Goland之template模板

一、模板基本使用

使用的包是html/template，而不是text/template

```
unc main() {
    tplText := "你好 {{ . }}"

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    // 普通字符串
    tpl.Execute(os.Stdout, "1111111111111")
    fmt.Println()

    // Execute 可以传递的参数有，切片、映射、自定义结构体等
    tpl.Execute(os.Stdout, []int{1, 2, 3, 4})
    fmt.Println()

    tpl.Execute(os.Stdout, map[string]string{"name": "kk"})
    fmt.Println()

    tpl.Execute(os.Stdout, struct {
        Id   int
        Name string
    }{1, "kk"})
    fmt.Println()
}
```

二、template数据渲染

2.1、切片渲染

通过index渲染切片中的数据

```
func main() {
    tplText := "我是切片的数据 {{ index . 0 }}   {{ index . 1 }}"

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    // Execute 可以传递的参数有，切片、映射、自定义结构体
    tpl.Execute(os.Stdout, []int{1, 2, 3, 4})
    fmt.Println()

}
```

2.2、 map（映射）渲染

通过 **.keyname 获取value**，如果**keyname不存在则为空**

```
func main() {
    tplText := "我是map的数据 {{ .name }}-{{ .addr }}{{ .ttt }}"

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    tpl.Execute(os.Stdout, map[string]string{"name": "kk", "addr": "hainan"})
    fmt.Println()
}
```

2.3、结构体

通过. 属性获取对应的值，如果某个属性不存在。那么后面的就会停止获取

```
func main() {
    tplText := "我是struct的数据 {{ .Id }}{{ .ttt }}{{ .Name }}" // 我是struct的数据
1
    tplText := "我是struct的数据 {{ .Id }} {{ .Name }}"          // 我是struct的数据
1 kk

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    tpl.Execute(os.Stdout, struct {
        Id   int
        Name string
    }{1, "kk"})
    fmt.Println()

}
```

三、模板语法

3.1、if-else

```
func main() {
    // 必须要end结尾
    tplText := `我是struct的数据
    {{ .Id }} {{ .Name }}
    {{ if eq .Sex 1}}男{{ else }}女{{end}}
    `

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    tpl.Execute(os.Stdout, struct {
        Id   int
        Name string
        Sex  int // 1 的时候是男，0 的时候是女
    }{1, "kk", 1})
```

```
    fmt.Println()
}
```

3.2、循环语法

```
func main() {
    // 必须要end结尾，range一个就便利一个对象，想拿到每个数据的值，就继续
    tplText := `
        {{ range .}}
            {{ .Id }}-{{ .Name }}-{{ if eq .Sex 1}}男{{ else }}女{{end}}
        {{ end }}
    `

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    // 匿名结构体切片
    tpl.Execute(os.Stdout, []struct {
        Id   int
        Name string
        Sex  int // 1 的时候是男，0 的时候是女
    }{{1, "kk", 1}, {2, "kk", 0}})
    fmt.Println()
}

----------遍历循环结构体----------
type Addr struct {
    Street string
    No     int
}

func main() {
    // 必须要end结尾，range一个就便利一个对象，想拿到每个数据的值，就继续
    tplText := `
        {{ range .}}
            {{ .Id }}-{{ .Name }}-{{ if eq .Sex 1}}男{{ else }}女{{end}}
            {{ .Addr }}
            {{ .Addr.Street }}
        {{ end }}
    `

    // Must 语法检查
    tpl := template.Must(template.New("tpl").Parse(tplText))

    // 匿名结构体切片
    tpl.Execute(os.Stdout, []struct {
        Id   int
        Name string
        Sex  int // 1 的时候是男，0 的时候是女
        Addr Addr
    }{{1, "kk", 1, Addr{"上海", 111}}, {2, "kk", 0, Addr{"北京", 222}}})
    fmt.Println()
}
```

3.3、模板自定义函数

```
func main() {
    // 必须要end结尾，range一个就便利一个对象，想拿到每个数据的值，就继续
    tplText := `{{ upper . }}`

    // 自定义函数
    // key是一个字符串，也是函数名称，在模板中调用的
    funcs := template.FuncMap{
        "upper": strings.ToUpper,
    }

    // Must 语法检查，Funcs传递自定义参数
    tpl := template.Must(template.New("tpl").Funcs(funcs).Parse(tplText))
    tpl.Execute(os.Stdout, "asdfghjkl")
    fmt.Println()
}
```

3.4、块的替换

```
func main() {
    // 必须要end结尾，块的替换
    // 使用 下面自定义的content 替换 block-content
    tplText := `替换后的内容：{{ block "content" . }} {{ . }} {{ end }}`

    tpl := template.Must(template.New("tpl").Parse(tplText))

    tpl, _ = tpl.Parse(`{{ define "content" }} {{ len . }} {{ end }}`)

    tpl.Execute(os.Stdout, "asdfghjkl")

}
```

3.5、默认模板

```
func main() {
    // 必须要end结尾，块的替换
    tplText :=
        `
    {{ define "len" }} {{ len . }} {{ end }}
    {{ define "raw" }} {{ . }} {{ end }}

    现在会解析这2个模板
    {{ template "len" . }}
    {{ template "raw" . }}
    `

    tpl := template.Must(template.New("tpl").Parse(tplText))
    // 指定调用模板
    tpl.Execute(os.Stdout, "aaaa")
}

-----------------------
func main() {
    // 必须要end结尾，块的替换
    tplText :=
```

```
    {{ define "len" }} {{ len . }} {{ end }}
    {{ define "raw" }} {{ . }} {{ end }}
    `

    tpl := template.Must(template.New("tpl").Parse(tplText))
    // 指定调用模板
    tpl.ExecuteTemplate(os.Stdout, "len", "aaaa")
}
```

3.6、结合文件使用

```
func main() {

    // 直接解析到文件里面,能指定多个文件
    tpl := template.Must(template.ParseFiles("html/index.html",
"html/len.html"))

    // 指定模板名字(文件名)
    tpl.ExecuteTemplate(os.Stdout, "index.html", []int{1, 2, 4})
    tpl.ExecuteTemplate(os.Stdout, "len.html", []int{1, 2, 4})
}
```

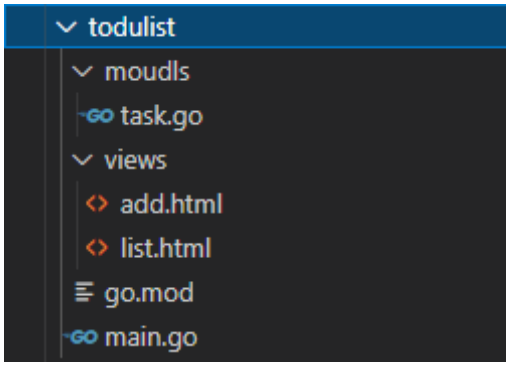
```
index.html
{{ range . }}
    {{ . }}
{{ end }}


<!-- 模板中调用另一个模板 -->
{{ template "len.html" . }}


len.html
{{ len . }}
```

四、结合代码去使用

4.1、目录结构（go mod init todulist）



4.2、main.go文件代码

```
func main() {

    http.HandleFunc("/list/", func(reponse http.ResponseWriter, request
*http.Request) {

        // 指定文件路径
        tpl := template.Must(template.ParseFiles("views/list.html"))
        // 写到响应reponse，指定要输出到的文件list.html  指定输入的数据（可有切片、map、结
构体）
        // moudls.GetTasks数据渲染到list.html,然后通过reponse展示到页面
        tpl.ExecuteTemplate(reponse, "list.html", moudls.GetTasks())
    })

    http.HandleFunc("/add/", func(reponse http.ResponseWriter, request
*http.Request) {

        // 判断请求是否是POST请求，是的话就提交数据
        if request.Method == http.MethodPost {

            // 获取数据
            name := request.PostFormValue("name")

            // 提交数据，moudls层定义的数据
            moudls.AddTasks(name)

            // 临时重定向回去
            http.Redirect(reponse, request, "/list/", 302)
        }

        // 指定模板解析到某个文件
        tpl := template.Must(template.ParseFiles("views/add.html"))

        // 输出到reponse，指定要渲染的文件list.html，这里不用传递数据!
        tpl.ExecuteTemplate(reponse, "add.html", nil)
    })

    http.ListenAndServe(":9999", nil)
}
```

4.3、task.go文件代码

```
package moudls

type Task struct {
    ID     int
    Name   string
    Status string
}

var tasks = []*Task{&Task{1, "kk", "1"}, &Task{2, "kk", "0"}}

// 展示数据
func GetTasks() []*Task {
    return tasks
```

```
}

// 提交数据
func AddTasks(name string) {
    tasks = append(tasks, &Task{len(tasks), name, "新增"})
}
```

4.4、两个html文件代码

```
--------add.html-----------
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>ADD</title>
</head>

<body>
    <form action="/add/" method="post">
        <label> 任务名: </label>
        <input type="text" name="name" value=""> <br />
        <input type="submit" value="新增" />
        <input type="submit" value="取消" />
    </form>
</body>
</html>


-----list.html--------
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    任务列表

    <a href="/add/">新增</a> <br>
    {{ range .}}
        ID    {{ .ID }}

        NAME  {{ .Name }}

        状态   {{ .Status }}
    {{ end }}
</body>
</html>
```