|  |
| --- |
| **毕 业 论 文** |
|  |
|  |
| **微服务架构之服务网格的设计与应用** |

|  |  |  |
| --- | --- | --- |
| **学生姓名** | 刘晓璐 |  |
| **学 号** | 030418111415 |  |
| **专业班级** | 10班 |  |
| **指导教师** | 谭老师 |  |
| **完成日期** | 2022年5月 |  |

**微服务架构之服务网格的研究与应用**

摘要

随着互联网和软件技术的不断发展，软件系统架构从单体架构演进到如今的面向服务架构。近年来，面向服务的架构方式由于其灵活、解耦、规范化等诸多优势，被广泛应用于大型企业系统架构。微服务架构去中心化、高度自治、自动化等特点进一步对系统架构进行解耦，并且和敏捷开发、DevOps文化高度契合，可以使用较低的成本，满足企业软件体系结构发展需要，提高研发效率，规范服务治理。在Spring Cloud，Dubbo等微服务治理框架的支持下，微服务框架成为了目前最主流的分布式应用解决方案，然而目前市面上这些主流的微服务框架都是需要与业务应用耦合在一起使用的，随着微服应用的规模和复杂度不断增长，微服务之间的网络通信以及服务治理变得越来越难以管理，并且整个微服务应用的运维成本会直线上升。为了解决这个问题，本文通过对国内外服务网格技术的研究，来解决微服务之间的治理问题，目标是使开发人员更专注于业务的开发，像微服务之间的注册与发现、RPC调用、负载均衡等服务治理操作对开发人员透明。并通过对目前市面上主流的服务网格框架Istio学习研究，完成仿豆瓣电影详情页的服务网格应用实战，实现电影详情、评分、推荐等功能，并基于Istio实现微服务的灰度发布。

关键词：微服务；服务网格；Istio；灰度发布；仿豆瓣

**Research And Implement of Micro-Services Architecture Service Mesh**

**ABSTRACT**

With the continuous development of Internet and software technology, software system architecture has evolved from single architecture to service-oriented architecture.  In recent years, service-oriented architecture has been widely used in large-scale enterprise system architecture due to its advantages of flexibility, decoupling and standardization.  Decentralization, high autonomy, automation and other features of microservice architecture further decouple the system architecture, and it is highly compatible with agile development and DevOps culture. It can meet the development needs of enterprise software architecture with low cost, improve research and development efficiency, and standardize service governance.  Supported by Spring Cloud, Dubbo and other microservice governance frameworks, microservice frameworks have become the most mainstream distributed application solutions at present. However, these mainstream microservice frameworks on the market need to be used together with business applications. With the increasing scale and complexity of microservice applications,  Network communication and service governance between microservices are becoming more and more difficult to manage, and the operation and maintenance cost of the entire microservice application will rise sharply.  In order to solve this problem, this paper aims to solve the governance problem between microservices by studying the service grid technology at home and abroad. The goal is to make developers more focused on business development, and the service governance operations such as registration and discovery between microservices, RPC call and load balancing are transparent to developers.  And through the study of the current mainstream service grid framework Istio on the market, to complete the application practice of the service mesh of the details page of the imitation Douban movie, to achieve movie details, scoring, recommendation and other functions, and achieve the grayscale release of micro-services based on Istio.

**Key Words:** microservice; service mesh; istio; grayscale release; imitation of douban

目录

[摘要 I](#_Toc96)

[ABSTRACT II](#_Toc28547)

[1. 绪论 5](#_Toc20998)

[1.1 技术架构演进 5](#_Toc30503)

[1.1.1 发展历史时间轴 5](#_Toc19573)

[1.1.2 单机小型机时代 6](#_Toc15982)

[1.1.3 垂直拆分 6](#_Toc4875)

[1.1.4 集群化负载均衡架构 7](#_Toc21769)

[1.1.5 服务化改造架构 8](#_Toc20800)

[1.1.6 服务治理 9](#_Toc20727)

[1.1.7 微服务时代 10](#_Toc1011)

[1.1.8 服务网格新时期 （Service Mesh） 11](#_Toc26375)

[1.2 目前微服务架构面临的一些挑战 14](#_Toc21084)

[2. Service Mesh 16](#_Toc30876)

[2.1 什么是Service Mesh 16](#_Toc6994)

[2.2 Service Mesh产品 16](#_Toc18493)

[2.2.1 CNCF 16](#_Toc16903)

[2.2.2 Linkerd 17](#_Toc21496)

[2.2.3 Envoy 18](#_Toc23916)

[2.2.4 Istio 18](#_Toc2985)

[2.2.5 Conduit 19](#_Toc18387)

[2.2.6 国内产品 19](#_Toc2719)

[3. Istio 19](#_Toc14479)

[3.1 Istio架构 19](#_Toc30514)

[3.2 为什么使用Istio? 20](#_Toc8095)

[3.3 核心特性 21](#_Toc74)

[3.3.1 流量管理 21](#_Toc31655)

[3.3.2 安全 21](#_Toc28649)

[3.3.3 可观察性 22](#_Toc25798)

[3.3.4 平台支持 22](#_Toc17204)

[4. Istio快速入门 22](#_Toc6924)

[4.1 搭建Kubernetes集群 22](#_Toc31630)

[4.1.1 环境准备 23](#_Toc7574)

[4.1.2 前置工作 23](#_Toc14787)

[4.1.3 搭建集群 25](#_Toc14238)

[4.2 搭建Istio环境 27](#_Toc31574)

[4.2.1 下载Istio 27](#_Toc6916)

[4.2.2 安装Istio 28](#_Toc21355)

[4.3 Bookinfo示例 30](#_Toc7188)

[4.3.1 应用说明 30](#_Toc1003)

[4.3.2 应用部署 31](#_Toc157)

[4.3.3 启动应用服务 32](#_Toc16592)

[4.3.4 确定Ingress的IP 33](#_Toc9361)

[4.3.5 应用默认目标规则 35](#_Toc30389)

[4.4 体验Istio 36](#_Toc2240)

[4.4.1 按照版本路由 36](#_Toc30506)

[4.4.2 按照不同用户身份路由 39](#_Toc17528)

[5. 项目实战 41](#_Toc16472)

[5.1 项目说明 41](#_Toc22795)

[5.2 搭建Neo4J 43](#_Toc750)

[5.3 创建工程 45](#_Toc7810)

[5.4 movie-common 53](#_Toc24702)

[5.5 movie-info 53](#_Toc31507)

[5.6 movie-recommend 57](#_Toc28075)

[5.7 movie-rating 60](#_Toc25358)

[5.8 movie-web 64](#_Toc7894)

[5.9 制作docker镜像 69](#_Toc19800)

[5.10 编写Istio相关文件 71](#_Toc15153)

[5.11 实施部署 79](#_Toc26808)

[6. 结论与展望 81](#_Toc4445)

[6.1 结论 81](#_Toc120)

[6.2 展望 82](#_Toc16889)

1. **绪论**
   1. 技术架构演进
      1. 发展历史时间轴

随着互联网技术的不断发展，技术架构由最初的单机小型机时代发展到如今的服务网格新时期。如图1-1-1：

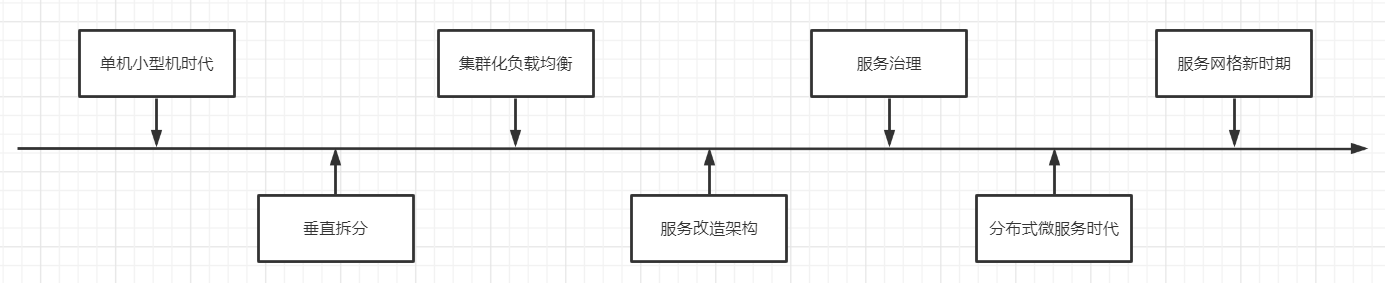


图1-1-1

* + 1. 单机小型机时代

第一个计算机网络诞生1969年，也就是美军的阿帕网，阿帕网能够实现与其它计算机进行联机操作，但是早期仅仅是为了军事目的而服务，2000年初，中国的网名大约890万，很多⼈都不知道互联网为何物，因此⼤多数服务业务单一且简单，采用典型的单机+数据库模式，所有的功能都写在一个应用里并进行集中部署。如图1-1-2所示：

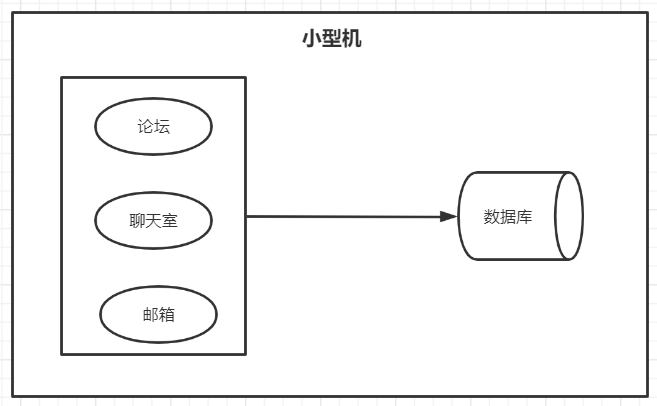


图 1-1-2

说明：论坛业务、聊天室业务、邮箱业务全部都耦合在一台⼩型机上面，所有的业务数据也都存储在一台数据库上。

* + 1. 垂直拆分

随着应用的日益复杂与多样化，开发者对系统的容灾，伸缩以及业务响应能力有了更高的要求，如果⼩型机和数据库中任何一个出现故障，整个系统都会崩溃，若某个板块的功能需要更新，那么整个系统都需要重新发布，显然，对于业务迅速发展的万物互联网时代是不允许的。如何保障可用性的同时快速响应业务的变化，需要将系统进行拆分，将上面的应用拆分出多个子应用。如图1-1-3所示：

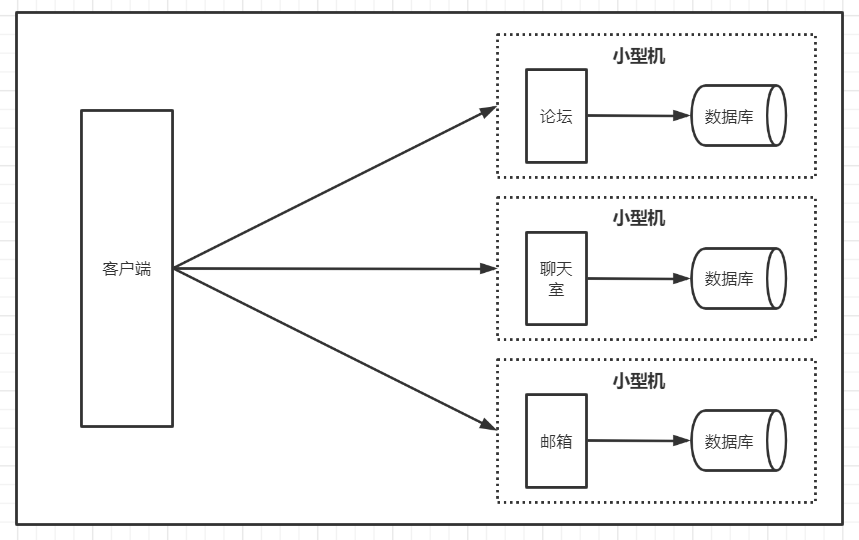


图 1-1-3

优点：应用间进行了解耦，系统容错提高了，也解决了独立应用发布的问题。

应用垂直拆分解决了应用发布的问题，但是随着用户数量的增加，单机的计算能力依旧是杯水车薪。

* + 1. 集群化负载均衡架构

随着用户量越来越大，就意味着需要更多的⼩型机，但是⼩型机价格昂贵，操作维护成本高。此时更优的选择是采用多台PC机部署同一个应用的方案，但是此时就需要对这些应用做负载均衡，因为客户端不知道请求会落到哪一个后端PC应用上的。负载均衡可以分为硬件层面和软件层面。硬件层面有F5，软件负载层面有LVS、Nginx、Haproxy，负载均衡的思想是对外暴露一个统一的接口，根据用户的请求进行对应规则转发，同时负载均衡还可以做限流等等，有了负载均衡之后，后端的应用可以根据流量的大小进行动态扩容，我们称之为“水平扩展”。如图1-1-4所示：

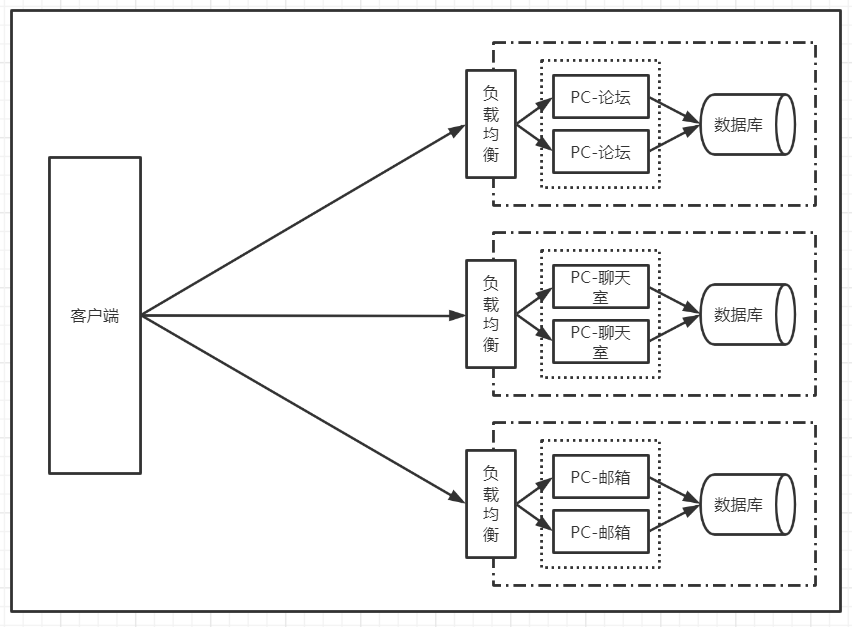


图 1-1-4

阿里巴巴在2008提出去“IOE”，也就是IBM⼩型机、Oracle数据库，EMC存储，全部改成集群化负载均衡架构，在2013年支付宝最后一台IBM⼩型机下线，

优点：通过水平扩展，增强了系统的并发能力。

* + 1. 服务化改造架构

虽然系统经过了垂直拆分，但是拆分之后发现在论坛和聊天室中有重复的功能，比如，用户注册、发邮件等等，一旦项目大了，集群部署多了，这些重复的功能无疑会造成资源浪费，所以会把重复功能抽取出来，名字叫“XX服务（Service）”。如图1-1-5所示：

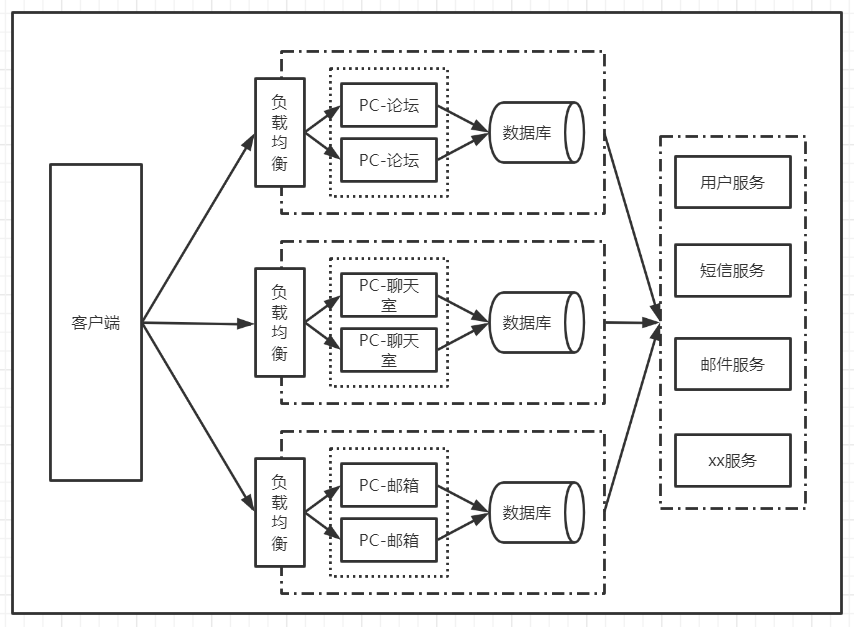


图 1-1-5

为了解决服务跟服务如何相互调用，需要一个程序之间的通信协议，所以就有了远程过程调用（RPC），作用就是让服务之间的程序调用变得像本地调用一样的简单，优点：在前面的架构之上解决了业务重用的问题。

* + 1. 服务治理

随着业务的增大，基础服务越来越多，调用网的关系由最初的几个增加到几十上百，造成了调用链路错综复杂,需要对服务进行治理。服务治理要求如下：

* 当我们服务节点数几十上百的时候，需要对服务有动态的感知，引⼊了注册中⼼；
* 当服务链路调用很长的时候如何实现链路的监控；
* 单个服务的异常，如何能避免整条链路的异常（雪崩），需要考虑熔断、降级、限流。

典型框架比如有：Dubbo，默认采用的是Zookeeper作为注册中⼼。如图1-1-6所示：

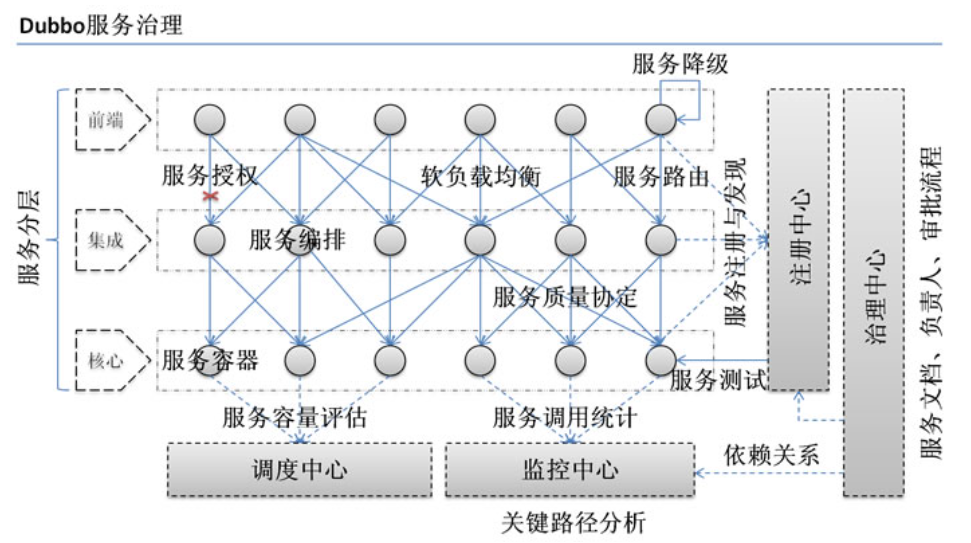


图 1-1-6

* + 1. 微服务时代

微服务是在2012年提出的概念，微服务的希望的重点是一个服务只负责一个独立的功能。拆分原则，任何一个需求不会因为发布或者维护而影响到不相关的服务，一切可以做到独立部署运维。比如打车系统，对于微服务来说，需要根据业务进行拆分为乘客服务、司机服务、行程服务、账单服务、支付服务、通知服务等微服务。典型代表：Spring Cloud，相对于传统分布式架构，Spring Cloud使用的是HTTP作为RPC远程调用，配合上注册中⼼Eureka和API网关Zuul，可以做到细分内部服务的同时又可以对外暴露统一的接⼝，让外部对系统内部架构无感，此外Spring Cloud的config组件还可以把配置统一管理。如图1-1-7所示：

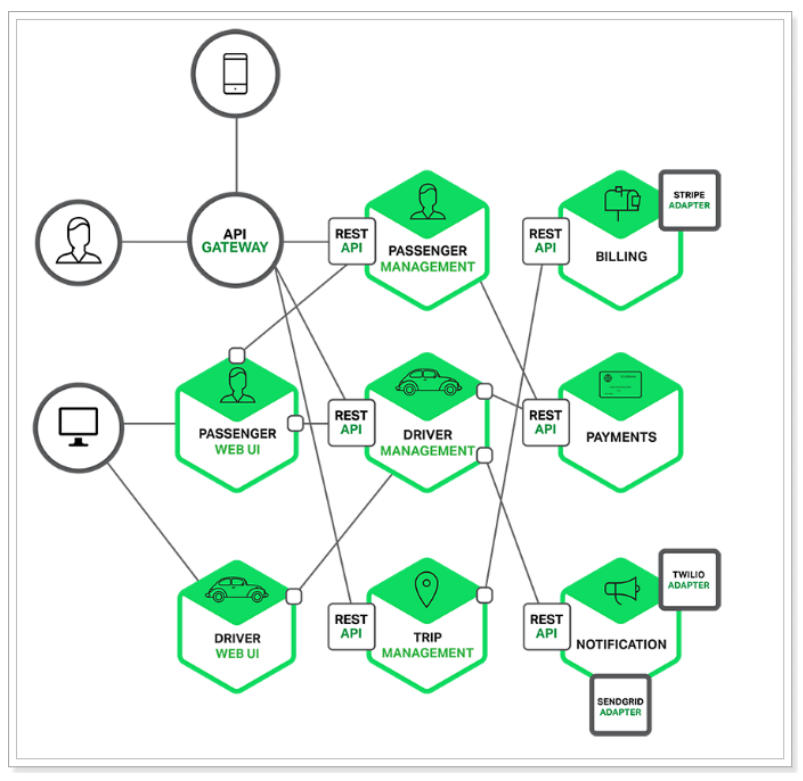


图 1-1-7

Spring Cloud微服务架构存在的不足：

* Spring Cloud属于侵⼊式框架，在项目中需要添加Spring Cloud Maven依赖，加上Spring Cloud组件注解，写配置，打成jar的时候还必须要把非业务的代码也要融合在一起。
* 微服务中的服务支持不同语言开发，也需要维护不同语言和非业务代码的成本；
* 业务代码开发者应该把更多的精力投⼊到业务熟悉度上，而不应该是⾮业务上，Spring Cloud虽然能解决微服务领域的很多问题，但是学习成本还是较⼤的。
* 互联网公司产品的版本升级是非常频繁的，为了维护各个版本的兼容性、权限、流量等，因为Spring Cloud是“代码侵⼊式的框架”，这时候版本的升级就注定要让非业务代码一起，一旦出现问题，再加上多语言之间的调用，工程师会非常痛苦。
* 我们已经感觉到了，服务拆分的越细，只是感觉上轻量级解耦了，但是维护成本却越高了。
  + 1. 服务网格新时期 （Service Mesh）

Service Mesh主要解决的问题就希望开发⼈员对于业务的聚焦，服务发现、服务注册、负载均衡等对于开发⼈员透明，可以更加专注业务逻辑的实现。如果将为微服务提供通信服务的这部分逻辑从应用程序进程中抽取出来，作为一个单独的进程进行部署，并将其作为服务间的通信代理，可以得到如下图1-1-8a所示的架构：

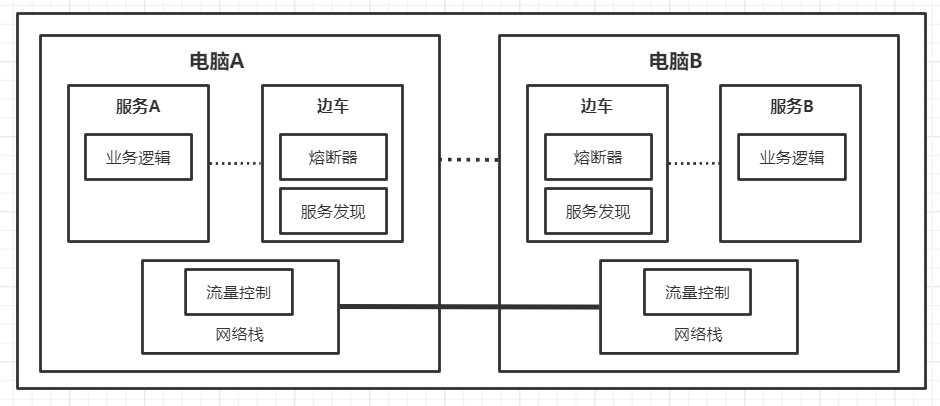


图1-1-8a

Sidecar，翻译成中⽂是边车，非常的形象。如图1-1-8b：



图1-1-8b

当服务⼤量部署时，随着服务部署的Sidecar代理之间的连接形成了一个如下图1-1-8c所示的网格，该网格成为了微服务的通讯基础设施层，承载了微服务之间的所有流量，被称之为Service Mesh（服务网格）。

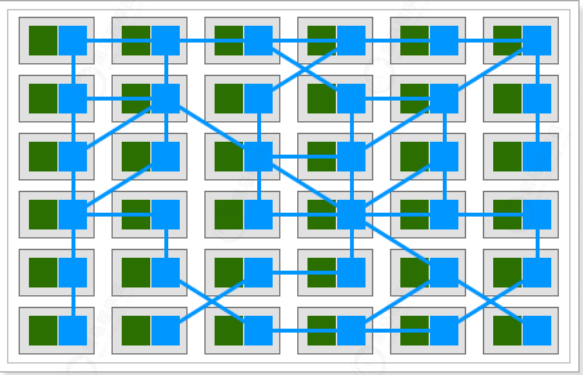


图1-1-8c

服务网格中有数量众多的Sidecar代理，如果对每个代理分别进行设置，工作量将非常巨大。为了更方便地对服务网格中的代理进行统一集中控制，在服务网格上增加了控制面板组件。如图1-1-8d所示：

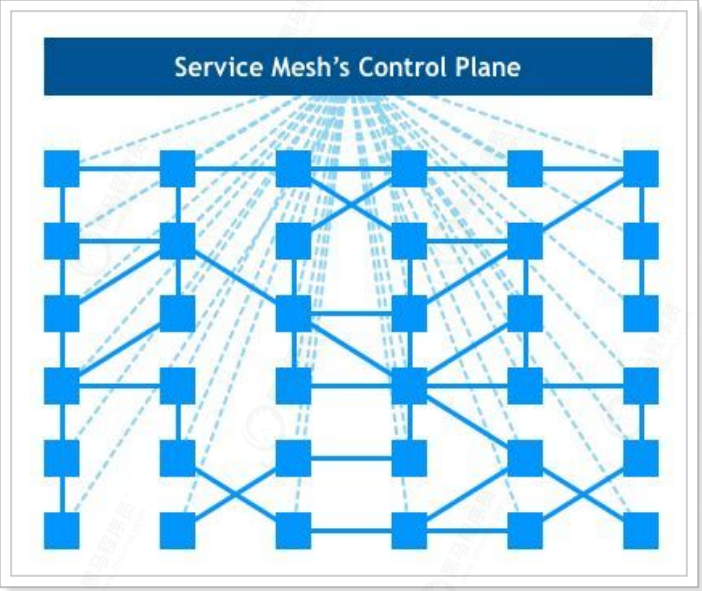


图1-1-8d

* 1. 目前微服务架构面临的一些挑战

目前，微服务的架构方式在企业中得到了极大的发展，主要原因是其解决了传统的单体架构中存在的问题。如图1-2所示：

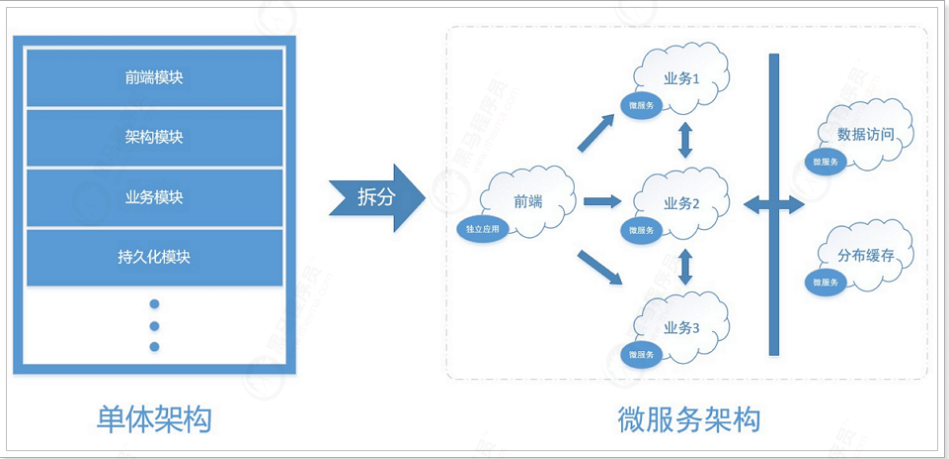


图1-2

当单体架构拆分成微服务架构就可以高枕无忧了吗？ 显然不是的。

微服务架构体系中同样也存在很多的挑战，比如：

* 原来的单个应用拆分成了许多分散的微服务，它们之间相互调用才能完成一个任务，而一旦某个过程出错（组件越多，出错的概率也就越⼤），就非常难以排查。
* 如果用户请求的响应太慢，我们就需要知道到底哪些地方⽐较慢？整个链路的调用各阶段耗时是多少？哪些调用是并发执行的，哪些是串行的？这些问题需要我们能非常清楚整个集群的调用以及流量情况。
* 微服务拆分成这么多组件，如果单个组件出错的概率不变，那么整体有地方出错的概率就会增⼤。服务调用的时候如果没有错误处理机制，那么会导致非常多的问题。
* 应用数量的增多，对于日常的应用发布来说也是个难题。应用的发布需要非常谨慎，如果应用都是 一次性升级的，出现错误会导致整个线上应用不可用，影响范围太⼤。
* 很多情况我们需要同时存在不同的版本，使用AB测试验证哪个版本更好。
* 如果版本升级改动了API，并且互相有依赖，那么我们还希望能自动地控制发布期间不同版本访问不同的地址。这些问题都需要智能的流量控制机制。
* 为了保证整个系统的安全性，每个应用都需要实现一套相似的认证、授权、HTTPS、限流等功能。

那么，Service Mesh就是为了解决以上问题才出现的。

1. **Service Mesh**
   1. 什么是Service Mesh

服务网格用来描述组成这些应用程序的微服务网络以及它们之间的交互。随着服务网格的规模和复杂性不断的增⻓，它将会变得越来越难以理解和管理。它的需求包括服务发现、负载均衡、故障恢复、度量和监控等。服务网格通常还有更复杂的运维需求，比如A/B 测试、金丝雀发布、速率限制、访问控制和端到端认证。如图2-1所示：

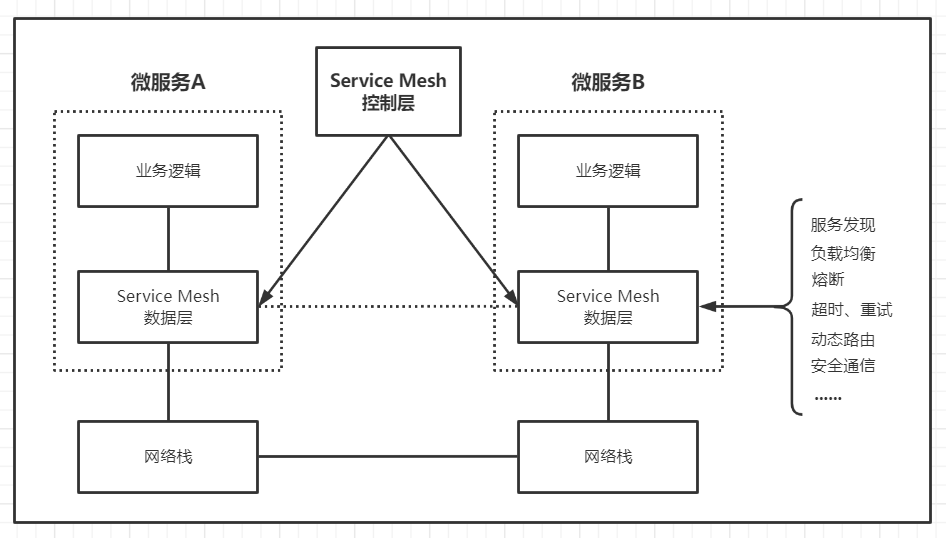


图2-1

* 1. Service Mesh产品
     1. CNCF

CNCF是一个开源软件基金会，致力于使云原生计算具有普遍性和可持续性。云原生计算使用开源软件技术栈将应用程序部署为微服务，将每个部分打包到自己的容器中，并动态编排这些容器以优化资源利用率。云原生技术使软件开发⼈员能够更快地构建出色的产品。官网：https://www.cncf.io/，常用的已经毕业的云原生项目：

* Kubernetes
  + Kubernetes是世界上最受欢迎的容器编排平台也是第一个CNCF项目。 Kubernetes帮助用户构建、扩展和管理应用程序及其动态生命周期。
* Prometheus
  + Prometheus为云原生应用程序提供实时监控、警报包括强大的查询和可视化能力，并与许多流⾏的开源数据导⼊、导出工具集成。
* Jaeger
  + Jaeger是由Uber开发的分布式追踪系统，用于监控其⼤型微服务环境。 Jaeger被设计为具有高度可扩展性和可用性，它具有现代 UI，旨在与云原生系统（如OpenTracing、Kubernetes和Prometheus）集成。
* Envoy
  + Envoy是最初在Lyft创建的Service Mesh（服务网格），现在用于Google、Apple、Netflix等公司内部。Envoy是用C++编写的，旨在最大限度地减少内存和CPU占用空间，同时提供诸如负载均衡、网络深度可观察性、微服务环境中的跟踪和数据库活动等功能。
* Containerd
  + Containerd是由Docker开发并基于Docker Engine运行时的行业标准容器运行时组件。作为容器生态系统的选择，Containerd通过提供运行时，可以将Docker和OCI容器镜像作为新平台或产品的一部分进行管理。
    1. Linkerd

Linkerd是Buoyant公司2016年率先开源的高性能网络代理程序，是业界的第一款Service Mesh产品，甚至可以说Linkerd的诞生标志着Service Mesh时代的开始，其引领后来Service Mesh的快速发展。其主要用于解决分布式环境中服务之间通信面临的一些问题，比如网络不可靠、不安全、延迟丢包等问题。Linkerd使用Scala语言编写，运行于JVM，底层基于Twitter的Finagle库，并对其做相应的扩展。最主要的是Linkerd具有快速、轻量级、高性能等特点，每秒以最⼩的时延及负载处理万级请求，易于水平扩展，经过生产线测试及验证，可运行任何平台的生产级Service Mesh工具。

* + 1. Envoy

Envoy也是一款高性能的网络代理程序，于2016年10月份由Lyft公司开源，为云原生应用而设计，可作为边界入口，处理外部流量，当然，也作为内部服务间通信代理，实现服务间可靠通信。Envoy的实现借鉴现有生产级代理及负载均衡器，如Nginx、HAProxy、硬件负载均衡器及云负载均衡器的实践经验，同时基于C++编写及Lyft公司生产实践证明，Envoy性能非常优秀、稳定。Envoy既可用作独立代理层运行，也可作为Service Mesh架构中数据平面层，因此通常Envoy跟服务运行在一起，将应用的网络功能抽象化，Envoy提供通用网络功能，实现平台及语言无关。

* + 1. Istio

2017年5月24⽇，Google, IBM和Lyft共同发布Istio的第一个公开版本(0.1)。Istio为一款开源的为微服务提供服务间连接、管理以及安全保障的平台软件，支持运行在Kubernetes、Mesos等容器管理工具，但不限于Kubernetes、Mesos，其底层依赖于Envoy。Istio提供一种简单的方法实现服务间的负载均衡、服务间认证、监控等功能，而且无需应用层代码调整。其控制平面由Pilot、Citadel和Galley组成，数据平面由Envoy实现，通常情况下，数据平面代理Envoy以Sidecar模式部署，使得所有服务间的网络通信均由Envoy实现，而Istio的控制平面则负责服务间流量管理、安全通信策略等功能。

* + 1. Conduit

Conduit于2017年12月发布，作为由Buoyant继Linkerd后赞助的另一个开源项目。Conduit旨在彻底简化用户在Kubernetes使用服务网格的复杂度，提高用户体验，而不是像Linkerd一样针对各种平台进行优化。

* + 1. 国内产品

国内很多团队也已经在着手研究了，这些团队主要分为四类体系：

* 以蚂蚁金服为首的开源系：蚂蚁金服自研的SOFA (Scalable Open Financial Architecture)Mesh在开始的时候走的就是开源路线，他们参考了Istio及Envoy的设计思想，重新实现了自己的Service Mesh系统，旁路网关（Sidecar）基于Go语言，该系统的前身是已经开源的SOFA RPC框架。蚂蚁金服于2018年7月正式将其开源，正式的可以用于生产的框架可能还需要一些时间。
* 以华为为代表的自研系：华为可能在Service Mesh概念出来前就已经有类似的想法了，只是没有抽取出一个公共的概念。无论是华为早期的HSA还是之后出现的CSE Mesher，都是对Service Mesh的探索。CSE Mesher的整个架构都是基于华为自身微服务架构经验研发的，其Sidecar也是用Go语言编写的。如其官方文档所述，其资源占用非常⼩，常规状态下仅为30MB。
* 以腾讯为代表的拿来主义系：腾讯的Tencent Service Mesh对开源的产品（如 Istio）进行定制，强化吸收后再加⼊自身特殊的业务逻辑。腾讯选择的Sidecar是Envoy，使用C++编写，⽐较符合腾讯的技术栈。其公开的技术并不多，仍然以内部⼩范围使用为主。
* 以UCloud为代表的适配系：主要也是依赖开源方案，但不是完全将其产品引⼊，只是对其中几个关键部分添加适配器，以适应企业现有产品，以最⼩的变更成本引⼊Service Mesh体系。

1. **Istio**
   1. Istio架构

实际上Istio就是Service Mesh架构的一种实现，服务之间的通信（比如这里的Service A访问Service B）会通过代理（默认是Envoy）来进行。而且中间的网络协议支持HTTP/1.1，HTTP/2，gRPC或者TCP，可以说覆盖了主流的通信协议。代理这一层，称之为数据平面。控制平面做了进一步的细分，分成了Pilot、Citadel和Galley，它们的各自功能如下：

* Pilot：为Envoy提供了服务发现，流量管理和智能路由（AB测试、金丝雀发布等），以及错误处理（超时、重试、熔断）功能。
* Citadel：为服务之间提供认证和证书管理，可以让服务自动升级成TLS协议。
* Galley：Galley是Istio的配置验证、提取、处理和分发组件。它负责将其余的Istio 组件与从底层平台（例如 Kubernetes）获取用户配置的细节隔离开来。

数据平面会和控制平面通信，一方面可以获取需要的服务之间的信息，另一方面也可以汇报服务调用的Metrics数据，如图3-1所示：

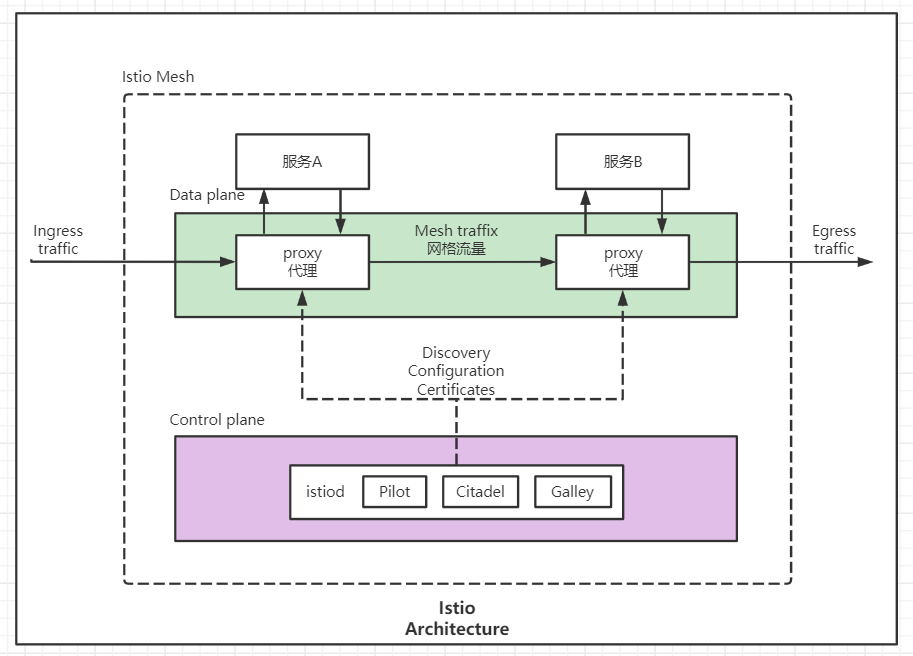


图3-1

* 1. 为什么使用Istio?

通过负载均衡、服务间的身份验证、监控等方法，Istio可以轻松地创建一个已经部署了服务的网络，而服务的代码只需很少更改甚至无需更改。通过在整个环境中部署一个特殊的Sidecar代理为服务添加Istio的支持，而代理会拦截微服务之间的所有网络通信，然后使用其控制平面的功能来配置和管理Istio，这包括：

* 为HTTP、gRPC、WebSocket和TCP流量自动负载均衡。
* 通过丰富的路由规则、重试、故障转移和故障注⼊对流量行为进行细粒度控制。
* 可插拔的策略层和配置API，支持访问控制、速率限制和配额。
* 集群内（包括集群的入口和出口）所有流量的自动化度量、日志记录和追踪。
* 在具有强大的基于身份验证和授权的集群中实现安全的服务间通信。

Istio为可扩展性而设计，可以满足不同的部署需求。

* 1. 核心特性

Istio以统一的方式提供了许多跨服务网络的关键功能。

* + 1. 流量管理

Istio简单的规则配置和流量路由允许您控制服务之间的流量和API调用过程。 Istio简化了服务级属性（如熔断器、超时和重试）的配置，并且让它轻而易举的执行重要的任务（如A/B测试、金丝雀发布和按流量百分比划分的分阶段发布）。有了更好的对流量的可视性和开箱即用的故障恢复特性，就可以在问题产生之前捕获它们，无论面对什么情况都可以使调用更可靠，网络更健壮。

* + 1. 安全

Istio的安全特性解放了开发⼈员，使其只需要专注于应用程序级别的安全。 Istio提供了底层的安全通信通道，并为大规模的服务通信管理认证、授权和加密。有了Istio，服务通信在默认情况下就是受保护的，可以让您在跨不同协议和运行时的情况下实施一致的策略，而所有这些都只需要很少甚至不需要修改应用程序。Istio是独立于平台的，可以与Kubernetes（或基础设施）的网络策略一起使用。但它更强大，能够在网络和应用层面保护pod到pod或者服务到服务之间的通信。

* + 1. 可观察性

Istio健壮的追踪、监控和日志特性让您能够深⼊的了解服务网格部署。通过Istio的监控能力，可以真正的了解到服务的性能是如何影响上游和下游的，而它的定制Dashboard提供了对所有服务性能的可视化能力，并让您看到它如何影响其他进程。Istio的Mixer组件负责策略控制和遥测数据收集。它提供了后端抽象和中介，将一部分Istio与后端的基础设施实现细节隔离开来，并为运维⼈员提供了对网格与后端基础实施之间交互的细粒度控制。所有这些特性都使您能够更有效地设置、监控和加强服务的SLO(Service Level Objectives)。当然，底线是您可以快速有效地检测到并修复出现的问题。

* + 1. 平台支持

Istio独立于平台，被设计为可以在各种环境中运行，包括跨云、内部环境、Kubernetes、Mesos等等。您可以在Kubernetes或是装有Consul的Nomad环境上部署Istio。Istio目前支持：

* Kubernetes上的服务部署
* 基于Consul的服务注册
* 服务运行在独立的虚拟机上

1. **Istio快速入门**
   1. 搭建K8S集群

Istio运行在K8S(kubernetes)平台是最佳的选择，所以我们先搭建K8S环境。

* + 1. 环境准备

在VMware上创建3台Centos7虚拟机，配置如图4-1-1所示：

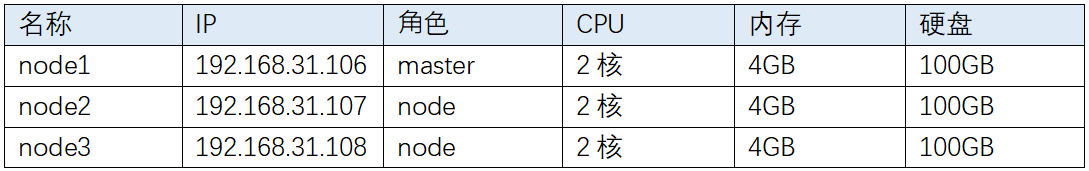


图4-1-1

* + 1. 前置工作

搭建K8S之前，需要做一些前置的准备工作，否则不能完成集群的搭建，分别登陆三台虚拟机执行如下操作：

* 修改主机名为node1，node2，node3,如图4-1-2a：



图4-1-2a

* 更新yum源，并且完成yum update操作,如图4-1-2b：

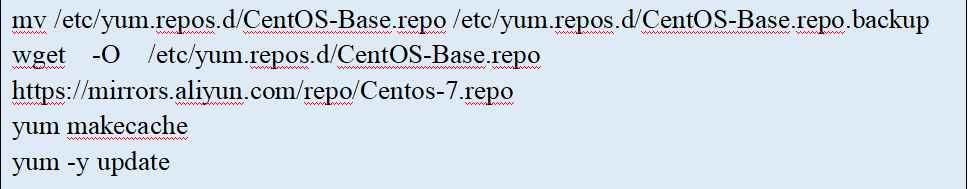


图4-1-2b

* 安装docker服务，如图4-1-2c：

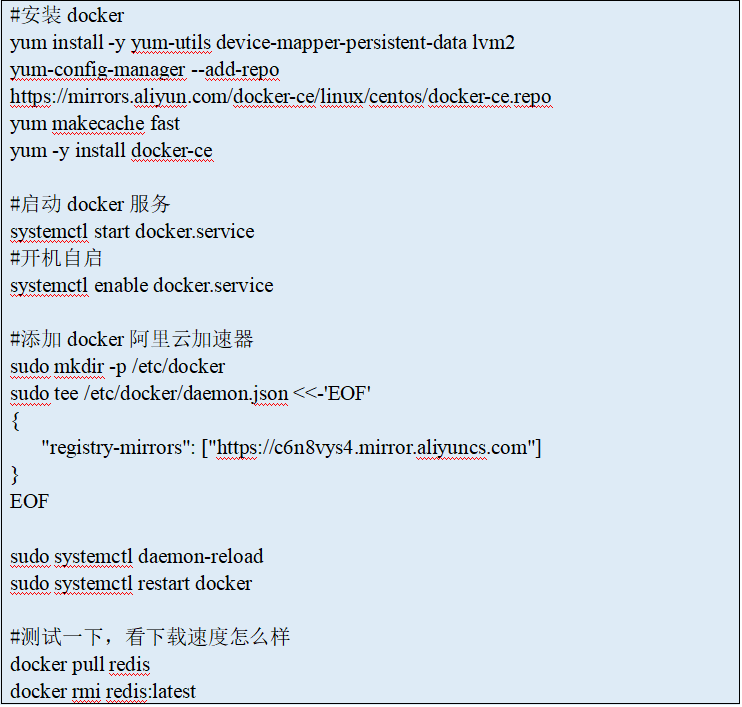


图4-1-2c

* 关闭防火墙，并添加host映射，如图4-1-2d：

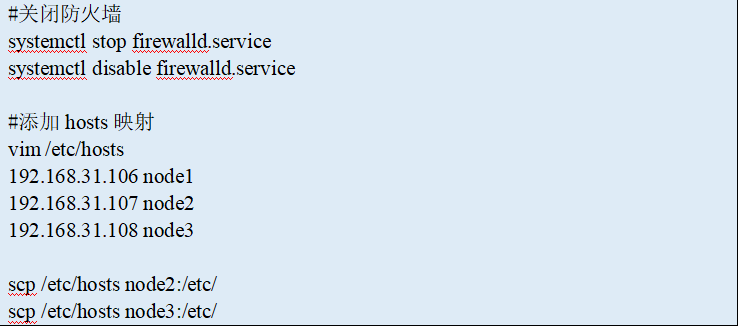


图4-1-2d

* 设置虚拟机免密登录，如图4-1-2e：

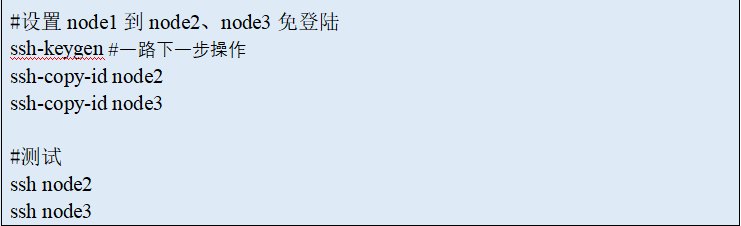


图4-1-2e

* + 1. 搭建集群

开始搭建K8S集群，搭建步骤如下：

* 修改系统参数，禁用SELinux以及swap,如图4-1-3a：

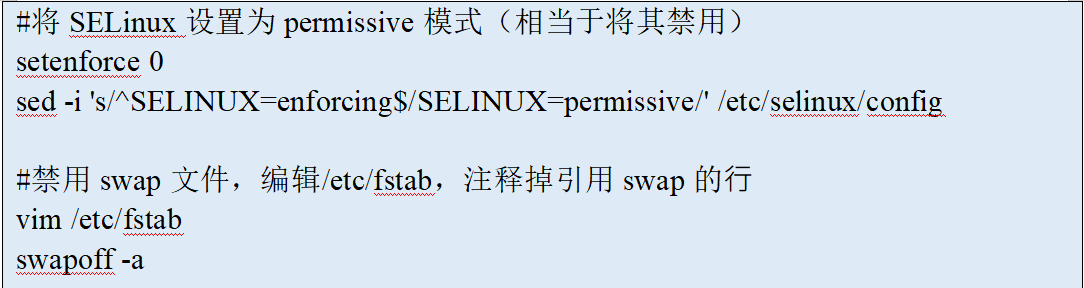


图4-1-3a

* 设置网桥参数，如图4-1-3b：

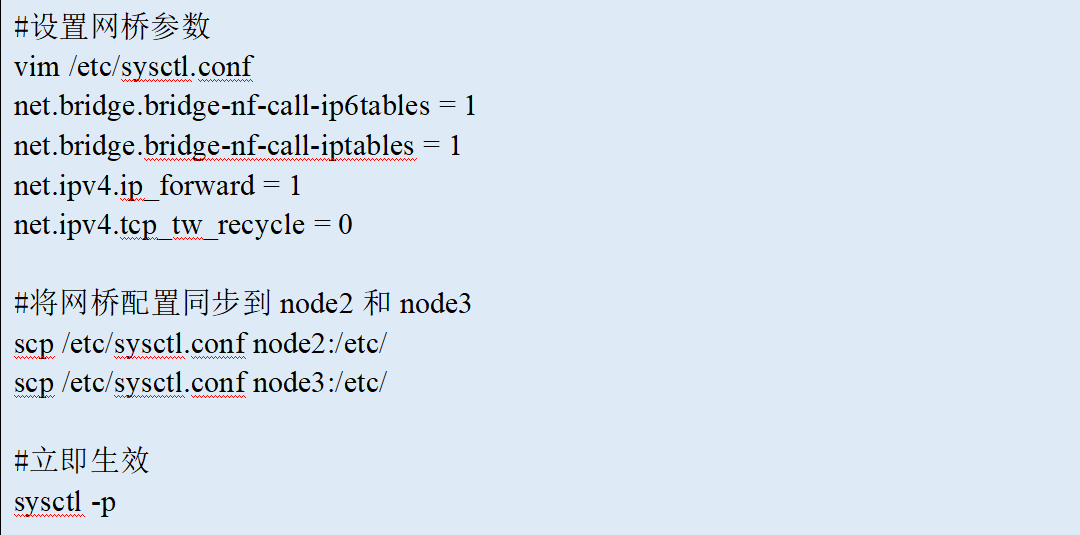


图4-1-3b

* 安装kubectl，如图4-1-3c：



图4-1-3c

* 初始化kubectl，如图4-1-3d：



图4-1-3d

* 设置网络并测试kubectl，如图4-1-3e：

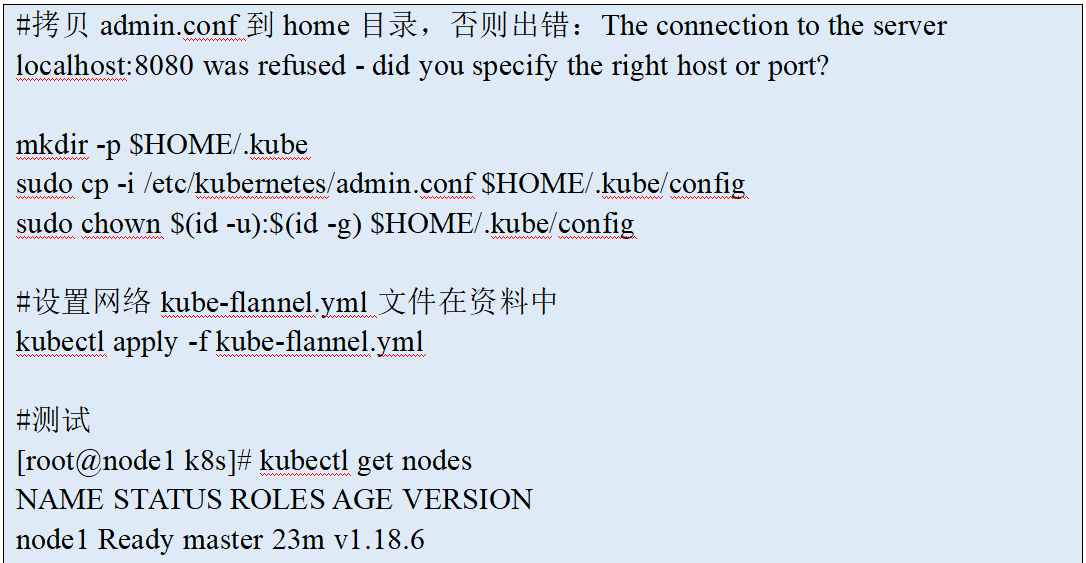


图4-1-3e

* 将node2、node3加⼊到集群,如图4-1-3f：



图4-1-3f

经过以上步骤，我们已经安装好了K8S集群环境，查看正在运行的pod，执行“kubectl get pod --all-namespaces -o wide”,结果如图4-1-3所示：



图4-1-3

* 1. 搭建Istio环境
     1. 下载Istio

下载Istio，下载内容将包含：安装文件、示例和istioctl命令行工具，操作如下：

* 访问https://istio.io/downloadIstio页面下载与您操作系统对应的安装文件；
* 切换到Istio安装目录下，可看到如下内容：
* samples/ 目录下，有示例应用程序
* bin/ 目录下，包含istioctl的客户端文件，istioctl工具用于手动注⼊Envoy sidecar代理；
* 将istioctl客户端路径增加到path环境变量中。
  + 1. 安装Istio

使用demo配置文件安装Istio，安装步骤如下：

* 安装demo配置，执行命令“istioctl manifest apply --set profile=demo”；
* 为了验证是否安装成功，需要先确保以下Kubernetes服务正确部署，然后验证除jaegeragent服务外的其他服务，是否均有正确的CLUSTER-IP，执行命令“kubectl get svc -n istio-system”；

如果集群运行在一个不支持外部负载均衡器的环境中（例如：minikube）， istioingressgateway的EXTERNAL-IP将显示为状态。请使用服务的NodePort或端口转发来访问网关。请确保关联的Kubernetes pod已经部署，并且STATUS为Running，通过命令“kubectl get pods -n istio-system”查看。

* 1. Bookinfo示例
     1. 应用说明

这个示例部署了一个用于演示多种Istio特性的应用，该应用由四个单独的微服务构成。这个应用模仿在线书店的一个分类，显示一本书的信息。页面上会显示一本书的描述，书籍的细节（ISBN、⻚数 等），以及关于这本书的一些评论。Bookinfo应用分为四个单独的微服务：

* productpage：这个微服务会调用details和reviews两个微服务，用来生成页面；
* details：这个微服务中包含了书籍的信息；
* reviews：这个微服务中包含了书籍相关的评论。它还会调用ratings微服务；
* ratings：这个微服务中包含了由书籍评价组成的评级信息。

reviews微服务有3个版本：

* v1版本不会调用ratings服务；
* v2版本会调用ratings服务，并使用1到5个黑色星形图标来显示评分信息；
* v3版本会调用ratings服务，并使用1到5个红色星形图标来显示评分信息。

下图4-3-1展示了这个应用的端到端架构：

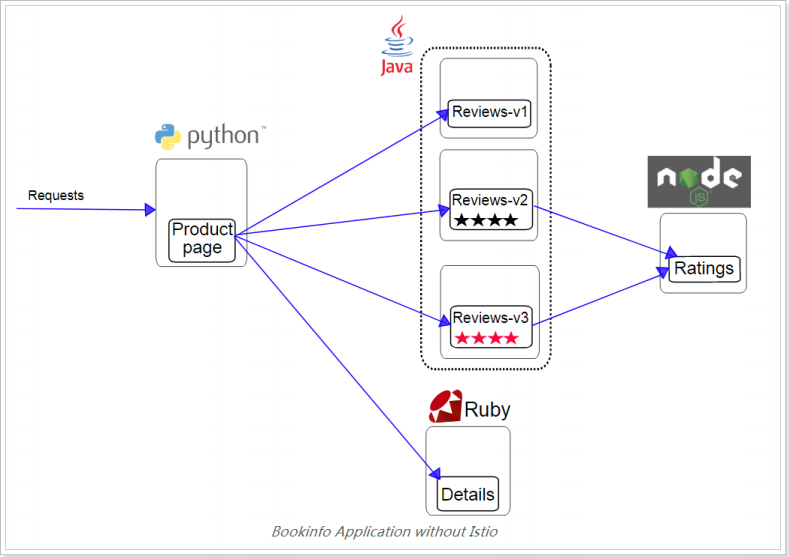


图4-3-1

Bookinfo应用中的几个微服务是由不同的语言编写的。这些服务对Istio并无依赖，但是构成了一个有代表性的服务网格的例子：它由多个服务、多个语言构成，并且reviews服务具有多个版本。

* + 1. 应用部署

要在Istio中运行这一应用，无需对应用自身做出任何改变。您只要简单的在Istio环境中对服务进行配置和运行，具体一点说就是把Envoy sidecar注⼊到每个服务之中。最终的部署结果将如下图所示，如图4-3-2：

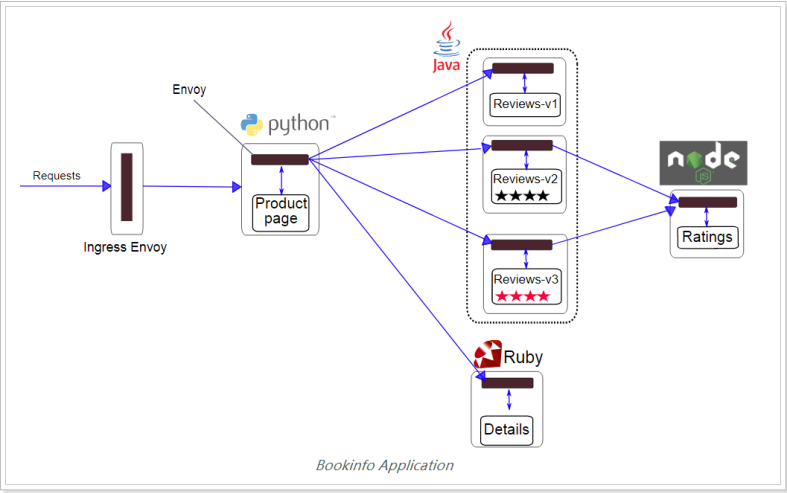


图4-3-2

所有的微服务都和Envoy sidecar集成在一起，被集成服务所有的出⼊流量都被sidecar所劫持，这样就为外部控制准备了所需的Hook，然后就可以利用Istio控制平面为应用提供服务路由、遥测数据收集以及策略实施等功能。

* + 1. 启动应用服务

启动Bookinfo项目操作如下：

* 进⼊Istio安装目录；
* Istio默认自动注⼊sidecar，请为default命名空间打上标签istio-injection=enabled，执行命令“ kubectl label namespace default istio-injection=enabled”；
* 使用kubectl部署应用，执行命令“kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml”

上面的命令会启动全部的四个服务，其中也包括了reviews服务的三个版本（v1、v2以及v3）。在实际部署中，微服务版本的启动过程需要持续一段时间，并不是同时完成的。

* + 1. 确定Ingress的IP

现在Bookinfo服务启动并运行中，您需要使应用程序可以从外部访问Kubernetes集群，例如使用浏览器，可以用Istio Gateway来实现这个目标，操作如下：

* 为应用程序定义Ingress网关，执行命令“kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml”；
* 设置访问网关的INGRESS\_HOST和INGRESS\_PORT变量，执行如图4-3-4a所示操作：

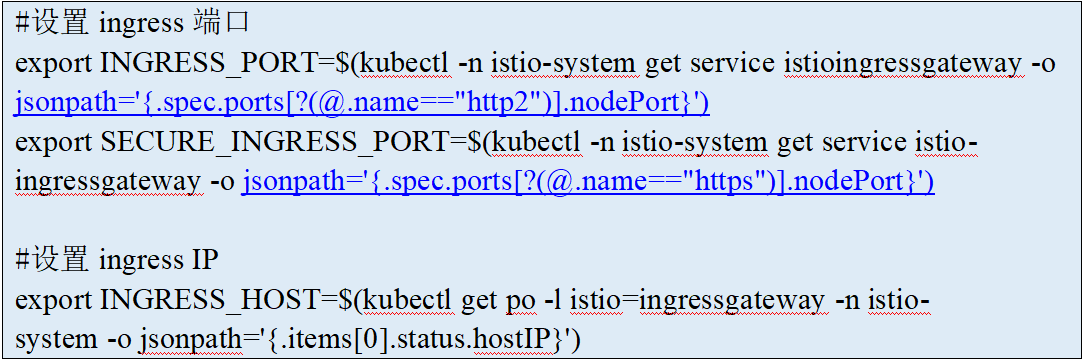


图4-3-4a

* 设置GATEWAY\_URL，执行命令“export GATEWAY\_URL=$INGRESS\_HOST:$INGRESS\_PORT”。

可以用浏览器打开网址http://$GATEWAY\_URL/productpage，来浏览应用的Web页面。如果刷新几次应用的页面，就会看到productpage页面中会随机展示reviews服务的不同版本的效果（红色、黑色的星形或者没有显示）。reviews服务出现这种情况是因为我们还没有使用Istio来控制版本的路由。如图4-3-4b：

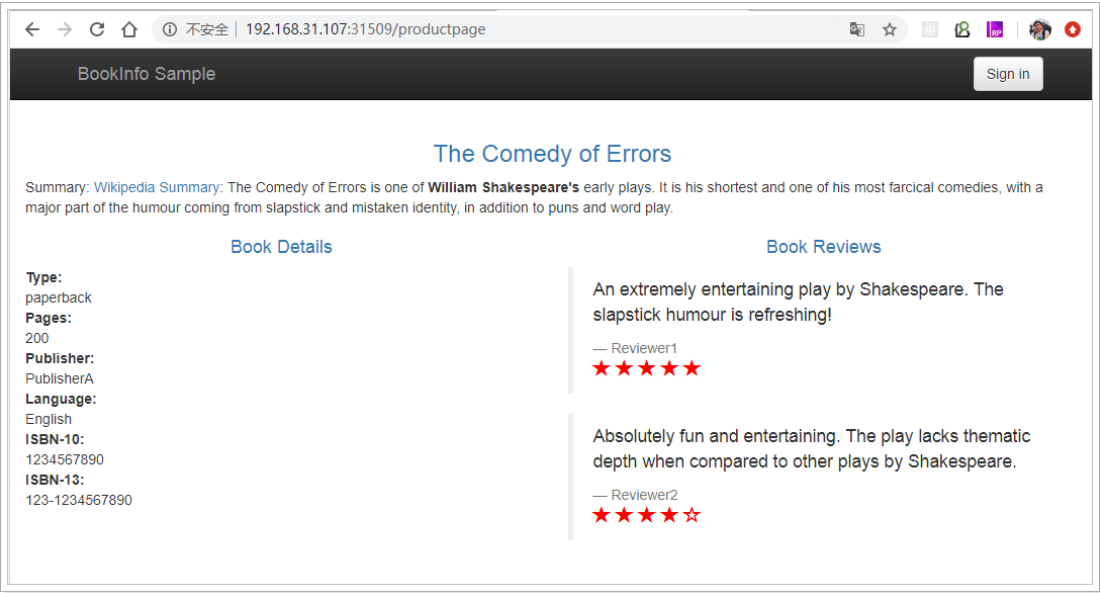


图4-3-4b

* + 1. 应用默认目标规则

在使用Istio控制Bookinfo版本路由之前，需要在目标规则中定义好可用的版本，命名为subsets，操作如图4-3-5。

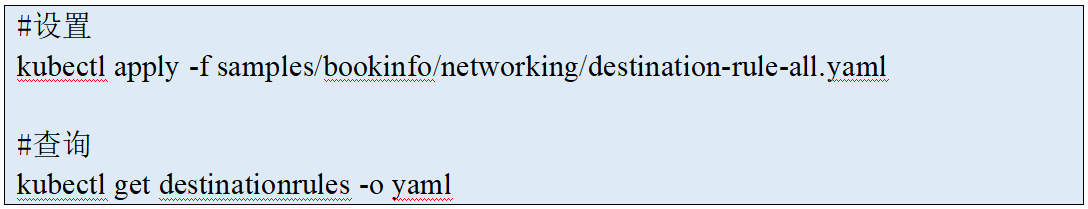


图4-3-5

至此，Istio完成了全部的接管，第一个示例部署完成。

* 1. 体验Istio
     1. 按照版本路由

reviews有三个版本，默认情况下，会进行轮询，也就是我们看到的，每一次刷新都会有不同的效果。如果我们需要将请求全部切换到某一个版本，使用Istio是非常简单的，只需要添加虚拟服务即可。

* 将请求流量全部切换到v1版本，执行命令“kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml”经过测试，发现reviews不再切换样式。如图4-4-1：

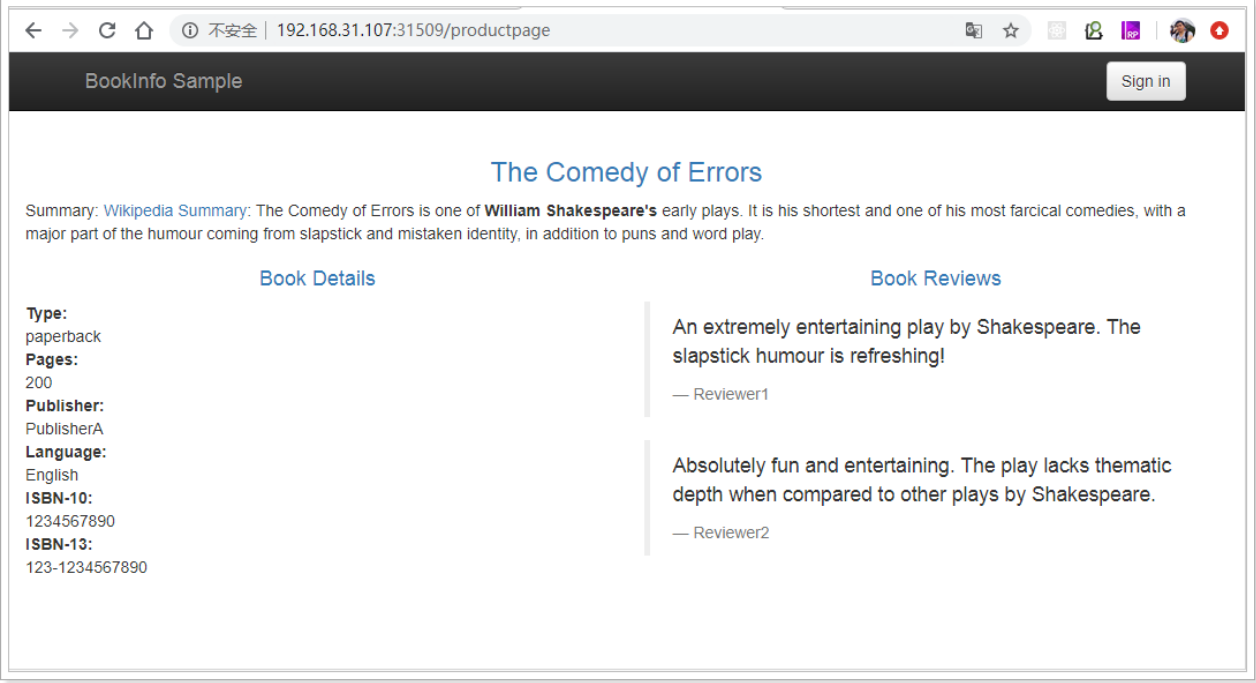


图4-4-1

* 还可以将部分流量转移到v3版本，基于此可以实现灰度发布、A/B测试等,执行命令“kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml”将50%的流量转移到v3版本，刷新浏览器中的/productpage页面，大约有50%的几率会看到页面中出带红色星级的评价内容，这是因为v3版本的reviews访问了带星级评级的ratings服务，但v1版本却没有，如果认为reviews:v3微服务已经稳定，可以通过应用此virtual service规则将100%的流量路由到reviews:v3，执行命令“kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-v3.yaml”这样，所有的请求都转向到了v3了。如果需要删除虚拟网络，可以执行命令“kubectl delete -f samples/bookinfo/networking/virtual-service-all-v1.yaml”
  + 1. 按照不同用户身份路由

接下来，将更改路由配置，以便将来自特定用户的所有流量路由到特定服务版本。在这，来自名为Jason的用户的所有流量将被路由到服务reviews:v2。请注意，Istio对用户身份没有任何特殊的内置机制。事实上，productpage服务在所有到reviews服务的HTTP请求中都增加了一个自定义的end-user请求头，从而达到了本例子的效果。请记住，reviews:v2是包含星级评分功能的版本，操作如下：

* 运行命令“kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml”以启用基于用户的路由；
* 查看规则是否已生效，如图4-4-2：

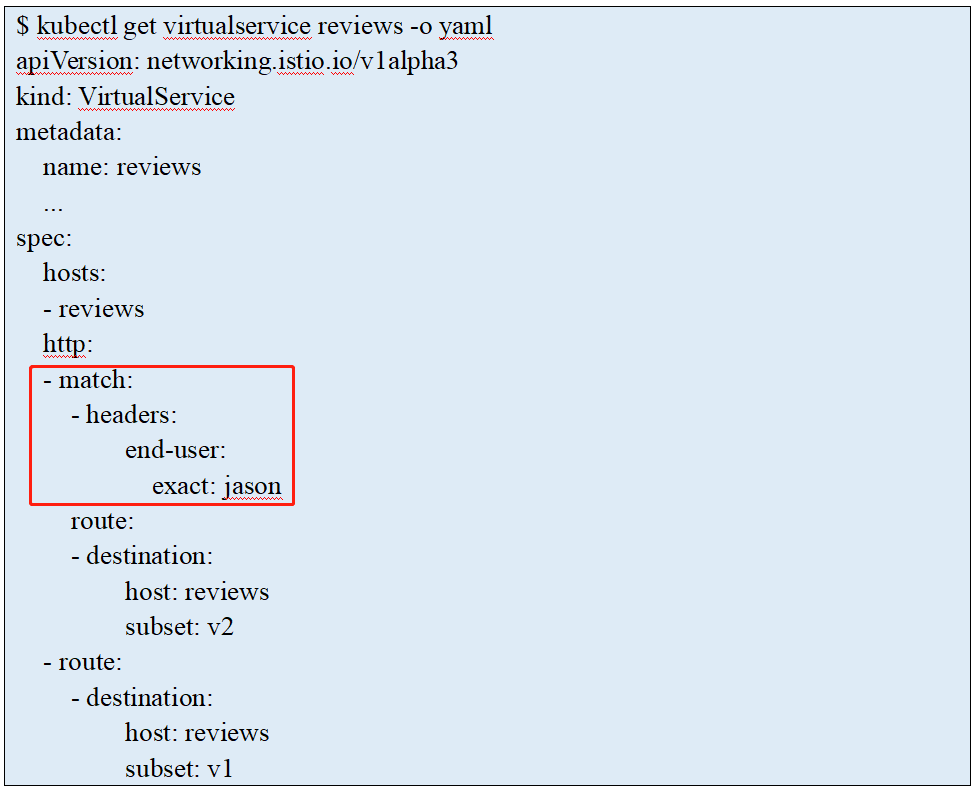


图4-4-2

1. **项目实战**
   1. 项目说明

该项目是仿照豆瓣电影网址实现的电影信息网页，其底层数据使用的Neo4j（图数据库）存储，主体技术使用SpringBoot实现，我们其分析，可以划分为4个微服务：

* 1区：电影信息区域，划分为一个微服务，称之为：movie-info
* 2区：电影的评分区域，划分为一个微服务，称之为：movie-rating
* 3区：电影的推荐，划分为一个微服务，称之为：movie-recommend
* 整体页面划分为一个微服务，称之为：movie-web。

下图5-1a是电影页面的截图：





图5-1a

下图5-1b是Neo4j数据库数据截图：

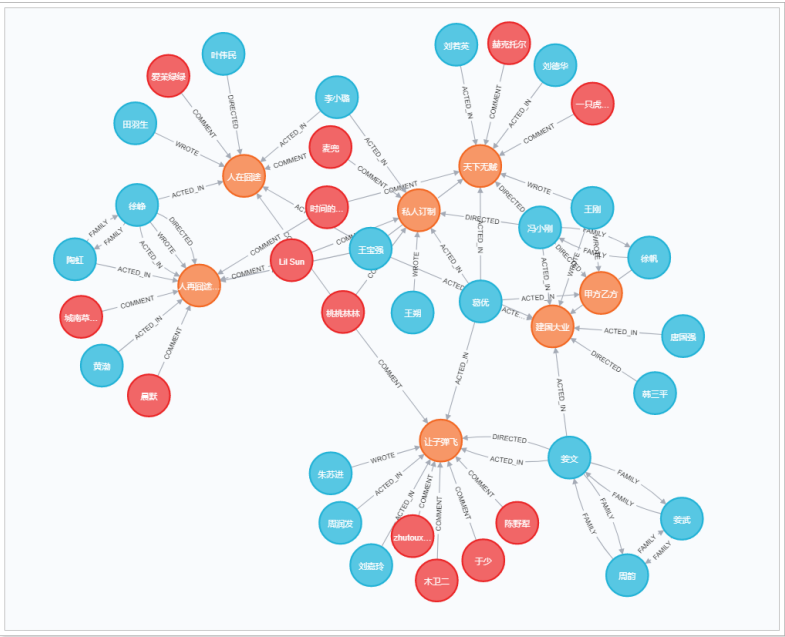


图5-1b

橙色球：电影，蓝色球：参与⼈，红色球：普通用户

* 1. 搭建Neo4J

使用docker进行搭建Neo4j环境，操作如图5-2a：

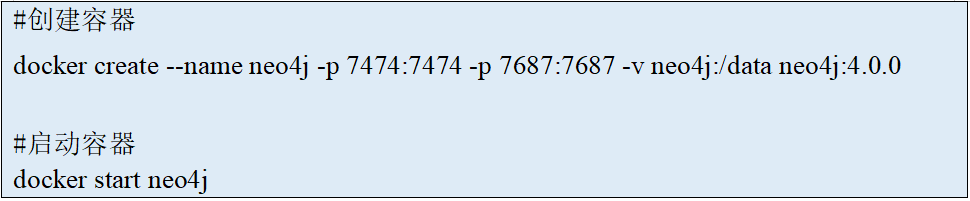


图5-2a

通过http://192.168.31.106:7474/browser/进行登录，如图5-2b:

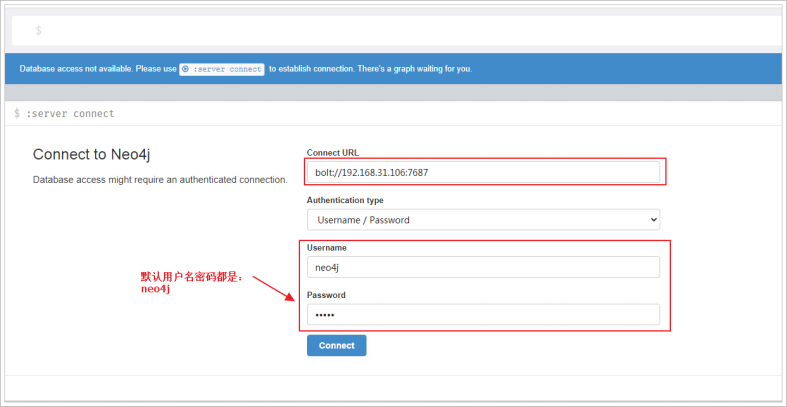


图5-2b

* 1. 创建工程

创建仿豆瓣微服务工程，工程结构如下图5-3：

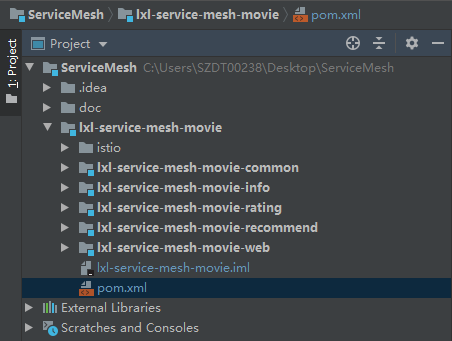
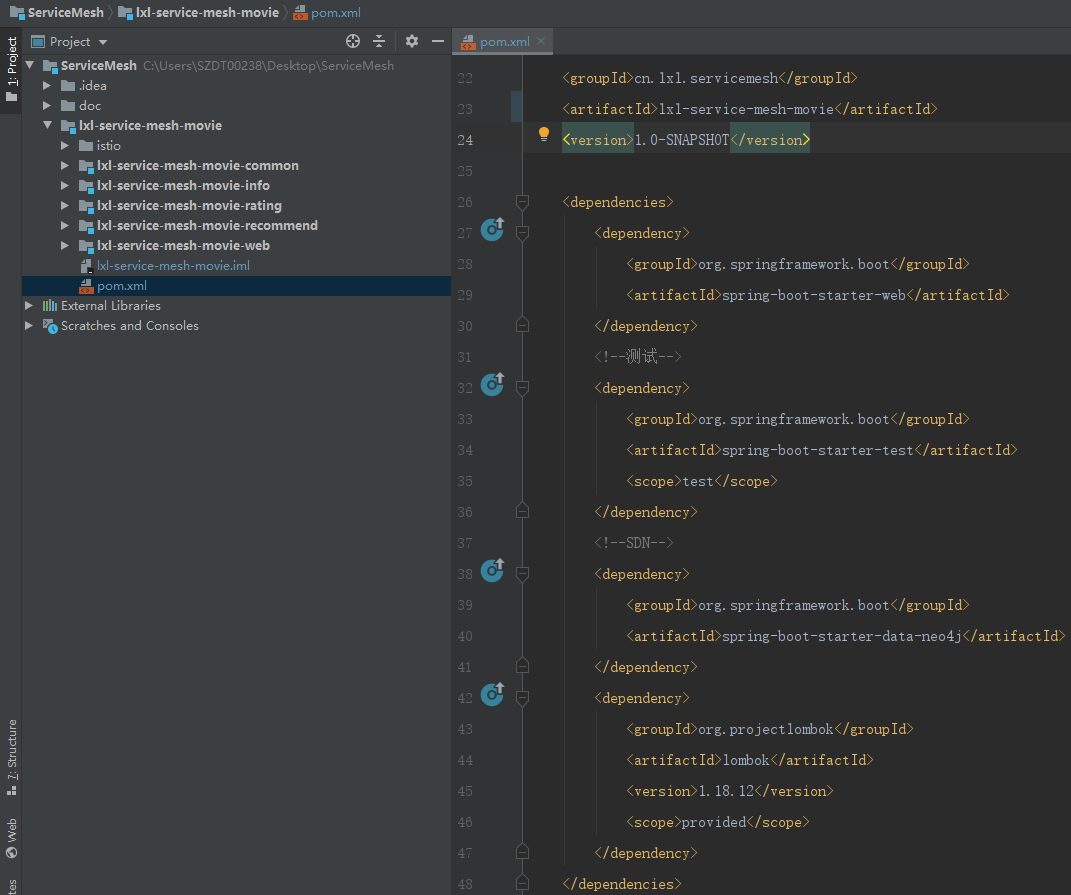


图5-3

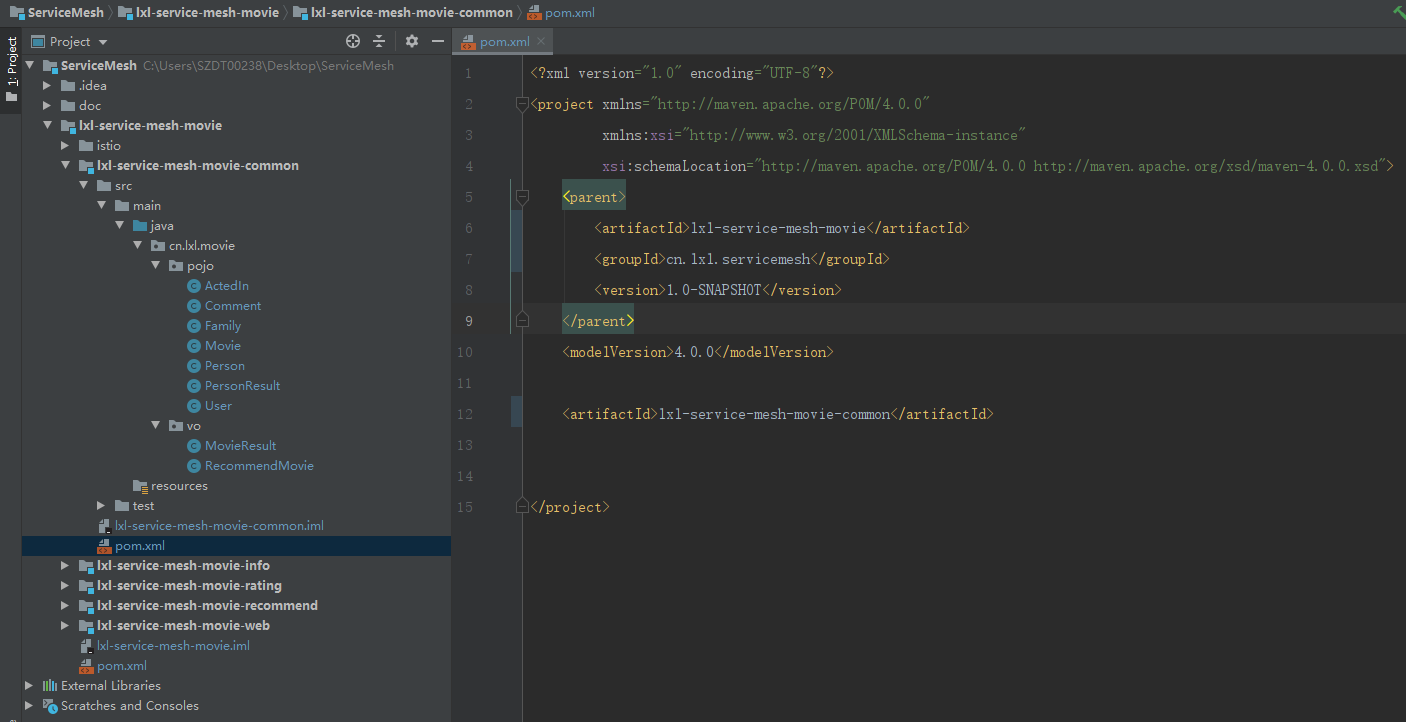
* + 1. lxl-service-mesh-douban

仿豆瓣项目父POM工程,如图5-3-1：



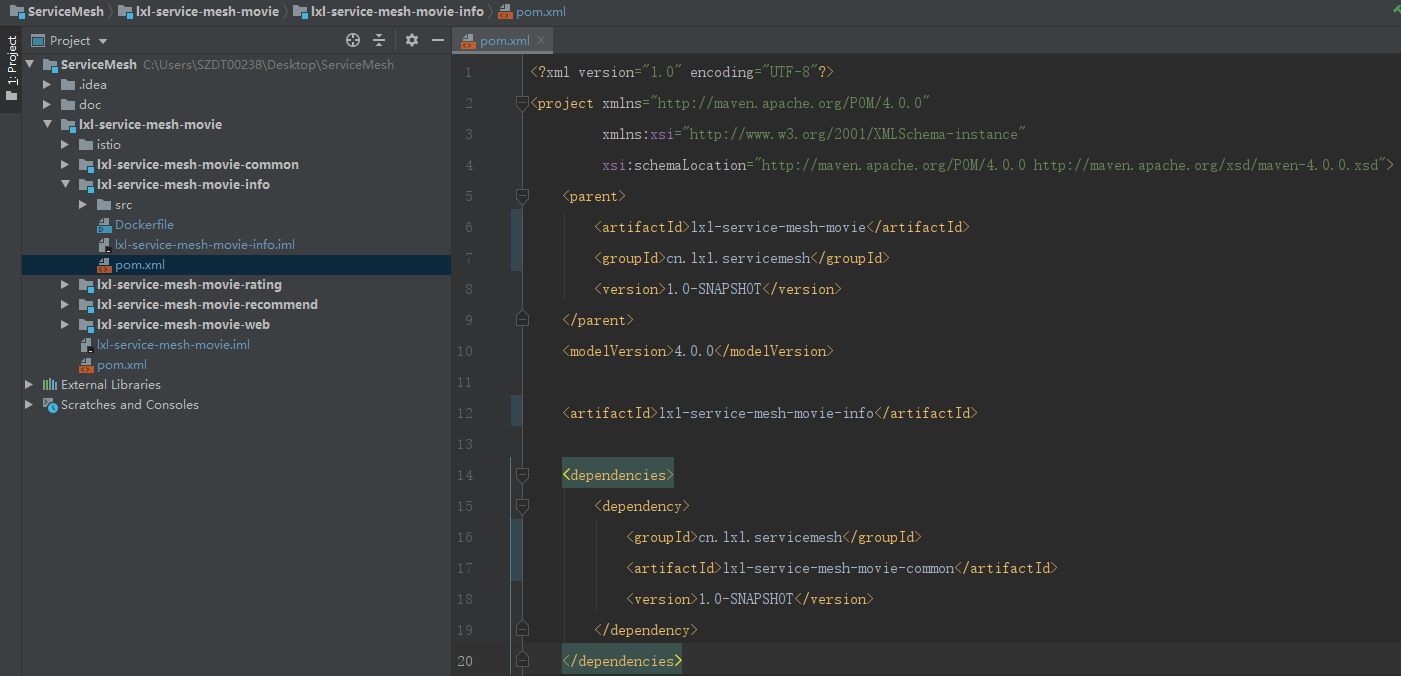
* + 1. lxl-service-mesh-movie-common

公共模块工程结构如图5-3-2：



* + 1. lxl-service-mesh-movie-info

电影详情模块工程结构如图5-3-3：



* + 1. lxl-service-mesh-movie-rating

电影评分模块工程结构如图5-3-4：

* + 1. lxl-service-mesh-movie-recommend

电影推荐模块工程结构如图5-3-5：

* + 1. lxl-service-mesh-movie-web

图5-4

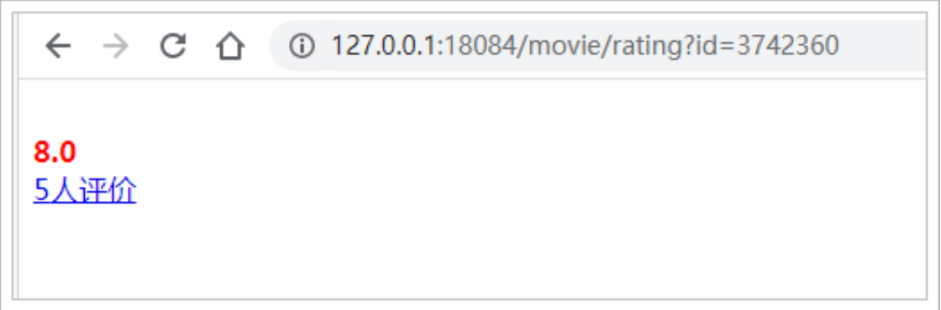


图5-7-7

图5-8-1

* + 1. 配置



图5-8-7

|  |
| --- |
|  |

1. **结论与展望**
   1. 结论

本文通过分析服务架构的演进，将目前市面上主流的传统微服务架构和基于服务网格的微服务架构进行对比，指出了传统微服务架构目前存在的问题，以及服务网格这种新的微服务架构是如何来解决传统微服务架构存在的问题。通过对比目前市面上主流的服务网格框架，最终选择了开源的Istio服务网格框架进行研究学习，最后基于Istio框架构建了仿豆瓣电影详情功能，发现Istio可以完美的实现业务与服务通信之间解耦，使开发人员只需要专注于业务的开发，极大的提高了开发效率。

* 1. 展望

目前服务网格技术还没有普及，只是在一些大型公司非核心项目上得到了落地，相信随着服务网格技术的不断发展，未来大量的微服务系统架构将会采用服务网格技术。