

课程说明

- BookInfo示例分析
 - 为什么要配置网关、虚拟服务？
 - 服务之间是如何通信的？
- Istio进阶
 - Sidecar流量接管原理
 - 超时与重试
 - 熔断器
 - 可视化网络
- 实战
 - 单体架构向服务网格的演变

1、BookInfo示例分析

前面我们已经将BookInfo进行了部署以及一些功能的演示，从大体上理解了Istio到底是什么，在解决什么问题。下面我们针对BookInfo示例来解释一些疑惑。

1.1、为什么要配置网关、虚拟服务？

1.1.1、概念

首先，需要了解网关以及虚拟服务的概念。

- 网关 (Gateway)
 - 网关是服务网格的边界，用于处理HTTP、TCP 入口与出口流量，Service Mesh 中的服务只能通过网关对外暴露接口，以方便管控，其配置中可以定义暴露的端口以及传输层面上的配置（是否需要启用 TLS）。
 - BookInfo 配置了一个对外暴露 80 端口的网关服务，并不绑定任何域名（这样才能以 IP 方式进行访问）。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - "*"

```

- 如果指定了hosts，那么必须通过该域名进入

```

apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    app: my-gateway-controller
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - ext-host.example.com
    tls:
      mode: SIMPLE
      serverCertificate: /tmp/tls.crt
      privateKey: /tmp/tls.key

```

- 虚拟服务

- 虚拟服务就是配置如何在服务网格内将请求路由到服务，这基于 Istio 和平台提供的基本的连接性和服务发现能力。每个虚拟服务包含一组路由规则，Istio 按顺序评估它们，Istio 将每个给定的请求匹配到虚拟服务指定的实际目标地址。

```

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: v3

```

- hosts: 虚拟服务主机名可以是 IP 地址、DNS 名称, 或者依赖于平台的一个简称 (例如 Kubernetes 服务的短名称), 也可以使用通配符 (“*”) 前缀, 创建一组匹配所有服务的路由规则。
- 路由规则
 - 在 http 字段包含了虚拟服务的路由规则, 用来描述匹配条件和路由行为, 它们把 HTTP/1.1、HTTP2 和 gRPC 等流量发送到 hosts 字段指定的目标 (您也可以用 tcp 和 tls 片段为 TCP 和未终止的 TLS 流量设置路由规则)。
 - 示例中的第一个路由规则有一个条件, 因此以 match 字段开始。希望此路由应用于来自 “json” 用户的所有请求, 所以使用 headers、end-user 和 exact 字段选择适当的请求。
 - route 部分的 destination 字段指定了符合此条件的流量的实际目标地址。本例中, 使用的是kubernetes的服务名。
- 路由优先级
 - 路由规则按从上到下的顺序选择, 虚拟服务中定义的第一条规则有最高优先级。
 - 本示例中, 不满足第一个路由规则的流量均流向一个默认的目标, 该目标在第二条规则中指定。因此, 第二条规则没有 match 条件, 直接将流量导向 v3 子集。
- 必须把网关绑定到虚拟服务上, 网关才能生效。 (bookinfo-gateway.yaml)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
            number: 9080
```

1.1.2、IngressGateway

istio-ingressgateway 是 Istio 安装完成后就会启动的服务，其作用是作为整个服务网络的边缘网关，即客户端请求由此进入网格整体。在边缘架设网关的作用不仅是做一层统一代理，更是对服务接口的统一管理。

需要注意的是，这里 Istio 所描述的 IngressGateway 并不是一个应用，其名称为IngressGateway，而是对流量入口网关的一个泛指，与之相对应的还有一个叫作 EgressGateway的出口网关，是对流出流量的统一代理。

在默认情况下，Envoy 是没有任务路由转发规则的，需要在 Pilot 统一配置后，才会生效，而这里的配置正是 Virtual Service。

bookinfo-gateway.yaml:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway #网关名称，在虚拟服务中会使用到
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers: #配置Ingress服务
  - port:
      number: 80 #对外暴露的端口
      name: http
      protocol: HTTP #对外暴露的协议
    hosts: #不加域名的限制，可以直接通过ip地址访问
    - "*"

---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
```

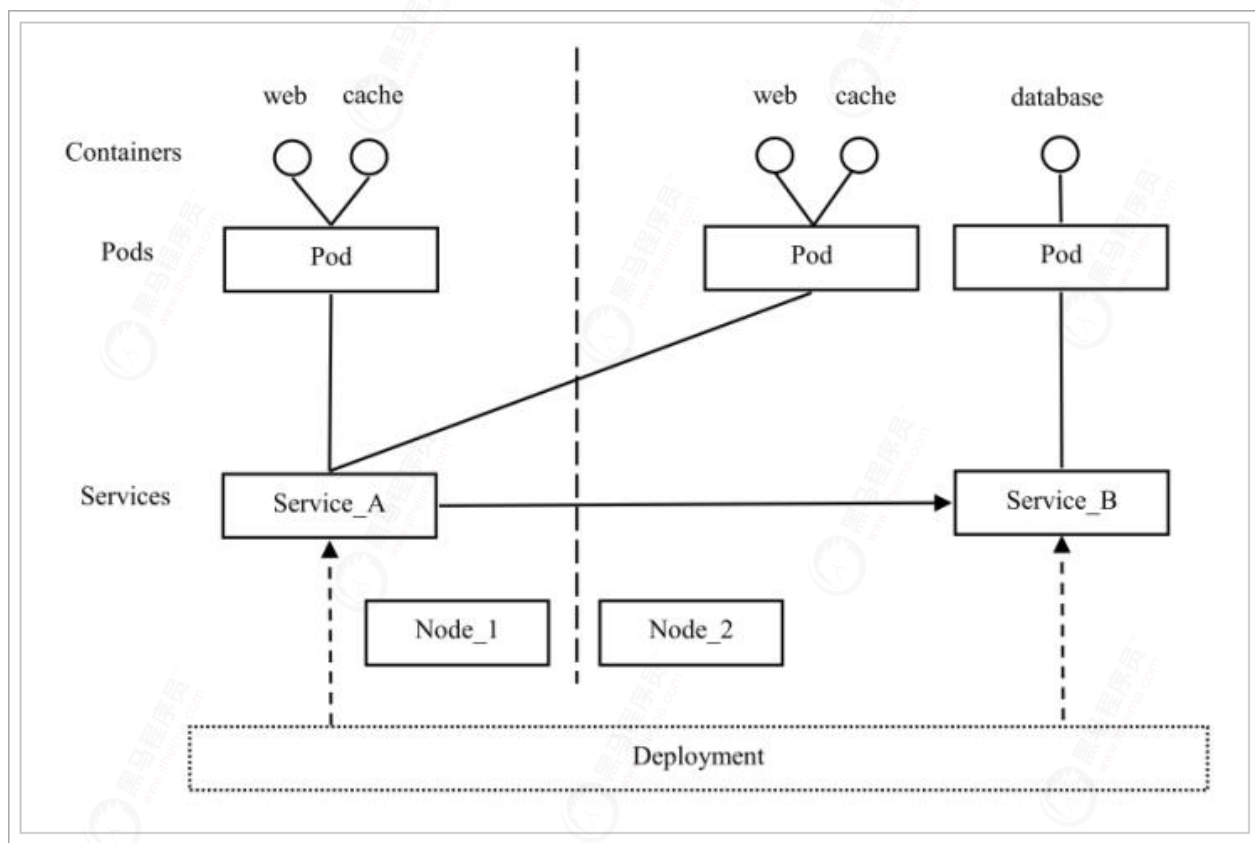
```
prefix: /api/v1/products
route:
- destination:
    host: productpage
    port:
        number: 9080
```

根据上面的配置，网关只要收到匹配路径的 HTTP 请求，就会将请求转发给 productpage 服务。

1.2、服务之间是如何通信的？

1.2.1、K8S四层网络架构

Istio运行于Kubernetes平台，Kubernetes 中的物理资源是以 Node 为单位来进行分配的——也是最大的资源分配单位——Node 中可以有很多 Pod，每个 Pod 中又可以有多个 Container，多个 Pod 以 Service 为标识聚合对外提供服务，而最下层的 Deployment 是对整体服务体系的抽象，它的实体便是 yaml 文件。Deployment、Service、Pod、Container 便构成 Kubernetes 的四层部署结构。



Kubernetes 的 Service 之所以会允许存在多个 Pod，是因为：

- 一是为了让应用可以以多版本的形式存在，比如 App 可以以 v1、v2 及 v3 的版本存在，对外暴露的都是同一服务名，如果不加以控制，那么流量将会平均地分配到以上三个版本中；
- 二是对应用进行更好的容错，比如可以将一个服务的 Pod 复制到多个 Node 上，这样当一台机器宕机的时候，应用服务仍然是可以正常工作的。

1.2.2、服务间的通信


```
[root@node1 ~]# kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
details-v1-6c9f8bcbcb-ph5cc        1/2     Running   4           2d17h
productpage-v1-7df7cb7f86-wvxst    1/2     Running   4           2d17h
ratings-v1-65cff55fb8-j6g47        1/2     Running   4           2d17h
reviews-v1-7bccdbbf96-krsl5        1/2     Running   4           2d17h
reviews-v2-7c9685df46-zpkgb        1/2     Running   4           2d17h
reviews-v3-58fc46b64-5cbq7         1/2     Running   4           2d17h
[root@node1 ~]# kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
details         ClusterIP   10.104.8.121    <none>        9080/TCP   2d17h
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP    2d22h
productpage     ClusterIP   10.107.142.19   <none>        9080/TCP   2d17h
ratings         ClusterIP   10.98.194.79    <none>        9080/TCP   2d17h
reviews         ClusterIP   10.105.9.217    <none>        9080/TCP   2d17h
```

可以看到，reviews 服务就被划分成了 v1、v2、v3 三个版本，但是查看服务的时候却只有一个reviews 服务。

每个服务都分配了一个cluster-ip，这个ip地址并不是网卡层面的地址，而是Kubernetes平台的分配的虚拟ip，所以对于该ip的调用，kubernetes就会将流量分给该ip下对应的pod。

该ip地址还对应一个内部域名，例如一个服务叫 **reviews**，它的命名空间是 **default**，在 **Kubernetes** 中其默认的域名为**svc.cluster.local**，所以只需要访问 **reviews.default.svc.cluster.local** 这个域名就可以解析到其对应的 **ClusterIP**，也可以通过服务名直接访问。

在reviews的源码中，通过http协议，调用ratings服务，如下：

```
//LibertyRestEndpoint.java
@Path("/")
public class LibertyRestEndpoint extends Application {
    .....

    private final static String services_domain =
System.getenv("SERVICES_DOMAIN") == null ? "" : ("." +
System.getenv("SERVICES_DOMAIN"));
    private final static String ratings_hostname =
System.getenv("RATINGS_HOSTNAME") == null ? "ratings" :
System.getenv("RATINGS_HOSTNAME");

    // 定义请求地址
    private final static String ratings_service = "http://" + ratings_hostname
+ services_domain + ":9080/ratings";

    .....

    private JsonObject getRatings(String productId, HttpHeaders requestHeaders) {
        ClientBuilder cb = ClientBuilder.newBuilder();
        Integer timeout = star_color.equals("black") ? 10000 : 2500;
        cb.property("com.ibm.ws.jaxrs.client.connection.timeout", timeout);
```

```

        cb.property("com.ibm.ws.jaxrs.client.receive.timeout", timeout);
        Client client = cb.build();
        //发起请求
        WebTarget ratingsTarget = client.target(ratings_service + "/" +
productId);
        Invocation.Builder builder =
ratingsTarget.request(MediaType.APPLICATION_JSON);
        for (String header : headers_to_propagate) {
            String value = requestHeaders.getHeaderString(header);
            if (value != null) {
                builder.header(header,value);
            }
        }
        try {
            Response r = builder.get();

            int statusCode = r.getStatusInfo().getStatusCode();
            if (statusCode == Response.Status.OK.getStatusCode()) {
                try (StringReader stringReader = new
StringReader(r.readEntity(String.class));
                    JsonReader jsonReader = Json.createReader(stringReader)) {
                    return jsonReader.readObject();
                }
            } else {
                System.out.println("Error: unable to contact " + ratings_service + "
got status of " + statusCode);
                return null;
            }
        } catch (ProcessingException e) {
            System.err.println("Error: unable to contact " + ratings_service + "
got exception " + e);
            return null;
        }
    }
}

```

在容器中进行验证，由于reviews容器中没有curl、wget等基础命令，所以需要借助于alpine容器进行验证。

#拉取镜像

```
docker pull alpine:3.8
```

```
[root@node2 ~]# docker run -it --name=alpine alpine:3.8
```

```
/ # apk update
```

```
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/main/x86_64/APKINDEX.tar.gz
```

```

fetch http://dl-
cdn.alpinelinux.org/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
v3.8.5-53-g001e8c2217 [http://dl-cdn.alpinelinux.org/alpine/v3.8/main]
v3.8.5-37-gf06ffe835a [http://dl-cdn.alpinelinux.org/alpine/v3.8/community]
OK: 9563 distinct packages available
/ # apk add curl
(1/5) Installing ca-certificates (20191127-r2)
(2/5) Installing nghttp2-libs (1.39.2-r0)
(3/5) Installing libssh2 (1.9.0-r1)
(4/5) Installing libcurl (7.61.1-r3)
(5/5) Installing curl (7.61.1-r3)
Executing busybox-1.28.4-r3.trigger
Executing ca-certificates-20191127-r2.trigger
OK: 6 MiB in 18 packages
/ #

#将该容器打成新版本, 方便后面使用
docker commit alpine alpine:my-3.8

#通过--ipc --net --pid 三个参数实现与目标容器共享命名空间, 也就共享了容器资源, 就相当于进入了目标容器进行操作
#container:7583cba4a6d2, 需要通过docker ps | grep reviews, 进行查找, 找到k8s_reviews_reviews-v3-xxxx的容器id

[root@node2 ~]# docker run -it --rm --net=container:7583cba4a6d2 --pid=container:7583cba4a6d2 --ipc=container:7583cba4a6d2 alpine:my-3.8
/ # ping ratings    #尝试ping ratings域名, 发现所映射的ip地址与 kubectl get services 查询到的ip地址相同, 说明了可以通过短域名访问目标
PING ratings (10.98.194.79): 56 data bytes
^C
--- ratings ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
/ # ping ratings.default.svc.cluster.local    #通过长域名, 实现相同的效果
PING ratings.default.svc.cluster.local (10.98.194.79): 56 data bytes
^C
--- ratings.default.svc.cluster.local ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
/ #
/ # curl http://ratings:9080/ratings/123    #通过curl访问地址, 可以得到正确的结果, 说明服务是能够访问通的
{"id":123,"ratings":{"Reviewer1":5,"Reviewer2":4}}/ #
/ #

```

2、Istio进阶

2.1、Sidecar流量接管原理

Sidecar 的部署支持自动注入与手动配置两种方式，若想使用自动注入的方式，目前则是需要 Kubernetes 配合的，只需要将应用对应的 Namespace 设置一个 label 即可。例如 Namespace 如果是 default，则可以标记如下：

```
kubectl label namespace default istio-injection=enabled
```

如果选择手动注入，需要这样操作：

```
kubectl apply -f <(istioctl kube-inject -f
samples/bookinfo/platform/kube/bookinfo.yaml)
```

在 Istio 架构中，Sidecar 是与应用一一对应的，那么它到底是存在于什么地方呢？

```
[root@node2 ~]# docker ps |grep reviews-v3
793dfb428eaf        istio/proxyv2                "/usr/local/bin/pilo..." 11 hours ago        Up 11 hours
k8s_istio-proxy_reviews-v3-58fc46b64-5cbq7_default_16752b99-0d35-4e4b-8c3e-5bade4f32a5c_3
7583cba4a6d2        02c5c151ee4e                "/opt/ibm/helpers/ru..." 11 hours ago        Up 11 hours
k8s_reviews_reviews-v3-58fc46b64-5cbq7_default_16752b99-0d35-4e4b-8c3e-5bade4f32a5c_3
a153639b96bc        registry.cn-hangzhou.aliyuncs.com/itcast/pause:3.2    "/pause"                  11 hours ago        Up 11 hours
k8s_POD_reviews-v3-58fc46b64-5cbq7_default_16752b99-0d35-4e4b-8c3e-5bade4f32a5c_10
[root@node2 ~]#
```

可以看到，与 reviews-v3 相关的容器有三个，分别是：

- k8s_istio-proxy_reviews-v3-xxx
- k8s_reviews_reviews-v3-xxx
- k8s_POD_reviews-v3-xxx

其中，第二个是应用本身，第三个是 pause 容器，第一个就是 Sidecar 了，每个 pod 都有这样三个一组的容器。

pause 容器的作用：

在 Linux 系统启动时，任何进程都是由第一个进程 init 派生出来的，因此在正常情况下，它们都拥有同样的命名空间。当一个进程结束或者异常退出的时候，它的父进程负责处理它的返回结果，以让操作系统正常回收其进程空间；不过有些时候，一些父进程却没有很好地处理子进程的退出情况，就是说没有调用 wait 命令来处理其返回代码，其原因可能是代码写得不好或者意外崩溃了。

这个时候，操作系统就会将失去父进程的进程挂到 PID 为 1 的进程下，一般场景下就是 init 进程，而这种进程也被形象地称为孤儿进程（Orphan Process）。不过在 Docker 容器中，容器独立命名空间中的每个应用进程自身就是第一个进程（即 PID=1），如果这个时候，应用进程——例如 Nginx——的子进程又调用 fork 或者 exec 参数创建了更多的子进程，那么当出现问题时，它们就会被挂到 Nginx 下面。这样的问题在于，虽然从层级关系上，进程间的关系是对了，但 Nginx 本身跟 Init 并不一样，它没有处理僵尸进程的逻辑，所以即便挂在了它下面也没用。

pause 容器没有实质性的业务逻辑，仅仅处理子进程返回时的各种信号，以让其顺利退出。

k8s_istio-proxy 容器是如何劫持流量的？通过 alpine 容器分别进入应用容器和 sidecar 容器进行查询：

```
[root@node3 ~]# docker ps | grep reviews-v3
```

```

53eaaf7250a7          istio/proxyv2
"/usr/local/bin/pilo..." 40 minutes ago      Up 40 minutes
k8s_istio-proxy_reviews-v3-58fc46b64-16244_default_92bd4472-340e-4c45-
87d9-7edblaabf025_0
eeafad8f956b          istio/examples-bookinfo-reviews-v3
"/opt/ibm/helpers/ru..." 40 minutes ago      Up 40 minutes
k8s_reviews_reviews-v3-58fc46b64-16244_default_92bd4472-340e-4c45-87d9-
7edblaabf025_0
0eb274c2f015          registry.cn-hangzhou.aliyuncs.com/itcast/pause:3.2
"/pause"              41 minutes ago      Up 41 minutes
k8s_POD_reviews-v3-58fc46b64-16244_default_92bd4472-340e-4c45-87d9-
7edblaabf025_0
[root@node3 ~]#

```

#sidecar容器

```

[root@node3 ~]# docker run -it --rm --net=container:53eaaf7250a7 --
pid=container:53eaaf7250a7 --ipc=container:53eaaf7250a7 alpine:my-3.8
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 06:E2:F2:9E:B9:72
          inet addr:10.244.2.34  Bcast:10.244.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:9425 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6544 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1216915 (1.1 MiB)  TX bytes:5622857 (5.3 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:8158 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8158 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6129436 (5.8 MiB)  TX bytes:6129436 (5.8 MiB)

```

```

/ # ^C
/ # [root@node3 ~]#

```

#应用容器

```

[root@node3 ~]# docker run -it --rm --net=container:eeafad8f956b --
pid=container:eeafad8f956b --ipc=container:eeafad8f956b alpine:my-3.8
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 06:E2:F2:9E:B9:72
          inet addr:10.244.2.34  Bcast:10.244.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:9550 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6631 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1229027 (1.1 MiB)  TX bytes:5687028 (5.4 MiB)

```

```

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:8266 errors:0 dropped:0 overruns:0 frame:0
        TX packets:8266 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:6204098 (5.9 MiB)  TX bytes:6204098 (5.9 MiB)

/ # [root@node3 ~]#

```

可以看到，sidecar容器和应用容器的ip地址是完全一致的，也就是说明他们俩共享的ip地址。那么，共享了ip地址后，sidecar又是如何劫持流量的呢？为了弄明白这个问题，我们需要看一下，Istio在注入sidecar时到底做了哪些事情。

```

# istioctl kube-inject -f samples/bookinfo/platform/kube/bookinfo.yaml >
/tmp/bookinfo-kube-inject.yaml
# 下面截取关键部分，完整文件请查看资料中的 bookinfo-kube-inject.yaml 文件

```

```

spec:
  containers:
  - image: docker.io/istio/examples-bookinfo-productpage-v1.15.1
    imagePullPolicy: IfNotPresent
    name: productpage
    ports:
    - containerPort: 9080  #服务对外暴露的端口
    resources: {}
    volumeMounts:
    - mountPath: /tmp
      name: tmp
    .....略.....
    - name: ISTIO_KUBE_APP_PROBERS
      value: '{}'
    image: docker.io/istio/proxyv2:1.6.5  #sidecar 代理
    imagePullPolicy: Always
    name: istio-proxy
    ports:
    - containerPort: 15090
      name: http-envoy-prom
      protocol: TCP
    readinessProbe:
      failureThreshold: 30
      httpGet:
        path: /healthz/ready
        port: 15021
      initialDelaySeconds: 1
      periodSeconds: 2
    resources:
      limits:
        cpu: "2"

```

```
    memory: 1Gi
  requests:
    cpu: 10m
    memory: 40Mi
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
    privileged: false
    readOnlyRootFilesystem: true
    runAsGroup: 1337
    runAsNonRoot: true
    runAsUser: 1337
  volumeMounts:
    - mountPath: /var/run/secrets/istio
      name: istiod-ca-cert
    - mountPath: /var/lib/istio/data
      name: istio-data
    - mountPath: /etc/istio/proxy
      name: istio-envoy
    - mountPath: /etc/istio/pod
      name: istio-podinfo
  initContainers:
    - args:
        - istio-iptables
        - -p
        - "15001"
        - -z
        - "15006"
        - -u
        - "1337"
        - -m
        - REDIRECT
        - -i
        - '*'
        - -x
        - ""
        - -b
        - '*'
        - -d
        - 15090,15021,15020
      env:
        - name: DNS_AGENT
      image: docker.io/istio/proxyv2:1.6.5 #init 容器
      imagePullPolicy: Always
      name: istio-init
      resources:
        limits:
```

```
    cpu: 100m
    memory: 50Mi
  requests:
    cpu: 10m
    memory: 10Mi
  .....略.....
```

Istio 给应用 Pod 注入的配置主要包括：

- Init 容器 istio-init：用于 pod 中设置 iptables 端口转发
- Sidecar 容器 istio-proxy：运行 sidecar 代理，如 Envoy 或 MOSN

2.1.1、Init 容器解析

Istio 在 pod 中注入的 Init 容器名为 `istio-init`，我们在上面 Istio 注入完成后的 YAML 文件中看到了该容器的启动命令是：

```
istio-iptables -p 15001 -z 15006 -u 1337 -m REDIRECT -i '*' -x "" -b '*' -d
15090,15021,15020
```

Init 容器的启动入口是 `istio-iptables` 命令行，该命令行工具的用法如下：

```
$ istio-iptables [flags]
  -p: 指定重定向所有 TCP 流量的 sidecar 端口（默认为 $ENVOY_PORT = 15001）
  -m: 指定入站连接重定向到 sidecar 的模式，“REDIRECT”或“TPROXY”（默认为
$ISTIO_INBOUND_INTERCEPTION_MODE）
  -b: 逗号分隔的入站端口列表，其流量将重定向到 Envoy（可选）。使用通配符“*”表示重定向所有
端口。为空时表示禁用所有入站重定向（默认为 $ISTIO_INBOUND_PORTS）
  -d: 指定要从重定向到 sidecar 中排除的入站端口列表（可选），以逗号格式分隔。使用通配符“*”
表示重定向所有入站流量（默认为 $ISTIO_LOCAL_EXCLUDE_PORTS）
  -o: 逗号分隔的出站端口列表，不包括重定向到 Envoy 的端口。
  -i: 指定重定向到 sidecar 的 IP 地址范围（可选），以逗号分隔的 CIDR 格式列表。使用通配符
“*”表示重定向所有出站流量。空列表将禁用所有出站重定向（默认为 $ISTIO_SERVICE_CIDR）
  -x: 指定将从重定向中排除的 IP 地址范围，以逗号分隔的 CIDR 格式列表。使用通配符“*”表示
重定向所有出站流量（默认为 $ISTIO_SERVICE_EXCLUDE_CIDR）。
  -k: 逗号分隔的虚拟接口列表，其入站流量（来自虚拟机的）将被视为出站流量。
  -g: 指定不应用重定向的用户的 GID。（默认值与 -u param 相同）
  -u: 指定不应用重定向的用户的 UID。通常情况下，这是代理容器的 UID（默认值是 1337，即
istio-proxy 的 UID）。
  -z: 所有进入 pod/VM 的 TCP 流量应被重定向到的端口（默认 $INBOUND_CAPTURE_PORT =
15006）。
```

命令解析：

这条启动命令的作用是：

- 将应用容器的所有流量都转发到 sidecar 的 15006 端口。
- 使用 `istio-proxy` 用户身份运行，UID 为 1337，即 sidecar 所处的用户空间，这也是 `istio-proxy` 容器默认使用的用户，见 YAML 配置中的 `runAsUser` 字段。

- 使用默认的 REDIRECT 模式来重定向流量。
- 将所有出站流量都重定向到 sidecar 代理（通过 15001 端口）。

因为 Init 容器初始化完毕后就会自动终止，因为我们无法登陆到容器中查看 iptables 信息，但是 Init 容器初始化结果会保留到应用容器和 sidecar 容器中。

2.1.2、iptables 注入解析

查看与 productpage 有关的 iptables 规则如下：

```
[root@node3 ~]# docker top `docker ps|grep "istio-proxy_productpage"|cut -d " " -f1`
UID                PID                PPID              C
STIME              TTY                TIME              CMD
1337               5941               5924              0
02:30              ?                  00:00:11
/usr/local/bin/pilot-agent proxy sidecar --domain default.svc.cluster.local --
serviceCluster productpage.default --proxyLogLevel=warning --
proxyComponentLogLevel=misc:error --trust-domain=cluster.local --concurrency 2
1337               5968               5941              0
02:30              ?                  00:01:15
/usr/local/bin/envoy -c etc/istio/proxy/envoy-rev0.json --restart-epoch 0 --
drain-time-s 45 --parent-shutdown-time-s 60 --service-cluster
productpage.default --service-node sidecar~10.244.2.32~productpage-v1-
7df7cb7f86-wvxst.default~default.svc.cluster.local --max-obj-name-len 189 --
local-address-ip-version v4 --log-format %Y-%m-%dT%T.%fZ?%l?envoy %n?%v -l
warning --component-log-level misc:error --concurrency 2

# 进入 nsenter 进入 sidecar 容器的命名空间（以上任何一个都可以）
nsenter -n --target 5941

# 查看 NAT 表中规则配置的详细信息。
[root@node3 ~]# iptables -t nat -L -v
# PREROUTING 链：用于目标地址转换（DNAT），将所有入站 TCP 流量跳转到 ISTIO_INBOUND 链
上。
Chain PREROUTING (policy ACCEPT 14987 packets, 899K bytes)
  pkts bytes target    prot opt in     out     source    destination
14987 899K ISTIO_INBOUND tcp -- any    any     anywhere  anywhere

# INPUT 链：处理输入数据包，非 TCP 流量将继续 OUTPUT 链。
Chain INPUT (policy ACCEPT 14987 packets, 899K bytes)
  pkts bytes target    prot opt in     out     source    destination

# OUTPUT 链：将所有出站数据包跳转到 ISTIO_OUTPUT 链上。
Chain OUTPUT (policy ACCEPT 3795 packets, 332K bytes)
  pkts bytes target    prot opt in     out     source    destination
431 25860 ISTIO_OUTPUT tcp -- any    any     anywhere  anywhere
```

POSTROUTING 链：所有数据包流出网卡时都要先进入POSTROUTING 链，内核根据数据包目的地判断是否需要转发出去，我们看到此处未做任何处理。

```
Chain POSTROUTING (policy ACCEPT 3795 packets, 332K bytes)
  pkts bytes target      prot opt in      out     source      destination
```

ISTIO_INBOUND 链：将所有入站流量重定向到 ISTIO_IN_REDIRECT 链上，目的地为 15090 (mixer 使用) 和 15020 (Ingress gateway 使用, 用于 Pilot 健康检查) 端口的流量除外，发送到以上两个端口的流量将返回 iptables 规则链的调用点，即 PREROUTING 链的后继 POSTROUTING。

Istio 1.6版本中，检测端口改为了 15021

```
Chain ISTIO_INBOUND (1 references)
  pkts bytes target      prot opt in      out     source      destination
    0      0 RETURN      tcp  --  any     any     anywhere    anywhere
      tcp dpt:ssh
    5    300 RETURN      tcp  --  any     any     anywhere    anywhere
      tcp dpt:15090
14982  899K RETURN      tcp  --  any     any     anywhere    anywhere
      tcp dpt:15021
    0      0 RETURN      tcp  --  any     any     anywhere    anywhere
      tcp dpt:15020
    0      0 ISTIO_IN_REDIRECT tcp  --  any     any     anywhere    anywhere
anywhere
```

ISTIO_IN_REDIRECT 链：将所有的入站流量跳转到本地的 15006 端口，至此成功的拦截了流量到 sidecar 中。

```
Chain ISTIO_IN_REDIRECT (3 references)
  pkts bytes target      prot opt in      out     source      destination
    0      0 REDIRECT     tcp  --  any     any     anywhere    anywhere
      redir ports 15006
```

ISTIO_OUTPUT 链：选择需要重定向到 Envoy (即本地) 的出站流量，所有非 localhost 的流量全部转发到 ISTIO_REDIRECT。为了避免流量在该 Pod 中无限循环，所有到 istio-proxy 用户空间的流量都返回到它的调用点中的下一条规则，本例中即 OUTPUT 链，因为跳出 ISTIO_OUTPUT 规则之后就进入下一条链 POSTROUTING。如果目的地非 localhost 就跳转到 ISTIO_REDIRECT；如果流量是来自 istio-proxy 用户空间的，那么就跳出该链，返回它的调用链继续执行下一条规则 (OUTPUT 的下一条规则，无需对流量进行处理)；所有的非 istio-proxy 用户空间的目的地是 localhost 的流量就跳转到 ISTIO_REDIRECT。

```
Chain ISTIO_OUTPUT (1 references)
  pkts bytes target      prot opt in      out     source      destination
    0      0 RETURN      all  --  any     lo      127.0.0.6    anywhere
    0      0 ISTIO_IN_REDIRECT all  --  any     lo      anywhere
!localhost                                owner UID match 1337
    0      0 RETURN      all  --  any     lo      anywhere    anywhere
      ! owner UID match 1337
431  25860 RETURN      all  --  any     any     anywhere    anywhere
      owner UID match 1337
    0      0 ISTIO_IN_REDIRECT all  --  any     lo      anywhere
!localhost                                owner GID match 1337
```

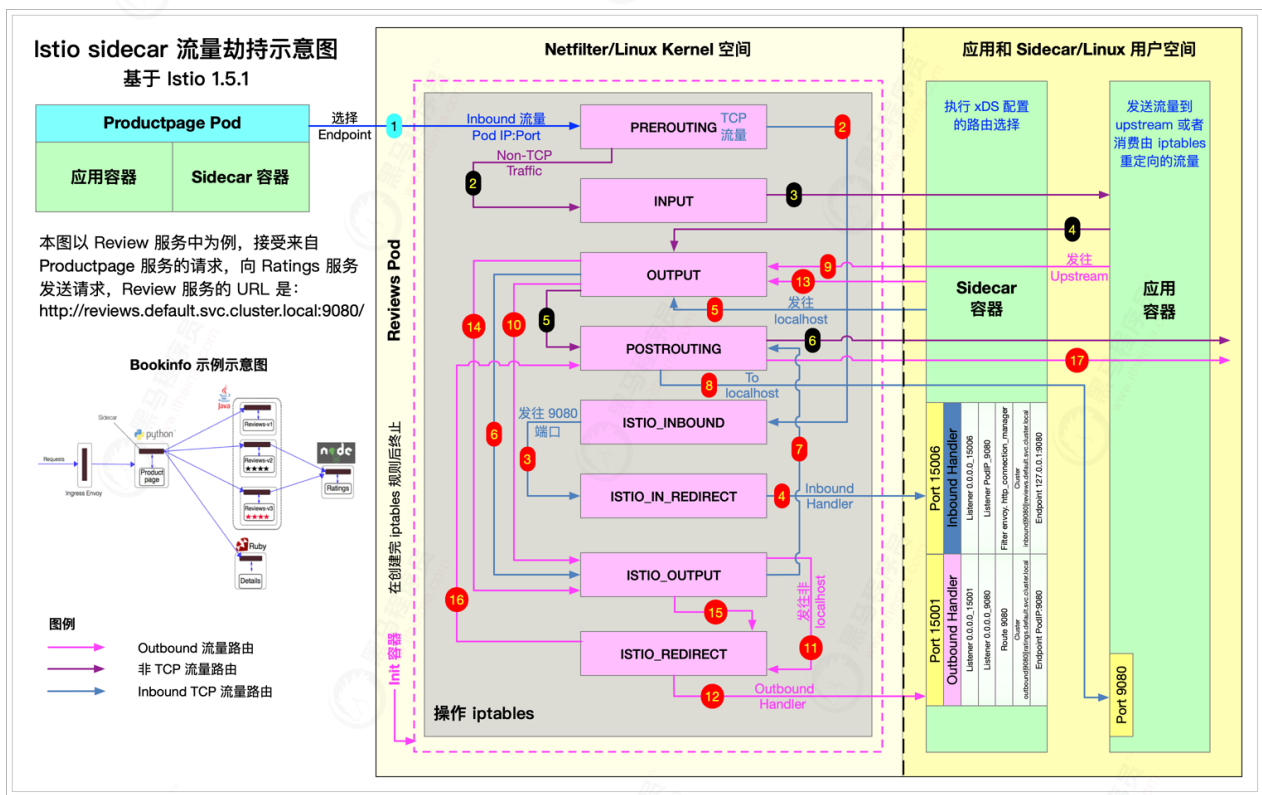
```

0    0 RETURN      all -- any    lo      anywhere      anywhere
      ! owner GID match 1337
0    0 RETURN      all -- any    any      anywhere      anywhere
      owner GID match 1337
0    0 RETURN      all -- any    any      anywhere      localhost
0    0 ISTIO_REDIRECT all -- any    any      anywhere      anywhere
anywhere

# ISTIO_REDIRECT 链: 将所有流量重定向到 Sidecar (即本地) 的 15001 端口。
Chain ISTIO_REDIRECT (1 references)
  pkts bytes target      prot opt in      out      source      destination
    0    0 REDIRECT      tcp  --  any      any      anywhere     anywhere
      redir ports 15001

```

流程示例:



2.2、超时与重试

2.2.1、超时

超时是 Envoy 代理等待来自给定服务的答复的时间量, 以确保服务不会因为等待答复而无限期的挂起, 并在可预测的时间范围内调用成功或失败。HTTP 请求的默认超时时间是 15 秒, 这意味着如果服务在 15 秒内没有响应, 调用将失败。

对于某些应用程序和服务, Istio 的缺省超时可能不合适。例如, 超时太长可能会由于等待失败服务的回复而导致过度的延迟; 而超时过短则可能在等待涉及多个服务返回的操作时触发不必要地失败。

为了找到并使用最佳超时设置, Istio 允许您使用虚拟服务按服务轻松地动态调整超时, 而不必修改您的业务代码。

下面的示例是一个虚拟服务，它对 ratings 服务的 v1 子集的调用指定 10 秒超时：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    timeout: 10s
```

2.2.2、重试

重试设置指定如果初始调用失败，Envoy 代理尝试连接服务的最大次数。通过确保调用不会因为临时过载的服务或网络等问题而永久失败，重试可以提高服务可用性和应用程序的性能。重试之间的间隔（25ms+）是可变的，并由 Istio 自动确定，从而防止被调用服务被请求淹没。默认情况下，在第一次失败后，Envoy 代理不会重新尝试连接服务。

与超时一样，Istio 默认的重试行为在延迟方面可能不适合您的应用程序需求（对失败的服务进行过多的重试会降低速度）或可用性。

您可以在虚拟服务中按服务调整重试设置，而不必修改业务代码。您还可以通过添加每次重试的超时来进一步细化重试行为，并指定每次重试都试图成功连接到服务所等待的时间量。

下面的示例配置了在初始调用失败后最多重试 3 次来连接到服务子集，每个重试都有 2 秒的超时。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    retries:
      attempts: 3
      perTryTimeout: 2s
```

2.3、熔断器

熔断器是 Istio 为创建具有弹性的微服务应用提供的有用的机制。在熔断器中，设置一个对服务中的单个主机调用的限制，例如并发连接的数量或对该主机调用失败的次数。一旦限制被触发，熔断器就会“跳闸”并停止连接到该主机。

使用熔断模式可以快速失败而不必让客户端尝试连接到过载或有故障的主机。

2.3.1、部署httpbin

httpbin是一个开源项目，使用Python+Flask编写，利用它可以测试各种HTTP请求和响应。官网：<http://httpbin.org/>

```
kubectl apply -f samples/httpbin/httpbin.yaml
```

2.3.2、配置熔断器

创建一个目标规则，在调用 httpbin 服务时应用熔断设置：

```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1 #最大连接数
      http:
        http1MaxPendingRequests: 1 #http请求pending状态的最大请求数
        maxRequestsPerConnection: 1 #在一定时间内限制对后端服务发起的最大请求数
    outlierDetection: #熔断设置
      consecutiveErrors: 1 #从连接池开始拒绝连接，已经连接失败的次数，当通过HTTP访问
      interval: 1s #拒绝访问扫描的时间间隔，即在interval（1s）内连续发生1个
      consecutiveErrors错误，则触发服务熔断，格式是1h/1m/1s/1ms，但必须大于等于1ms。即分析是否
      需要剔除的频率，多久分析一次，默认10秒。
      baseEjectionTime: 3m #最短拒绝访问时长。这个时间主机将保持拒绝访问，且如果决绝访问
      达到一定的次数。格式:1h/1m/1s/1ms，但必须大于等于1ms。实例被剔除后，至少多久不得返回负载均
      衡池，默认是30秒。
      maxEjectionPercent: 100 #服务在负载均衡池中被拒绝访问（被移除）的最大百分比，负载
      均衡池中最多有多大比例被剔除，默认是10%。
EOF
```

验证目标规则是否已正确创建：

```
kubectl get destinationrule httpbin -o yaml

apiVersion: networking.istio.io/v1alpha3
```



```

kind: DestinationRule
metadata:
  name: httpbin
  ...
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
      tcp:
        maxConnections: 1
    outlierDetection:
      baseEjectionTime: 180.000s
      consecutiveErrors: 1
      interval: 1.000s
      maxEjectionPercent: 100

```

2.3.3、客户端

创建客户端程序以发送流量到 httpbin 服务。这是一个名为 Fortio 的负载测试客户的，其可以控制连接数、并发数及发送 HTTP 请求的延迟。通过 Fortio 能够有效的触发前面在 DestinationRule 中设置的熔断策略。

1. 向客户端注入 Istio Sidecar 代理，以便 Istio 对其网络交互进行管理：

```
$ kubectl apply -f <(istioctl kube-inject -f samples/httpbin/sample-client/fortio-deploy.yaml)
```

2. 登入客户端 Pod 并使用 Fortio 工具调用 httpbin 服务。-curl 参数表明发送一次调用：

```

$ FORTIO_POD=$(kubectl get pod | grep fortio | awk '{ print $1 }')
$ kubectl exec -it $FORTIO_POD -c fortio -- /usr/bin/fortio load -curl
http://httpbin:8000/get
HTTP/1.1 200 OK
server: envoy
date: Tue, 16 Jan 2018 23:47:00 GMT
content-type: application/json
access-control-allow-origin: *
access-control-allow-credentials: true
content-length: 445
x-envoy-upstream-service-time: 36

{
  "args": {},
  "headers": {
    "Content-Length": "0",
    "Host": "httpbin:8000",

```

```

    "User-Agent": "istio/fortio-0.6.2",
    "X-B3-Sampled": "1",
    "X-B3-Spanid": "824fbd828d809bf4",
    "X-B3-Traceid": "824fbd828d809bf4",
    "X-Ot-Span-Context":
    "824fbd828d809bf4;824fbd828d809bf4;0000000000000000",
    "X-Request-Id": "1ad2de20-806e-9622-949a-bd1d9735a3f4"
  },
  "origin": "127.0.0.1",
  "url": "http://httpbin:8000/get"
}

```

可以看到调用后端服务的请求已经成功！接下来，可以测试熔断。

2.3.4、触发熔断器

在 DestinationRule 配置中，定义了 maxConnections: 1 和 http1MaxPendingRequests: 1。这些规则意味着，如果并发的连接和请求数超过一个，在 istio-proxy 进行进一步的请求和连接时，后续请求或连接将被阻止。

1. 发送并发数为 2 的连接（-c 2），请求 20 次（-n 20）：

```

[root@node1 istio-1.6.5]# kubectl exec -it $FORTIO_POD -c fortio --
/usr/bin/fortio load -c 2 -qps 0 -n 20 -loglevel Warning
http://httpbin:8000/get
03:59:25 I logger.go:97> Log level is now 3 Warning (was 2 Info)
Fortio 1.3.1 running at 0 queries per second, 2->2 procs, for 20 calls:
http://httpbin:8000/get
Starting at max qps with 2 thread(s) [gomax 2] for exactly 20 calls (10
per thread + 0)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
03:59:25 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
Ended after 79.166124ms : 20 calls. qps=252.63
Aggregated Function Time : count 20 avg 0.0064311497 +/- 0.007472 min
0.000340298 max 0.032824602 sum 0.128622994
# range, mid point, percentile, count
>= 0.000340298 <= 0.001 , 0.000670149 , 10.00, 2
> 0.001 <= 0.002 , 0.0015 , 20.00, 2
> 0.002 <= 0.003 , 0.0025 , 40.00, 4
> 0.003 <= 0.004 , 0.0035 , 60.00, 4
> 0.004 <= 0.005 , 0.0045 , 65.00, 1
> 0.006 <= 0.007 , 0.0065 , 80.00, 3
> 0.012 <= 0.014 , 0.013 , 85.00, 1
> 0.014 <= 0.016 , 0.015 , 90.00, 1
> 0.016 <= 0.018 , 0.017 , 95.00, 1

```



```

04:01:42 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
04:01:42 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
Ended after 32.153704ms : 30 calls. qps=933.02
Aggregated Function Time : count 30 avg 0.0019156712 +/- 0.001801 min
0.000270969 max 0.006581956 sum 0.057470135
# range, mid point, percentile, count
>= 0.000270969 <= 0.001 , 0.000635485 , 56.67, 17
> 0.002 <= 0.003 , 0.0025 , 70.00, 4
> 0.003 <= 0.004 , 0.0035 , 86.67, 5
> 0.004 <= 0.005 , 0.0045 , 93.33, 2
> 0.005 <= 0.006 , 0.0055 , 96.67, 1
> 0.006 <= 0.00658196 , 0.00629098 , 100.00, 1
# target 50% 0.000908871
# target 75% 0.0033
# target 90% 0.0045
# target 99% 0.00640737
# target 99.9% 0.0065645
Sockets used: 20 (for perfect keepalive, would be 3)
Code 200 : 11 (36.7 %)
Code 503 : 19 (63.3 %)
Response Header Sizes : count 30 avg 84.333333 +/- 110.8 min 0 max 230 sum
2530
Response Body/Total Sizes : count 30 avg 464.666667 +/- 294 min 241 max 851
sum 13940
All done 30 calls (plus 0 warmup) 1.916 ms avg, 933.0 qps

```

可以看到，只有 36.7% 的请求成功，其余的均被熔断器拦截：

```

Code 200 : 11 (36.7 %)
Code 503 : 19 (63.3 %)

```

3. 查询 `istio-proxy` 状态以了解更多熔断详情:

```

[root@node1 istio-1.6.5]# kubectl exec $FORTIO_POD -c istio-proxy --
pilot-agent request GET stats | grep httpbin | grep pending
cluster.outbound|8000||httpbin.default.svc.cluster.local.circuit_breakers.
default.rq_pending_open: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.circuit_breakers.
high.rq_pending_open: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_pendi
ng_active: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_pendi
ng_failure_eject: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_pendi
ng_overflow: 72
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_pendi
ng_total: 59

```

可以看到 `upstream_rq_pending_overflow` 值 59，这意味着，目前为止已有 59 个调用被标记为熔断。

2.3.5、清理

1. 清理规则:

```
$ kubectl delete destinationrule httpbin
```

2. 下线 httpbin 服务和客户端:

```
$ kubectl delete deploy httpbin fortio-deploy
$ kubectl delete svc httpbin
```

2.4、可视化网络

在Istio中可以使用Kiali进行可视化的管理服务网格。Kiali官网: <https://www.kiali.io/>

下面我们还是基于demo环境以及Bookinfo进行演示，在demo环境中已经默认安装了kiali，默认的用户名密码均为：admin。

2.4.1、启动服务

1. 要验证服务是否在您的群集中运行，请运行以下命令:

```
$ kubectl -n istio-system get svc kiali
```

2. 要确定 Bookinfo URL

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].status.hostIP}')
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

3. 将流量发送到网格，有三种选择:

- 在浏览器中访问 `http://$GATEWAY_URL/productpage`
- 多次使用以下命令:

```
$ curl http://$GATEWAY_URL/productpage
```

- 如果您在系统中安装了 `watch` 命令，请通过以下方式连续发送请求，时间间隔为1秒:

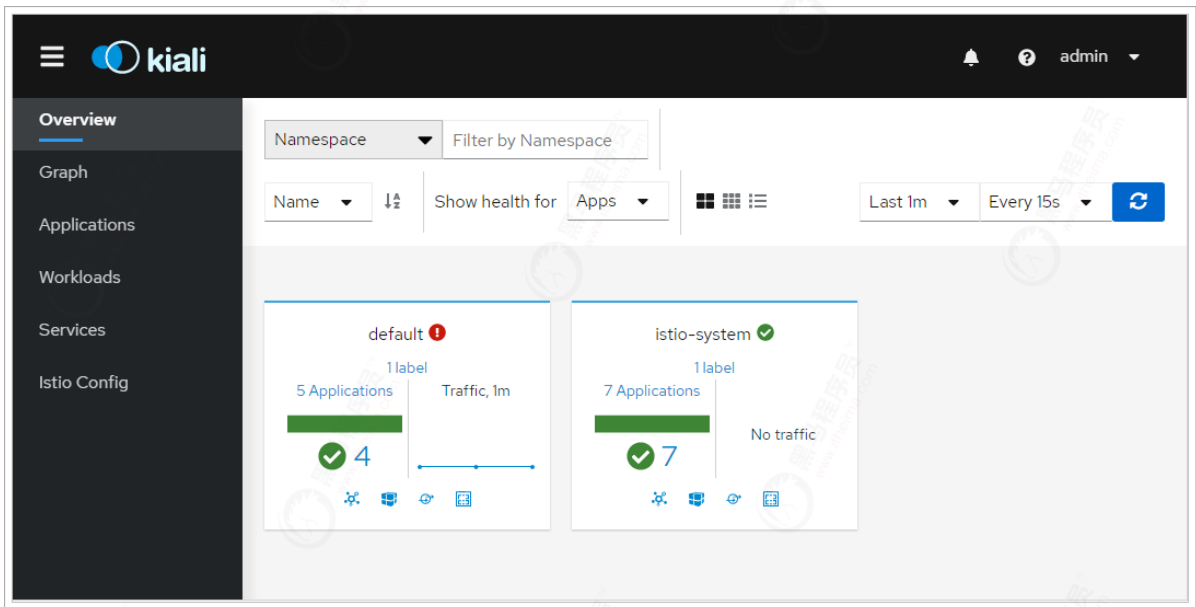
```
$ watch -n 1 curl -o /dev/null -s -w %{http_code}
$GATEWAY_URL/productpage
```


4. 要打开 Kiali UI, 请在您的 Kubernetes 环境中执行以下命令:

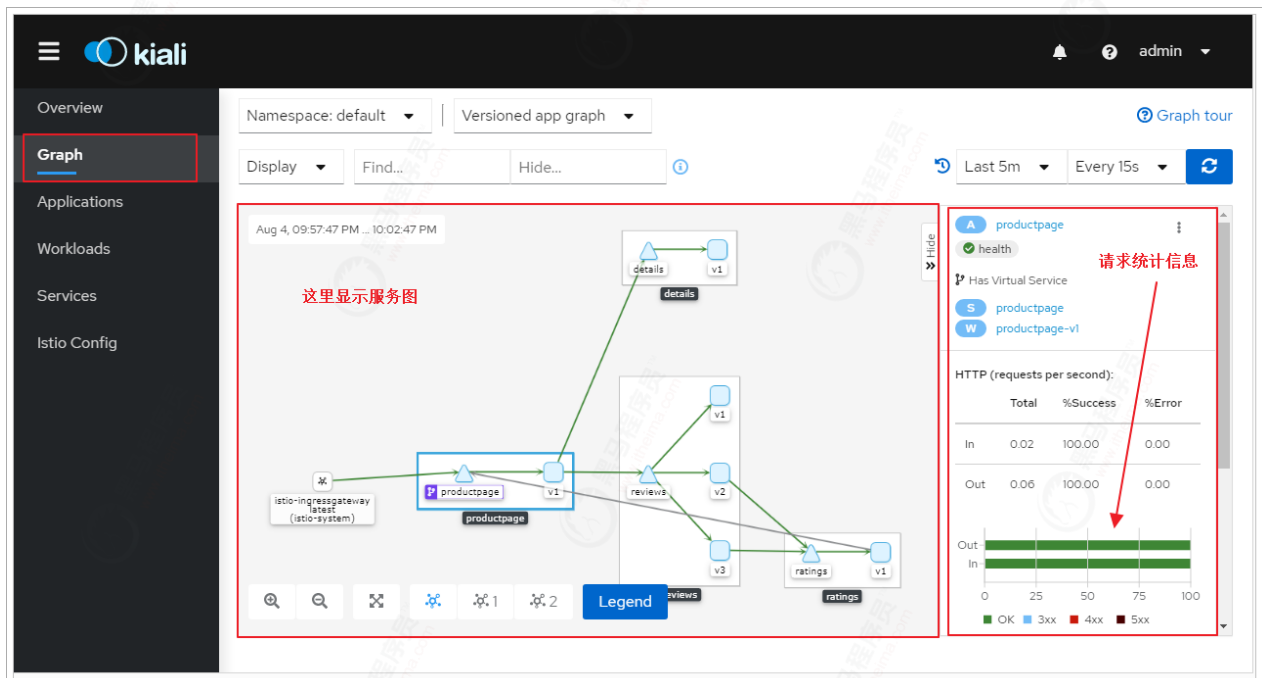
```
$ istioctl dashboard kiali --address 192.168.31.106
```

5. 要登录 Kiali UI, 请到 Kiali 登录界面, 然后输入默认的用户名密码。

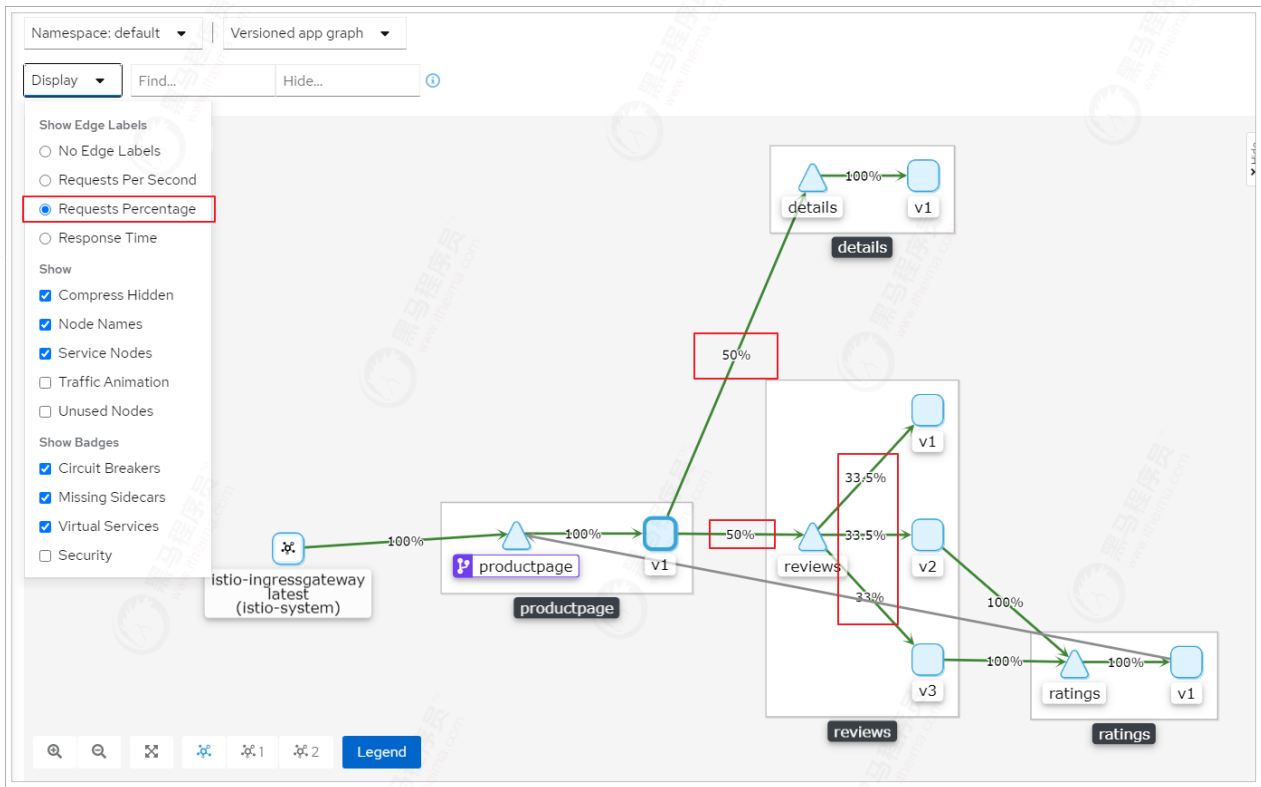
6. 登录后立即显示的 **Overview** 页面中查看网格的概述。**Overview** 页面显示了网格中具有服务的所有名称空间。以下屏幕截图显示了类似的页面:



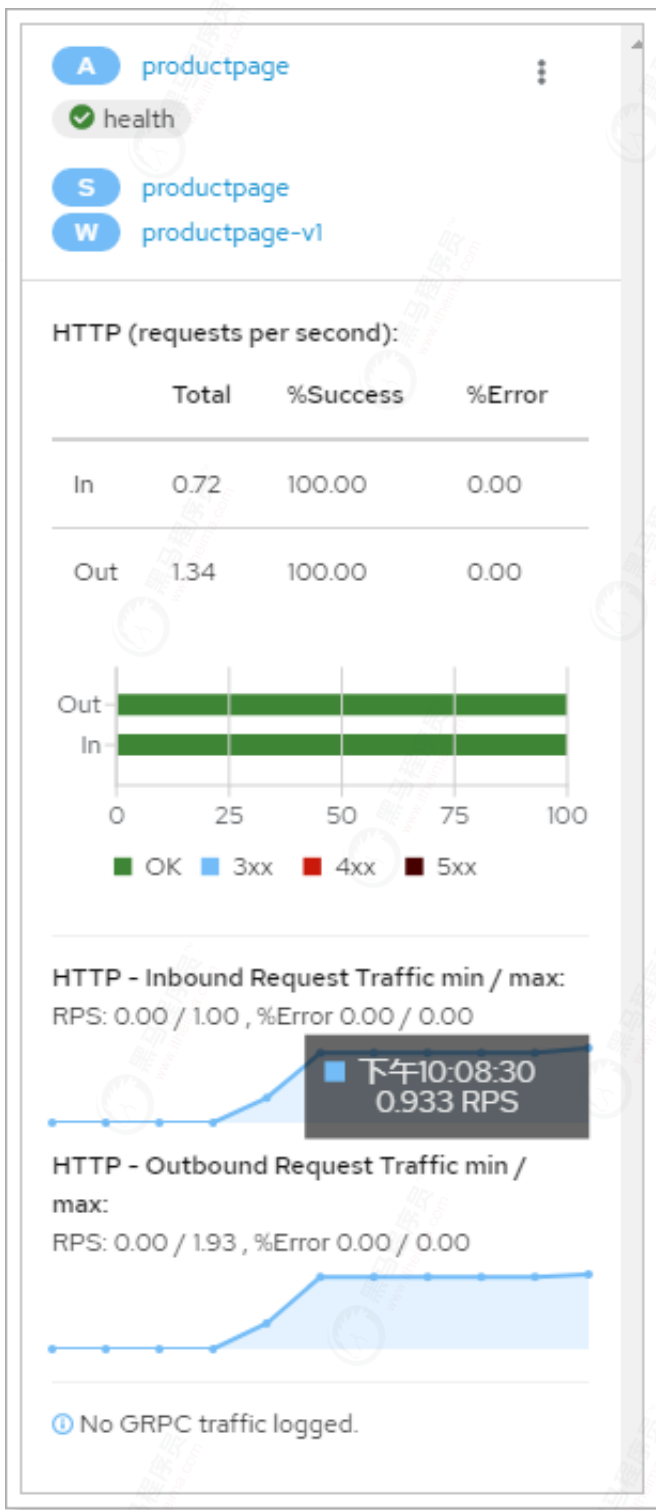
2.4.2、图



查看流量分配的百分比:

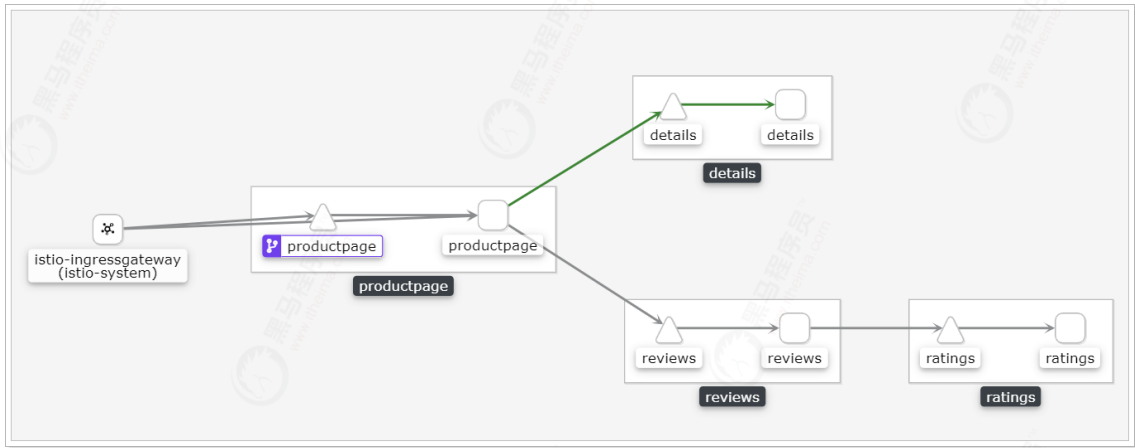


请求统计数据，RPS数据（最小V最大的比值）



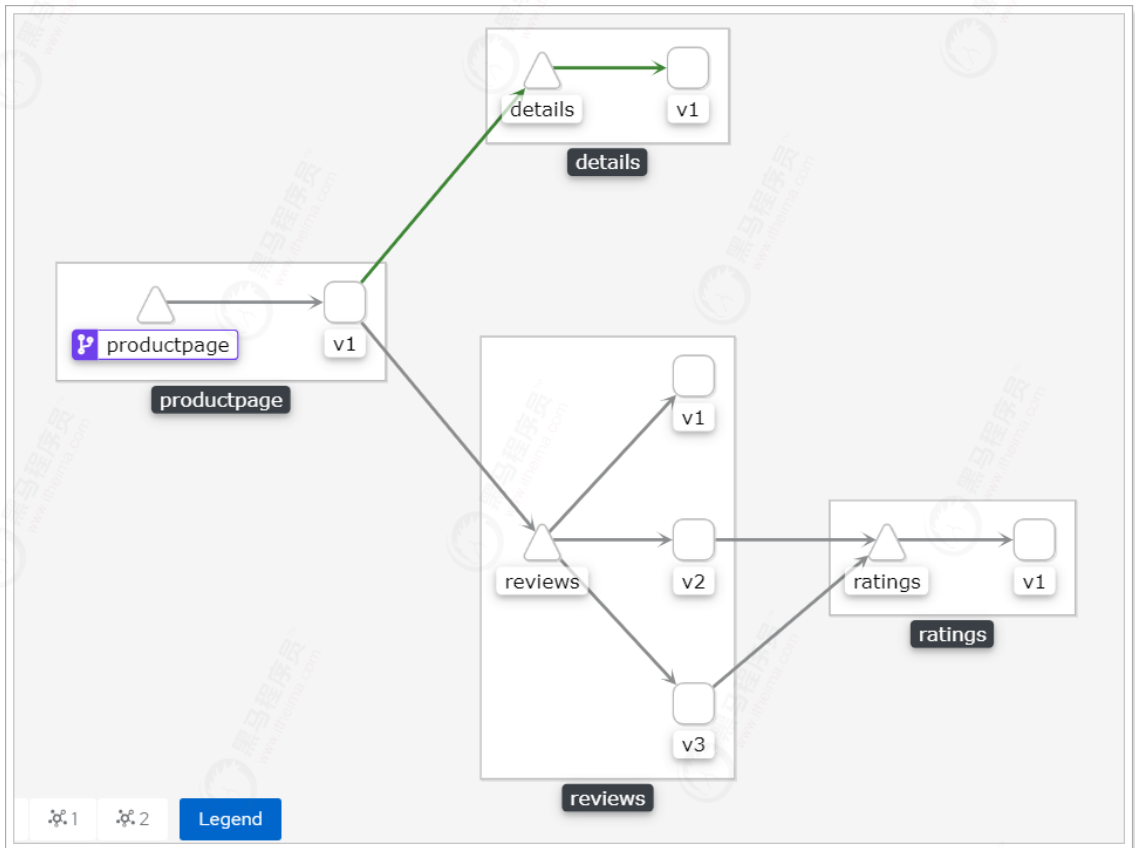
显示不同的图表类型，有四种类型：

- App
 - 图形类型将一个应用程序的所有版本聚合到一个图形节点中。



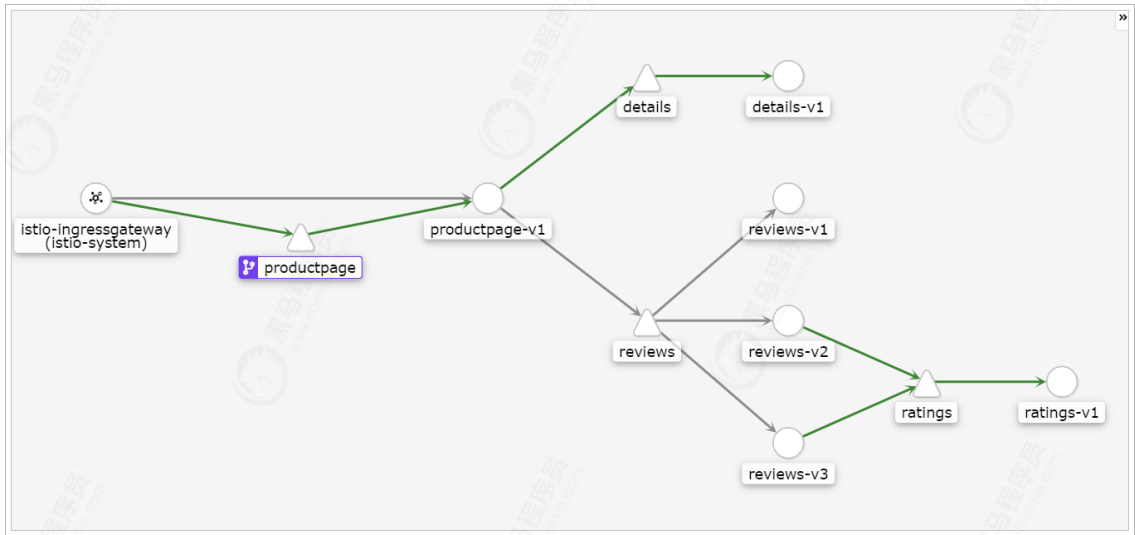
- Versioned App

- 图类型显示每个应用程序版本的节点，但是特定应用程序的所有版本都组合在一起。

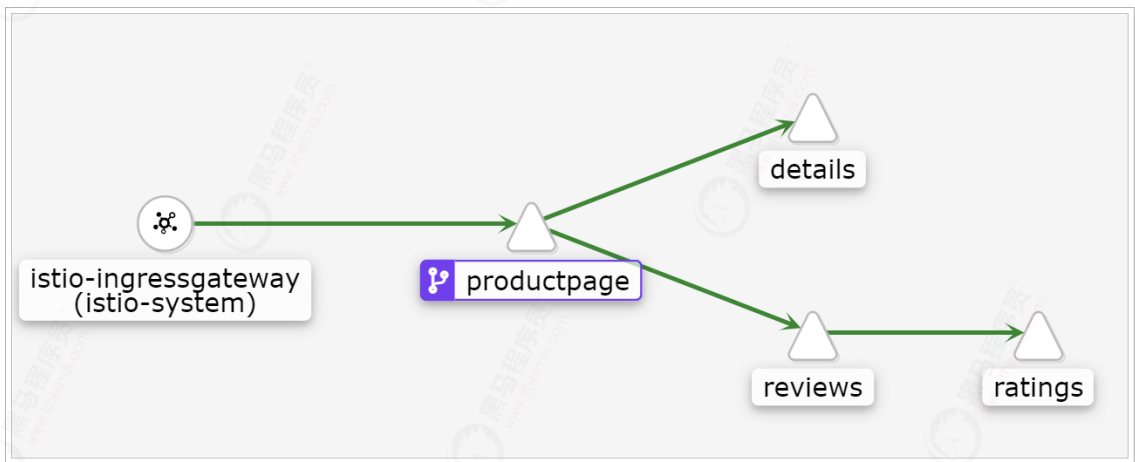


- Workload

- 图类型显示了服务网格中每个工作负载的节点。



- Service
 - 图类型显示网格中每个服务的节点。



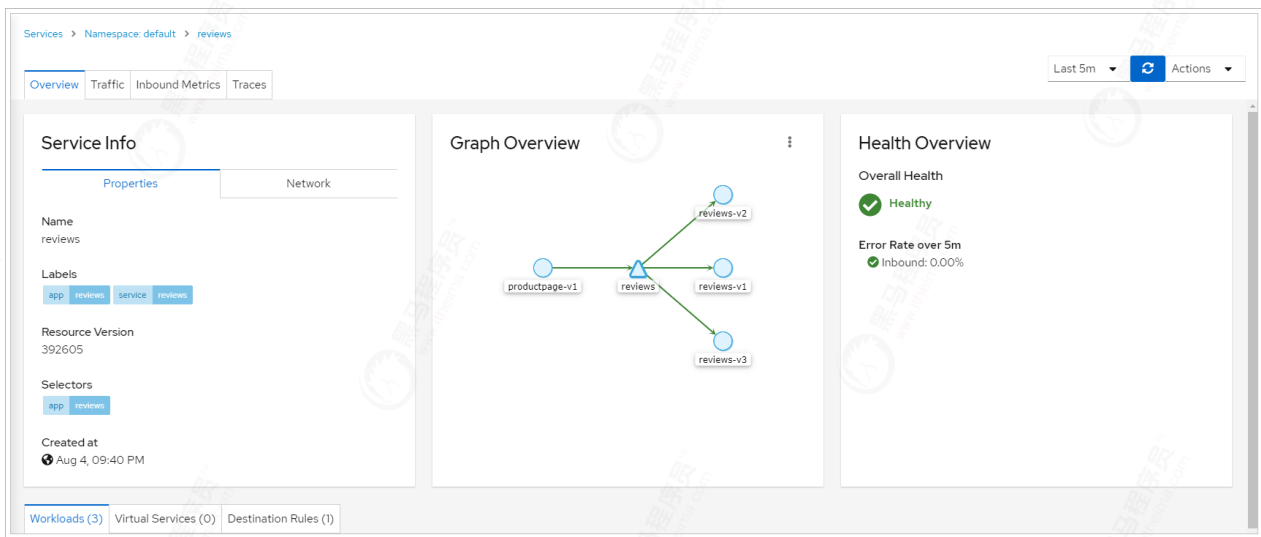
2.4.2、路由加权

默认路由规则会平均分配浏览到每个可用节点，通过kiali可以可行可视化的调节：

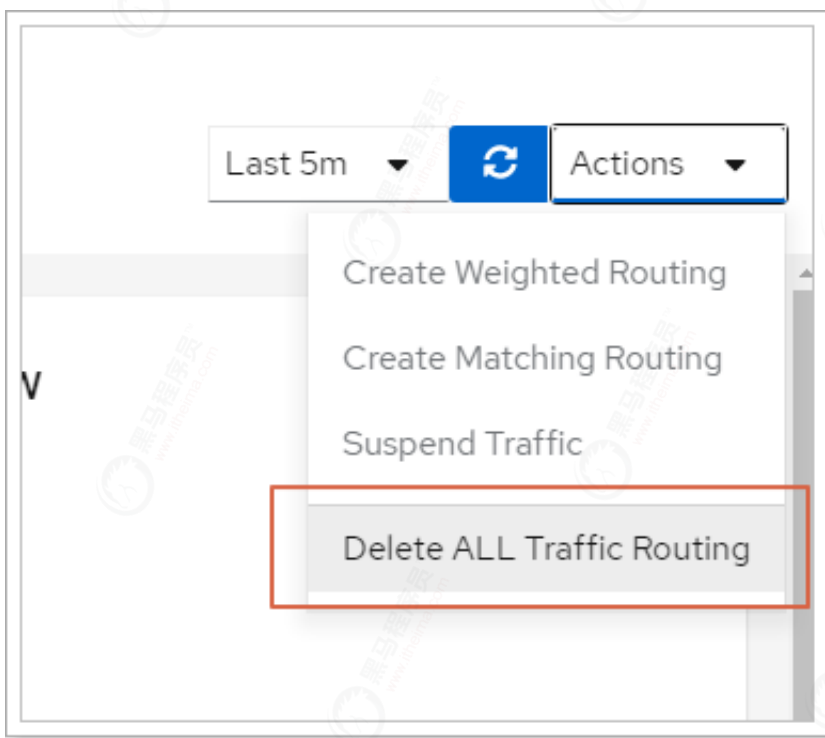
第一步，查看servers列表：

Name	Namespace	Labels	Health	Configuration	Details
details	default	app: details service: details	✓	✓	
fortio	default	app: fortio	⚠	✓	
kubernetes	default	component: apiserver provider: kubernetes	⚠	✓	
productpage	default	app: productpage service: productpage	✓	✓	
ratings	default	app: ratings service: ratings	✓	✓	
reviews	default	app: reviews service: reviews	✓	✓	

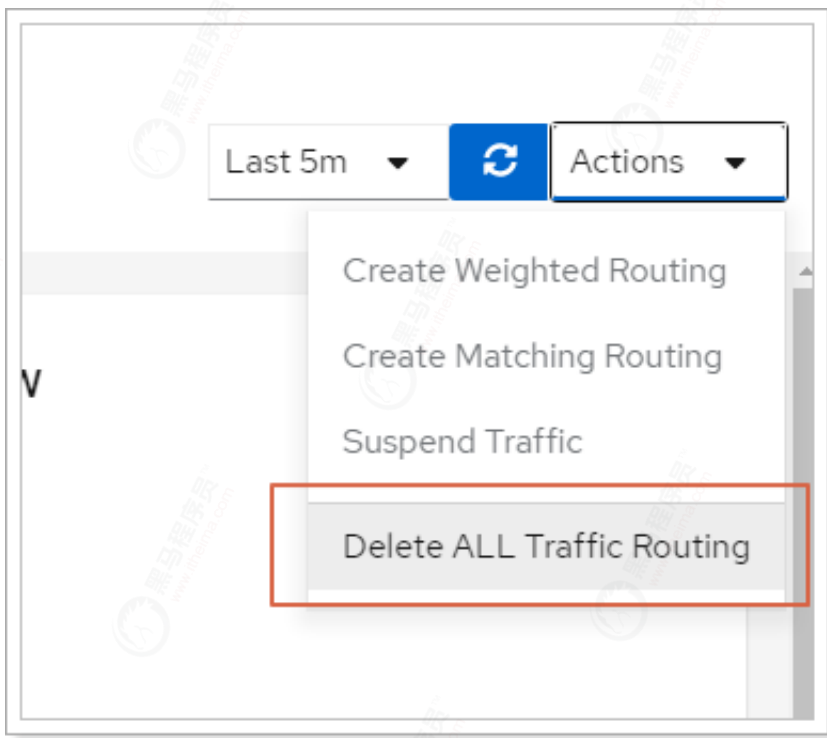
第二步，进入reviews服务：



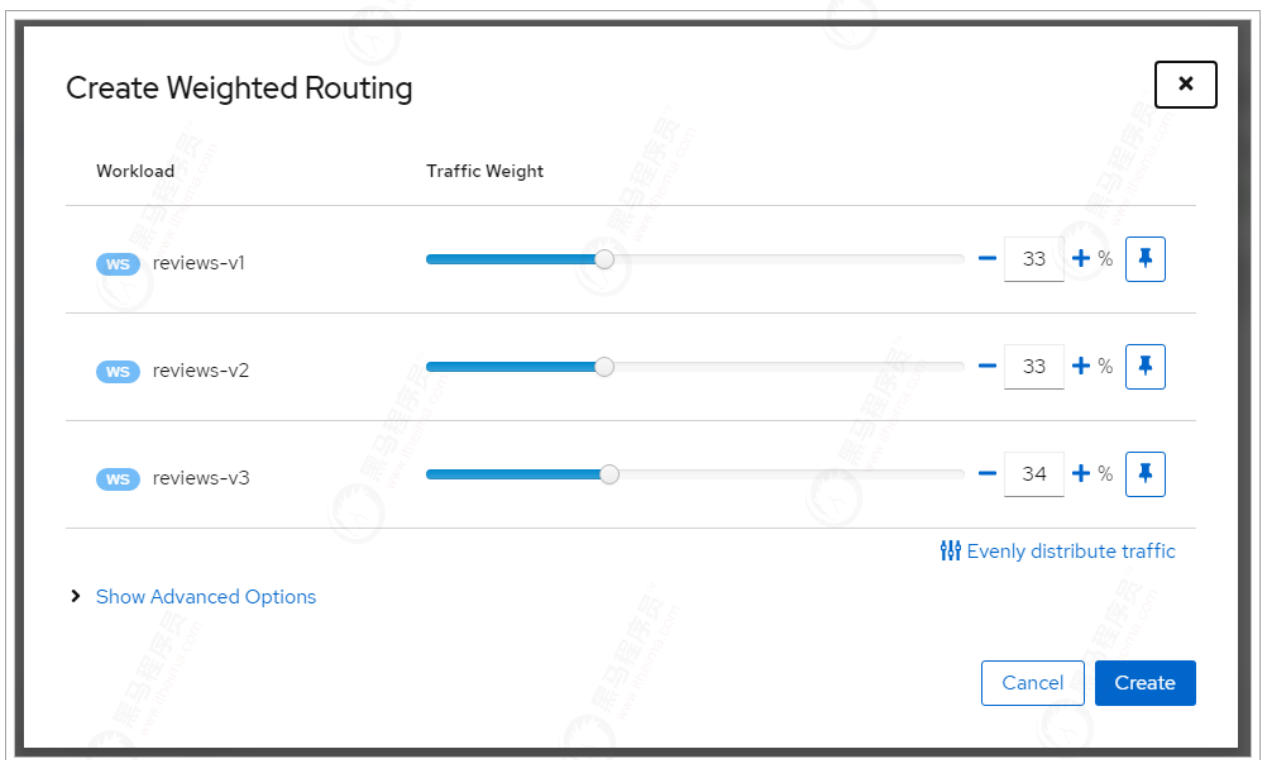
第三步，删除原有路由规则：



第四步，创建权重的规则：



默认情况：



进行调整：

Create Weighted Routing

Workload	Traffic Weight
WS reviews-v1	- 15 + %
WS reviews-v2	- 30 + %
WS reviews-v3	- 55 + %

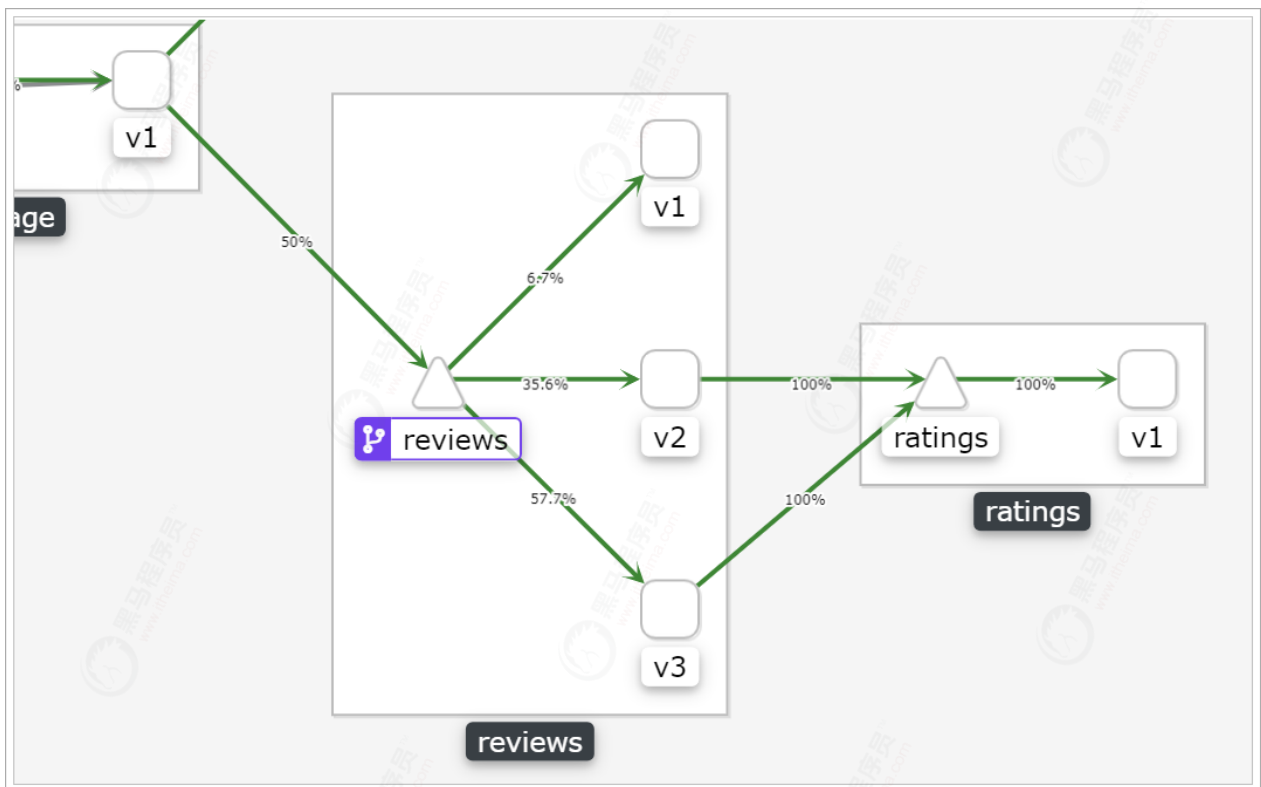
Evenly distribute traffic

[Show Advanced Options](#)

Cancel Create

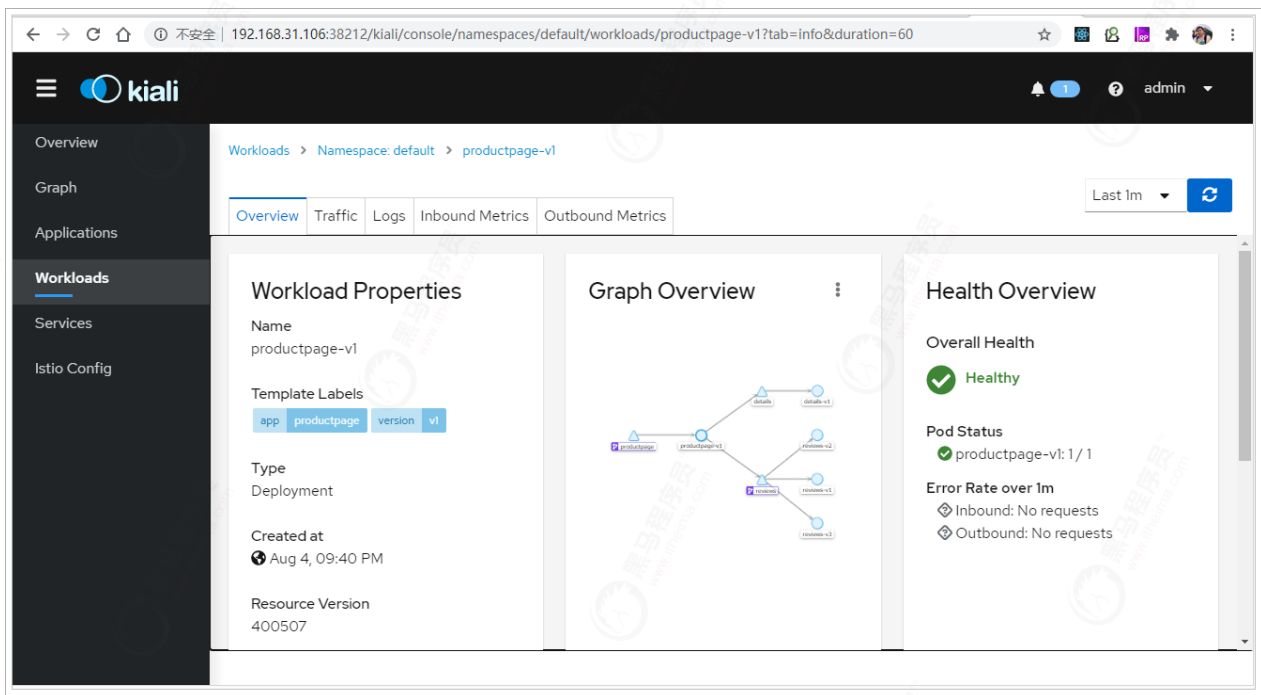
保存操作。

第五步，通过watch执行一段时间，观察效果：

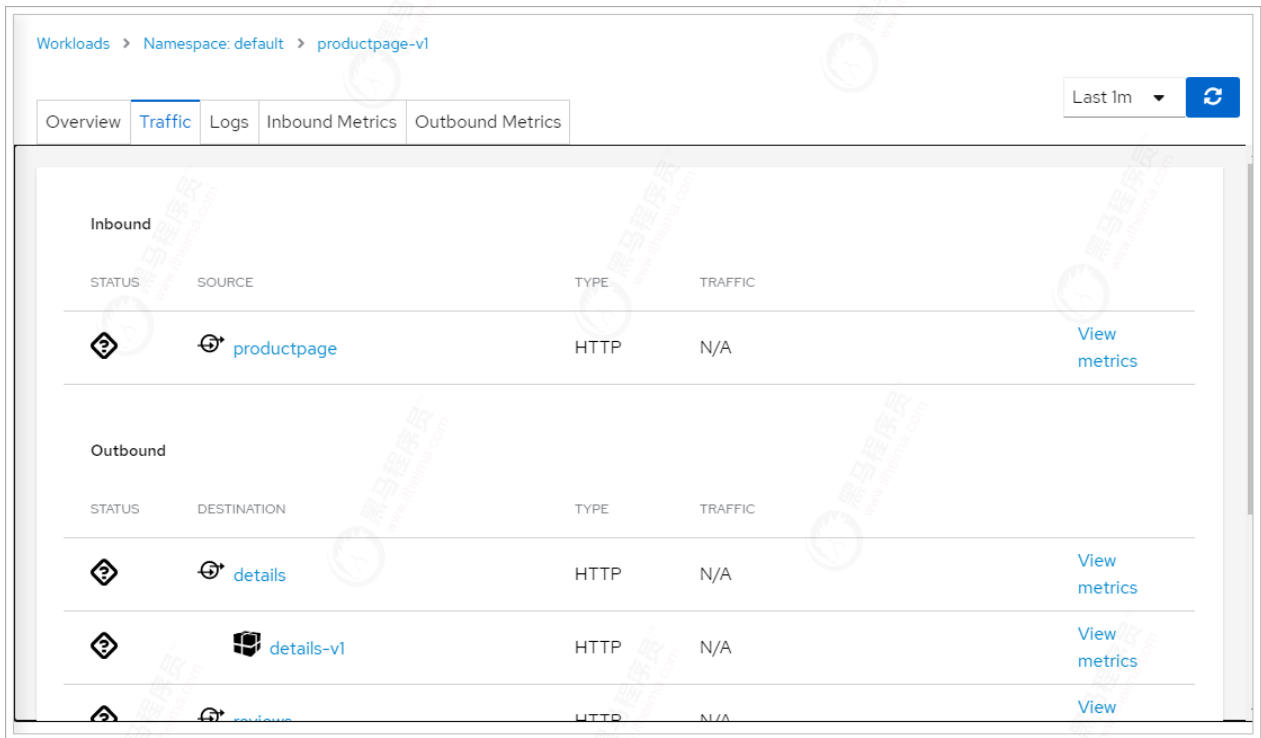


可以看到，分配到reviews的v1、v2、v3的百分比发生了变化。

2.4.3、查看工作负载



入站、出站信息：



日志信息：

Workloads > Namespace: default > productpage-v1

Overview Traffic **Logs** Inbound Metrics Outbound Metrics

Pod productpage-v1-7df7cb7f86-gq9vw Side by Side Last 500 lines Last 10m Every 15s

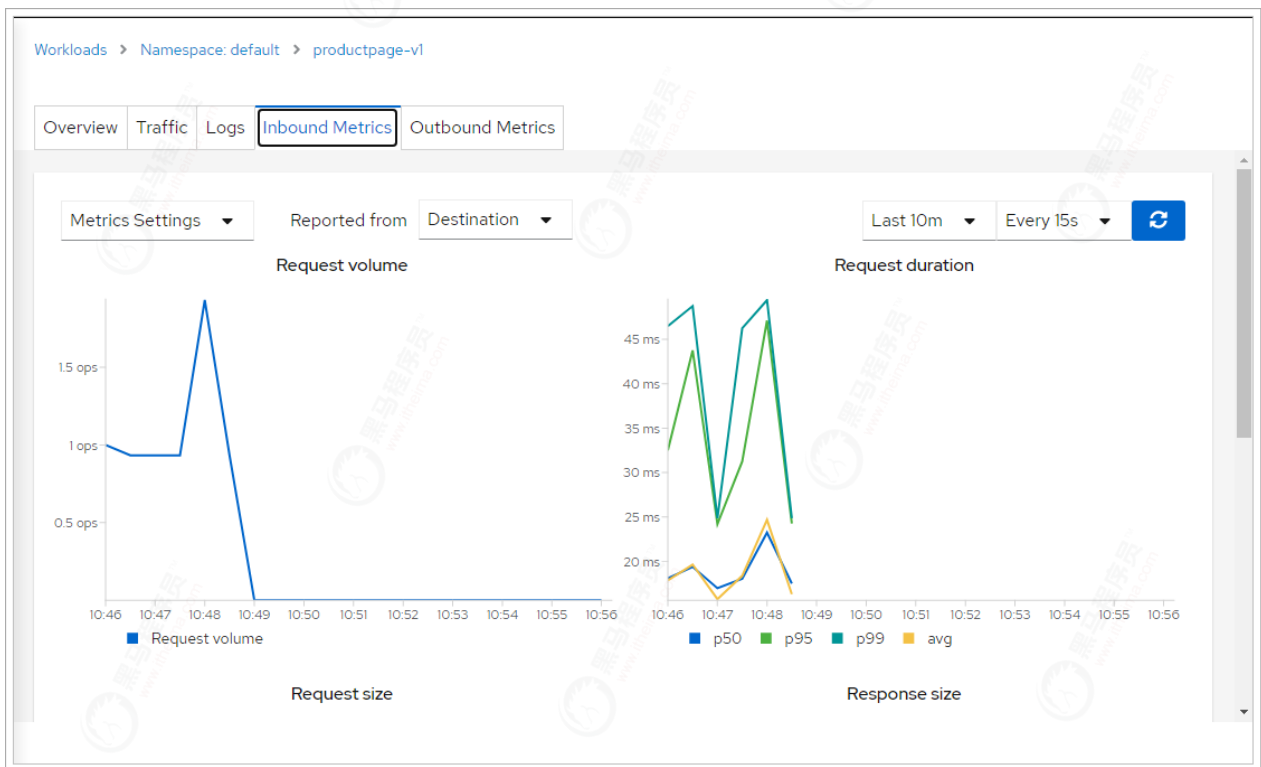
productpage-v1

```
2020-08-04T14:48:29.190660117Z header: content-language: en-US
2020-08-04T14:48:29.190661771Z header: content-length: 375
2020-08-04T14:48:29.190663393Z header: x-envoy-upstream-service-time: 6
2020-08-04T14:48:29.190665030Z header: server: envoy
2020-08-04T14:48:29.190666677Z DEBUG:urllib3.connectionpool:http://reviews:9080 "GET /reviews/0 HTTP/1.1" 200 375
2020-08-04T14:48:29.190970184Z INFO:werkzeug::ffff:127.0.0.1 - - [04/Aug/2020 14:48:29] "GET /productpage HTTP/1.1" 200
```

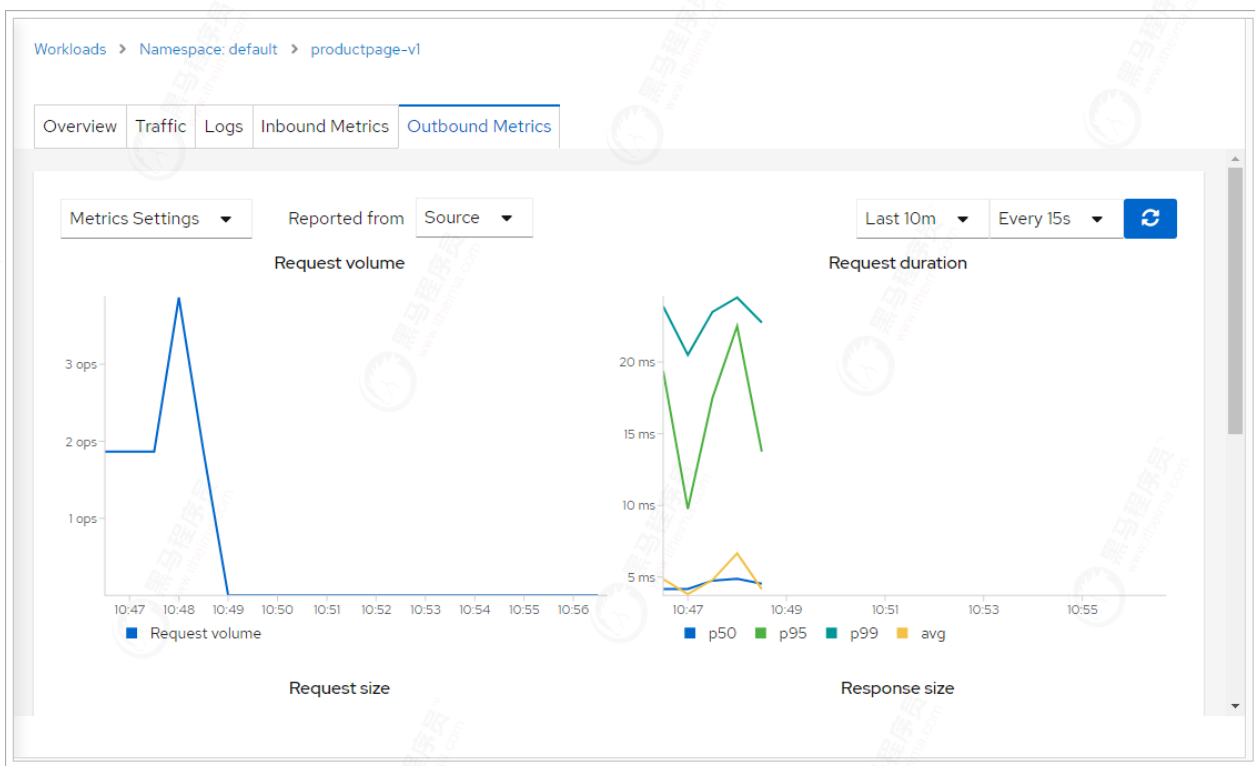
Istio proxy

```
2020-08-04T14:48:36.472222682Z [2020-08-04T14:48:28.136Z] "GET /productpage HTTP/1.1" 200 - "-" 0 5179 26 26 "10.244
2020-08-04T14:48:36.472226043Z [2020-08-04T14:48:29.179Z] "GET /details/0 HTTP/1.1" 200 - "-" 0 178 1 1 "curl/7.3
2020-08-04T14:48:36.472228314Z [2020-08-04T14:48:29.182Z] "GET /reviews/0 HTTP/1.1" 200 - "-" 0 375 6 6 "curl/7.3
2020-08-04T14:48:36.472251282Z [2020-08-04T14:48:29.176Z] "GET /productpage HTTP/1.1" 200 - "-" 0 5179 14 14 "10.244
```

进站指标信息：



出站指标信息：



3、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。

3.1、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。

说明：对于一些对于图数据库不熟悉的同学，将聚焦到服务演变的过程，不要聚焦到Neo4j上，就将其看做的是mysql数据一样。（其实他们是不一样的）

下面是电影页面的截图，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。



想看 看过 评价: ☆☆☆☆☆

写短评 写影评 分享到

推荐

让子弹飞的剧情简介 ·····

民国年间，花钱捐得县长的马邦德（葛优 饰）携妻（刘嘉玲 饰）及随从走马上任。途经南国某地，遭劫匪张麻子（姜文 饰）一伙伏击，随从尽死，只夫妻二人侥幸活命。马为保命，谎称自己是县长的汤 爷。为汤爷许下的财富所动，张麻子摇身一变化身县长，带着手下赶赴鹅城上任。有道是天高皇帝远，鹅城地处偏僻，一方霸主黄四郎（周润发 饰）只手遮天，全然不将这个新来的县长放在眼里。张麻子痛打了黄的武教头（姜武 饰），黄则设计害死张的义子小六（张默 饰）。原本只想赚钱的马邦德，怎么也想不到竟会卷入这场土匪和恶霸的角力之中。鹅城上空愁云密布，血雨腥风在所难免.....

让子弹飞的演职员 ·····



姜文
导演



刘嘉玲
饰 县长夫人



周润发
饰 黄四郎



葛优
饰 马邦德



姜文
饰 张牧之

喜欢这部电影的人也喜欢 ·····



人在囧途



私人订制



建国大业



天下无贼



甲方乙方

让子弹飞的短评 ····· (全部 191039 条)

我要写短评

热门 / 最新 / 好友

陈野犁 看过★★★★★ 2010-12-07

因为对《让子弹飞》吹捧得无耻到惊世骇俗的地步，本短评已被和谐。

桃桃林林 看过★★★★★ 2010-12-14

姜文玩出来的电影，从头到尾都很放松，或者说是太放松了，节奏总有点怪，有些情节也太儿戏。对白也不好，细节不如《鬼子来了》那么智慧，即便如此，这仍然是今年华语片最棒的。或者只能怪剩下那些太面货。请千万一定尽量降低期待。重看一遍，感觉这片舞台味真重啊。

于少 看过★★★★★ 2010-11-27

牛着逼！

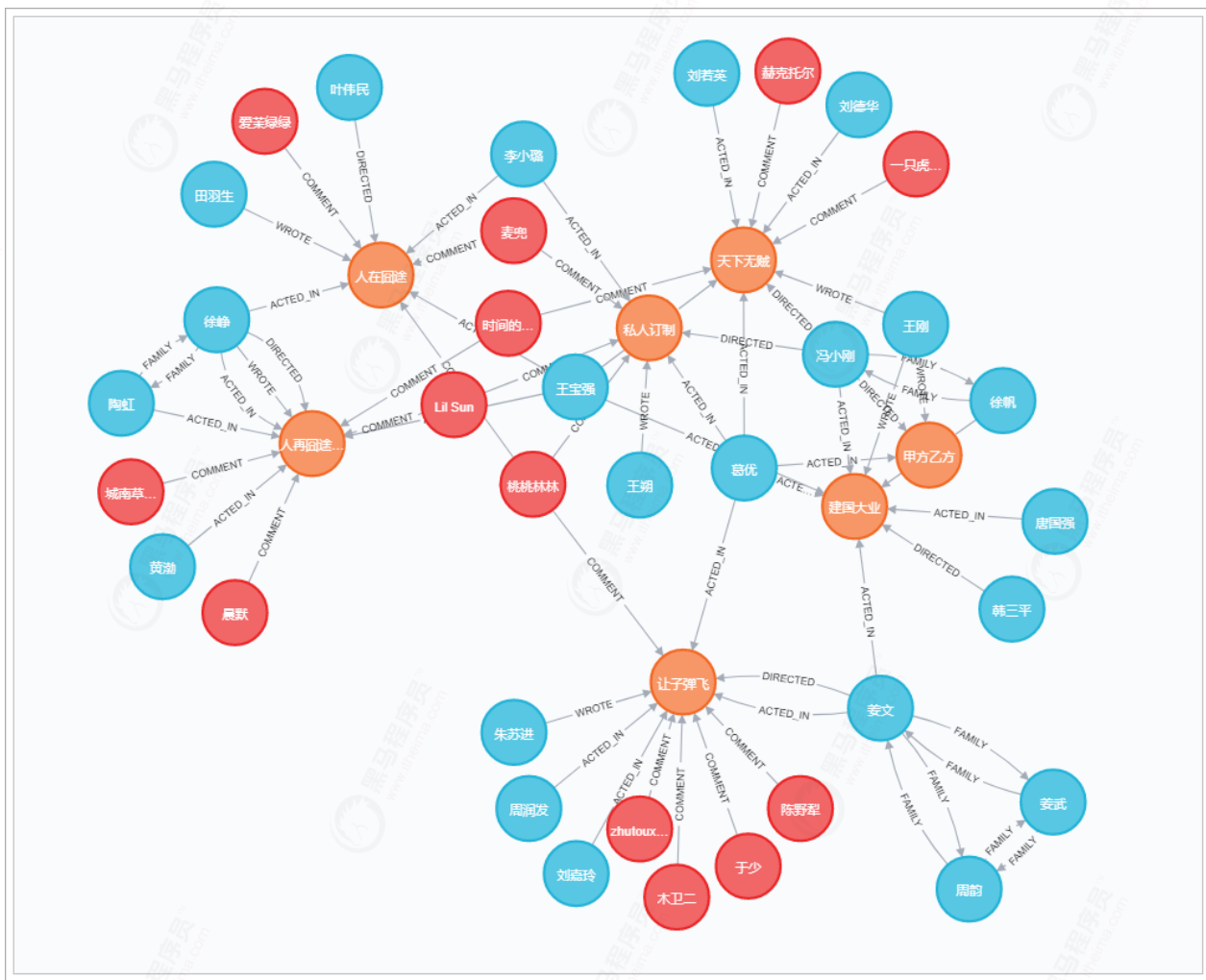
木卫二 看过★★★★★ 2010-12-04

你给我翻译翻译，神马叫做TMD的惊喜！

zhutouxiaoduizhang 看过★★★★★ 2010-12-05

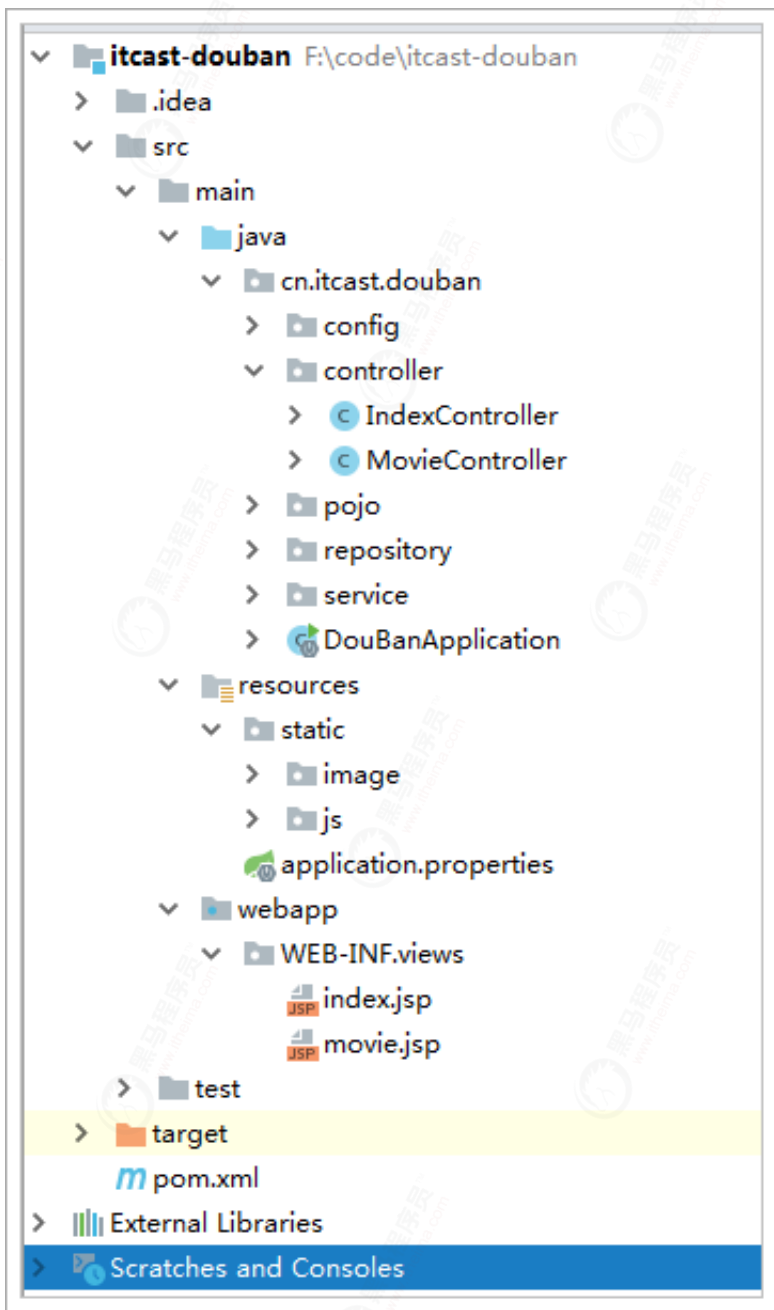
仍然有浪漫的想象。癫狂的审美、极致的表演，然而姜文这次说，他要让所有人都看懂，于是他不过对任何细枝末节的解释。当彪悍的人生开始解释，所以，情怀没了，诗意没了，理所当然。难道所有导演的审美都要假装去迎合观众而连带着水准一起拉低？好吧，那市场胜利了，这下子都能看懂，都能笑了。

数据：



橙色球：电影，蓝色球：参与人，红色球：普通用户；

项目结构：



3.2、搭建Neo4j环境

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：

\$

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

\$:server connect

Connect to Neo4j

Database access might require an authenticated connection.

Connect URL

bolt://192.168.31.106:7687

Authentication type

Username / Password

Username

neo4j

Password

.....

Connect

默认用户名密码都是：
neo4j

首次登录需要修改密码，设置为：neo4j123

\$:server connect

Connect to Neo4j

Database access might require an authenticated connection.

New password

.....

Repeat new password

.....

Change password

在上面的命令输入框中写入数据脚本，点击右三角进行执行：

```
152 (chengnancaomusheng)-[:COMMENT {score:4, date:"2012-12-05", desc:"作为导演的徐峥够心平气和，自恋也够克制，王宝强挺好但搞笑桥段依然无任何进步，他出现在任何一部喜剧里都会把效果充分掌握成春晚，不知道会不会破坏徐峥初衷，黄渤根本就是浪费，希望徐峥一直拍，但诚意满满和优秀处女作之类永远不是一部电影加分的标准。今年最好看喜剧，但也只是很正常的喜剧而已。"}]-(renzaijiongtuzhitaijiong),
153 (LilSun)-[:COMMENT {score:5, date:"2012-12-05", desc:"去影院花钱再看一次也值得。"}]-(renzaijiongtuzhitaijiong),
154 (shijiandemeigui)-[:COMMENT {score:3, date:"2012-12-10", desc:"说想象的好，纯乐呵，仍然是一衰一损的捧逗组合，再加个T-1000型的屎屁虫，三人互相拆台，插科打诨。公路喜剧的特点就是“衰”，徐峥仍然以这个字为中心，出彩都在王宝强一人身上，黄渤就是个出气筒，只增色不抢戏。"}]-(renzaijiongtuzhitaijiong)
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)

数据文件在资料目录下的 <电影数据.txt> 文件中。

neo4j\$ CREATE (rangzidanfei:Movie {movieId:3742360,title:'让子弹飞',released:'2010-12-16(中国大陆)',types:["剧情","喜剧","动作","西部"],producerCountry:'中国大陆 / 中国'})

Added 42 labels, created 42 nodes, set 398 properties, created 66 relationships, completed after 102 ms.

Table

Code

Added 42 labels, created 42 nodes, set 398 properties, created 66 relationships, completed after 102 ms.

执行成功。

执行查询，命令：match (n) return n


```
<module>itcast-service-mesh-movie-rating</module>
<module>itcast-service-mesh-movie-common</module>
</modules>

<groupId>cn.itcast.servicemesh</groupId>
<artifactId>itcast-service-mesh</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>
```


3.3.2、itcast-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>itcast-service-mesh</artifactId>
    <groupId>cn.itcast.servicemesh</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>itcast-service-mesh-movie-common</artifactId>

</project>
```

3.3.3、itcast-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>itcast-service-mesh</artifactId>
    <groupId>cn.itcast.servicemesh</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>itcast-service-mesh-movie-info</artifactId>

  <dependencies>
    <dependency>
      <groupId>cn.itcast.servicemesh</groupId>
      <artifactId>itcast-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>
      </plugin>
    </plugins>
  </build>
</project>
```

```

        <configuration>
            <mainClass>cn.itcast.movie.MovieInfoApplication</mainClass>
            <outputDirectory>F:\\publish</outputDirectory>
        </configuration>
        <executions>
            <execution>
                <goals>
                    <goal>repackage</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>

</project>

```

3.3.4、itcast-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>itcast-service-mesh</artifactId>
        <groupId>cn.itcast.servicemesh</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-service-mesh-movie-rating</artifactId>

    <dependencies>
        <dependency>
            <groupId>cn.itcast.servicemesh</groupId>
            <artifactId>itcast-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.itcast.movie.MovieRatingApplication</mainClass>

```

```

        <outputDirectory>F:\\publish</outputDirectory>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>repackage</goal>
            </goals>
        </execution>
    </executions>
</plugin>
</plugins>
</build>

</project>

```

3.3.5、itcast-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>itcast-service-mesh</artifactId>
        <groupId>cn.itcast.servicemesh</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-service-mesh-movie-recommend</artifactId>

    <dependencies>
        <dependency>
            <groupId>cn.itcast.servicemesh</groupId>
            <artifactId>itcast-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.itcast.movie.MovieRecommendApplication</mainClass>
                <outputDirectory>F:\\publish</outputDirectory>
            </configuration>
        </plugins>
    </build>

```

```

        <executions>
            <execution>
                <goals>
                    <goal>repackage</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>

</project>

```

3.3.6、itcast-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>itcast-service-mesh</artifactId>
        <groupId>cn.itcast.servicemesh</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-service-mesh-movie-web</artifactId>

    <dependencies>
        <dependency>
            <groupId>cn.itcast.servicemesh</groupId>
            <artifactId>itcast-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>

```

```

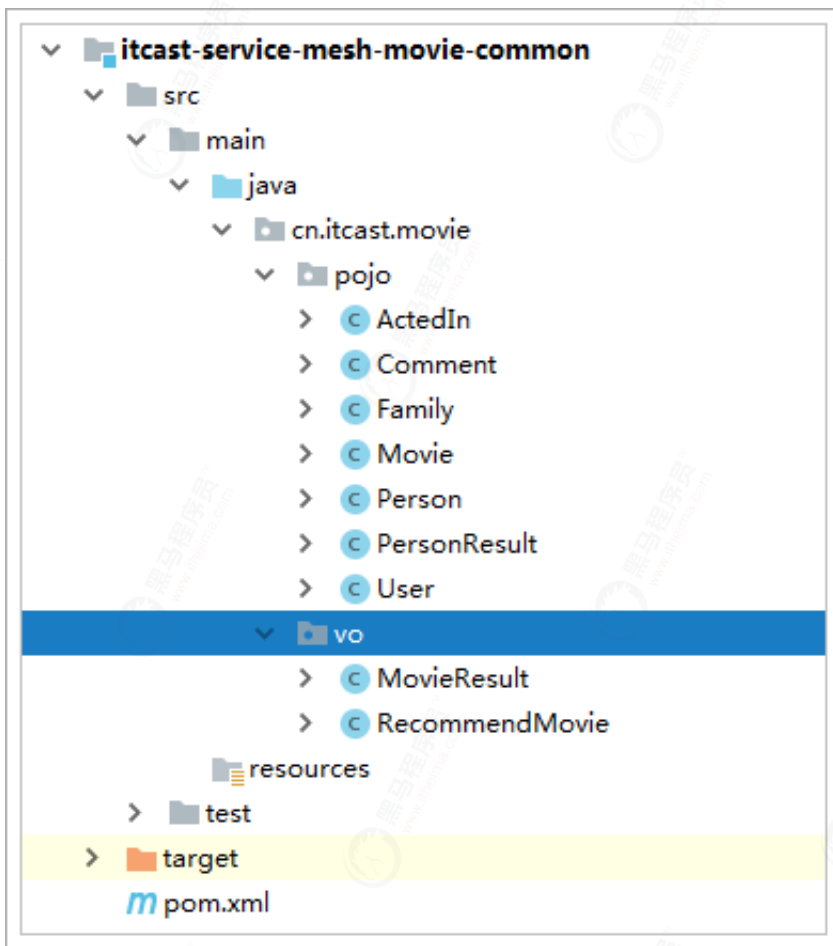
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>1.4.2.RELEASE</version>
      <configuration>
        <mainClass>cn.itcast.movie.MovieApplication</mainClass>
        <outputDirectory>F:\publish</outputDirectory>
      </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>

```

3.4、movie-common

itcast-service-mesh-movie-common工程中，存放了公用的pojo、vo对象，在资料目录文件可以找到。



3.5、movie-info

该工程进行对象电影信息进行查询。

3.5.1、config

```
package cn.itcast.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.itcast.movie.repository")
@EntityScan("cn.itcast.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.itcast.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.itcast")
@Configuration
public class MyConfig implements WebMvcConfigurer {

}

```

3.5.2、repository

```

package cn.itcast.movie.repository;

import cn.itcast.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);
}

```

3.5.3、service

```

package cn.itcast.movie.service;

import cn.itcast.movie.pojo.Movie;
import cn.itcast.movie.repository.MovieRepository;
import cn.itcast.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);

        //对象数据的拷贝
        BeanUtils.copyProperties(movie, movieResult);

        return movieResult;
    }
}

```

3.5.4、controller

```

package cn.itcast.movie.controller;

import cn.itcast.movie.service.MovieService;
import cn.itcast.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);
    }
}
```

3.5.5、启动类

```
package cn.itcast.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);
    }
}
```

3.5.6、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
```

3.5.7、测试



3.6、movie-recommend

3.6.1、config

与movie-info工程中一样，这里就省略了。

3.6.2、repository

```
package cn.itcast.movie.repository;

import cn.itcast.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);
}

```

3.6.3、service

```

package cn.itcast.movie.service;

import cn.itcast.movie.pojo.Movie;
import cn.itcast.movie.repository.MovieRepository;
import cn.itcast.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;
    }
}

```

3.6.4、controller

```

package cn.itcast.movie.controller;

import cn.itcast.movie.service.MovieService;
import cn.itcast.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);
    }
}

```

3.6.5、启动类

```

package cn.itcast.movie;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MovieRecommendApplication {

    public static void main(String[] args) {
        SpringApplication.run(MovieRecommendApplication.class, args);
    }
}

```

3.6.6、配置

application.properties

```

server.port=18083

#连接地址
spring.data.neo4j.uri=bolt://neo4j-server:7687

```

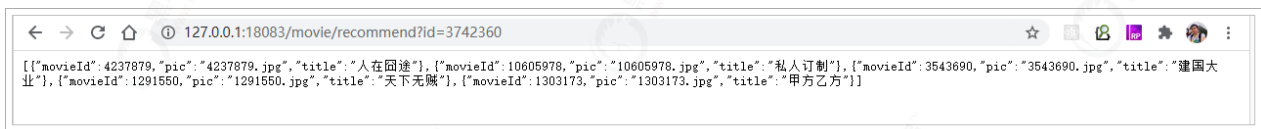


```
#用户名
spring.data.neo4j.username=neo4j
#密码
spring.data.neo4j.password=neo4j123

#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
```

3.6.7、测试



3.7、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。

3.7.1、config

与movie-info工程中一样，这里就省略了。

3.7.2、repository

```
package cn.itcast.movie.repository;

import cn.itcast.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);
}
```

3.7.3、service

```
package cn.itcast.movie.service;
```

```
import cn.itcast.movie.pojo.Comment;
import cn.itcast.movie.pojo.Movie;
import cn.itcast.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_sect1\">\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\">黑马评分</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\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));
}
}

```

3.7.4、controller

```

package cn.itcast.movie.controller;

import cn.itcast.movie.service.MovieService;
import cn.itcast.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);
    }
}

```

3.7.5、启动类

```

package cn.itcast.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);
    }

}

```

3.7.6、配置

```

server.port=18084

#连接地址
spring.data.neo4j.uri=bolt://neo4j-server:7687

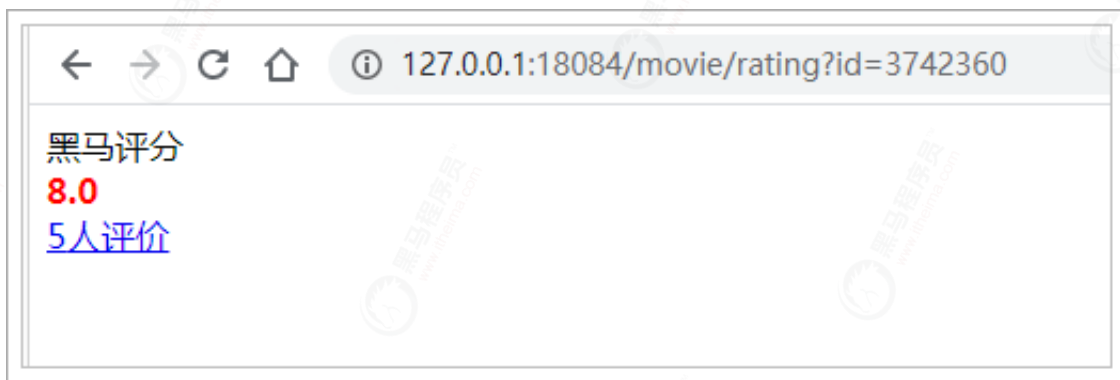
#用户名
spring.data.neo4j.username=neo4j
#密码
spring.data.neo4j.password=neo4j123

#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

```

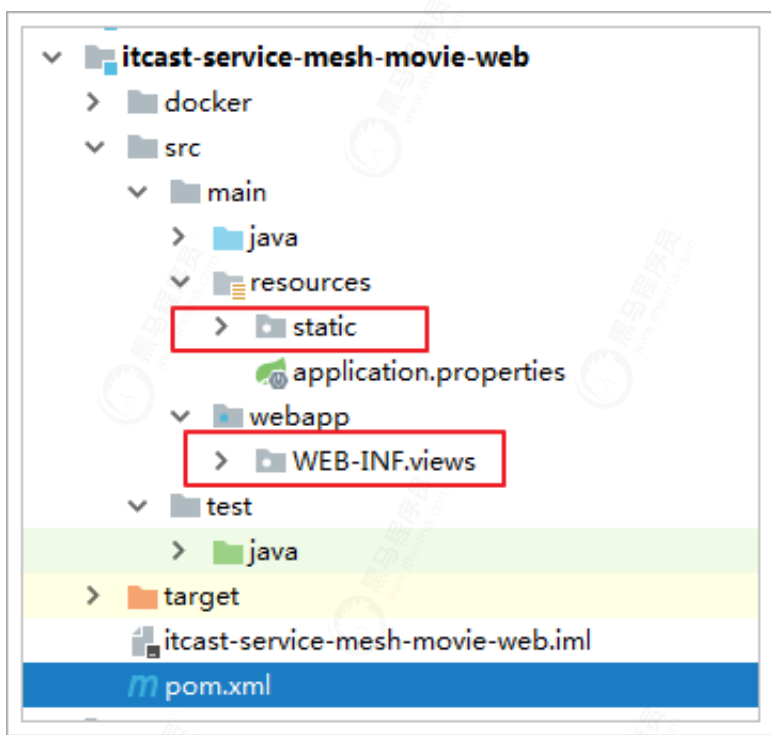
3.7.7、测试



3.8、movie-web

3.8.1、静态资源

将资料中的静态资源导入：



3.8.2、配置

application.properties

```
server.port=18081

spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

#logging.level.org.springframework=DEBUG
```

3.8.3、config

```

package cn.itcast.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.itcast")
@Configuration
public class MyConfig implements WebMvcConfigurer {

}

```

```

package cn.itcast.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;
    }
}

```

3.8.4、service

```

package cn.itcast.movie.service;

```



```

import cn.itcast.movie.vo.MovieResult;
import cn.itcast.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=\"" +
                    "<dt>\n" +
                    "<a href=\"" + movie.getMovieId() + "\">\n" +
                    "<img src=\"" + movie.getPic() + "\"/>\n" +
                    "</a>\n" +
                    "</dt>\n" +
                    "<dd>\n" +
                    "<a href=\"" + 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);
    }
}

```

3.8.5、controller

```

package cn.itcast.movie.controller;

import cn.itcast.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;

/**
 * 影片页
 */
@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;
    }
}

```

3.8.6、启动类

```
package cn.itcast.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);
    }
}
```

3.8.7、测试



3.9、制作docker镜像

编写Dockerfile:

```
#itcast-service-mesh-movie-info
FROM openjdk:8-jdk-alpine
COPY ./itcast-service-mesh-movie-info-1.0-SNAPSHOT.jar /movie/itcast-service-mesh-movie-info-1.0-SNAPSHOT.jar
EXPOSE 18082

ENTRYPOINT [ "java", "-jar", "/movie/itcast-service-mesh-movie-info-1.0-SNAPSHOT.jar" ]
```

```

#itcast-service-mesh-movie-rating
FROM openjdk:8-jdk-alpine
COPY ./itcast-service-mesh-movie-rating-1.0-SNAPSHOT.jar /movie/itcast-service-
mesh-movie-rating-1.0-SNAPSHOT.jar
EXPOSE 18084

ENTRYPOINT [ "java","-jar","/movie/itcast-service-mesh-movie-rating-1.0-
SNAPSHOT.jar" ]

#itcast-service-mesh-movie-recommend
FROM openjdk:8-jdk-alpine
COPY ./itcast-service-mesh-movie-recommend-1.0-SNAPSHOT.jar /movie/itcast-
service-mesh-movie-recommend-1.0-SNAPSHOT.jar
EXPOSE 18083

ENTRYPOINT [ "java","-jar","/movie/itcast-service-mesh-movie-recommend-1.0-
SNAPSHOT.jar" ]

#itcast-service-mesh-movie-web
FROM openjdk:8-jdk-alpine
COPY ./itcast-service-mesh-movie-web-1.0-SNAPSHOT.jar /movie/itcast-service-
mesh-movie-web-1.0-SNAPSHOT.jar
EXPOSE 18081

ENTRYPOINT [ "java","-jar","/movie/itcast-service-mesh-movie-web-1.0-
SNAPSHOT.jar" ]

```

制作镜像:

```

mkdir -p /movie/movie-info
cd /movie/movie-info
vim Dockerfile
#拷贝上面的内容到文件中

#构建镜像
docker build -t itcast-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=neo4j-
server:192.168.31.106 itcast-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 itcast-service-mesh-movie-recommend:1.0 .

```

```
docker create --name movie-recommend -p 18083:18083 --add-host=neo4j-server:192.168.31.106 itcast-service-mesh-movie-recommend:1.0
docker start movie-recommend

docker build -t itcast-service-mesh-movie-rating:1.0 .
docker create --name movie-rating -p 18084:18084 --add-host=neo4j-server:192.168.31.106 itcast-service-mesh-movie-rating:1.0
docker start movie-rating

docker create --name movie-rating-v2 -p 18085:18084 --add-host=neo4j-server:192.168.31.106 --env MOVIE_VERSION=v2 itcast-service-mesh-movie-rating:1.0
docker start movie-rating-v2

docker build -t itcast-service-mesh-movie-web:1.0 .
docker create --name movie-web -p 18081:18081 --add-host=movie-info:192.168.31.106 --add-host=movie-recommend:192.168.31.106 --add-host=movie-rating:192.168.31.106 itcast-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 itcast-service-mesh-movie-info:1.0 registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-info:1.0
docker push registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-info:1.0
```

#recommend

```
docker tag itcast-service-mesh-movie-recommend:1.0 registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-recommend:1.0
docker push registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-recommend:1.0
```

#rating

```
docker tag itcast-service-mesh-movie-rating:1.0 registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-rating:1.0
```



```
docker push registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-
rating:1.0
```

```
#web
```

```
docker tag itcast-service-mesh-movie-web:1.0 registry.cn-
hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-web:1.0
docker push registry.cn-hangzhou.aliyuncs.com/itcast/itcast-service-mesh-movie-
web:1.0
```

3.10、编写Istio相关文件

3.10.1、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/itcast/itcast-service-mesh-
movie-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/itcast/itcast-service-mesh-
movie-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/itcast/itcast-service-mesh-
movie-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/itcast/itcast-service-mesh-
movie-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/itcast/itcast-service-mesh-
movie-web:1.0
          imagePullPolicy: IfNotPresent
          ports:

```

```
- containerPort: 18081
```

```
---
```

3.10.2、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
```

3.10.3、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
---
```

3.10.4、流量全部导入到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
```

3.11、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。

3.11.1、Istio初始化失败

如果出现如下错误：

```
[root@node1 istio-1.6.5]# istioctl manifest apply --set profile=demo
Detected that your cluster does not support third party JWT authentication. Falling back to less secure f
ps/best-practices/security/#configure-third-party-service-account-tokens for details.
✓ Istio core installed
✓ Istiod installed
✓ Egress gateways installed
✓ Ingress gateways installed
✗ Addons encountered an error: failed to wait for resource: resources not ready after 5m0s: timed out wai
Deployment/istio-system/kiali
- Pruning removed resources
Error: failed to apply manifests: errors occurred during operation
[root@node1 istio-1.6.5]#
```

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/itcast/kiali:v1.18
docker tag registry.cn-hangzhou.aliyuncs.com/itcast/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

3.11.2、实施

```
mkdir /movie
cd /movie/
#将destination-rule-all.yaml movie-gateway.yaml movie.yaml virtual-service-
rating-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
movie-info-deployment-c8f455f69-4hknf	0/2	Init:0/1
40s <none> node3 <none>	<none>	
movie-info-deployment-c8f455f69-psbb4	0/2	Init:0/1
40s <none> node2 <none>	<none>	
movie-rating-deployment-v1-667fb7c8cc-j8ggj	0/2	PodInitializing
40s 10.244.1.8 node2 <none>	<none>	
movie-rating-deployment-v2-577d48b98f-fg2nj	0/2	Init:0/1
40s <none> node3 <none>	<none>	
movie-recommend-deployment-747d6dc646-8zv7j	0/2	PodInitializing
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)
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
movie-info	ClusterIP	10.108.31.87	<none>	18082/TCP
movie-rating	ClusterIP	10.107.230.15	<none>	18084/TCP
movie-recommend	ClusterIP	10.97.144.103	<none>	18083/TCP
movie-web	ClusterIP	10.104.57.57	<none>	18081/TCP
neo4j-server	NodePort	10.106.196.213	<none>	7474:31001/TCP,7687:31002/TCP

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

on.

Connect URL

bolt://192.168.31.106:31002

Authentication type

Username / Password

Username

neo4j

Password

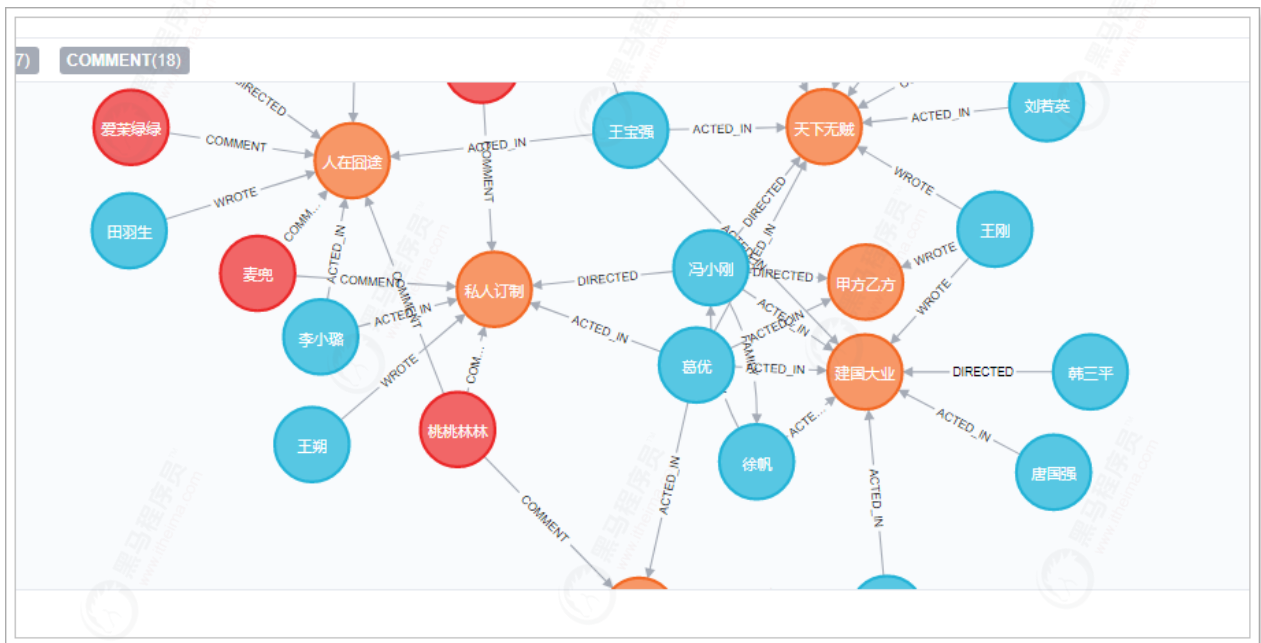
.....

Connect

152 (chengnancaomusheng)-[:COMMENT {score:4, date:"2012-12-05", desc:"作为导演的徐峥心平气和，自恋也够克制，王宝强挺好但搞笑桥段依然无任何进步，他出现在任何一部喜剧里都会把效果充分掌握成春晚，不知道会不会破坏徐峥初衷。黄渤根本就是浪费。希望徐峥一直拍，但诚意满满和优秀处女作之类永远不是一部电影加分的标准。今年最好看喜剧，但也只是很正常的喜剧而已。"}]→(renzaijiongtuzhitaijiong),

153 (LilSun)-[:COMMENT {score:5, date:"2012-12-05", desc:"去影院花钱再看一次也值得。"}]→(renzaijiongtuzhitaijiong),

154 (shijiangdemeigui)-[:COMMENT {score:3, date:"2012-12-10", desc:"设想象的好，纯乐呵。仍然是一衰一损的捧逗组合，再加个T-1000型的跟屁虫，三人互相拆台，插科打诨。公路喜剧的特点就是“衰”，徐峥仍然以这个字为中心，出彩都在王宝强一人身上，黄渤就是个出气筒，只增色不抢戏。"}]→(renzaijiongtuzhitaijiong)



#查看pod是否已经就绪

```
[root@node1 movie]# kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE
IP				
NOMINATED	NODE			
READINESS	GATES			
movie-info-deployment-c8f455f69-4hknf	2/2	Running	0	10m
10.244.2.9	node3	<none>		
movie-info-deployment-c8f455f69-psbb4	2/2	Running	0	10m
10.244.1.10	node2	<none>		

movie-rating-deployment-v1-667fb7c8cc-j8ggj	2/2	Running	0	10m
10.244.1.8 node2 <none>	<none>			
movie-rating-deployment-v2-577d48b98f-fg2nj	2/2	Running	0	10m
10.244.2.11 node3 <none>	<none>			
movie-recommend-deployment-747d6dc646-8zv7j	2/2	Running	0	10m
10.244.1.7 node2 <none>	<none>			
movie-recommend-deployment-747d6dc646-g9bcb	2/2	Running	0	10m
10.244.2.7 node3 <none>	<none>			
movie-recommend-deployment-747d6dc646-sxbj7	2/2	Running	0	10m
10.244.2.8 node3 <none>	<none>			
movie-web-deployment-6996d99464-cc88t	2/2	Running	0	10m
10.244.1.11 node2 <none>	<none>			
movie-web-deployment-6996d99464-gjfm6	2/2	Running	0	10m
10.244.2.10 node3 <none>	<none>			
movie-web-deployment-6996d99464-t29tz	2/2	Running	0	10m
10.244.1.9 node2 <none>	<none>			
neo4j-deployment-6bb95596c8-sld6n	2/2	Running	0	10m
10.244.2.6 node3 <none>	<none>			

#可以看到所有的pod状态都为Running, 说明已经准备就绪

#创建网关

```
[root@node1 movie]# kubectl apply -f movie-gateway.yaml
gateway.networking.istio.io/movie-gateway created
virtualservice.networking.istio.io/movie created
```

#查询istio-ingressgateway入口的端口信息

```
kubectl -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}'
[root@node1 movie]# kubectl -n istio-system get service istio-ingressgateway
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
istio-ingressgateway	LoadBalancer	10.97.58.190	<pending>	15020:31630/TCP,80:31876/TCP,443:31678/TCP,31400:30106/TCP,15443:30236/TCP	3h17m

#其中31876就是映射到80端口, 也就是入口端口

测试: <http://192.168.31.106:31876/index?id=10574622>

豆瓣电影

搜索电影、电视剧、综艺、影人

192.168.31.106:31876/index?id=10574622

豆瓣 读书 电影 音乐 同城 小组 阅读 FM 时间 豆品 更多

下载豆瓣客户端 登录/注册

豆瓣 2019 年度电影榜单

影讯&购票 选电影 电视剧 排行榜 分类 影评 2019年度榜单 2019书影音报告

No.67 豆瓣电影Top250

人再囧途之泰囧 (2012)

导演: 徐峥 / 编剧: 徐峥 / 主演: 徐峥 / 陶虹 / 王宝强 / 黄渤 / 类型: 喜剧 / 制片国家/地区: 中国大陆 / 语言: 汉语普通话 / 英语 / 泰语 / 上映日期: 2012-12-12(中国大陆) / 片长: 105分钟 / 又名: 泰囧 / 人在囧途2 / Lost in Thailand

黑马评分 8.0 ★★★★★ 4人评价

在哪儿看这部电影

腾讯视频 VIP免费观看
1905电影网 免费观看
哔哩哔哩 免费观看

豆瓣成员常用的标签

黑色幽默 中国 喜剧 人性 剧情
搞笑 内地 2010

以下豆列推荐 (全部)

★豆瓣高分电影榜★ (上) 9.7-8.6分 (影志)
上座黑色幽默 (内正)
高智商乱斗电影 (中间元素)
中文电影250强总榜 (1905—2012) (东方快车)
同时入选IMDB250和豆瓣电影250的电影 (东方快车)

想看 看过 评价: ☆☆☆☆☆
写短评 写影评 分享到 推荐

人再囧途之泰囧的剧情介绍

商业成功人士徐朗(徐峥 饰)用了五年时间发明了一种叫“油霸”的神奇产品——每次汽车加油只需加到三分之二，再滴入2滴“油霸”，油箱的汽油就会变成满满一箱。徐朗的同学，兼商业竞争对手高博(黄渤 饰)想把这个发明一次性卖给法国人。但徐朗坚决不同意，他希望深入开发研究，把“油霸”发扬光大，得到更远的收益。两个人各抒己见，争论不休，一直无果。由于两人身份相同，唯有得到公司最大股东周扬的授权书，方可达到各自目的。当得知周扬在泰国后，徐朗立刻启程寻找。而高博获悉后将一枚跟踪器放在徐朗身上一起去了泰国。飞机上，徐朗遇到了王宝(王宝强 饰)，别有心机地想利用他来摆脱对手高博的追赶，可他不仅没甩掉王宝，还成了他的“贴身保姆”……

#设置默认的网络规则

```
[root@node1 movie]# kubectl apply -f destination-rule-all.yaml
destinationrule.networking.istio.io/movie-web created
destinationrule.networking.istio.io/movie-info created
destinationrule.networking.istio.io/movie-recommend created
destinationrule.networking.istio.io/movie-rating created
destinationrule.networking.istio.io/movie-web unchanged
```

#下面进行测试，反复刷新几次，可以看到评分部分，红/黑 进行切换。

#如果需要将流量全部导向v2，则可以执行下面这个规则

```
[root@node1 movie]# kubectl apply -f virtual-service-rating-v2.yaml
virtualservice.networking.istio.io/movie-rating created
```

#可以看到无论怎么刷新，都是现实红色，v1版本的rating就可以下线了。

#如果不需要此规则，可以将其删除

```
[root@node1 movie]# kubectl delete -f virtual-service-rating-v2.yaml
virtualservice.networking.istio.io "movie-rating" deleted
```

#如果需要进行实例的扩容，可以使用k8s进行扩容处理，同样也是受Istio管理

```
kubectl scale deployment/movie-rating-deployment-v2 --replicas=3
```

#查询pod，发现movie-rating-deployment-v2已经扩展为3个实例

```
[root@node1 movie]# kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
movie-info-deployment-c8f455f69-4hknf	2/2	Running	0	30m
movie-info-deployment-c8f455f69-psbb4	2/2	Running	0	30m
movie-rating-deployment-v1-667fb7c8cc-j8ggj	2/2	Running	0	30m

movie-rating-deployment-v2-577d48b98f-d46jj	2/2	Running	0	42s
movie-rating-deployment-v2-577d48b98f-fg2nj	2/2	Running	0	30m
movie-rating-deployment-v2-577d48b98f-kbmbv	2/2	Running	0	42s
movie-recommend-deployment-747d6dc646-8zv7j	2/2	Running	0	30m
movie-recommend-deployment-747d6dc646-g9bcb	2/2	Running	0	30m
movie-recommend-deployment-747d6dc646-sxbj7	2/2	Running	0	30m
movie-web-deployment-6996d99464-cc88t	2/2	Running	0	30m
movie-web-deployment-6996d99464-gjfm6	2/2	Running	0	30m
movie-web-deployment-6996d99464-t29tz	2/2	Running	0	30m
neo4j-deployment-6bb95596c8-sld6n	2/2	Running	0	30m