|  |
| --- |
| **毕 业 论 文** |
|  |
|  |
| **微服务架构Service Mesh的设计与应用** |

|  |  |  |
| --- | --- | --- |
| **学生姓名** | 刘晓璐 |  |
| **学 号** | 030418111415 |  |
| **专业班级** | XXX 班 |  |
| **指导教师** | XXX |  |
| **完成日期** | 201X年X月 |  |

**微服务架构Service Mesh的设计与应用**

摘 要

微服务架构正处于兴起阶段，在SpringCloud,Dubbo框架的支持下，微服务框架成为了最主流的分布式应用解决方案，然而随着微服务实施水平的不断深化，不论是市面上的哪种框架，都会存在一些问题，本文将分析当前微服务架构存在的一些问题，并且针对这些问题找到基于Service Mesh的Istio框架这种比较好的解决方案。最后在实际的运用场景区域卫生综合管理平台进行探讨，说明了Service Mesh的这种架构思想优于传统的微服务的架构思想。

关键词：微服务；Service Mesh;Istio;区域卫生综合管理平台

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

**ABSTRACT**

Micro-services Architecture is on the rise. with the support of SpringCloud and Dubbo, it has become the main solution for the distributed application. However, with the deepening of Micro-services implementation level, it will be some problems no matter which framework is on the market. This article analysises the key problem in the current Micro-services Architecture. At the same time, this article finds the relatively good solution for the Istio architecture which is based on Service Mesh for these problems. At last, this article discusses in the practical application scenario on region health management platform. It demonstrates this framework is better than the traditional micro-services framework)

**Key Words**: Micro-services; Service Mesh; Istio; region health management platform

目 录

摘 要 I

**ABSTRACT** II

1 引言 1

2 研究现状 2

2.1微服务架构 2

2.2 当前微服务架构存在的问题 2

2.2.1 技术门槛高 2

2.2.1 多语言支持不足 2

2.2.1 代码侵入性强 2

2.3 解决方案 2

3 Service Mesh 4

3.1 Service Mesh定义 4

3.2 边车模式 4

3.3 服务网格由来 4

4 Istio微服务框架 4

4.1 Istio介绍 4

4.2 Istio架构 4

4.2.1 Envoy 2

4.2.2 Mixer 2

4.2.3 Pilot 2

4.2.4 Istio-Auth 2

4.3 Istio功能以及特性 4

4.3.1 流量管理 2

4.3.2 策略与执行 2

4.3.3 服务身份以及安全 2

5 Service Mesh的应用 4

5.1 区域卫生综合管理平台实例分析 4

6 结束语 4

参 考 文 献 5

附 录 7

致 谢 8

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：

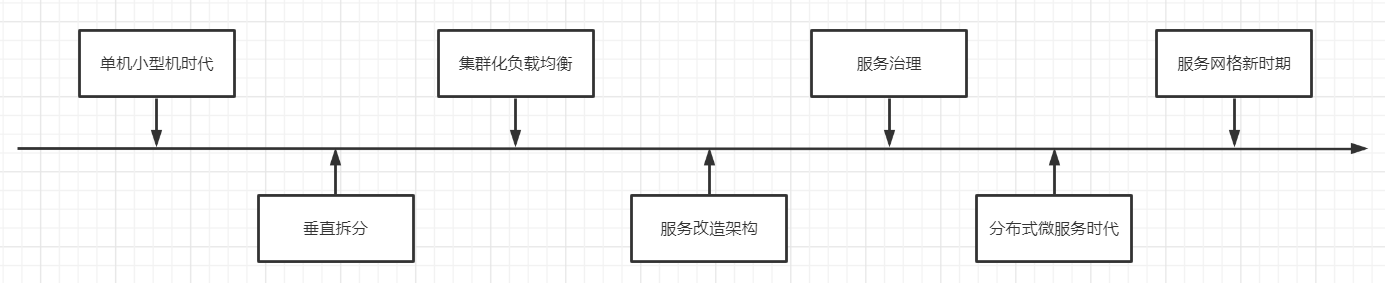


图1-1-1

* + 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，中国的网⺠⼤约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：

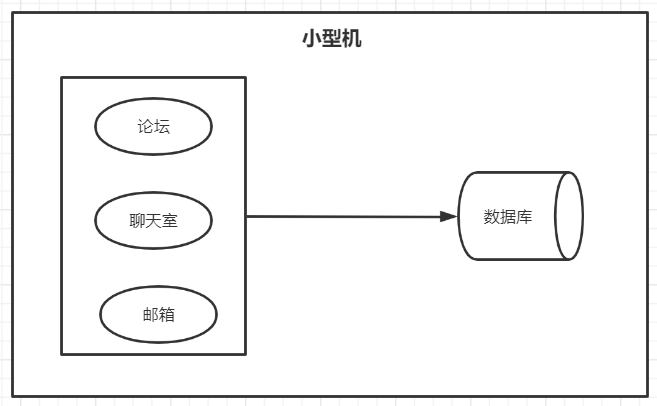


图 1-1-2

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：

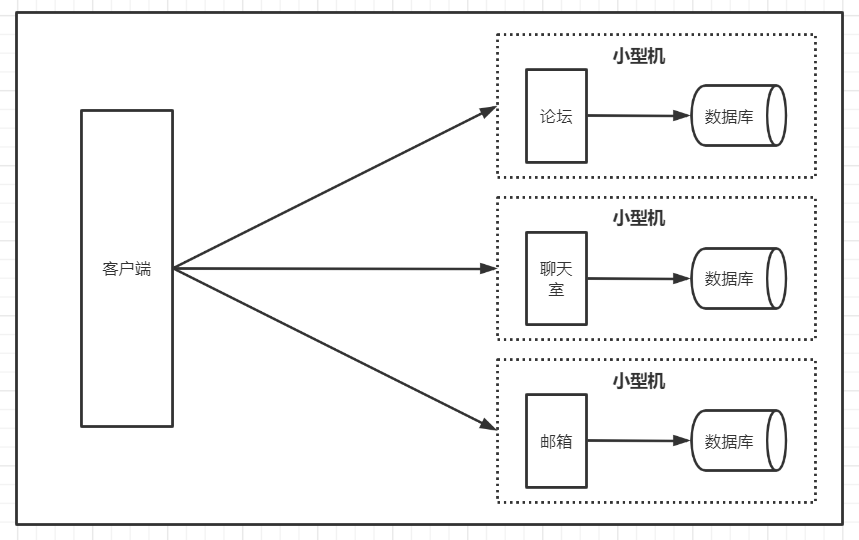


图 1-1-3

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⻋薪。

* + 1. 集群化负载均衡架构

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，我们称为"水平扩展"。如图1-1-4：

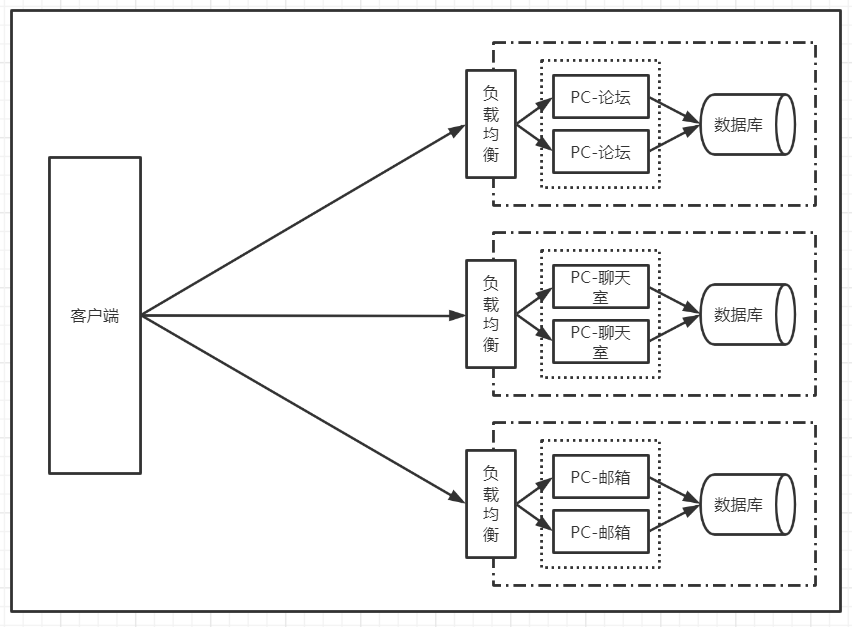


图 1-1-4

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：

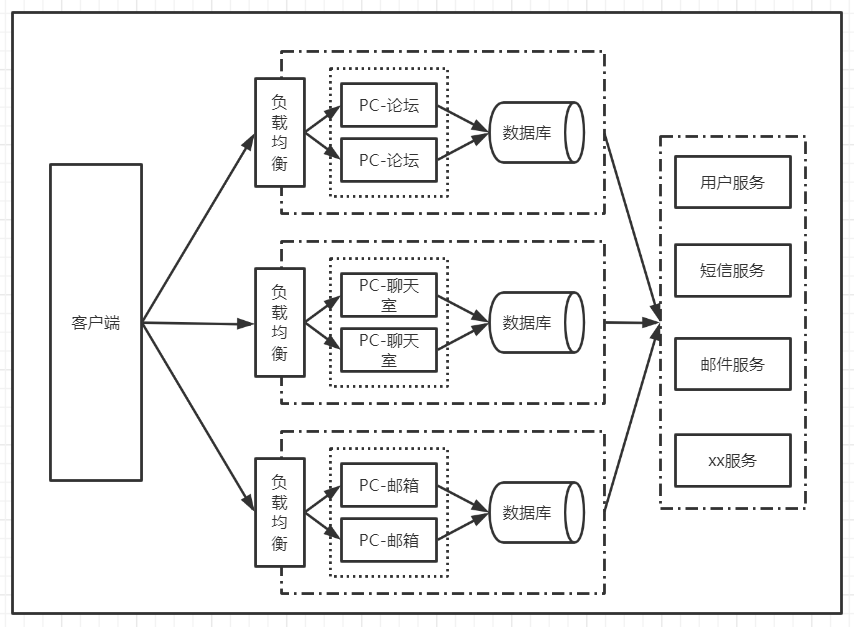


图 1-1-5

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：

* 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。
* 当服务链路调用很⻓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：

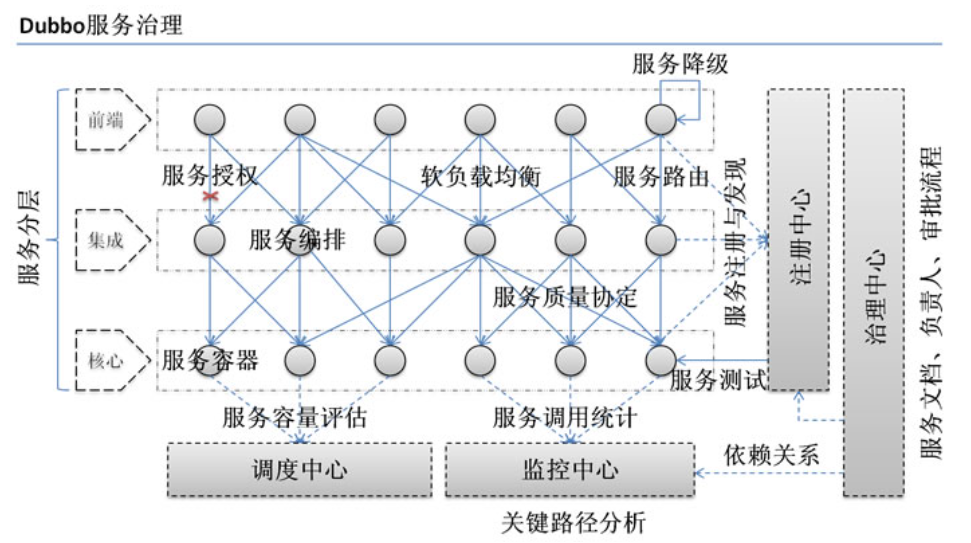


图 1-1-6

* + 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。 ⽐如传统的“用户中⼼”服务，对于微服务来说，需要根据业务再次拆分，可能需要拆分成“买家服务”、 “卖家服务”、“商家服务”等。典型代表：Spring Cloud，相对于传统分布式架构，SpringCloud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：

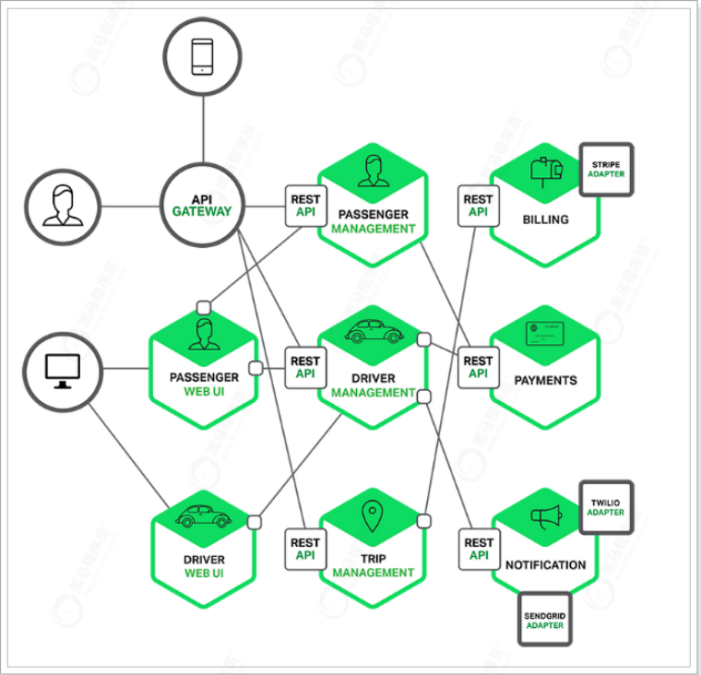


图 1-1-7

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：

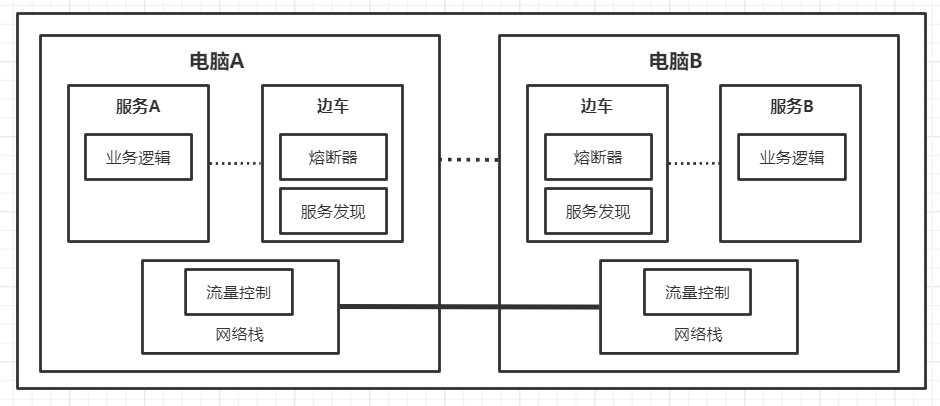


图1-1-8a

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）。

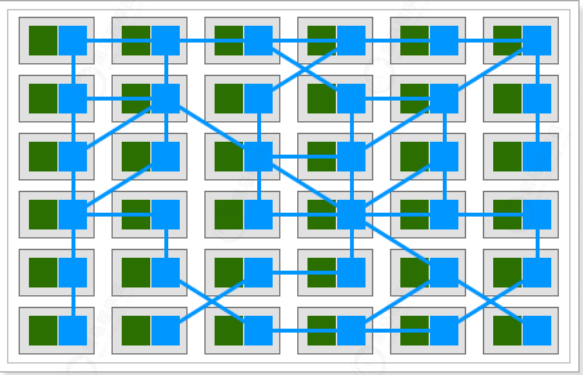


图1-1-8c

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组件。如图1-1-8d：

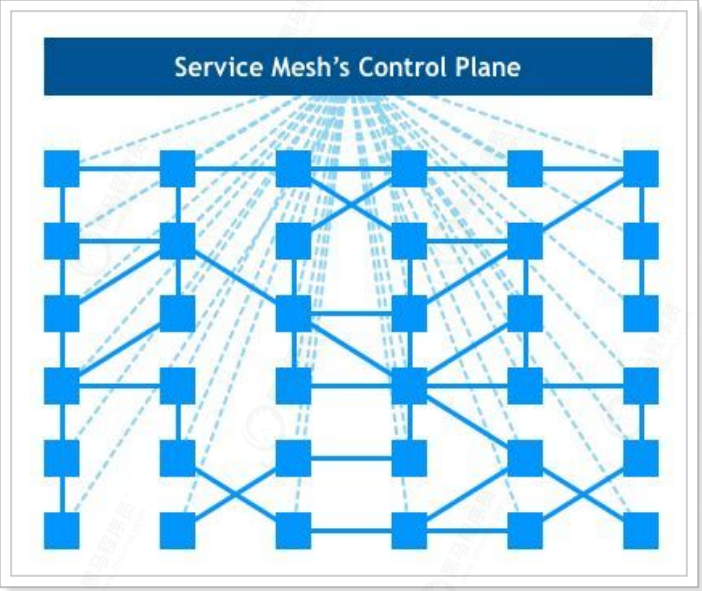


图1-1-8d

* 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：

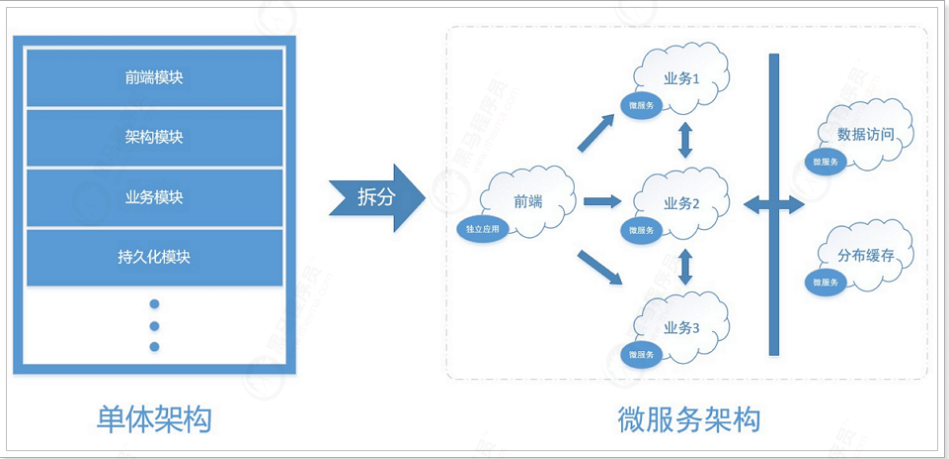


图1-2

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：

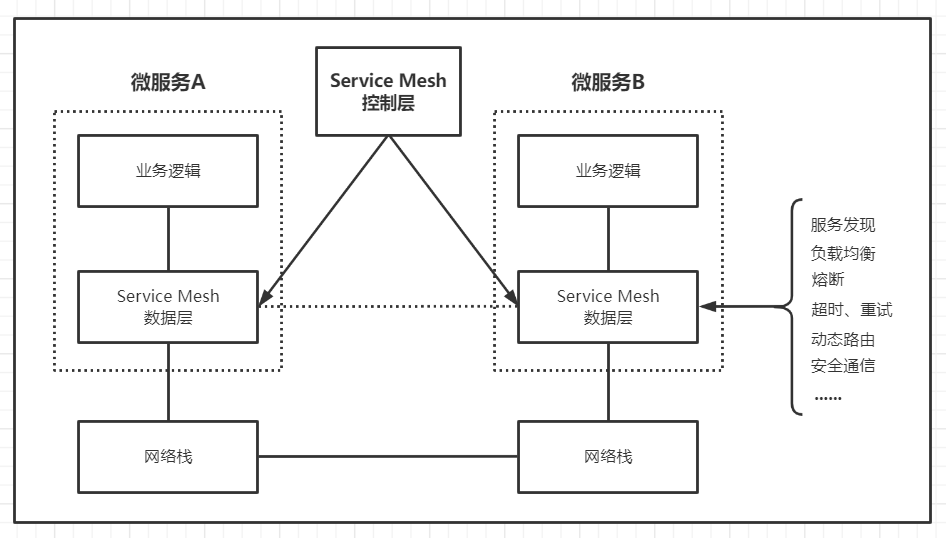


图2-1

* 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

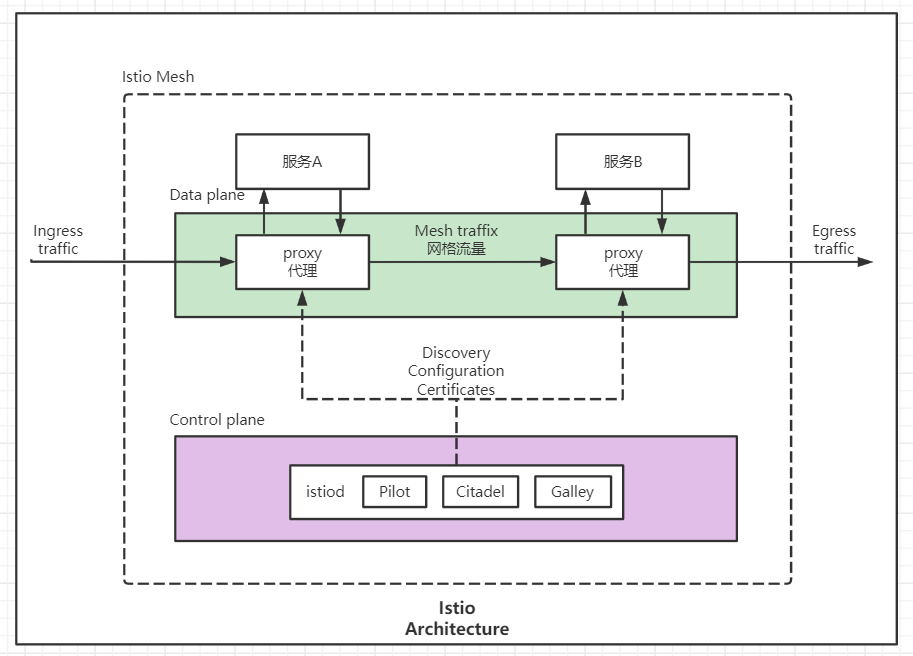


图3-1

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。

* 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。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kubernetes集群

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

* + 1. 环境准备

准备3台Centos7虚拟机：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | IP | ⻆色 | CPU | 内存 | 硬盘 |
| node1 | 192.168.31.106 | master | 2核 | 4GB | 100GB |
| node2 | 192.168.31.107 | node | 2核 | 4GB | 100GB |
| node3 | 192.168.31.108 | node | 2核 | 4GB | 100GB |

* + 1. 前置工作

搭建K8S之前，需要⼀些前置的准备工作，否则不能完成集群的搭建。

|  |
| --- |
| #修改主机名  hostnamectl set-hostname node2  hostnamectl set-hostname node3 |
| #更新yum源，并且完成yum update操作  mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup  wget -O /etc/yum.repos.d/CentOS-Base.repo https://mirrors.aliyun.com/repo/Centos-7.repo  yum makecache  yum -y update |
| #安装docker  yum install -y yum-utils device-mapper-persistent-data lvm2  yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo  yum makecache fast  yum -y install docker-ce  #启动docker服务  systemctl start docker.service  #开机⾃启 systemctl enable docker.service  #添加docker阿⾥云加速器 sudo mkdir -p /etc/docker sudo tee /etc/docker/daemon.json <<-'EOF' { "registry-mirrors": ["https://c6n8vys4.mirror.aliyuncs.com"] } EOF sudo systemctl daemon-reload sudo systemctl restart docker  #测试⼀下，看下载速度怎么样  docker pull redis  docker rmi redis:latest |
| #关闭防⽕墙  systemctl stop firewalld.service  systemctl disable firewalld.service  #添加hosts映射  vim /etc/hosts  192.168.31.106 node1  192.168.31.107 node2  192.168.31.108 node3  scp /etc/hosts node2:/etc/  scp /etc/hosts node3:/etc/ |
| #设置node1到node2、node3免登陆  ssh-keygen #⼀路下⼀步操作  ssh-copy-id node2  ssh-copy-id node3  #测试  ssh node2  ssh node3 |

* + 1. 搭建集群

|  |
| --- |
| #修改系统参数  # 将 SELinux 设置为 permissive 模式（相当于将其禁用）  setenforce 0  sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config  # 禁用swap⽂件，编辑/etc/fstab，注释掉引用 swap 的⾏  vim /etc/fstab  swapoff -a  # 设置网桥参数  vim /etc/sysctl.conf  net.bridge.bridge-nf-call-ip6tables = 1  net.bridge.bridge-nf-call-iptables = 1  net.ipv4.ip\_forward = 1  net.ipv4.tcp\_tw\_recycle = 0  scp /etc/sysctl.conf node2:/etc/  scp /etc/sysctl.conf node3:/etc/  #⽴即生效  sysctl -p  #安装kubectl  vim /etc/yum.repos.d/kubernetes.repo  [kubernetes]  name=Kubernetes baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/  enabled=1  gpgcheck=0  yum list kubectl –showduplicates  yum install -y kubectl.x86\_64  kubectl version  yum install -y kubelet kubeadm --disableexcludes=kubernetes  #拉取所需要的镜像  kubeadm config images pull --image-repository=registry.cnhangzhou.aliyuncs.com/lxl --kubernetes-version=v1.18.6  #如果拉取失败，尝试这个：kubeadm config images pull --image-repository=lank8s.cn -- kubernetes-version=v1.18.6  #开始初始化，如果使用lank8s.cn拉取的镜像，需要指定--image-repository=lank8s.cn  kubeadm init --apiserver-advertise-address 192.168.31.106 --pod-networkcidr=10.244.0.0/16 --image-repository=registry.cn-hangzhou.aliyuncs.com/lxl --kubernetes-version=v1.18.6  #当看到 Your Kubernetes control-plane has initialized successfully! 说明初始化成功了！  #拷⻉admin.conf到home目录，否则出错：The connection to the server localhost:8080 was refused - did you specify the right host or port?  mkdir -p $HOME/.kube  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  sudo chown $(id -u):$(id -g) $HOME/.kube/config  #设置网络 kube-flannel.yml ⽂件在资料中  kubectl apply -f kube-flannel.yml  #测试  [root@node1 k8s]# kubectl get nodes  NAME STATUS ROLES AGE VERSION  node1 Ready master 23m v1.18.6  #将node2、node3加⼊到集群，token要替换成自己的  kubeadm join 192.168.31.106:6443 --token ekw4eu.cfi77sji1jyczhj6 --discoverytoken-ca-cert-hash sha256:21de4177eaf76353dd060f2a783c9dafe17636437ade020bc40d60a8ab903483  #测试  [root@node1 k8s]# kubectl get nodes  NAME STATUS ROLES AGE VERSION  node1 Ready master 31m v1.18.6  node2 Ready <none> 6m46s v1.18.6  node3 Ready <none> 2m21s v1.18.6  #说明集群组件成功了  #如果需要删除集群，执⾏ kubeadm reset ，3台机器都需要执⾏  #查看正在执⾏的pod  kubectl get pod --all-namespaces -o wide |

查看正在执⾏的pod，kubectl get pod --all-namespaces -o wide,如图4-1-3：

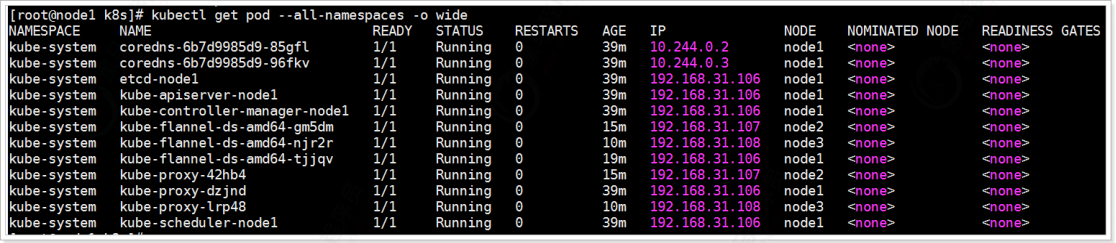


图4-1-3

* 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。

1. 访问 Istio release 页面下载与您操作系统对应的安装⽂件。在 macOS 或 Linux 系统中，也可以通 过以下命令下载最新版本的 Istio：

|  |
| --- |
| $ curl -L https://istio.io/downloadIstio | sh - |

1. 切换到Istio包所在目录下。例如：Istio 包名为 istio-1.6.5 ，则：

|  |
| --- |
| $ cd istio-1.6.5 |

安装目录包含如下内容：

* 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。

1. 将 istioctl 客户端路径增加到 path 环境变量中，macOS 或 Linux 系统的增加方式如下：

|  |
| --- |
| $ export PATH=$PWD/bin:$PATH |

安装 bash 自动补全⽂件，如果您使用 bash，istioctl 自动补全的⽂件位于 tools 目录，通过复制 istioctl.bash⽂件到您的home目录，然后添加下⾏内容到您的 .bashrc ⽂件执⾏ istioctl tab 补全⽂件：

|  |
| --- |
| source ~/istioctl.bash |

如果istioctl补全⽂件已经正确安装，在您输⼊istioctl命令时通过按Tab键，它会返回⼀组推荐命令供您选择：

|  |
| --- |
| $ istioctl proxy-<TAB>  proxy-config proxy-status |

* + 1. 安装Istio

请按照以下步骤在您所选的平台上使用demo配置⽂件安装 Istio。

1. 安装demo配置

|  |
| --- |
| $ istioctl manifest apply --set profile=demo |

1. 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 ：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| $ kubectl get svc -n istio-system | | | | | |
| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
| grafana | ClusterIP | 172.21.211.123 | <none> | 3000/TCP | 2m |
| istio-citadel | ClusterIP | 172.21.177.222 | <none> | 8060/TCP,15014/TCP | 2m |
| istio-egressgateway | ClusterIP | 172.21.113.24 | <none> | 80/TCP,443/TCP,15443/TCP | 2m |
| istio-galley | ClusterIP | 172.21.132.247 | <none> | 443/TCP,15014/TCP,9901/TCP | 2m |
| istio-ingressgateway | LoadBalancer | 172.21.144.254 | 52.116.22.242 | 15020:31831/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30318/TCP ,15030:32645/TCP,15031:31933/TCP,15032:31188/TCP,15443:30838/TCP | 2m |
| istio-pilot | ClusterIP | 172.21.105.205 | <none> | 15010/TCP,15011/TCP,8080/TCP,15014/TCP | 2m |
| istio-policy | ClusterIP | 172.21.14.236 | <none> | 9091/TCP,15004/TCP,15014/TCP | 2m |
| istio-sidecar-injector | ClusterIP | 172.21.155.47 | <none> | 443/TCP,15014/TCP | 2m |
| istio-telemetry | ClusterIP | 172.21.196.79 | <none> | 9091/TCP,15004/TCP,15014/TCP,42422/TCP | 2m |
| jaeger-agent | ClusterIP | None | <none> | 5775/UDP,6831/UDP,6832/UDP | 2m |
| jaeger-collector | ClusterIP | 172.21.135.51 | <none> | 14267/TCP,14268/TCP | 2m |
| jaeger-query | ClusterIP | 172.21.26.187 | <none> | 16686/TCP | 2m |
| kiali | ClusterIP | 172.21.155.201 | <none> | 20001/TCP | 2m |
| prometheus | ClusterIP | 172.21.63.159 | <none> | 9090/TCP | 2m |
| tracing | ClusterIP | 172.21.2.245 | <none> | 80/TCP | 2m |
| zipkin | ClusterIP | 172.21.182.245 | <none> | 9411/TCP | 2m |

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 ：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $ kubectl get pods -n istio-system | | | | |
| NAME | READY | STATUS | RESTARTS | AGE |
| grafana-f8467cc6-rbjlg | 1/1 | Running | 0 | 1m |
| istio-citadel-78df5b548f-g5cpw | 1/1 | Running | 0 | 1m |
| istio-egressgateway-78569df5c4-zwtb5 | 1/1 | Running | 0 | 1m |
| istio-galley-74d5f764fc-q7nrk | 1/1 | Running | 0 | 1m |
| istio-ingressgateway-7ddcfd665c-dmtqz | 1/1 | Running | 0 | 1m |
| istio-pilot-f479bbf5c-qwr28 | 1/1 | Running | 0 | 1m |
| istio-policy-6fccc5c868-xhblv | 1/1 | Running | 2 | 1m |
| istio-sidecar-injector-78499d85b8-x44m6 | 1/1 | Running | 0 | 1m |
| istio-telemetry-78b96c6cb6-ldm9q | 1/1 | Running | 2 | 1m |
| istio-tracing-69b5f778b7-s2zvw | 1/1 | Running | 0 | 1m |
| kiali-99f7467dc-6rvwp | 1/1 | Running | 0 | 1m |
| prometheus-67cdb66cbb-9w2hm | 1/1 | Running | 0 | 1m |

* 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。

下图展示了这个应用的端到端架构。如图4-3-1：

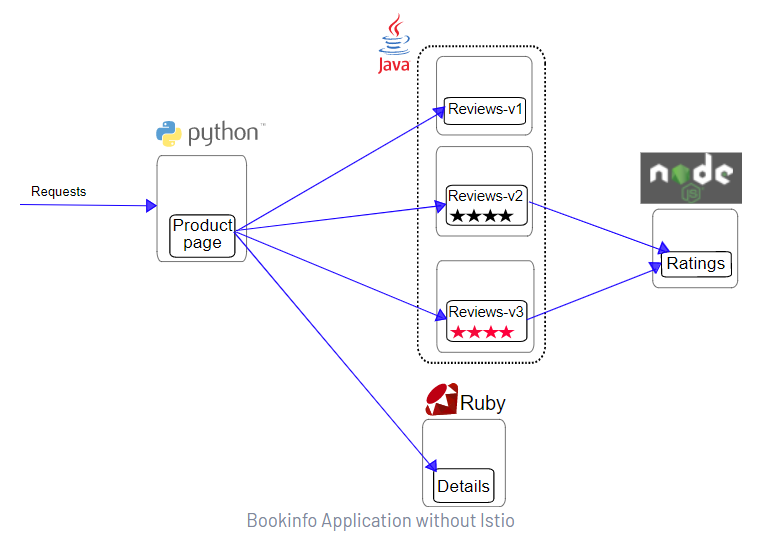


图4-3-1

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：

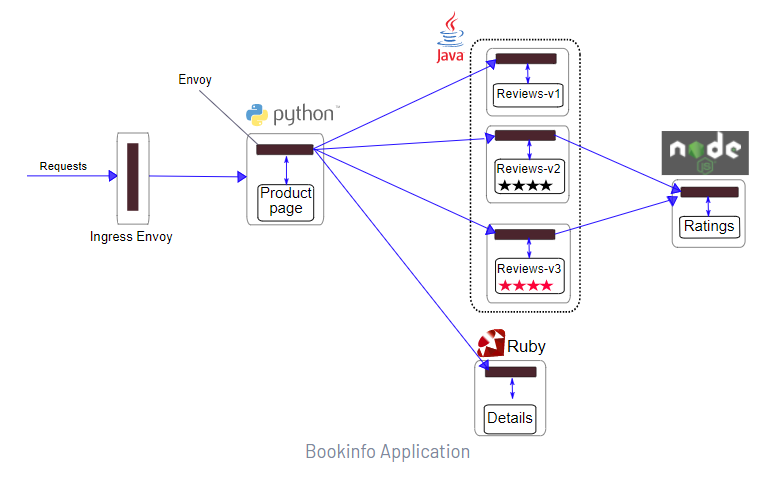


图4-3-2

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

1. 进⼊Istio安装目录。
2. Istio默认自动注⼊Sidecar. 请为 default命名空间打上标签 istio-injection=enabled ：

|  |
| --- |
| $ kubectl label namespace default istio-injection=enabled |

1. 使用 kubectl 部署应用：

|  |
| --- |
| $ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml |

如果您在安装过程中禁用了Sidecar自动注⼊功能而选择手动注⼊Sidecar，请在部署应用之前使用 istioctl kube-inject命令修改 bookinfo.yaml⽂件。

|  |
| --- |
| $ kubectl apply -f <(istioctl kube-inject -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. 确认所有的服务和 Pod 都已经正确的定义和启动：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $ kubectl get services | | | | |
| NAME | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
| details | 10.0.0.31 | <none> | 9080/TCP | 6m |
| kubernetes | 10.0.0.1 | <none> | 443/TCP | 7d |
| productpage | 10.0.0.120 | <none> | 9080/TCP | 6m |
| ratings | 10.0.0.15 | <none> | 9080/TCP | 6m |
| reviews | 10.0.0.170 | <none> | 9080/TCP | 6m |
| $ kubectl get pods | | | | |
| NAME | READY | STATUS | RESTARTS | AGE |
| details-v1-1520924117-48z17 | 2/2 | Running | 0 | 6m |
| productpage-v1-560495357-jk1lz | 2/2 | Running | 0 | 7d |
| ratings-v1-734492171-rnr5l | 2/2 | Running | 0 | 6m |
| reviews-v1-874083890-f0qf0 | 2/2 | Running | 0 | 6m |
| reviews-v3-1813607990-8ch52 | 2/2 | Running | 0 | 6m |

1. 要确认Bookinfo 应用是否正在运行，请在某个Pod中用curl 命令对应用发送请求，例如 ratings ：

|  |
| --- |
| $ kubectl exec -it $(kubectl get pod -l app=ratings -o  jsonpath='{.items[0].metadata.name}') -c ratings -- curl  productpage:9080/productpage | grep -o ""  <title>Simple Bookstore App</title> |

* + 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。

1. 为应用程序定义Ingress网关：

|  |
| --- |
| $ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml |

1. 确认网关创建完成：

|  |
| --- |
| $ kubectl get gateway  NAME AGE  bookinfo-gateway 32s |

1. 设置访问网关的INGRESS\_HOST和INGRESS\_PORT变量。确认并设置。

|  |
| --- |
| #设置 ingress 端⼝  export INGRESS\_PORT=$(kubectl -n istio-system get service istioingressgateway -o [jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')](mailto:jsonpath='{.spec.ports[?(@.name==\"http2\")].nodePort}'))  export SECURE\_INGRESS\_PORT=$(kubectl -n istio-system get service istio-  ingressgateway -o [jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')](mailto:jsonpath='{.spec.ports[?(@.name==\"https\")].nodePort}'))  #设置ingress IP  export INGRESS\_HOST=$(kubectl get po -l istio=ingressgateway -n istio-  system -o jsonpath='{.items[0].status.hostIP}') |

1. 设置GATEWAY\_URL ：

|  |
| --- |
| 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4-3-4：

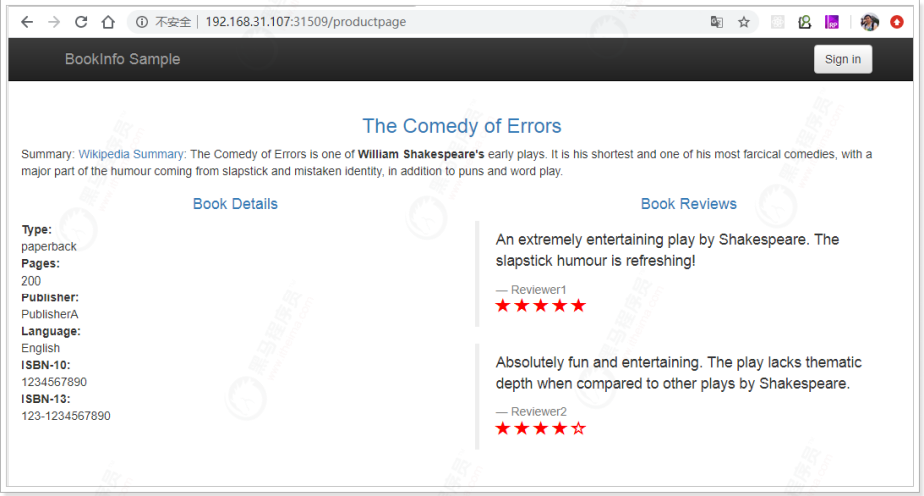


图4-3-4

* + 1. 应用默认目标规则

在使用Istio控制 Bookinfo版本路由之前，您需要在目标规则中定义好可用的版本，命名为 subsets 。

|  |
| --- |
| #设置 kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml  #查询 kubectl get destinationrules -o yaml |

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，需要Istio是非常简单的，只需要添加虚拟服务即可。

示例：

|  |
| --- |
| #virtual-service-all-v1.yaml是官方提供的示例⽂件  kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml |

其内容如下：

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: productpage  spec:  hosts:  - productpage  http:  - route:  - destination:  host: productpage  subset: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: reviews  spec:  hosts:  - reviews  http:  - route:  - destination:  host: reviews  subset: v1 #在这⾥指定了所有的http请求都通过v1完成，而v1在默认的规则中有定义  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: ratings  spec:  hosts:  - ratings  http:  - route:  - destination:  host: ratings  subset: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: details  spec:  hosts:  - details  http:  - route:  - destination:  host: details  subset: v1  --- |

经过测试，发现reviews不再切换样式。如图4-4-1：

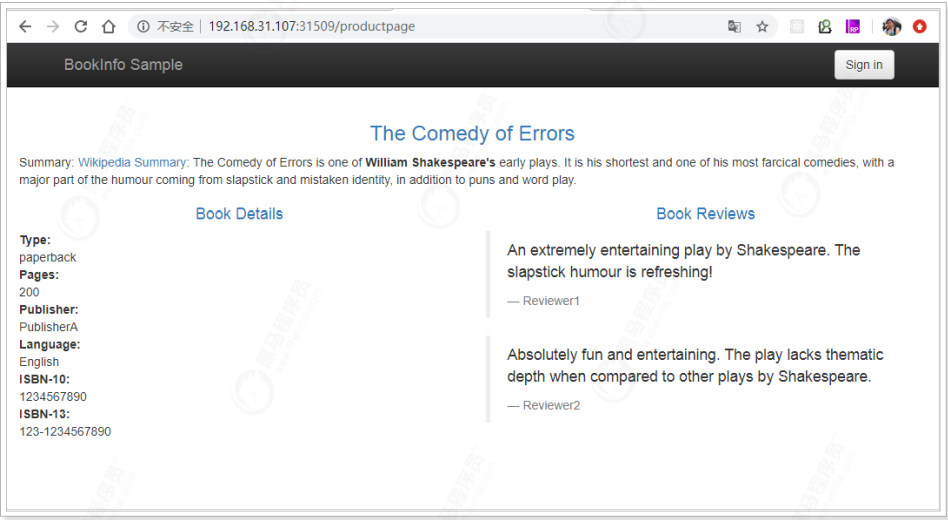


图4-4-1

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：

|  |
| --- |
| #将50%的流程转移到v3  kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml |

内容如下：

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: reviews  spec:  hosts:  - reviews  http:  - route:  - destination:  host: reviews  subset: v1  weight: 50  - destination:  host: reviews  subset: v3  weight: 50 |

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：

|  |
| --- |
| 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：

|  |
| --- |
| 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，您将更改路由配置，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。

1. 运行以下命令以启用基于用户的路由：

|  |
| --- |
| $ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-  test-v2.yaml |

1. 确认规则已创建：

|  |
| --- |
| $ kubectl get virtualservice reviews -o yaml  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: reviews  ...  spec:  hosts:  - reviews  http:  - match:  - headers:  end-user:  exact: jason  route:  - destination:  host: reviews  subset: v2  - route:  - destination:  host: reviews  subset: v1 |

1. 在 Bookinfo 应用程序的/productpage 上，以用户jason身份登录，刷新浏览器，你看到了什么？星级评分显示在每个评论旁边。
2. 以其他用户身份登录（选择您想要的任何名称）。 刷新浏览器，现在星星消失了，这是因为除了Jason之外，所有用户的流量都被路由到 reviews:v1。

您已成功配置Istio以根据用户身份路由流量。

1. **项目实战**

在实战环节中，我们将完成从⼀个单体项目到微服务的拆分，再到服务网格的演变。

* 1. 项目说明

项目是仿照豆瓣电影网址实现的电影信息网，其底层数据使用的Neo4j（图数据库）存储，主体技术使用SpringBoot实现。，我们将其分析，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：

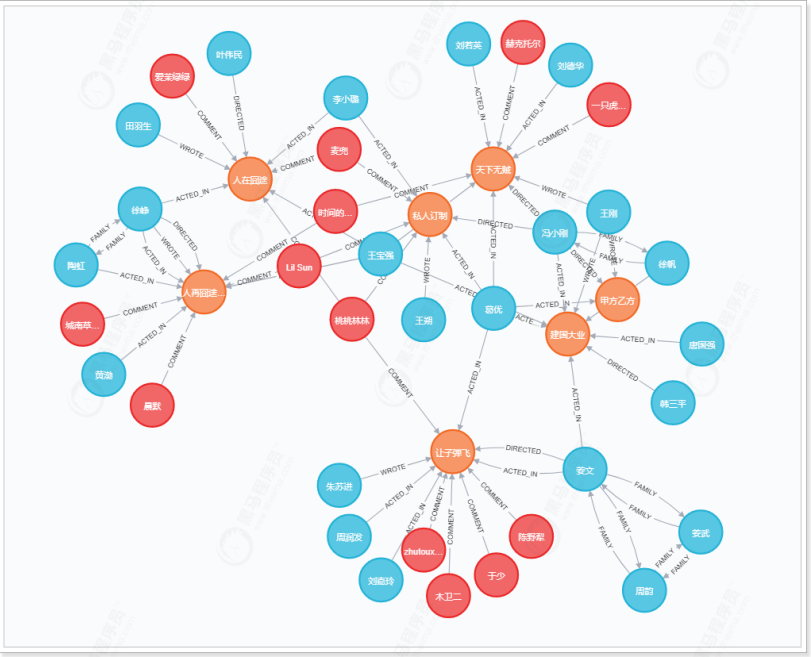


图5-1b

橙色球：电影，蓝色球：参与⼈，红色球：普通用户；

前端项目结构如下图5-1c：

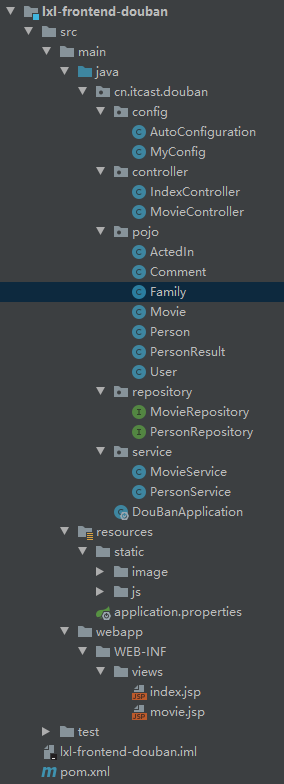


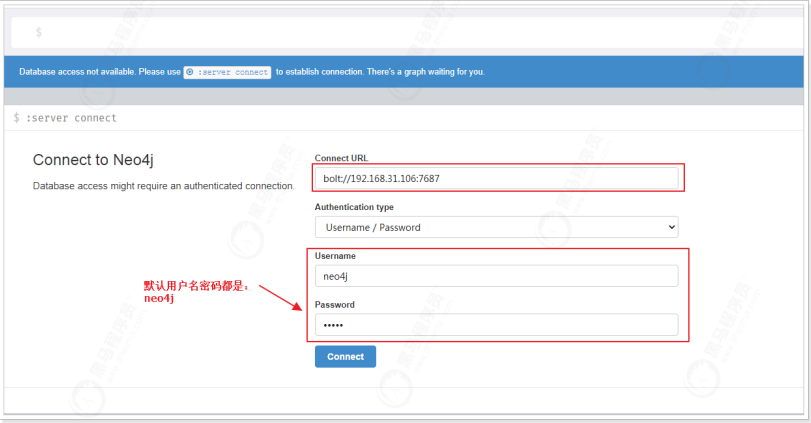
图5-1c

* 1. 搭建Neo4J

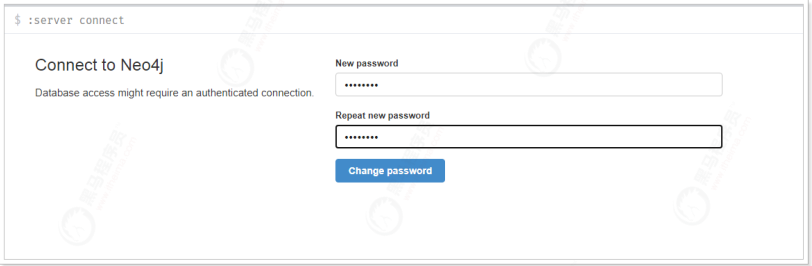
使用docker进行搭建Neo4j环境：

|  |
| --- |
| #创建容器  docker create --name neo4j -p 7474:7474 -p 7687:7687 -v neo4j:/data neo4j:4.0.0  #启动容器  docker start neo4j |

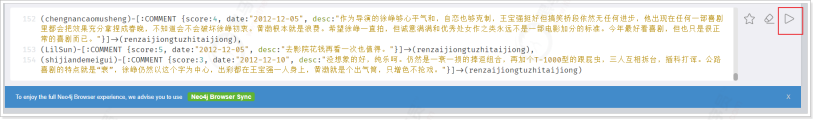
通过http://192.168.31.106:7474/browser/进行登录：



⾸次登录需要修改密码，设置为：neo4j123



在上面的命令输⼊框中写⼊数据脚本，点击右三⻆进行执⾏：

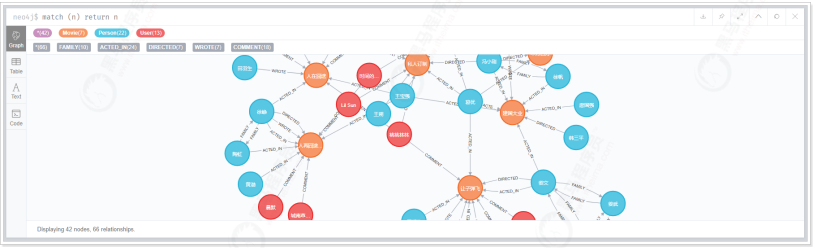


数据⽂件在资料目录下的 <电影数据.txt> ⽂件中。



执⾏成功。

执⾏查询，命令：match (n) return n



* 1. 创建工程

创建后端微服务工程，工程结构如下图5-3：

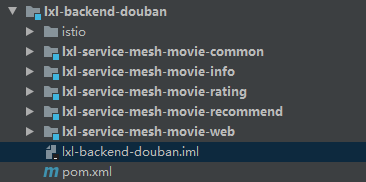


图5-3

* + 1. lxl-backend-mesh

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  [http://maven.apache.org/xsd/maven-4.0.0.xsd">](http://maven.apache.org/xsd/maven-4.0.0.xsd\">)  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.2.5.RELEASE</version>  <relativePath/>  </parent>  <packaging>pom</packaging>  <modules>  <module>lxl-service-mesh-movie-web</module>  <module>lxl-service-mesh-movie-info</module>  <module>lxl-service-mesh-movie-recommend</module>  <module>lxl-service-mesh-movie-rating</module>  <module>lxl-service-mesh-movie-common</module>  </modules>  <groupId>cn.lxl.servicemesh</groupId>  <artifactId>lxl-backend-douban</artifactId>  <version>1.0-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <!--测试-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!--SDN-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-neo4j</artifactId>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.12</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build>  </project> |

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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>lxl-backend-douban</artifactId>  <groupId>cn.lxl.servicemesh</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>lxl-service-mesh-movie-common</artifactId>  </project> |

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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>lxl-backend-douban</artifactId>  <groupId>cn.lxl.servicemesh</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>lxl-service-mesh-movie-info</artifactId>  <dependencies>  <dependency>  <groupId>cn.lxl.servicemesh</groupId>  <artifactId>lxl-service-mesh-movie-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <mainClass>cn.lxl.movie.MovieInfoApplication</mainClass>  <outputDirectory>D:\\publish</outputDirectory>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>lxl-backend-douban</artifactId>  <groupId>cn.lxl.servicemesh</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>lxl-service-mesh-movie-rating</artifactId>  <dependencies>  <dependency>  <groupId>cn.lxl.servicemesh</groupId>  <artifactId>lxl-service-mesh-movie-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>    <mainClass>cn.lxl.movie.MovieRatingApplication</mainClass>  <outputDirectory>D:\\publish</outputDirectory>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>lxl-backend-douban</artifactId>  <groupId>cn.lxl.servicemesh</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>lxl-service-mesh-movie-recommend</artifactId>  <dependencies>  <dependency>  <groupId>cn.lxl.servicemesh</groupId>  <artifactId>lxl-service-mesh-movie-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>    <mainClass>cn.lxl.movie.MovieRecommendApplication</mainClass>  <outputDirectory>D:\\publish</outputDirectory>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>lxl-backend-douban</artifactId>  <groupId>cn.lxl.servicemesh</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>lxl-service-mesh-movie-web</artifactId>  <dependencies>  <dependency>  <groupId>cn.lxl.servicemesh</groupId>  <artifactId>lxl-service-mesh-movie-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <!--⽀持jsp-->  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  <!--⽀持热部署修改jsp⽴即生效-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <version>1.4.2.RELEASE</version>  <configuration>  <mainClass>cn.lxl.movie.MovieApplication</mainClass>  <outputDirectory>D:\\publish</outputDirectory>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  <resources>  <resource>  <directory>src/main/webapp</directory>  <targetPath>META-INF/resources</targetPath>  <includes>  <include>\*\*/\*\*</include>  </includes>  </resource>  <resource>  <directory>src/main/resources</directory>  <filtering>false</filtering>  <includes>  <include>\*\*/\*\*</include>  </includes>  </resource>  </resources>  </build>  </project> |

* 1. movie-common

lxl-service-mesh-movie-common工程中，存放了公用的pojo、vo对象,如下图5-4：。

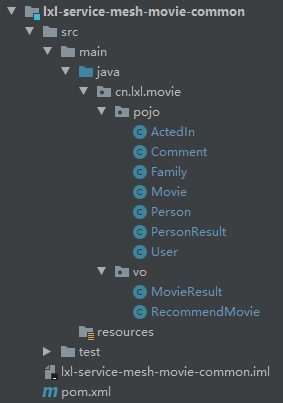


图5-4

* 1. movie-info

该工程进行对象电影信息进行查询。

* + 1. Config

|  |
| --- |
| package cn.lxl.movie.config;  import lombok.Setter;  import org.springframework.boot.autoconfigure.data.neo4j.Neo4jProperties;  import org.springframework.boot.autoconfigure.domain.EntityScan;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  Import org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;  import org.springframework.transaction.annotation.EnableTransactionManagement;  @Configuration  @EnableNeo4jRepositories(basePackages = "cn.lxl.movie.repository")  @EntityScan("cn.lxl.movie.pojo") //配置实体扫描包  @EnableTransactionManagement //激活事务管理  @ConfigurationProperties(prefix = "spring.data.neo4j")  @Setter //设置set方法，如果不设置，值不会被注⼊  public class AutoConfiguration {  private Integer connectionPoolSize;  private Integer connectionLivenessCheckTimeout;  /\*\*  \* 自定义配置  \*  \* @param neo4jProperties  \* @return  \*/  @Bean  public org.neo4j.ogm.config.Configuration configuration(Neo4jProperties neo4jProperties) {  org.neo4j.ogm.config.Configuration.Builder builder = new  org.neo4j.ogm.config.Configuration.Builder();  builder.uri(neo4jProperties.getUri())  .credentials(neo4jProperties.getUsername(),  neo4jProperties.getPassword())  .connectionPoolSize(connectionPoolSize)    .connectionLivenessCheckTimeout(connectionLivenessCheckTimeout);  return builder.build();  }  } |
| package cn.lxl.movie.config;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  @ComponentScan(basePackages = "cn.lxl")  @Configuration  public class MyConfig implements WebMvcConfigurer {  } |

* + 1. Repository

|  |
| --- |
| package cn.lxl.movie.repository;  import cn.lxl.movie.pojo.Movie;  import org.springframework.data.neo4j.repository.Neo4jRepository;  public interface MovieRepository extends Neo4jRepository<Movie, Long> {  /\*\*根据电影id查询数据  \*  \* @param movieId  \* @return  \*/  Movie findByMovieId(Long movieId);  } |

* + 1. Service

|  |
| --- |
| package cn.lxl.movie.service;  import cn.lxl.movie.pojo.Movie;  import cn.lxl.movie.repository.MovieRepository;  import cn.lxl.movie.vo.MovieResult;  import org.springframework.beans.BeanUtils;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  @Service  public class MovieService {  @Autowired  private MovieRepository movieRepository;  public MovieResult queryByMovieId(Long movieId) {  MovieResult movieResult = new MovieResult();  Movie movie = this.movieRepository.findByMovieId(movieId);  //对象数据的拷⻉  BeanUtils.copyProperties(movie, movieResult);  return movieResult;  }  } |

* + 1. Controller

|  |
| --- |
| package cn.lxl.movie.controller;  import cn.lxl.movie.service.MovieService;  import cn.lxl.movie.vo.MovieResult;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class MovieController {  @Autowired  private MovieService movieService;  @RequestMapping("movie")  public MovieResult index(@RequestParam(value = "id", defaultValue =  "3742360") Long movieId) {  return this.movieService.queryByMovieId(movieId);  }  } |

* + 1. 启动类

|  |
| --- |
| package cn.lxl.movie;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class MovieInfoApplication {  public static void main(String[] args) {  SpringApplication.run(MovieInfoApplication.class, args);  }  } |

* + 1. 配置

application.properties：

|  |
| --- |
| server.port=18082  #连接地址  spring.data.neo4j.uri=bolt://neo4j-server:7687  #用户名  spring.data.neo4j.username=neo4j  #密码  spring.data.neo4j.password=neo4j123  #连接池⼤⼩  spring.data.neo4j.connection-pool-size=200  #测试超时时间  spring.data.neo4j.connection-liveness-check-timeout=100  #logging.level.org.springframework=DEBUG |

* + 1. 测试

电影信息查询接口测试结果如下图5-5-7：



图5-5-7

* 1. movie-recommend
     1. Config

与movie-info工程中⼀样，这⾥就省略了。

* + 1. Repository

|  |
| --- |
| package cn.lxl.movie.repository;  import cn.lxl.movie.pojo.Movie;  import org.springframework.data.neo4j.annotation.Query;  import org.springframework.data.neo4j.repository.Neo4jRepository;  import org.springframework.data.repository.query.Param;  import java.util.List;  public interface MovieRepository extends Neo4jRepository<Movie, Long> {  /\*\*  \* 喜欢这部电影的⼈也喜欢  \*  \* @param movieId  \* @return  \*/  @Query("match (m:Movie) <-- (User) --> (x:Movie)\n" +  "where m.movieId = $movieId and m.movieId <> x.movieId\n" +  "return x")  List<Movie> recommentMovie(@Param("movieId") Long movieId);  } |

* + 1. Service

|  |
| --- |
| package cn.lxl.movie.service;  import cn.lxl.movie.pojo.Movie;  import cn.lxl.movie.repository.MovieRepository;  import cn.lxl.movie.vo.RecommendMovie;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.util.CollectionUtils;  import java.util.ArrayList;  import java.util.List;  @Service  public class MovieService {  @Autowired  private MovieRepository movieRepository;  public List<RecommendMovie> recommendMovies(Long movieId) {  List<Movie> movieList = this.movieRepository.recommentMovie(movieId);  if (CollectionUtils.isEmpty(movieList)) {  return new ArrayList<>(0);  }  List<RecommendMovie> list = new ArrayList<>();  for (Movie movie : movieList) {  RecommendMovie recommendMovie = new RecommendMovie();  recommendMovie.setMovieId(movie.getMovieId());  recommendMovie.setPic(movie.getPic());  recommendMovie.setTitle(movie.getTitle());  list.add(recommendMovie);  }  return list;  }  } |

* + 1. Controller

|  |
| --- |
| package cn.lxl.movie.controller;  import cn.lxl.movie.service.MovieService;  import cn.lxl.movie.vo.RecommendMovie;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  import java.util.List;  @RestController  public class MovieController {  @Autowired  private MovieService movieService;  @RequestMapping("movie/recommend")  public List<RecommendMovie> index(@RequestParam(value = "id", defaultValue  = "3742360") Long movieId) {  return this.movieService.recommendMovies(movieId);  }  } |

* + 1. 启动类

|  |
| --- |
| package cn.lxl.movie.controller;  import cn.lxl.movie.service.MovieService;  import cn.lxl.movie.vo.RecommendMovie;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  import java.util.List;  @RestController  public class MovieController {  @Autowired  private MovieService movieService;  @RequestMapping("movie/recommend")  public List<RecommendMovie> index(@RequestParam(value = "id", defaultValue  = "3742360") Long movieId) {  return this.movieService.recommendMovies(movieId);  }  } |

* + 1. 配置

application.properties

|  |
| --- |
| server.port=18083  #连接地址  spring.data.neo4j.uri=bolt://neo4j-server:7687  #用户名  spring.data.neo4j.username=neo4j  #密码  spring.data.neo4j.password=neo4j123  #连接池⼤⼩  spring.data.neo4j.connection-pool-size=200  #测试超时时间  spring.data.neo4j.connection-liveness-check-timeout=100  #logging.level.org.springframework=DEBUG |

* + 1. 测试

推荐影片接口测试结果如下图5-6-7：

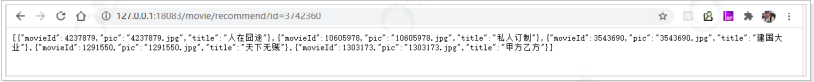


图5-6-7

* 1. movie-rating

电影评分会有2个版本，如果系统环境变量中有MOVIE\_VERSION的值为v2时，评分显示红色，否则显示 黑色。

* + 1. Config

与movie-info工程中⼀样，这⾥就省略了。

* + 1. Repository

|  |
| --- |
| package cn.lxl.movie.repository;  import cn.lxl.movie.pojo.Movie;  import org.springframework.data.neo4j.repository.Neo4jRepository;  public interface MovieRepository extends Neo4jRepository<Movie, Long> {  /\*\*根据业务id查询数据  \*  \* @param movieId  \* @return  \*/  Movie findByMovieId(Long movieId);  } |

* + 1. Service

|  |
| --- |
| package cn.lxl.movie.service;  import cn.lxl.movie.pojo.Comment;  import cn.lxl.movie.pojo.Movie;  import cn.lxl.movie.repository.MovieRepository;  import org.apache.commons.lang3.StringUtils;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.util.CollectionUtils;  import java.math.BigDecimal;  import java.math.RoundingMode;  import java.util.List;  import java.util.Map;  @Service  public class MovieService {  @Autowired  private MovieRepository movieRepository;  public String movieRating(Long movieId) {  Movie movie = this.movieRepository.findByMovieId(movieId);  List<Comment> commentList = movie.getComments();  if(CollectionUtils.isEmpty(commentList)){  return "";  }  int total = 0;  for (Comment comment : commentList) {  total+=comment.getScore() \* 2;  }  //计算评分  double score = total / commentList.size();  //计算评分星级  BigDecimal bg = new BigDecimal(score \* 5).setScale(2, RoundingMode.UP);  int i = bg.intValue();  String bigstar = "bigstar";  if(i % 5 ==0){  bigstar += i;  }else{  bigstar += Math.round(i/10D)\*10;  }  String v2Style = System.getenv("MOVIE\_VERSION");  if(StringUtils.equals(v2Style, "v2")){  v2Style = "style=\"color: red;\"";  }else{  v2Style = "";  }  String html = "<div id=\"interest\_sectl\">\n" +  "\t<div class=\"rating\_wrap clearbox\" rel=\"v:rating\">\n" +  "\t\t<div class=\"clearfix\">\n" +  "\t\t\t<div class=\"rating\_logo ll\">⿊⻢评分</div>\n" +  "\t\t</div>\n" +  "\t\t<div class=\"rating\_self clearfix\"  typeof=\"v:Rating\">\n" +  "\t\t\t<strong class=\"ll rating\_num\" property=\"v:average\"  "+v2Style+">"+String.format("%.1f", score)+"</strong>\n" +  "\t\t\t<span property=\"v:best\" content=\"10.0\"></span>\n" +  "\t\t\t<div class=\"rating\_right \">\n" +  "\t\t\t\t<div class=\"ll bigstar "+bigstar+"\"></div>\n" +  "\t\t\t\t<div class=\"rating\_sum\">\n" +  "\t\t\t\t\t<a href=\"collections\" class=\"rating\_people\">\n"  +  "\t\t\t\t\t\t<span property=\"v:votes\">"+commentList.size()+"  </span>⼈评价\n" +  "\t\t\t\t\t</a>\n" +  "\t\t\t\t</div>\n" +  "\t\t\t</div>\n" +  "\t\t</div>\n" +  "\t</div>\n" +  "</div>";  return html;  }  public static void main(String[] args) {  Map<String, String> map = System.getenv();  map.forEach((s, s2) -> System.out.println(s +" " + s2));  }  } |

* + 1. Controller

|  |
| --- |
| package cn.lxl.movie.controller;  import cn.lxl.movie.service.MovieService;  import cn.lxl.movie.vo.RecommendMovie;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  import java.util.List;  @RestController  public class MovieController {  @Autowired  private MovieService movieService;  @RequestMapping("movie/rating")  public String index(@RequestParam(value = "id", defaultValue = "3742360")  Long movieId) {  return this.movieService.movieRating(movieId);  }  } |

* + 1. 启动

|  |
| --- |
| package cn.lxl.movie;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class MovieRatingApplication {  public static void main(String[] args) {  SpringApplication.run(MovieRatingApplication.class, args);  }  } |

* + 1. 配置

|  |
| --- |
| server.port=18084  #连接地址  spring.data.neo4j.uri=bolt://neo4j-server:7687  #用户名  spring.data.neo4j.username=neo4j  #密码  spring.data.neo4j.password=neo4j123  #连接池⼤⼩  spring.data.neo4j.connection-pool-size=200  #测试超时时间  spring.data.neo4j.connection-liveness-check-timeout=100  #logging.level.org.springframework=DEBUG |

* + 1. 测试

评分接口测试结果如下图5-7-7：

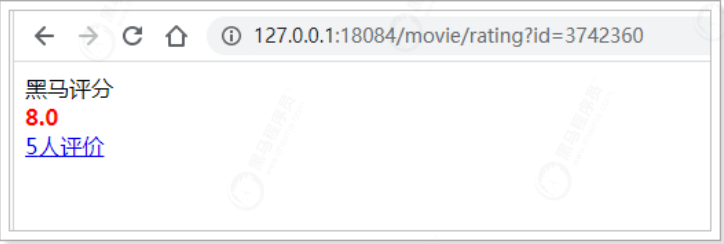


图5-7-7

* 1. movie-web
     1. 静态资源

将获取的静态资源导⼊工程，结构如下图5-8-1：



图5-8-1

* + 1. 配置

application.properties

|  |
| --- |
| server.port=18081  spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  #logging.level.org.springframework=DEBUG |

* + 1. Config

|  |
| --- |
| package cn.lxl.movie.config;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  @ComponentScan(basePackages = "cn.lxl")  @Configuration  public class MyConfig implements WebMvcConfigurer {  } |
| package cn.lxl.movie.config;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.http.client.ClientHttpRequestFactory;  import org.springframework.http.client.SimpleClientHttpRequestFactory;  import org.springframework.http.converter.StringHttpMessageConverter;  import org.springframework.web.client.RestTemplate;  import java.nio.charset.Charset;  @Configuration  public class RestTemplateConfig {  @Bean  public RestTemplate restTemplate(ClientHttpRequestFactory factory) {  RestTemplate restTemplate = new RestTemplate(factory);  // ⽀持中⽂编码  restTemplate.getMessageConverters().set(1, new  StringHttpMessageConverter(Charset.forName("UTF-8")));  return restTemplate;  }  @Bean  public ClientHttpRequestFactory simpleClientHttpRequestFactory() {  SimpleClientHttpRequestFactory factory = new  SimpleClientHttpRequestFactory();  factory.setReadTimeout(5000);//单位为ms  factory.setConnectTimeout(5000);//单位为ms  return factory;  }  } |

* + 1. Service

|  |
| --- |
| package cn.lxl.movie.service;  import cn.lxl.movie.vo.MovieResult;  import cn.lxl.movie.vo.RecommendMovie;  import com.fasterxml.jackson.core.JsonProcessingException;  import com.fasterxml.jackson.databind.ObjectMapper;  import com.fasterxml.jackson.databind.type.CollectionType;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.web.client.RestTemplate;  import java.util.List;  @Service  public class MovieService {  @Autowired  private RestTemplate restTemplate;  private static final ObjectMapper MAPPER = new ObjectMapper();  private final static String SERVICES\_DOMAIN =  System.getenv("SERVICES\_DOMAIN") == null ? "" : ("." +  System.getenv("SERVICES\_DOMAIN"));    //info服务  private final static String INFO\_HOSTNAME = System.getenv("INFO\_HOSTNAME")  == null ? "movie-info" : System.getenv("INFO\_HOSTNAME");  private final static String INFO\_URL = "http://" + INFO\_HOSTNAME +  SERVICES\_DOMAIN + ":18082/movie";  //recommend服务  private final static String RECOMMEND\_HOSTNAME =  System.getenv("RECOMMEND\_HOSTNAME") == null ? "movie-recommend" :  System.getenv("RECOMMEND\_HOSTNAME");  private final static String RECOMMEND\_URL = "http://" + RECOMMEND\_HOSTNAME  + SERVICES\_DOMAIN + ":18083/movie/recommend";  //rating服务  private final static String RATING\_HOSTNAME =  System.getenv("RATING\_HOSTNAME") == null ? "movie-rating" :  System.getenv("RATING\_HOSTNAME");  private final static String RATING\_URL = "http://" + RATING\_HOSTNAME +  SERVICES\_DOMAIN + ":18084/movie/rating";    public MovieResult queryByMovieId(Long movieId) {  String jsonData = this.restTemplate.getForObject(INFO\_URL + "?id=" +  movieId, String.class);  try {  MovieResult movieResult = MAPPER.readValue(jsonData,  MovieResult.class);  return movieResult;  } catch (JsonProcessingException e) {  e.printStackTrace();  }  return null;  }  public String recommendMovie(Long movieId) {  String jsonData = this.restTemplate.getForObject(RECOMMEND\_URL + "?id="  + movieId, String.class);  try {  CollectionType javaType =  MAPPER.getTypeFactory().constructCollectionType(List.class,  RecommendMovie.class);  List<RecommendMovie> recommendMovies = MAPPER.readValue(jsonData,  javaType);  StringBuilder sb = new StringBuilder();  for (RecommendMovie movie : recommendMovies) {  sb.append("<dl class=\"\">\n" +  " <dt>\n" +  " <a href=\"/index?id="+movie.getMovieId()+"\">\n" +  " <img src=\"/image/movie/"+movie.getPic()+"\"/>\n" +  " </a>\n" +  " </dt>\n" +  " <dd>\n" +  " <a href=\"/index?id="+movie.getMovieId()+"\"  class=\"\">"+movie.getTitle()+"</a>\n" +  " </dd>\n" +  " </dl>");  }  return sb.toString();  } catch (JsonProcessingException e) {  e.printStackTrace();  }  return null;  }  public String ratingMovie(Long movieId) {  return this.restTemplate.getForObject(RATING\_URL + "?id=" + movieId,  String.class);  }  } |

* + 1. Controller

|  |
| --- |
| package cn.lxl.movie.controller;  import cn.lxl.movie.service.MovieService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.servlet.ModelAndView;  /\*\*  \* 影⽚⻚  \*/  @Controller  public class MovieController {  @Autowired  private MovieService movieService;  @RequestMapping("index")  public ModelAndView index(@RequestParam(value = "id", defaultValue =  "3742360") Long movieId) {  ModelAndView mv = new ModelAndView("index");  //查询电影信息  mv.addObject("movie", this.movieService.queryByMovieId(movieId));  //电影推荐  mv.addObject("recommendMovie",  this.movieService.recommendMovie(movieId));  //电影评分  mv.addObject("ratingMovie", this.movieService.ratingMovie(movieId));  return mv;  }  } |

* + 1. 启动类

|  |
| --- |
| package cn.lxl.movie;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class MovieApplication {  public static void main(String[] args) {  SpringApplication.run(MovieApplication.class, args);  }  } |

* + 1. 测试



* 1. 制作docker镜像

编写Dockerfile：

|  |
| --- |
| #lxl-service-mesh-movie-info  FROM openjdk:8-jdk-alpine  COPY ./lxl-service-mesh-movie-info-1.0-SNAPSHOT.jar /movie/lxl-servicemesh-movie-info-1.0-SNAPSHOT.jar  EXPOSE 18082  ENTRYPOINT [ "java","-jar","/movie/lxl-service-mesh-movie-info-1.0-  SNAPSHOT.jar" ]  #lxl-service-mesh-movie-rating  FROM openjdk:8-jdk-alpine  COPY ./lxl-service-mesh-movie-rating-1.0-SNAPSHOT.jar /movie/lxl-servicemesh-movie-rating-1.0-SNAPSHOT.jar  EXPOSE 18084  ENTRYPOINT [ "java","-jar","/movie/lxl-service-mesh-movie-rating-1.0-  SNAPSHOT.jar" ]  #lxl-service-mesh-movie-recommend  FROM openjdk:8-jdk-alpine  COPY ./lxl-service-mesh-movie-recommend-1.0-SNAPSHOT.jar /movie/lxlservice-mesh-movie-recommend-1.0-SNAPSHOT.jar  EXPOSE 18083  ENTRYPOINT [ "java","-jar","/movie/lxl-service-mesh-movie-recommend-1.0-  SNAPSHOT.jar" ]  #lxl-service-mesh-movie-web  FROM openjdk:8-jdk-alpine  COPY ./lxl-service-mesh-movie-web-1.0-SNAPSHOT.jar /movie/lxl-servicemesh-movie-web-1.0-SNAPSHOT.jar  EXPOSE 18081  ENTRYPOINT [ "java","-jar","/movie/lxl-service-mesh-movie-web-1.0-  SNAPSHOT.jar" ] |

制作镜像：

|  |
| --- |
| mkdir -p /movie/movie-info  cd /movie/movie-info  vim Dockerfile  #拷⻉上面的内容到⽂件中  #构建镜像  docker build -t lxl-service-mesh-movie-info:1.0 .  #基于此镜像创建容器，进行测试  docker create --name movie-info -p 18082:18082 --add-host=neo4jserver:192.168.31.106 lxl-service-mesh-movie-info:1.0  docker start movie-info  #打开浏览器进行测试：http://192.168.31.106:18082/movie  #测试成功后即可把容器停⽌删除  docker stop movie-info  docker rm movie-info  docker build -t lxl-service-mesh-movie-recommend:1.0 .  docker create --name movie-recommend -p 18083:18083 --add-host=neo4jserver:192.168.31.106 lxl-service-mesh-movie-recommend:1.0  docker start movie-recommend  docker build -t lxl-service-mesh-movie-rating:1.0 .  docker create --name movie-rating -p 18084:18084 --add-host=neo4jserver:192.168.31.106 lxl-service-mesh-movie-rating:1.0  docker start movie-rating  docker create --name movie-rating-v2 -p 18085:18084 --add-host=neo4jserver:192.168.31.106 --env MOVIE\_VERSION=v2 lxl-service-mesh-movierating:1.0  docker start movie-rating-v2  docker build -t lxl-service-mesh-movie-web:1.0 .  docker create --name movie-web -p 18081:18081 --add-host=movieinfo:192.168.31.106 --add-host=movie-recommend:192.168.31.106 --add-host=movierating:192.168.31.106 lxl-service-mesh-movie-web:1.0  docker start movie-web  #为了方便使用，将镜像上传到阿⾥云镜像仓库，这个是免费的，自己可以申请即可  #登录到阿⾥云，改成自己的账号  [root@node1 movie-info]# docker login --username=hi31888179@aliyun.com  registry.cn-hangzhou.aliyuncs.com  Password:  WARNING! Your password will be stored unencrypted in /root/.docker/config.json.  Configure a credential helper to remove this warning. See  https://docs.docker.com/engine/reference/commandline/login/#credentials-store  Login Succeeded  #info  docker tag lxl-service-mesh-movie-info:1.0 registry.cnhangzhou.aliyuncs.com/lxl/lxl-service-mesh-movie-info:1.0  docker push registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-mesh-movieinfo:1.0  #recommend  docker tag lxl-service-mesh-movie-recommend:1.0 registry.cnhangzhou.aliyuncs.com/lxl/lxl-service-mesh-movie-recommend:1.0  docker push registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-mesh-movierecommend:1.0  #rating  docker tag lxl-service-mesh-movie-rating:1.0 registry.cnhangzhou.aliyuncs.com/lxl/lxl-service-mesh-movie-rating:1.0  movie.yaml  docker push registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-mesh-movierating:1.0  #web  docker tag lxl-service-mesh-movie-web:1.0 registry.cnhangzhou.aliyuncs.com/lxl/lxl-service-mesh-movie-web:1.0  docker push registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-mesh-movieweb:1.0 |

* 1. 编写Istio相关⽂件
     1. k8s编排容器

movie.yaml

|  |
| --- |
| ###############################################################################  ###################  # neo4j service  ###############################################################################  ###################  apiVersion: v1  kind: Service  metadata:  name: neo4j-server  spec:  ports:  - port: 7474  name: http  nodePort: 31001  - port: 7687  name: blot  nodePort: 31002  selector:  app: neo4j-app  type: NodePort  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: neo4j-deployment  labels:  app: neo4j-app  spec:  replicas: 1  selector:  matchLabels:  app: neo4j-app  template:  metadata:  labels:  app: neo4j-app  spec:  containers:  - name: neo4j  image: neo4j:4.0.0  imagePullPolicy: IfNotPresent  ports:  - containerPort: 7474  - containerPort: 7687  ---  ###############################################################################  ###################  # movie info service  ###############################################################################  ###################  apiVersion: v1  kind: Service  metadata:  name: movie-info  spec:  ports:  - port: 18082  name: http  selector:  app: movie-info-app  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: movie-info-deployment  spec:  replicas: 2  selector:  matchLabels:  app: movie-info-app  version: v1  template:  metadata:  labels:  app: movie-info-app  version: v1  spec:  containers:  - name: movie-info  image: registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-meshmovie-info:1.0  imagePullPolicy: IfNotPresent  ports:  - containerPort: 18082  ---  ###############################################################################  ###################  # movie recommend service  ###############################################################################  ###################  apiVersion: v1  kind: Service  metadata:  name: movie-recommend  spec:  ports:  - port: 18083  name: http  selector:  app: movie-recommend-app  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: movie-recommend-deployment  spec:  replicas: 3  selector:  matchLabels:  app: movie-recommend-app  version: v1  template:  metadata:  labels:  app: movie-recommend-app  version: v1  spec:  containers:  - name: movie-recommend  image: registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-meshmovie-recommend:1.0  imagePullPolicy: IfNotPresent  ports:  - containerPort: 18083  ---  ###############################################################################  ###################  # movie rating service  ###############################################################################  ###################  apiVersion: v1  kind: Service  metadata:  name: movie-rating  spec:  ports:  - port: 18084  name: http  selector:  app: movie-rating-app  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: movie-rating-deployment-v1  spec:  replicas: 1  selector:  matchLabels:  app: movie-rating-app  version: v1  template:  metadata:  labels:  app: movie-rating-app  version: v1  spec:  containers:  - name: movie-rating  image: registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-meshmovie-rating:1.0  imagePullPolicy: IfNotPresent  ports:  - containerPort: 18083  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: movie-rating-deployment-v2  spec:  replicas: 1  selector:  matchLabels:  app: movie-rating-app  version: v2  template:  metadata:  labels:  app: movie-rating-app  version: v2  spec:  containers:  - name: movie-rating  image: registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-meshmovie-rating:1.0  imagePullPolicy: IfNotPresent  ports:  - containerPort: 18083  env:  - name: MOVIE\_VERSION  value: "v2"  ---  ###############################################################################  ###################  # movie web service  ###############################################################################  ###################  apiVersion: v1  kind: Service  metadata:  name: movie-web  spec:  ports:  - port: 18081  name: http  selector:  app: movie-web-app  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: movie-web-deployment  spec:  replicas: 3  selector:  matchLabels:  app: movie-web-app  version: v1  template:  metadata:  labels:  app: movie-web-app  version: v1  spec:  containers:  - name: movie-web  image: registry.cn-hangzhou.aliyuncs.com/lxl/lxl-service-meshmovie-web:1.0  imagePullPolicy: IfNotPresent  ports:  destination-rule-all.yaml：  - containerPort: 18081  --- |

* + 1. 网关

movie-gateway.yaml

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: Gateway  metadata:  name: movie-gateway  spec:  selector:  istio: ingressgateway # use istio default controller  servers:  - port:  number: 80  name: http  protocol: HTTP  hosts:  - "\*"  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: movie  spec:  hosts:  - "\*"  gateways:  - movie-gateway  http:  - match:  - uri:  prefix: /index  - uri:  prefix: /image  - uri:  prefix: /js  route:  - destination:  host: movie-web  port:  number: 18081 |

* + 1. 默认路由规则

destination-rule-all.yaml：

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: movie-web  spec:  host: movie-web  subsets:  - name: v1  labels:  version: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: movie-info  spec:  host: movie-info  subsets:  - name: v1  labels:  version: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: movie-recommend  spec:  host: movie-recommend  subsets:  - name: v1  labels:  version: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: movie-rating  spec:  host: movie-rating  subsets:  - name: v1  labels:  version: v1  - name: v2  labels:  version: v2  ---  apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: movie-web  spec:  host: movie-web  subsets:  - name: v1  labels:  version: v1  --- |

* + 1. 流量全部导入v2

virtual-service-rating-v2.yaml：

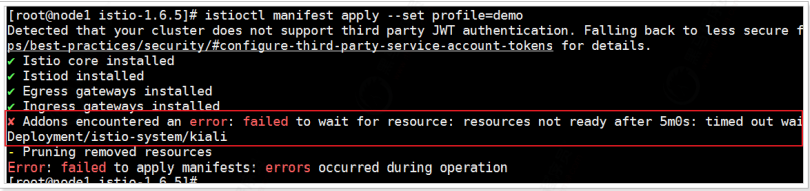
|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: movie-rating  spec:  hosts:  - movie-rating  http:  - route:  - destination:  host: movie-rating  subset: v2 |

* 1. 实施部署

实施部署前，建议按照前⾯的⽂档，重新搭建k8s环境以及Istio环境，再开始部署。

* + 1. Istio初始化失败

如果出现如下错误：



解决方案如下：

|  |
| --- |
| #说明：默认使用的kiali镜像是从quay.io拉取，有些时候是拉取不下来的  #解决方案：从阿⾥云仓库拉取，再重新打tag即可  docker pull registry.cn-hangzhou.aliyuncs.com/lxl/kiali:v1.18  docker tag registry.cn-hangzhou.aliyuncs.com/lxl/kiali:v1.18  quay.io/kiali/kiali:v1.18  #需要注意的是，通过 kubectl get pod -n istio-system -o wide 定位到pod所在的机器，再执  ⾏上面的操作 |

* + 1. 实施

|  |
| --- |
| mkdir /movie  cd /movie/  #将destination-rule-all.yaml movie-gateway.yaml movie.yaml virtual-servicerating-v2.yaml ⽂件上传到该目录  #设置自动注⼊  kubectl label namespace default istio-injection=enabled  #部署应用  [root@node1 movie]# kubectl apply -f movie.yaml  service/neo4j-server created  deployment.apps/neo4j-deployment created  service/movie-info created  deployment.apps/movie-info-deployment created  service/movie-recommend created  deployment.apps/movie-recommend-deployment created  service/movie-rating created  deployment.apps/movie-rating-deployment-v1 created  deployment.apps/movie-rating-deployment-v2 created  service/movie-web created  deployment.apps/movie-web-deployment created  #查看pod，可以看到正在初始化中  [root@node1 movie]# kubectl get pod -o wide  NAME READY STATUS  RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  movie-info-deployment-c8f455f69-4hknf 0/2 Init:0/1 0  40s <none> node3 <none> <none>  movie-info-deployment-c8f455f69-psbb4 0/2 Init:0/1 0  40s <none> node2 <none> <none>  movie-rating-deployment-v1-667fb7c8cc-j8ggj 0/2 PodInitializing 0  40s 10.244.1.8 node2 <none> <none>  movie-rating-deployment-v2-577d48b98f-fg2nj 0/2 Init:0/1 0  40s <none> node3 <none> <none>  movie-recommend-deployment-747d6dc646-8zv7j 0/2 PodInitializing 0  40s 10.244.1.7 node2 <none> <none>  movie-recommend-deployment-747d6dc646-g9bcb 0/2 PodInitializing 0  40s 10.244.2.7 node3 <none> <none>  movie-recommend-deployment-747d6dc646-sxbj7 0/2 Init:0/1 0  40s <none> node3 <none> <none>  movie-web-deployment-6996d99464-cc88t 0/2 Init:0/1 0  40s <none> node2 <none> <none>  movie-web-deployment-6996d99464-gjfm6 0/2 Init:0/1 0  40s <none> node3 <none> <none>  movie-web-deployment-6996d99464-t29tz 0/2 Init:0/1 0  40s <none> node2 <none> <none>  neo4j-deployment-6bb95596c8-sld6n 0/2 PodInitializing 0  40s 10.244.2.6 node3 <none> <none>  #查看服务，可以看到neo4j-server是NodePort类型，外部可以直接访问  [root@node1 movie]# kubectl get service -o wide  NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)  AGE SELECTOR  kubernetes ClusterIP 10.96.0.1 <none> 443/TCP  3h15m <none>  movie-info ClusterIP 10.108.31.87 <none> 18082/TCP  117s app=movie-info-app  movie-rating ClusterIP 10.107.230.15 <none> 18084/TCP  117s app=movie-rating-app  movie-recommend ClusterIP 10.97.144.103 <none> 18083/TCP  117s app=movie-recommend-app  movie-web ClusterIP 10.104.57.57 <none> 18081/TCP  117s app=movie-web-app  neo4j-server NodePort 10.106.196.213 <none>  7474:31001/TCP,7687:31002/TCP 117s app=neo4j-app |

导⼊Neo4j数据：http://192.168.31.106:31001/browser