语言规范：https://golang.org/ref/spec编码风格：https://golang.org/doc/effective\_go.html https://github.com/golang/go/wiki/CodeReviewComments标准库源码： 解压go-go{$ver}.tar.gz , 目录go/src 众多值得学习的代码常见开源项目：https://github.com/golang/go/wiki/Projects如 Kubernetes、etcd

# Go安装

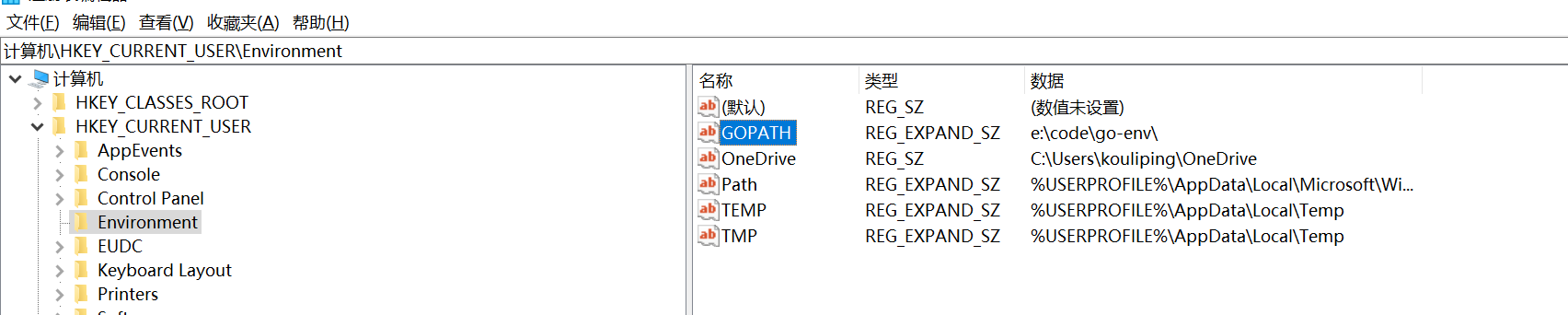
1st：安装golang

https://www.cnblogs.com/big-devil/p/9161529.html

修改GOPATH：

Win+r，输入regedit打开注册表

设置gopath



运行资源管理器，重启explorer.exe进程即可生效

# Channel

package main

import (

"fmt"

"time"

)

func main() {

c := make(chan int)

go func() {

fmt.Println("before sleep")

time.Sleep(2 \* time.Second)

fmt.Println("after sleep")

c <- 10

}()

fmt.Println("before <-c")

w, ok := <-c // 死循环等待有信号输入，可用于进程间同步，ok表示channel是否被关闭

fmt.Println("after <-c ", w, ok)

}

若将c<-10

# 小功能

## 增加转义符：

strings.Split

strings.Join

## 命名检查，四则运算

regexp 包

NAME\_ZH = "^([\u4e00-\u9fa5]|[a-zA-Z0-9\_]){0,16}$"

regexpNameZh = regexp.MustCompile(NAME\_ZH)

if !regexpNameZh.MatchString(ispMeta.IspName) {

return err

}

## 网页下载

net/http

证书检查：

Errors:

Error就是个interface类型，内部定义为

type error interface {

Error() string

}

# strings包

# io包

LimitedReader类型

type LimitedReader struct {

R Reader //underlying reader，最终读取操作通过R.Read完成

N int64 //max byte remaining

}

从R中读取但最多只返回N字节的数据

该结构体只实现了Read方法(Reader接口)

Eg：

safe := &io.LimitedReader{R: r.Body, N: KMaxReqBodySize}

//定义一个结构体，从r.body中读取数据，最长是N

buf, err := ioutil.ReadAll(safe)

//把safe转为字符串格式，存到buffer

r.Body.Close()

// 解析完http请求后，要close

//r \*http.Request

# net/http包

利用get、head、post、postfrom实现HTTP/HTTPS请求，HTTP包中包含了http客户端和服务端的实现，使用完后必须使用close关闭

type Request struct {

Method string // GET/POST/PUT等，对于client，该字段为空时表示为GET

URL url.URL // 对于server，从URI中解析出来，对于client，指定要链接的server

Proto string // 请求的协议。Client总是HTTP/1.1。

ProtoMajor int // 对于HTTP/1.0，该字段为1，下一字段为0

ProtoMinor int //

Header Header

Body io.ReadCloser //请求体。对于client 请求，空body表示没有请求体，比如GET。http client transport负责调用Close方法。对于server，不会为空，没有的话会返回EOF，server会关闭。

ContentLength int64

TransferEncoding []string

Close bool // 应答完成(server) 发送完成(client)是否关闭连接

Host string //

Form url.Values // 解析出的数据

Form url.Values // 解析出的数据，来自POST或PUT的body参数

Trailer Header // 附加头，body后

RemoteAddr string // 发送请求的地址

RequestURI string // 发送请求的URI

TLS \*tls.ConnectionState

}

# xorm

xorm是针对go语言的orm第三方库，提供简单丰富的api完成对数据库的各种操作，包括mysql、postgreSQL、SQLite3、MsSQL

orm：object relational mapping，对象关系映射，将关系型数据库转化为对象进行处理

import(

"github.com/go-sql-driver/mysql"

"github.com/go-xorm/xorm"

)

driverName := "mysql"

eng, err := xorm.NewEngine(driverName, dataSourceName)

//创建orm引擎

type IspMeta struct {

Id int64 `xorm:"id"`

Isp string `xorm:"isp"`

Level int8 `xorm:"level"`

IspName string `xorm:"isp\_name"`

Uerp string `xorm:"uerp"`

Desc string `xorm:"describe"`

}

// 定义模型

func (i \*IspMeta) TableName() string {

return "isp\_meta"

} 怎么解析出的表名字呢？find时候会调用该方法么

func GetIspMetaList(pageIndex int64, pageSize int64) ([]IspMeta, error) {

ispsMeta := make([]IspMeta, 0, pageSize)

err := common.XEngPool.Asc("id").Limit(int(pageSize), int((pageIndex-1)\*pageSize)).Find(&ispsMeta)

return ispsMeta, err

}

## Get方法：

查询单条数据

根据id/where/user来获取单条数据（或组合条件）

user := new(User)

Engine.Id(id).Get(user)

Engine.Where(“id=?”,id)Get(user)

User.Id = id

Engine.Get(user)

User中的飞控field自动成为检查条件与前面的方法条件组合在一起查询

返回has, err

## Find方法：

获取多条数据

传入map或slice

## Count方法：

获取统计

## 创建engine

Goroutine安全的，退出时自动关闭

func newDb(dbConf \*DbConf) (\*xorm.Engine, error) {

driverName := "mysql"

dsnConfig := &mysql.Config{

User: dbConf.User,

Passwd: dbConf.PassWord,

Addr: dbConf.Host,

Net: "tcp",

DBName: dbConf.Name,

AllowNativePasswords: true,

ReadTimeout: time.Duration(60) \* time.Second, // I/O read timeout

WriteTimeout: time.Duration(60) \* time.Second, // I/O write timeout

Timeout: time.Duration(60) \* time.Second, // Dial timeout

}

dataSourceName := dsnConfig.FormatDSN()

fmt.Printf("dataSourceName %s\n", dataSourceName)

//dataSourceName := "root:123456@tcp(172.28.17.130:3306)/cloud\_wafapi\_db?allowNativePasswords=true&readTimeout=1s&timeout=1s&writeTimeout=1s&maxAllowedPacket=0"

dB, err := xorm.NewEngine(driverName, dataSourceName)

if err != nil {

fmt.Println("xorm new engin err: %s ", err.Error())

return nil, err

}

dB.ShowSQL(false)

err = dB.Ping()

if err != nil {

fmt.Println("db ping err: %a", err.Error())

return nil, err

}

return dB, nil

}

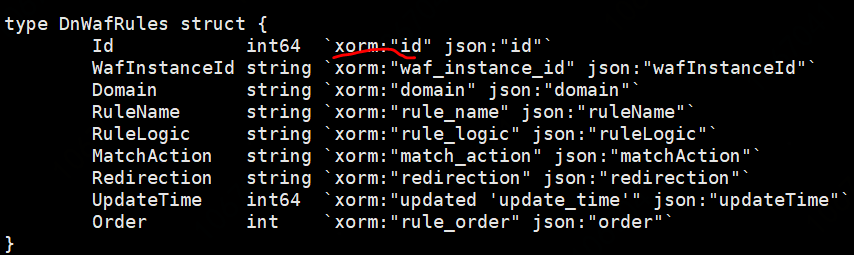
## 导入导出sql语句

engine.DumpAllFile(fpath string)。

engine.ImportFile(fpath string)

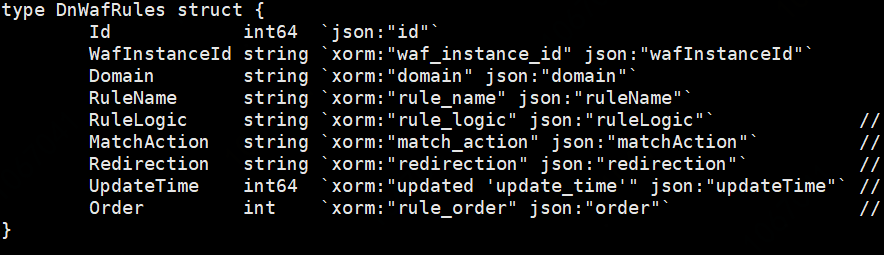
## ID方法

ID condition is error, expect 0 primarykeys, there are 1



改为：

结构中的Id自动映射为table中的id



## Join方法



# reflect包

func Indirect(v vlaue) val

返回v指向的值，若v为空指针返回0，若v不是指针类型，直接返回v

# func [ValueOf](https://golang.org/src/reflect/value.go#L2150)

func ValueOf(i interface{}) [Value](https://godoc.org/reflect#Value)

ValueOf returns a new Value initialized to the concrete value stored in the interface i. ValueOf(nil) returns the zero Value.

返回一个指针，指针指向的内容被初始化为interface i中的值，valueof(nil) = 0

# strconv

字符串和其他类型转换

# flag

解析入参

# URL编码：

escape encondeURI：两种编码方式，都生成%\*的格式，即百分号编码

对需要编码的ASCII字符，表示为两个16进制的数字，然后再前面放转义符%，置入URI的相应位置

escape:

非标准

%uxxxx，xxxx是用4个十六进制表示

encodeURI：

W3C标准

非ASCII转为UTF-8字节序后表示

怎么解读“凹”的UTF-8编码二进制11100101 10000111 10111001？

1. 第一个字节的第一位不是0，确认第一个字节不是一个字符；
2. 第一个字节的第一位是1，而且连续有3个1，就表示该字符占用了3个字节。
3. 于是，计算机清晰地知道 11100101 10000111 10111001 三个字节表示一个符号，而不是分别表示三个符号，解读完毕。