稀土掘金 CloudWeGo

Kitex在森马电商场景的落地实践

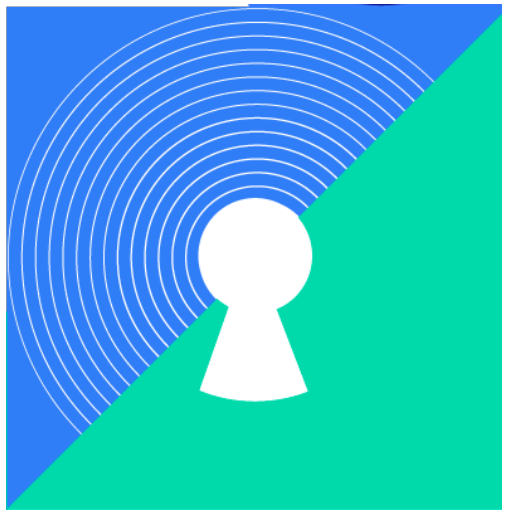
业务不断增长情况下，Kitex在微服务场景下的应用

梁东坡 森马电商开发工程师

CloudWeGo | 稀土掘金 出品

2022/06/25

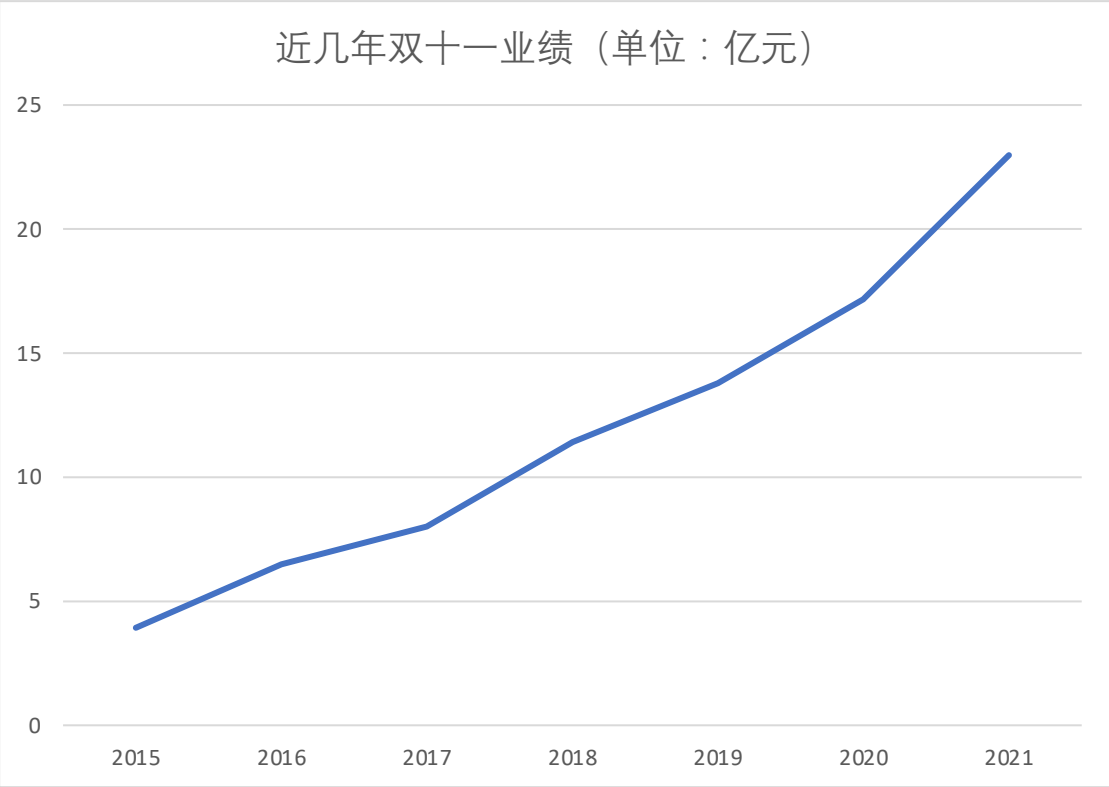
目录



订单流转中心-天枢

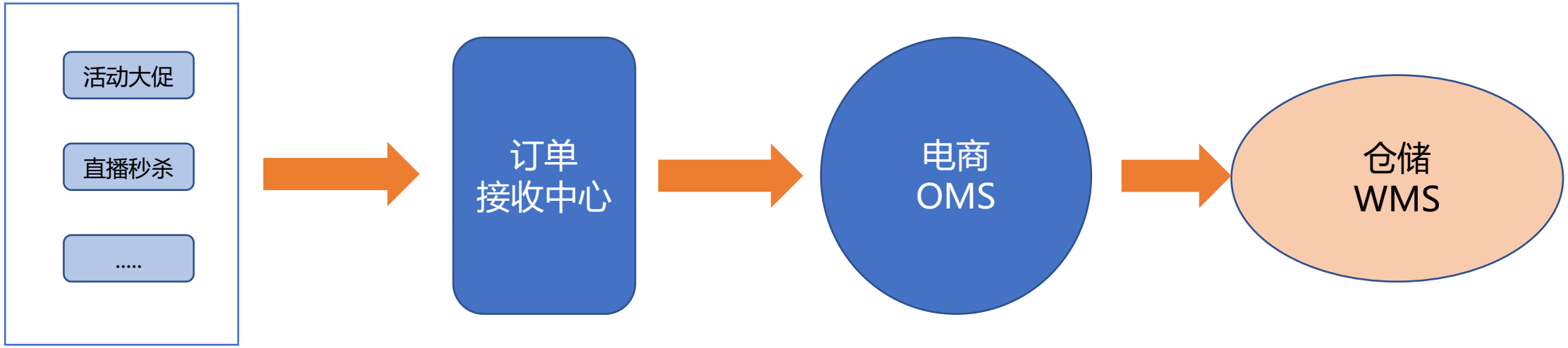
- 森马电商在运营的电商平台，如：天猫、抖店、京东、拼多多等大大小小的平台有**几十家**
- 由于每个平台的接口和对接的方式不统一，我们专门开发了这套系统，去**统一对接平台**，然后把数据处理成统一的格式下发到下游系统，如：OMS和WMS
- 该平台在电商活动，如618，双十一等订单峰值流量下发挥了**重要作用**

业务增长



- 随着业务的增长（2021年GMV超百亿），对系统的**性能**和**稳定性**要求越来越高
- 随着系统的规模增长，集群内的pod数量和Service不断的增加，对系统底层架构有很大的考验
- 目前迁移的平台有：有赞、抖音、拼多多、快手等，集群内的pod数已经超过200个，后续会接入京东、唯品会、天猫等平台后，pod数会成倍的增长，更需要一个**成熟**的系统架构作为支撑

面临的问题



当遇到活动如双十一，618大促，特别是直播时订单量短时间内，暴增的情况下

- 原有的系统架构无法支撑或者不能及时处理订单数据
- 影响发货及库存同步
- 间接产生不同类型的资损

技术挑战

高并发

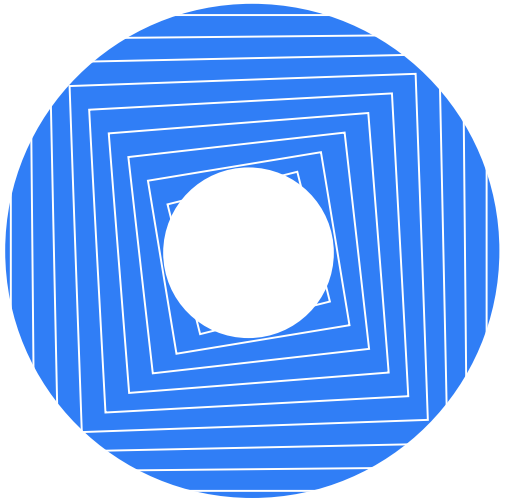
在电商业务场景下，不管是面向用户，如：秒杀，还是面向业务，如：订单处理，如果实现不了高并发，系统就很难做大，很难适应业务的增长

高性能

除了高并发来实现业务的快速处理外，性能也是一个挑战，例如在当前疫情状态下，各行各业都在降本增效解决不了高性能，就会不断的增加服务器资源，使成本大大增加

技术保障

我们电商行业的公司，大多资源和精力都在销售端，运营端，技术方面投入相对薄弱，这个时候，在技术选型上需要从：可靠、安全、支持等维度去考量



技术选型

- 开发语言
- 微服务框架
- 应用场景

如何选择

开发语言

语言没有好坏之分，只有相关场景下合适不合适

我们从性能、多线程、编译、效率等方面综合考虑，选择了Golang

众多微服务框架，该怎么选择

团队分别用了 Google开源的 的 gRPC 和 字节跳动开源的 CloudWeGo Kitex 做了技术评估和性能压测，经过专业的测试同学的压力测试，最终选择了 CloudWeGo Kitex

选择Kitex的原因

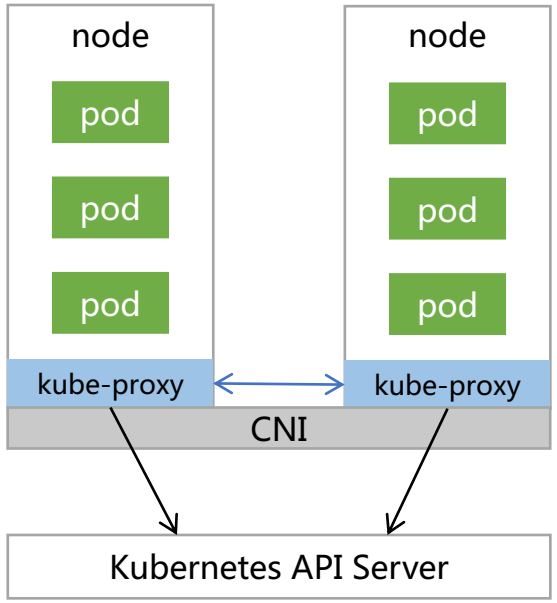
- 背后强大的技术团队和及时有效的技术支持
- 经过压力测试，性能优于其他微服务框架

关于微服务

服务注册中心，是选择常用的开源的注册中心（Zookeeper、Eureka、Nacos、Consul和ETCD）还是选择云原生的服务网格（istio）

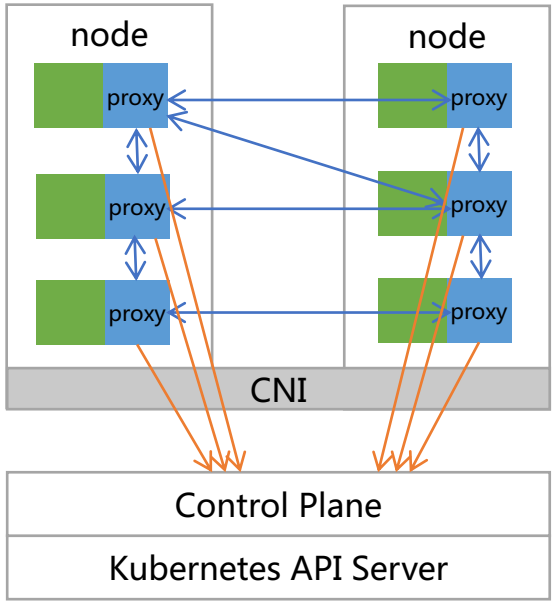
微服务集群的两种形式（以流量转发、服务注册和服务发现维度）

Kubernetes native



Kubernetes 集群中的每个节点都部署了一个 kube-proxy 组件，该组件与 Kubernetes API Server 进行通信

基于Istio的服务网格



- 通俗的讲：Istio**接管了**k8s的网络，通过 sidecar proxy 的方式将 Kubernetes 中的流量控制从服务层中抽离出来
- Istio 基于 Enovy 的 xDS 协议扩展了其**控制平面**
- 每个 pod 中放入原有的 kube-proxy 路由转发功能

两种形式对于我们开发者的区别

Kubernetes native

由于k8s 负载均衡不支持 RPC协（HTTP2）

故而需要额外的**服务注册中心**

我们常用的第三方服务注册中心：

- *zookeeper*
- *etcd*
- *consul*

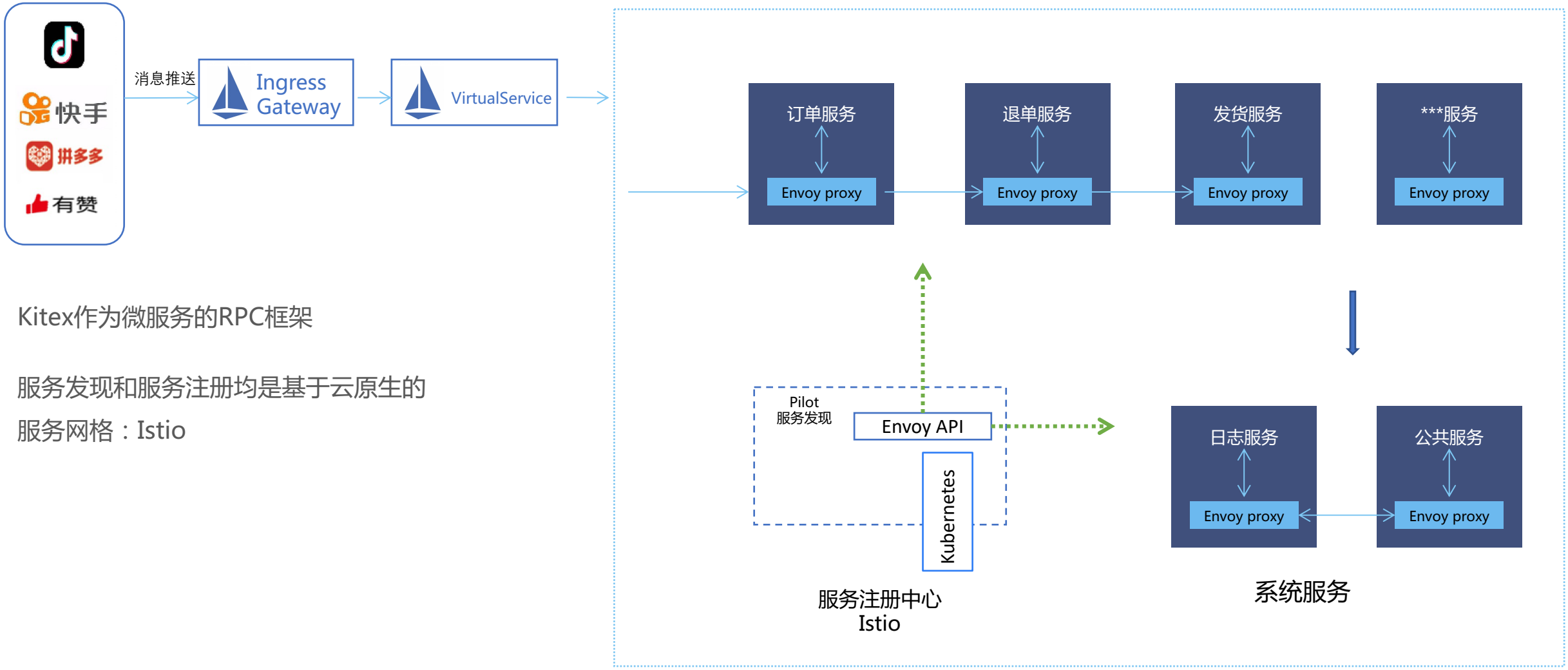
基于Istio服务网格

Istio 具备了：流量管理、策略控制、可观察性

将“应用程序”与“网络”**解耦**

我们不需要另外使用第三方注册中心

系统基本架构



Kitex作为微服务的RPC框架

服务发现和服务注册均是基于云原生的服务网格：Istio

Kitex 接入 istio

服务端

```
addr, _ := net.ResolveTCPAddr( network: "tcp",  address: ":9000")
svr := serverDouyin.NewServer(new(ServerDouyinImpl),server.WithServiceAddr(addr))
err := svr.Run()
if err != nil {
    log.SugarLogger.Error(err.Error())
}
```

客户端

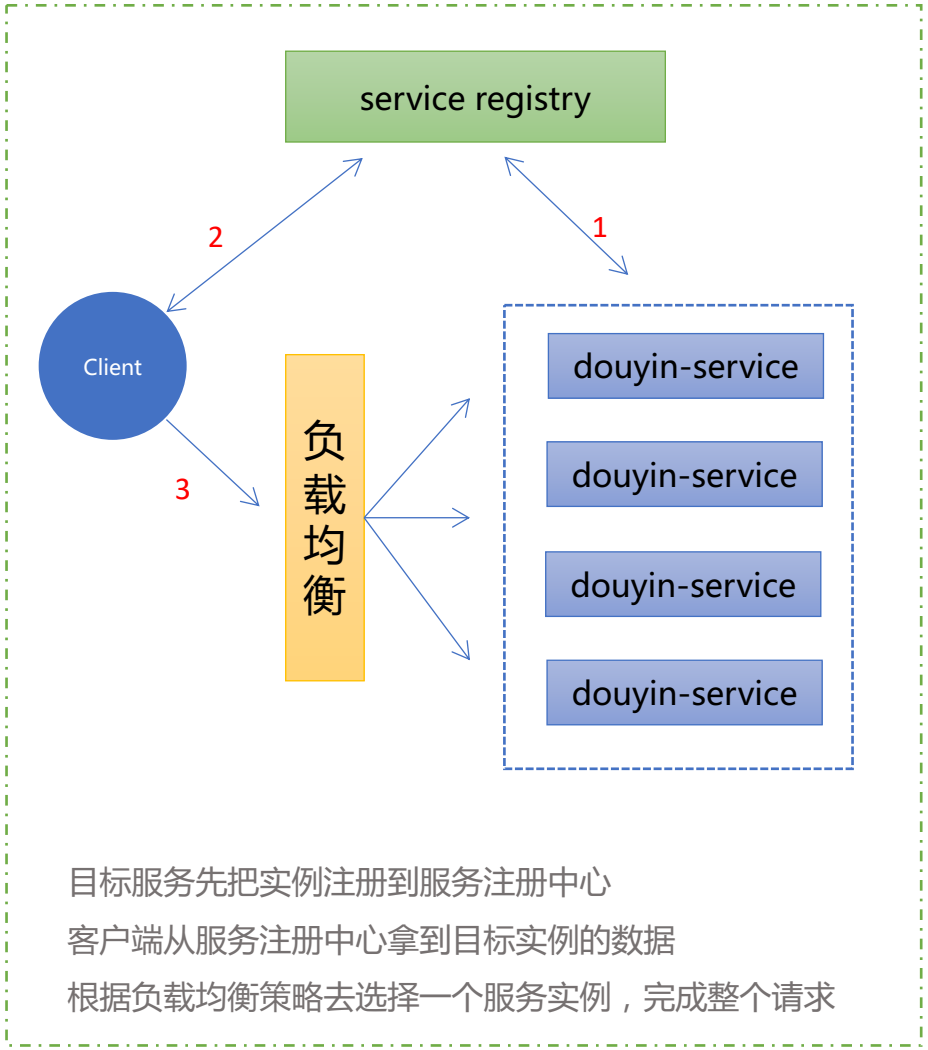
```
c,err := serverdouyin.NewClient( destService: "serverDouyin", client.WithHostPorts( hostports...: "server-host:9000"))

req := serverDouyin.GoodsRequest{
    PlatformCode: "57",
}
res,err := c.Goods(context.Background(), &req)
```

客户端的 server-host 要写实际集群中的内网地址

例如：server-douyin.default.svc.cluster.local

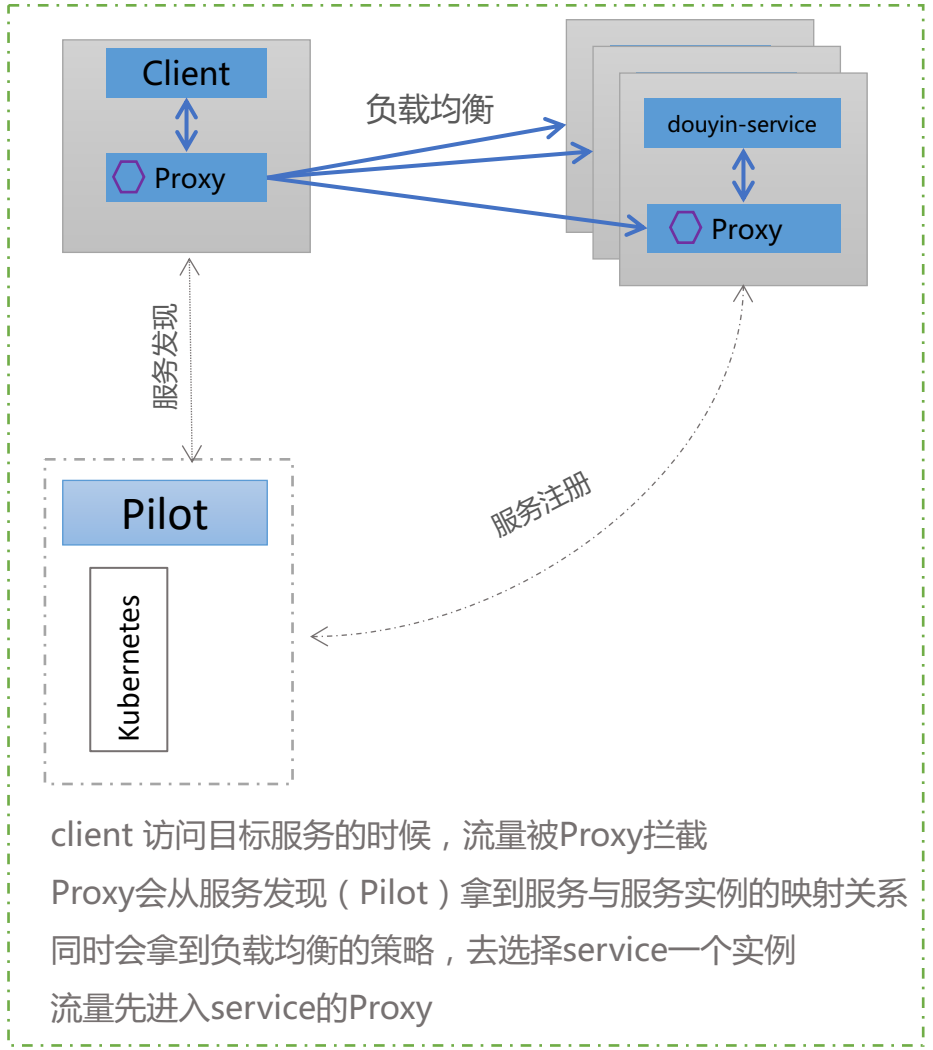
两种服务注册（与发现）流程



目标服务先把实例注册到服务注册中心

客户端从服务注册中心拿到目标实例的数据

根据负载均衡策略去选择一个服务实例，完成整个请求



client 访问目标服务的时候，流量被Proxy拦截

Proxy会从服务发现（Pilot）拿到服务与服务实例的映射关系

同时会拿到负载均衡的策略，去选择service一个实例

流量先进入service的Proxy

Kitex 使用GRPC协议

客户端在创建的时候指定使用GRPC协议

```
// 使用 WithTransportProtocol 指定 transport
cli, err := service.NewClient(destService, client.WithTransportProtocol(transport.GRPC))
```

Kitex 接入 istio

1.为命名空间开启自动注入：

kubectl label namespace default istio-injection=enabled

2.把 Go 代码打包的镜像部署到集群中

例如我们创建了一个 Deployment, 名为：server-douyin ，另外作为服务端的话需要创建相应的Service

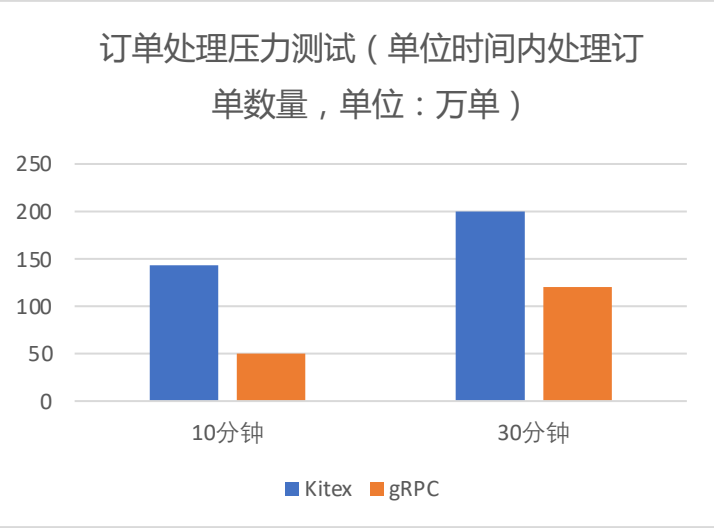
流量拦截和流量代理

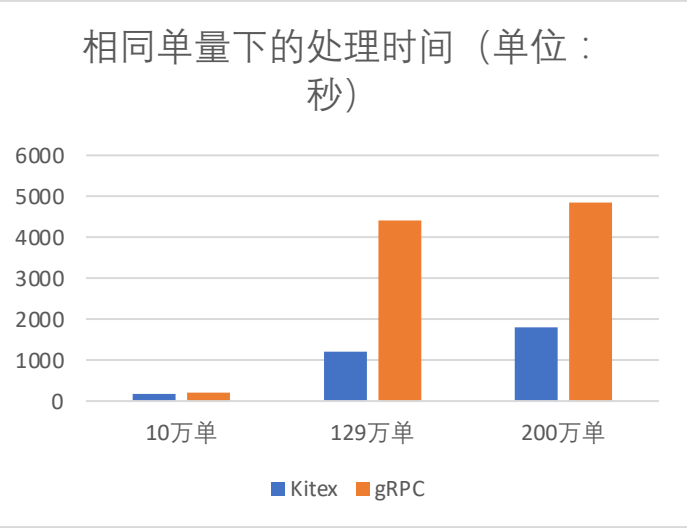
开发的应用容器

压测对比

在相同服务器硬件资源和网络环境下压测对比

- 压测工具：JMeter
- 阿里云ECS（8 vCPU 16 GiB 5 台）
- 集群：Kubernetes 1.20.11
- 服务网格Istio v1.10.5.39





为什么性能会有这么大优势

Netpoll

- 连接利用率
- 调度延迟优化
- 优化I/O调用
- 序列化/反序列化优化
- .....

更多资料可以查看cloudwego 官网:

https://www.cloudwego.io

博客：《字节跳动 Go RPC 框架 Kitex 性能优化实践》

技术支持

官方给了强有力的技术支持

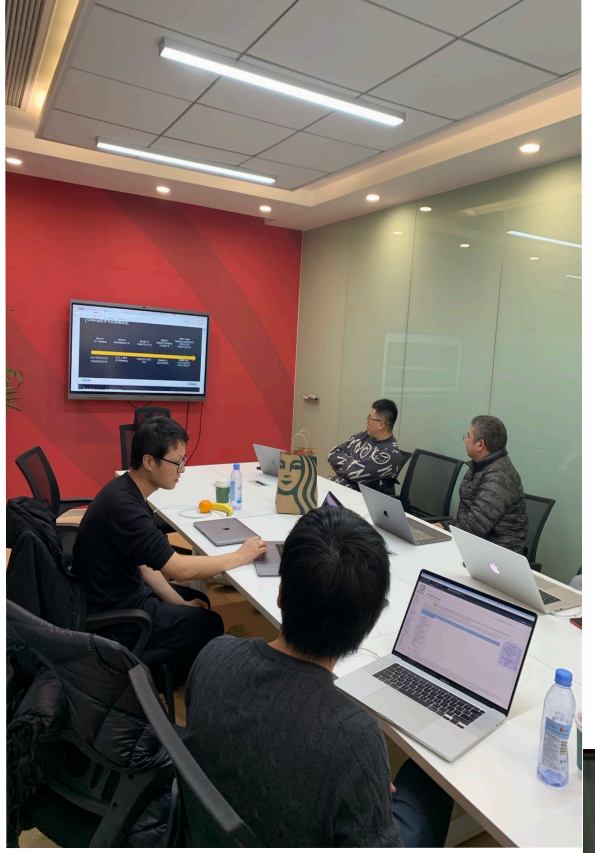
现场支持

远程协助

信心和底气

不管遇到什么样的技术难题都不要怕

后面有强大的技术团队



后续规划

系统发展方向

合作共赢

Thrift, Protobuf 如何选择

Protobuf

项目初期选择gRPC协议是因为选择了Istio 服务网格

主要是因为 Istio 有多流量转发和服务治理等功能

例如在我们电商场景下，不同平台的推送消息，都可以通过VirtualService去转发到不同的服务，相当方便

Thrift

字节官方对此做了很多性能上的优化

如：使用SIMD优化Thrift 编码，减少函数调用，减少内存操作等

开源了 Thrift 编解码器： Frugal ，进一步提升了性能和开发效率

Thrift, Protobuf 如何选择

Istio 不足

每个 pod 中放入原有的 kube-proxy 路由转发功能，会增加响应延迟

由于 sidecar 拦截流量时跳数更多，会消耗更多的资源

Thrift 优势

Kitex 默认支持的协议

高性能 Thrift 编解码器 : Frugal

Frugal 特点:

- **无需生成代码**
- **高性能** (在多核场景下，Frugal 的性能可以达到传统编解码方式的 5 倍！)
- **稳定性**

服务、合作共赢

扫描二维码加入群聊

稳定的系统

开发的相关电商产品不仅为自己电商品牌使用

产品成熟后还可以服务于其他相似的电商公司

希望和官方有更深的技术合作

如：电商云

THANKS

CloudWeGo 稀土掘金