

学习者：李浩洋

课题组：CrowdOS小组

**Istio学习笔记**

**RESUME**

PERSONAL

目录

[一、 服务网格（Service Mesh） 3](#_Toc10542)

[二、 Sidecar模式 6](#_Toc7607)

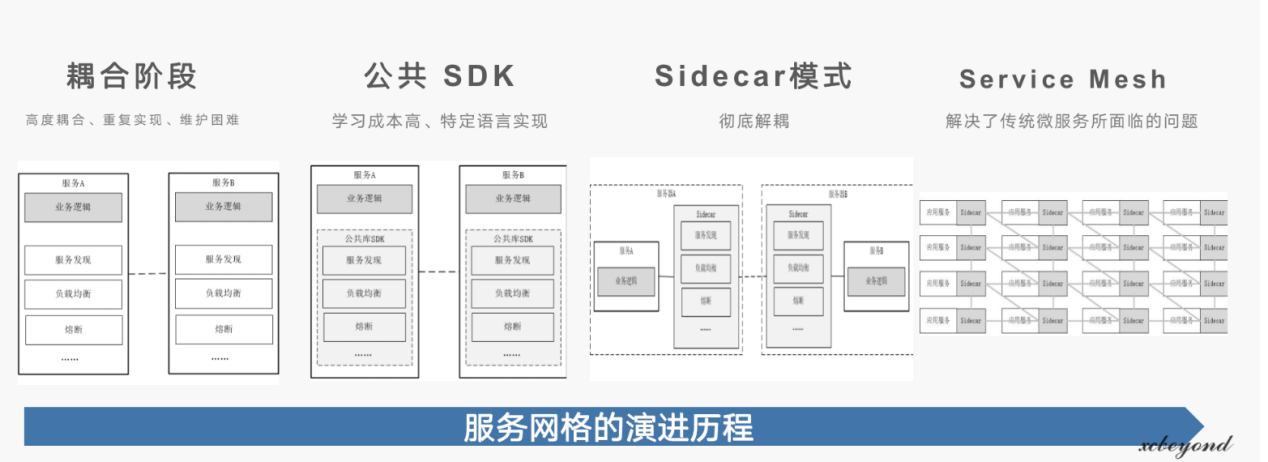
1. **服务网格（Service Mesh）**

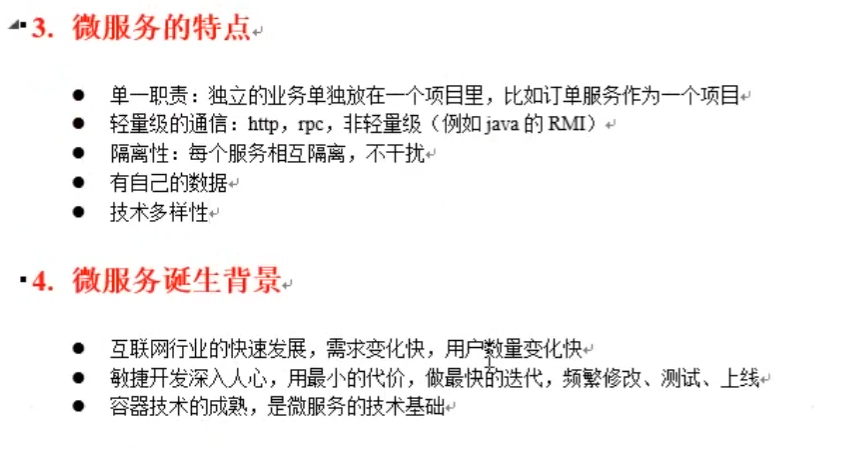
耦合阶段：高度耦合、重复实现、维护困难，在耦合架构设计中体现的最为突出，单体架构就是典型的代表。

公共 SDK：让基础设施功能设计成为公共 SDK，提高利用率，是解藕最有效的途径，比如 Spring Cloud 就是类似的方式。但学习成本高、特定语言实现，却将一部分人拦在了门外。

Sidecar 模式：再次深度解藕，不单单功能解藕，更从跨语言、更新发布和运维等方面入手，实现对业务服务的零侵入，更解藕于开发语言和单一技术栈，实现了完全隔离，为部署、升级带来了便利，做到了真正的基础设施层与业务逻辑层的彻底解耦。另一方面，Sidecar 可以更加快速地为应用服务提供更灵活的扩展，而不需要应用服务的大量改造。

Service Mesh：把 Sidecar 模式充分应用到一个庞大的微服务架构系统中来，为每个应用服务配套部署一个 Sidecar 代理，完成服务间复杂的通信，最终就会得到一个的网络拓扑结构，这就是 Service Mesh，又称之为“服务网格“。它从本质上解决了传统微服务所面临的问题。





**服务网格的定义：**服务网格（Service Mesh），作为服务间通信的基础设施层。是轻量级高性能网络代理，提供安全的、快速的、可靠地服务间通讯，与实际应用部署一起，但对应用透明。应用作为服务的发起方，只需要用最简单的方式将请求发送给本地的服务网格代理，然后网格代理会进行后续的操作，如服务发现，[负载均衡](https://cloud.tencent.com/product/clb?from_column=20065&from=20065" \t "https://cloud.tencent.com/developer/article/_blank)，最后将请求转发给目标服务。



**服务网格的核心价值：**

为下沉到基础设施提供可能：将微服务通信下沉到基础设施层，屏蔽了微服务处理各种通信问题的复杂度。

帮助应用轻量化，专注业务：开发者无需关心通信层和服务治理的具体实现，真正像本地调用一样使用微服务，通信相关的一切工作直接交给 Service Mesh，让开发者更关注于业务的开发。

实现应用的云原生化：加速应用上云，实现云原生化。

**服务网格的主要功能特性：**



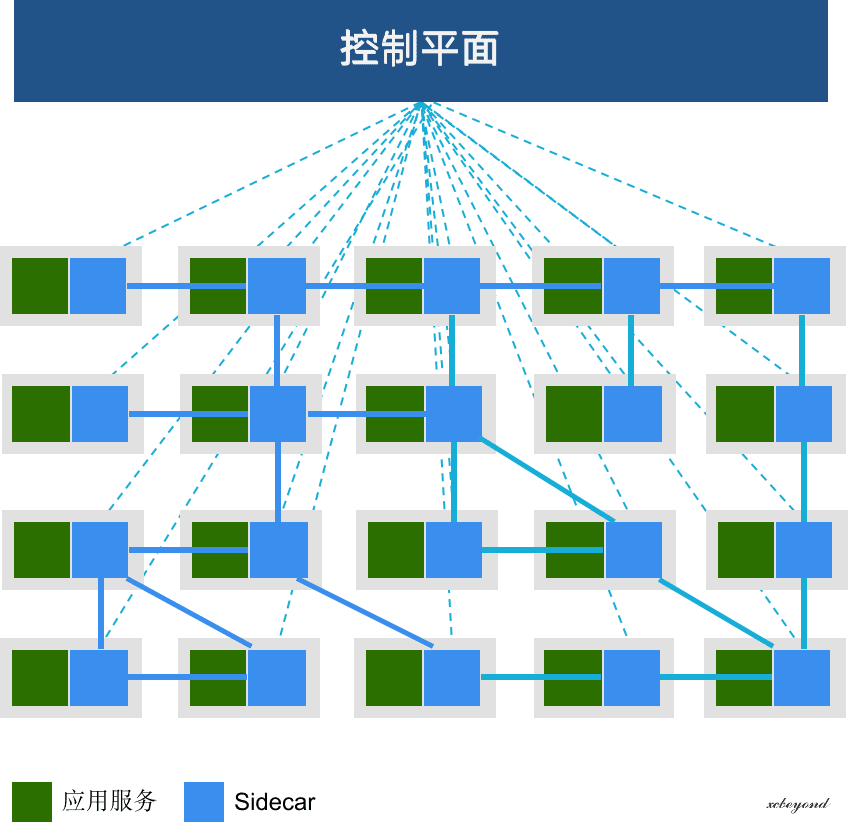
**流量控制：**为应用提供智能路由（如，金丝雀发布、A/B 测试等）、超时重试、熔断、故障注入、流量镜像等各种控制能力。

**策略：**可以为流量设置配额、黑白名单等策略。

**网络安全：**提供服务间访问控制、 TLS 加密通信。

**可观测性：**为所有通信生成详细的遥测数据，包括指标数据、日志、追踪，提供给应用系统完整的监控能力。

服务网格整体框架如下：



主要核心内容为：

**控制平面（Control）：** 控制和管理数据平面中的 Sidecar 代理，完成配置分发、服务发现、流量路由、授权鉴权等功能，以达到对数据平面的统一管理。

**数据平面（Data Plane）：** 由整个网格内的 Sidecar 代理组成，这些代理以 Sidecar 的形式和应用服务一起部署。这些代理负责协调和控制应用服务之间的所有网络通信。每一个 Sidecar 会接管进入和离开服务的流量，并配合控制平面完成流量控制等方面的功能。

1. **Sidecar模式**

**什么是Sidercar模式？**

Sidecar模式是一种将应用功能从应用本身剥离出来作为单独进程的方式。该模式允许我们向应用无侵入添加多种功能，避免了为满足第三方组件需求而向应用添加额外的配置代码。

就像边车加装在摩托车上一样，在软件架构中，sidecar附加到主应用，或者叫父应用上，以扩展/增强功能特性，同时Sidecar与主应用是松耦合的。

举个例子，假设现在有6个相互通信的微服务，每个微服务都需要具有可观察性、监控、日志记录、配置、断路器等功能，而所有这些功能都是在微服务中使用一些第三方库实现的。

**Sidecar模式的好处：**

通过将公用基础设施相关功能抽象到不同的层来降低微服务的代码复杂性

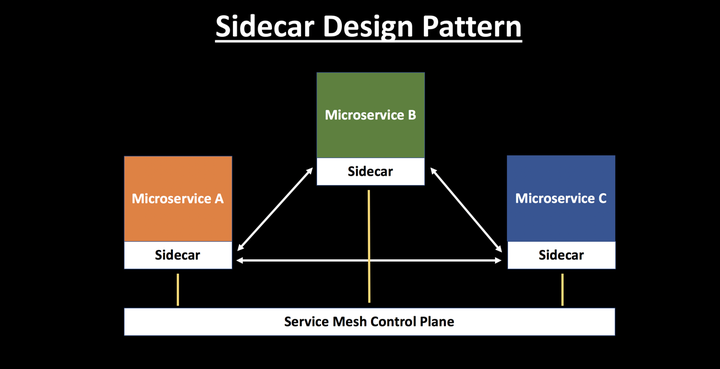
由于我们不需要在每个微服务中编写配置代码，因此减少了微服务架构中的代码重复

P应用和底层平台之间实现了松耦合

**Sidecar模式如何工作：**

Service Mesh层可以位于应用程序侧的Sidecar容器中，同一sidecar的多个副本可以附在每个应用旁。

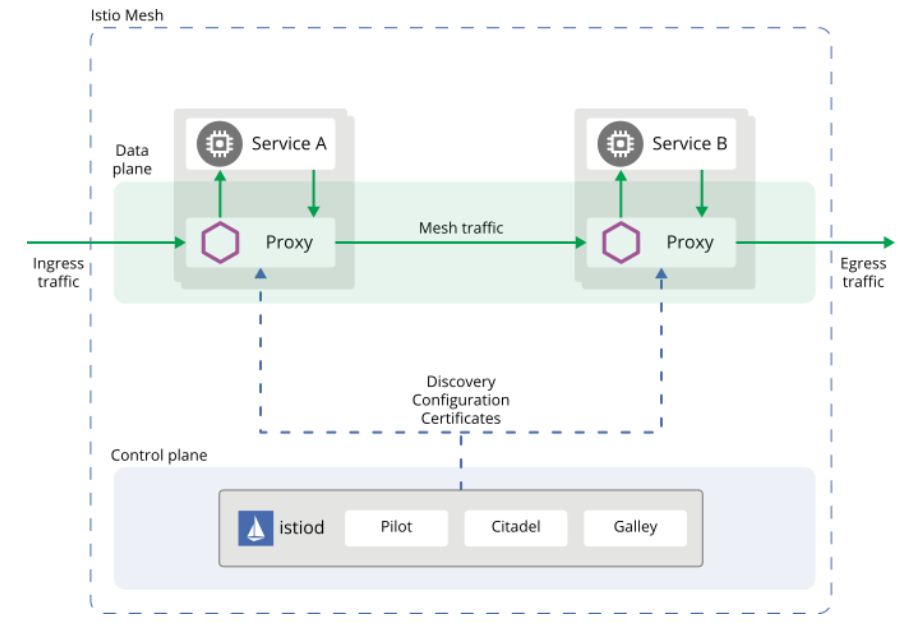
来自单个服务的所有传入和传出网络流量均通过Sidecar代理，完成微服务之间的流量管理、遥测数据收集以及策略的执行等等。从某种意义上来说，服务对于网络是无感知的，只知道所附加的sidecar代理。这就是Sidecar模式工作的本质，它将网络依赖抽象成了Sidecar。



1. **Istio框架**

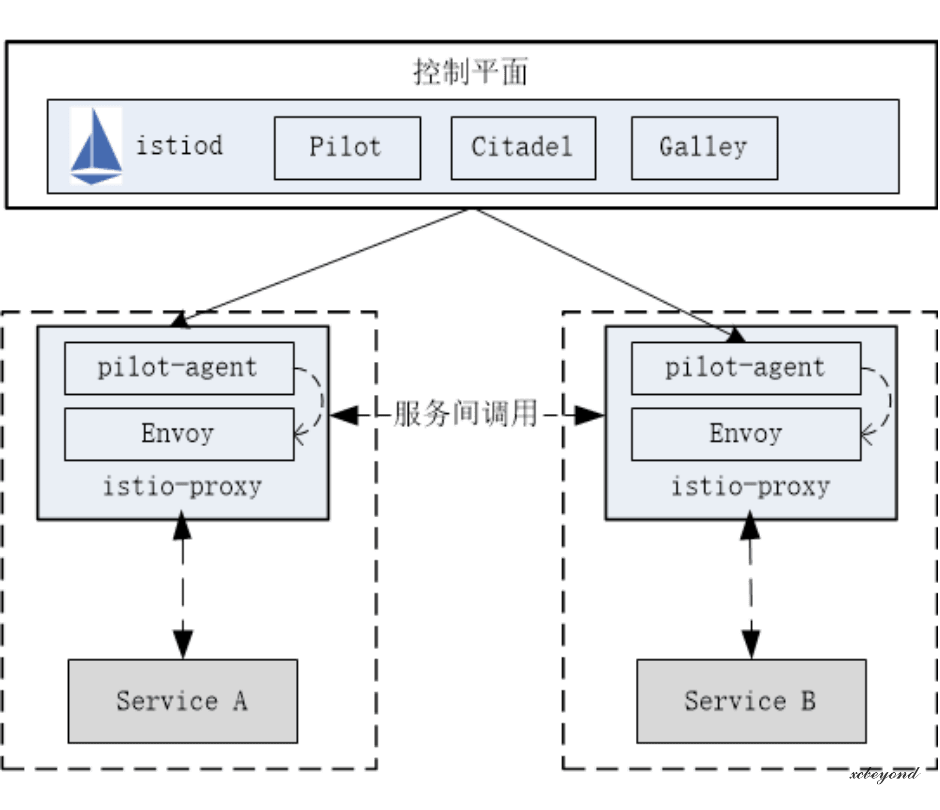
Istio 是由 Google 、IBM 和 Lyft 发起的开源的服务网格框架。该项目在 2017 年推出，截止目前已发布了 1.10.1 版本。

Istio 提供了一个完整的解决方案，为整个服务网格提供行为洞察和操作控制，以满足微服务应用程序的多样化需求。



**Istio数据平面**

Istio 的数据平面默认使用 Envoy ，是基于 Envoy 新增了一些扩展（即：istio-proxy），C++ 语言编写。



主要职责：

**服务发现**：探测所有可用的上游或下游服务实例。

**健康检测**：探测上游或下游服务实例是否健康，是否准备好接收网络流量。

**流量路由**：将网络请求路由到正确的上游或下游服务。

**负载均衡**：在对上游或下游服务进行请求时，选择合适的服务实例接收请求，同时负责处理超时、断路、重试等情况。

**身份验证和授权**：在 istio-agent 与 istiod 配合下，对网络请求进行身份验证、权限验证，以决定是否响应以及如何响应，使用 mTLS 或其他机制对链路进行加密等。

**链路追踪**：对于每个请求，生成详细的统计信息、日志记录和分布式追踪数据，以便操作人员能够理解调用路径并在出现问题时进行调试。

**Istio控制平面**

控制平面由原来分散、独立部署的三个组件（Pilot、Citadel、Galley）整合为一个独立的 istiod，变成了一个单进程、多模块的组织形态（下图右图），极大的降低了原来部署的复杂度。

**Pilot**：负责 Istio 数据平面的 xDS 配置管理，具体包括服务发现、配置规则发现、xDS 配置下发。

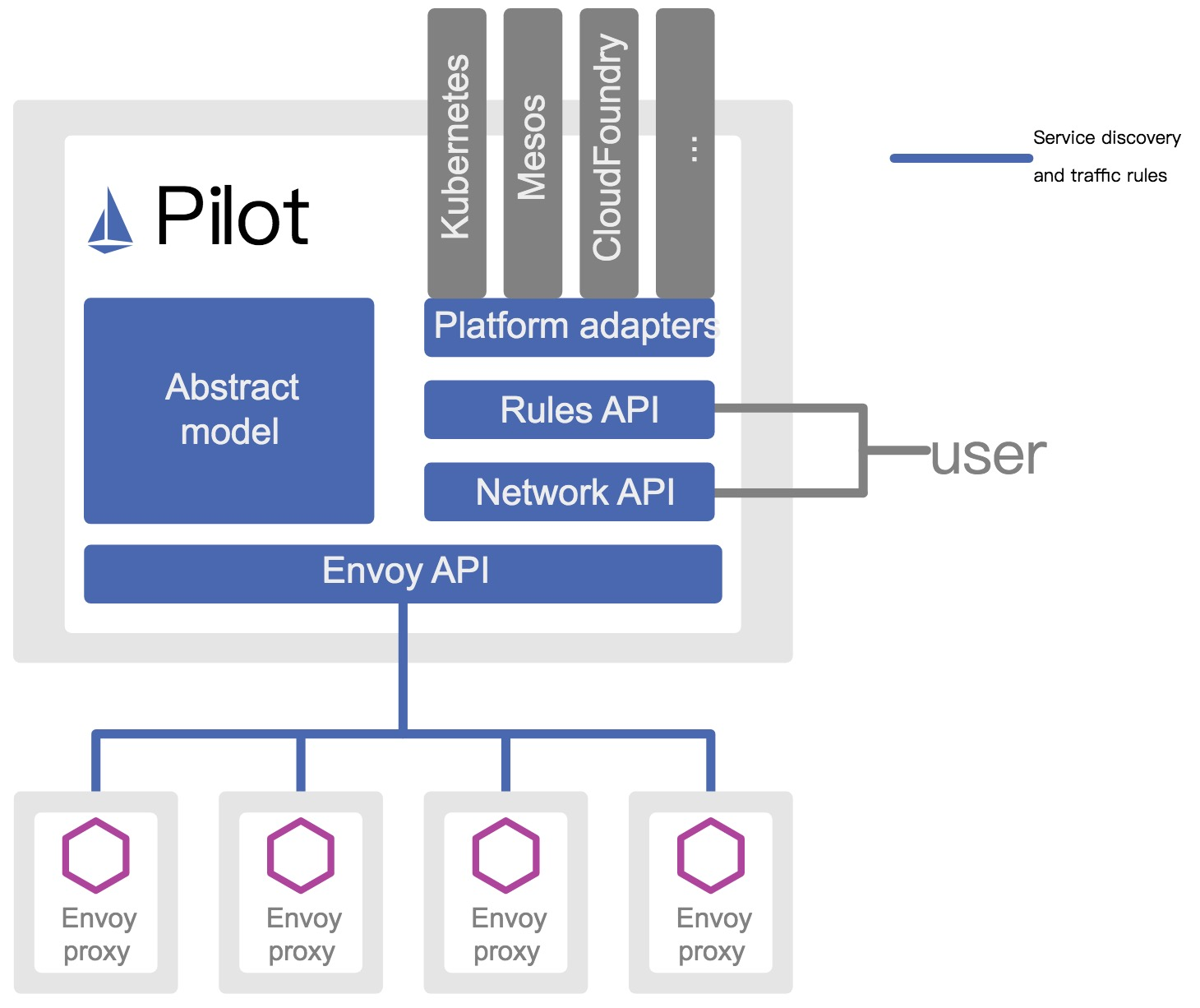
**Citadel**：负责安全证书的管理和发放，实现授权和认证等操作。

**Galley**：负责配置的验证、提取和处理等功能，将 Istio 和底层平台(如,Kubernetes)进行解耦。

其中，Citadel、Galley 组件逐步在弱化，在 Istio 版本迭代中，已经基本看不见它们的踪迹了。（已经不断整合在其它组件中）

接下来，着重看看 Pilot 组件。

Pilot 是 Istio 中的核心组件，用于管理和配置部署在特定 Istio 服务网格中的所有 Sidecar 代理实例。它管理 Sidecar 代理之间的路由流量规则，并配置故障恢复功能，如超时、重试和熔断等。

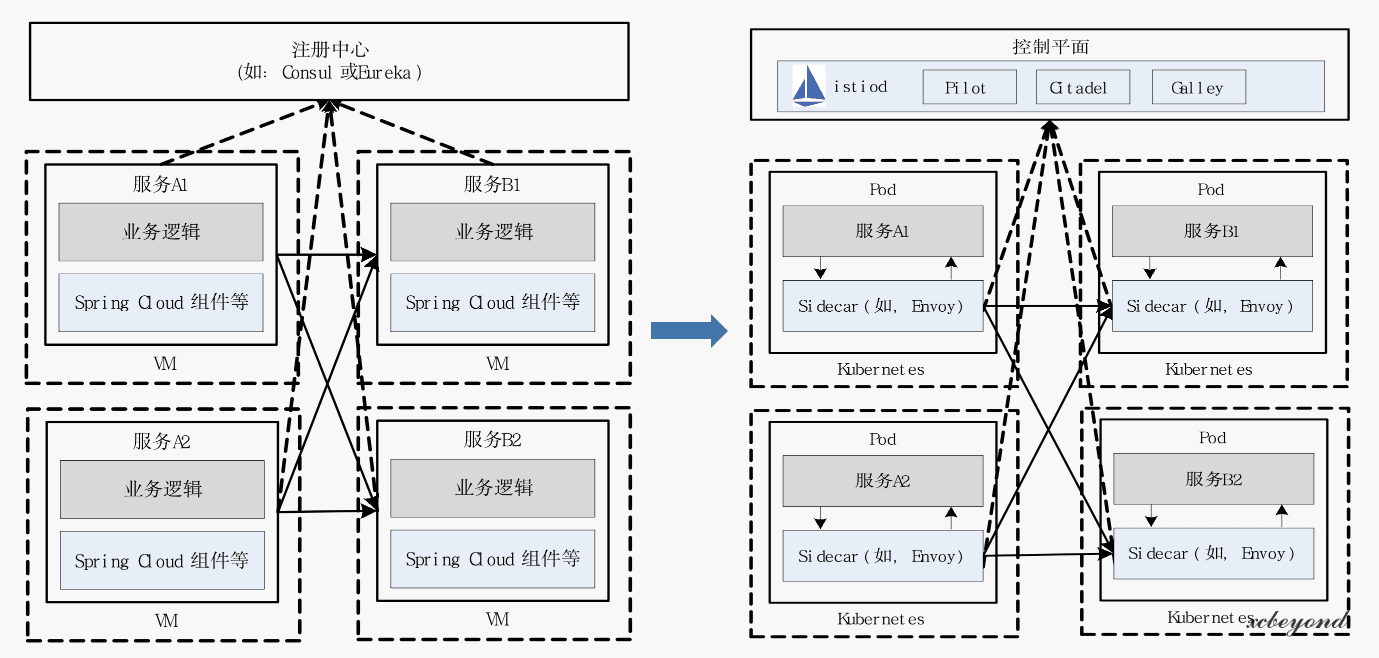


**抽象模型（Abstract model）**：为了实现对不同[服务注册中心](https://cloud.tencent.com/product/src?from_column=20065&from=20065" \t "https://cloud.tencent.com/developer/article/_blank) （如，Kubernetes、Consul） 的支持，完成对不同输入来源数据的抽象，形成统一的存储格式。

**平台适配器 （Platform adapters）**：借助平台适配器 Pilot 实现服务注册中心数据到抽象模型之间的数据转换。

**xDS API**：是源于 Envoy 项目的标准数据平面 API， 将服务信息和流量规则下发到数据平面的 Sidecar。通过采用该标准 API， Istio 将控制平面和数据平面进行了解耦，为多种数据平面 Sidecar 实现提供了可能性，如：蚂蚁金服开源的 Golang 版本的 MOSN。

**用户 API（User API）**：提供了面向业务的高层抽象，可以被运维人员理解和使用。



为了解决微服务框架的侵入性问题，我们引入服务网格。