**Golang开发**

**一、搭建开发环境**

1. go版本：

可以下载最新稳定版本，这里使用的是1.15.7版本

1. 下载安装：

官网下载地址：<https://golang.google.cn/dl/>

运行wget https://dl.google.com/go/go1.15.7.linux-amd64.tar.gz

或者下载tar.gz包传到服务器上进行解压

# 解压，官网推荐路径

tar -C /usr/local -zxvf go1.11.5.linux-amd64.tar.gz

# 添加go安装路径的bin目录/usr/loacl/go/bin到PATH变量中。添加到/etc/profile，需要root权限

vi /etc/profile

# 在最后一行添加

export GOROOT=/usr/local/go

export PATH=$PATH:$GOROOT/bin

# 保存退出后source一下，更新配置文件

source /etc/profile

# 执行go version，如果出现版本号则表示配置完成。

go version

**二、配置运行环境**

1. 配置GOPATH

（由于go运行有时需要一些外部依赖包，所以需要配置GOPATH，可以利用module导入依赖）

# 先配置go的工作路径GOPATH，编辑 ~/.bash\_profile文件

vi ~/.bash\_profile

# 在最后一行添加，$HOME/go为自己创建的工作路径

export GOPATH=$HOME/go

# 保存退出后source一下

source ~/.bash\_profile

1. 开启modules

go env -w GO111MODULE=on

# 添加阿里云依赖下载地址，方便下载依赖包

go env -w GOPROXY="https://mirrors.aliyun.com/goproxy/"

1. 运行代码

// 编写简单代码hello.go

package main

import "fmt"

func main() {

fmt.Printf("hello, world\n")

}

到代码文件目录下运行：go run hello.go

1. **操作oracle数据库**
2. 下载安装环境

在官网https://www.oracle.com/database/technologies/instant-client/linux-x86-64-downloads.html

下载oracle-instantclient-basic-21.1.0.0.0-1.x86\_64.rpm，安装客户端

# 默认安装路径为/usr/lib/oracle

sudo rpm -ivh oracle-instantclient-basic-21.1.0.0.0-1.x86\_64.rpm

1. 数据库连接

代码oracleConn.go：

package main  
  
import (  
 "database/sql"  
 "fmt"  
 \_ "github.com/godror/godror"  
)  
  
*//数据库配置*const (  
 host = "localhost"  
 port = 1521  
 user = "username"  
 sqlpassword = "userpwd"  
 dbname = "orcl"  
)  
  
func main() {  
 *// 用户名/密码@IP:端口/实例名* osqlInfo := fmt.Sprintf("%s/%s@%s:%d/%s", user, sqlpassword, host, port, dbname)  
 fmt.Println(osqlInfo)  
 db, err := sql.Open("godror", osqlInfo)  
 if err != nil {  
 panic(err)  
 }

*// 类似于finally，最后运行*  
 defer db.Close()

err = db.Ping()  
 if err != nil {

*// 中止执行，并打印*  
 panic(err)  
 }  
 fmt.Println("conn success!")}

1. 下载依赖，并运行：

在目录下运行go mod init，或者go mod [路径]，初始化modules文件，生成go.mod文件。可以在go.mod文件中添加所需的依赖，之后运行go mod tidy，会下载依赖，生成go.sum文件。

require (

github.com/godror/godror v0.23.1

)

# 运行代码

go run oracleConn.go

# 或者先编译再运行，会先生成oracleConn执行文件

go build oracleConn.go

./oracleConn

1. 数据库查询

*// 查询*func query(db \*sql.DB, sqlstmt string) {

*// 结构体用于封装查询结果*  
 type useracct struct {  
 useracct string  
 userpwd string  
 }  
 stmt, err := db.Prepare(sqlstmt)  
  
 if err != nil {  
 panic(err)  
 }  
 rows, err := stmt.Query()  
 if err != nil {  
 panic(err)  
 }  
  
 fmt.Println("query result:")  
 for rows.Next() {  
 tmpData := &useracct{}  
 rows.Scan(&tmpData.useracct, &tmpData.userpwd)  
 fmt.Println(tmpData)  
 }  
  
 defer func() {  
 stmt.Close()  
 rows.Close()  
 }()  
}

1. 插入数据

func insert(db \*sql.DB, useracct string, userpwd string) {  
 sqlStatement := "INSERT INTO useracct(useracct, userpwd) VALUES (:useracct, :userpwd)"  
 stmt, err := db.Prepare(sqlStatement)  
 defer stmt.Close()  
 if err != nil {  
 panic(err)  
 }  
 \_, err = stmt.Exec(useracct, userpwd)  
 if err != nil {  
 panic(err)  
 }  
 fmt.Println("insert success!")  
}

1. 修改数据

func update(db \*sql.DB, useracct string, userpwd string) {  
 sqlStatement := "UPDATE USERACCT SET USERPWD = :x1 WHERE USERACCT = :x2"  
 stmt, err := db.Prepare(sqlStatement)  
 defer stmt.Close()  
 if err != nil {  
 panic(err)  
 }  
 \_, err = stmt.Exec(userpwd, useracct)  
 if err != nil {  
 panic(err)  
 }  
 fmt.Printf("update useracct = %s set userpwd = %s success!\n", useracct, userpwd)  
}

1. 删除数据

func del(db \*sql.DB, useracct string) {  
 sqlStatement := "DELETE FROM USERACCT WHERE USERACCT = :useracct"  
  
 stmt, err := db.Prepare(sqlStatement)  
  
 defer stmt.Close()  
  
 if err != nil {  
 panic(err)  
 }  
  
 \_, err = stmt.Exec(useracct)  
 if err != nil {  
 panic(err)  
 }  
  
 fmt.Printf("delete useracct = %s success!\n", useracct)  
}

1. **操作redis**
2. 连接redis（导入依赖，操作如之前oracle数据库一样）

package main

import (  
 "fmt"  
 "github.com/garyburd/redigo/redis"  
)  
  
func main() {  
 conn,err := redisConn()  
  
 if err != nil {  
 fmt.Println("connect redis error :",err)  
 return  
 }  
  
 defer conn.Close()

}  
  
*// 连接*func redisConn() (redis.Conn, error){  
 do := redis.DialPassword("password")  
 conn,err := redis.Dial("tcp","localhost:6379", do)  
 if err != nil {  
 fmt.Println("connect redis error :",err)  
 return nil, err  
 }  
 return conn, err  
}

1. 获取单个key的值

func redisGotString(conn redis.Conn, key string) (string, error) {  
 val, err := redis.String(conn.Do("GET", key))  
 if err != nil {  
 fmt.Println("redis get error:", err)  
 } else {  
 fmt.Printf("Got %s: %s \n", key, val)  
 }  
 return val, err  
}

1. 判断key是否存在

func redisExist(conn redis.Conn, key string) (bool, error) {  
 flag, err := redis.Bool(conn.Do("EXISTS", key))  
  
 if err != nil {  
 fmt.Println("redis exits operation error:", err)  
 return false, nil  
 }  
 if flag {  
 fmt.Printf("%s is exists in redis \n", key)  
 return flag, nil  
 }  
 fmt.Printf("%s is not exists in redis \n", key)  
 return flag, nil  
}

1. 设置key,value

func redisSetString(conn redis.Conn, key string, value string){  
 num, err := redis.Int(conn.Do("SET", key, value))  
 if err != nil {  
 fmt.Println("redis set error:", err)  
 } else {  
 if int64(num) > 0 {  
 fmt.Printf("redis set string success：%s \n", value)  
 } else {  
 fmt.Printf("redis set string fail, maybe is exists：%s \n", value)  
 }  
 }  
}

1. 设置，setnx

func redisSetNX(conn redis.Conn, key string, param string){  
 num, err := redis.Int(conn.Do("SETNX", key, param))  
 if err != nil {  
 fmt.Println("redis set error:", err)  
 } else {  
 if num > 0 {  
 fmt.Printf("redis setnx success：%s \n", param)  
 } else {  
 fmt.Printf("redis setnx fail, maybe is exists：%s \n", param)  
 }  
 }  
}

1. 删除

func redisDel(conn redis.Conn, key string){  
 n, err := redis.Int(conn.Do("DEL", key))  
  
 if err != nil {  
 fmt.Println("redis del error:", err)  
 } else {  
 fmt.Printf("redis del key %s success, del number is %d \n", key, n)  
 }  
  
}

1. **rocketmq消费程序**
2. 消费程序

package main  
  
import (  
 "fmt"  
 "gitee.com/zhucheer/orange/queue"  
 "time"  
)

func main() {topic := "topDemo01"  
 addr := []string{"192.168.248.195:9876", "192.168.248.57:9876"}rocketConsumer(addr, topic)}

func rocketConsumer(addr []string, topic string) {  
 mqConsumerClient := queue.RegisterRocketConsumerMust(addr, topic)  
  
 fmt.Println("===============consumer beginning=============")

*// 常驻内存*  
 for {  
 fmt.Println("===============consumer receive=============")  
 *// run consumer listener, return consumer.ConsumeSuccess* mqConsumerClient.ListenReceiveMsgDo(topic, func(mqMsg queue.MqMsg) {  
 *// receive msg* fmt.Println(time.Now().Format("2006-01-02 15:04:05"), ":receive====>",mqMsg.MsgId, mqMsg.BodyString())  
 })

*// 为防止死循环带来的cpu压力，此处增加睡眠*  
 time.Sleep(time.Second \* 20)  
 }  
  
}