# gRPC入门

<https://www.cnblogs.com/takumicx/p/10059448.html>

Todo 程序待调通,已经在Idea中建工程。

[https://developers.google.com/protocol-buffers/](https://developers.google.com/protocol-buffers/" \t "https://www.cnblogs.com/takumicx/p/_blank)  
[https://grpc.io/docs/guides/concepts.html#service-definition](https://grpc.io/docs/guides/concepts.html" \l "service-definition" \t "https://www.cnblogs.com/takumicx/p/_blank)

实战

<https://www.jianshu.com/p/6c0fe5aa2f66>

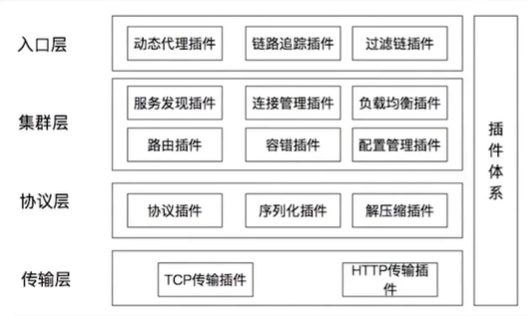
<https://www.misscoconut.me/2020/07/27/gRPC%20%E7%AC%94%E8%AE%B0/>

协议分析

<https://blog.csdn.net/hugo_lei/article/details/106098217>

### **1.2 RPC 框架**

RPC 抽象来看可以归纳为两方面问题：****1.调用方与提供方协议约定问题；**** ****2.网络传输问题****，解决这两块基本问题后，还得需要工程落地，解决服务发现、负载均衡、限流熔断等等问题，于是有了 RPC 框架的出现。



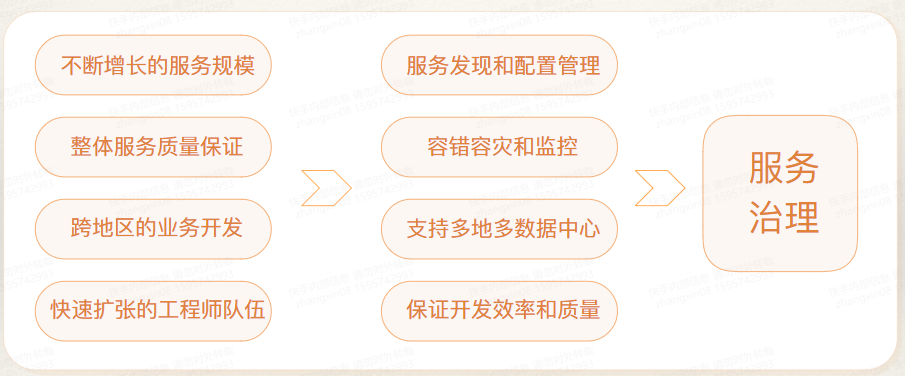
​

上图是一个典型的 RPC 框架的基本架构，主要分成四块：

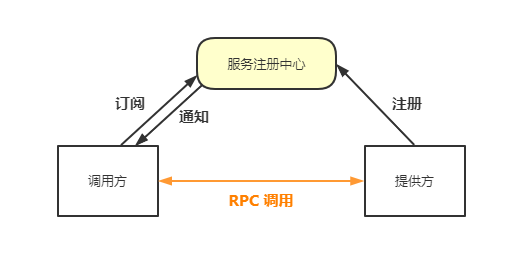
1. 入口层：动态代理机制在服务提供方包装好服务的接口，暴露给调用方；对调用方提供代理对象，屏蔽其感知远程过程调用；
2. 集群层：解决分布式场景下的服务发现、节点负载均衡、容错、路由管理等等问题；
3. 协议层: 约定调用方与提供方数据包的格式，提供反序列化、解压缩等功能支持；
4. 网络传输: 提供 TCP 长链接传输、HTTP 通信等功能。

#### **服务治理（SOA）**

服务治理解决什么问题？



* 服务发现机制



类似 DNS 的机制，服务提供方向注册中心进行注册，调用方订阅服注册中心；注册中心拿到当前提供方服务的地址后推送给调用方。

### **1.3 RPC vs HTTP Service**

相比于 HTTP 服务，RPC 具有以下优势：

* 网络传输方面：基于 TCP 长链接，省去了 HTTP 连接建立过程中的性能损耗，提高传输效率；
* 传输消息方面：传输的消息序列化成二进制对象，数据包体积减小，节省带宽；
* 开发者友好：RPC 牺牲了消息的可读性来提高易用性，对于调用方开发者只需要调用本地接口即可调用远程方法；
* 面向服务的封装：成熟的 RPC 框架封装了服务发现、负载均衡、熔断降级等等面向服务的高级特性，单纯使用 HTTP 服务调用则少了这些特性。

同时，也存在一些劣势：

* 数据包可读性差，提高团队成员学习成本；
* 交互性单一：只能调用方主动调用服务端方法，在需要服务端下发通知等场景下不适用。

如何选择 RPC 或 HTTP 服务？

1. 公司内部给其他业务提供接口时，优先考虑 RPC 服务，使用方为公司外部的服务，优先考虑 HTTP 服务；
2. 对于消耗特定资源的（例如 GPU/CPU/大内存资源），有特殊依赖的（比如运行环境需要依赖外部的 ffmpeg），集中实现效率比较高的（比如某个业务集中到几台机器，可以使用大内存缓存等）,选 RPC；
3. 一些经常变化的业务逻辑，依赖很广泛，又需要实现可以收敛且能随时更新，选择 RPC。

### **gRPC 特性**

优势

支持多种语言；

基于 IDL 文件定义服务，通过 proto3 工具生成指定语言的数据结构、服务端接口以及客户端 Stub；

网络通信依赖 Netty 实现的 HTTP/2 协议封装，支持双向流、消息头压缩、单 TCP 的多路复用、服务端推送等特性，这些特性使得 gRPC 在移动端设备上更加省电和节省网络流量；

序列化支持 PB（Protocol Buffer）和 JSON，PB 是一种语言无关的高性能序列化框架，基于 HTTP/2 + PB, 保障了 RPC 调用的高性能。

缺点

* + 服务治理相关能力缺失，负载均衡和服务发现等功能需要开发者扩展实现。