HZ BOOKS
华章IT

零基础学习Istio的实用教程，从实战角度出发，全面介绍Istio技术要点与应用技巧。

从微服务架构搭建，到使用Istio强的大功能进行管理，包含大量案例和实操步骤。

實戰

毛广献 编著

Istio in Practice

Istio入门与实战

机械工业出版社
China Machine Press

實戰

Istio in Practice

Istio入门与实战

毛广献 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Istio 入门与实战 / 毛广献编著 . —北京：机械工业出版社，2019.4
（实战）

ISBN 978-7-111-62524-7

I. I… II. 毛… III. 互联网络 – 网络服务器 IV. TP368.5

中国版本图书馆 CIP 数据核字（2019）第 068126 号

Istio 入门与实战

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：吴 怡 责任校对：殷 虹
印 刷：三河市宏图印务有限公司 版 次：2019 年 5 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：20
书 号：ISBN 978-7-111-62524-7 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 读者信箱：hzit@hzbook.com

内容简介

本书从实战的角度全面介绍Istio技术要点与应用技巧，可帮助读者快速搭建微服务架构并进行管理。主要内容包括：Istio架构设计与主要功能；如何部署微服务架构应用；Istio的服务流量管理功能，如流量拆分、A/B 测试和灰度发布；Istio的弹性服务功能，如负载均衡、连接池、服务健康检查，服务熔断、超时、重试、限流功能的配置；如何用故障注入方式来测试服务的稳定性，提前发现问题并修复；服务间通信加密和访问权限的控制；服务指标、日志的收集，服务调用链追踪；Istio 部署后的维护工作等。

作者简介

毛广献 某游戏社区与数据服务公司运维工程师，参与过多项业务技术架构设计与应用开发运维。对分布式、微服务等技术有着独到的见解。他是热爱技术的极客，喜欢研究新技术和开源项目。

封面设计 锡 彬

华章IT
HZBOOKS | Information Technology

Preface 前　言

近几年来，容器技术的飞速发展使得微服务技术更容易落地，微服务架构在业界逐渐流行起来。但是微服务架构对基础设施要求较高，微服务依赖持续集成、服务注册、服务发现、负载均衡、健康检测、配置管理、服务路由、服务容错、日志收集、指标收集、调用链追踪等，而构建这一套基础设施的成本巨大。因此，微服务相关的框架逐渐露出水面，比如Java 语言的 Spring Cloud 框架。虽然这些微服务框架为我们提供了很多便利，但由于这些框架是与编程语言绑定的，使得我们应用的技术栈受到了限制。即使后来有其他的微服务框架也支持多编程语言的技术栈，但由于这些微服务框架代码对服务代码入侵严重，给后续服务框架的 bug 修复和版本升级带来了一定的困难。所以，服务网格的概念一经提出，便得到了很多人的支持，人们对这项技术抱有很大的期望，希望能解决当前微服务所遇到的问题。

当我第一次接触服务网格技术的时候，就觉得服务网格将来可能会像 IaaS、PaaS 一样成为业界的主流技术，会得到广泛应用。其实，服务网格并没有提供什么新的概念和功能，它只是把原来服务框架所做的功能完全独立出来，整合了一个服务网格的基础设施层。这个改变看似很小，但是能使服务与服务治理功能实现完全解耦，这个影响是巨大的。

2017 年 5 月，谷歌、IBM、Lyft 等公司共同努力实现的开源服务网格 Istio 正式发布了第一个版本。而后又一个服务网格开源实现 Conduit 开始启航，服务网格进入了更多技术人员的视野，经过努力，Istio 在 2018 年 7 月正式发布了 1.0.0 版本。

2018 年是服务网格快速发展的一年，Istio 发布的 1.0.0 版本标志着 Istio 已经成熟到可以接受生产流量的考验。2019 年服务网格将会持续保持热度，作为一名技术人员，现在是时候了解一下服务网格技术了。而在所有的开源服务网格实现中，最成熟的肯定是 Istio。因此，要学习和了解服务网格，首先应该学习使用 Istio。

由于 Istio 官方文档使用英文编写，而且只简单演示了 Istio 提供的功能，没有基于一般的使用场景，我在学习 Istio 时就很迷茫。于是，我结合自己的学习经验和方法编写了这本如

何在实战中学习 Istio 的书，以便技术人员能以最简单的方式上手 Istio，理解 Istio，并能在生产环境中应用 Istio 服务网格。

由于本书是一本实战类型的书，书中没有大篇的理论知识。对于 Istio 提供的功能，本书只简单地描述其作用和使用方式，然后使用实验来演示效果。我相信，只要你跟着书中的实验操作，并理解这些实验的目的，学习完本书后，你就一定能熟练使用 Istio，并在以后的 Istio 使用中得心应手。

对服务网格感兴趣的人都可以阅读本书。如果你想了解服务网格，想知道服务网格提供了哪些功能，能解决什么问题，本书将是一个不错的选择。如果你想了解 Istio，体验 Istio，将来有可能将 Istio 应用于生产环境，那么阅读本书将是一个不错的开始。

本书主要内容如下：

第 1 章介绍服务网格的由来以及服务网格能给我们带来什么，接着介绍开源的服务网格实现 Istio 的主要功能特性及其架构设计。

第 2 章说明本书后续实验相关的环境和实验中的注意事项，并详细介绍后续实验中将会使用的微服务架构应用，以及微服务的容器化构建。

第 3 章简单介绍 Vagrant 及其使用方法。使用 Vagrant 可以快速创建实验环境，这对于我们后续实验环境的准备提供了非常大的便利，接着对创建实验环境的场景进行模拟，帮助读者熟悉 Vagrant 的使用流程。

第 4 章介绍如何使用 Kubeadm 快速创建一个多节点的 Kubernetes 集群。Kubernetes 集群是后续部署 Istio 的基础。

第 5 章介绍如何以官方示例的方式部署一个包含完整功能的 Istio 集群，以及如何以最小资源的方式部署一个能满足大部分实验场景要求的 Istio 集群。此外，还简单介绍了 Istio 中常用的资源，以及常用的 istioctl、kubectl 命令。

第 6 章介绍微服务架构应用如何部署在 Kubernetes 集群中，以及如何访问部署在 Kubernetes 集群中的服务，还简单介绍了如何在 Istio 集群中部署和对外暴露服务。

第 7 章介绍 Istio 提供的服务流量管理功能，包括管理网格的入口和出口流量，根据请求进行流量拆分，如何借助 Istio 实现 A/B 测试和灰度发布等功能。

第 8 章介绍如何让部署在 Istio 中的服务更具弹性，包括负载均衡、连接池、服务健康检测、服务熔断、服务超时、服务重试、服务限流功能的配置。

第 9 章介绍服务故障注入的相关功能，提前给服务注入故障，可以测试服务在故障中的稳定性，提前发现问题并修复问题。

第 10 章介绍服务间通信加密和服务间访问权限的控制。Istio 提供了双向 TLS 进行服务

间的通信加密，使用 RBAC 来实现细粒度的服务访问权限控制。

第 11 章介绍如何提升服务的可观测性。在 Istio 中通过简单的配置，就可以实现服务的指标收集、日志收集。通过传递指定的服务请求头，就可以轻松实现服务的调用链追踪功能，这不但增强了服务的可观测性，还大大减轻了运维人员和开发人员的负担。

第 12 章介绍 Istio 部署后的维护工作。通过部署开源的第三方仪表板工具，我们可以更方便地管理 Istio。接着介绍如何在不停机的情况下升级 Istio，如何使用 Helm 以定制化的方式部署 Istio，以及当 Istio 出现故障时应该如何排查并解决问题。最后介绍了在 Istio 中一个请求从发出到响应的整个流程。

第 13 章介绍一些不适合放在其他章节的 Istio 功能，包括跨域、跳转、TCP 路由、TLS 路由等，以及如何在 Gateway 上启用 HTTPS，如何为部署在 Istio 中的服务开启健康检查功能，如何使用 Envoy 代理 Ingressgateway 来实现把集群内的服务暴露给集群外部使用。最后还简单介绍了 Mixer 和 Adapter 模型。

本书源码

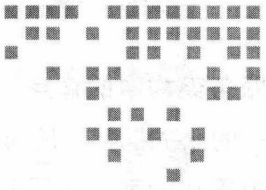
本书所有示例代码都放在 GitHub 上，地址为 https://github.com/mgxian/istio-lab，读者可以查看或下载。由于部署 Kubernetes 集群和 Istio 的过程中会涉及比较多的命令，我也把相关的命令放在了源码的 cmd 目录下。

由于作者水平和时间有限，书中难免会有一些纰漏和错误，欢迎读者及时指正。非常希望和大家一起学习和讨论服务网格与 Istio，并共同推动服务网格和 Istio 在国内的发展。可以通过电子邮件 will835559313@163.com 联系到我。

致谢

感谢所有在本书撰写、出版过程中给予过帮助的人。这里要特别感谢机械工业出版社的吴怡编辑，没有她的鼎力相助，就没有本书。同时也要感谢我的家人和朋友，没有他们的支持和理解，我不可能在有限的时间里完成本书。最后要感谢阅读本书的读者，非常感谢大家的支持！

目　录 *Contents*



第 1 章 Chapter 1

服务网格与 Istio

本章介绍服务网格的由来，以及服务网格给服务开发部署带来什么样的变化，并介绍一个成熟的开源服务网格实现——Istio，这也是本书主要学习的服务网格。通过本章，我们可以了解 Istio 的架构设计，了解 Istio 实现了哪些服务网格功能，从而为后面的学习和实验打下基础。

1.1 服务网格简介

服务网格出现的大环境如下：

- **容器技术的广泛应用**。由于 Docker 的出现，容器技术得到更广泛的认可和应用，各种服务于容器的工具如雨后春笋般涌现，出现了众多容器部署、容器集群、容器编排等平台，例如 Swarm、Mesos、Kubernetes。由于容器具备轻量级、启动速度快、性能损失小、扩容缩容快、开发与生产环境统一等特性，越来越多的公司开始尝试使用容器来部署服务，容器技术的飞速发展也大大加速了微服务的应用。
- **微服务的快速流行**。随着近几年云计算的飞速发展，公有云也越来越成熟，微服务架构模式在大公司的兴起，特别是在 Netflix、亚马逊等公司的大规模实践，使得越来越多的公司开始尝试使用微服务架构来重构应用。当微服务的服务数量越来越大时，微服务间的服务通信也越来越重要，我们所看到的一个应用，有可能背后需要协调成百上千个微服务来处理用户的请求。随着服务数和服务实例数的不断增长，服务可能上线下线，服务实例也可能出现上线下线和宕机

的情况，服务之间的通信变得异常复杂，每个服务都需要自己处理复杂的服务间通信。

- **目前微服务架构中的痛点。**面对复杂的服务间通信问题，一般的解决方案是为服务开发统一的服务框架，所有服务依赖于服务框架开发，所有服务间通信、服务注册、服务路由等功能都由底层服务框架来实现，这样做固然可以在某种程度上解决服务间通信的问题，但是由于底层服务框架的限制，业务人员可能无法基于实际情况选择合适的技术栈；由于所有服务都依赖于底层的服务框架代码库，当框架代码需要更新时，业务开发人员可能并不能立即更新服务框架，导致服务框架整体升级困难。后来 Netflix 开源了自己的微服务间通信组件，之后被 Spring Cloud 集成到了一起，组成了 Java 语言的通用微服务技术栈，而其他编程语言可能并没有如此强大功能的开源组件，只能继续饱受微服务间通信的各种痛。

基于以上服务间通信出现的问题，有人开始思考：能不能把服务间的复杂通信分层并下沉到基础设施层，让应用无感知呢？答案是肯定的。于是服务网格开始渐渐浮出水面，越来越多的人看到了服务网格的价值，尝试把服务网格应用于微服务实践中。

1.1.1 服务网格的概念与特点

服务网格（service mesh）这个概念来源于 Buoyant 公司的 CEO Willian Morgan 的文章“What's a service mesh？ And why do I need one？”。服务网格是一个专注于处理服务间通信的基础设施层，它负责在现代云原生应用组成的复杂服务拓扑中可靠地传递请求。在实践中，服务网格通常是一组随着应用代码部署的轻量级网络代理，而应用不用感知它的存在。

服务网格的特点如下：

- 轻量级的网络代理。
- 应用无感知。
- 应用之间的流量由服务网格接管。
- 把服务间调用可能出现的超时、重试、监控、追踪等工作下沉到服务网格层处理。

服务网格的原理大致如图 1-1 所示，深色部分代表应用，浅灰色部分代表服务网格中轻量级的网络代理，代理之间可以相互通信，而应用之间的通信完全由代理来进行。如果只看代理部分，可以看到一个网状结构，服务网格由此得名。

服务网格一般由数据平面（data plane）和控制平面（control plane）组成，数据平面负责在服务中部署一个称为“边车”（sidecar）的请求代理，控制平面负责请求代理之间的交互，以及用户与请求代理的交互。服务网格的基本架构如图 1-2 所示。

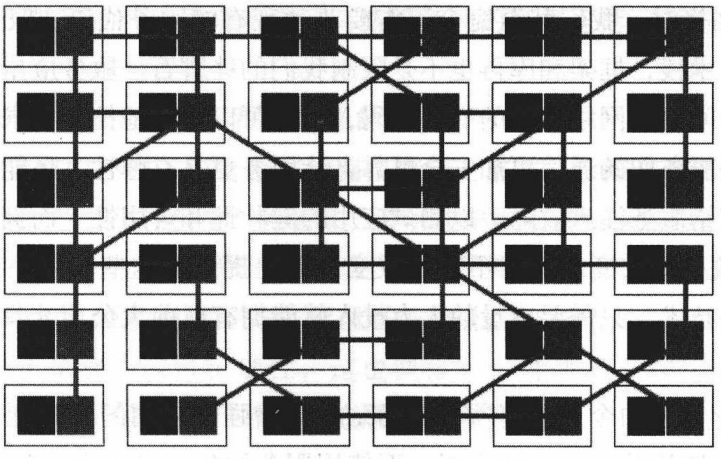


图 1-1　服务网格原理（图片来源：Pattern：Service Mesh）

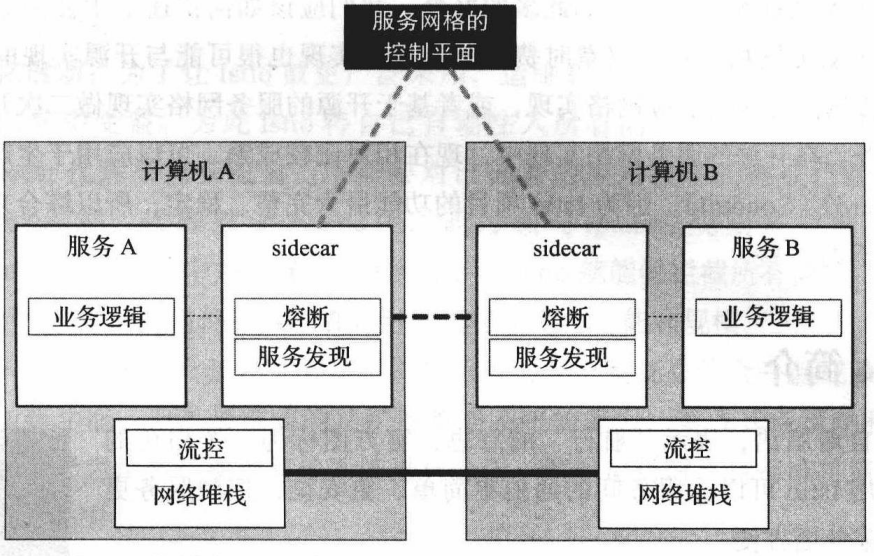


图 1-2　服务网格架构（图片来源：Pattern：Service Mesh）

1.1.2　服务网格的优势

微服务架构流行以后，服务的数量在不断增长。在不使用服务网格的情况下，每个服务都需要自己管理复杂的服务间网络通信，开发人员不得不使用各种库和框架来更好地处理服务间的复杂网络通信问题，这导致代码中包含很多与业务逻辑完全不相关的代码，稍有不慎就有可能给业务带来额外的复杂度和 bug。

当服务规模逐渐变大，复杂度增加，服务间的通信也变得越来越难理解和管理，这就要求服务治理包含很多功能，例如：服务发现、负载均衡、故障转移、服务度量指标收集和监控等。

在使用服务网格时，我们甚至完全不需要改动现有的服务代码，服务开发完全可以使用不同的语言和技术栈，框架和库再也不是限制我们的绊脚石。服务应用代码中将不再需要那些用于处理服务间复杂网络通信的底层代码，我们可以更好地控制请求的流量，对服务进行更好的路由，使服务间的通信更加安全可靠，让服务更具有弹性，还能让我们更好地观测服务，并可以提前给服务注入故障，以测试应用的健壮性和可用性。而拥有这些功能只需要我们的服务做出微小的改变，甚至不需要改变。以上提到的这些功能，在中小规模的公司中，使用服务网格技术，只需要少量的人力投入就能拥有以前大公司才具备的高级服务治理能力。

在云原生大行其道的今天，容器和 Kubernetes 增强了应用的伸缩能力，开发者可以基于此快速地创造出依赖关系复杂的应用；而使用服务网格，开发者不用关心应用的服务管理，只需要专注于业务逻辑的开发，这将赋予开发者更多的创造性。

既然服务网格能给我们带来如此多的好处，我们应该如何快速上手使用服务网格呢？从头开发一个服务网格平台不仅费时费力，最后的实现也很可能与开源实现的功能基本一致，不如直接选择开源的服务网格实现，或者基于开源的服务网格实现做二次开发以适应自己公司的业务。在开源的服务网格实现中，现在相对比较成熟、可以应用于生产环境的只有 Istio 和 Linkerd2（Conduit）。因为 Istio 项目的功能最为完整、稳定，所以综合来看，选择使用 Istio 更为合理一些。

1.2 Istio 简介

Istio 出自希腊语，表示“航行”的意思，官方图标为一个白色的小帆船。使用 Istio 可以让服务间的通信更简单、更安全，控制服务更容易，观测服务更方便。

Istio 是由 Google、IBM、Lyft 公司主导开发的影响力最大的开源服务网格实现，使用 Go 语言编码，由于 Go 语言的性能较好，使得 Istio 性能不错。这个项目由众多代码贡献者完成。

Istio 能有效减少部署的复杂性，可以方便地将 Istio 应用到已有的分布式应用系统架构中，Istio 包含的 API 可以方便地集成任何日志平台、监控平台、数据收集平台和访问策略系统。Istio 有丰富的功能，可以帮助开发者更加高效地运行一个分布式微服务架构，并提供一个统一的方式来保护、连接、监控微服务。Istio 提供了一个完整的解决方案，可以满足多样化的分布式微服务的需求。

1. Istio 的主要功能特性

Istio 可以让你轻松部署一个服务网格，而不需要在服务代码中做任何改变。只需要在

你的环境中部署一个特殊的代理用来拦截所有微服务间的网络通信，就可以通过控制平面配置和管理 Istio。Istio 的功能特性如下：

- HTTP、gRPC、WebSocket、TCP 流量的自动负载均衡。
- 细粒度的流量路由控制，包含丰富的路由控制、重试、故障转移和故障注入。
- 可插拔的访问控制策略层，支持 ACL、请求速率限制和请求配额。
- 集群内度量指标，日志和调用链的自动收集，管理集群的入口、出口流量。
- 使用基于身份的认证和授权方式来管理服务间通信的安全。

由于 Istio 提供了足够多的可扩展性，这也使得 Istio 能满足多样化的需求。基于 Istio 你完全可以搭建出一套适合自己公司基础设施层的服务网格。

2. Istio 的设计目标

Istio 的架构设计中有几个关键目标，这些目标对于系统应对大规模流量和高性能地进行服务处理至关重要：

- **最大化透明**：为了让 Istio 被更广泛采用，运维和开发人员只需要付出很少的代价就可以从中受益。为此 Istio 将自己自动注入所有的网络路径的服务中。Istio 使用 Sidecar 代理来捕获流量，不需要对已部署的应用程序代码进行任何改动。在 Kubernetes 中，代理被注入 Pod 中，通过编写 iptables 规则来捕获流量。注入 Sidecar 代理到 Pod 中并且修改路由规则后，Istio 就能够拦截所有流量。这个原则也适用于性能，所有组件和 API 在设计时都必须考虑性能和规模。
- **增量**：随着运维人员和开发人员越来越依赖 Istio，系统必然会一起成长。Istio 会继续添加新功能，但是最重要的是扩展策略系统的能力，集成其他策略和控制来源，并将网格行为信号传播到其他系统进行分析。策略运行时支持标准扩展机制以便插入其他服务中。
- **可移植性**：Istio 必须支持以最小的代价在任何云和本机环境上运行。将 Istio 上的服务进行迁移也是可行的。
- **策略一致性**：在服务间的 API 调用中，策略的应用使得可以对网格间行为进行全面的控制，但对于不需要在 API 级别表达的资源来说，对资源应用策略也同样重要。例如，将配额应用到 ML 训练任务消耗的 CPU 数量上，比将配额应用到启动这个工作的调用上更为有用。因此，策略系统作为独特的服务来维护，具有自己的 API，而不是将其放到 Sidecar 代理中，这容许服务根据需要直接与其集成。

1.3　Istio 的架构设计

Istio 的架构设计在逻辑上分为数据平面和控制平面：