延长电池寿命。
- IDL 使用了ProtoBuf，ProtoBuf 是由 Google 开发的一种数据序列化协议，它的压缩和传输效率极高，语法也简单，所以被广泛应用在数据存储和通信协议上。
- 多语言支持，能够基于多种语言自动生成对应语言的客户端和服务端的代码。

2. Thrift

再来看下 Thrift，Thrift 是一种轻量级的跨语言 RPC 通信方案，支持多达 25 种编程语言。为了支持多种语言，跟 gRPC 一样，Thrift 也有一套自己的接口定义语言 IDL，可以通过代码生成器，生

成各种编程语言的 Client 端和 Server 端的 SDK 代码，这样就保证了不同语言之间可以相互通信。它的架构图可以用下图来描述。



（图片来源：https://github.com/apache/thrift/raw/master/doc/images/thrift-layers.png）

从这张图上可以看出 Thrift RPC 框架的特性。

- 支持多种序列化格式：如 Binary、Compact、JSON、Multiplexed 等。
- 支持多种通信方式：如 Socket、Framed、File、Memory、zlib 等。
- 服务端支持多种处理方式：如 Simple 、Thread Pool、Non-Blocking 等。

3. 对比选型

那么涉及跨语言的服务调用场景，到底该选择 gRPC 还是 Thrift 呢？

从成熟度上来讲，Thrift 因为诞生的时间要早于 gRPC，所以使用的范围要高于 gRPC，在 HBase、Hadoop、Scribe、Cassandra 等许多开源组件中都得到了广泛地应用。而且 Thrift 支持多达 25 种语言，这要比 gRPC 支持的语言更多，所以如果遇到 gRPC 不支持的语言场景下，选择 Thrift 更合适。

但 gRPC 作为后起之秀，因为采用了 HTTP/2 作为通信协议、ProtoBuf 作为数据序列化格式，在移动端设备的应用以及对传输带宽比较敏感的场景下具有很大的优势，而且开发文档丰富，根据 ProtoBuf 文件生成的代码要比 Thrift 更简洁一些，从使用难易程度上更占优势，所以如果使用的语言平台 gRPC 支持的话，建议还是采用 gRPC 比较好。

总结

以上就是我对几种使用最广泛的开源 RPC 框架的选型建议，也是基于它们目前现状所作出的判断，从长远来看，支持多语言是 RPC 框架未来的发展趋势。正是基于此判断，各个 RPC 框架都提供了 Sidecar 组件来支持多语言平台之间的 RPC 调用。

- Dubbo 在去年年底又重启了维护，并且宣称要引入 Sidecar 组件来构建Dubbo Mesh提供多语言支持。
- Motan 也在去年对外开源了其内部的 Sidecar 组件：Motan-go，目前支持 PHP、Java 语言之间的相互调用。
- Spring Cloud 也提供了 Sidecar 组件spring-cloud-netflix-sideca，可以让其他语言也可以使用 Spring Cloud 的组件。

所以未来语言不会成为使用上面这几种 RPC 框架的约束，而 gRPC 和 Thrift 虽然支持跨语言的 RPC 调用，但是因为它们只提供了最基本的 RPC 框架功能，缺乏一系列配套的服务化组件和服务治理功能的支撑，所以使用它们作为跨语言调用的 RPC 框架，就需要自己考虑注册中心、熔断、限流、监控、分布式追踪等功能的实现，不过好在大多数功能都有开源实现，可以直接采用。

思考题

同样是支持跨语言的 RPC 调用，你觉得 gRPC 这类的跨语言服务框架和 Motan-go 这类的 Sidecar 方案有什么区别？在使用过程中都需要注意什么？

欢迎你在留言区写下自己的思考，与我一起讨论。

精选留言

Saily 1

tars不是介绍说支持C++、Java、PHP、NodeJS ，最近还发布了Go语言版本的

2018-09-22

大龄小学生 0

老师，rpc和mq的优势是什么？感觉rpc能做的mq都能做。

2018-09-23

gggwvg 0

Tars有完整的服务治理生态，支持多语言。

2018-09-22

everpan 0

tars支撑java php 最近还支持go了

2018-09-22

long.mr 0

胡老师，c++的，是不是也可以考虑下baidu rpc呢,性能在rpc里还是很强的哈~，对比的时候可以考虑下呀😄

2018-09-22

GS 0

大神见多识广

2018-09-22