架构概述

2022-5-18 15:59

1 前言

与存量系统互通, 保持框架的开放性和可扩展性是tRPC的核心诉求. 为了实现此目标, tRPC的框架设计遵循了插件化架构设计理念. 本文首先会简单介绍什么是插件化架构. 然后会从tRPC的总体架构, 分层模型, 性能设计三个方面来介绍tRPC的架构设计. 通过本文的介绍, 希望大家能对tRPC的核心概念, tRPC包处理流程, 系统设计理念有一个初步的认识, 为tRPC服务开发, 插件使用和开发, 协议开发,以及了解拦截器的开发和使用做必要的知识储备.

2 插件化架构

`插件化架构` 在后面的描述中会被频繁提到,是tRPC架构设计的核心概念之一, 所以在介绍tRPC的总体架构之前, 我们先谈谈什么是插件化架构. 要理解插件化架构, 我们可以把它分为 "基于接口机制的插件工厂"和"基于AOP思想的拦截器"两部分来介绍.

2.1 插件工厂

tRPC核心框架是采用基于接口编程的思想, 通过把框架功能抽象成一系列的插件组件, 注册到插件工厂, 并由插件工厂实例化插件. tRPC框架负责串联这些插件组件, 拼装出完整的框架功能. 我们可以把插件模型分为以下三层:

1. 框架设计层: 框架只定义标准接口，没有任何插件实现，与平台完全解耦；
2. 插件实现层: 将插件按框架标准接口封装起来即可实现框架插件；
3. 用户使用层: 业务开发只需要引入自己需要的插件即可，按需索取，拿来即用。

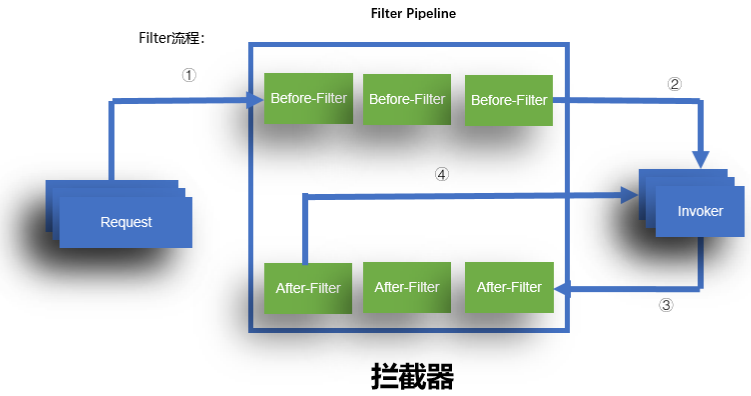
2.2 拦截器

为了使框架有更强的可扩展性, tRPC引入了拦截器这个概念,它借鉴了java面向切面(AOP)的编程思想. 具体做法是通过在框架指定的动作前后设置埋点, 然后通过在埋点地方插入一系列的filter, 来实现将系统功能引入到框架中.

面向切面编程（AOP）是一种编程范式，主要用于处理系统中分布于各个模块的横切逻辑，常用来解决模调监控，横切日志，链路跟踪，过载保护，预处理等这些跟接口请求相关的问题。要实现AOP，首先我们需要把框架的功能模块抽象为两个部分：核心关注点和横切关注点。

每次业务调用，均会有相应的业务逻辑被执行，这部分就是核心关注点。但除此之外，在每次调用时，还有诸多例行性、常规化要做的处理，比如需要将用户数据进行预处理、编解码、传输等，这些都是横切关注点，而他们往往与业务逻辑并不直接相关。

场景拦截器的常见实现有递归调用，链式回调， 数组遍历等方式。 tRPC-Cpp使用的是数组遍历，tRPC-Go使用的是递归调用，具体实现请参考各语言的拦截器介绍文档。虽然具体实现方式不同，但是达到的效果是在RPC调用前后放置拦截器。

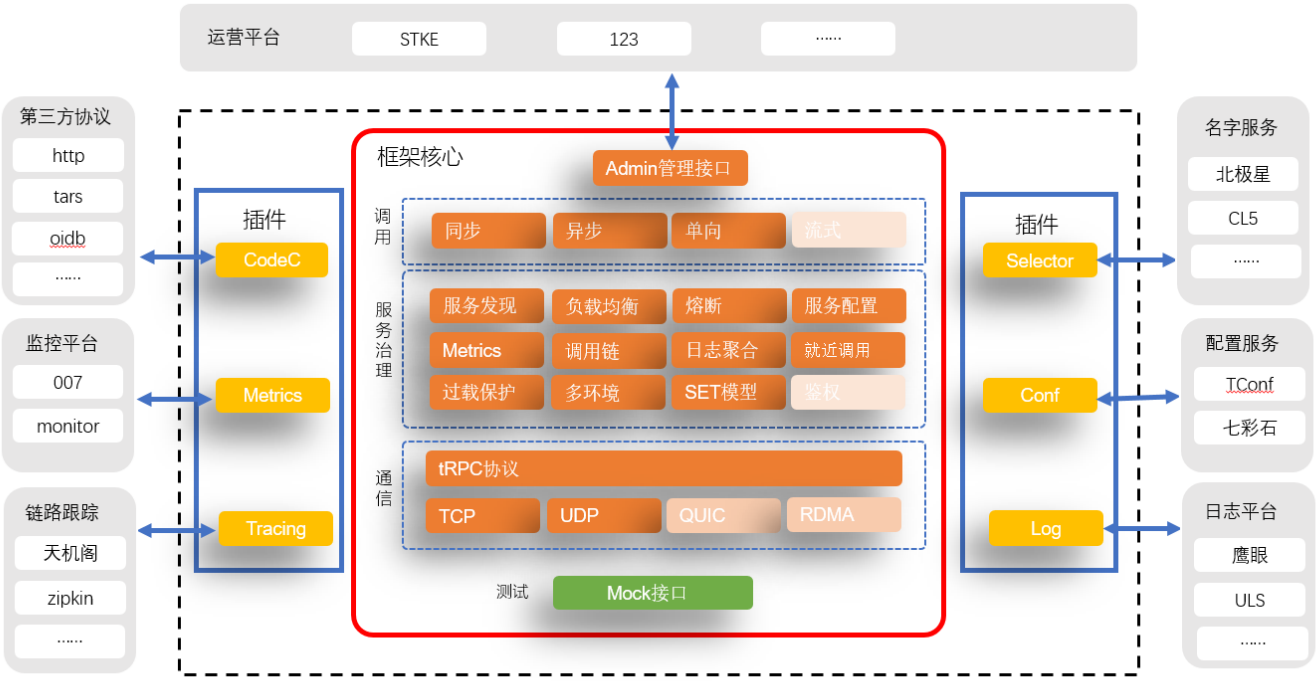


拦截器

拦截器的最终目的是通过对业务逻辑与系统级服务的解耦, 允许各自进行内聚性开发. 通过filter预设埋点, 为程序提供了一种在不修改源代码的情况下给程序动态添加或替换功能的机制.

3 总体架构

tRPC的总体架构由"框架核心"和"插件"两部分组成. 如图所示, 虚线框内为tRPC, 其中中间的红色实线框为框架核心, 蓝色框为插件部分.



3.1 框架核心

tRPC框架核心由以下三部分组成:

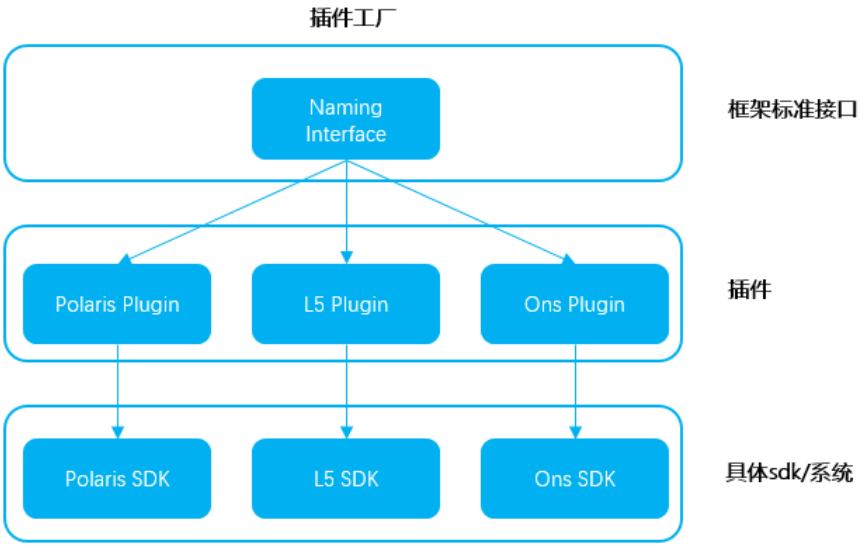
- 通信层: 负责数据的传输和协议的编解码. 框架内置支持tcp/udp/quic等通信协议, 传输协议采用基于PB的tRPC协议来承载RPC调用, 支持通过codec插件来使用其它传输协议
- 服务治理层: 负责将服务治理功能抽象成插件组件, 采用插件化架构, 通过调用插件和外部服务治理系统完成对接, 实现服务发现，负载均衡，监控，调用链等服务治理功能
- 调用层: 封装服务和服务代理实体, 提供RPC调用接口, 支持业务用同步,异步,单向以及流式调用等方式进行服务间调用

此外框架核心还提供了admin管理接口. 运营平台可以通过调用admin接口对服务进行管理. 管理接口包括更新配置，查看版本, 修改日志级别等功能. 同时框架也支持用户自定义管理接口,以满足业务定制化需求.

tRPC提供脚手架工具，用来生成服务间调用的Mock接口桩代码, 业务可以通过它对服务调用进行MOCK,方便单元测试.

3.2 插件

插件是框架核心和外部服务治理组件串联起来的桥梁. 插件一边需要按框架标准接口实现插件, 注册到框架核心, 并完成插件实例化；另一边插件需要调用外部服务治理服务的SDK/API, 实现如服务发现，负载均衡，监控，调用链等服务治理功能.下图是一个典型的名字系统实现方式:



tRPC框架对插件按功能大致分为下面几个组件:

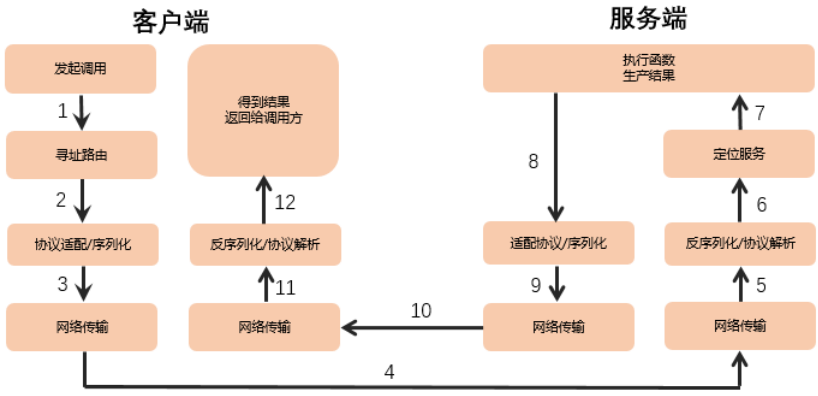
- Selector: 提供服务发现（selector）、负载均衡（loadbalance）、熔断（circuitbreaker）标准接口. 插件通过实现这些接口来实现基于服务名的寻址功能
- Conf: 提供获取配置的标准接口. 通过抽象出"数据源"概念, 来实现从本地配置文件、配置中心等不同数据源读取配置. 通过插件来扩展对各种数据格式(json,yaml, toml等)的支持. Conf组件也提供了watch机制,来实现配置的动态更新
- Log: 提供了统一的日志打印和日志上报接口. 日志插件通过实现日志上报接口来完成和远程日志系统的对接
- Metrics: 提供监控指标的上报接口, 支持常见的单维上报，如counter、gauge等，也支持多维上报. 插件通过实现

上报接口来完成和监控平台的对接

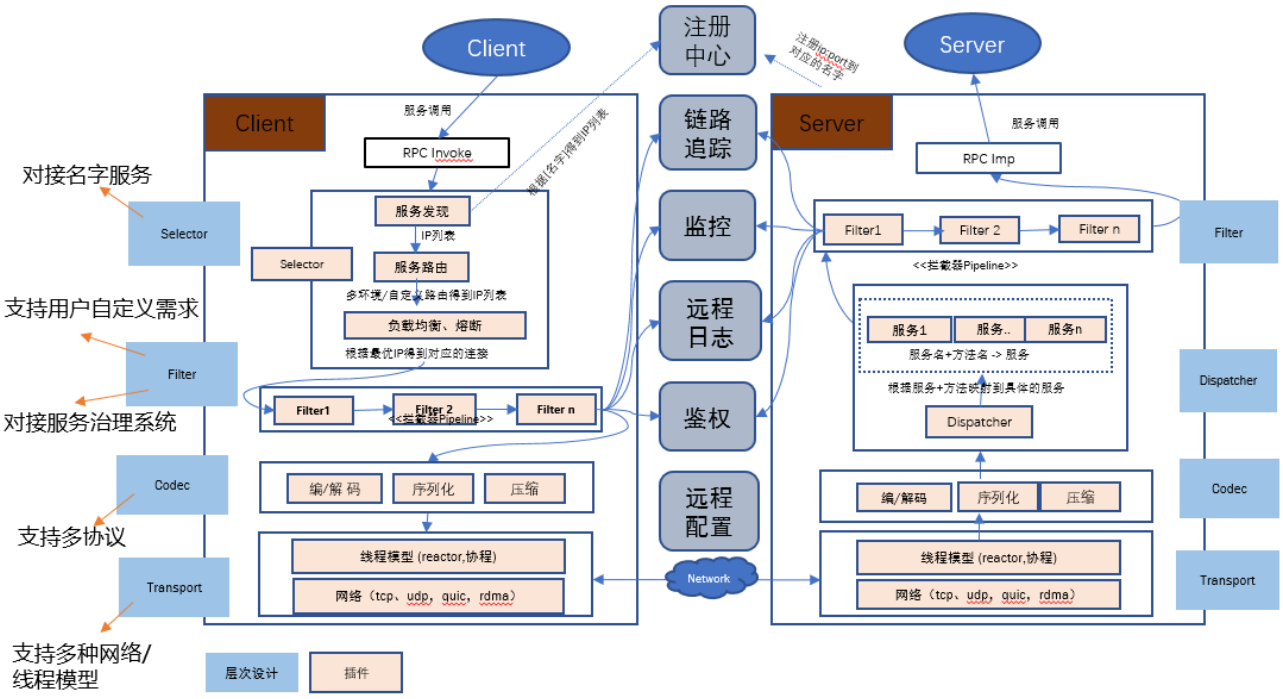
- Tracing: 调用链跟踪插件采用filter(拦截器)的机制来实现和调用链跟踪平台对接.
- Codec: 提供协议编解码相关的接口，允许通过插件的方式来扩展业务协议、序列化方式、数据压缩方式等协议处理

4 分层设计

rpc调用从开发者角度来说, 它可以让你像本地函数调用一样进行跨节点的函数调用. 从微服务架构来说，rpc是微服务之间的一种通信方式。通过rpc，一个微服务可以向远程计算机请求另一个服务，而不需要了解底层网络使用了什么协议. 无论rpc使用什么样的协议进行数据传输, 通常一个完整的RPC过程, 都可以用下面这张图来描述:



图中描述步骤是一个RPC调用必须要经过的环节. 基于此共性, 我们把框架分为服务端和客户端, 分别在纵向进行分层. 服务端分为Transport, Codec, Dispatcher和Filter层. 客户端分为Transport, Codec. Filter和Selector层, tRPC框架分层结构如图所示:



在这张图中, 我们新增了一层filter层(拦截器), 其主要目的是采用AOP的思想, 把服务治理的大部分功能(比如监控指标上报, 调用链跟踪, 远程日志, 鉴权等)以横切关注点的方式,插入到报文处理的流程中. 正如我们在插件化架构章节所讲的, 此设计大大的增强了系统的可扩展性.

系统对每一层进行模块化拆分, 各个核心模块都使用了插件化的实现. 通过基于接口编程的思想,串联rpc调用的全流程. 对于一些模块,框架也采用了更细粒度的模块拆分. 比如selector模块被细分为服务发现, 服务路由,负载均衡和熔断子模块. Codec层也被细分为编解码, 序列化和压缩三个子模块.

正是通过分层设计, 细粒度的模块拆分和插件化的实现, 使框架具备很强的扩展性和开放性. 业务可以灵活替换插件,实现与不同系统的对接, 也可以进行业务个性化定制.

5 OWNER

jesuzhang