RPC 框架及模拟实现 RPC 框架

一、什么是 RPC

RPC ，即 Remote Procedure Call ，在分布式计算中成为远程过程调用，也可称为远程过程调（面向过程语言）用或远程方法调用（面向对象语言）。一个通俗的描述是：客户端在不知道调用细节的情况下，调用存在于远程计算机上的某个对象，就像调用本地应用程序中的对象一样。比较正式的描述是：一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

RPC 是一种进程间通信的模式，程序分布在不同的地址空间里，如果在同一主机里，RPC 可以通过不同的模拟地址空间进行通讯。使用的架构为客户端/服务器(C/S)。

完整的 RPC 框架主要包括：客户端、服务端、注册中心三部分，客户端中有服务发现、调用模块、RPC 协议等组件，服务端中有服务暴露、处理程序、线程池、RPC 协议等组件。

二、RPC 核心功能

RPC 协议部分就是 RPC 的核心功能，其主要组成部分为：客户端、客户端 Stub 、网络传输模块、服务端 Stub 、服务端等。

客户端：服务调用方。

客户端 Stub：存放服务端地址信息，将客户端的请求参数数据信息打包成网络消息，在通过网络传输发送给服务端。

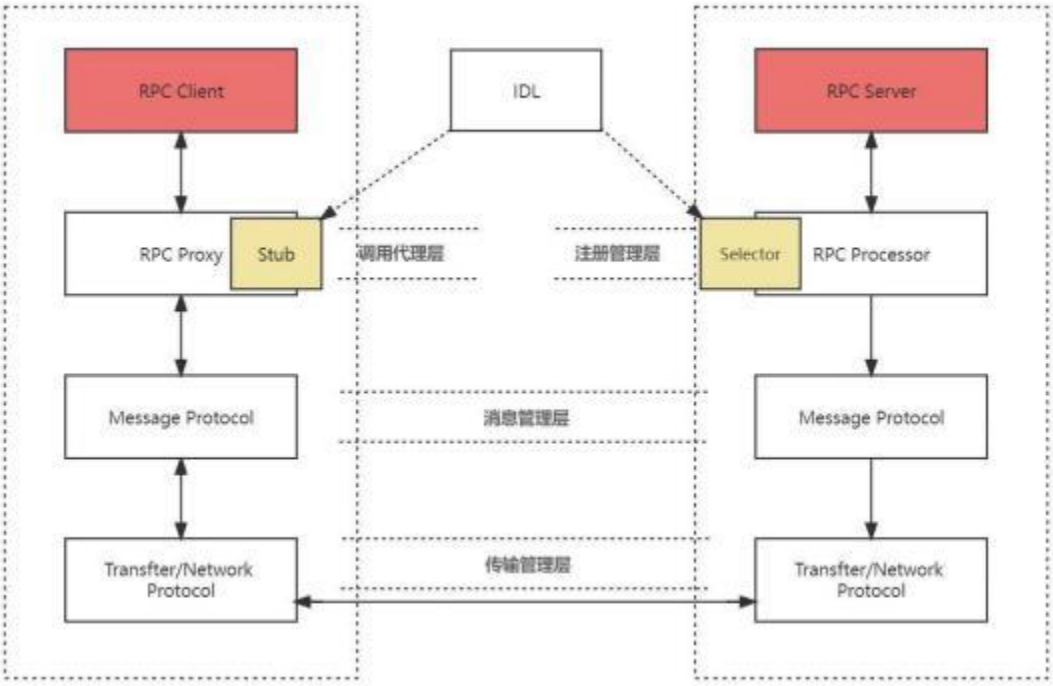
服务端 Stub：接收客户端发送过来的请求信息并进行解包，然后在调用本地服务进行处理。

网络传输模块：底层传输，可以是 TCP 或 HTTP。

服务端：服务提供方。

三、RPC 的主要组成部分

RPC 的协议，如下图所示：



Client：RPC 协议的调用方。最理想的情况是 RPC Client 在完全不知道有 RPC 框架存在的情况下发起对远程服务的调用。但实际情况来说 Client 或多或少的都需要指定 RPC 框架的一些细节。

Server：在 RPC 规范中，这个 Server 并不是提供 RPC 服务器 IP 、端口监听的模块。而是远程服务方法的具体实现（在 JAVA 中就是 RPC 服务接口的具体实现）。其中的代码是最普通的和业务相关的代码，甚至其接口实现类本身都不知道将被某一个 RPC 远程客户端调用。

Stub/Proxy：RPC 代理存在于客户端，因为要实现客户端对 RPC 框架“透明”调用，那么客户端不可能自行去管理消息格式、不可能自己去管理网络传输协议，也不可能自己去判断调用过程是否有异常。这一切工作在客户端都是交给 RPC 框架中的“代理”层来处理的。

Message Protocol：在上文我们已经说到，一次完整的 client-server 的交互肯定是携带某种两端都能识别的，共同约定的消息格式。RPC 的消息管理层专门对网络传输所承载的消息信息进行编码和解码操作。目前流行的技术趋势是不同的 RPC 实现，为了加强自身框架的效率都有一套（或者几套）私有的消息格式。

Transfer/Network Protocol：传输协议层负责管理 RPC 框架所使用的网络协议、网络 IO 模型。例如 Hessian 的传输协议基于 HTTP（应用层协议）；而 Thrift 的传输协议基于 TCP（传输层协议）。传输层还需要统一 RPC 客户端和RPC 服务端所使用的 IO 模型；

Selector/Processor：存在于 RPC 服务端，用于服务器端某一个 RPC 接口的

实现的特性（它并不知道自己是一个将要被 RPC 提供给第三方系统调用的服务）。所以在 RPC 框架中应该有一种“负责执行 RPC 接口实现”的角色。包括：管理 RPC 接口的注册、判断客户端的请求权限、控制接口实现类的执行在内的各种工作。

IDL：实际上 IDL（接口定义语言）并不是 RPC 实现中所必须的。但是需要跨语言的 RPC 框架一定会有 IDL 部分的存在。这是因为要找到一个各种语言能够理解的消息结构、接口定义的描述形式。如果您的 RPC 实现没有考虑跨语言性，那么IDL 部分就不需要包括，例如 JAVA RMI 因为就是为了在 JAVA 语言间进行使用，所以 JAVA RMI 就没有相应的 IDL。

在物理服务器性能相同的情况下，以下几个因素会对一款 RPC 框架的性能产生直接影响：

使用的网络 IO 模型：RPC 服务器可以只支持传统的阻塞式同步 IO，也可以做一些改进让 RPC 服务器支持非阻塞式同步 IO，或者在服务器上实现对多路 IO 模型的支持。这样的 RPC 服务器的性能在高并发状态下，会有很大的差别。特别是单位处理性能下对内存、CPU 资源的使用率。

基于的网络协议：一般来说您可以选择让您的 RPC 使用应用层协议，例如 HTTP 或者 HTTP/2 协议，或者使用TCP 协议，让您的 RPC 框架工作在传输层。工作在哪一层网络上会对 RPC 框架的工作性能产生一定的影响，但是对 RPC 最终的性能影响并不大。但是至少从各种主流的 RPC 实现来看，没有采用UDP 协议做为主要的传输协议的。

消息封装格式：选择或者定义一种消息格式的封装，要考虑的问题包括：消息的易读性、描述单位内容时的消息体大小、编码难度、解码难度、解决半包/粘包问题的难易度。当然如果您只是想定义一种 RPC 专用的消息格式，那么消息的易读性可能不是最需要考虑的。消息封装格式的设计是目前各种 RPC 框架性能差异的最重要原因，这就是为什么几乎所有主流的 RPC 框架都会设计私有的消息封装格式的原因。dubbo 中消息体数据包含 dubbo 版本号、接口名称、接口版本、方法名称、参数类型列表、参数、附加信息

Schema 和序列化（Schema & Data Serialization）：序列化和反序列化，是对象到二进制数据的转换，程序是可以理解对象的，对象一般含有 schema 或者结构，基于这些语义来做特定的业务逻辑处理。考察一个序列化框架一般会关注以下几点：

Encoding format，是 human readable（是否能直观看懂 json） 还是 binary(二进制)。

Schema declaration，也叫作契约声明，基于 IDL，比如 Protocol

Buffers/Thrift，还是自描述的，比如 JSON、XML。另外还需要看是否是强类型的。

语言平台的中立性，比如 Java 的 Native Serialization 就只能自己玩，而 Protocol Buffers 可以跨各种语言和平台。

新老契约的兼容性，比如 IDL 加了一个字段，老数据是否还可以反序列化成功。

和压缩算法的契合度，跑 benchmark (基准)和实际应用都会结合各种压缩算法，例如：gzip、snappy。

性能，这是最重要的，序列化、反序列化的时间，序列化后数据的字节大小是考察重点。

序列化方式非常多，常见的有 Protocol Buffers， Avro，Thrift，XML，JSON，MessagePack，Kyro，Hessian，Protostuff，Java Native Serialize，FST 。

实现的服务处理管理方式：在高并发请求下，如何管理注册的服务也是一个性能影响点。您可以让 RPC 的 Selector/Processor 使用单个线程运行服务的具体实现（这意味着上一个客户端的请求没有处理完，下一个客户端的请求就需要等待）、您也可以为每一个 RPC 具体服务的实现开启一个独立的线程运行（可以一次处理多个请求，但是操作系统对于“可运行的最大线程数”是有限制的）、也可以线程池来运行 RPC 具体的服务实现（ 目前看来，在单个服务节点的情况下，这种方式是比较好的）、还可以通过注册代理的方式让多个服务节点来运行具体的 RPC 服务实现。

四、RPC 相关主流框架

4.1 国内RPC 框架

Dubbo，来自阿里巴巴 http://dubbo.io/

Dubbo：是阿里集团开源的一个极为出名的 RPC 框架，在很多互联网公司和企业应用中广泛使用。协议和序列化框架都可以插拔是极其鲜明的特色。

Motan，新浪微博自用 https://github.com/weibocom/motan

Dubbox，当当基于 dubbo 的 https://github.com/dangdangdotcom/dubbox

Rpcx，基于 Golang 的 https://github.com/smallnest/rpcx

4.2 国外 RPC 框架

Avro from hadoop https://avro.apache.org

Finagle by twitter https://twitter.github.io/finagle

Hessian from cuacho http://hessian.caucho.com

Coral Service inside amazon (not open sourced)

gRPC：是 Google 公布的开源软件，基于 HTTP 2.0 协议，并支持常见的众多编程语言。RPC 框架是基于 HTTP 协议实现的，底层使用到了 Netty 框架的支持。

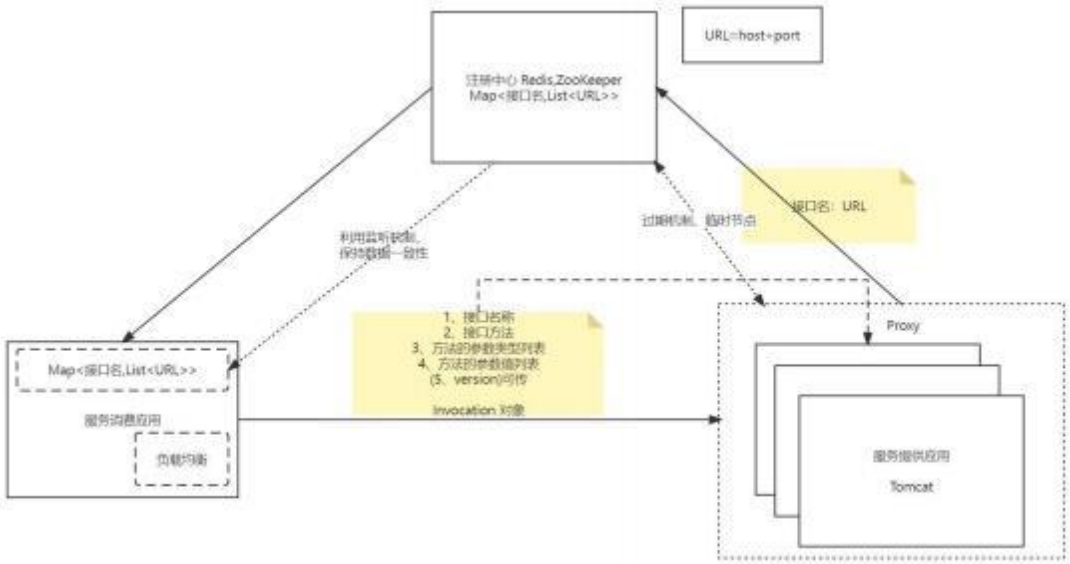
Thrift 是 Facebook 的开源 RPC 框架，主要是一个跨语言的服务开发框架。

用户只要在其之上进行二次开发就行，应用对于底层的 RPC 通讯等都是透明的。不过这个对于用户来说需要学习特定领域语言这个特性，还是有一定成本的。

五、模拟实现 RPC

5.1 模拟实现 RPC 流程图

模拟实现 RCP 流程图，如下图所示：



5.2 模拟实现 RPC 框架展示

演示 RPC 源码地址：模拟实现 RPC 框架