

马士兵教育-申专

课程内容：Go-Micro微服务

介绍：

项目目的：

1:认识Go-Micro微服务

演变史 流程图

单体 -

垂直/水平拆 -

SOA架构 - (service Oriented Architecture 面向服务的架构) - 将重复性功能抽离，通过ESB服务总线访问

系统与业务界限模糊不利于开发；ESB接口协议不利于维护；抽取粒度较大耦合性依旧

微服务：(SOA的一种变种，相对更轻量级，没有什么实质性的区别)

服务层 抽取不同的微服务。

粒度细利于开发，可维护性更强 比ESB更轻量

分布式-集群

中台：是微服务的升级，原本零散的服务，负责提供接口功能。如用户 商品 订单 权限 等。在中台里 升级为商品中心(中台) 订单中心，每个中心更强调体系，包括更好的边界划分和业务抽象，更好地监控和系统运营能力（稳定性，故障定位）不同中台围绕核心业务，自成体系，不同中台即相互独立又构成整体，共同组成基础业务平台即中台。

Go-Micro基本应用：

```
import (
    "github.com/gin-gonic/gin"
    "go-micro.dev/v4/web"
)

func main() {
    router := gin.Default()

    router.Handle("GET", "/login", func(context *gin.Context) {
        context.String(200, "login ...")
    })

    service := web.NewService(
        web.Address(":8081"),
        web.Handler(router),
    )
    service.Run()
```

完成RPC通信：

服务端

```
package main

import (
    "fmt"
    "net/http"
    "net/rpc"
)

type User struct {
    Name, Pwd string
    Age, Score int
}

/**
```

注意进行RPC通信需要

1, 方法只能由两个可序列化的参数 (!channel / !func), 参数1请求的参数, 参数2是一个指针 是给客户端的响应

2, 方法需要返回error 而且是公开的方法

```
*/  
  
func (user *User) GetUserAge(u User, resp *int) error {  
    *resp = u.Age * u.Score  
    return nil  
}  
  
func main() {  
    //1, 需要注册的服务  
    user := new(User)  
    //2, 注册服务  
    rpc.Register(user)  
    //3, 绑定到Http协议上  
    rpc.HandleHTTP()  
    err := http.ListenAndServe(":8081", nil)  
    if err != nil {  
        fmt.Println(err)  
    }  
}
```

客户端:

```
package main  
  
import (  
    "fmt"  
    "net/rpc"  
)  
  
type User struct {  
    Name, Pwd string  
    Age, Score int  
}  
  
func main() {  
    //1连接远程RPC服务  
    conn, err := rpc.DialHTTP("tcp", "localhost:8081")  
    if err != nil {  
        fmt.Println(err)  
    }  
    //2远程服务的调用 参数1 服务的方法, 参数2 resq 参数三 resp  
    res := 0  
    conn.Call("User.GetUserAge", User{"张三", "123", 18, 80}, &res)
```

```
    fmt.Println("客户端收到服务的返回: ", res)
}
```

实现商品的微服务

server端

```
package main

import (
    "fmt"
    "net"
    "net/rpc"
)

type User struct {
    Name, Pwd string
    Age, Score int
}

type Goods struct {
}

type AddGoodsReq struct {
    Id    int
    Name  string
}

type AddGoodsResp struct {
    Success bool
    Message string
}

func (user *Goods) AddGoods(req string, resp *string) error {
    fmt.Println("gorm 已入库")
    *resp = "gorm 已入库"
    return nil
}

func (user *Goods) AddGood(req AddGoodsReq, resp *AddGoodsResp) error {
    fmt.Println("gorm 已入库", req)
    *resp = AddGoodsResp{
        Success: true,
        Message: "添加成功",
    }
    return nil
}

/**
```

注意进行RPC通信需要

1, 方法只能由两个可序列化的参数(!channel / !func), 参数1请求的参数, 参数2是一个指针 是给客户端的响应

2, 方法需要返回error 而且是公开的方法

```
*/  
  
func (user *User) GetUserAge(u User, resp *int) error {  
    *resp = u.Age * u.Score  
    return nil  
}  
  
func main() {  
    // //1, 需要注册的服务  
    // user := new(User)  
    // //2, 注册服务  
    // rpc.Register(user)  
    // //3, 绑定到Http协议上  
    // rpc.HandleHTTP()  
    // err := http.ListenAndServe(":8081", nil)  
    // if err != nil {  
    //     fmt.Println(err)  
    // }  
    //注册服务 Goods  
    err1 := rpc.RegisterName("goods", new(Goods))  
    if err1 != nil {  
        fmt.Println("插入有误", err1)  
    }  
    //监听端口  
    listen, err2 := net.Listen("tcp", "127.0.0.1:8081")  
    if err2 != nil {  
        fmt.Println("插入有误", err2)  
    }  
    //关闭  
    defer listen.Close()  
    for {  
        //接收连接  
        conn, err3 := listen.Accept()  
        if err3 != nil {  
            fmt.Println("插入有误", err3)  
        }  
        //处理连接  
        rpc.ServeConn(conn)  
    }  
}
```

client端:

```
package main

import (
    "fmt"
    "net/rpc"
)

type User struct {
    Name, Pwd string
    Age, Score int
}

type Goods struct {
}

//这边不要写
// func (user *User) AddGoods(req string, resp *string) error {
//     fmt.Println("gorm 已入库")
//     return nil
// }

type AddGoodsReq struct {
    Id int
    Name string
}

type AddGoodsResp struct {
    Success bool
    Message string
}

func main() {
    // //1连接远程RPC服务
    // conn, err := rpc.DialHTTP("tcp", "localhost:8081")
    // if err != nil {
    //     fmt.Println(err)
    // }
    // //2远程服务的调用 参数1 服务的方法, 参数2 req 参数三 resp
    // res := 0
    // conn.Call("User.GetUserAge", User{"张三", "123", 18, 80}, &res)
    // fmt.Println("客户端收到服务的返回:", res)

    conn, err1 := rpc.Dial("tcp", "127.0.0.1:8081")
    if err1 != nil {
        fmt.Println(err1)
    }
    defer conn.Close()
    var resp string
    conn.Call("goods.AddGoods", "商品1", &resp)
    fmt.Println(resp)
    //var req AddGoodsReq
}
```

```

var respA AddGoodsResp

conn.Call("goods.AddGood", AddGoodsReq{
    Id: 1,
    Name: "商品2",
}, &respA)
fmt.Println(respA)
}

```

GRPC 与 Protobuf

go 标准库 RPC 默认的 god 编码 ,RPC 通信 序列化 跨语言都无法完成

官方也提供 net/rpc/jsonrpc 实现 json 的序列化 , 支持跨语言调用。jsonrpc 基于 TCP 协议 , 不支持 HTTP 协议传输

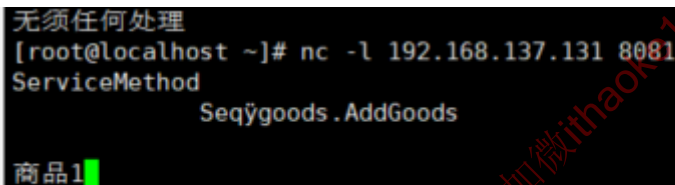
nc 网络工具, linux 瑞士军刀

centos 安装: yum install -y nc

然后通过 nc 作为微服务启动 server nc -l 192.168.137.131 8081

修改客户端连接代码 并测试

```
conn, err1 := rpc.Dial("tcp", "192.168.137.131:8081")
```



```

无须任何处理
[root@localhost ~]# nc -l 192.168.137.131 8081
ServiceMethod
SeqYgoods.AddGoods
商品1

```

json 编码器 修改服务端:

```
rpc.ServeCodec(jsonrpc.NewServerCodec(conn)) //注意包是 net/rpc/jsonrpc
```

并且客户端端:

```
conn, err1 := jsonrpc.Dial("tcp", "192.168.137.131:8081")
```

然后启动测试:

```
1 192.168.137.131 x +
[root@localhost ~]# nc -l 192.168.137.131 8081
{"method":"goods.AddGoods","params":["商品1"],"id":0}
[root@localhost ~]# nc -l 192.168.137.131 8081
{"method":"goods.AddGoods","params":["商品1"],"id":0}
```

客户端方案二（备用）

```
conn, err1 := net.Dial("tcp", "192.168.137.131:8081")
if err1 != nil {
    fmt.Println(err1)
}
defer conn.Close()
//此处会报问题
client := rpc.NewClientWithCodec(jsonrpc.NewClientCodec(conn))

var resp string
client.Call("goods.AddGoods", "商品1", &resp)
fmt.Println(resp)
//var req AddGoodsReq
var respA AddGoodsResp

client.Call("goods.AddGood", AddGoodsReq{
    Id: 1,
    Name: "商品2",
}, &respA)
fmt.Println(respA)
```

Protobuf介绍:

官网: <https://developers.google.cn/protocol-buffers/>

是与语言 平台无关。可扩展的序列化结构化数据。通常用通信协议 数据存储

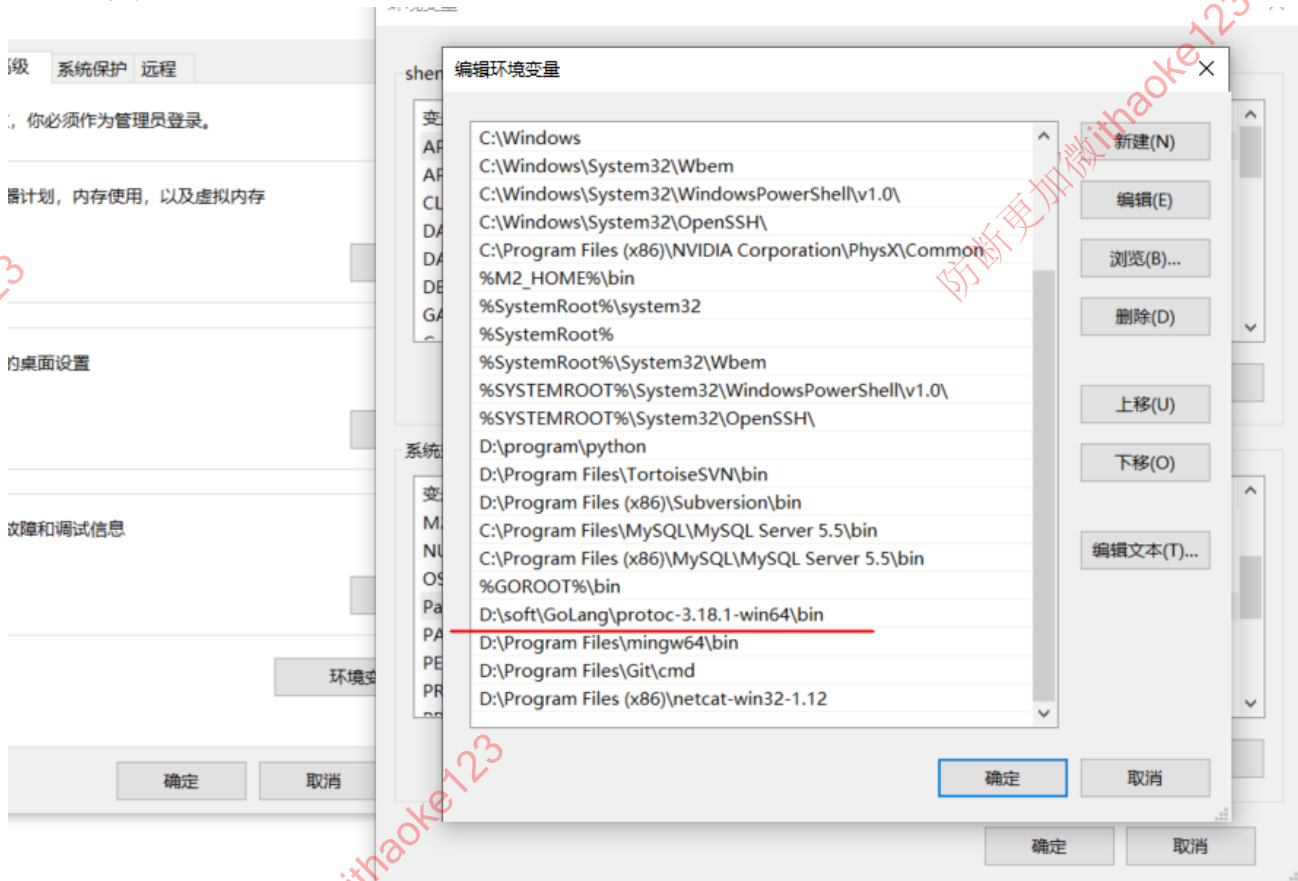
兼容性/跨语言 更快 更小 更简单 （可读性 无法标识复杂概念）

安装与使用:

<https://github.com/protocolbuffers/protobuf>

根据操作系统 下载 zip

配置环境变量：



查看版本：protoc --version

新建 test.proto 的文件

```
message Person {  
    required string name = 1;  
    required int32 id = 2;  
    optional string email = 3;  
}
```

安装 go语言 插件

go install github.com/golang/protobuf/protoc-gen-go@latest

接下来 protobuf的使用：

创建 test.proto文件

```
syntax = "proto3";
option go_package="/grpc";
package grpc ;
message Order {
    int64 id = 1;
    string no = 2;
}
```

执行命令生成各种 语言的文件

protoc -I . --go_out=. test.proto

protoc -I . --cpp_out=. test.proto

protoc -I . --java_out=. test.proto

ProtoBuf综合应用:

请求参数

参数名称	参数说明	请求类型	是否必须	数据类型	schema
activityType	1: 正常购买, 2: 免费领取, 3: 秒杀	query	true	integer(int32)	
productSkuId	商品SKU-ID	query	true	integer(int64)	
quantity	购买数量	query	true	integer(int32)	
activityId	活动ID (参加活动时必填)	query	false	integer(int64)	
activityTimeId	活动时间ID (参加活动时必填)	query	false	integer(int64)	
isVirtual	false: 否, true: 是	query	false	boolean	
recipientAddressId	收货地址ID (非虚拟商品订单必传)	query	false	integer(int64)	

响应参数

参数名称	参数说明	类型	schema
discountAmount	优惠金额	number(bigdecimal)	number(bigdecimal)
expireTime	支付失效时间	string(date-time)	string(date-time)
payAmount	实付金额	number(bigdecimal)	number(bigdecimal)
productAmount	实际商品金额	number(bigdecimal)	number(bigdecimal)
products	商品详情信息	array	预订单商品信息VO
productId	商品ID	integer(int64)	
productImageUrl	商品图片	string	
productName	商品名称	string	
productPrice	商品单价	number(bigdecimal)	
productSkuId	商品SKU-ID	integer(int64)	
quantity	购买数量	integer(int32)	
realAmount	实际金额	number(bigdecimal)	
realPrice	实际价格	number(bigdecimal)	
skuDescribe	SKU规格描述	string	
serverTime	服务器时间	string(date-time)	string(date-time)
shippingAmount	运费	number(bigdecimal)	number(bigdecimal)
totalAmount	总金额	number(bigdecimal)	number(bigdecimal)

```
req:
  activityType      1: 正常购买, 2: 免费领取, 3: 秒杀      query      true
  integer(int32)
  productSkuId      商品SKU-ID      query      true
  integer(int64)
  quantity          购买数量      query      true
  integer(int32)
  activityId        活动ID (参加活动时必须)      query      false
  integer(int64)
  activityTimeId    活动时间ID (参加活动时必须)      query      false
  integer(int64)
```

```

isVirtual    false: 否, true: 是    query    false
boolean
recipientAddressId  收货地址ID（非虚拟商品订单必传）    query    false
integer(int64)

resp:
{
    "discountAmount": 0,
    "expireTime": "",
    "payAmount": 0,
    "productAmount": 0,
    "products": [
        {
            "productId": 0,
            "productImageUrl": "",
            "productName": "",
            "productPrice": 0,
            "productSkuId": 0,
            "quantity": 0,
            "realAmount": 0,
            "realPrice": 0,
            "skuDescribe": ""
        }
    ],
    "serverTime": "",
    "shippingAmount": 0,
    "totalAmount": 0
}

```

proto代码:

```

syntax = "proto3";    // 版本号
option go_package="./;proto";    //参数1 表示生成到哪个目录 , 参数2 表示生成的文件的package
package proto ;    //默认在哪个包
//结构体
message Order {
    int64 id = 1;
    string no = 2;
    string tel =3;
    string address = 4;
    string addTime = 5;
    OrderStatus status = 6;
    message Producets {
        int64 id = 1;
        string title = 2;
    }
}

```

```

        double price =3;
        int32 num = 4;
    }
    Producets producets = 7;
}
//也可写这里
// message Producets {
//     int64 id = 1;
//     string title = 2;
//     double price =3;
//     int32 num = 4;
// }
//请求 request struct
message OrderRequest {
    int32 activityType = 1;
}
//响应 resp struct
message OrderResp{
    int32 discountAmount = 1;
    string code = 2;
    string msg = 3;
}
enum OrderStatus {
    FIRST_VAL = 0 ;    //The first enum value must be zero in proto3.
    SECOND_VAL = 1;
}
//RPC 服务 接口
service SearchService {
    //rpc 服务
    rpc SearchOrder (OrderRequest) returns (OrderResp){}
}

```

核对go mod 版本

```

module msb_gomicro

//replace google.golang.org/grpc => google.golang.org/grpc v1.26.0

go 1.17

require (
    google.golang.org/grpc v1.47.0
    google.golang.org/protobuf v1.28.0
)

```

```
require (
    github.com/golang/protobuf v1.5.2 // indirect
    golang.org/x/net v0.0.0-20201021035429-f5854403a974 // indirect
    golang.org/x/sys v0.0.0-20210119212857-b64e53b001e4 // indirect
    golang.org/x/text v0.3.3 // indirect
    google.golang.org/genproto v0.0.0-20200526211855-cb27e3aa2013 //
indirect
)
```

执行: `protoc --proto_path=. --go_out=plugins=grpc,paths=source_relative:. test.proto`

生成的 test.pb.go

```
// Code generated by protoc-gen-go. DO NOT EDIT.
// versions:
//  protoc-gen-go v1.26.0
//  protoc v3.18.1
//  source: test.proto

package proto

import (
    context "context"
    grpc "google.golang.org/grpc"
    codes "google.golang.org/grpc/codes"
    status "google.golang.org/grpc/status"
    protoreflect "google.golang.org/protobuf/reflect/protoreflect"
    protoimpl "google.golang.org/protobuf/runtime/protoimpl"
    reflect "reflect"
    sync "sync"
)

const (
    // Verify that this generated code is sufficiently up-to-date.
    _ = protoimpl.EnforceVersion(20 - protoimpl.MinVersion)
    // Verify that runtime/protoimpl is sufficiently up-to-date.
    _ = protoimpl.EnforceVersion(protoimpl.MaxVersion - 20)
)

type OrderStatus int32

const (
```

```

    OrderStatus_FIRST_VAL OrderStatus = 0 //The first enum value must
be zero in proto3.
    OrderStatus_SECOND_VAL OrderStatus = 1
)

// Enum value maps for OrderStatus.
var (
    OrderStatus_name = map[int32]string{
        0: "FIRST_VAL",
        1: "SECOND_VAL",
    }
    OrderStatus_value = map[string]int32{
        "FIRST_VAL": 0,
        "SECOND_VAL": 1,
    }
)

func (x OrderStatus) Enum() *OrderStatus {
    p := new(OrderStatus)
    *p = x
    return p
}

func (x OrderStatus) String() string {
    return protoimpl.X.EnumStringOf(x.Descriptor(),
    protorelect.EnumNumber(x))
}

func (OrderStatus) Descriptor() protorelect.EnumDescriptor {
    return file_test_proto_enumTypes[0].Descriptor()
}

func (OrderStatus) Type() protorelect.EnumType {
    return &file_test_proto_enumTypes[0]
}

func (x OrderStatus) Number() protorelect.EnumNumber {
    return protorelect.EnumNumber(x)
}

// Deprecated: Use OrderStatus.Descriptor instead.
func (OrderStatus) EnumDescriptor() ([]byte, []int) {
    return file_test_proto_rawDescGZIP(), []int{0}
}

//结构体
type Order struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache

```

```

unknownFields protoimpl.UnknownFields

Id          int64          `protobuf:"varint,1,opt,name=id,proto3"
json:"id,omitempty"`
No          string         `protobuf:"bytes,2,opt,name=no,proto3"
json:"no,omitempty"`
Tel         string         `protobuf:"bytes,3,opt,name=tel,proto3"
json:"tel,omitempty"`
Address     string
`protobuf:"bytes,4,opt,name=address,proto3" json:"address,omitempty"`
AddTime     string
`protobuf:"bytes,5,opt,name=addTime,proto3" json:"addTime,omitempty"`
Status      OrderStatus
`protobuf:"varint,6,opt,name=status,proto3,enum=proto.OrderStatus"
json:"status,omitempty"`
Producets   *Order_Producets
`protobuf:"bytes,7,opt,name=producets,proto3"
json:"producets,omitempty"`
}

func (x *Order) Reset() {
    *x = Order{}
    if protoimpl.UnsafeEnabled {
        mi := &file_test_proto_msgTypes[0]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}

func (x *Order) String() string {
    return protoimpl.X.MessageStringOf(x)
}

func (*Order) ProtoMessage() {}

func (x *Order) ProtoReflect() protoreflect.Message {
    mi := &file_test_proto_msgTypes[0]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}

// Deprecated: Use Order.ProtoReflect.Descriptor instead.
func (*Order) Descriptor() ([]byte, []int) {

```



```
        return file_test_proto_rawDescGZIP(), []int{0}
    }

    func (x *Order) GetId() int64 {
        if x != nil {
            return x.Id
        }
        return 0
    }

    func (x *Order) GetNo() string {
        if x != nil {
            return x.No
        }
        return ""
    }

    func (x *Order) GetTel() string {
        if x != nil {
            return x.Tel
        }
        return ""
    }

    func (x *Order) GetAddress() string {
        if x != nil {
            return x.Address
        }
        return ""
    }

    func (x *Order) GetAddTime() string {
        if x != nil {
            return x.AddTime
        }
        return ""
    }

    func (x *Order) GetStatus() OrderStatus {
        if x != nil {
            return x.Status
        }
        return OrderStatus_FIRST_VAL
    }

    func (x *Order) GetProducets() *Order_Producets {
        if x != nil {
            return x.Producets
        }
    }
```

```

        return nil
    }

    //也可写这里
    // message Producets {
    //     int64 id = 1;
    //     string title = 2;
    //     double price = 3;
    //     int32 num = 4;
    // }
    //请求 request struct
    type OrderRequest struct {
        state          protoimpl.MessageState
        sizeCache       protoimpl.SizeCache
        unknownFields   protoimpl.UnknownFields

        ActivityType int32 `protobuf:"varint,1,opt,name=activityType,proto3"
        json:"activityType,omitempty"`
    }

    func (x *OrderRequest) Reset() {
        *x = OrderRequest{}
        if protoimpl.UnsafeEnabled {
            mi := &file_test_proto_msgTypes[1]
            ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
            ms.StoreMessageInfo(mi)
        }
    }

    func (x *OrderRequest) String() string {
        return protoimpl.X.MessageStringOf(x)
    }

    func (*OrderRequest) ProtoMessage() {}

    func (x *OrderRequest) ProtoReflect() protoreflect.Message {
        mi := &file_test_proto_msgTypes[1]
        if protoimpl.UnsafeEnabled && x != nil {
            ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
            if ms.LoadMessageInfo() == nil {
                ms.StoreMessageInfo(mi)
            }
            return ms
        }
        return mi.MessageOf(x)
    }

    // Deprecated: Use OrderRequest.ProtoReflect.Descriptor instead.
    func (*OrderRequest) Descriptor() ([]byte, []int) {

```

```

        return file_test_proto_rawDescGZIP(), []int{1}
    }

    func (x *OrderRequest) GetActivityType() int32 {
        if x != nil {
            return x.ActivityType
        }
        return 0
    }

    //响应 resp struct
    type OrderResp struct {
        state          protoimpl.MessageState
        sizeCache      protoimpl.SizeCache
        unknownFields  protoimpl.UnknownFields

        DiscountAmount int32
        `protobuf:"varint,1,opt,name=discountAmount,proto3"
json:"discountAmount,omitempty"`
        Code           string `protobuf:"bytes,2,opt,name=code,proto3"
json:"code,omitempty"`
        Msg            string `protobuf:"bytes,3,opt,name=msg,proto3"
json:"msg,omitempty"`
    }

    func (x *OrderResp) Reset() {
        *x = OrderResp{}
        if protoimpl.UnsafeEnabled {
            mi := &file_test_proto_msgTypes[2]
            ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
            ms.StoreMessageInfo(mi)
        }
    }

    func (x *OrderResp) String() string {
        return protoimpl.X.MessageStringOf(x)
    }

    func (*OrderResp) ProtoMessage() {}

    func (x *OrderResp) ProtoReflect() protoreflect.Message {
        mi := &file_test_proto_msgTypes[2]
        if protoimpl.UnsafeEnabled && x != nil {
            ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
            if ms.LoadMessageInfo() == nil {
                ms.StoreMessageInfo(mi)
            }
            return ms
        }
    }

```

```

        return mi.MessageOf(x)
    }

    // Deprecated: Use OrderResp.ProtoReflect.Descriptor instead.
    func (*OrderResp) Descriptor() ([]byte, []int) {
        return file_test_proto_rawDescGZIP(), []int{2}
    }

    func (x *OrderResp) GetDiscountAmount() int32 {
        if x != nil {
            return x.DiscountAmount
        }
        return 0
    }

    func (x *OrderResp) GetCode() string {
        if x != nil {
            return x.Code
        }
        return ""
    }

    func (x *OrderResp) GetMsg() string {
        if x != nil {
            return x.Msg
        }
        return ""
    }

    type Order_Products struct {
        state          protoimpl.MessageState
        sizeCache      protoimpl.SizeCache
        unknownFields  protoimpl.UnknownFields

        Id      int64      `protobuf:"varint,1,opt,name=id,proto3"
json:"id,omitempty"`
        Title   string     `protobuf:"bytes,2,opt,name=title,proto3"
json:"title,omitempty"`
        Price   float64    `protobuf:"fixed64,3,opt,name=price,proto3"
json:"price,omitempty"`
        Num     int32      `protobuf:"varint,4,opt,name=num,proto3"
json:"num,omitempty"`
    }

    func (x *Order_Products) Reset() {
        *x = Order_Products{}
        if protoimpl.UnsafeEnabled {
            mi := &file_test_proto_msgTypes[3]
            ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))

```

```

        ms.StoreMessageInfo(mi)
    }
}

func (x *Order_Products) String() string {
    return protoimpl.X.MessageStringOf(x)
}

func (*Order_Products) ProtoMessage() {}

func (x *Order_Products) ProtoReflect() protoreflect.Message {
    mi := &file_test_proto_msgTypes[3]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}

// Deprecated: Use Order_Products.ProtoReflect.Descriptor instead.
func (*Order_Products) Descriptor() ([]byte, []int) {
    return file_test_proto_rawDescGZIP(), []int{0, 0}
}

func (x *Order_Products) GetId() int64 {
    if x != nil {
        return x.Id
    }
    return 0
}

func (x *Order_Products) GetTitle() string {
    if x != nil {
        return x.Title
    }
    return ""
}

func (x *Order_Products) GetPrice() float64 {
    if x != nil {
        return x.Price
    }
    return 0
}

func (x *Order_Products) GetNum() int32 {

```

```

        if x != nil {
            return x.Num
        }
        return 0
    }

var File_test_proto protoreflect.FileDescriptor

var file_test_proto_rawDesc = []byte{
    0x0a, 0x0a, 0x74, 0x65, 0x73, 0x74, 0x2e, 0x70, 0x72, 0x6f, 0x74,
    0x6f, 0x12, 0x05, 0x70, 0x72,
    0x6f, 0x74, 0x6f, 0x22, 0xaa, 0x02, 0x0a, 0x05, 0x4f, 0x72, 0x64,
    0x65, 0x72, 0x12, 0x0e, 0x0a,
    0x02, 0x69, 0x64, 0x18, 0x01, 0x20, 0x01, 0x28, 0x03, 0x52, 0x02,
    0x69, 0x64, 0x12, 0x0e, 0x0a,
    0x02, 0x6e, 0x6f, 0x18, 0x02, 0x20, 0x01, 0x28, 0x09, 0x52, 0x02,
    0x6e, 0x6f, 0x12, 0x10, 0x0a,
    0x03, 0x74, 0x65, 0x6c, 0x18, 0x03, 0x20, 0x01, 0x28, 0x09, 0x52,
    0x03, 0x74, 0x65, 0x6c, 0x12,
    0x18, 0x0a, 0x07, 0x61, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x18,
    0x04, 0x20, 0x01, 0x28, 0x09,
    0x52, 0x07, 0x61, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x12, 0x18,
    0x0a, 0x07, 0x61, 0x64, 0x64,
    0x54, 0x69, 0x6d, 0x65, 0x18, 0x05, 0x20, 0x01, 0x28, 0x09, 0x52,
    0x07, 0x61, 0x64, 0x64, 0x54,
    0x69, 0x6d, 0x65, 0x12, 0x2a, 0x0a, 0x06, 0x73, 0x74, 0x61, 0x74,
    0x75, 0x73, 0x18, 0x06, 0x20,
    0x01, 0x28, 0x0e, 0x32, 0x12, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f,
    0x2e, 0x4f, 0x72, 0x64, 0x65,
    0x72, 0x53, 0x74, 0x61, 0x74, 0x75, 0x73, 0x52, 0x06, 0x73, 0x74,
    0x61, 0x74, 0x75, 0x73, 0x12,
    0x34, 0x0a, 0x09, 0x70, 0x72, 0x6f, 0x64, 0x75, 0x63, 0x65, 0x74,
    0x73, 0x18, 0x07, 0x20, 0x01,
    0x28, 0x0b, 0x32, 0x16, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e,
    0x4f, 0x72, 0x64, 0x65, 0x72,
    0x2e, 0x50, 0x72, 0x6f, 0x64, 0x75, 0x63, 0x65, 0x74, 0x73, 0x52,
    0x09, 0x70, 0x72, 0x6f, 0x64,
    0x75, 0x63, 0x65, 0x74, 0x73, 0x1a, 0x59, 0x0a, 0x09, 0x50, 0x72,
    0x6f, 0x64, 0x75, 0x63, 0x65,
    0x74, 0x73, 0x12, 0x0e, 0x0a, 0x02, 0x69, 0x64, 0x18, 0x01, 0x20,
    0x01, 0x28, 0x03, 0x52, 0x02,
    0x69, 0x64, 0x12, 0x14, 0x0a, 0x05, 0x74, 0x69, 0x74, 0x6c, 0x65,
    0x18, 0x02, 0x20, 0x01, 0x28,
    0x09, 0x52, 0x05, 0x74, 0x69, 0x74, 0x6c, 0x65, 0x12, 0x14, 0x0a,
    0x05, 0x70, 0x72, 0x69, 0x63,
    0x65, 0x18, 0x03, 0x20, 0x01, 0x28, 0x01, 0x52, 0x05, 0x70, 0x72,
    0x69, 0x63, 0x65, 0x12, 0x10,
    0x0a, 0x03, 0x6e, 0x75, 0x6d, 0x18, 0x04, 0x20, 0x01, 0x28, 0x05,
    0x52, 0x03, 0x6e, 0x75, 0x6d,

```

```
0x22, 0x32, 0x0a, 0x0c, 0x4f, 0x72, 0x64, 0x65, 0x72, 0x52, 0x65,
0x71, 0x75, 0x65, 0x73, 0x74,
0x12, 0x22, 0x0a, 0x0c, 0x61, 0x63, 0x74, 0x69, 0x76, 0x69, 0x74,
0x79, 0x54, 0x79, 0x70, 0x65,
0x18, 0x01, 0x20, 0x01, 0x28, 0x05, 0x52, 0x0c, 0x61, 0x63, 0x74,
0x69, 0x76, 0x69, 0x74, 0x79,
0x54, 0x79, 0x70, 0x65, 0x22, 0x59, 0x0a, 0x09, 0x4f, 0x72, 0x64,
0x65, 0x72, 0x52, 0x65, 0x73,
0x70, 0x12, 0x26, 0x0a, 0x0e, 0x64, 0x69, 0x73, 0x63, 0x6f, 0x75,
0x6e, 0x74, 0x41, 0x6d, 0x6f,
0x75, 0x6e, 0x74, 0x18, 0x01, 0x20, 0x01, 0x28, 0x05, 0x52, 0x0e,
0x64, 0x69, 0x73, 0x63, 0x6f,
0x75, 0x6e, 0x74, 0x41, 0x6d, 0x6f, 0x75, 0x6e, 0x74, 0x12, 0x12,
0x0a, 0x04, 0x63, 0x6f, 0x64,
0x65, 0x18, 0x02, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x63, 0x6f,
0x64, 0x65, 0x12, 0x10, 0x0a,
0x03, 0x6d, 0x73, 0x67, 0x18, 0x03, 0x20, 0x01, 0x28, 0x09, 0x52,
0x03, 0x6d, 0x73, 0x67, 0x2a,
0x2c, 0x0a, 0x0b, 0x4f, 0x72, 0x64, 0x65, 0x72, 0x53, 0x74, 0x61,
0x74, 0x75, 0x73, 0x12, 0x0d,
0x0a, 0x09, 0x46, 0x49, 0x52, 0x53, 0x54, 0x5f, 0x56, 0x41, 0x4c,
0x10, 0x00, 0x12, 0x0e, 0x0a,
0x0a, 0x53, 0x45, 0x43, 0x4f, 0x4e, 0x44, 0x5f, 0x56, 0x41, 0x4c,
0x10, 0x01, 0x32, 0x47, 0x0a,
0x0d, 0x53, 0x65, 0x61, 0x72, 0x63, 0x68, 0x53, 0x65, 0x72, 0x76,
0x69, 0x63, 0x65, 0x12, 0x36,
0x0a, 0x0b, 0x53, 0x65, 0x61, 0x72, 0x63, 0x68, 0x4f, 0x72, 0x64,
0x65, 0x72, 0x12, 0x13, 0x2e,
0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x4f, 0x72, 0x64, 0x65, 0x72,
0x52, 0x65, 0x71, 0x75, 0x65,
0x73, 0x74, 0x1a, 0x10, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e,
0x4f, 0x72, 0x64, 0x65, 0x72,
0x52, 0x65, 0x73, 0x70, 0x22, 0x00, 0x42, 0x0a, 0x5a, 0x08, 0x2e,
0x2f, 0x3b, 0x70, 0x72, 0x6f,
0x74, 0x6f, 0x62, 0x06, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x33,
}
```

```
var (
    file_test_proto_rawDescOnce sync.Once
    file_test_proto_rawDescData = file_test_proto_rawDesc
)

func file_test_proto_rawDescGZIP() []byte {
    file_test_proto_rawDescOnce.Do(func() {
        file_test_proto_rawDescData =
protoimpl.X.CompressGZIP(file_test_proto_rawDescData)
    })
    return file_test_proto_rawDescData
}
```

```

var file_test_proto_enumTypes = make([]protoimpl.EnumInfo, 1)
var file_test_proto_msgTypes = make([]protoimpl.MessageInfo, 4)
var file_test_proto_goTypes = []interface{}{
    (OrderStatus)(0),          // 0: proto.OrderStatus
    (*Order)(nil),             // 1: proto.Order
    (*OrderRequest)(nil),      // 2: proto.OrderRequest
    (*OrderResp)(nil),         // 3: proto.OrderResp
    (*Order_Products)(nil),    // 4: proto.Order.Products
}
var file_test_proto_depIdxs = []int32{
    0, // 0: proto.Order.status:type_name -> proto.OrderStatus
    4, // 1: proto.Order.products:type_name -> proto.Order.Products
    2, // 2: proto.SearchService.SearchOrder:input_type ->
proto.OrderRequest
    3, // 3: proto.SearchService.SearchOrder:output_type ->
proto.OrderResp
    3, // [3:4] is the sub-list for method output_type
    2, // [2:3] is the sub-list for method input_type
    2, // [2:2] is the sub-list for extension type_name
    2, // [2:2] is the sub-list for extension extendee
    0, // [0:2] is the sub-list for field type_name
}

func init() { file_test_proto_init() }
func file_test_proto_init() {
    if File_test_proto != nil {
        return
    }
    if !protoimpl.UnsafeEnabled {
        file_test_proto_msgTypes[0].Exporter = func(v interface{}, i
int) interface{} { {
            switch v := v.(*Order); i {
            case 0:
                return &v.state
            case 1:
                return &v.sizeCache
            case 2:
                return &v.unknownFields
            default:
                return nil
            }
        }
        file_test_proto_msgTypes[1].Exporter = func(v interface{}, i
int) interface{} { {
            switch v := v.(*OrderRequest); i {
            case 0:
                return &v.state
            case 1:

```



```

        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}

file_test_proto_msgTypes[2].Exporter = func(v interface{}, i
int) interface{} {
    switch v := v.(*OrderResp); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}

file_test_proto_msgTypes[3].Exporter = func(v interface{}, i
int) interface{} {
    switch v := v.(*Order_Products); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
}

type x struct{}
out := protoimpl.TypeBuilder{
    File: protoimpl.DescBuilder{
        GoPackagePath: reflect.TypeOf(x{}).PkgPath(),
        RawDescriptor: file_test_proto_rawDesc,
        NumEnums:      1,
        NumMessages:    4,
        NumExtensions:  0,
        NumServices:    1,
    },
    GoTypes:           file_test_proto_goTypes,
    DependencyIndexes: file_test_proto_depIdxs,
    EnumInfos:         file_test_proto_enumTypes,
    MessageInfos:      file_test_proto_msgTypes,
}.Build()

```

```

    File_test_proto = out.File
    file_test_proto_rawDesc = nil
    file_test_proto_goTypes = nil
    file_test_proto_depIdxs = nil
}

// Reference imports to suppress errors if they are not otherwise used.
var _ context.Context
var _ grpc.ClientConnInterface

// This is a compile-time assertion to ensure that this generated file
// is compatible with the grpc package it is being compiled against.
const _ = grpc.SupportPackageIsVersion6

// SearchServiceClient is the client API for SearchService service.
//
// For semantics around ctx use and closing/ending streaming RPCs,
// please refer to
// https://godoc.org/google.golang.org/grpc#ClientConn.NewStream.
type SearchServiceClient interface {
    //rpc 服务
    SearchOrder(ctx context.Context, in *OrderRequest, opts
...grpc.CallOption) (*OrderResp, error)
}

type searchServiceClient struct {
    cc grpc.ClientConnInterface
}

func NewSearchServiceClient(cc grpc.ClientConnInterface)
SearchServiceClient {
    return &searchServiceClient{cc}
}

func (c *searchServiceClient) SearchOrder(ctx context.Context, in
*OrderRequest, opts ...grpc.CallOption) (*OrderResp, error) {
    out := new(OrderResp)
    err := c.cc.Invoke(ctx, "/proto.SearchService/SearchOrder", in, out,
opts...)
    if err != nil {
        return nil, err
    }
    return out, nil
}

// SearchServiceServer is the server API for SearchService service.
type SearchServiceServer interface {
    //rpc 服务
    SearchOrder(context.Context, *OrderRequest) (*OrderResp, error)
}

```

```

}

// UnimplementedSearchServiceServer can be embedded to have forward
compatible implementations.
type UnimplementedSearchServiceServer struct {
}

func (*UnimplementedSearchServiceServer) SearchOrder(context.Context,
*OrderRequest) (*OrderResp, error) {
    return nil, status.Errorf(codes.Unimplemented, "method SearchOrder
not implemented")
}

func RegisterSearchServiceServer(s *grpc.Server, srv
SearchServiceServer) {
    s.RegisterService(&_SearchService_serviceDesc, srv)
}

func _SearchService_SearchOrder_Handler(srv interface{}, ctx
context.Context, dec func(interface{}) error, interceptor
grpc.UnaryServerInterceptor) (interface{}, error) {
    in := new(OrderRequest)
    if err := dec(in); err != nil {
        return nil, err
    }
    if interceptor == nil {
        return srv.(SearchServiceServer).SearchOrder(ctx, in)
    }
    info := &grpc.UnaryServerInfo{
        Server:      srv,
        FullMethod:  "/proto.SearchService/SearchOrder",
    }
    handler := func(ctx context.Context, req interface{}) (interface{},
error) {
        return srv.(SearchServiceServer).SearchOrder(ctx, req.
(*OrderRequest))
    }
    return interceptor(ctx, in, info, handler)
}

var _SearchService_serviceDesc = grpc.ServiceDesc{
    ServiceName: "proto.SearchService",
    HandlerType: (*SearchServiceServer)(nil),
    Methods: []grpc.MethodDesc{
        {
            MethodName: "SearchOrder",
            Handler:    _SearchService_SearchOrder_Handler,
        },
    },
}

```

```
Streams:  []grpc.StreamDesc{},
Metadata: "test.proto",
}
```

server服务端:

```
package main

import (
    "context"
    "errors"
    "fmt"
    pro "msb_gomicro/proto"
    "net"

    "google.golang.org/grpc"

)

//定义个空的接口
type OrderInfoService struct{}

func (order *OrderInfoService) SearchOrder(cxt context.Context, req
*pro.OrderRequest) (*pro.OrderResp, error) {
    reqType := req.ActivityType //获取请求参数
    resp := &pro.OrderResp{}
    if reqType == 1 { //数据库操作
        resp.DiscountAmount = 100
        resp.Code = "200"
        resp.Msg = "查询成功"
    } else {
        resp.DiscountAmount = 0
        resp.Code = "500"
        resp.Msg = "查询失败"
        return resp, errors.New("not found this Order!")
    }
    return resp, nil
}

/**
    服务端 : 1, 开启监听 ,    2, 实例化GPRC服务端    3, 在GRPC上注册一个微服务    4,
    启动服务
    **/
func main() {
    //1, 开启监听
    addr := "127.0.0.1:8081"
    listener, err := net.Listen("tcp", addr)
    if err != nil {
        fmt.Println("监听异常:", err)
    }
}
```

```

    }
    fmt.Println("服务监听端口:", addr)
    //2, 实例化GPRC服务端
    server := grpc.NewServer()
    //3, 在GRPC上注册一个微服务
    s := OrderInfoService{} //空接口服务
    pro.RegisterSearchServiceServer(server, &s)
    //4, 启动服务
    server.Serve(listener)
}

```

client端:

```

package main

import (
    "context"
    "encoding/json"
    "fmt"
    pro "msb_gomicro/proto"

    "google.golang.org/grpc"
)

// 客户端 1, 连接服务      2实例化grpc客户端      3发起请求/调用服务
func main() {
    //1, 连接服务
    conn, err := grpc.Dial("127.0.0.1:8081", grpc.WithInsecure())
    if err != nil {
        fmt.Println("连接异常: ", err)
    }
    defer conn.Close()
    //2实例化grpc客户端
    client := pro.NewSearchServiceClient(conn)
    //3发起请求/调用服务
    req := new(pro.OrderRequest)
    req.ActivityType = 2
    resp, err1 := client.SearchOrder(context.Background(), req)
    if err1 != nil {
        fmt.Println("响应异常: ", err1)
    }
    data, err2 := json.Marshal(resp)
    if err2 != nil {
        fmt.Println("响应异常: ", err2)
    } else {
        fmt.Println("响应成功: ", string(data))
    }
}

```

```
}
```

多语言 grpc + proto通信

首先：需要maven - grpc 插件

pom.xml

```
<?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>msbedu-rpc-example</artifactId>
        <groupId>org.example</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>grpc-client</artifactId>
    <dependencies>
        <!-- grpc + protobuf -->
        <dependency>
            <groupId>com.google.protobuf</groupId>
            <artifactId>protobuf-java</artifactId>
            <version>3.18.1</version>
        </dependency>
        <!--
https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java-
util -->
        <dependency>
            <groupId>com.google.protobuf</groupId>
            <artifactId>protobuf-java-util</artifactId>
            <version>3.18.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/io.grpc/grpc-all -->
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-all</artifactId>
            <version>1.11.0</version>
        </dependency>
    </dependencies>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>
```

```

<build>
  <extensions>
    <extension>
      <groupId>kr.motd.maven</groupId>
      <artifactId>os-maven-plugin</artifactId>
      <version>1.5.0.Final</version>
    </extension>
  </extensions>
  <plugins>
    <plugin>
      <groupId>org.xolstice.maven.plugins</groupId>
      <artifactId>protobuf-maven-plugin</artifactId>
      <version>0.5.0</version>
      <configuration>
        <protocArtifact>

com.google.protobuf:protoc:3.1.0:exe:${os.detected.classifier}
        </protocArtifact>
        <pluginId>grpc-java</pluginId>
        <pluginArtifact>

io.grpc:protoc-gen-grpc-
java:1.11.0:exe:${os.detected.classifier}
        </pluginArtifact>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>compile</goal>
            <goal>compile-custom</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</project>

```

将proto文件放到项目src/main/proto目录然后点击:

TestGrpcProto

```
import com.OrderProto;
import com.SearchServiceGrpc;
import com.google.protobuf.InvalidProtocolBufferException;
import com.google.protobuf.util.JsonFormat;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import com.SearchServiceGrpc.SearchServiceBlockingStub;

import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.concurrent.TimeUnit;

/**
 * @Auth:MCA-ShenZhuan
 * @Description: TODO
 * @Date:2022/7/12 14:38
 */
public class TestGrpcProto {
    public static void main(String[] args) {
        try {
            // 生成订单对象
            OrderProto.Order.Builder orderBuilder =
            OrderProto.Order.newBuilder();
            orderBuilder.setId(1);
            orderBuilder.setAddress("长沙");
            OrderProto.Order order1 = orderBuilder.build();
            // proto对象
            System.out.println("The order object is:\n" + order1);
            // 序列化
            byte[] orderByte = order1.toByteArray();
            System.out.println("The order after encode is:\n" +
            Arrays.toString(orderByte));
            // 反序列化
            OrderProto.Order newOrder =
            OrderProto.Order.parseFrom(orderByte);
            System.out.println("The Order after decode is:\n" +
            newOrder);
            // 转换json
            System.out.println("The Order json format is:\n" +
            JsonFormat.printer().print(order1));
            //服务端 Server server =
            ServerBuilder.forPort(8081).addService(new
            SearchServiceGrpc.SearchServiceImplBase().bindService());
            //客户端 grpc通信
            TestGrpcProto clientMain = new
            TestGrpcProto("127.0.0.1", 8082);
```

```

        OrderProto.OrderResp helloReply =
clientMain.blockingStub.searchOrder(OrderProto.OrderRequest.newBuilder()
    .setActivityType(1).build());
        System.out.println("The OrderResp after decode is:\n" +
helloReply);
        //msg中文处理
        //System.out.println(new
String(helloReply.getMsg().getBytes(StandardCharsets.UTF_8),
StandardCharsets.UTF_8));
        // 反序列化
        OrderProto.OrderResp strReply =
OrderProto.OrderResp.parseFrom(helloReply.toByteArray());
        // 转换json
        System.out.println("The OrderResp json format is:\n" +
JsonFormat.printer().print(strReply));
        clientMain.shutdown();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private final SearchServiceGrpc.SearchServiceBlockingStub
blockingStub;
private final ManagedChannel managedChannel;

public TestGrpcProto(String ip,int port) {
    //usePlaintext表示明文传输, 否则需要配置ssl
    this.managedChannel =
ManagedChannelBuilder.forAddress(ip,port).usePlaintext(true).build();
    this.blockingStub =
SearchServiceGrpc.newBlockingStub(this.managedChannel);//new
SearchServiceBlockingStub(this.managedChannel);
    System.out.println("Connected to server at "+ip+": "+port);
    Runtime.getRuntime().addShutdownHook(new Thread(() -> {
        try {
            this.managedChannel.shutdownNow().awaitTermination(5,
TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace(System.err);
        }
        System.err.println("Client shut down.");
    }));
}

private void shutdown() throws InterruptedException {
    this.managedChannel.shutdown().awaitTermination(5L,
TimeUnit.SECONDS);
}
}

```

Go-Micro + grpc + proto综合应用

说明:

go-micro各个版本之间的兼容性问题一直被诟病，前几年基本用的v2

go-micro之后分流两个分支:

一个延续了go-micro，只不过转到了其公司CEO的个人Github仓库中，访问地址:

asim/go-micro: A Go microservices framework (github.com)

一个转向了云原生方向，名字叫Micro，访问地址:

micro/micro: API first cloud platform (github.com)

不过都还是开源的，当前的许可证都是Apache 2.0，不是某些人说的不能商用了，当然无法保证以后不会改许可证。

我们今天先带大家用最新的go-micro 版本v4开发gRPC + Protobuf 服务的方式。不排除部分方法兼容性调整问题

gomicro github地址:

<https://github.com/micro/go-micro>

下载:

```
go get github.com/asim/go-micro/plugins/client/grpc/v4
```

```
go get github.com/asim/go-micro/plugins/server/grpc/v4
```

```
go get go-micro.dev/v4
```

重写一个 hello.proto

```

syntax = "proto3";
option go_package = "/proto";
package proto;
message HiRequest {
    string Name = 1;
}
message HiResponse {
    string Name = 1;
}
service Hello {
    rpc Hi (HiRequest) returns (HiResponse);
}

```

生成 gomicro 文件

`protoc --proto_path=. --go_out=plugins=grpc,paths=source_relative:. hello.proto`

`protoc --go_out=. --go_opt=paths=source_relative --micro_out=. --micro_opt=paths=source_relative hello.proto`

查看是否生成 `hello.pb.go` 以及 `hello.pb.micro.go`

代码：server端及client端

```

package main

import (
    "context"
    "fmt"
    pro "msb_gomicro/proto"
    "time"

    "github.com/asim/go-micro/plugins/server/grpc/v4"
    "go-micro.dev/v4"
)

type Hello struct{}

func (hello *Hello) Hi(cxt context.Context, req *pro.HiRequest, resp *pro.HiResponse) error {
    fmt.Println("req :", req.Name)
    resp.Name = "resp > " + req.Name
    return nil
}

func main() {
    grpcServer := grpc.NewServer()
    rpcServer := micro.NewService(

```

```

        micro.RegisterTTL(time.Second*30),
        micro.RegisterInterval(time.Second*10),
        micro.Server(grpcServer),
        micro.Name("msb-shenzhuan"),
        micro.Address("127.0.0.1:8081"),
        micro.Version("lasted"),
    )
    pro.RegisterHelloHandler(rpcServer.Server(), &Hello{})
    if err := rpcServer.Run(); err != nil {
        fmt.Println("err :", err)
    }
}

```

client端

```

/**
 * @Auth:ShenZ
 * @Description:
 * @CreateDate:2022/07/16 13:26:12
 */
package main

import (
    "bufio"
    "context"
    "fmt"
    pro "msb_gomicro/proto"
    "os"

    grpc "github.com/asim/go-micro/plugins/client/grpc/v4"
    "go-micro.dev/v4"
)

func main() {
    server := micro.NewService(
        micro.Client(grpc.NewClient())
    )
    server.Init()
    client := pro.NewHelloService("msb-shenzhuan", server.Client())
    resp, err := client.Hi(context.TODO(), &pro.HiRequest{Name:
"ShenZhuan"})
    if err != nil {
        fmt.Println("err:", err)
    }
    fmt.Println("resived :", resp)
    fmt.Println("输入回车结束程序！")
    in := bufio.NewReader(os.Stdin)
    _, _ = in.ReadLine()
}

```

```
}
```

照葫芦画瓢

server

```
type Order struct{}

func (order *Order) SearchOrder(cxt context.Context, req
*pro.OrderRequest, resp *pro.OrderResp) error {
    reqType := req.ActivityType //获取请求参数
    fmt.Println("接收到客户端请求: ", req)
    if reqType == 1 { //数据库操作
        resp.DiscountAmount = 100
        resp.Code = "200"
        resp.Msg = "查询成功"
    } else {
        resp.DiscountAmount = 0
        resp.Code = "500"
        resp.Msg = "查询失败"
        return errors.New("not found this Order!")
    }
    return nil
}
```

然后在 main 里面加入

```
pro.RegisterSearchServiceHandler(rpcServer.Server(), &Order{})
```

client

```
client := pro.NewSearchService("msb-shenzhuan", server.Client())
resp, err := client.SearchOrder(context.TODO(),
&pro.OrderRequest{ActivityType: 1})
```

注册与发现

为什么要：因为一套微服务架构中有很多个服务需要管理,也就是说会有很多对grpc。如果一一对应的进行管理会很繁琐所以我们需要有一个管理发现的机制。

常用的服务注册与发现框架：Consul, Eureka, ZooKeeper, etcd

Feature	Consul	Zookeeper	Etcd	Eureka
服务健康检查	服务状态, 内存, 硬盘等	(弱)长连接, keepalive	连接心跳	可配支持
多数据中心	支持	—	—	—
kv存储服务	支持	支持	支持	—
一致性	raft	paxos	raft	—
CAP定理	CA	CP	CP	AP
使用接口(多语言能力)	支持http和dns	客户端	http/grpc	http (sidecar)
watch支持	全量/支持long polling	支持	支持 long polling	支持 long polling/大部分增量
自身监控	metrics	—	metrics	metrics
安全	acl/https	acl	https支持 (弱)	—
Spring Cloud集成	已支持	已支持	已支持	已支持

注册中心 Consul 基本介绍

Consul 是 HashiCorp 公司推出的开源工具，用于实现分布式系统的服务发现与配置。Consul 使用 Go 语言编写，因此具有天然可移植性（支持Linux、windows和Mac OS X）。Consul采用主从模式的设计，使得集群的数量可以大规模扩展，集群间通过RPC的方式调用(HTTP和DNS)。

Consul 内置了服务注册与发现框架、分布一致性协议实现、健康检查、Key/Value 存储、多数据中心方案，不再需要依赖其他工具（比如 ZooKeeper 等），使用起来也较为简单。

Consul 遵循CAP原理中的CP原则，保证了强一致性和分区容错性，且使用的是Raft算法，比zookeeper使用的Paxos算法更加简单。虽然保证了强一致性，但是可用性就相应下降了，例如服务注册的时间会稍长一些，因为 Consul 的 raft 协议要求必须过半数的节点都写入成功才认为注册成功；在leader挂掉了之后，重新选举出leader之前会导致Consul 服务不可用。

是一种服务网格解决方案

提供具有服务发现，配置和分段功能的全功能控制平台

附带一个简单的内置代理，开箱即用

关键功能：

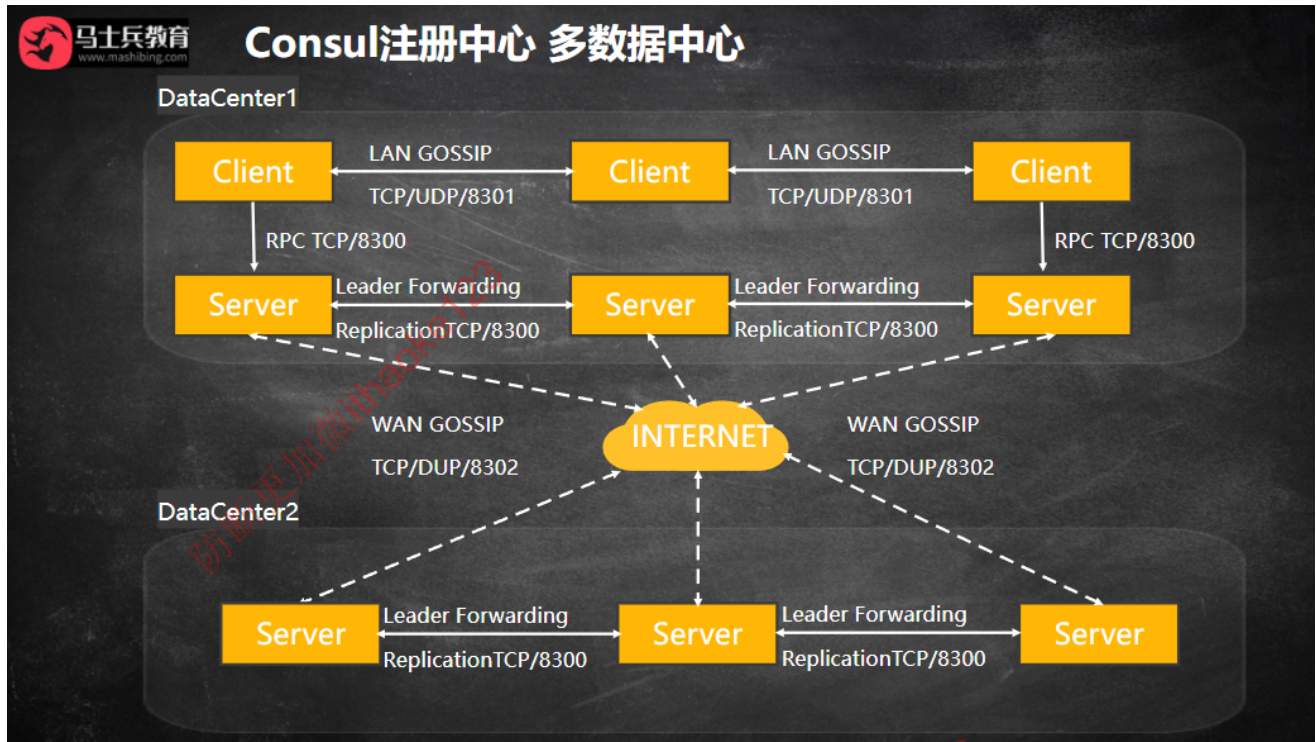
服务注册与发现，

运行状态检测，

KV存储（包括动态配置，功能标记，协调，选举等），

安全服务通信，

多数据中心（不同云）



两个重要的协议：

LAN Pool 局域网池 Client & Server , Server & Server

WAN Pool 广域网池 全局唯一 不同数据中心的交互

Consul镜像下载

\$ docker pull consul # 默认拉取latest

\$ docker pull consul:1.6.1 # 拉取指定版本

运行：

```
docker run -d -p 8500:8500 --restart=always --name=consul consul:1.6.1 agent -server -bootstrap -ui -node=1 -client='0.0.0.0'
```

说明：

- agent: 表示启动 Agent 进程。
- server: 表示启动 Consul Server 模式
- client: 表示启动 Consul Client 模式。
- bootstrap: 表示这个节点是 Server-Leader，每个数据中心只能运行一台服务器。技术角度上讲 Leader 是通过 Raft 算法选举的，但是集群第一次启动时需要一个引导 Leader，在引导集群后，建议不要使用此标志。
- ui: 表示启动 Web UI 管理器，默认开放端口 8500，所以上面使用 Docker 命令把 8500 端口对外开放。
- node: 节点的名称，集群中必须是唯一的，默认是该节点的主机名。
- client: consul 服务监听地址，这个地址提供 HTTP、DNS、RPC 等服务，默认是 127.0.0.1 所以不对外提供服务，如果你要对外提供服务改成 0.0.0.0
- join: 表示加入到某一个集群中去。如：-join=192.168.0.11

关闭防火墙：8500

systemctl status firewalld

\$ systemctl start firewalld

\$ systemctl stop firewalld

\$ firewall-cmd --list-ports

\$ firewall-cmd --zone=public --add-port=2181/tcp --permanent

\$ firewall-cmd --reload

注意：

如果是阿里云服务器，需要设置安全组：

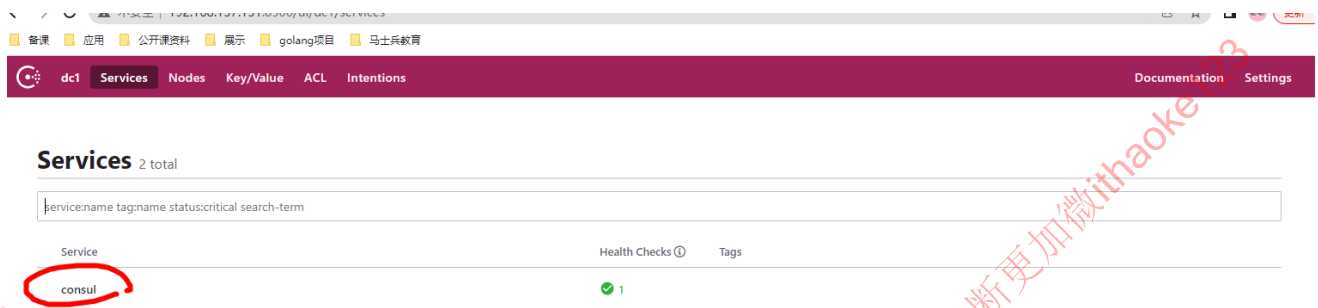
来到实例管理页面，点击更多，点击网络和安全组，点击安全组配置。

测试：<http://192.168.137.131:8500>

<http://192.168.137.131:8500/ui/dc1/services>

说明：

1. services: 放置服务
2. nodes: 放置 consul 节点
3. key/value: 放置一些配置信息
4. dc1: 配置数据中心



将上一节的服务加入注册中心

默认会找: github.com/micro/go-plugins/config/source/consul

v4需要: `go get github.com/asim/go-micro/plugins/registry/consul/v4`

`go mod tidy`

导包: `consul "github.com/asim/go-micro/plugins/registry/consul/v4"`
`"go-micro.dev/v4/registry"`

关键代码:

//注册到consul

```
consulReg := consul.NewRegistry(func(options *registry.Options) {  
    options.Addrs = []string{"192.168.137.131:8500"}  
})
```

`rpcServer := micro.NewService(里面加入`

`micro.Registry(consulReg),`

完整代码:

```
package main  
  
import (  
    "context"  
    "errors"  
    "fmt"  
    pro "msb_gomicro/proto"  
    "time"  
  
    consul "github.com/asim/go-micro/plugins/registry/consul/v4"  
    "github.com/asim/go-micro/plugins/server/grpc/v4"
```

```

    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

type Hello struct{}

func (hello *Hello) Hi(cxt context.Context, req *pro.HiRequest, resp
*pro.HiResponse) error {
    fmt.Println("req :", req.Name)
    resp.Name = "resp > " + req.Name
    return nil
}

type Order struct{}

func (order *Order) SearchOrder(cxt context.Context, req
*pro.OrderRequest, resp *pro.OrderResp) error {
    reqType := req.ActivityType //获取请求参数
    fmt.Println("接收到客户端请求: ", req)
    if reqType == 1 { //数据库操作
        resp.DiscountAmount = 100
        resp.Code = "200"
        resp.Msg = "查询成功"
    } else {
        resp.DiscountAmount = 0
        resp.Code = "500"
        resp.Msg = "查询失败"
        return errors.New("not found this Order!")
    }
    return nil
}

// //设置配置中心
// func GetConsulConfig(host string, port int64, prefix string)
// (config.Config, error) {
//     consulSource := consul.NewSource(
//         // //设置配置中心地址
//         consul.WithAddress(host+": "+strconv.FormatInt(port, 10)),
//         // //设置前缀
//         consul.WithPrefix(prefix),
//         // //是否可略前缀
//         consul.StripPrefix(true),
//     )
//     // //初始化配置
//     config, err := config.NewConfig()
//     if err != nil {
//         return config, err
//     }

```

```
// //加载配置
// err = config.Load(consulSource)
// return config, err
// }

func main() {
    grpcServer := grpc.NewServer()
    //注册到consul
    consulReg := consul.NewRegistry(func(options *registry.Options) {
        options.Addrs = []string{"192.168.137.131:8500"}
    })
    rpcServer := micro.NewService(
        micro.RegisterTTL(time.Second*30),
        micro.RegisterInterval(time.Second*10),
        micro.Server(grpcServer),
        micro.Name("msb-shenzhuan1"),
        micro.Address("127.0.0.1:8081"),
        micro.Version("lasted"),
        //注册consul
        micro.Registry(consulReg),
    )
    pro.RegisterHelloHandler(rpcServer.Server(), &Hello{})
    pro.RegisterSearchServiceHandler(rpcServer.Server(), &Order{})
    if err := rpcServer.Run(); err != nil {
        fmt.Println("err :", err)
    }
}
```

客户端

可能异常:

To upgrade to the versions selected by go 1.

go mod tidy -go=1.16 && go mod tidy

If reproducibility with go 1.16 is not needed

go mod tidy -compat=1.17

For other options, see:

解决方案:

go mod tidy -go="1.16"

常见通信异常:

2022/07/19 14:59:49

```
{"id":"go.micro.client.transport","code":500,"detail":"malformed HTTP response
"\x00\x00\x06\x04\x00\x00\x00\x00\x00\x00\x05\x00\x00@\x00""status":"Internal Server Error"}
```

客户端服务端协议一致性。

解决方案: server 与 client 统一不用grpc

*grpcStream does not implement client.Stream (missing CloseSend method)

如果都用v4 grpc的,客户端会提示 没有重写 client stream的 关闭方法

注意:讲服务端的 注释//grpcServer := grpc.NewServer() //micro.Server(grpcServer),

```
package main

import (
    "context"
    "log"
    pro "msb_gomicro/proto"
    "net/http"
    "strconv"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"
    _ "github.com/asim/go-micro/plugins/server/grpc/v4"

    "github.com/gin-gonic/gin"
    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

func main() {
    router := gin.Default()
    router.GET("/", func(c *gin.Context) {
        //grpcServer := grpc.NewServer()
        //注册到consul
        consulReg := consul.NewRegistry(func(options *registry.Options) {
            {
                options.Addrs = []string{"192.168.137.131:8500"}
            }
        })
        rpcServer := micro.NewService(
            micro.Registry(consulReg),
        )
        client := pro.NewSearchService("msb-shenzhuan1",
            rpcServer.Client())
        ActivityType, _ :=
            strconv.Atoi(c.Request.FormValue("ActivityType"))
        resp, err := client.SearchOrder(context.TODO(),
            &pro.OrderRequest{ActivityType: int32(ActivityType)})

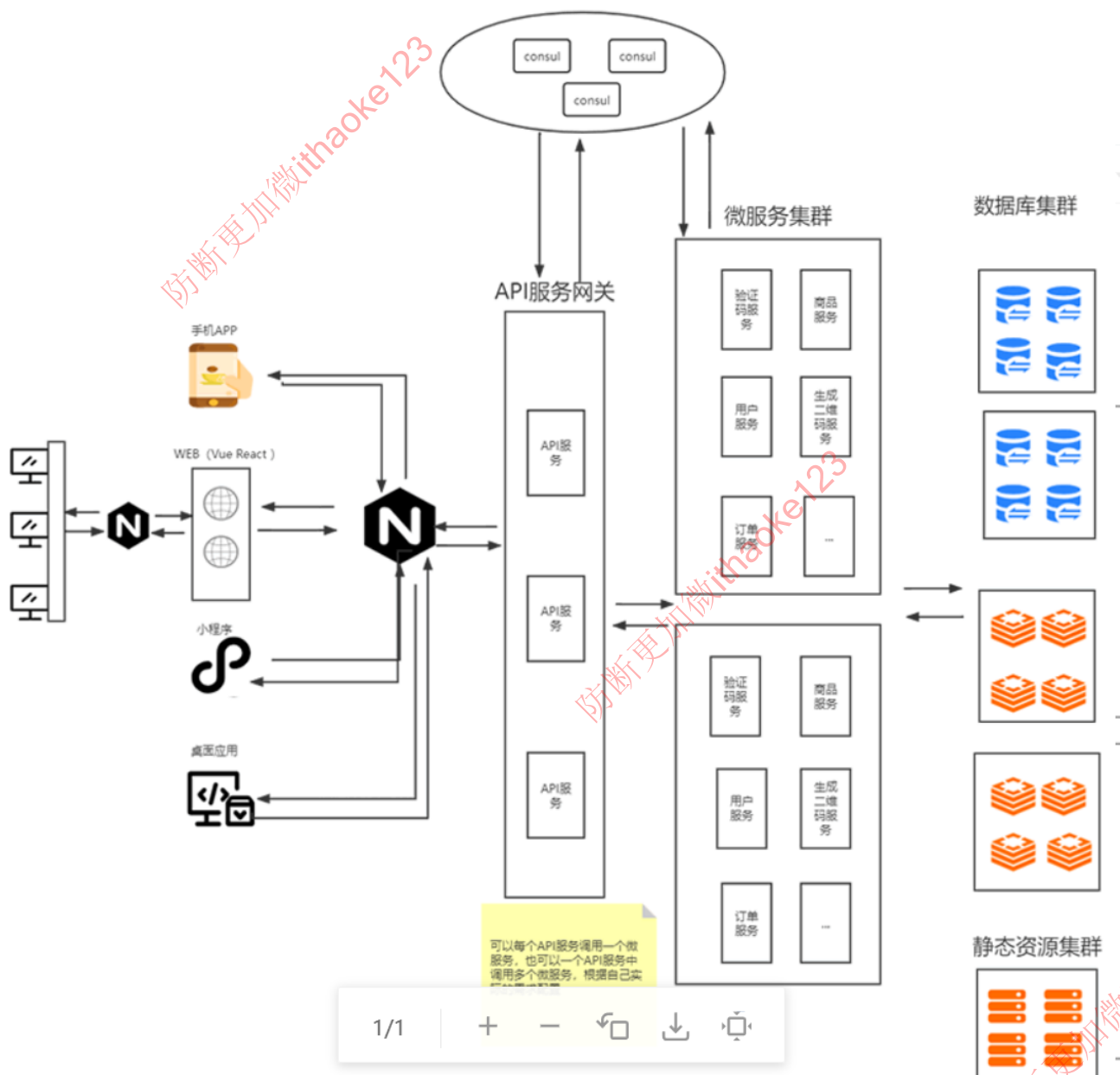
        // resp, err := client.SearchOrder(context.TODO(),
            &pro.OrderRequest{ActivityType: 1})
    })
}
```

```

    if err != nil {
        log.Println(err.Error())
        c.String(http.StatusBadRequest, "search failed !")
        return
    }
    c.String(http.StatusOK, resp.Msg)
}
router.Run(":6666")
}

```

严选商城系统架构图



生产环境: <https://you-app.mashibing.com/>

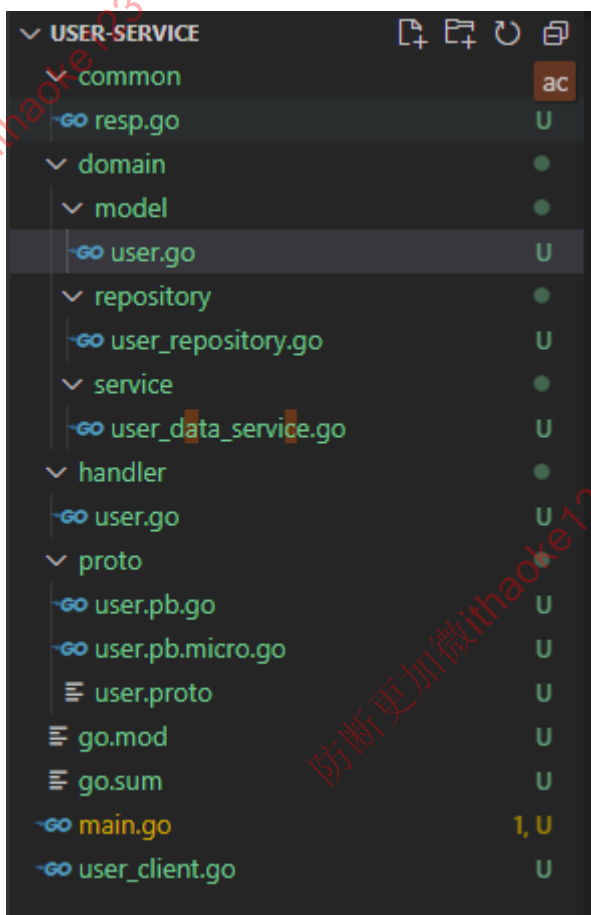
用户 商品 购物车 订单 等微服务

```
protoc --proto_path=. --go_out=plugins=grpc,paths=source_relative:. *.proto
```

```
protoc --go_out=. --go_opt=paths=source_relative --micro_out=. --  
micro_opt=paths=source_relative *.proto
```

用户服务:

目录结构:



model / user.go

```
package model

type User struct {
    Avatar          string
    ClientId        int32
    EmployeeId      int32
```

```

    Nickname      string
    Phone         string
    SessionId     string
    Token         string
    TokenExpireTime string
    UnionId       string
}

func (table *User) TableName() string {
    return "user"
}

```

repository/user_repository.go

```

package repository

import (
    "errors"
    "gouser/domain/model"

    "gorm.io/gorm"
)

/**
    int32 clientId = 1;
    string phone = 2;
    int32 systemId = 3;
    string verificationCode = 4;
**/
//接口
type IUserRepository interface {
    Login(int32, string, int32, string) (*model.User, error)
}

//创建实例
func NewUserRepository(db *gorm.DB) IUserRepository {
    return &UserRepository{mysqlDB: db}
}

//数据DB
type UserRepository struct {
    mysqlDB *gorm.DB
}

//重写接口方法
func (u *UserRepository) Login(clientId int32, phone string, systemId
int32, verificationCode string) (user *model.User, err error) {
    user = &model.User{}

```



```

        if clientId == 0 && systemId == 0 && verificationCode == "6666" {
            return user, u.mysqlDB.Where("phone = ? ",
phone).Find(user).Error
        } else {
            return user, errors.New("参数不匹配")
        }
    }
}

```

service /user_data_service.go

```

package service

import (
    "gouser/domain/model"
    "gouser/domain/repository"
)

type IUserDataService interface {
    Login(int32, string, int32, string) (*model.User, error)
}

type UserDataService struct {
    userRepository repository.IUserRepository
}

func NewUserDataService(userRepository repository.IUserRepository)
IUserDataService {
    return &UserDataService{userRepository: userRepository}
}

//重写接口方法
func (u *UserDataService) Login(clientId int32, phone string, systemId
int32, verificationCode string) (user *model.User, err error) {

    return u.userRepository.Login(clientId, phone, systemId,
verificationCode)
}

/*  clientId, _ := strconv.Atoi(c.Request.FormValue("clientId"))
phone := c.Request.FormValue("phone")
systemId, _ := strconv.Atoi(c.Request.FormValue("systemId"))
verificationCode := c.Request.FormValue("verificationCode")
*/

```

proto / user.proto

```

/**
 * @Auth:ShenZ
 * @Description:
 */
syntax = "proto3";    // 版本号
option go_package=".:/proto";    //参数1 表示生成到哪个目录 , 参数2 表示生成的
文件的package
package proto ;    //默认在哪个包
//结构体
/**
    "avatar": "",
    "clientId": 0,
    "employeeId": 0,
    "id": 0,
    "nickname": "",
    "phone": "",
    "sessionId": "",
    "systemId": 0,
    "token": "",
    "tokenExpireTime": "",
    "unionId": ""
**/
message User {
    string avatar = 1;
    int32 clientId = 2;
    int32 employeeId = 3;
    string nickname = 4;
    string phone = 5;
    string sessionId = 6;
    string token = 7;
    string tokenExpireTime = 8;
    string unionId = 9;
}
/**也可写这里
{
    "clientId": 0,
    "phone": "",
    "systemId": 0,
    "verificationCode": ""
}
**/
//请求 request struct
message LoginRequest {
    int32 clientId = 1;
    string phone = 2;
    int32 systemId = 3;
    string verificationCode = 4;
}
//响应 resp struct

```

防断更加微ithaoke123

防

```
text"
"
ser/domain/model"
ser/domain/service"
ser/proto"
```

```
, systemId int32, v
text, loginRequest
o.LoginResp) error +
ce.Login(loginReques
t.SystemId,
```

防斷更加微ithaoke123

```

    resp.User.ClientId = userModel.ClientId
    resp.User.EmployeeId = userModel.EmployeeId
    resp.User.Nickname = userModel.Nickname
    resp.User.SessionId = userModel.SessionId
    resp.User.Phone = userModel.Phone
    resp.User.TokenExpireTime = userModel.TokenExpireTime
    resp.User.UnionId = userModel.UnionId
    return resp
}

```

工具类 comm/resp.go

```

package common

import (
    "encoding/json"
    "fmt"
    "net/http"
)

type H struct {
    Code          int
    Message       string
    TraceId       string
    Data          interface{}
    Rows          interface{}
    Total         interface{}
    SkyWalkingDynamicField string
}

func Resp(w http.ResponseWriter, code int, data interface{}, message string) {
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    h := H{
        Code:      code,
        Data:      data,
        Message:   message,
    }
    ret, err := json.Marshal(h)
    if err != nil {
        fmt.Println(err)
    }
    w.Write(ret)
}

```

```

func RespList(w http.ResponseWriter, code int, data interface{}, message
string, rows interface{}, total interface{}, skyWalkingDynamicField
string) {
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    h := H{
        Code:          code,
        Data:           data,
        Message:        message,
        Rows:           rows,
        Total:          total,
        SkyWalkingDynamicField: skyWalkingDynamicField,
    }
    ret, err := json.Marshal(h)
    if err != nil {
        fmt.Println(err)
    }
    w.Write(ret)
}

/**
200 OKLoginSuccessVO
201 Created
401 Unauthorized
403 Forbidden
404 Not Found
**/

func RespOK(w http.ResponseWriter, code int, data interface{}, message
string) {
    Resp(w, 200, data, message)
}

func RespCreated(w http.ResponseWriter, code int, data interface{},
message string) {
    Resp(w, 201, data, message)
}

func RespListOK(w http.ResponseWriter, code int, data interface{},
message string, rows interface{}, total interface{},
skyWalkingDynamicField string) {
    RespList(w, 200, data, message, rows, total, skyWalkingDynamicField)
}

func RespListFail(w http.ResponseWriter, code int, data interface{},
message string, rows interface{}, total interface{},
skyWalkingDynamicField string) {
    RespList(w, 500, data, message, rows, total, skyWalkingDynamicField)
}

```

服务端注册

```
package main

import (
    "gouser/domain/repository"
    "gouser/domain/service"
    "gouser/handler"
    "gouser/proto"
    "log"
    "os"
    "time"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
    "gorm.io/gorm/logger"

    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

func main() {
    // 1. consul注册中心
    consulReist := consul.NewRegistry(func(options *registry.Options) {
        options.Addrs = []string{"192.168.137.131:8500"}
    })
    repcService := micro.NewService(
        micro.RegisterTTL(time.Second*30),
        micro.RegisterInterval(time.Second*30),
        micro.Name("shop-user"),
        micro.Address(":8081"),
        micro.Version("v1"),
        micro.Registry(consulReist),
    )

    // 2. 初始化db
    newLogger := logger.New(
        log.New(os.Stdout, "\r\n", log.LstdFlags),
        logger.Config{
            SlowThreshold: time.Second,
            LogLevel:      logger.Info,
            Colorful:      true,
        },
    )
}
```

```

    db, err :=
gorm.Open(mysql.Open("root:mashibing123@tcp(8.142.25.43:3306)/user_center?charset=utf8mb4&parseTime=True&loc=Local"), &gorm.Config{Logger:
newLogger})
    if err != nil {
        log.Println("db err :", err)
    }
    //3.创建服务实例
    userDataService :=
service.NewUserDataService(repository.NewUserRepository(db))
    //4.注册handler
    proto.RegisterLoginHandler(repcService.Server(),
&handler.User{userDataService})

    //5.启动服务
    if err := repcService.Run(); err != nil {
        log.Println("start user service err :", err)
    }
}

```

客户端服务发现

```

package main

import (
    "context"
    "fmt"
    "gouser/proto"
    "log"
    "net/http"
    "strconv"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"

    "github.com/gin-gonic/gin"
    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

func main() {
    router := gin.Default()
    router.GET("/", func(c *gin.Context) {
        //注册到consul
        consulReg := consul.NewRegistry(func(options *registry.Options)
{

```

```

        options.Addr = []string{"192.168.137.131:8500"}
    })
    rpcServer := micro.NewService(
        micro.Registry(consulReg),
    )
    client := proto.NewLoginService("shop-user", rpcServer.Client())
    clientId, _ := strconv.Atoi(c.Request.FormValue("clientId"))
    phone := c.Request.FormValue("phone")
    systemId, _ := strconv.Atoi(c.Request.FormValue("systemId"))
    verificationCode := c.Request.FormValue("verificationCode")
    req := &proto.LoginRequest{
        ClientId:      int32(clientId),
        Phone:         phone,
        SystemId:      int32(systemId),
        VerificationCode: verificationCode,
    }
    resp, err := client.Login(context.TODO(), req)

    // resp, err := client.SearchOrder(context.TODO(),
    &proto.OrderRequest{ActivityType: 1})

    if err != nil {
        log.Println(err.Error())
        c.String(http.StatusBadRequest, "search failed !")
        return
    }
    fmt.Println(resp)
    c.String(http.StatusOK, "登录成功")
})
router.Run(":6666")
}

```

最后启动 服务 和客户端

调整 用户服务：

首先核对前端需要的响应结果 proto里面加入 id ，记得重新生成文件


```

message User {
    string avatar = 1;
    int32 clientId = 2;
    int32 employeeId = 3;
    string nickname = 4;
    string phone = 5;
    string sessionId = 6;
    string token = 7;
    string tokenExpireTime = 8;
    string unionId = 9;
    int32 id = 10;
}

```

client端移到 client包 登录失败的响应 修改

```

package main

import (
    "context"
    "gouser/common"
    "gouser/proto"
    "log"
    "strconv"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"

    "github.com/gin-gonic/gin"
    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

func main() {
    router := gin.Default()
    router.GET("/", func(c *gin.Context) {
        //注册到consul
        consulReg := consul.NewRegistry(func(options *registry.Options) {
            options.Addrs = []string{"192.168.137.131:8500"}
        })
        rpcServer := micro.NewService(
            micro.Registry(consulReg),
        )
        client := proto.NewLoginService("shop-user", rpcServer.Client())
        clientId, _ := strconv.Atoi(c.Request.FormValue("clientId"))
        phone := c.Request.FormValue("phone")
        systemId, _ := strconv.Atoi(c.Request.FormValue("systemId"))
        verificationCode := c.Request.FormValue("verificationCode")
    })
}

```

```

req := &proto.LoginRequest{
    ClientId:      int32(clientId),
    Phone:         phone,
    SystemId:      int32(systemId),
    VerificationCode: verificationCode,
}
resp, err := client.Login(context.TODO(), req)

if err != nil {
    log.Println(err.Error())
    //c.String(http.StatusBadRequest, "search failed !")
    common.RespFail(c.Writer, resp, "登录失败")
    return
}
common.RespOK(c.Writer, resp, "登录成功")
})
router.Run(":6666")
}

```

md5加密工具类

```

package common

import (
    "crypto/md5"
    "encoding/hex"
    "fmt"
    "strings"
)

//小写
func Md5Encode(data string) string {
    h := md5.New()
    h.Write([]byte(data))
    tempStr := h.Sum(nil)
    return hex.EncodeToString(tempStr)
}

//大写
func MD5Encode(data string) string {
    return strings.ToUpper(Md5Encode(data))
}

//加密
func MakePassword(plainpwd, salt string) string {
    return Md5Encode(plainpwd + salt)
}

```

//解密

```
func ValidPassword(plainpwd, salt string, password string) bool {  
    md := Md5Encode(plainpwd + salt)  
    fmt.Println(md + "          " + password)  
    return md == password  
}
```

resp.go 的修改

```
package common  
  
import (  
    "encoding/json"  
    "fmt"  
    "net/http"  
)  
  
type H struct {  
    Code          string  
    Message       string  
    TraceId      string  
    Data          interface{}  
    Rows          interface{}  
    Total         interface{}  
    SkyWalkingDynamicField string  
}  
  
func Resp(w http.ResponseWriter, code string, data interface{}, message string) {  
    w.Header().Set("Content-Type", "application/json")  
    w.WriteHeader(http.StatusOK)  
    h := H{  
        Code:    code,  
        Data:    data,  
        Message: message,  
    }  
    ret, err := json.Marshal(h)  
    if err != nil {  
        fmt.Println(err)  
    }  
    w.Write(ret)  
}  
  
func RespList(w http.ResponseWriter, code string, data interface{}, message string, rows interface{}, total interface{}, skyWalkingDynamicField string) {
```

```

w.Header().Set("Content-Type", "application/json")
w.WriteHeader(http.StatusOK)
h := H{
    Code:           code,
    Data:           data,
    Message:        message,
    Rows:           rows,
    Total:          total,
    SkyWalkingDynamicField: skyWalkingDynamicField,
}
ret, err := json.Marshal(h)
if err != nil {
    fmt.Println(err)
}
w.Write(ret)
}

/**
200 OKLoginSuccessVO
201 Created
401 Unauthorized
403 Forbidden
404 Not Found
**/

func RespOK(w http.ResponseWriter, data interface{}, message string) {
    Resp(w, "SUCCESS", data, message)
}

func RespFail(w http.ResponseWriter, data interface{}, message string) {
    Resp(w, "TOKEN_FAIL", data, message)
}

func RespListOK(w http.ResponseWriter, data interface{}, message string,
rows interface{}, total interface{}, skyWalkingDynamicField string) {
    RespList(w, "SUCCESS", data, message, rows, total,
skyWalkingDynamicField)
}

func RespListFail(w http.ResponseWriter, data interface{}, message
string, rows interface{}, total interface{}, skyWalkingDynamicField
string) {
    RespList(w, "TOKEN_FAIL", data, message, rows, total,
skyWalkingDynamicField)
}

```

model/user.go 跟数据库字段 核对 没有的删了 not null 给默认值

```

package model

import "time"

```

```

type User struct {
    //gorm.Model
    ID          int32
    Avatar      string `gorm:"default:'https://msb-edu-dev.oss-cn-beijing.aliyuncs.com/default-heading.png'"`
    ClientId    int32  `gorm:"default:1"`
    Nickname    string `gorm:"default:'随机名称'"`
    Phone       string
    Password    string `gorm:"default:'1234'"`
    SystemId    string `gorm:"default:1"`
    LastLoginTime time.Time
    CreateTime  time.Time
    IsDeleted   int32  `gorm:"default:0"`
    UnionId     string `gorm:"default:'1'"`
}

func (table *User) TableName() string {
    return "user"
}

```

repository 里面 加入 判断没有查到就新建用户

```

package repository

import (
    "errors"
    "gouser/domain/model"

    "gorm.io/gorm"
)

/**
    int32 clientId = 1;
    string phone = 2;
    int32 systemId = 3;
    string verificationCode = 4;
**/
//接口
type IUserRepository interface {
    Login(int32, string, int32, string) (*model.User, error)
}

//创建实例
func NewUserRepository(db *gorm.DB) IUserRepository {
    return &UserRepository{mysqlDB: db}
}

//数据DB

```

```

type UserRepository struct {
    mysqlDB *gorm.DB
}

//重写接口方法
func (u *UserRepository) Login(clientId int32, phone string, systemId int32, verificationCode string) (user *model.User, err error) {
    user = &model.User{}
    if clientId == 0 && systemId == 0 && verificationCode == "6666" {
        u.mysqlDB.Where("phone = ? ", phone).Find(user)
        //未找到就注册一个
        if user.ID == 0 {
            user.Phone = phone
            u.mysqlDB.Create(&user)
            //u.mysqlDB.Select("Nickname", "Avatar", "Phone",
            "ClientId").Create(&user)
        }
        return user, nil
        //return user, u.mysqlDB.Where("phone = ? ",
        phone).Find(user).Error
    } else {
        return user, errors.New("参数不匹配")
    }
}

```

handler 修改 根据查询的结果拼接 响应的报文

```

package handler

import (
    "context"
    "fmt"
    "gouser/common"
    "gouser/domain/model"
    "gouser/domain/service"
    "gouser/proto"
    "log"
    "time"

)

type User struct {
    UserDataService service.IUserDataService
}

// 登录 (clientId int32, phone string, systemId int32, verifi
func (u *User) Login(ctx context.Context, loginRequest
*proto.LoginRequest, loginResp *proto.LoginResp) error {

```

Withaoke123

```
request.ClientId,
```

51

unionId

防断更加微jithaoke123

```
go get github.com/spf13/viper/remote
```

```
package common

import (
```

```
package common

import (
```

```

        "log"
        "os"
        "time"

        "github.com/spf13/viper"
        _ "github.com/spf13/viper/remote"
        "gorm.io/driver/mysql"
        "gorm.io/gorm"
        "gorm.io/gorm/logger"
    )

func GetConsulConfig(url string, fileKey string) (*viper.Viper, error) {
    conf := viper.New()
    conf.AddRemoteProvider("consul", url, fileKey)
    conf.SetConfigType("json")
    err := conf.ReadRemoteConfig()
    if err != nil {
        log.Println("viper conf err :", err)
    }
    return conf, nil
}

/**
{
    "host": "192.168.137.131",
    "port": "3306",
    "user": "root",
    "pwd": "mashibing123",
    "database": "user_center"
}
**/

// type MySQLConfig struct {
//     Host      string `json:"host"`
//     Post      string `json:"port"`
//     User      string `json:"user"`
//     Pwd       string `json:"pwd"`
//     Database  string `json:"database"`
// }

func GetMysqlFromConsul(vip *viper.Viper) (db *gorm.DB, err error) {
    newLogger := logger.New(
        log.New(os.Stdout, "\r\n", log.LstdFlags),
        logger.Config{
            SlowThreshold: time.Second,
            LogLevel:      logger.Info,
            Colorful:      true,
        },
    )

```



```

    str := vip.GetString("user") + ":" + vip.GetString("pwd") + "@tcp("
+ vip.GetString("host") + ":" + vip.GetString("port") + ")/" +
vip.GetString("database") + "?charset=utf8mb4&parseTime=True&loc=Local"
    db, errr := gorm.Open(mysql.Open(str), &gorm.Config{Logger:
newLogger}) // "root:mashibing123@tcp(8.142.25.43:3306)/user_center?
charset=utf8mb4&parseTime=True&loc=Local", &gorm.Config{Logger:
newLogger})
    if errr != nil {
        log.Println("db err :", errr)
    }

    return db, nil
}

```

然后修改 main.go

```

package main

import (
    "gouser/common"
    "gouser/domain/repository"
    "gouser/domain/service"
    "gouser/handler"
    "gouser/proto"
    "log"
    "time"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"
    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

const (
    consulStr = "http://192.168.137.131:8500"
    fileKey    = "mysql-user"
)

func main() {
    //0 配置中心
    consulConfig, err := common.GetConsulConfig(consulStr, fileKey)
    if err != nil {
        log.Println("consulConfig err :", err)
    }
    // 1.consul注册中心
    consulReist := consul.NewRegistry(func(options *registry.Options) {
        options.Addrs = []string{consulStr}
    })
    repcService := micro.NewService(

```

```

        micro.RegisterTTL(time.Second*30),
        micro.RegisterInterval(time.Second*30),
        micro.Name("shop-user"),
        micro.Address(":8081"),
        micro.Version("v1"),
        micro.Registry(consulReist),
    )
    //2.初始化db
    db, _ := common.GetMysqlFromConsul(consulConfig)
    //3.创建服务实例
    userDataService :=
service.NewUserDataService(repository.NewUserRepository(db))
    //4.注册handler
    proto.RegisterLoginHandler(repcService.Server(),
&handler.User{userDataService})

    //5.启动服务
    if err := repcService.Run(); err != nil {
        log.Println("start user service err :", err)
    }
}

```

客户端优化:

```

package main

import (
    "context"
    "gouser/common"
    "gouser/proto"
    "log"
    "strconv"

    consul "github.com/asim/go-micro/plugins/registry/consul/v4"
    "go-micro.dev/v4/web"

    "github.com/gin-gonic/gin"
    "go-micro.dev/v4"
    "go-micro.dev/v4/registry"
)

//获取远程服务的客户端
func getClient() proto.LoginService {
    //注册到consul
    consulReg := consul.NewRegistry(func(options *registry.Options) {
        options.Addrs = []string{"192.168.137.131:8500"}
    })

```

```

rpcServer := micro.NewService(
    micro.Registry(consulReg),
)
return proto.NewLoginService("shop-user", rpcServer.Client())
}
func main() {
    router := gin.Default()

    router.Handle("GET", "toLogin", func(context *gin.Context) {
        context.String(200, "to Logging ....")
    })

    router.GET("/login", func(c *gin.Context) {
        //获取远程服务的客户端
        client := getClient()
        //获取页面参数
        clientId, _ := strconv.Atoi(c.Request.FormValue("clientId"))
        phone := c.Request.FormValue("phone")
        systemId, _ := strconv.Atoi(c.Request.FormValue("systemId"))
        verificationCode := c.Request.FormValue("verificationCode")
        //拼接请求信息
        req := &proto.LoginRequest{
            ClientId:      int32(clientId),
            Phone:          phone,
            SystemId:       int32(systemId),
            VerificationCode: verificationCode,
        }
        //远程调用服务
        resp, err := client.Login(context.TODO(), req)
        //根据响应做输出
        if err != nil {
            log.Println(err.Error())
            //c.String(http.StatusBadRequest, "search failed !")
            common.RespFail(c.Writer, resp, "登录失败")
            return
        }
        common.RespOK(c.Writer, resp, "登录成功")
    })

    service := web.NewService(
        web.Address(":8081"),
        web.Handler(router),
    )
    service.Run()
    //router.Run(":6666")
}

```

商品服务：

业务需求分析

根据页面请求以及Swagger文档：

1. mall/product/app/product/recommended
2. product/app/product/page?length=15&pageIndex=1
3. /product/app/product/42
4. product/sku?productId=42
5. product/app/productCategory/listCategoryAndProduct/1

接口文档：mall/product/app/product/byCategory

categoryNavigation

product/app/product/listSkuAndCommentCount/{productId}

~ product/app/product/page?length=15&pageIndex=1 OK

~ /app/product/recommended OK

页面请求 /app/product/recommended

```
{
  "code": "SUCCESS",
  "message": "操作成功",
  "traceId":
  "e56a82ffabab45df8d0d36bc22483ae3.14706.16589936352884607",
  "data": [
    {
      "skuId": 141,
      "attributeSymbolList": "",
      "name": "",
      "sellPrice": 78,
      "stock": 169
    }
  ],
  "skyWalkingDynamicField": null
}
```

~/mall/product/app/product/sku OK

~ /mall/product/app/product/{productId}

```
{
  "code": "SUCCESS",
  "message": "操作成功",
  "traceId":
  "e56a82ffabab45df8d0d36bc22483ae3.14705.16589934196324607",
  "data": {
    "records": [
      {
        "id": 100,
        "name": "C5现代简约小闹钟",
        "startingPrice": 42,
        "mainPicture": "https://msb-edu-prod.oss-cn-
beijing.aliyuncs.com/mall-product/product/198b9969-1ca6-4560-827a-
a642acfbdb053.png",
        "labelList": [ ],
        "singleBuyLimit": null,
        "isEnabled": null,
        "productType": null
      },
      {}
    ],
    "total": 62,
    "size": 15,
    "current": 2,
    "orders": [ ],
    "optimizeCountSql": true,
    "searchCount": true,
    "countId": null,
    "maxLimit": null,
    "pages": 5
  },
  "skyWalkingDynamicField": null
}
```

product/app/product/page?length=15&pageIndex=1

```
{
  "code": "SUCCESS",
  "message": "操作成功",
  "traceId":
  "e56a82ffabab45df8d0d36bc22483ae3.14704.16590816131230013",
  "data": {
    "records": [
      {
```

```

        "id": 115,
        "name": "马歇尔MARSHALL STANMORE II 无线蓝牙音响家用复古重低音小
音箱",
        "startingPrice": 3300,
        "mainPicture": "https://msb-edu-prod.oss-cn-
beijing.aliyuncs.com/mall-product/product/d4f1f19e-2e90-4ac5-bbe3-
1eba02085a0f.jpg",
        "labelList": [ ],
        "singleBuyLimit": null,
        "isEnabled": null,
        "productType": null
    },
    {}},
],
"total": 63,
"size": 15,
"current": 1,
"orders": [ ],
"optimizeCountSql": true,
"searchCount": true,
"countId": null,
"maxLimit": null,
"pages": 5
},
"skyWalkingDynamicField": null
}

```

实体model

```

// product_sku.go

package model

/**
"attributeSymbolList": "",
    "name": "",
    "sellPrice": 0,
    "skuId": 0,
    "stock": 0
}
**/
type ProductSku struct {
    //gorm.Model
    SkuId          int32 `gorm:"column:id"`
    Name           string

```

```

        ProductId          int32 `gorm:"default:1"`
        AttributeSymbolList string
        SellPrice           float32
        Stock               int32 `gorm:"default:1"`
    }

    func (table *ProductSku) TableName() string {
        return "product_sku"
    }

//product.go
package model

import "time"

type Product struct {
    //gorm.Model
    ID          int32
    Name         string
    ProductType int32 `gorm:"default:1"`
    CategoryId  int32
    StartingPrice float32
    TotalStock  int32 `gorm:"default:'1234'"`
    MainPicture string `gorm:"default:1"`
    RemoteAreaPostage float32
    SingleBuyLimit int32
    IsEnable       int32 `gorm:"default:0"`
    Remark         string `gorm:"default:'1'"`
    CreateUser     int32 `gorm:"default:'1'"`
    CreateTime     time.Time
    UpdateUser     int32
    UpdateTime     time.Time
    IsDeleted      bool
}

func (table *Product) TableName() string {
    return "product"
}

```

然后请求proto

```

/**
 * @Auth:ShenZ
 * @Description: product page
 */
syntax = "proto3";    // 版本号

```

```

option go_package="./;proto";    //参数1 表示生成到哪个目录 ， 参数2 表示生成的
文件的package
package proto ;    //默认在哪个包

//结构体
/**
    "id": 115,
    "name": "马歇尔MARSHALL STANMORE II 无线蓝牙音响家用复古重低音小音
箱",
    "startingPrice": 3300,
    "mainPicture": "https://msb-edu-prod.oss-cn-
beijing.aliyuncs.com/mall-product/product/d4f1f19e-2e90-4ac5-bbe3-
1eba02085a0f.jpg",
    "labelList": [ ],
    "singleBuyLimit": null,
    "isEnabled": null,
    "productType": null
**/
message Product {
    int32 id = 1;
    string name = 2;
    int32 startingPrice = 3;
    string mainPicture = 4;
    map<string,string> labelList = 5;
    int32 singleBuyLimit = 6;
    string token = 7;
    bool isEnabled = 8;
    int32 productType = 9;
}
/**
前端请求信息
{
    "clientId": 0,
    "phone": "",
    "systemId": 0,
    "verificationCode": ""
}
**/
//请求 request struct
message PageReq {
    int32 length = 1;
    string pageIndex = 2;
}
//响应 resp struct
/**
**/
message PageResp{

```



```

    Product product = 1;
}
//RPC 服务 接口
service Page {
    //rpc 服务
    rpc Page (PageReq) returns (PageResp) {}
}

```

protoc --proto_path=. --go_out=plugins=grpc,paths=source_relative:.
product.proto

protoc --go_out=. --go_opt=paths=source_relative --micro_out=. --
micro_opt=paths=source_relative product.proto

Repository 和 service 层

```

package repository

import (
    "errors"
    "fmt"
    "goproduct/domain/model"

    "gorm.io/gorm"
)

/**
    int32 clientId = 1;
    string phone = 2;
    int32 systemId = 3;
    string verificationCode = 4;
**/
//接口
type IProductRepository interface {
    Page(int32, int32) ([]model.Product, error)
}

//创建实例
func NewProductRepository(db *gorm.DB) IProductRepository {
    return &ProductRepository{mysqlDB: db}
}

//数据DB
type ProductRepository struct {
    mysqlDB *gorm.DB
}

```

//重写接口方法

//product/app/product/page?length=15&pageIndex=1

```
func (u *ProductRepository) Page(length int32, pageIndex int32) (products []model.Product, err error) {  
    arr := make([]model.Product, length)  
    if length > 0 && pageIndex > 0 {  
        u.mysqlDB = u.mysqlDB.Limit(int(length)).Offset((int(pageIndex) - 1) * int(length))  
        if err := u.mysqlDB.Find(&arr).Error; err != nil {  
            fmt.Println("query product err :", err)  
        }  
        return arr, nil  
    }  
    return arr, errors.New("参数不匹配")  
}
```

//service

package service

```
import (  
    "goproduct/domain/model"  
    "goproduct/domain/repository"  
)
```

```
type IProductDataService interface {  
    Page(int32, int32) (products []model.Product, err error)  
}
```

```
type ProductDataService struct {  
    productRepository repository.IProductRepository  
}
```

```
func NewProductDataService(productRepository repository.IProductRepository) IProductDataService {  
    return &ProductDataService{productRepository: productRepository}  
}
```

//重写接口方法

```
func (u *ProductDataService) Page(length int32, pageIndex int32) (products []model.Product, err error) {  
  
    return u.productRepository.Page(length, pageIndex)  
}
```

ithaoke123

防断更加微ithaoke123

防断更加微ithaoke123

防

防断更加微ithaoke123

ithaoke123

防断更加微ithaoke123