IO操作1

作者：少林之巅

目录

格式化输入

1. 从终端获取用户的输入

   fmt.Scanf(format string, a…interface{}): **格式化输入，空格作为分隔符，占位符和格式化输出一致**

   fmt.Scan(a …interface{}): **从终端获取用户输入，存储在ScanIn中的参数里，空格和换行符作为分隔符**

   fmt.ScanIn(a …interface{}): **从终端获取用户输入，存储在ScanIn中的参数里，空格作为分隔符，遇到换行符结束**

格式化输入背后的原理

2. 终端操作实例

```
package main

import (
        "fmt"
)

var (
        firstName, lastName, s string
        i                      int
        f                      float32
        input                  = "56.12 / 5212 / Go"
        format                 = "%f / %d / %s"
)

func main() {
        fmt.Println("Please enter your full name: ")
        fmt.Scanln(&firstName, &lastName)
        // fmt.Scanf("%s %s", &firstName, &lastName)
        fmt.Printf("Hi %s %s!\n", firstName, lastName) // Hi Chris Naegels
        fmt.Sscanf(input, format, &f, &i, &s)
        fmt.Println("From the string we read: ", f, i, s)
}
```

格式化输入

3. 从字符串中获取输入

fmt.Sscanf(str, format string, a…interface{}): **格式化输入，空格作为分隔符，占位符和格式化输出一致**

fmt.Sscan(str string, a …interface{}): **从终端获取用户输入，存储在ScanIn中的参数里，空格和换行符作为分隔符**

fmt.SscanIn(str string, a …interface{}): **从终端获取用户输入，存储在ScanIn中的参数里，空格作为分隔符，遇到换行符结束**

格式化输出

4. 格式化输出

fmt.Printf(format string, a…interface{}): **格式化输出，并打印到终端**

fmt.Println(a …interface{}): **把零个或多个变量打印到终端， 并换行**

fmt.Print(a …interface{}): **把零个或多个变量打印到终端**

格式化输出

5. 格式化并返回字符串

fmt.Sprintf(format string, a…interface{}): **格式化并返回字符串**

fmt.Sprintln(a …interface{}): **把零个或多个变量按空格进行格式化并换行，返回字符串**

fmt.Sprint(a …interface{}): **把零个或多个变量按空格进行格式化，返回字符串**

终端输入输出背后的原理

6. 终端其实是一个文件

终端相关文件的实例

os.Stdin：标准输入的文件实例，类型为*File

os.Stdout：标准输出的文件实例，类型为*File

os.Stderr：标准错误输出的文件实例，类型为*File

终端读写

7. 以文件的方式操作终端

终端读取：

File.Read(b []byte)

终端输出：

File.Write(b []byte)

File.WriteString(str string)

终端读写

8. 以文件的方式操作终端

终端读取：

File.Read(b []byte)

终端输出：

File.Write(b []byte)

File.WriteString(str string)

格式化输入

9. 从文件获取输入

fmt.Fscanf(file, format string, a…interface{}): **从文件格式化输入，空格作为分隔符，占位符和**格式化输出一致

fmt.Fscan(file, a …interface{}): **从文件获取用户输入，存储在ScanIn中的参数里，空格和换行符作为分隔符**

fmt.FscanIn(file, a …interface{}): **从文件获取用户输入，存储在ScanIn中的参数里，空格作为分隔符，遇到换行符结束**

格式化输出

10. 格式化输出到文件中

fmt.Fprintf(file, format string, a…interface{}): **格式化输出，并写入到文件中**

fmt.Fprintln(file, a …interface{}): **把零个或多个变量写入到文件中， 并换行**

fmt.Fprint(file, a …interface{}): **把零个或多个变量写入到文件**

终端读写

11. 带缓冲区的读写：

```
package main

import (
        "bufio"
        "fmt"
        "os"
)

var inputReader *bufio.Reader
var input string
var err error

func main() {
        inputReader = bufio.NewReader(os.Stdin)
        fmt.Println("Please enter some input: ")
        input, err = inputReader.ReadString('\n')
        if err == nil {
                fmt.Printf("The input was: %s\n", input)
        }
}
```

如何从终端读取带空格的字符串？

命令行参数处理

1. os.Args命令行参数的切片

```
package main

import (
    "fmt"
    "os"
)

func main() {

    who := "Alice"
    if len(os.Args) > 1 {
        who += strings.Join(os.Args[1:], " ")
    }

    fmt.Println("Good Morning", who)
}
```

命令行参数处理

2. 使用flag包获取命令行参数

```
package main

import (
	"flag"
	"fmt"
)

func parseArgs() {
	flag.IntVar(&length, "l", 16, "-l 生成密码的长度")
	flag.StringVar(&charset, "t", "num",
		`-t 制定密码生成的字符集,
		num:只使用数字[0-9],
		char:只使用英文字母[a-zA-Z],
		mix: 使用数字和字母,
		advance:使用数字、字母以及特殊字符`)
	flag.Parse()
}

func main() {
    parseArgs()
}
```

命令行参数处理

3. urfave/cli包的使用

```
package main
import (
  "fmt"
  "os"
  "github.com/urfave/cli"
)
func main() {
  app := cli.NewApp()
  app.Name = "greet"
  app.Usage = "fight the loneliness!"
  app.Action = func(c *cli.Context) error {
    fmt.Println("Hello friend!")
    return nil
  }
  app.Run(os.Args)
}
```

4. 获取命令行参数

```
package main

import (
  "fmt"
  "os"

  "github.com/urfave/cli"
)

func main() {
  app := cli.NewApp()

  app.Action = func(c *cli.Context) error {
    fmt.Printf("Hello %q", c.Args().Get(0))
    return nil
  }

  app.Run(os.Args)
}
```

命令行参数处理

4. 获取选项参数

```
package main
import (
    "fmt"
    "os"
    "github.com/urfave/cli"
)
func main() {
    var language string
    var recusive bool
    app := cli.NewApp()
    app.Flags = []cli.Flag{
        cli.StringFlag{
            Name:        "lang, l",
            Value:       "english",
            Usage:       "language for the greeting",
            Destination: &language,
        },
        cli.BoolFlag{
            Name:        "recusive, r",
            Usage:       "recusive for the greeting",
            Destination: &recusive,
        },
    }
    app.Action = func(c *cli.Context) error {
        var cmd string
        if c.NArg() > 0 {
            cmd = c.Args()[0]
            fmt.Println("cmd is ", cmd)
        }
        fmt.Println("recusive is ", recusive)
        fmt.Println("language is ", language)
        return nil
    }
    app.Run(os.Args)
}
```

课后练习

1. 实现一个简易的计算器，支持加减乘除以及带括号的计算表达式，用户从终端输入表达式，程序输出计算结果。