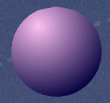
编程项目训练营

<基础课程第三周>



轻程计划

目录



```go
hello.go > main
package main

import (
    "fmt"
)

type user struct {
    name string
}

func main() {
    u := user{"tang"}
    //Printf 格式化输出
    fmt.Printf("% + v\n", u)      //格式化输出结构
    fmt.Printf("%#v\n", u)       //输出值的 Go 语言表示方法
    fmt.Printf("%T\n", u)        //输出值的类型的 Go 语言表示
    fmt.Printf("%t\n", true)     //输出值的 true 或 false
    fmt.Printf("%b\n", 1024)     //二进制表示
    fmt.Printf("%c\n", 11111111) //数值对应的 Unicode 编码字符
    fmt.Printf("%d\n", 10)       //十进制表示
    fmt.Printf("%o\n", 8)        ///八进制表示
    fmt.Printf("%q\n", 22)       //转化为十六进制并附上单引号
    fmt.Printf("%x\n", 1223)     //十六进制表示，用a-f表示
    fmt.Printf("%X\n", 1223)     //十六进制表示，用A-F表示
    fmt.Printf("%U\n", 1233)     //Unicode表示
    fmt.Printf("%b\n", 12.34)    //无小数部分，两位指数的科学计数法6946802425218990p-49
    fmt.Printf("%e\n", 12.345)   //科学计数法，e表示
    fmt.Printf("%E\n", 12.34455) //科学计数法，E表示
    fmt.Printf("%f\n", 12.3456)  //有小数部分，无指数部分
    fmt.Printf("%g\n", 12.3456)  //根据实际情况采用%e或%f输出
    fmt.Printf("%G\n", 12.3456)  //根据实际情况采用%E或%f输出
    fmt.Printf("%s\n", "wqdew")  //直接输出字符串或者[]byte
    fmt.Printf("%q\n", "dedede") //双引号括起来的字符串
    fmt.Printf("%x\n", "abczxc") //每个字节用两字节十六进制表示，a-f表示
    fmt.Printf("%X\n", "asdzxc") //每个字节用两字节十六进制表示，A-F表示
    fmt.Printf("%p\n", 0x123)    //0x开头的十六进制数表示
}

```

PART1

Golang 标准库

Go 语言中，为了方便开发者使用，将 IO 操作封装在了如下几个包中：

- io 为 IO 原语（I/O primitives）提供基本的接口
- io/ioutil 封装一些实用的 I/O 函数
- fmt 实现格式化 I/O，类似 C 语言中的 printf 和 scanf
- bufio 实现带缓冲I/O

文本

```go
hello.go > main
1   package main
2
3   import (
4       "fmt"
5       "strings"
6       "unicode"
7   )
8
9   type user struct {
10      name string
11  }
12
13  func main() {
14      fmt.Println(strings.ToLower("HELLO WORLD"))
15      fmt.Println(strings.ToLower("Ā Á Ǎ À"))
16      fmt.Println(strings.ToLowerSpecial(unicode.TurkishCase, "壹"))
17      fmt.Println(strings.ToLowerSpecial(unicode.TurkishCase, "HELLO WORLD"))
18      fmt.Println(strings.ToLower("Önnek İş"))
19      fmt.Println(strings.ToLowerSpecial(unicode.TurkishCase, "Önnek İş"))
20  }
21
```

```
PROBLEMS 12  OUTPUT  TERMINAL  DEBUG CONSOLE

➜  hello go run hello.go
hello world
ā á ǎ à
壹
hello world
önnek iş
önnek iş
➜  hello
```

几乎任何程序都离不开文本（字符串）

- *strings* 包提供了很多操作字符串的简单函数，通常一般的字符串操作需求都可以在这个包中找到。
- *strconv* 包提供了基本数据类型和字符串之间的转换。
- 进行复杂的文本处理必然离不开正则表达式。
- Go 代码使用 UTF-8 编码（且不能带 BOM），同时标识符支持 Unicode 字符。

时间

```go
package main

import (
    "fmt"
    "time"
)

type user struct {
    name string
}

func main() {
    t, _ := time.Parse("2006-01-02 15:04:05", "2020-06-21 12:20:00")
    fmt.Println(t.Hour())
}
```

```
PROBLEMS 12  OUTPUT  TERMINAL  DEBUG CONSOLE
→ hello go run hello.go
12
→ hello
```

- 实际开发中，经常会遇到日期和时间相关的操作，比如：格式化日期和时间，解析一个日期时间字符串等。Go 语言通过标准库 time 包处理日期和时间相关的问题。

为什么是 2006-01-02 15:04:05

可能你已经注意到：`2006-01-02 15:04:05` 这个字符串了。没错，这是固定写法，类似于其他语言中 `Y-m-d H:i:s` 等。为什么采用这种形式？又为什么是这个时间点而不是其他时间点？

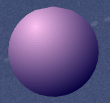
- 官方说，使用具体的时间，比使用 `Y-m-d H:i:s` 更容易理解和记忆；这么一说还真是 ~
- 而选择这个时间点，也是出于好记的考虑，官方的例子：`Mon Jan 2 15:04:05 MST 2006`，另一种形式 `01/02 03:04:05PM '06 -0700`，对应是 1、2、3、4、5、6、7；常见的格式：`2006-01-02 15:04:05`，很好记：2006 年 1 月 2 日 3 点 4 分 5 秒 ~

参考资料:

- https://github.com/polaris1119/The-Golang-Standard-Library-by-Example
- https://github.com/hantmac/Mastering_Go_ZH_CN
- https://studygolang.com/pkgdoc

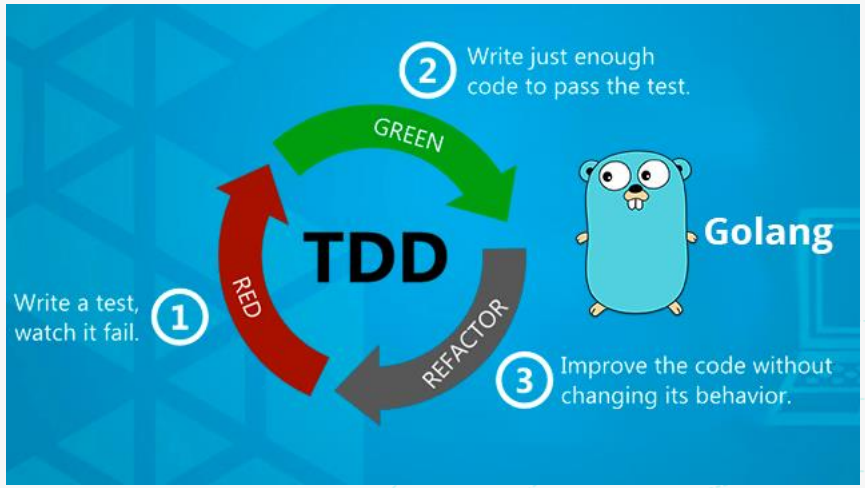
轻程计划

目录



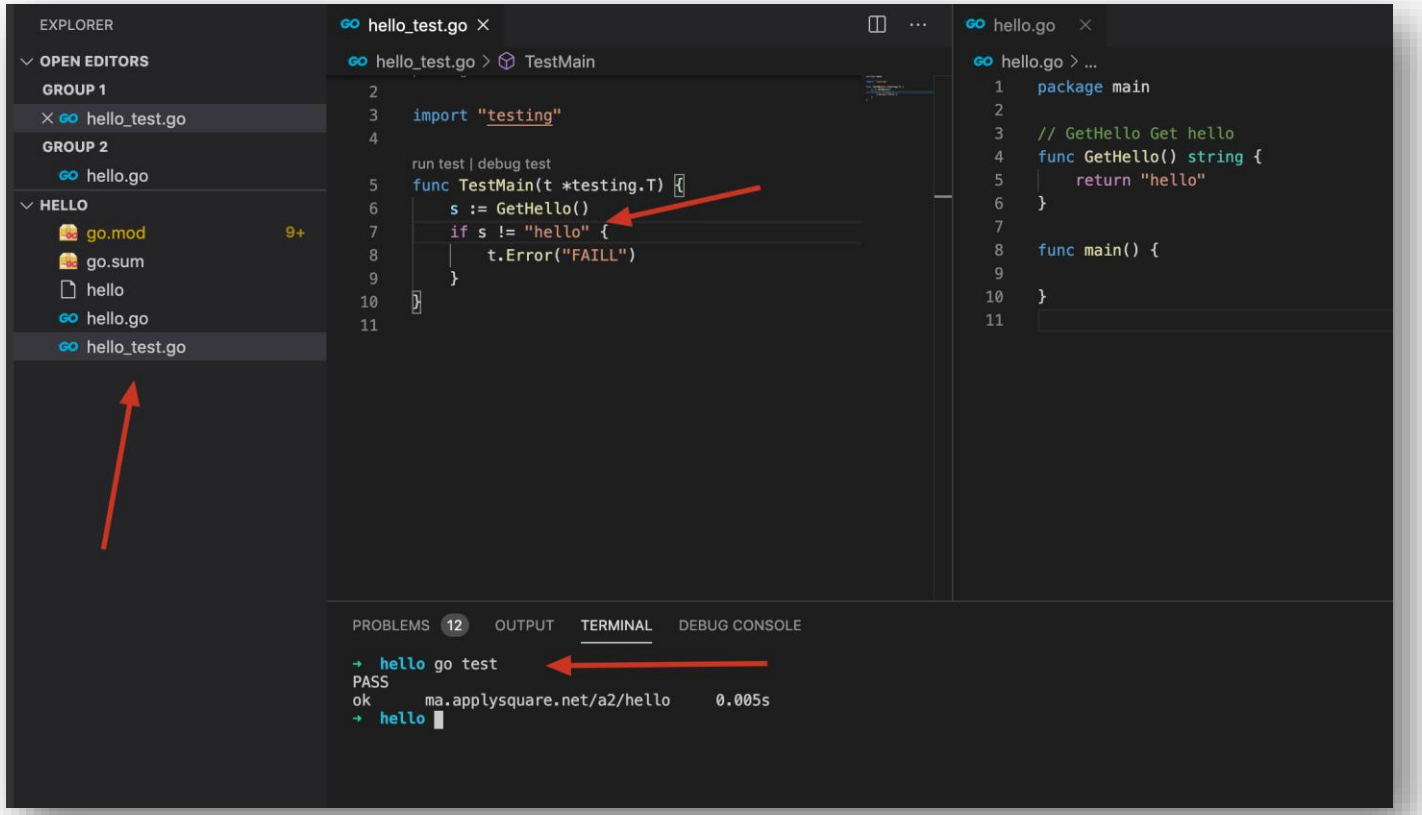
以单元测试为荣，以手工测试为耻。

单元测试是一段**自动化**的代码，这段代码调用被测试的工作单元，之后对这个单元的单个最终结果的某些假设进行检验。单元测试几乎都是用单元测试框架编写的。单元测试容易编写，能快速运行。单元测试**可靠、可读、并且可维护**。只要产品代码不发生变化，单元测试的**结果是稳定的**。



感兴趣的同学可以了解： https://www.xenonstack.com/blog/test-driven-development-golang/

testing 方便进行 Go 包的自动化单元测试、基准测试



Go 语言从开发初期就注意了测试用例的编写。特别是静态语言，由于调试没有动态语言那么方便，所以能最快最方便地编写一个测试用例就显得非常重要了。

- hello_test.go，命名规则。
- 通过 go test 命令，能够自动执行如下形式的任何函数：
  - func TestXxx(*testing.T)
  - 注意：Xxx 可以是任何字母数字字符串，但是第一个字母不能是小些字母。
  - 在这些函数中，使用 Error, Fail 或相关方法来发出失败信号。

示例

sse_test.go > ...

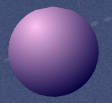
```
田欧, a year ago | 2 authors (Manu Mtz-Almeida and others)
// Copyright 2014 Manu Martinez-Almeida.  All rights reserved.      Manu Mtz-Almeida, 5 years ago
// Use of this source code is governed by a MIT style
// license that can be found in the LICENSE file.

run package tests | run file tests | run package benchmarks | run file benchmarks
package sse

import (
    "bytes"
    "net/http/httptest"
    "testing"

    "github.com/stretchr/testify/assert"
)

run test | debug test
func TestEncodeOnlyData(t *testing.T) {
    w := new(bytes.Buffer)
    event := Event{
        Data: "junk\n\njk\nid:fake",
    }
    err := Encode(w, event)
    assert.NoError(t, err)
    assert.Equal(t, w.String(),
        `data:junk
data:
data:jk
data:id:fake

`)

    decoded, _ := Decode(w)
    assert.Equal(t, "message", decoded[0].Event)
    assert.Equal(t, decoded[0].Data, []Event{event}[0].Data)
}

run test | debug test
func TestEncodeWithEvent(t *testing.T) {
    w := new(bytes.Buffer)
    event := Event{
        Event: "t\n:<>\r\test",
        Data:  "junk\n\njk\nid:fake",
    }
    err := Encode(w, event)
    assert.NoError(t, err)
    assert.Equal(t, w.String(),
        `event:t\n:<>\r est
data:junk
data:
data:jk
```

- 示例代码：
- https://github.com/gin-contrib/sse/blob/master/sse_test.go

- Assert库，更加方便的断言。
  - assert.NoError(t, err)
  - assert.Equal(t, w.String(), "id:12345\nevent:abc\nretry:10\ndata:some data\n\n")

轻程计划

目录



PART3· Json解析

```go
hello.go > main
package main

import (
    "encoding/json"
    "fmt"
)

type Message struct {
    Name string
    Body string
    Time int64
}

func main() {
    var m Message
    b := []byte(`{"Name":"Alice","Body":"Hello","Time":1294706395881547000}`)
    err := json.Unmarshal(b, &m)
    if err != nil {
        fmt.Println("json Unmarshal error")
    }

    fmt.Println(m.Name)

}
```

```
PROBLEMS 14  OUTPUT  TERMINAL  DEBUG CONSOLE
→ hello go run hello.go
Alice
→ hello
```

- json包实现了json对象的编解码，参见RFC 4627。Json对象和go类型的映射关系请参见Marshal和Unmarshal函数的文档。

- 参考资料：
  https://www.jianshu.com/p/b9c15e3e04c8

PART3· Gjson 库

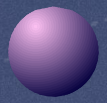
```
hello.go > json
1   package main
2
3   import "github.com/tidwall/gjson"
4
5   const json = `{"name":{"first":"Janet","last":"Prichard"},"age":47}`
6
7   func main() {
8       value := gjson.Get(json, "name.last")
9       println(value.String())
10  }
11

PROBLEMS 14  OUTPUT  TERMINAL  DEBUG CONSOLE

→  hello go run hello.go
Prichard
→  hello
```

- Get JSON values quickly - JSON parser for Go
- https://github.com/tidwall/gjson

```
→  hello go get -u github.com/tidwall/gjson
go: downloading github.com/tidwall/gjson v1.6.0
go: github.com/tidwall/gjson upgrade => v1.6.0
go: downloading github.com/tidwall/match v1.0.1
go: downloading github.com/tidwall/pretty v1.0.0
go: github.com/tidwall/pretty upgrade => v1.0.1
go: downloading github.com/tidwall/pretty v1.0.1
→  hello
```

轻程计划

目录



- 新建一个Go项目，使用go mod进行包。完成一个命令行程序。读取互联网上天气预报API数据，实时显示本地天气信息。
- 参考资料：https://blog.csdn.net/QQ459932400/article/details/91128126

提交形式 > > >

- 教学立方平台第三周作业
- PDF文档
  - Fork作业仓库
  - Github代码截图，并分享自己作业的Github代码地址。
  - 虚拟机上执行main.go的截图