K8s 技术交流群　　和宽哥一对一交流

课程推荐，提供跪舔式售后服务：

K8s 高薪全栈架构师课程：https://edu.51cto.com/sd/518e5

K8s CKA 认证课程：https://edu.51cto.com/sd/fbbc8

K8s CKS 认证课程：https://edu.51cto.com/sd/affb5

K8s 全栈架构师+CKA 套餐：https://edu.51cto.com/topic/4973.html

超级套购：

K8s 全栈架构师+CKA+CKS：https://edu.51cto.com/topic/5174.html

关注宽哥得永生：

https://edu.51cto.com/lecturer/11062970.html?type=2

第一章　Istio 入门实践

1.1 Istio 安装

版本选择： https://istio.io/latest/docs/releases/supported-releases/#support-status-of-istio-releases

1.1.1 使用 Operator 部署 Istio

首先下载 Istio 的安装包：

```
# wget https://github.com/istio/istio/releases/download/1.13.0/istio-1.13.0-linux-amd64.tar.gz
```

解压后，将 Istio 的客户端工具 istioctl，移动到/usr/local/bin 目录下：

```
# tar xf istio-1.13.0-linux-amd64.tar.gz
# cd istio-1.13.0
# mv bin/istioctl /usr/local/bin/
# istioctl version
no running Istio pods in "istio-system"
1.13.0
```

接下来安装 Istio 的 Operator，可以使用 istioctl 一键部署：

```
# istioctl operator init
Installing operator controller in namespace: istio-operator using image: istio/operator:1.13.0
Operator controller will watch namespaces: istio-system
✓ Istio operator installed
✓ Installation complete
```

出现 Installation complete 后，查看 Pod 是否正常：

```
# kubectl  get po -n istio-operator
NAME                               READY   STATUS    RESTARTS   AGE
istio-operator-7f546b959b-cq4c9   1/1     Running   0          116s
```

之后通过定义 IstioOperator 资源，在 Kubernetes 中安装 Istio：

```
# cat istio-operator.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  namespace: istio-system
  name: example-istiocontrolplane
spec:
  profile: default
  components: # 自定义组件配置
    ingressGateways: # 自定义 ingressGateway 配置
      - name: istio-ingressgateway
        enabled: true # 开启 ingressGateway
        k8s:          # 自定义 ingressGateway 的 Kubernetes 配置
          service:      #将 Service 类型改成 NodePort
            type: NodePort
            ports:
            - port: 15020
              nodePort: 30520
              name: status-port
            - port: 80  # 流量入口 80 端口映射到 NodePort 的 30080，之后通过节点 IP+30080 即可访问 Istio 服务
```

```
    nodePort: 30080
    name: http2
    targetPort: 8080
  - port: 443
    nodePort: 30443
    name: https
    targetPort: 8443
```

安装 Istio：

```
# istioctl manifest apply -f  istio-operator.yaml
This will install the Istio 1.13.0 default profile with ["Istio core"
"Istiod" "Ingress gateways"] components into the cluster. Proceed? (y/N) y
✔ Istio core installed
✔ Istiod installed
✔ Ingress gateways installed
✔ Installation complete
Thank you for installing Istio 1.11.  Please take a few minutes to tell
us about your install/upgrade experience!
https://forms.gle/kWULBRjUv7hHci7T6
```

查看创建的 Service 和 Pod：

```
# kubectl  get svc,po -n istio-system
NAME                          TYPE       CLUSTER-IP       EXTERNAL-IP
PORT(S)                                   AGE
service/istio-ingressgateway   NodePort    192.168.99.93    <none>
15020:30020/TCP,80:30080/TCP,443:30443/TCP   83s
service/istiod              ClusterIP   192.168.117.146   <none>
15010/TCP,15012/TCP,443/TCP,15014/TCP       5m42s

NAME                                    READY   STATUS    RESTARTS   AGE
pod/istio-ingressgateway-5684974946-vmw6v   1/1     Running   0
6m4s
pod/istiod-7859559dd-gwp6p                1/1     Running   0          6m42s
```

1.1.2 配置自动注入

修改 APIServer 的配置文件，添加 MutatingAdmissionWebhook,ValidatingAdmissionWebhook（如果 K8s 版本大于 1.16 默认已经开启）：

```
# vi /etc/kubernetes/manifests/kube-apiserver.yaml # 二进制安装方式需要找到
APIServer 的 Service 文件
    - --enable-admission-plugins=
MutatingAdmissionWebhook,ValidatingAdmissionWebhook # 本示例省略了其它配置项，读
者需要追加这两项即可
```

接下来创建一个测试的 Namespace，并添加一个 istio-injection=enabled 的标签，之后在该 Namespace 下创建的 Pod 就会被自动注入 Istio 的 Proxy。

创建 Namespace 并添加 Label：

```
# kubectl create ns istio-test
# kubectl label namespace istio-test istio-injection=enabled
```

切换目录至 istio 的安装包，然后创建测试应用，此时创建的 Pod 会被自动注入一个 istio-proxy 的容器：

```
# kubectl apply -f samples/sleep/sleep.yaml -n istio-test
service/sleep created
deployment.extensions/sleep created
```

查看部署的容器：

```
    # kubectl get po -n istio-test
    NAME                    READY   STATUS    RESTARTS   AGE
    sleep-86cf99dfd6-h2nzh   2/2     Running   0          92s
```

1.1.3 可视化工具 Kiali

Kiali 为 Istio 提供了可视化的界面，可以在 Kiali 上进行观测流量的走向、调用链，同时还可以使用 Kiali 进行配置管理，给用户带来了很好的体验。

接下来在 Kubernetes 中安装 Kiali 工具，首先进入到 Istio 的安装包目录：

```
    # kubectl create -f samples/addons/kiali.yaml
    serviceaccount/kiali created
    configmap/kiali created
    clusterrole.rbac.authorization.k8s.io/kiali-viewer created
    clusterrole.rbac.authorization.k8s.io/kiali created
    clusterrolebinding.rbac.authorization.k8s.io/kiali created
    role.rbac.authorization.k8s.io/kiali-controlplane created
    rolebinding.rbac.authorization.k8s.io/kiali-controlplane created
    service/kiali created
    deployment.apps/kiali created
```

查看部署状态：

```
    # kubectl  get po,svc -n istio-system -l app=kiali
    NAME                         READY   STATUS    RESTARTS   AGE
    pod/kiali-fd9f88575-zbphq   1/1     Running   0          9m39s

    NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
    service/kiali   ClusterIP   192.168.55.181   <none>
20001/TCP,9090/TCP   8m40s
```

之后可以将 Service 类型改成 NodePort，或者配置 Ingress 即可访问 Kiali 服务：

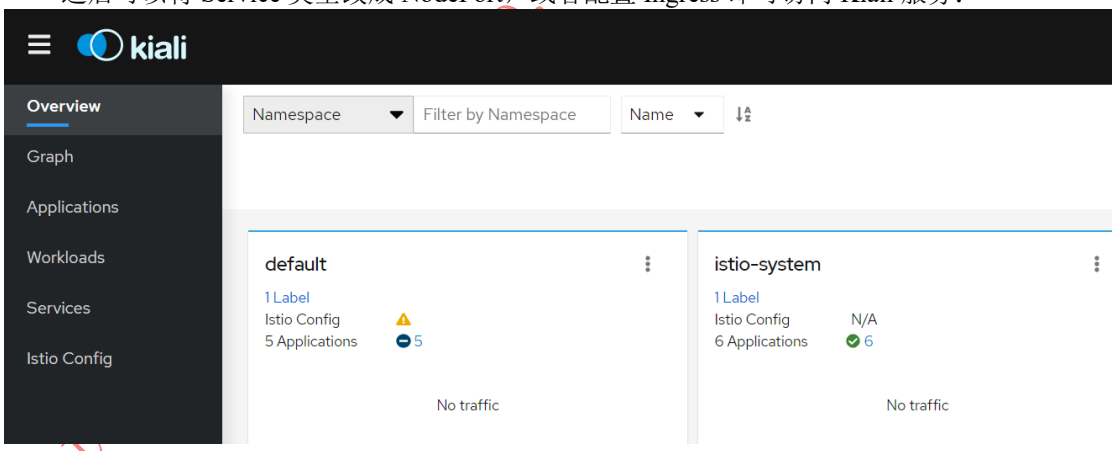

图 1.1-1 Kiali 图

在 Graph 页面可以看到应用的流量等信息（暂时没有可展示的项目，后面小结安装测试项目后即可看到完整的图形）：

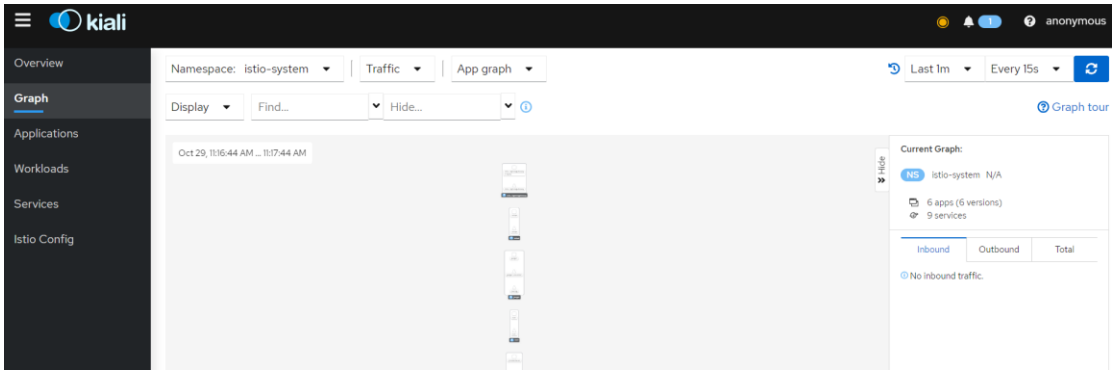

图 1.1-2 Kiali Graph

Istio Config 可以看到当前的 Istio 配置，可以查看配置详情以及更新配置：

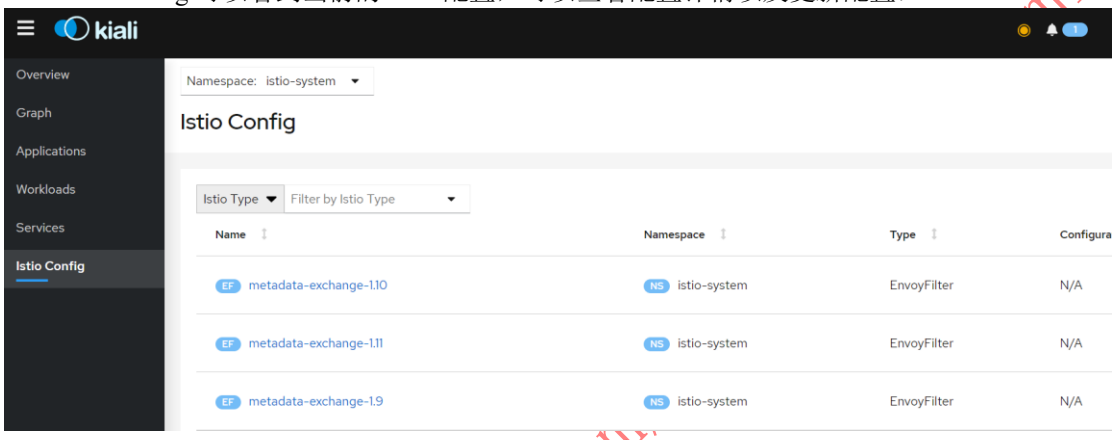

图 1.1-3 Istio 配置

除了 Kiali 之外，还需要一个链路追踪的工具，安装该工具可以在 Kiali 的 Workloads 页面，查看某个服务的 Traces 信息。直接安装即可：

```
    # kubectl create -f samples/addons/jaeger.yaml
    deployment.apps/jaeger created
    service/tracing created
    service/zipkin created
    service/jaeger-collector created
```

1.1.4 Prometheus 和 Grafana

Istio 默认暴露了很多监控指标，比如请求数量统计、请求持续时间以及 Service 和工作负载的指标，这些指标可以使用 Prometheus 进行收集，Grafana 进行展示。

Istio 内置了 Prometheus 和 Grafana 的安装文件，直接安装即可（也可以使用外置的 Prometheus 和 Grafana）：

```
    # kubectl create -f samples/addons/prometheus.yaml -f
samples/addons/grafana.yaml
    serviceaccount/prometheus created
    configmap/prometheus created
    clusterrole.rbac.authorization.k8s.io/prometheus created
    clusterrolebinding.rbac.authorization.k8s.io/prometheus created
    service/prometheus created
    deployment.apps/prometheus created
    serviceaccount/grafana created
    configmap/grafana created
    service/grafana created
    deployment.apps/grafana created
    configmap/istio-grafana-dashboards created
```

```
    configmap/istio-services-grafana-dashboards created
```

查看创建的 Pod 和 Service：

```
    # kubectl get svc,pod -n istio-system
    NAME                           TYPE       CLUSTER-IP       EXTERNAL-IP
PORT(S)                                AGE
    service/grafana                ClusterIP   192.168.58.97     <none>
3000/TCP                               36m
    service/istio-ingressgateway    NodePort    192.168.99.93     <none>
15020:30020/TCP,80:30080/TCP,443:30443/TCP   42h
    service/istiod                 ClusterIP   192.168.117.146   <none>
15010/TCP,15012/TCP,443/TCP,15014/TCP       42h
    service/jaeger-collector       ClusterIP   192.168.174.239   <none>
14268/TCP,14250/TCP,9411/TCP               36m
    service/kiali                  NodePort    192.168.55.181    <none>
20001:31600/TCP,9090:31235/TCP             41m
    service/knative-local-gateway   ClusterIP   192.168.93.127    <none>
80/TCP                                 41h
    service/prometheus             ClusterIP   192.168.168.212   <none>
9090/TCP                               36m
    service/tracing                ClusterIP   192.168.155.244   <none>
80/TCP,16685/TCP                        36m
    service/zipkin                 ClusterIP   192.168.114.253   <none>
9411/TCP                               36m

    NAME                                    READY   STATUS    RESTARTS   AGE
    pod/grafana-68cc7d6d78-tpx74            1/1     Running   0          37m
    pod/istio-ingressgateway-5684974946-vmw6v   1/1     Running   0          42h
    pod/istiod-7859559dd-gwp6p              1/1     Running   0          42h
    pod/jaeger-5d44bc5c5d-9wwj7             1/1     Running   0          37m
    pod/kiali-fd9f88575-zbphq               1/1     Running   0          42m
    pod/prometheus-77b49cb997-zkrm2          2/2     Running   0          37m
```

同样的方式，将 Grafana 的 Service 改成 NodePort 或者添加 Ingress，之后访问即可：

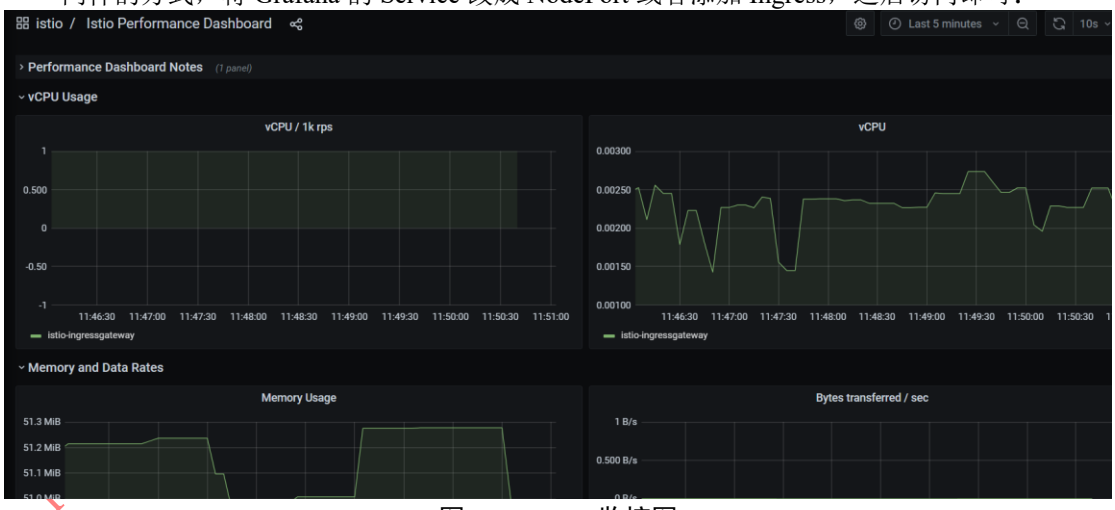

图 1.1-4 Isito 监控图

1.2 Istio 流量治理实践

1.2.1 部署测试用例

首先在 Kubernetes 集群中，部署 Bookinfo 项目。

Istio 提供了用于测试功能的应用程序 Bookinfo，可以直接通过下述命令创建 Bookinfo：

```
# kubectl create ns bookinfo
namespace/bookinfo created
# kubectl label ns bookinfo istio-injection=enabled
namespace/bookinfo labeled
# kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml -n
bookinfo
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
deployment.apps/productpage-v1 created
```

查看部署的 Pod 和 Service：

```
# kubectl get po -n bookinfo
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-79f774bdb9-6scbp       2/2     Running   0          2m55s
productpage-v1-6b746f74dc-6qn4j   2/2     Running   0          2m55s
ratings-v1-b6994bb9-2kbln         2/2     Running   0          2m55s
reviews-v1-545db77b95-h87qb       2/2     Running   0          2m55s
reviews-v2-7bf8c9648f-2k2t4       2/2     Running   0          2m55s
reviews-v3-84779c7bbc-62l5z       2/2     Running   0          2m55s

# kubectl get svc -n bookinfo
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
details       ClusterIP   192.168.150.9    <none>        9080/TCP   12m
productpage   ClusterIP   192.168.99.245   <none>        9080/TCP   12m
ratings       ClusterIP   192.168.116.254  <none>        9080/TCP   12m
reviews       ClusterIP   192.168.72.11    <none>        9080/TCP   12m
```

之后可以通过 productpage Service 的 ClusterIP 访问 Bookinfo 项目：

```
# curl 192.168.99.245:9080
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Bookstore App</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
...
```

接下来创建 Istio 的 Gateway 和 VirtualService 实现域名访问 Bookinfo 项目。

首先创建 Gateway，假设域名是 bookinfo.kubeasy.com。Gateway 配置如下所示：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # 使用默认的 istio ingress gateway
  servers:
  - port:
      number: 80
      name: http
```

```
      protocol: HTTP
    hosts:
    - "*" # 发布域名
```

接下来配置 VirtualService，实现对不同微服务的路由：

```
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "bookinfo.kubeasy.com"
  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
```

官方示例已经提供了该文件，修改 Gateway 和 VirtualService 的 hosts 为 bookinfo.kubeasy.com，然后创建即可：

```
# grep "hosts" -A 1 samples/bookinfo/networking/bookinfo-gateway.yaml
    hosts:
    - "bookinfo.kubeasy.com"
--
  hosts:
  - "bookinfo.kubeasy.com"
# 创建资源
# kubectl create -f samples/bookinfo/networking/bookinfo-gateway.yaml -n bookinfo
gateway.networking.istio.io/bookinfo-gateway created
virtualservice.networking.istio.io/bookinfo created
```

查看资源：

```
# kubectl  get gw,vs -n bookinfo
NAME                                          AGE
gateway.networking.istio.io/bookinfo-gateway   42s

NAME                                        GATEWAYS             HOSTS
AGE
virtualservice.networking.istio.io/bookinfo   ["bookinfo-gateway"]
["bookinfo.kubeasy.com"]   42s
```

接下来将域名 bookinfo.kubeasy.com 解析至集群任意一个安装了 kube-proxy 的节点 IP 上，然后通过 ingressgateway 的 Service 的 NodePort 即可访问到 Bookinfo：

```
# kubectl get svc -n istio-system istio-ingressgateway
NAME                   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
```

```
    istio-ingressgateway   NodePort   192.168.99.93   <none>
15020:30020/TCP,80:30080/TCP,443:30443/TCP   21d
```

绑定 hosts 后，通过 bookinfo.kubeasy.com+ingressgateway 80 端口的 NodePort 即可访问该服务，比如本次示例的 bookinfo.kubeasy.com:30080/productpage：

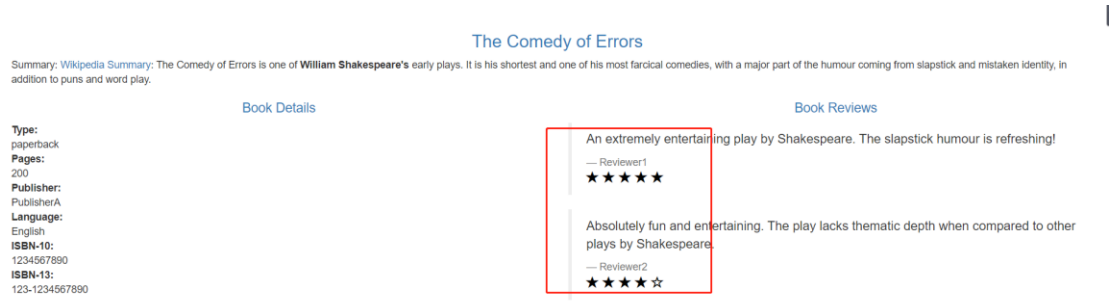

图 1.2-1 访问 Bookinfo

多次刷新可以看到 Reviewer 处的星星会在黑色、红色和消失之间来回替换，是因为之前部署了三个不同版本的 reviews，每个版本具有不同的显示效果。

此时通过 Kiali 页面可以看到 Bookinfo 的调用链路：

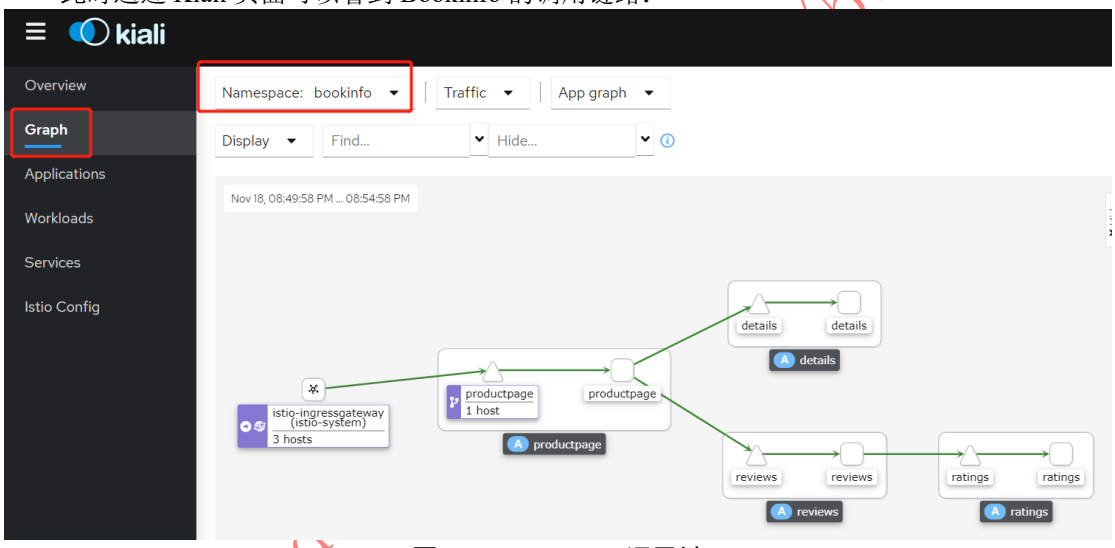

图 1.2-2 Bookinfo 调用链

1.2.2 Istio 实现灰度部署

首先将 reviews 的所有流量指向 v1 版本，此时需要通过 DestinationRule 将 reviews 分成三个版本：

```
# vim reviews-dr.yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1 # subset v1 指向具有 version=v1 的 Pod
  - name: v2
    labels:
```

```
      version: v2 # subset v2 指向具有 version=v2 的 Pod
  - name: v3
    labels:
      version: v3 # subset v3 指向具有 version=v3 的 Pod
```

创建并查看该 DestinationRule：

```
# kubectl create -f reviews-dr.yaml -n bookinfo
destinationrule.networking.istio.io/reviews created
# kubectl get dr -n bookinfo
NAME      HOST      AGE
reviews   reviews   9s
```

此时只是创建了 DestinationRule，并没有做任何限制，所以通过浏览器访问该服务时，并没有什么变化。但是 Kiali 中可以看到 reviews 变成了三个版本：

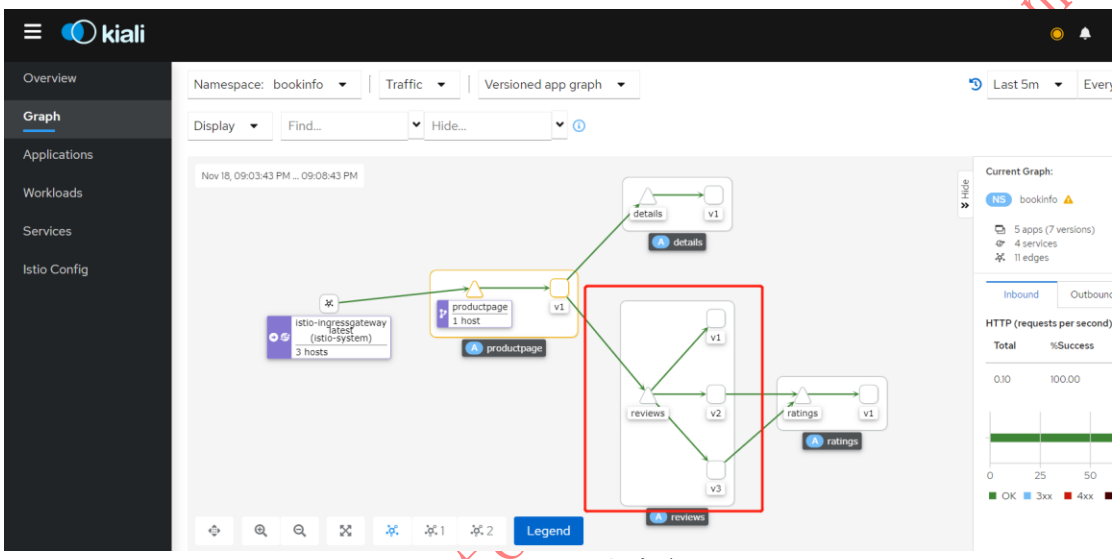

图 1.2-8 reviews 版本流量

接下来配置 VirtualService 将所有流量指向 reviews 的 v1 版本：

```
# vim reviews-v1-all.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1 # 将流量指向 v1
# 创建该 VirtualService
# kubectl create -f reviews-v1-all.yaml -n bookinfo
virtualservice.networking.istio.io/reviews created
```

此时再次刷新浏览器，Reviews 处不再显示评分：

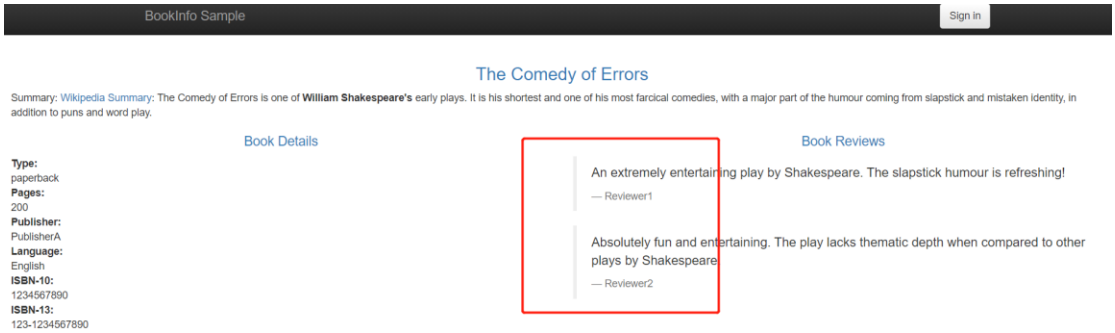

图 1.2-4 屏蔽 reviews

同时 Kiali 也不再显示 reviews 服务调用 ratings 服务：

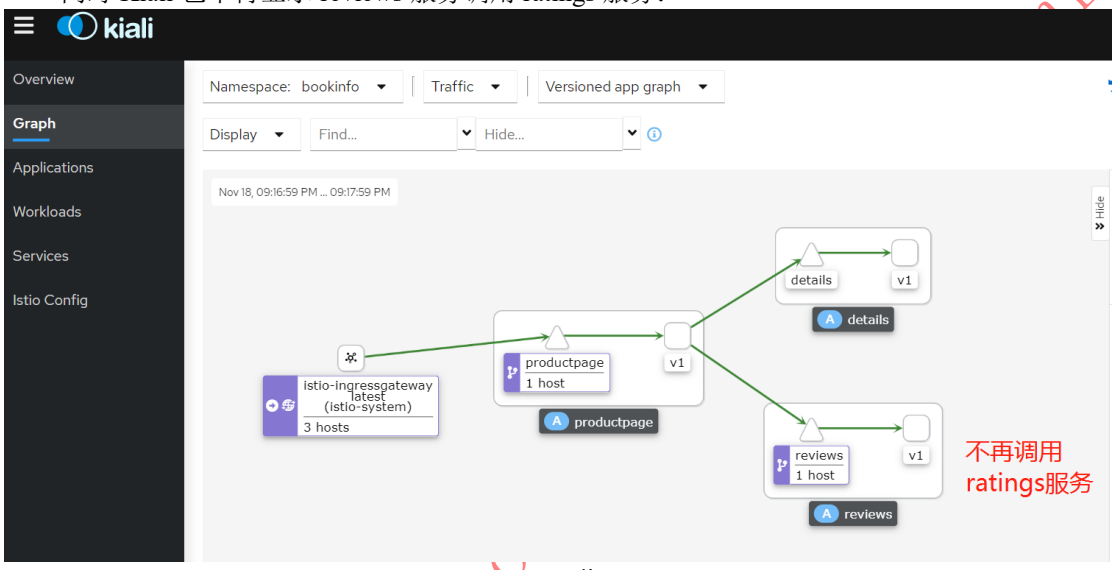

图 1.2-5 屏蔽 ratings

接下修改 VirtualService，将 20%的流量导向 v2 版本：

```
# vim reviews-20v2-80v1.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1 # 将 80%流量指向 v1
      weight: 80 # 只需要配置一个 weight 参数即可
    - destination:
        host: reviews
        subset: v2 # 将 20%流量指向 v2
      weight: 20
```

由于之前已经创建了 VirtualService，此处进行 replace 即可：

```
# kubectl replace -f reviews-20v2-80v1.yaml  -n bookinfo
virtualservice.networking.istio.io/reviews replaced
```

再次使用浏览器访问时，会有 20%的访问会出现评分（仅有黑色）：

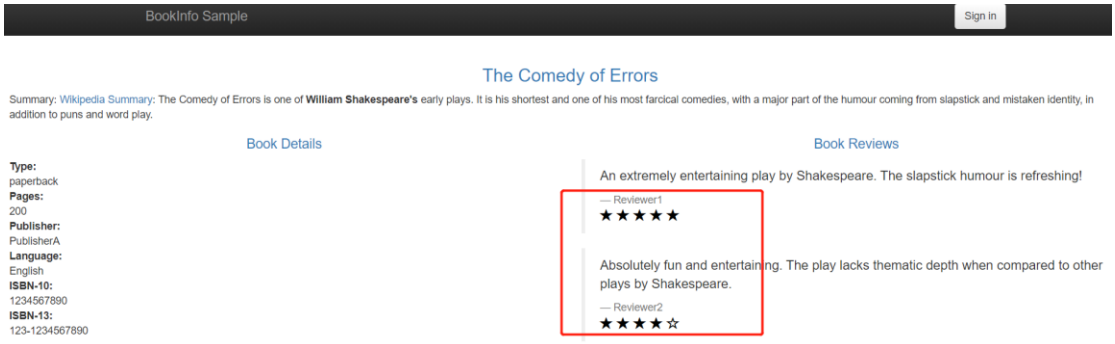

图 1.2-6 review v2 灰度

此时从 Kiali 页面即可看到 v2 版本的 review 调用 ratings：

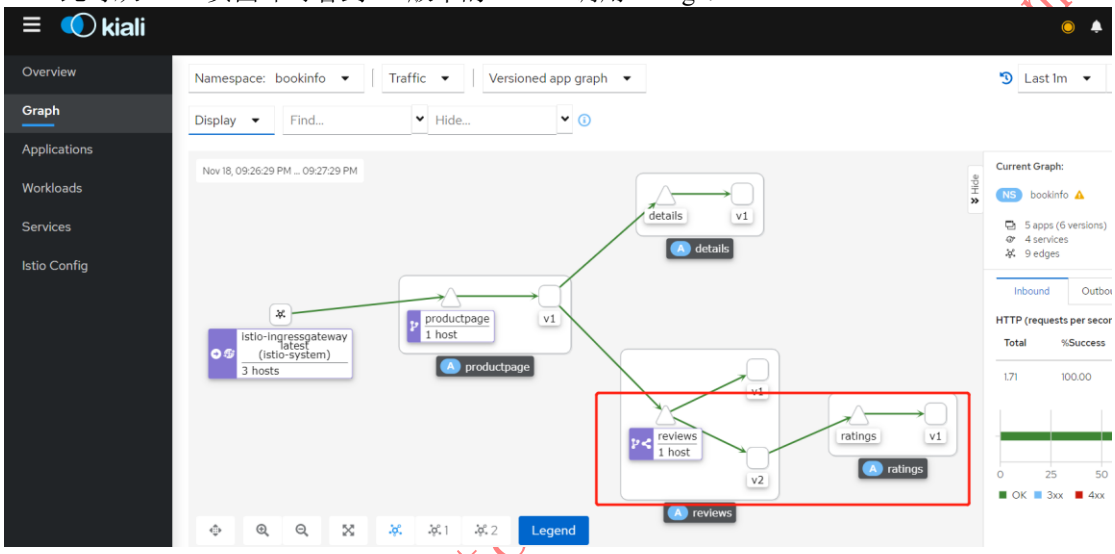

图 1.2-7 review 调用 ratings

假设 v2 版本已经没有任何问题，可以将流量全部指向 v2 版本：

```
# vim reviews-v2-all.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2 # 指向 v2
# kubectl replace -f reviews-v2-all.yaml -n bookinfo
virtualservice.networking.istio.io/reviews replaced
```

Kiali 只有 v2 的调用链：

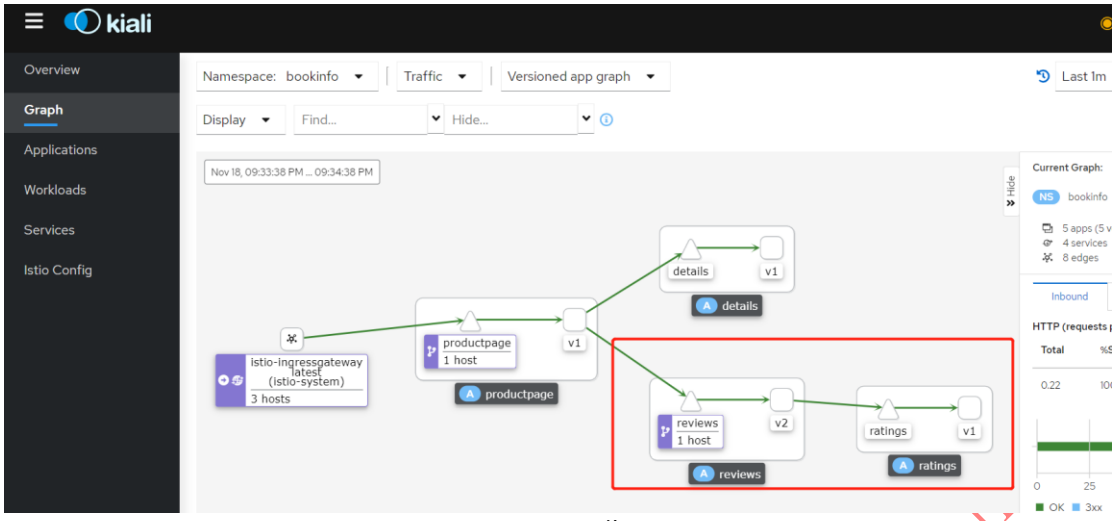

图 1.2-8 屏蔽 v1

1.2.3 Istio 实现 AB 测试

请求头匹配：https://istio.io/latest/docs/reference/config/networking/virtual-service/#HTTPMatchRequest

再次修改 reviews 的 VirtualService，将 jason 用户指向 v3，其他用户依旧使用 v2 版本：

```
# cat reviews-jasonv3.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers: # 匹配请求头
        end-user: # 匹配请求头的 key 为 end-user
          exact: jason # value 为 jason
    route:
    - destination:
        host: reviews
        subset: v3 # 匹配到 end-user=jason 路由至 v3 版本
  - route:
    - destination:
        host: reviews
        subset: v2 # 其余的路由至 v2 版本
# 更新该 VirtualService
# kubectl replace -f reviews-jasonv3.yaml  -n bookinfo
virtualservice.networking.istio.io/reviews replaced
```

接下来首先使用未登录用户的方式访问 bookinfo：

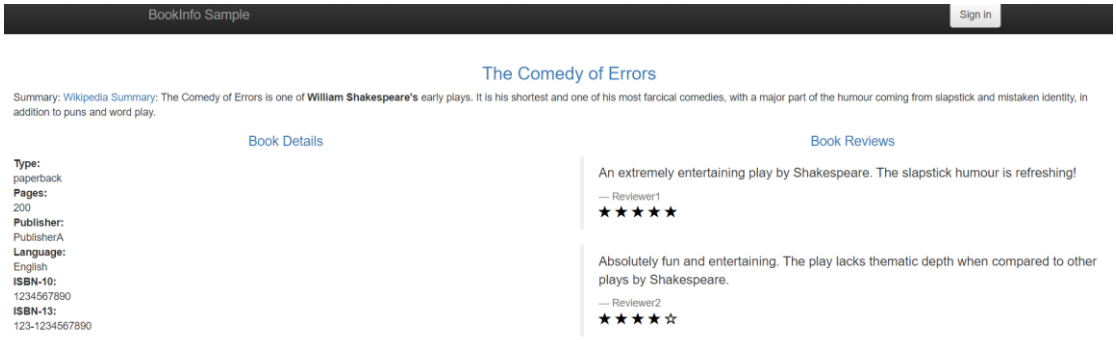

图 1.2-9 未登录用户测试

可以看到页面只显示黑色五角星。接下来登录 jason 用户（密码不限）：

图 1.2-10 登录 jason

登录后页面评分只显示红色：

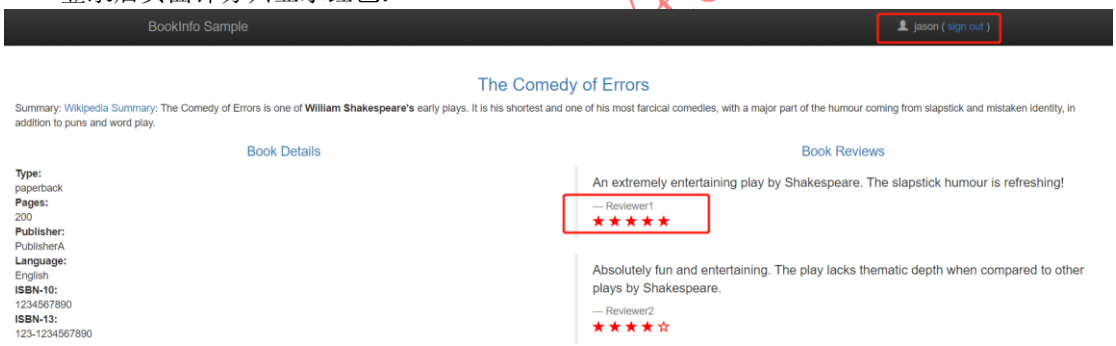

图 1.2-11 登录用户测试

1.2.4 Istio 地址重写和重定向

地址重定向：https://istio.io/latest/docs/reference/config/networking/virtual-service/#HTTPRedirect

比如将 bookinfo.kubeasy.com/gx，跳转到 edu.51cto.com/lecturer/11062970.html：

```
# kubectl  edit vs bookinfo -n bookinfo
- match:
  - uri:
      prefix: /gx # 匹配/gx
  redirect:
    authority: edu.51cto.com  # 跳转的域名
    uri: /lecturer/11062970.html  # 跳转的路径
```

通过浏览器访问 bookinfo.kubeasy.com/gx（如果是学习环境需要加上端口号）即可跳转到 edu.51cto.com/lecturer/11062970.html。当然也可以用 curl 进行测试：

```
# curl  -H"Host:bookinfo.kubeasy.com"
```

```
CHANGE_HERE_FOR_YOUR_NODE_IP:30080/gx -I
    HTTP/1.1 301 Moved Permanently
    location: http://ke.qq.com/course/2738602
    ...
    transfer-encoding: chunked
```

Istio 地址重写配置方式和重定向类似，假如将“/”重写为/productpage，配置如下：

```
    # kubectl  edit vs bookinfo -n bookinfo
      - match:
        - uri:
            exact: / # 匹配根路径
        rewrite:
          uri: /productpage # 重写为/productpage
        route:
        - destination:
            host: productpage
            port:
              number: 9080
```

保存退出后，访问 bookinfo.kubeasy.com 即可打开 bookinfo 的页面（未配置地址重写前，访问根路径会限制 404 错误）。

1.2.5 Istio 负载均衡算法

Istio 原生支持多种负载均衡算法，比如 ROUND_ROBIN、LEAST_CONN、RANDOM 等。假如一个应用存在多个副本（Pod），可以使用上述算法对多个 Pod 进行定制化的负载均衡配置。

每种负载均衡的策略如下：

- ROUND_ROBIN：默认，轮询算法，将请求依次分配给每一个实例；
- LEAST_CONN：最小连接数，随机选择两个健康实例，将请求分配成两个中连接数最少的那个；
- RANDOM：随机算法，将请求随机分配给其中一个实例；
- PASSTHROUGH：将连接转发到调用者请求的原始 IP 地址，而不进行任何形式的负载平衡，目前不推荐使用。

首先多次访问 bookinfo 的首页，之后观看 Kiali 中 reviews 的流量分配：

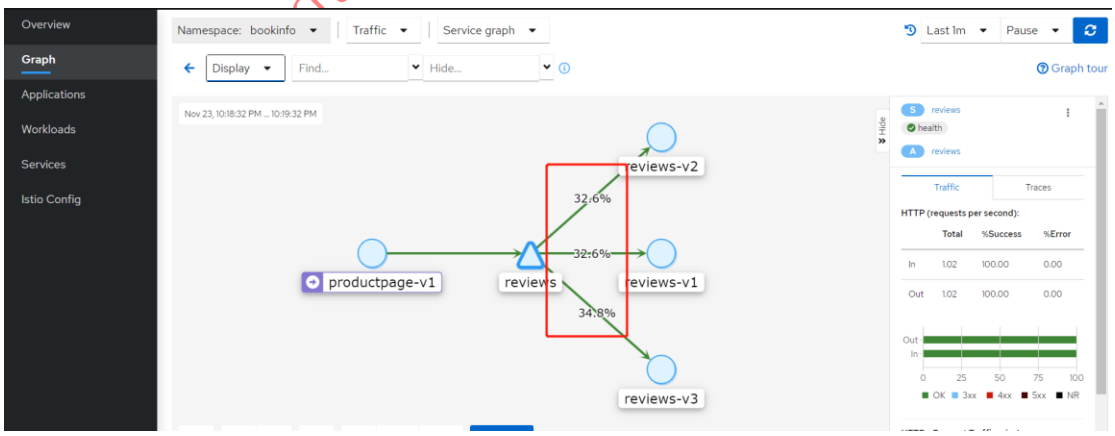

图 1.2-12 流量分配

可以看到此时的流量大概是 1:1:1，符合默认的 ROUND_ROBIN 算法。接下来将算法改成 RANDOM：

```
    # kubectl edit dr reviews -nbookinfo
    ...
    spec:
```

```
    trafficPolicy: # 添加路由策略，在 spec 下对所有的 subset 生效，也可以在 subset 中配置
      loadBalancer: # 配置负载均衡
        simple: RANDOM # 策略为 RANDOM
    host: reviews
    subsets:
    - labels:
        version: v1
      name: v1
    - labels:
        version: v2
      name: v2
    - labels:
        version: v3
      name: v3
```

保存退出后，再次访问：

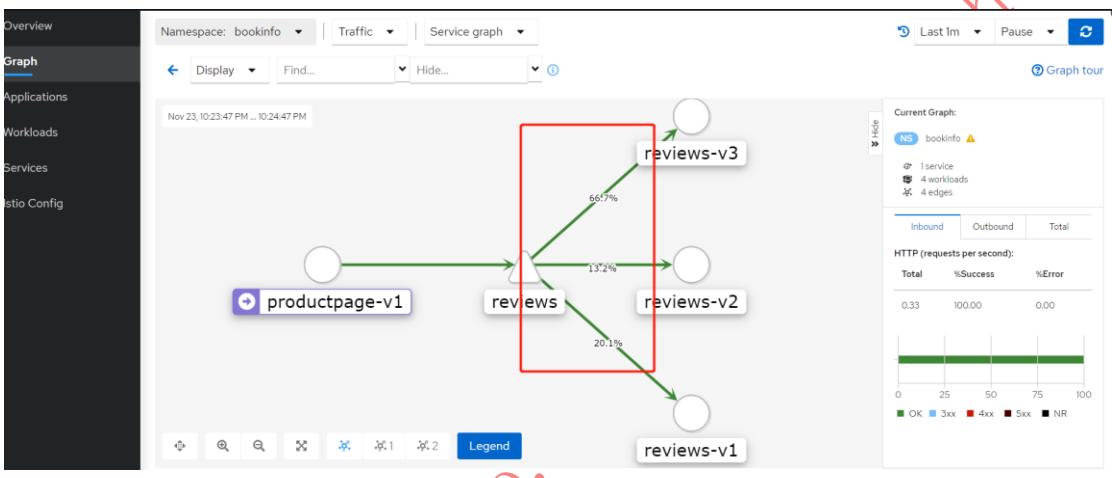

图 1.2-13 更改流量算法后流量分配

1.2.6 Istio 熔断

假设对 ratings 进行熔断，希望在并发请求数超过 3，并且存在 1 个以上的待处理请求，就触发熔断，此时可以配置 ratings 的 DestinationRule 如下所示：

```
# vim ratings-dr.yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: ratings
  namespace: bookinfo
spec:
  host: ratings
  trafficPolicy: # trafficPolicy 配置，可以配置在 subsets 级别
    connectionPool: # 连接池配置，可以单独使用限制程序的并发数
      tcp:
        maxConnections: 3 # 最大并发数为 3
      http:
        http1MaxPendingRequests: 1 # 最大的待处理请求
        maxRequestsPerConnection: 1 # 每个请求最大的链接数
    outlierDetection:  # 熔断探测配置
      consecutive5xxErrors: 1  # 如果连续出现的错误超过 1 次，就会被熔断
      interval: 10s  # 每 10 秒探测一次后端实例
```

```
        baseEjectionTime: 3m  # 熔断的时间
        maxEjectionPercent: 100 # 被熔断实例最大的百分比
    subsets:
    - labels:
        version: v1
      name: v1
```

保存退出后，部署测试工具 fortio，用于对容器业务进行压力测试：

```
# kubectl -n bookinfo apply -f samples/httpbin/sample-client/fortio-deploy.yaml
service/fortio created
deployment.apps/fortio-deploy created
```

等待 fortio 容器启动后，获取 fortio 容器 id：

```
# FORTIO_POD=$(kubectl get pod -n bookinfo | grep fortio | awk '{ print $1 }')
# echo $FORTIO_POD
fortio-deploy-75c4fbd7f9-5qp9j
```

发送一个请求，可以看到当前状态码为 200，即连接成功：

```
# kubectl exec -ti $FORTIO_POD -n bookinfo -- fortio load -curl http://ratings:9080/ratings/0
HTTP/1.1 200 OK
...
30
{"id":0,"ratings":{"Reviewer1":5,"Reviewer2":4}}
0
```

接下来更改为两个并发连接（-c 2），发送 20 请求（-n 20）：

```
# kubectl exec -ti $FORTIO_POD -n bookinfo -- fortio load -c 2 -qps 0 -n 20 -loglevel Warning  http://ratings:9080/ratings/0 | grep Code
Code 200 : 15 (75.0 %)
Code 503 : 5 (25.0 %)
```

可以看到 503 的结果为 5，说明触发了熔断。提高并发量，可以看到故障率更高：

```
# kubectl exec -ti $FORTIO_POD -n bookinfo -- fortio load -c 10 -qps 0 -n 20 -loglevel Warning  http://ratings:9080/ratings/0 | grep Code
Code 200 : 3 (15.0 %)
Code 503 : 17 (85.0 %)
```

最后可以查看 fortio 请求记录（upstream_rq_pending_overflow 表示熔断的次数）：

```
# kubectl -n bookinfo exec -it $FORTIO_POD  -c istio-proxy  -- sh -c 'curl localhost:15000/stats' | grep ratings | grep pending_overflow
cluster.outbound|9080|v1|ratings.bookinfo.svc.cluster.local.upstream_rq_pending_overflow: 0
cluster.outbound|9080||ratings.bookinfo.svc.cluster.local.upstream_rq_pending_overflow: 40
```

1.2.7 Istio 注入延迟故障

官方文档：https://istio.io/latest/docs/reference/config/networking/virtual-service/#HTTPFaultInjection

首先不添加任何延迟，看一下访问速度如何：

```
# 创建一个用于测试的工具
# kubectl run -ti -n bookinfo debug-tools --image=registry.cn-beijing.aliyuncs.com/dotbalo/debug-tools
```

```
If you don't see a command prompt, try pressing enter.
(09:30 debug-tools:/)
(09:31 debug-tools:/) time curl -I -s details:9080
...
real    0m0.007s
user    0m0.001s
sys 0m0.003s
```

不添加任何故障延迟时，0.007 秒左右就会返回结果。接下来注入一个 5s 的延迟：

```
# vim details-delay.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
spec:
  hosts:
  - details
  http:
  - fault: # 添加一个错误
      delay: # 添加类型为 delay 的故障
        percentage:  # 故障注入的百分比
          value: 100 # 对所有请求注入故障
        fixedDelay: 5s   # 注入的延迟时间
    route:
    - destination:
        host: details
# kubectl create -f details-delay.yaml -n bookinfo
virtualservice.networking.istio.io/details created
```

再次进行测试，返回时间已经达到了五秒：

```
(09:31 debug-tools:/) time curl -I -s details:9080
...
real    0m5.191s
user    0m0.003s
sys 0m0.003s
```

1.2.8 Istio 注入中断故障

中断故障注入只需要将 fault 的 delay 更改为 abort 即可：

```
# vim details-abort.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
spec:
  hosts:
  - details
  http:
  - fault:
      abort: # 更改为 abort 类型的故障
        percentage:
          value: 100
        httpStatus: 400 # 故障状态码
    route:
    - destination:
        host: details
```

替换该 VirtualService，再次访问测试：

```
# kubectl replace -f details-abort.yaml -n bookinfo
virtualservice.networking.istio.io/details replaced
# kubectl exec -ti debug-tools -n bookinfo -- bash
(09:40 debug-tools:/) curl details:9080 -I
```

```
HTTP/1.1 400 Bad Request # 直接返回 400 错误
content-length: 18
...
```

此时访问 Bookinfo 的页面会提示无法连接 Details 服务：

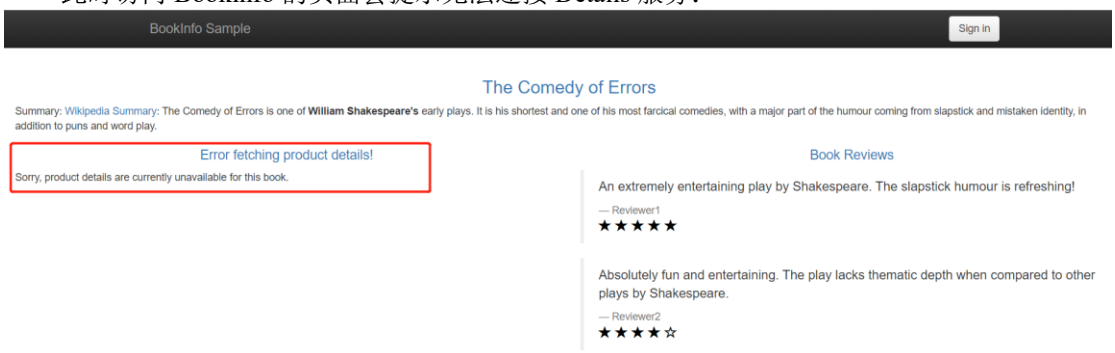

图 1.2-14 故障注入

1.2.9 Istio 快速超时配置

首先向 Ratings 服务注入一个 5 秒的延迟模拟 Ratings 服务响应比较慢：

```
# vim ratints-delay.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings-delay
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        percentage:
          value: 100
        fixedDelay: 5s
    route:
    - destination:
        host: ratings
# kubectl create -f ratints-delay.yaml -n bookinfo
virtualservice.networking.istio.io/ratings-delay created
```

访问测试：

```
# kubectl exec -ti debug-tools -n bookinfo -- bash
(10:24 debug-tools:/) time curl ratings:9080

real    0m5.074s
user    0m0.002s
sys 0m0.003s
```

此时在浏览器访问，可以看到整个页面都变得缓慢。接下来向 reviews 服务配置一个 1 秒超时：

```
# vim reviews-timeout.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
```

```
    http:
    -
      route:
      - destination:
          host: reviews
      timeout: 1s # 添加超时，时间为 1 秒
```