Go培训第七天

tony

Outline

1. 终端读写
2. 文件读写
3. 课后作业

终端读写

1. 终端读写

   操作终端相关文件句柄常量

   os.Stdin：标准输入

   os.Stdout：标准输出

   os.Stderr：标准错误输出

终端读写

2. 终端读写示例：

```
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)
}
```

终端读写

3. 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)
	}
}
```

终端读写

4. 练习，从终端读取一行字符串，统计英文、数字、空格以及其他字符的数量。

文件读写

1. os.File封装所有文件相关操作，之前讲的 os.Stdin,os.Stdout, os.Stderr都是 *os.File

   a. 打开一个文件进行读操作： os.Open(name string) (*File, error)

   b. 关闭一个文件：File.Close()

文件读写

3. 文件操作示例

```
package main

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

func main() {

	inputFile, err := os.Open("input.dat")
	if err != nil {
		fmt.Printf("open file err:%v\n", err)
		return
	}

	defer inputFile.Close()
	inputReader := bufio.NewReader(inputFile)
	for {
		inputString, readerError := inputReader.ReadString('\n')
		if readerError == io.EOF {
			return
		}
		fmt.Printf("The input was: %s", inputString)
	}
}
```

文件读写

4. 读取整个文件示例

```
package main

import (
        "fmt"
        "io/ioutil"
        "os"
)

func main() {

        inputFile := "products.txt"
        outputFile := "products_copy.txt"
        buf, err := ioutil.ReadFile(inputFile)
        if err != nil {
                fmt.Fprintf(os.Stderr, "File Error: %s\n", err)
                return
        }

        fmt.Printf("%s\n", string(buf))
        err = ioutil.WriteFile(outputFile, buf, 0x644)
        if err != nil {
                panic(err.Error())
        }
}
```

文件读写

5. 读取压缩文件示例

```
package main

import (
	"bufio"
	"compress/gzip"
	"fmt"
	"os"
)
func main() {
	fName := "MyFile.gz"
	var r *bufio.Reader
	fi, err := os.Open(fName)
	if err != nil {
		fmt.Fprintf(os.Stderr, "%v, Can’t open %s: error: %s\n", os.Args[0], fName, err)
		os.Exit(1)
	}
	fz, err := gzip.NewReader(fi)
	if err != nil {
		fmt.Fprintf(os.Stderr, "open gzip failed, err: %v\n", err)
		return
	}
	r = bufio.NewReader(fz)
	for {
		line, err := r.ReadString('\n')
		if err != nil {
			fmt.Println("Done reading file")
			os.Exit(0)
		}
		fmt.Println(line)
	}
}
```

文件读写

6. 文件写入

os.OpenFile(“output.dat”, os.O_WRONLY|os.O_CREATE, 0666)

第二个参数: 文件打开模式:

1. os.O_WRONLY：只写

2. os.O_CREATE：创建文件

3. os.O_RDONLY：只读

4. os.O_RDWR：读写

5. os.O_TRUNC：清空

第三个参数：权限控制：

r ——> 004

w——> 002

x——> 001

7. 文件写入示例

```
package main

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

func main() {
	outputFile, outputError := os.OpenFile("output.dat",
os.O_WRONLY|os.O_CREATE, 0666)
	if outputError != nil {
		fmt.Printf("An error occurred with file creation\n")
		return
	}

	defer outputFile.Close()
	outputWriter := bufio.NewWriter(outputFile)
	outputString := "hello world!\n"
	for i := 0; i < 10; i++ {
		outputWriter.WriteString(outputString)
	}
	outputWriter.Flush()
}
```

8. 拷贝文件

```go
package main

import (
	"fmt"
	"io"
	"os"
)

func main() {

	CopyFile("target.txt", "source.txt")
	fmt.Println("Copy done!")
}

func CopyFile(dstName, srcName string) (written int64, err error) {
	src, err := os.Open(srcName)
	if err != nil {
		return
	}
	defer src.Close()
	dst, err := os.OpenFile(dstName, os.O_WRONLY|os.O_CREATE, 0644)
	if err != nil {
		return
	}
	defer dst.Close()
	return io.Copy(dst, src)
}
```

命令行参数

9. os.Args是一个string的切片，用来存储所有的命令行参数

命令行参数

10. flag包的使用，用来解析命令行参数：

```
flag.BoolVar(&test, "b", false, "print on newline")
flag.StringVar(&str, "s", "", "print on newline")
flag.IntVar(&count, "c", 1001, "print on newline")
```

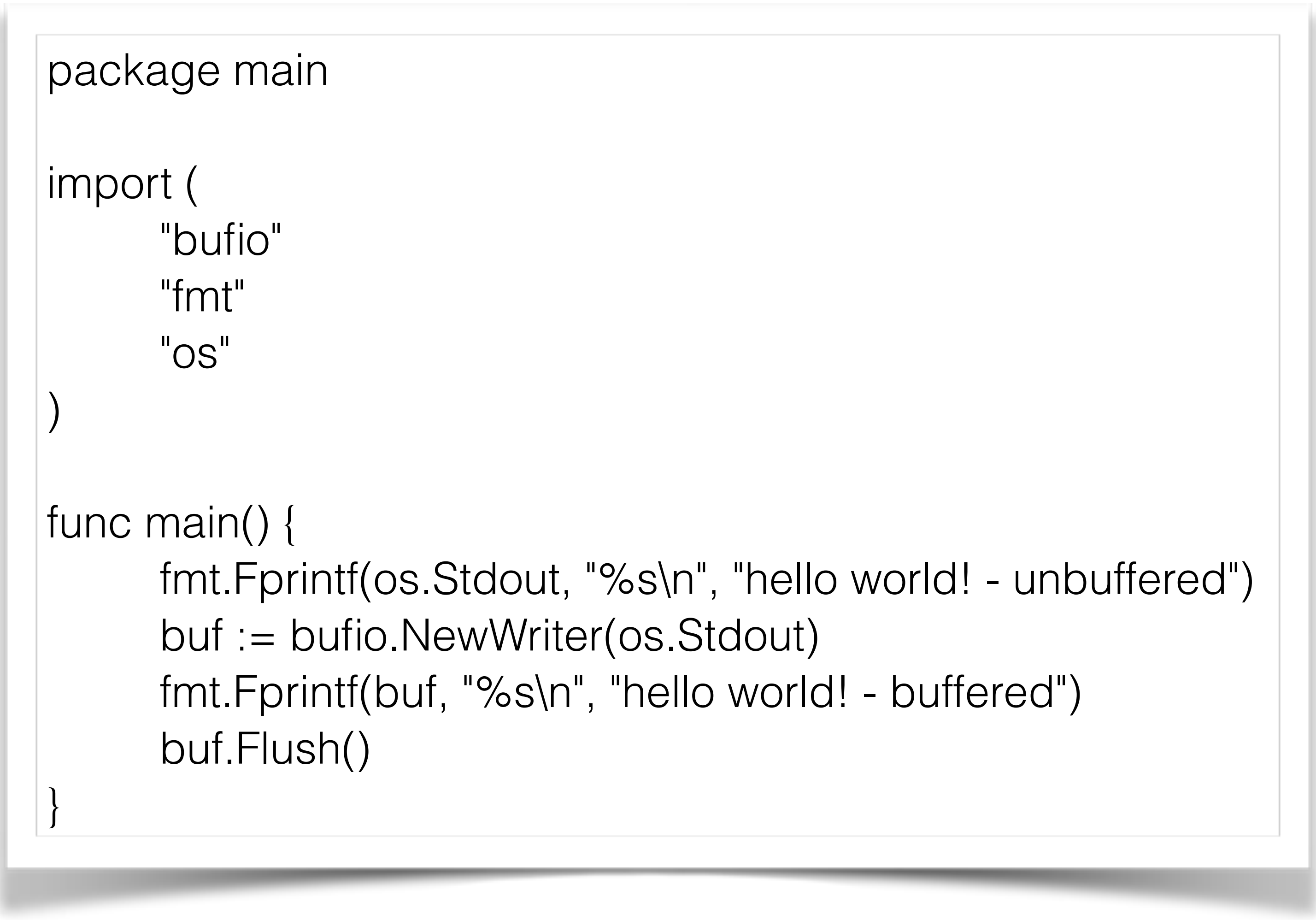
11.命令行参数解析

```go
package main

import (
	"flag" // command line option parser
	"fmt"
)

func main() {

	var test bool
	var str string
	var count int
	flag.BoolVar(&test, "b", false, "print on newline")
	flag.StringVar(&str, "s", "", "print on newline")
	flag.IntVar(&count, "c", 1001, "print on newline")
	flag.Parse()

	fmt.Println(test)
	fmt.Println(str)
	fmt.Println(count)
}
```

12.带缓冲区的文件读写

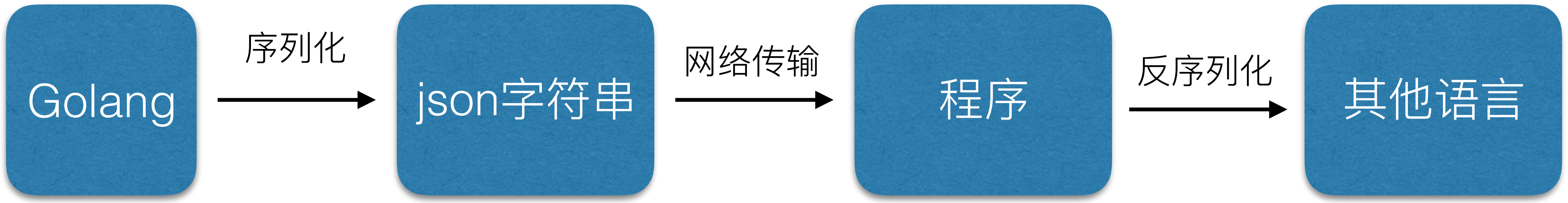
```
package main

import (
	"bufio"
	"flag"
	"fmt"
	"io"
	"os"
)

func cat(r *bufio.Reader) {
	for {
		buf, err := r.ReadBytes('\n')
		if err == io.EOF {
			break
		}
		fmt.Fprintf(os.Stdout, "%s", buf)

		return
	}
}

func main() {
	flag.Parse()
	if flag.NArg() == 0 {
		cat(bufio.NewReader(os.Stdin))
	}
	for i := 0; i < flag.NArg(); i++ {
		f, err := os.Open(flag.Arg(i))
		if err != nil {
			fmt.Fprintf(os.Stderr, "%s:error reading from %s: %s\n",
				os.Args[0], flag.Arg(i), err.Error())
		}
		continue
	}
	cat(bufio.NewReader(f))
}
```

12.带缓冲区的终端读写



```
package main

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

func main() {
	fmt.Fprintf(os.Stdout, "%s\n", "hello world! - unbuffered")
	buf := bufio.NewWriter(os.Stdout)
	fmt.Fprintf(buf, "%s\n", "hello world! - buffered")
	buf.Flush()
}
```

13.Json数据协议



14.Json数据协议

1. 导入包：Import “encoding/json”

2. 序列化: json.Marshal(data interface{})

3. 反序列化: json.UnMarshal(data []byte, v interface{})

15.json序列化结构体

16.json序列化map

错误处理

17.定义错误

```
package main

import (
	"errors"
	"fmt"
)

var errNotFound error = errors.New("Not found error")

func main() {
	fmt.Printf("error: %v", errNotFound)
}
```

21.Panic&Recover

```
package main

import (
	"fmt"
)

func badCall() {
	panic("bad end")
}

func test() {
	defer func() {
		if e := recover(); e != nil {
			fmt.Printf("Panicking %s\r\n", e)
		}
	}()
	badCall()
	fmt.Printf("After bad call\r\n")
}

func main() {
	fmt.Printf("Calling test\r\n")
	test()
	fmt.Printf("Test completed\r\n")
}
```

课后工作

1. 实现一个图书管理系统v3，具有以下功能：

    a. 增加持久化存储的功能

    b. 增加日志记录的功能