Day6 文件&数据库操作

tony

目录

1.文件基本操作

2. Mysql基本操作

3. Redis基本操作

4. 课后作业

文件读写

1. os.File封装所有文件相关操作，是一个自定义的struct

    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

2. 文件操作示例

```
func Read(filename string)  (string, error){
    //获得一个file
    f, err := os.Open(filename)
    if err != nil {
        fmt.Println("read fail")
        return ""
    }
    //把file读取到缓冲区中
    defer f.Close()
    var content []byte
    var buf [1024]byte

    for {
        //从file读取到buf中，n表示本次读到的字节数
        n, err := f.Read(buf)
        if err != nil && err != io.EOF{
            fmt.Println("read buf fail", err)
            return "", err
        }
        //说明读取结束
        if err == io.EOF{
            break
        }
        //读取到最终的缓冲区中
        content = append(content, buf[:n]...)
    }

    return string(content), nil
}
```

文件读写

3. 文件操作示例，使用bufio提高文件读取性能

```
func Read(filename string) (string, error) {
    fi, err := os.Open(filename)
    if err != nil {
        return “”, err
    }
    defer fi.Close()

    r := bufio.NewReader(fi)
    var content []byte
    var buf [1024]byte

    for {
        n, err := r.Read(buf)
        if err != nil && err != io.EOF {
            return “”, err
        }
        if err == io.EOF {
            break
        }

        content = append(content, buf...)
    }
    return string(content),  nil
}
```

文件读写

4. 文件操作示例，使用ioutil读取整个文件

```
func Read(filename string)  (string, error){
    content, err := ioutil.ReadFile(filename)
    if err != nil {
        return “”, err
    }
    return string(content)
}
```

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

文件读写

7. 文件写入

os.Create(“output.dat”)

文件读写

8. 文件写入实例

```
func CheckFileExist(fileName string) bool {
    _, err := os.Stat(fileName)
    if os.IsNotExist(err) {
        return false
    }
    return true
}
func Write(filename string,  content string) (error) {
    var f *os.File
    var err error

    if CheckFileExist(fileName) {  //文件存在
        f, err = os.OpenFile(fileName, os.O_APPEND, 0666) //打开文件
        if err != nil{
            return err
        }
    }else {  //文件不存在
        f, err = os.Create(fileName) //创建文件
        if err != nil {
            return err
        }
    }

    _, err := io.WriteString(f, strTest)
    if err != nil {
        return err
    }
    return nil
}
```

文件读写

9. 文件写入实例，使用ioutil直接写入

```
func Write()  {
    fileName := "file/test2"
    strTest := "测试测试"
    var d = []byte(strTest)
    err := ioutil.WriteFile(fileName, d, 0666)
    if err != nil {
        fmt.Println("write fail")
    }
    fmt.Println("write success")
}
```

文件读写

10.文件写入实例，使用bufio进行文件写入

```
func Write()  {
    fileName := "file/test3"
    f, err3 := os.Create(fileName) //创建文件
    if err3 != nil{
        fmt.Println("create file fail")
    }
    w := bufio.NewWriter(f) //创建新的 Writer 对象
    n4, err3 := w.WriteString("bufferedn")
    fmt.Printf("写入 %d 个字节n", n4)
    w.Flush()
    f.Close()
}
```

11.拷贝文件

```
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)
}
```

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。

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

```
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()
}
```

7. Golang中MYSQL驱动

A. https://github.com/go-sql-driver/mysql

B. Go本身不提供具体数据库驱动，只提供驱动接口和管理。

C. 各个数据库驱动需要第三方实现，并且注册到Go中的驱动管理中。

8. Mysql驱动，注册示例

GitHub, Inc. [US] | https://github.com/go-sql-driver/mysql/blob/master/driver.go

```
139                     maxap, err := mc.getSystemVar("max_allowed_packet")
140                     if err != nil {
141                             mc.Close()
142                             return nil, err
143                     }
144                     mc.maxAllowedPacket = stringToInt(maxap) - 1
145             }
146             if mc.maxAllowedPacket < maxPacketSize {
147                     mc.maxWriteSize = mc.maxAllowedPacket
148             }
149
150             // Handle DSN Params
151             err = mc.handleParams()
152             if err != nil {
153                     mc.Close()
154                     return nil, err
155             }
156
157             return mc, nil
158     }
159
160     func init() {
161             sql.Register("mysql", &MySQLDriver{})
162     }
```

9. 导入Mysql驱动

```
import (
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)
```

10. 连接数据库

```
func main() {
    dsn := "user:password@tcp(127.0.0.1:3306)/test"
    db, err := sql.Open("mysql",dsn)
    if err != nil {
        panic(err)
        return
    }
    defer db.Close()
}
```

11. 创建表

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT ‘’,
  `age` int(11) DEFAULT '0’,
   PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT  CHARSET=utf8mb4;
```

MYSQL开发

12. sql查询

A. 单行查询， Db.QueryRow

B 多行查询， Db.Query

13. Mysql 插入、更新和删除

A. Db.Exec(query， args…interface{})(Result, error)

B. 插入的主键id获取，Result. LastInsertId()

MYSQL开发

14. Mysql预处理

A. 一般sql处理流程

1. 客户端拼接好sql语句
2. 客户端发送sql语句到mysql服务器
3. mysql服务器解析sql语句并执行，把执行结果发送给客户端

B. 预处理处理流程

1. 把sql分为两部分，命令部分和数据部分。
2. 首先把命令部分发送给mysql服务器，mysql进行sql预处理
3. 然后把数据部分发送给mysql服务器，mysql进行占位符替换
4. mysql服务器执行sql语句并返回结果给客户端

15. Mysql预处理优势

A. 同一条sql反复执行，性能会很高。

B. 避免sql注入问题

16. Mysql预处理实例

A. 查询操作，

1. Db.Prepare(sql string) (*sql.Stmt, error)

2. Stmt.Query()

B. 更新操作（插入、更新、delete），

1. Db.Prepare(sql string) (*sql.Stmt, error)

2. Stmt.Exec()

17. Mysql事务

A. 应用场景，

1. 同时更新，多个表。
2. 同时更新多行数据。

B. 事务的ACID

1. 原子性
2. 一致性
3. 隔离性
4. 持久性

18. Mysql事务实例

A. Db.Begin()，开启一个事务

B. Db.Commit()，提交一个事务

C. Db.Rollback()，回滚一个事务

19. sqlx库介绍与使用

A. 使用更简单

B. 支持多数据库，mysql、postgresql、oracle、sqlite

20. sqlx库介绍与使用

A. 查询， sqlx.DB.Get和sqlx.DB.Select

B. 更新、插入和删除，sqlx.DB.Exec

C. 事务，sqlx.DB.Begin()、sqlx.DB.Commit、sqlx.DB.rollback

21. sql注入分析

A. Select *from user where name = ‘%s’，构造name=“1 ' or 1 = 1 or ‘”

B. 构造name=123' and (select count(*) from user) > 10#

C. 构造name=123' union select *from user #

避免手动拼接sql，使用占位符或预处理！

Redis使用和开发

11. Redis开发

A. 使用第三方的redis库， github.com/garyburd/redigo/redis

B. 连接redis

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
}
```

12. Redis开发

A. Set操作， 设置key-value

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("Set", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("Get", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

13. Redis开发

A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

14. Redis开发

A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

15. Redis开发

A. Mset操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("MSet", "abc", 100, "efg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Ints(c.Do("MGet", "abc", "efg"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 for _, v := range r {
 fmt.Println(v)
 }
}
```

16. Redis开发

A. 设置过期时间

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("expire", "abc", 10)
 if err != nil {
 fmt.Println(err)
 return
 }
}
```

17. Redis开发

A. 队列操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("lpush", "book_list", "abc", "ceg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.String(c.Do("lpop", "book_list"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

Redis使用和开发

18. Redis开发

A. Redis连接池

```
//初始化一个pool
func newPool(server, password string) *redis.Pool {
    return &redis.Pool{
        MaxIdle:     64,
        MaxActive:   1000,
        IdleTimeout: 240 * time.Second,
        Dial: func() (redis.Conn, error) {
            c, err := redis.Dial("tcp", server)
            if err != nil {
                return nil, err
                }
                /*
            if _, err := c.Do("AUTH", password); err != nil {
                c.Close()
                return nil, err
            }*/
            return c, err
        },
        TestOnBorrow: func(c redis.Conn, t time.Time) error {
            if time.Since(t) < time.Minute {
                return nil
            }
            _, err := c.Do("PING")
            return err
        },
    }
}
```

课后练习

1. 实现一个图书管理系统，具有以下功能：

   a. 书籍录入功能，书籍信息包括书名、副本数、作者、出版日期

   b. 书籍查询功能，按照书名、作者、出版日期等条件检索

   c. 学生信息管理功能，管理每个学生的姓名、年级、身份证、性别、借了什么书等信息

   d. 借书功能，学生可以查询想要的书籍，进行借出

   e. 书籍管理功能，可以看到每种书被哪些人借出了

   f. 数据采用Mysql，使用数据库实现图书管理相关功能