# Go mod配置

1. 配置文件：

配置开启go module模式

go env -w GO111MODULE=on

配置module获取的代理方式

go env -w GOPROXY=https://goproxy.cn,direct

生成bin包的路径

go env -w GOBIN=/Users/xxx/go/bin

1. 创建目录，并使用go mod进行初始化

mkdir go\_test

cd go\_test

go mod init go\_test

初始化之后就会生成对应的文件“go.mod”

1. 新增依赖

手动添加依赖

module go\_test  
require (  
 github.com/kataras/iris/v12  
)  
  
go 1.18

1. 使用依赖
2. package main  
     
   import (  
    "github.com/gin-gonic/gin"  
   )  
     
   func main() {  
    r := gin.Default()  
    r.GET("/ping", func(c \*gin.Context) {  
    c.JSON(200, gin.H{  
    "message": "pong",  
    })  
    })  
    r.Run() // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")  
   }

使用之后可以使用命令”go mod tidy”将没有下载的依赖下载下来并注册在go.mod中

# 使用web框架iris

参考网站：<https://www.yisu.com/zixun/604305.html>

启动iris可以监听端口，暴露api接口以供外部访问。

## Iris学习及测试

1. 引入iris依赖包

github.com/kataras/iris/v12 v12.1.8

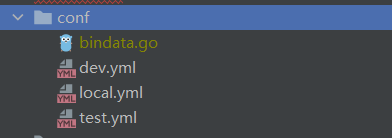
1. 使用iris对象开启端口监听

package main  
  
import (  
 "github.com/kataras/iris/v12"  
)  
  
func main() {  
 app := iris.New()  
 app.Handle("GET", "/ping", func(ctx iris.Context) {  
 ctx.JSON(iris.Map{"message": "pong"})  
 })  
 app.Run(iris.Addr(":8080"))  
}

1. 启动服务，访问<http://localhost:8080/ping>

## 生产环境iris实例化项目

1. yaml的配置文件



注意这个bindata，因为需要加载静态的资源文件，所以使用bindata这个插件。需要单独下载配置

1. ConfigUtil从配置文件中提取信息
2. package util  
     
   import (  
    "fmt"  
    "gopkg.in/yaml.v2"  
    "kaochong/kc-crm-feed/conf"  
    "log"  
    "os"  
    "strings"  
   )  
     
   var (  
    Db map[string]interface{}  
    Redis map[string]interface{}  
    Server map[string]interface{}  
    RocketMQ map[string]interface{}  
    Oss map[string]interface{}  
    Env = "local"  
    Consul map[string]interface{}  
    HttpClient map[string]interface{}  
    Other map[string]interface{}  
    KcHttpClient map[string]interface{}  
    Kick map[string]interface{}  
    Es map[string]interface{}  
   )  
     
   /\*  
   application run init  
   \*/  
   func init() {  
    //load config file by 启动参数  
    if len(os.Args) < 2 {  
    Env = "local"  
    } else {  
    Env = os.Args[1]  
    }  
    msg := fmt.Sprintf("env: %s", Env)  
    log.Println(msg)  
    prefix := "kc-crm-feed/conf/"  
    suffix := ".yml"  
    filePath := strings.Join([]string{prefix, suffix}, Env)  
    bs, \_ := conf.Asset(filePath)  
    configMap := make(map[string]interface{})  
    err := yaml.Unmarshal(bs, &configMap)  
    if err != nil {  
    log.Fatal(err)  
    return  
    }  
    data := configMap["db"].(map[interface{}]interface{})  
    Db = make(map[string]interface{})  
    for k, v := range data {  
    Db[k.(string)] = v  
    }  
    data = configMap["httpClient"].(map[interface{}]interface{})  
    HttpClient = make(map[string]interface{})  
    for k, v := range data {  
    HttpClient[k.(string)] = v  
    }  
     
    data = configMap["kcHttpClient"].(map[interface{}]interface{})  
    KcHttpClient = make(map[string]interface{})  
    for k, v := range data {  
    KcHttpClient[k.(string)] = v  
    }  
    //data = configMap["redis"].(map[interface{}]interface{})  
    //Redis = make(map[string]interface{})  
    //for k, v := range data {  
    // Redis[k.(string)] = v  
    //}  
    data = configMap["server"].(map[interface{}]interface{})  
    Server = make(map[string]interface{})  
    for k, v := range data {  
    Server[k.(string)] = v  
    }  
    data = configMap["rocketmq"].(map[interface{}]interface{})  
    RocketMQ = make(map[string]interface{})  
    for k, v := range data {  
    RocketMQ[k.(string)] = v  
    }  
    data = configMap["consul"].(map[interface{}]interface{})  
    Consul = make(map[string]interface{})  
    for k, v := range data {  
    Consul[k.(string)] = v  
    }  
    //data = configMap["es"].(map[interface{}]interface{})  
    //Es = make(map[string]interface{})  
    //for k, v := range data {  
    // Es[k.(string)] = v  
    //}  
    data = configMap["other"].(map[interface{}]interface{})  
    Other = make(map[string]interface{})  
    for k, v := range data {  
    Other[k.(string)] = v  
    }  
    return  
   }  
     
   func GetServerConfig() map[string]interface{} {  
    return Server  
   }  
     
   func GetDbConfig() map[string]interface{} {  
    return Db  
   }  
     
   func GetRocketMQConfig() map[string]interface{} {  
    return RocketMQ  
   }  
     
   func GetConsulConfig() map[string]interface{} {  
    return Consul  
   }  
     
   func GetHttpConfig() map[string]interface{} {  
    return HttpClient  
   }  
     
   func GetOtherConfig() map[string]interface{} {  
    return Other  
   }  
     
   func GetKickConfig() map[string]interface{} {  
    return Kick  
   }  
     
   func GetEsConfig() map[string]interface{} {  
    return Es  
   }  
     
   func GetKcHttpClientConfig() map[string]interface{} {  
    return KcHttpClient  
   }  
     
   /\*  
   master db config default  
   \*/  
   func GetDbMasterConfig(dbName string) map[string]interface{} {  
    dbConfig := GetDbConfig()  
    databaseConfig := dbConfig["database"].(map[string]interface{})  
    database := databaseConfig[dbName].(map[string]interface{})  
    return database["master"].(map[string]interface{})  
   }  
     
   /\*  
   master db config default  
   \*/  
   func GetDbMasterConfigDefault() map[string]interface{} {  
    dbConfig := GetDbConfig()  
    databaseConfig := dbConfig["database"].(map[string]interface{})  
    database := databaseConfig["kaochong"].(map[string]interface{})  
    return database["master"].(map[string]interface{})  
   }  
     
   /\*  
   slave db config by dbName  
   \*/  
   func GetDbSlaveConfig(dbName string) map[string]interface{} {  
    dbConfig := GetDbConfig()  
    databaseConfig := dbConfig["database"].(map[string]interface{})  
    database := databaseConfig[dbName].(map[string]interface{})  
    return database["slave"].(map[string]interface{})  
   }  
     
   /\*  
   slave db config default  
   \*/  
   func GetDbSlaveConfigDefault() map[string]interface{} {  
    dbConfig := GetDbConfig()  
    databaseConfig := dbConfig["database"].(map[string]interface{})  
    database := databaseConfig["kaochong"].(map[string]interface{})  
    return database["slave"].(map[string]interface{})  
   }  
     
   /\*  
   db pool  
   \*/  
   func GetDbPool() map[string]interface{} {  
    db := GetDbConfig()  
    pool := db["pool"].(map[string]interface{})  
    return pool  
   }  
     
   func GetRedisConfig() map[string]interface{} {  
    return Redis  
   }  
     
   func GetOssConfig() map[string]interface{} {  
    return Oss  
   }  
     
   func GetEnv() string {  
    return Env  
   }
3. 启动类加载配置文件信息，增加需要的配置，启动服务
4. package main  
     
   import (  
    "fmt"  
    "github.com/iris-contrib/swagger/v12"  
    "github.com/iris-contrib/swagger/v12/swaggerFiles"  
    "github.com/kataras/iris/v12"  
    "gitlab.xuanke.com/public-lib-go/kc\_arms/arm"  
    "gitlab.xuanke.com/public-lib-go/kc\_common/comm"  
    "kaochong/kc-crm-feed/datasource"  
    "kaochong/kc-crm-feed/rocketmq/consumer"  
    "kaochong/kc-crm-feed/rocketmq/producer"  
    "kaochong/kc-crm-feed/route/admin"  
    "kaochong/kc-crm-feed/util"  
    "net/http"  
    "time"  
   )  
     
   func main() {  
     
    app := iris.New().Configure(iris.WithoutBodyConsumptionOnUnmarshal)  
     
    // 全局中间件: 开始 Use, 结束 Done  
    //app.Use(middleware.CatchException)  
    //app.Use(middleware.Before)  
    //app.Done(middleware.After)  
     
    // 日志  
    app.Logger().SetLevel("debug")  
     
    // 首页  
    app.Get("/", func(ctx iris.Context) {  
    comm.Success(&ctx, nil, "Hello Kaochong Crm Feed!")  
    })  
     
    // 健康检查  
    app.Get("/actuator/health", func(ctx iris.Context) {  
    comm.Success(&ctx, nil, "Hello Kaochong Crm Feed!")  
    })  
     
    environment := util.Env  
    if environment == "local" || environment == "dev" {  
    swagHost := fmt.Sprintf("http://0.0.0.0:%d", util.GetServerConfig()["port"])  
    if environment == "dev" {  
    swagHost = "http://base-gateway.rdtest.xuanke.com/kc-crm-feed"  
    }  
    config := &swagger.Config{  
    URL: fmt.Sprintf("%s/swagger/doc.json", swagHost),  
    //TITLE: "用户中心接口文档",  
    }  
    app.Get("/swagger/{any:path}", swagger.CustomWrapHandler(config, swaggerFiles.Handler))  
    }  
     
    // 异常处理  
    app.OnAnyErrorCode(func(ctx iris.Context) {  
    comm.Fail(&ctx, ctx.GetStatusCode(), "error")  
    })  
     
    admin.Feed(app)  
     
    iris.RegisterOnInterrupt(func() {  
    //注销consul  
    if util.GetConsulConfig()["enable"] == nil || util.GetConsulConfig()["enable"].(bool) {  
    datasource.ConsulClient.Deregister(util.GetConsulConfig()["serviceName"].(string), util.GetServerConfig()["port"].(int))  
    }  
    //关闭RMQ消费者  
    consumer.CrmFeedRMQConsumer.StopConsumer()  
    })  
     
    // 注册退出回调  
    app.ConfigureHost(func(h \*iris.Supervisor) {  
    h.RegisterOnShutdown(func() {  
    //注销consul  
    if util.GetConsulConfig()["enable"] == nil || util.GetConsulConfig()["enable"].(bool) {  
    datasource.ConsulClient.Deregister(util.GetConsulConfig()["serviceName"].(string), util.GetServerConfig()["port"].(int))  
    }  
    ////关闭RMQ生产者  
    producer.RMQProducer.StopProducer()  
    ////关闭RMQ消费者  
    consumer.CrmFeedRMQConsumer.StopConsumer()  
    })  
    })  
     
    var opt arm.TracerOptions  
    opt.ServiceName = "kc-crm-feed"  
    opt.Ip = "0.0.0.0"  
    opt.Env = environment  
    // 设置启动参数  
    srv := &http.Server{  
    Addr: fmt.Sprintf(":%d", util.GetServerConfig()["port"]),  
    ReadTimeout: time.Duration(util.GetServerConfig()["readTimeout"].(int)) \* time.*Millisecond*,  
    WriteTimeout: time.Duration(util.GetServerConfig()["writeTimeout"].(int)) \* time.*Millisecond*,  
    Handler: arm.Wrap(app.Router, opt),  
    }  
     
    // 启动server  
    \_ = app.Run(  
    iris.Server(srv),  
    iris.WithoutServerError(iris.ErrServerClosed),  
    iris.WithOptimizations,  
    )  
   }

# golang基础语法

## 定义一个实体类

定义结构体

type Person struct {

}

package dto  
  
import "time"  
  
type Person struct {  
 //gorm标识的和数据库相关的字段的一些基本配置信息，json是转换的json字段  
 Id int `gorm:"column:id;primary\_key" json:"id"`  
 Name string `gorm:"column:name" json:"name"`  
 CustomerId int `gorm:"column:customer\_id" json:"customer\_id"`  
 Ctime time.Time `gorm:"type:datetime(0) not null;default:current\_timestamp;" json:"ctime"`  
 Utime time.Time `gorm:"type:datetime(0) not null on update current\_timestamp;omitempty;default:current\_timestamp;" json:"utime"`  
}

## 创建service方法

1. 构造方法

//初始化方法，带参数的  
func NewFeedService(ctx \*iris.Context) \*feedService {  
 return &feedService{  
 Ctx: ctx,  
 }  
}

1. 函数方法

func 函数名(参数)(返回值列表){}

/\*\*  
 \* 三元表达式  
 \*/  
func If(condition bool, trueVal, falseVal interface{}) interface{} {  
 if condition {  
 return trueVal  
 } else {  
 return falseVal  
 }  
}

1. 实例方法

func (s \*对象归属实例名) 方法名 (参数列表)(返回值列表){}

func (s \*feedService) GetFeedList(feedListReq \*req.FeedListReq) \*vo.FeedListVo {}

# golang高级语法及特性

## 协程和管道

### 简述

1. Go语言的并发基于goroutine

Go语言的并发是基于 goroutine 的，goroutine 类似于线程，但并非线程。可以将 goroutine 理解为一种虚拟线程。Go语言运行时会参与调度 goroutine，并将 goroutine 合理地分配到每个 CPU 中，最大限度地使用 CPU 性能。

1. Goroutine之间的通信基于管道Channel

多个 goroutine 中，Go语言使用通道（channel）进行通信，通道是一种内置的数据结构，可以让用户在不同的 goroutine 之间同步发送具有类型的消息。这让编程模型更倾向于在 goroutine 之间发送消息，而不是让多个 goroutine 争夺同一个数据的使用权。

1. 程序可以将需要并发的环节设计为生产者模式和消费者的模式，将数据放入通道。通道另外一端的代码将这些数据进行并发计算并返回结果
2. 实例代码
3. package main  
     
   import (  
    "fmt"  
    "math/rand"  
    "time"  
   )  
     
   // 数据生产者  
   func producer(header string, channel chan<- string) {  
    // 无限循环, 不停地生产数据  
    for {  
    // 将随机数和字符串格式化为字符串发送给通道  
    channel <- fmt.Sprintf("%s: %v", header, rand.Int31())  
    // 等待1秒  
    time.Sleep(time.*Second*)  
    }  
   }  
     
   // 数据消费者  
   func customer(channel <-chan string) {  
    // 不停地获取数据  
    for {  
    // 从通道中取出数据, 此处会阻塞直到信道中返回数据  
    message := <-channel  
    // 打印数据  
    fmt.Println(message)  
    }  
   }  
     
   func main() {  
    // 创建一个字符串类型的通道  
    channel := make(chan string)  
    // 创建producer()函数的并发goroutine  
    go producer("cat", channel)  
    go producer("dog", channel)  
    // 数据消费函数  
    customer(channel)  
   }

目的就是让多个协程之间相互通信。区别于Java的线程之间通信模型

### goroutine的用法

第一种：调用方法前增加go关键字

go GetThingDone(param1, param2);

第二种：匿名方法执行前增加go关键字

go func(param1,param2){}(val1,val2)

第三种：直接新建一个goroutine，在其中执行代码块

go{xxx}

### channel的用法

channel是类型相关的，每个channel只能指定一种类型，且需要在创建的时候声明

strChan :=make(channel string)

intChan :=make(channel int)

## 指针

可以简单理解成一种数据结构，这种数据结构记录的是内存地址，一般变量名定义成ptr。获取变量的内存地址使用&。使用指针获取对应的变量使用\*

ptr := &name

//将name的内存地址赋给ptr这个指针

name :=\*ptr

//获取ptr指针所指向的变量赋给name

# golang框架-gorm