tRPC介绍

2022-5-18 15:56

1 背景

tRPC是一套全新的远程过程调用的开发框架。可能大家会问，在公司内部和开源社区都有很多优秀和成熟的RPC开发框架, 为什么还需要重新构建一套新的RPC框架呢? tRPC产生的动机和背景是什么？我们通过下面两个问题来了解tRPC产生的背景.

1.1 为什么需要统一的RPC框架

为什么需要一个统一的RPC框架? 这是基于公司现状考虑的, 当前公司存量框架众多, 且基本上都存在:

- 框架插件化能力有限, 对于协同不够友好;
- 框架覆盖的语言类型有限, 无法覆盖主流的开发语言;
- 名字服务和协议不能互通, 甚至同一套框架下的变种都不能互通;
- 与devops工具链的对接存在问题, 处于年久失修状态.

公司各个业务使用的开发框架也呈现出一种非常杂乱的状态,有的业务甚至有同时使用几种不同开发框架的情况。这就给开发和运维在服务互通和服务治理上带来了很大的学习和维护成本,导致开发和运维效率都比较低。这就需要有一套RPC框架，能整合公司后台人力和历史研发、运营经验, 将存量框架收敛到一套，实现能与存量框架良好衔接，稳定并持续演进和运营。

1.2 为什么需要自研

为什么我们需要自研一套新的RPC框架, 而不是基于公司内部或者开源成熟框架去打造? 这其实是一个技术选型问题. tRPC的技术选型大致可以总结为以下几点:

- 多语言: 因为公司技术团使用的开发语言是多样的，需要开发框架必须支持多语言
- 能提供开发框架和服务治理能力
- 架构的开放性, 方便和存量系统对接: 包括协议对接和服务治理系统的对接
- 性能和成熟度

首先基于公司对多语言的需求，我重点对能支持多语言的 Istio，gRPC和公司内部的TAF微服框架进行了研究和分析。

- gRPC 是Google开发的一款优秀的开源RPC框架，它主要面向移动应用开发并基于HTTP/2协议标准而设计。它基于ProtoBuf序列化协议开发，支持多语言和流式通信。但是对于内网服务间通信来讲，我们并不需要HTTP/2这种复杂协议，HTTP/2对性能也有一定影响，而且 gRPC 对于多协议的支持改造难度偏大，修改后很难回归社区

- Istio是当前最热门的Service Mesh微服务架构解决方案，它可以作为服务间通信的基础设施层。其最大的特色是作为“边车”运行，对业务程序零侵入，大大降低了应用程序获取微服务治理能力的门槛。但是Istio没有提供开发框

架，其架构设计在性能上也有缺陷

- TAF是公司内部提供的一套于开发框架、服务治理和运营平台为一体化的总体解决方案，使用多年，系统成熟稳定。但是正是其一体化的设计，导致其很难和其它体系做对接，业务使用不够灵活，迁移成本偏大

综合上面的思考，所以我们认为采用自研的方式可控性更强, 框架设计也更能贴近业务需求. 从人性上讲，大家抛弃旧的团团罐罐，重新合力打造一套新的RPC框架更有凝聚力. 同时我们在开发框架上也有多年的积累，有能力去自研。基于这些综合因素，我们最终决定自研tRPC.

2 设计原则

tRPC设计理念是打造一个插件化架构，支持多语言、高性能、新老框架易互通，并便于业务测试的RPC框架, 其设计原则遵循以下原则:

- 使用简单：业务开发基于框架进行服务开发和服务构建应该简单方便
- 互通：框架设计和实现上应能解决与存量业务服务的互通；
- 通用并且高性能：框架应该适用于绝大多数用例场景，相比针对特定用例的框架，框架只会牺牲一点性能；
- 分层和模块化：框架整体架构应该按功能和特性进行分层，各个核心模块最好能够独立演进；
- 可拔插：对于名字路由、配置管理、监控上报、日志收集、调用跟踪、鉴权、心跳上报等，框架设计和实现上应该提供扩展点，以允许插入这些特性和默认实现；
- 可测试性：框架的设计和实现因考虑方便编写测试用例，以便框架的代码接入ci，进行持续集成，以保证框架的代码质量；

3 什么是tRPC

tRPC是由全公司OTeam共建的一款高性能，多语言，插件化，平台无关，开源协同，对接存量，面向未来，致力统一的远程过程调用开发框架。你可以使用tRPC：

- 内置了统一tRPC通信协议, 通过与多语言特性的结合, 为复杂产品形态下的多技术栈提供了通用解决方案
- 快速构建的支持多协议服务, 协议包括通用协议和公司内部私有协议
- 快速搭建流式服务，实现push，文件上传，消息下发等流式模型
- 通过tRPC插件生态, 业务自己选择服务治理体系, 实现和存量系统对接
- 基于框架接口进行二次开发, 包括: 通信协议, 插件, 存储接口, 拦截器等定制开发
- 通过tRPC测试工具和框架自动生成mock代码, 使服务的测试更简单, 更便捷

tRPC提供了6种开发语言(C++, Golang, Java, Python, Nodejs, Rust)的开发框架, 各个语言在设计原则和基本功能上进行了对齐, 同时各语言也与自身的语言生态紧密结合, 比如tRPC-Java实现了和SpringCloud, Springboot等开源生态的对接. tRPC-Python打通了tensorflow, pytorch, sklearn等机器学习生态组件, 整合AI全流程, 实现模型服务的快速服务化.

4 如何用好tRPC

微服务架构的实施和落地是需要一整套微服务运营和管理体系来保障的.tRPC在整个体系中的定位是一个能和微服务治理系统灵活对接的微服务框架. 我们推荐使用tRPC的最佳组合为: tRPC框架 + tRPC工具集 + 插件 + 微服务运营平台.

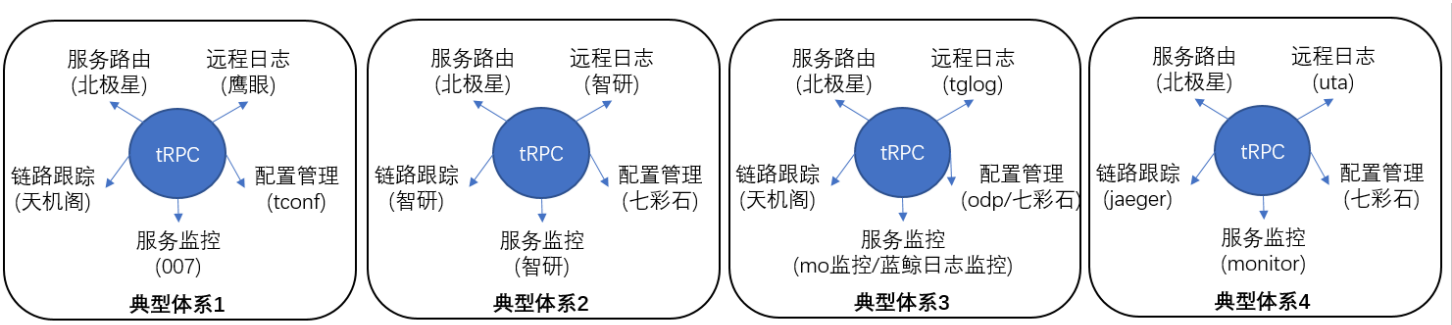
- tRPC框架: 一方面为开发人员提供开发框架, 屏蔽平台细节, 聚焦业务; 另一方面提供和微服务治理系统对接的能力, 简化平台对接. 同时还为开发人员提供二次开发的能力, 以满足业务个性化定制需求;
- tRPC工具集: 为开发人员提供编程开发脚手架和测试工具, 提高开发效率
- 插件: 包括服务治理插件和通信协议插件. tRPC社区提供了大量的插件, 已经支持了公司20多种存量协议，对接了

PCG/TEG/IEG/CSIG的各种服务治理系统，让业务迁移到trpc非常方便.

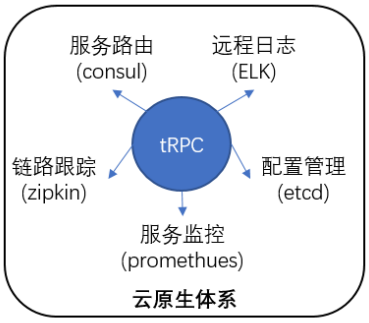
- 微服务运营平台: 负责DevOps全流程对接, 负责微服务的编排,调度,配置管理等运营管理

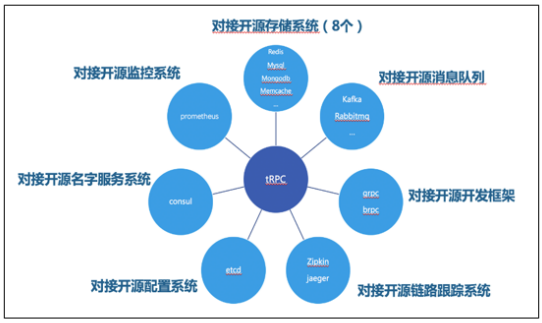
4.1 与服务治理体系的对接

正是基于tRPC的插件化设计, 决定了tRPC是平台无关的, 这也意味着tRPC能方便的和各生态系统实现对接, 完美融合. 下图是tRPC与公司内部运营系统对接的典型案例.



tRPC本身就是面向云原生的微服务架构,在设计之初，就注重其框架的开放性，tRPC通过插件化的设计，提供了快速对接外部开源生态系统的能力，目前已经对接了开源各种服务治理的生态系统，比如consul/promethus/zipkin/kafka等，其中对接的开源存储系统达到8个之多，基本实现对业界主流存储系统的覆盖，很好的解决了和开源社区组件对接的痛点问题.





5 OWNER

joeyzhong