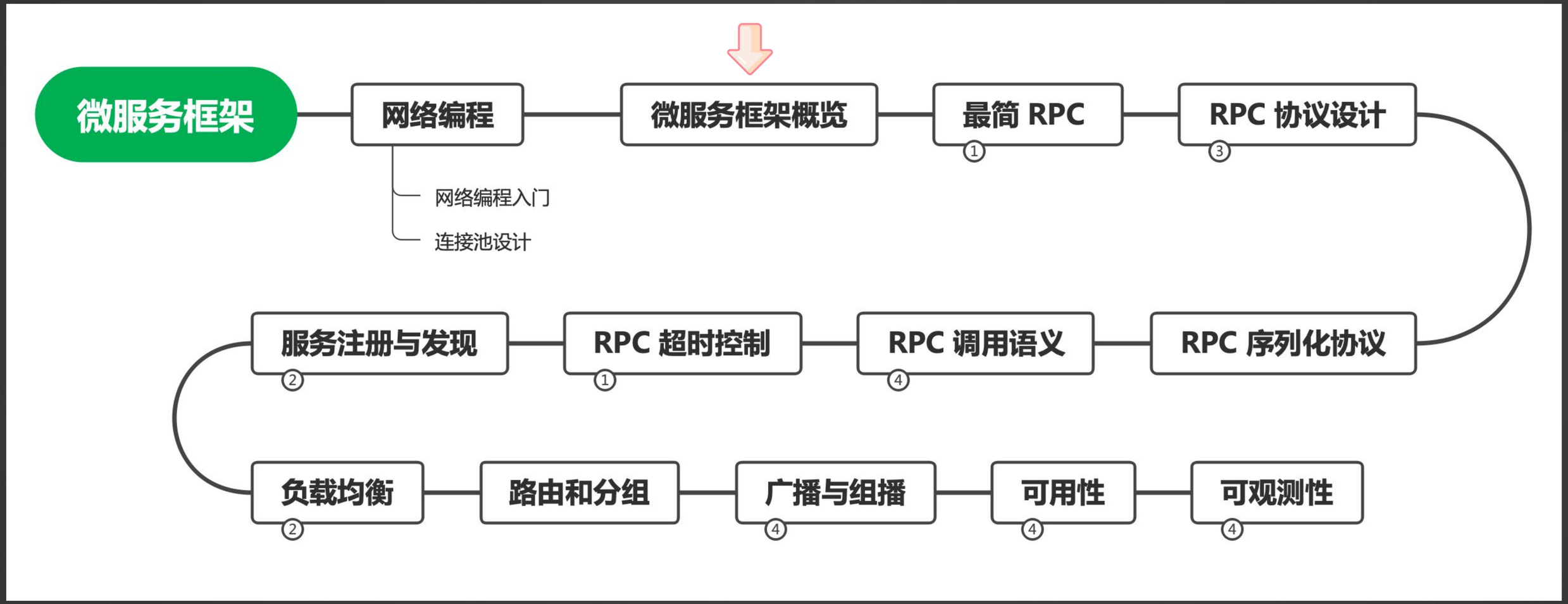
微服务框架——概览

大明

学习路线


微服务框架
网络编程
网络编程入门
连接池设计
微服务框架概览
最简 RPC
1
RPC 协议设计
3
RPC 序列化协议
RPC 调用语义
4
RPC 超时控制
1
服务注册与发现
2
负载均衡
2
路由和分组
广播与组播
4
可用性
4
可观测性
4

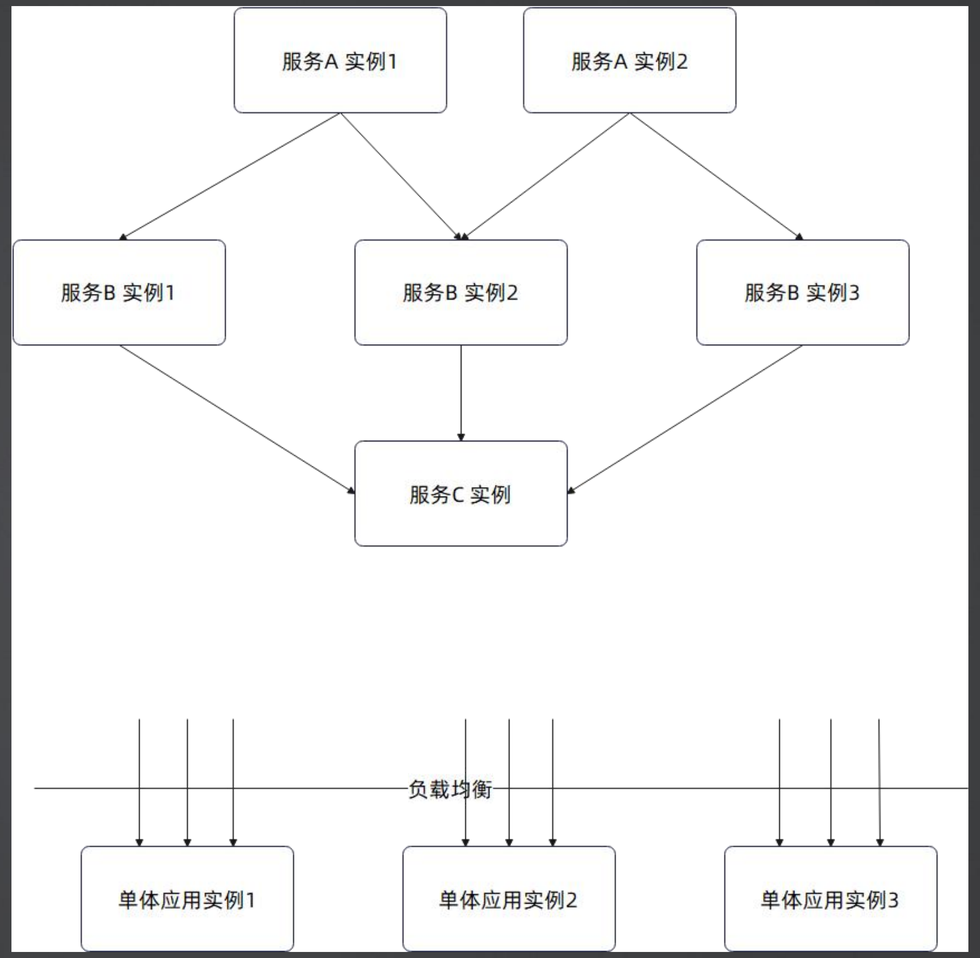
微服务架构

微服务框架是服务于微服务架构的。

微服务架构简单来说，就是指整个系统由多个组件组成，每一个组件都独立管理，组件之间通过网络来通信。

注意：单体应用可以部署多个实例，但是它依旧是单体应用。

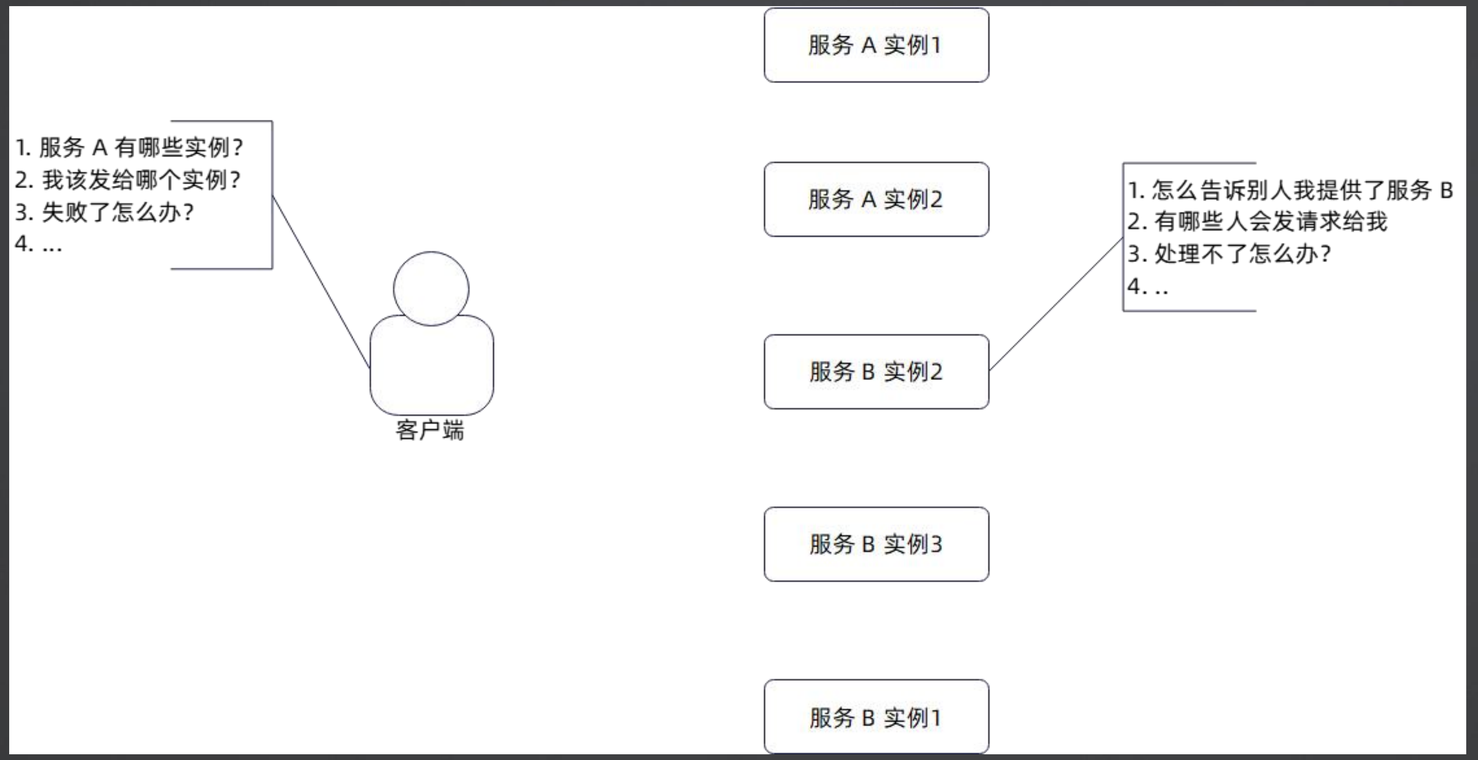
所以：一切问题都可以源于网络间通信。



单体应用即便部署了多个实例，但是不同的实例之间是不会有交互的。

微服务框架

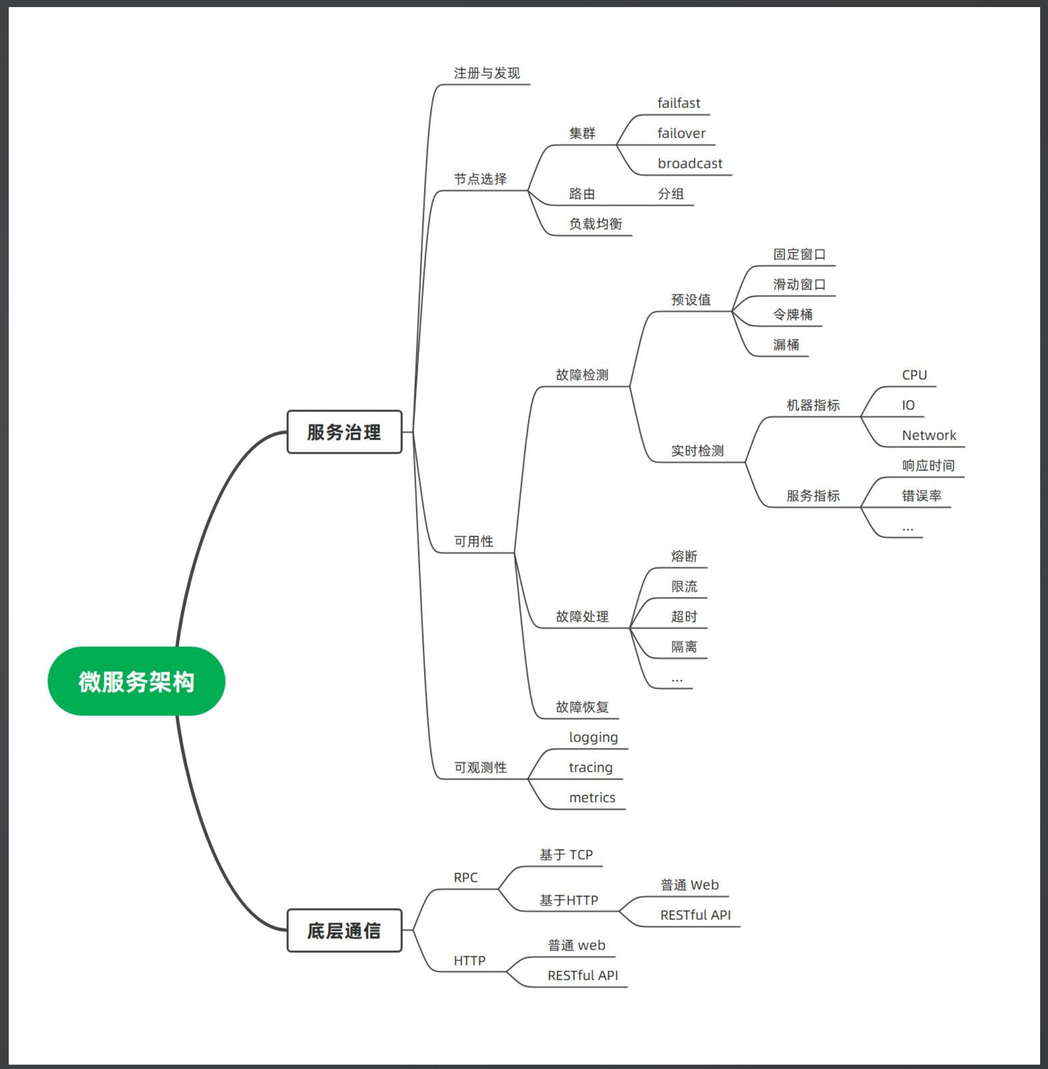
微服务框架就是要解决这种架构下，组件之间的发现、通信、容错等问题。



微服务框架主要问题

两个核心部分，通信 + 服务治理：

- 通信：即服务之间如何发起调用，一般就是 RPC，或者是 HTTP 直接通信
- 服务治理：涵盖从服务注册与发现到可观测性的全部内容



主要微服务框架类型

微服务框架又可以进一步分成：

- 纯粹的 RPC 框架：这一类框架的代表是早期的 gRPC，gRPC 发展到现在，也可以认为是定义了服务治理相关的接口，并且提供了一些默认实现。
- 服务治理框架：它们没有设计自己的微服务协议，比如说直接依赖于 gRPC 或者 HTTP 的。这一类框架专注在处理服务治理的问题，甚至于部分框架可能只专注于解决服务治理中的某一个方面。
- 大一统的微服务框架：这一类框架既有自己的通信协议，又有服务治理模块，典型的例子就是 Dubbo。早期的微服务框架比较多这种。

gRPC 简介

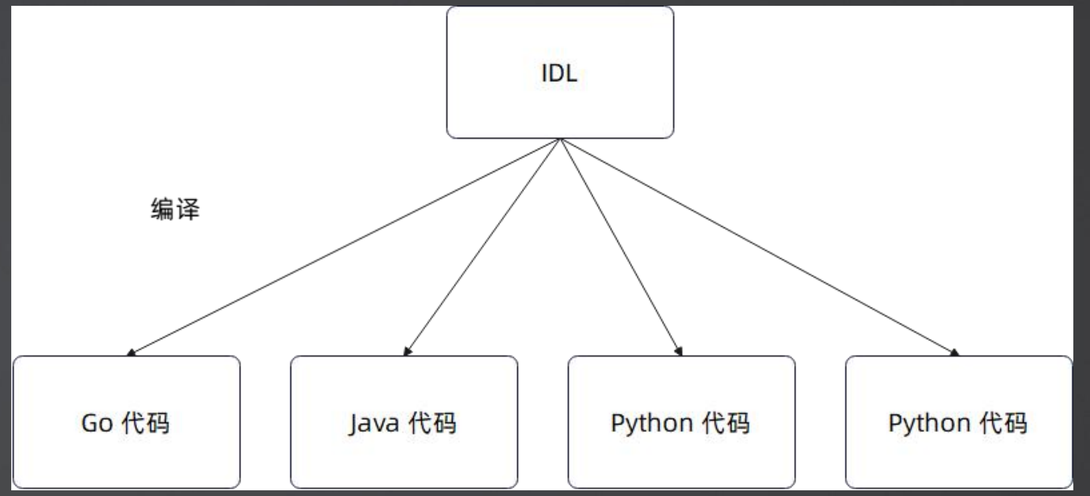
在底层通信协议上，我的建议遇事不决用 gRPC。如果是小型系统，那么可以考虑直接使用 HTTP 接口。

gRPC 比较学院派，它是典型的使用 IDL 来生成代码的 RPC 框架。

IDL（interface description/definition language）：接口描述语言/接口定义语言。

是指用一种中间语言来定义接口，而后为其它的语言生成对应代码的设计方案。

所以 gRPC 是多语言通信的首选。

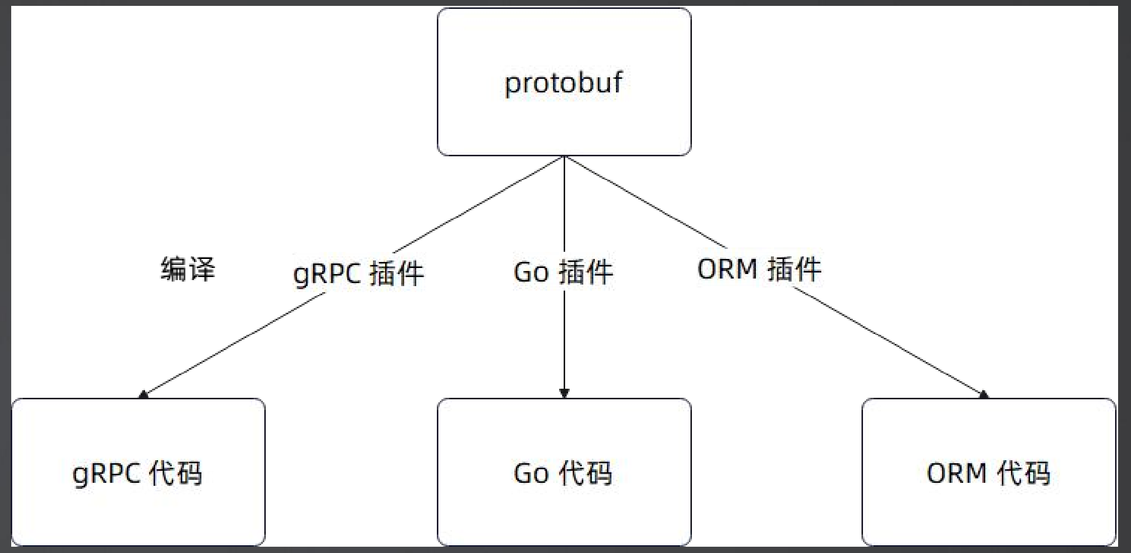


gRPC 与 protobuf

gRPC 使用的 IDL 是 protobuf。

protobuf 是一个独立的 IDL，也就是说你可以用 protobuf 来生成 gRPC 的代码，也可以用 protobuf 来生成其它 RPC 框架的代码。

protobuf 也定义了序列化格式，所以我们也常说使用 protobuf 来作为序列化协议。



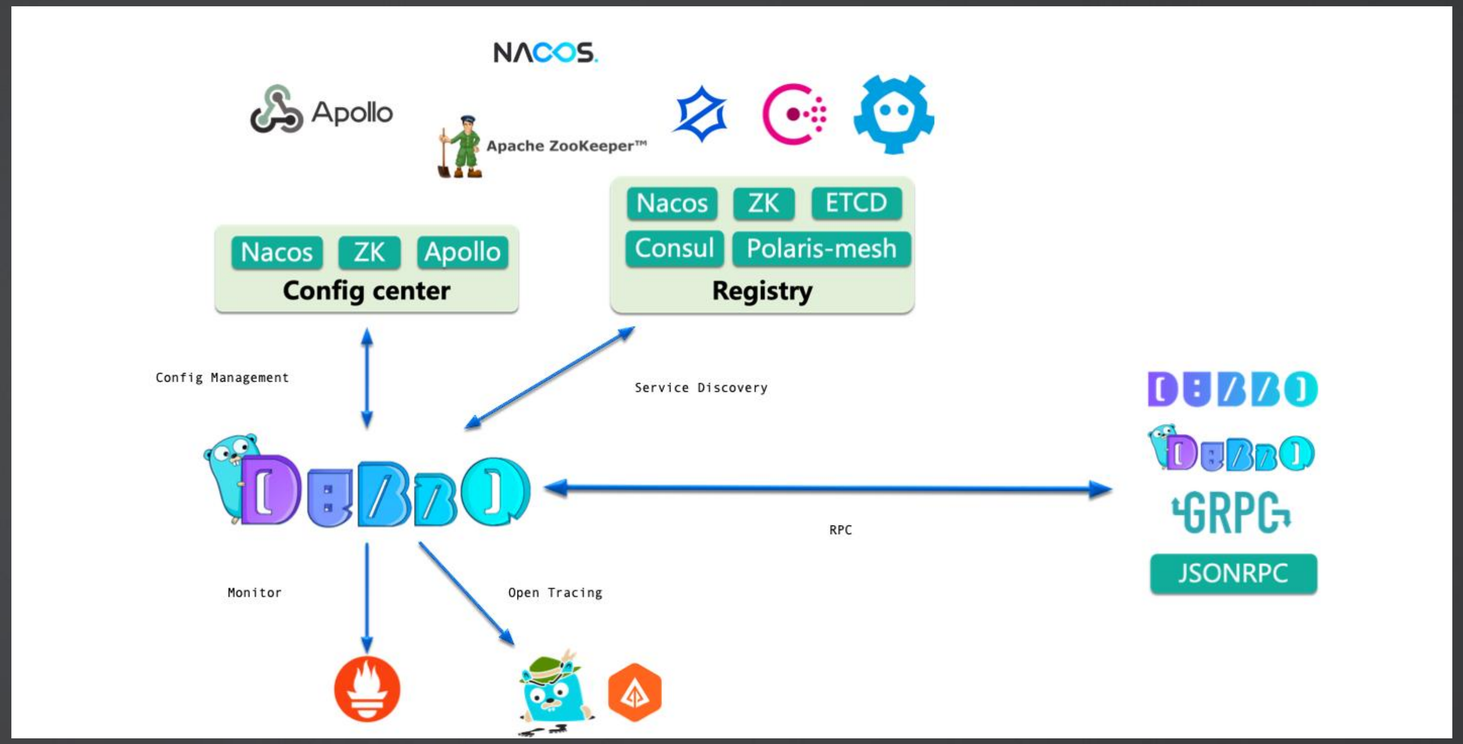
Dubbo 框架

Dubbo 是非常早就出现的微服务框架，一句话总结就是：全家桶。

也就是说，Dubbo 涵盖了从上层服务治理到底层通信协议设计的全方位的内容。

也因为 Dubbo 历经考验，所以基本上微服务相关的所有的话题，你都可以在 Dubbo 里面找到。

非常好的学习对象，也是设计微服务框架非常好的参考对象。

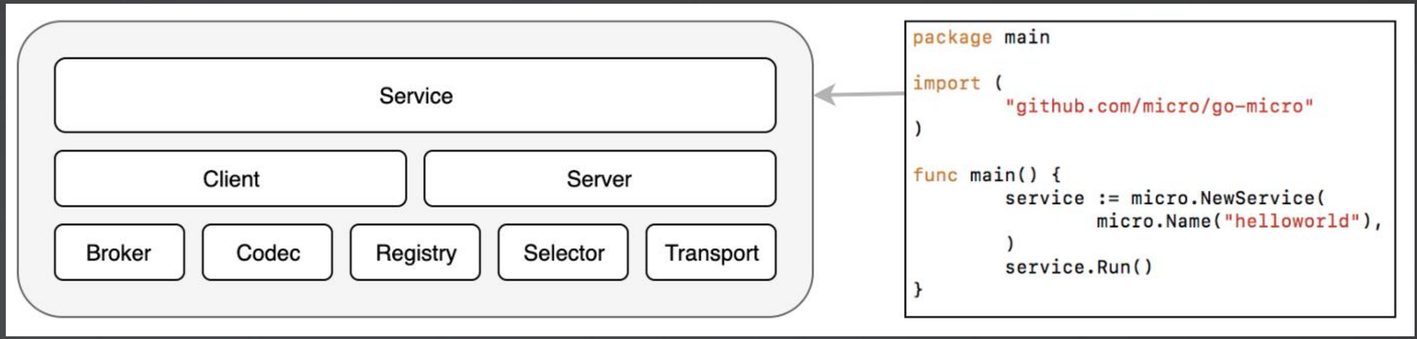


go-micro

go-micro 有自己的协议，它本质上也是利用了 protobuf 作为 IDL。

同时它也支持了 gRPC 和 HTTP。

go-micro 充分利用了插件机制，用户可以替换掉大部分实现，包括注册中心、负载均衡、底层协议。



Kratos

B 站开源出来的，毛老师作品。

主要聚焦在服务治理和快速开发上，也就是说它兼具一个微服务框架的功能，以及一个脚手架的功能。

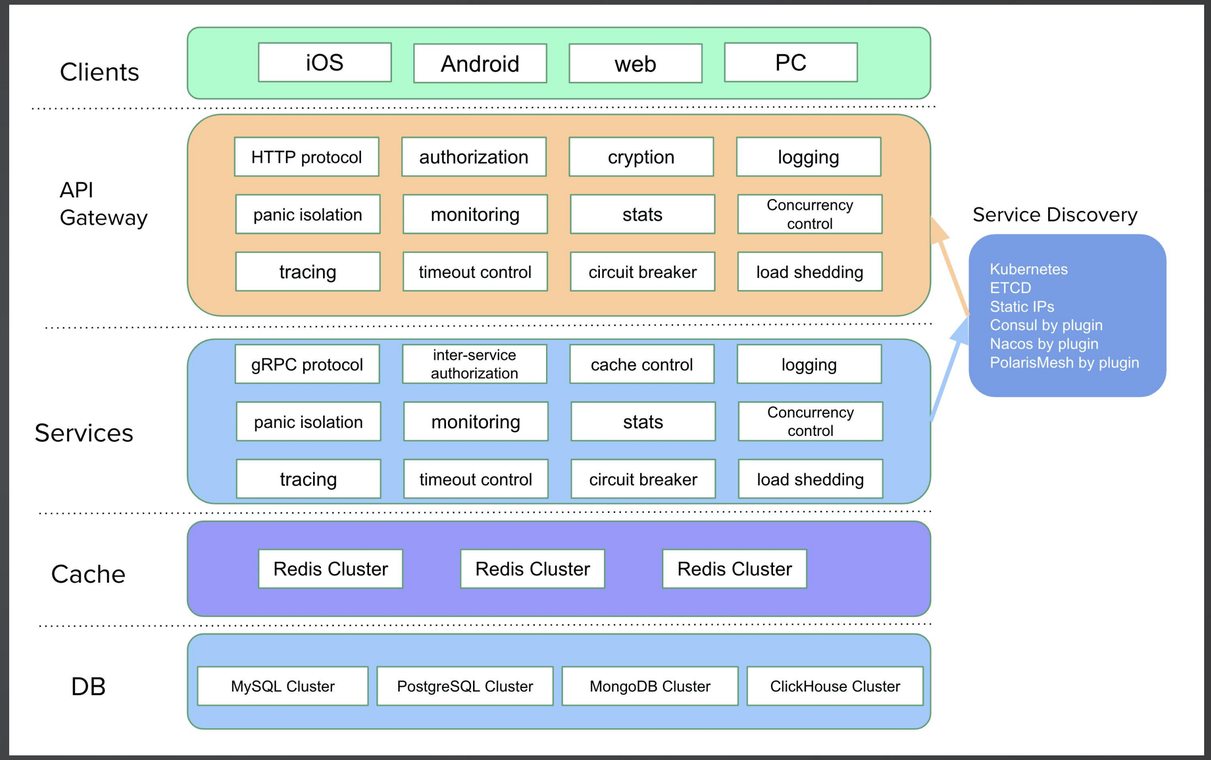
它依托于 gRPC 和 HTTP 来作为底层通信协议。

- APIs： 协议通信以 HTTP/gRPC 为基础，通过 Protobuf 进行定义；
- Errors： 通过 Protobuf 的 Enum 作为错误码定义，以及工具生成判定接口；
- Metadata： 在协议通信 HTTP/gRPC 中，通过 Middleware 规范化服务元信息传递；
- Config： 支持多数据源方式，进行配置合并铺平，通过 Atomic 方式支持动态配置；
- Logger： 标准日志接口，可方便集成三方 log 库，并可通过 fluentd 收集日志；
- Metrics： 统一指标接口，可以实现各种指标系统，默认集成 Prometheus；
- Tracing： 遵循 OpenTelemetry 规范定义，以实现微服务链路追踪；
- Encoding： 支持 Accept 和 Content-Type 进行自动选择内容编码；
- Transport： 通用的 HTTP /gRPC 传输层，实现统一的 Middleware 插件支持；
- Registry： 实现统一注册中心接口，可插件化对接各种注册中心；
- Validation: 通过Protobuf统一定义校验规则，并同时适用于HTTP/gRPC服务.
- SwaggerAPI: 通过集成第三方Swagger插件 能够自动生成Swagger API json并启动一个内置的Swagger UI服务.

go-zero

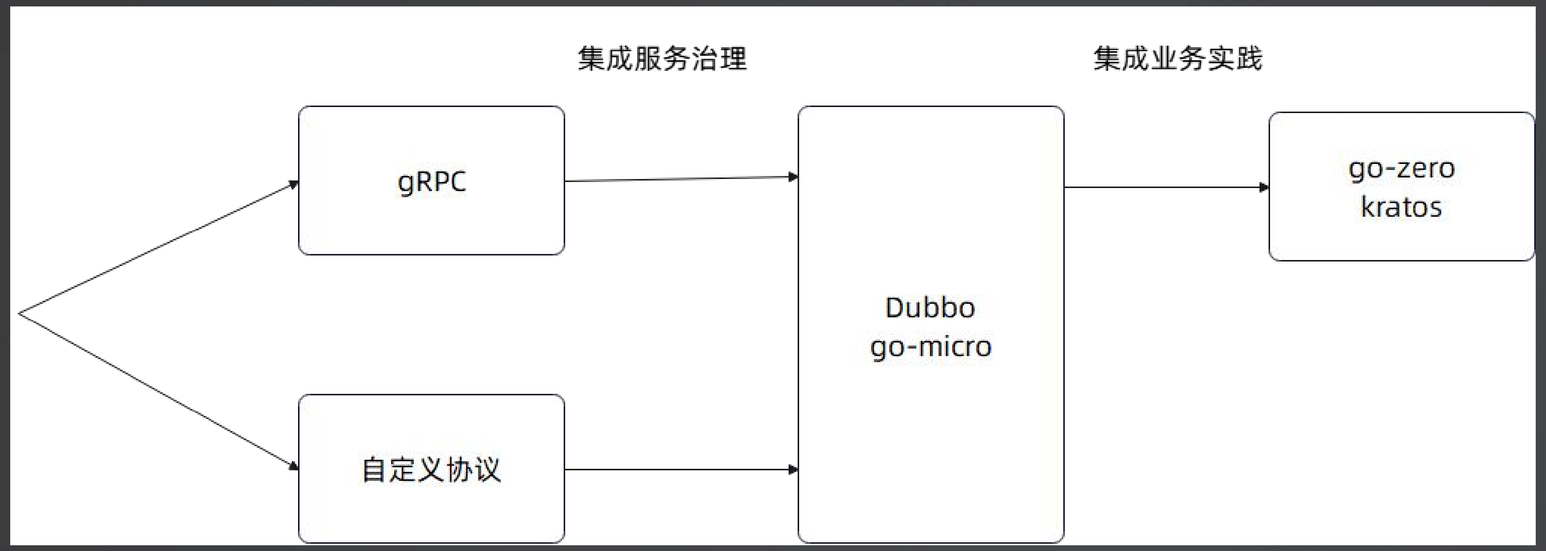
近两年很火的一个框架，它跟 Kratos 是很像的，也同样是聚焦在上层服务治理和快速开发上。

从这个角度来说， go-zero 和 Kratos 都是偏向业务的，里面集成了一些实践中本来应该是微服务框架使用者（而不是设计者）要考虑的内容。



总结

一张图来概括这些框架：



如何选择微服务框架？

懒人就选 go-zero 或者 Kratos 这种，集成了一些业务实践，开箱即用。还能享受到他们总结的最佳实践，比如说 Kratos 里面的错误处理。

稍微没那么懒，那么就可以考虑 go-micro、Dubbo 这种。相比之下，如果你担忧自己可能遇到什么高并发大集群的问题，那么 Dubbo 可能更加可靠一点（一般这种公司都自研）。

如果你想要最高自由度，那么就直接使用 gRPC 或者 HTTP 协议来通信，上面的服务治理需要的时候自己利用接口搞一个。

如何选择微服务框架？

选 gRPC，剩下就随意了

面试要点

- 微服务框架是什么？主要就是解决两个问题，通信和服务治理。
- 你了解 XXX 微服务框架吗？你们需要提前了解一下面试官使用的微服务框架，有针对性的准备。一般记住这个微服务框架的大体功能就可以，后面讨论到具体功能实现的时候再进一步深入了解。
- 为什么使用微服务架构？本质上是为了分而治之，将业务拆分之后独立治理、部署。
- RPC 框架和 RESTful 有什么区别？应该说，两者基本没关联，全是区别。唯一的关联，就是 RPC 框架可以利用 RESTful 来实现。RESTful 是指符合 REST 风格的 HTTP 接口，而 RPC 指的是远程过程调用，从本质上就是两回事。
- RCP 框架和 Web 框架有什么区别？基本也没什么关联，都是区别。唯一的共同点是可以通过对 Web 框架进行封装来实现 RPC 通信。

一般来说，上了体量的公司都会问微服务框架使用相关的问题，这部分在我们具体功能进一步讨论。

Q & A

THANKS