


加钉钉群成为 Contributor 并获取 ppt 下载地址



Dubbo社区贡献者

376人

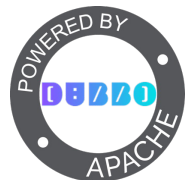


 扫一扫群二维码，立刻加入该群。

DubboMesh动手实践，从示例到原理

张国强

GitHub:zgq25302111



Apache Dubbo

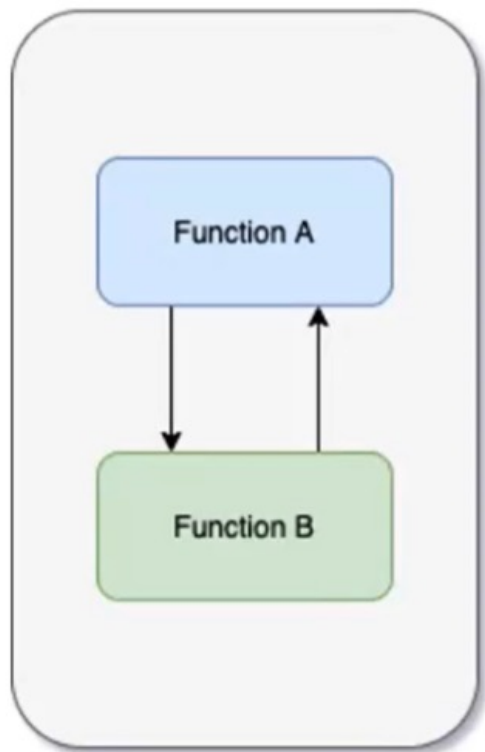
1. Dubbo Mesh 简介
2. Dubbo Sidecar Mesh demo
3. Dubbo Proxyless Mesh demo
4. Dubbo Mesh 体系展望

什么是RPC

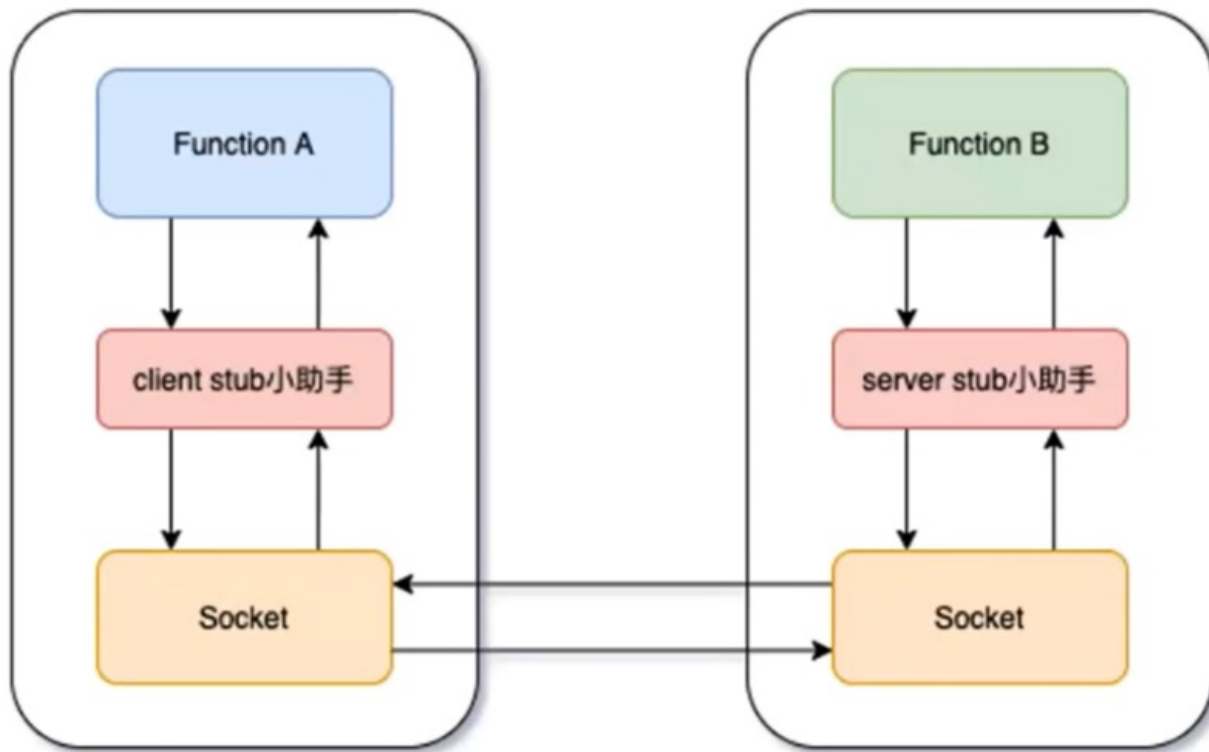


RPC, Remote Procedure Call 即远程过程调用, 远程过程调用其实对标的是本地过程调用。

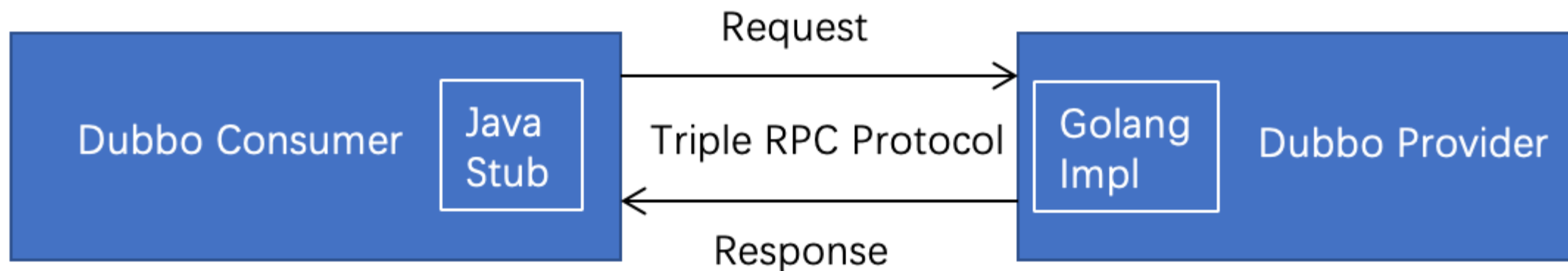
本地过程调用



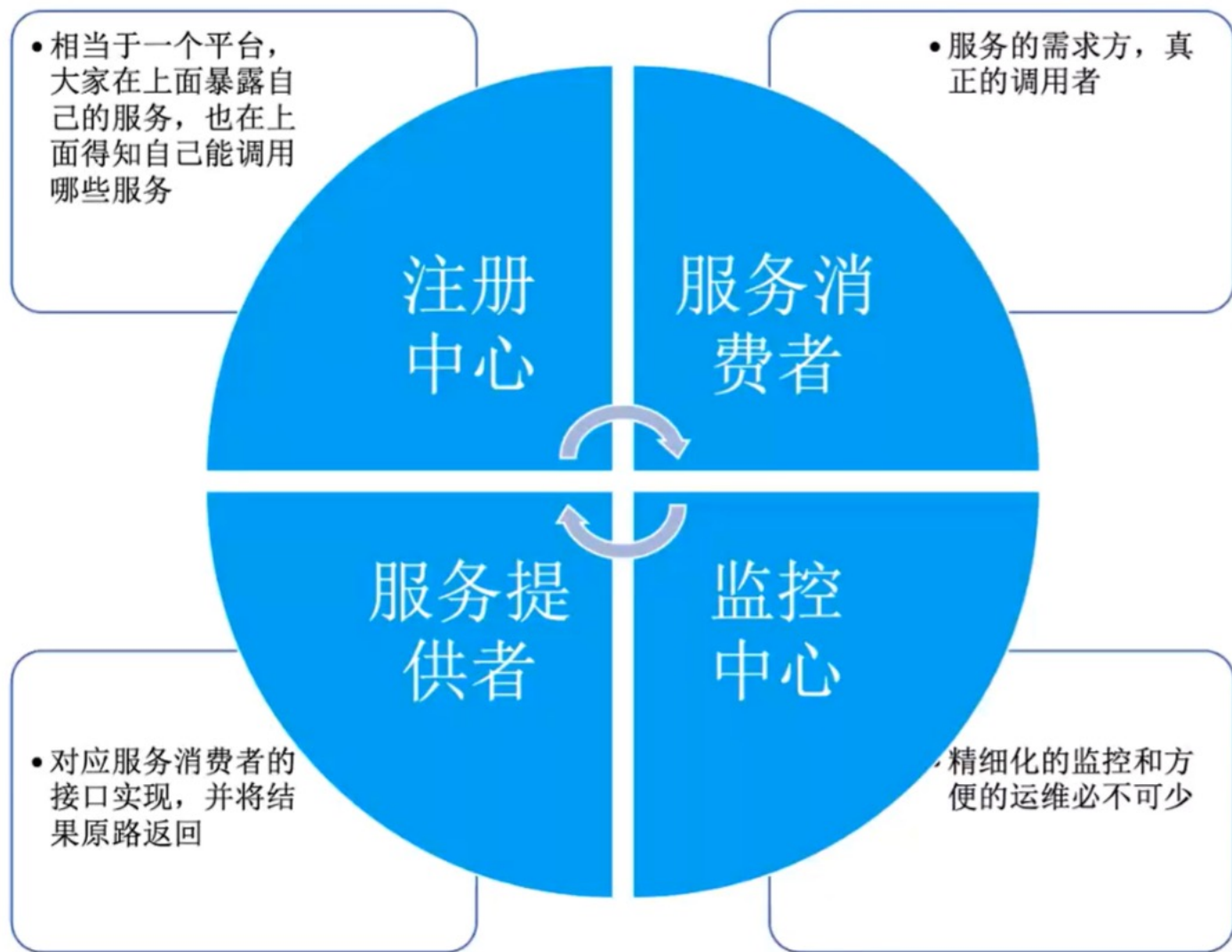
远程过程调用



Dubbo 基本工作流程



如何设计一个 RPC 框架

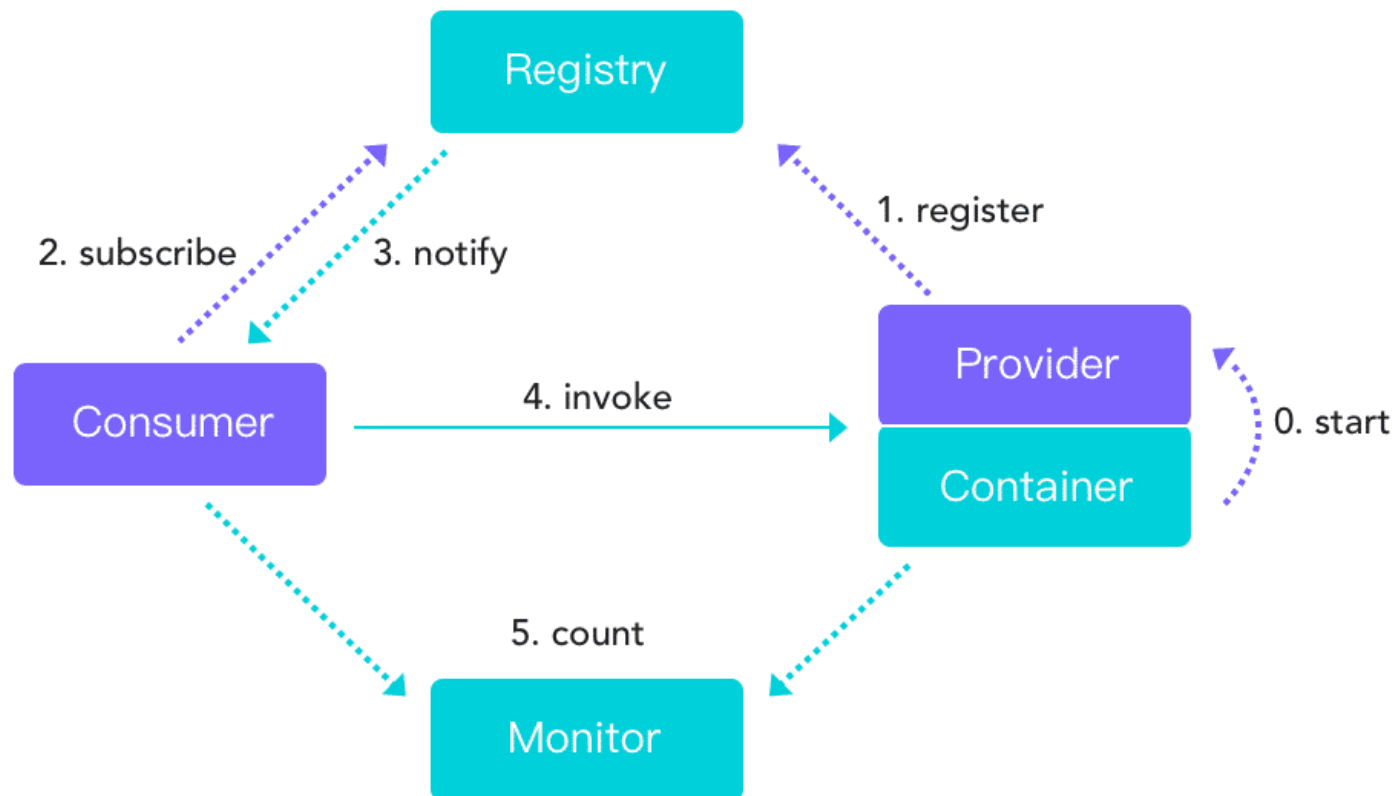


Dubbo Architecture

.....▶ init

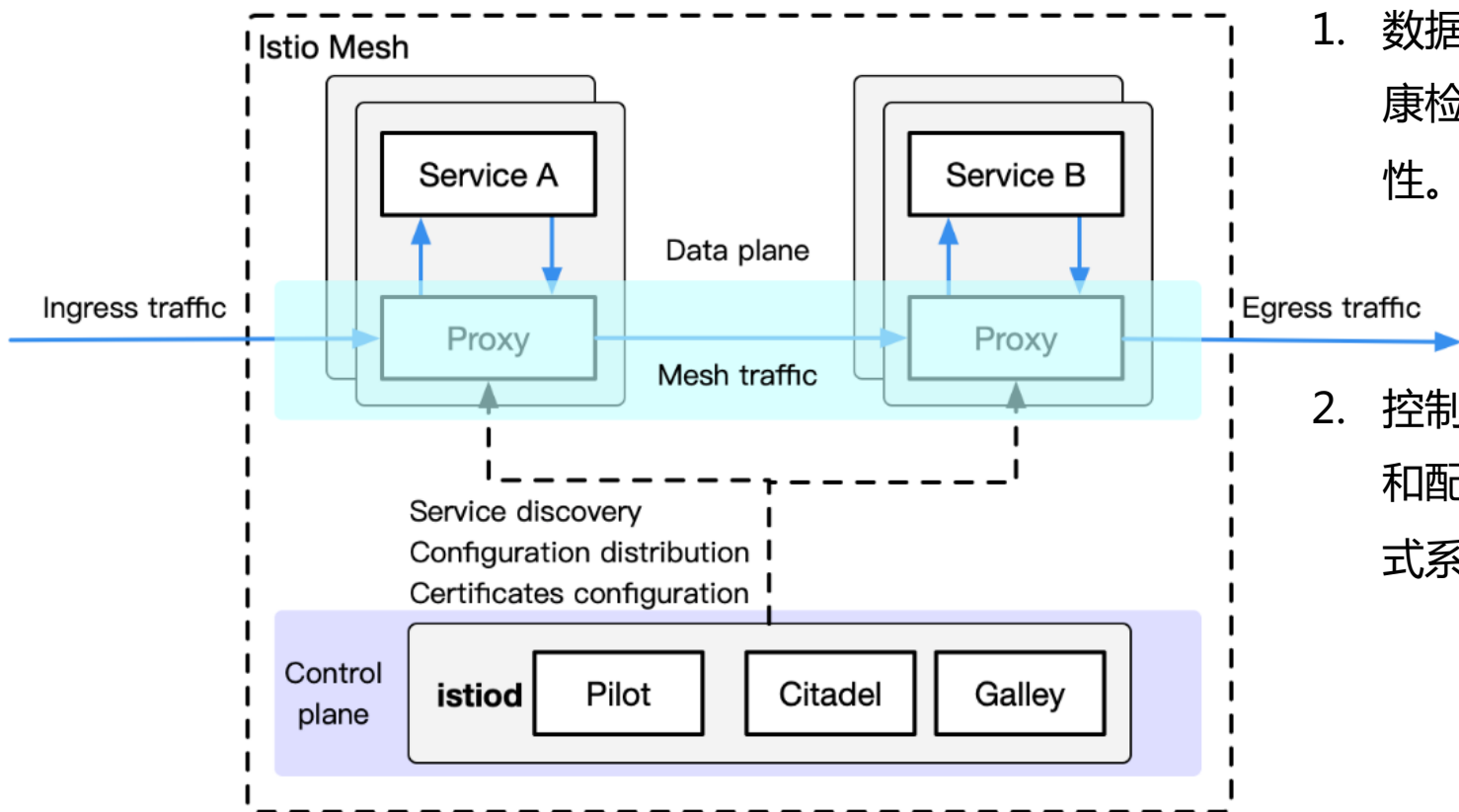
.....▶ async

——▶ sync



1. 与各微服务组件直接适配，耦合度高
2. 微服务组件间治理能力没有交互
3. 多语言实现复杂度高
4. 现有规则亟待升级，增加或升级规则成本高
5. 与云原生基础设施脱节严重

Istio Service Mesh

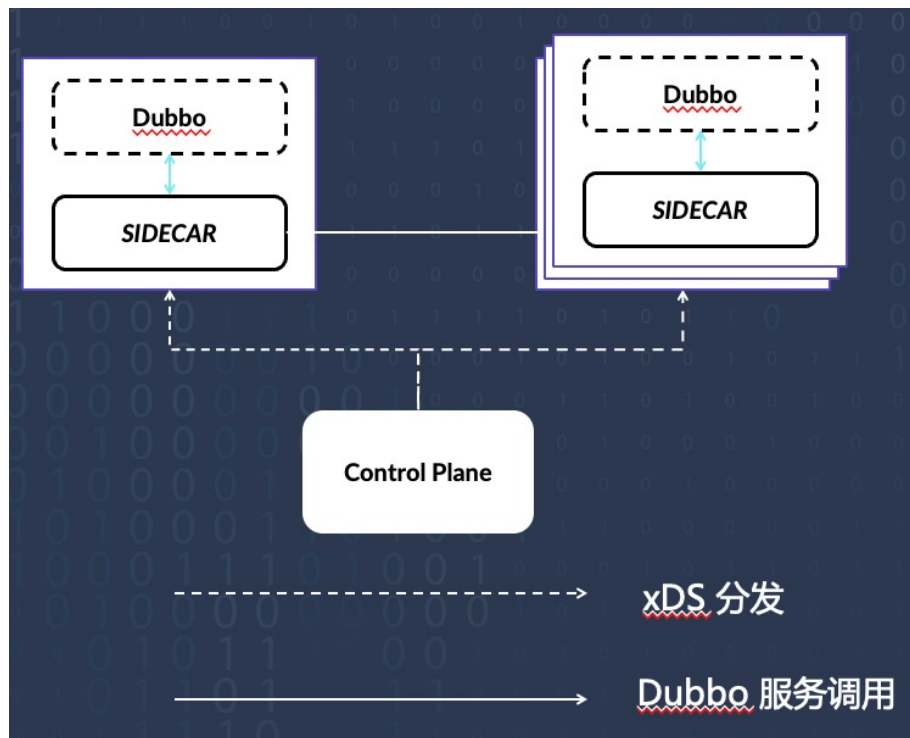


Istio service mesh architecture diagram

1. 数据面:实际处理数据包/请求。负责服务发现, 健康检查, 路由, 负载均衡, 身份验证/授权和可观察性。类比网络环境中的交换机。
2. 控制面: 为网格中所有正在运行的数据面提供策略和配置信息。控制面将所有数据面组织成一个分布式系统。类比网络环境中的路由器。

<https://blog.envoyproxy.io/service-mesh-data-plane-vs-control-plane-2774e720f7fc>

Dubbo Sidecar Mesh



Dubbo 提供了 ThinSDK 的部署模式，在此模式下，Dubbo ThinSDK 将只提供面向业务应用的编程 API、RPC 传输通信能力，其余服务治理 包括地址发现、负载均衡、路由寻址等都统一下沉到 Sidecar，Sidecar 负责与控制面直接通信并接收各种流量管控规则。

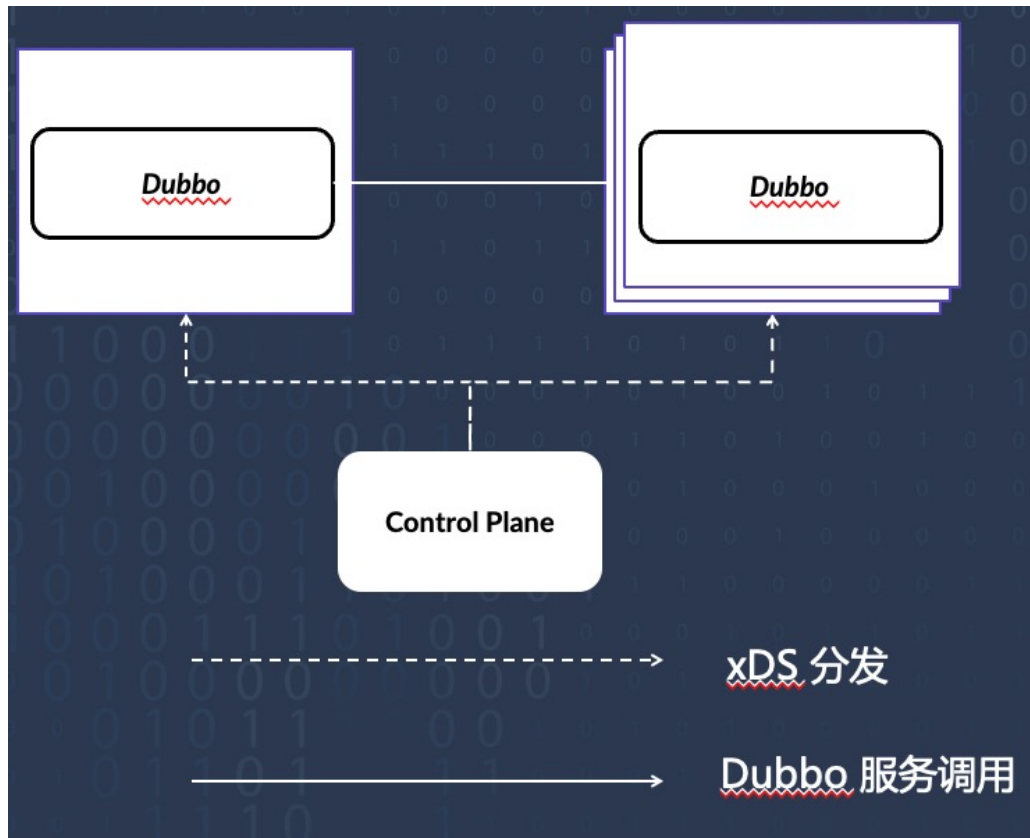
左图是基本部署架构图，Dubbo ThinSDK 与 Sidecar 部署在同一个 Pod 或容器中，通过在外围部署一个独立的控制平面，实现对流量和治理的统一管控。控制面与 Sidecar 之间通过图中虚线所示的 xDS 协议进行配置分发，而 Dubbo 进程间的通信不再是直连模式，转而通过 Sidecar 代理，Sidecar 拦截所有进出流量，并完成路由寻址等服务治理任务。

目前选型为：**Istio+Envoy**

1. Dubbo更注重 RPC 协议与编程模型，而Dubbo Mesh接管耦合在Dubbo业务进程中的微服务治理部分能力
2. 统一的控制面提供证书管理、可观测性、流量治理等能力
3. Sidecar 让 SDK 更轻量、侵入性更小，更好的实现平滑升级、流量拦截等
4. 多语言

1. Sidecar 通信带来了额外的性能损耗，这在复杂拓扑的网络调用中将变得尤其明显。
2. Sidecar 的存在让应用的声明周期管理变得更加复杂。
3. 部署环境受限，并不是所有的环境都能满足 Sidecar 部署与请求拦截要求。

Dubbo Proxyless Mesh

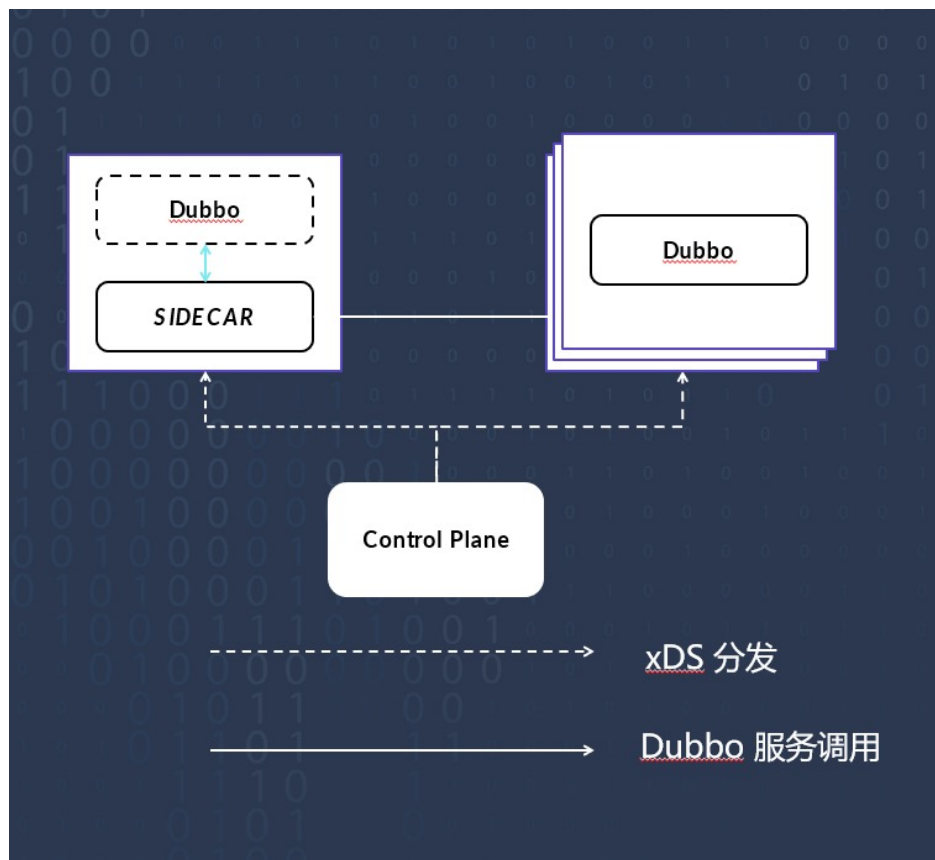


Proxyless 模式使得微服务又回到了 2.x 时代的部署架构。如下图所示，和我们上面看的 Dubbo 经典服务治理模式非常相似，所以说这个模式并不新鲜，Dubbo 从最开始就是这么样的设计模式。但相比于 Mesh 架构，Dubbo2 并没有强调控制面的统一管控，而这点恰好是 Service Mesh 所强调的，强调对流量、可观测性、证书等的标准化管控与治理，也是 Mesh 理念先进的地方。

通过不同语言版本的 Dubbo3 SDK 直接实现 xDS 协议解析，Dubbo 进程可以与控制面（Control Plane）直接通信，进而实现控制面对流量管控、服务治理、可观测性、安全等的统一管控，规避 Sidecar 模式带来的性能损耗与部署架构复杂性。

1. 没有额外的 Proxy 中转损耗，因此更适用于性能敏感应用
2. 更有利于遗留系统的平滑迁移
3. 架构简单，容易运维部署
4. 适用于几乎所有的部署环境

未来 Dubbo Mesh 部署形态



Sidecar 与 Proxyless 模式共存

- 共享服务治理 Control Plane
- 满足不同应用及场景的部署要求
- 适应复杂的基础设施环境
- 提高总体架构可用性

步骤：

1. 创建一个 Dubbo 应用(dubbo-samples-mesh-k8s)
2. 构建容器镜像并推送到镜像仓库 (本示例官方镜像)
3. 分别部署 Dubbo Provider 与 Dubbo Consumer 到 Kubernetes 并验证 Envoy 代理注入成功
4. 验证 Envoy 发现服务地址、正常拦截 RPC 流量并实现负载均衡
5. 基于 Istio VirtualService 实现按比例流量转发

<https://dubbo.apache.org/zh/overview/tasks/mesh/dubbo-mesh/>

Dubbo Proxyless Mesh demo



步骤：

1. 搭建 Kubernetes 环境
2. 搭建 Istio 环境
3. 拉取代码并构建
4. 构建镜像
5. 创建namespace
6. 部署容器

<https://dubbo.apache.org/zh/overview/tasks/mesh/proxyless/>

代码关键点



对比dubbo2 下需要提供 provideBy 信息，填写k8s服务名。

由于当前 Dubbo 版本受限于 istio 的通信模型无法获取接口所对应的应用名，即原生的XDS 协议，服务发现只做到了应用这个粒度（service），应用 和 接口的映射关系目前只能由编码方维护，因此需要配置 providedBy 参数来标记此服务来自哪个应用。未来我们将基于 Dubbo Mesh 的控制面实现自动的服务映射关系获取，届时将不需要独立配置参数即可将 Dubbo 运行在 Mesh 体系下，敬请期待。

```
@Component("annotatedConsumer")
public class GreetingServiceConsumer {

    2 usages
    private static final Logger LOGGER = LoggerFactory.getLogger(GreetingServiceConsumer.class);

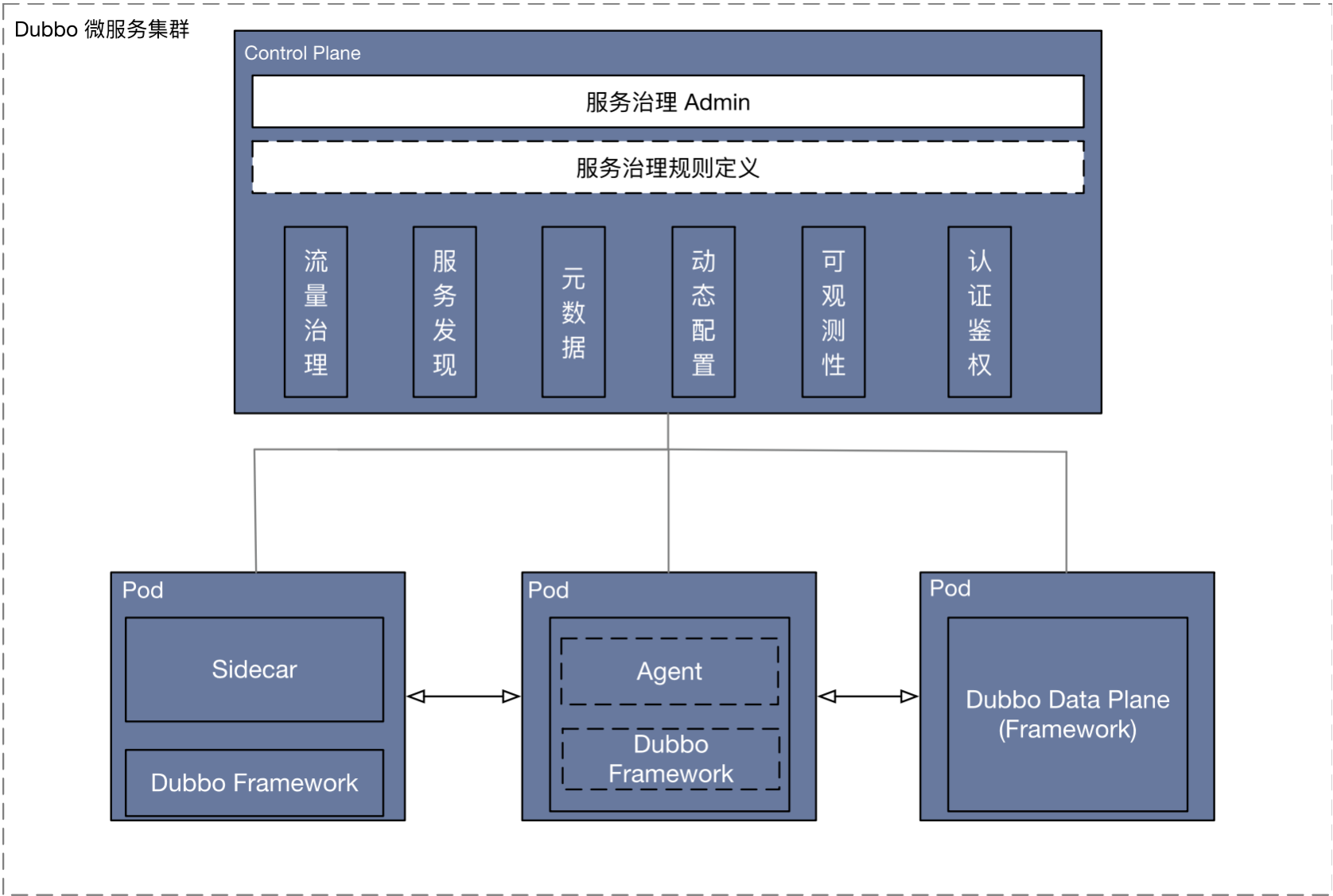
    3 usages
    @DubboReference(version = "1.0.0", providedBy = "dubbo-samples-mesh-provider" lazy = true, providerPort = 50052)
    private Greeter greeter;

    1 usage
    public void doSayHello(String name) {
        final GreeterReply reply = greeter.greet(GreeterRequest.newBuilder()
            .setName(name)
            .build());
        LOGGER.info("consumer Unary reply <-{}", reply);
    }
}
```

Dubbo Mesh 体系展望



目前dubbo thinsdk 已经成功抽出来了，距离理想的架构只差控制面了



控制面

1. 专注于策略的决策，配置，控制器集群独立部署
2. 由控制面向其他中间件交互，允许依赖多种中间件client
3. 架构选型上需要替换中间件单独升级控制面就可以了

管控协议

负责将控制面的控制指令，配置模型下发到数据面

数据面

专注于策略的执行，只依赖协议相关模型，干净纯粹

每周五 Dubbo 微服务技术直播，扫码关注任一渠道。可观看直播回放。

B 站直播



微信公众号
Apache Dubbo

