

# Golang 文件 目录操作

主讲教师：（大地）

合作网站：[www.itying.com](http://www.itying.com) （IT 营）

我的专栏：<https://www.itying.com/category-79-b0.html>

一、打开和关闭文件.....	1
二、file.Read() 读取文件.....	2
三、循环读取.....	3
四、bufio 读取文件.....	4
五、ioutil 读取整个文件.....	5
六、文件写入操作.....	5
七、文件重命名.....	7
八、复制文件.....	8
九、创建目录.....	10
十、删除目录和文件.....	11

## 一、打开和关闭文件

os.Open()函数能够打开一个文件，返回一个\*File 和一个 err。操作完成文件对象以后一定要记得关闭文件

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // 只读方式打开当前目录下的 main.go 文件
```

```
file, err := os.Open("./main.go")

if err != nil {

    fmt.Println("open file failed!, err:", err)

    return

}

fmt.Println(file) //&{0xc000078780}

defer file.Close() // 关闭文件

}
```

为了防止文件忘记关闭，我们通常使用 `defer` 注册文件关闭语句。

## 二、file.Read() 读取文件

### 1、基本使用

`Read` 方法定义如下：

```
func (f *File) Read(b []byte) (n int, err error)
```

它接收一个字节切片，返回读取的字节数和可能的具体错误，读到文件末尾时会返回 0 和 `io.EOF`。举个例子：

```
package main
import (
    "fmt"
    "io"
    "os"
)
func main() {
    // 只读方式打开当前目录下的 main.go 文件
    file, err := os.Open("./main.go")
    if err != nil {
        fmt.Println("open file failed!, err:", err)
        return
    }
    defer file.Close()
    // 使用 Read 方法读取数据，注意一次只会读取 128 个字节
    var tmp = make([]byte, 128)
```

```
n, err := file.Read(tmp)
if err == io.EOF {
    fmt.Println("文件读完了")
    return
}
if err != nil {
    fmt.Println("read file failed, err:", err)
    return
}
fmt.Printf("读取了%d 字节数据\n", n)
fmt.Println(string(tmp[:n]))
}
```

### 三、循环读取

使用 for 循环读取文件中的所有数据

```
func main() {
    // 只读方式打开当前目录下的 main.go 文件
    file, err := os.Open("./main.go")
    if err != nil {
        fmt.Println("open file failed!, err:", err)
        return
    }
    defer file.Close()
    // 循环读取文件
    var content []byte
    var tmp = make([]byte, 128)
    for {
        n, err := file.Read(tmp)
        if err == io.EOF {
            fmt.Println("文件读完了")
            break
        }
        if err != nil {
            fmt.Println("read file failed, err:", err)
            return
        }
        content = append(content, tmp[:n]...)
    }
    fmt.Println(string(content))
}
```

## 四、bufio 读取文件

bufio 是在 file 的基础上封装了一层 API，支持更多的功能。

```
package main

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

// bufio 按行读取示例
func main() {
    file, err := os.Open("C:/test.txt")
    if err != nil {
        fmt.Println("open file failed, err:", err)
        return
    }
    defer file.Close()
    reader := bufio.NewReader(file)
    for {
        line, err := reader.ReadString('\n') //注意是字符
        if err == io.EOF {
            if len(line) != 0 {
                fmt.Println(line)
            }
            fmt.Println("文件读完了")
            break
        }
        if err != nil {
            fmt.Println("read file failed, err:", err)
            return
        }
        fmt.Print(line)
    }
}
```

## 五、ioutil 读取整个文件

io/ioutil 包的 ReadFile 方法能够读取完整的文件，只需要将文件名作为参数传入。

```
package main

import (
    "fmt"
    "io/ioutil"
)

// ioutil.ReadFile 读取整个文件

func main() {
    content, err := ioutil.ReadFile("./main.go")

    if err != nil {
        fmt.Println("read file failed, err:", err)
        return
    }

    fmt.Println(string(content))
}
```

## 六、文件写入操作

os.OpenFile()函数能够以指定模式打开文件，从而实现文件写入相关功能。

```
func OpenFile(name string, flag int, perm FileMode) (*File, error) {
    ...
}
```

其中：

**name:** 要打开的文件名 **flag:** 打开文件的模式。 模式有以下几种：

模式	含义
os.O_WRONLY	只写
os.O_CREATE	创建文件
os.O_RDONLY	只读
os.O_RDWR	读写
os.O_TRUNC	清空
os.O_APPEND	追加

perm: 文件权限，一个八进制数。r（读）04，w（写）02，x（执行）01。

## 1、Write 和 WriteString

```
package main

import (
    "fmt"
    "os"
)

func main() {
    file, err := os.OpenFile("C:/test.txt", os.O_CREATE|os.O_RDWR, 0666)
    if err != nil {
        fmt.Println("open file failed, err:", err)
        return
    }
    defer file.Close()
    str := "你好 golang"
    file.Write([]byte(str))           //写入字节切片数据
    file.WriteString("直接写入的字符串数据") //直接写入字符串数据
}
```

## 2、bufio.NewWriter

```
package main

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

```
func main() {  
    file, err := os.OpenFile("C:/test.txt", os.O_CREATE|os.O_TRUNC|os.O_WRONLY, 0666)  
    if err != nil {  
        fmt.Println("open file failed, err:", err)  
        return  
    }  
    defer file.Close()  
    writer := bufio.NewWriter(file)  
    for i := 0; i < 10; i++ {  
        writer.WriteString("你好 goLang\r\n") //将数据先写入缓存  
    }  
    writer.Flush() //将缓存中的内容写入文件（注意）  
}
```

### 3、ioutil.WriteFile

```
package main  
import (  
    "fmt"  
    "io/ioutil"  
)  
func main() {  
    str := "hello goLang"  
    err := ioutil.WriteFile("C:/test.txt", []byte(str), 0666)  
    if err != nil {  
        fmt.Println("write file failed, err:", err)  
        return  
    }  
}
```

## 七、文件重命名

```
err := os.Rename("C:/test1.txt", "D:/test1.txt") //只能同盘操作  
  
if err != nil {  
    fmt.Println(err)  
}
```

```
}
```

## 八、复制文件

第一种复制文件方法：ioutil 进行复制

```
package main

import (
    "fmt"
    "io/ioutil"
)

//自己编写一个函数，接收两个文件路径 srcFileName dstFileName
func CopyFile(dstFileName string, srcFileName string) (err error) {
    input, err := ioutil.ReadFile(srcFileName)

    if err != nil {
        fmt.Println(err)
        return err
    }

    err = ioutil.WriteFile(dstFileName, input, 0644)

    if err != nil {
        fmt.Println("Error creating", dstFileName)
        fmt.Println(err)
        return err
    }

    return nil
}

func main() {
```



```
srcFile := "c:/test1.zip"

dstFile := "D:/test1.zip"

err := CopyFile(dstFile, srcFile)

if err == nil {

    fmt.Printf("拷贝完成\n")

} else {

    fmt.Printf("拷贝错误 err=%v\n", err)

}

}
```

第二种复制文件方法流的方式复制:

```
package main

import (

    "fmt"

    "io"

    "os"

)

//自己编写一个函数，接收两个文件路径 srcFileName dstFileName

func CopyFile(dstFileName string, srcFileName string) (err error) {

    source, _ := os.Open(srcFileName)

    destination, _ := os.OpenFile(dstFileName, os.O_CREATE|os.O_WRONLY, 0666)

    buf := make([]byte, 128)

    for {

        n, err := source.Read(buf)

        if err != nil && err != io.EOF {
```

```
        return err
    }

    if n == 0 {
        break
    }

    if _, err := destination.Write(buf[:n]); err != nil {
        return err
    }
}

}

func main() {
    //调用 CopyFile 完成文件拷贝

    srcFile := "c:/000.avi"

    dstFile := "D:/000.avi"

    err := CopyFile(dstFile, srcFile)

    if err == nil {
        fmt.Printf("拷贝完成\n")
    } else {
        fmt.Printf("拷贝错误 err=%v\n", err)
    }
}
```

## 九、创建目录

一次创建一个目录

```
err := os.Mkdir("./abc", 0666)

if err != nil {
```

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

一次创建多个目录

```
err := os.MkdirAll("dir1/dir2/dir3", 0666) //创建多级目录
if err != nil {
    fmt.Println(err)
}
```

## 十、删除目录和文件

### 1、删除一个目录或者文件

```
err := os.Remove("t.txt")
if err != nil {
    fmt.Println(err)
}
```

### 2、一次删除多个目录或者文件

```
err := os.RemoveAll("aaa")
if err != nil {
    fmt.Println(err)
}
```