Goland 传参数启动

代码 goCount = flag.Int("goCount", 0, "goroutine number")

设置 Run-configuration => program arguments=> -goCount=5

简单的多包引用 都是main同一个文件夹的情况

编译那里选 run kind ：dirctory

dirctory ：C:/Users/lenovo/Desktop/test/go/client/

直接就可以用 不用添加import

Web server

====

package main

import "net/http"

func SayHello(w http.ResponseWriter, req \*http.Request) {

w.Write([]byte("Hello"))

}

func main() {

http.HandleFunc("/", SayHello)

http.ListenAndServe(":3000", nil)

// TODO

}

非常重要！！！！！

访问 localhost:3000, 注意：代码端写 http:\\localhost:3000 浏览器直接localhost:3000 。(注意端口不能被事先占用，否则会立马返回 netstat -ano 找到端口对应的pid 再去任务管理器找程序)

==============

简易rest 服务器

package main

import (

"encoding/json"

"fmt"

"github.com/emicklei/go-restful"

"io"

"net/http"

)

func main() {

ws := new(restful.WebService)

ws.Route(ws.GET("/").To(hello))

restful.Add(ws) // ws被添加到默认的container restful.DefaultContainer中

fmt.Print("Server starting on port 3001\n")

http.ListenAndServe(":3001", nil)

}

func hello(req \*restful.Request, resp \*restful.Response) {

article := Article{"A Royal Baby", "A slow news week"} //类型名字用大写，变量用小写

b, \_ := json.Marshal(article) // 序列化 2进制数据

fmt. Println ( "哈哈 你来啦！"，string(b) )

io.WriteString(resp, string(b))

}

type Article struct {

Name string

Body string

}

==============

Client端 计算单线程返回时间

package main

import (

"flag"

"fmt"

"log"

"time"

"net/http"

"io/ioutil"

)

func connectServer() {

client := &http.Client{}

//url := "http://www.baidu.com"

url := "http://localhost:3001/"

reqest, err := http.NewRequest("GET", url, nil)

if err != nil {

panic(err)

}

response, \_ := client.Do(reqest)

defer response.Body.Close()

body, \_ := ioutil.ReadAll(response.Body)

body\_str := string(body)

fmt.Println(body\_str)

status := response.StatusCode

fmt.Println(status)

}

func checkError(err error) {

if err != nil {

log.Fatal("an error!", err.Error())

}

}

func main() {

var goCount \*int

goCount = flag.Int("goCount", 0, "goroutine number")

flag.Parse()

fmt.Println("go count = ", \*goCount)

tInsert := time.Now()

fmt.Println("tStart time: ", tInsert)

for i := 0; i < \*goCount; i++ {

fmt.Println("goroutine number: ", i)

connectServer()

}

elapsed := time.Since(tInsert)

fmt.Println("Time elapsed: ", elapsed)

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GoConvey

在命令行运行下面的命令：

go get github.com/smartystreets/goconvey

运行时间较长，运行完后你会发现：

在$GOPATH/src目录下新增了github.com子目录，该子目录里包含了GoConvey框架的库代码

在$GOPATH/bin目录下新增了GoConvey框架的可执行程序goconvey

环境变量path里面添加 C:\go\my\_package\bin

目标文件夹下（\*\_test.go ..）直接命令行 goconvey

访问 localhost:8080 (注意8080端口不要被占用)

GoConvey也支持自己定义一个断言函数 So的函数原型如下

func So(actual interface{}, assert assertion, expected ...interface{})

可以看到第二个参数是assertion 查看下assertion

type assertion func(actual interface{}, expected ...interface{}) string

可以看出只要传入一个符合assertion格式的函数就可以了 我自己写了个断言函数如下

func Test4(t \*testing.T) {

Convey("BrainWu is handsome!!!", t, func() {

So("BrainWu", ShouldBrainWuHandsome, "Handsome")

})

}

//符合assertion类型规范的函数 如果返回字符串为空值 那么将会被pass

//如果有内容 那么将不通过 且返回内容会被输出

func ShouldBrainWuHandsome(actual interface{}, expected ...interface{}) string {

if actual == "BrainWu" && expected[0] == "Handsome" {

return ""//pass

} else {

return "BrainWu is handsome!!!"//not pass and output the return value

}

}

有的时候我们想要忽略某些断言操作但又不想删掉 GoConvey提过了一种方法可以不去断言

一种是忽略Convey一种是忽略So

示例如下

func Test5(t \*testing.T) {

SkipConvey("Important stuff", t, func() { // This func() will not be executed!

Convey("More important stuff", func() {

So("asdf", ShouldEqual, "asdf")

})

})

Convey("1 Should Equal 2", t, func() {

// This assertion will not be executed and will show up as 'skipped' in the report

SkipSo(1, ShouldEqual, 2)

})

}

\*\*\*\*\*\*\*\*\*\*