Golang文件操作整理

2015-05-27 17:05 Goden 阅读 3808 次 1 人喜欢 0 条评论 (/articles/3154#commentForm) ☆ 收藏

最近做的一点事情，用到了golang中不少文件操作的相关内容，创建，删除，遍历，压缩之类的，这里整理整理，希望能掌握的系统一点，把模糊的地方理清楚。

基本操作

文件创建

创建文件的时候，一定要注意权限问题，一般默认的文件权限是 0666 关于权限的相关内容，具体可以参考鸟叔p141 这里还是再回顾下，文件属性 r w x r w x r w x，第一位是文件属性，一般常用的 "-" 表示的是普通文件，"d"表示的是目录，golang里面使用 os.Create 创建文件的时候貌似只能使用0xxx的形式。比如0666就表示创建了一个普通文件，文件所有者的权限，文件所属用户组的权限，以及其他人对此文件的权限都是110表示可读可写，不可执行。

文件删除

文件删除的时候，不管是普通文件还是目录文件，都可以用 err:=os.Remove(filename) 这样的操作来执行。当然要是想移除整个文件夹，直接使用 RemoveAll(path string) 操作即可。可以看一下RemoveAll函数的内部实现，整体上就是遍历，递归的操作过程，其他的类似的文件操作都可以用类似的模板来实现，下面以RemoveAll函数为模板，进行一下具体的分析，注意考虑到各种情况：

```
func RemoveAll(path string) error {
// Simple case: if Remove works, we're done.
//先尝试一下remove如果是普通文件 直接删掉 报错 则可能是目录中还有子文件
err := Remove(path)
//没错或者路径不存在 直接返回 nil
if err == nil || IsNotExist(err) {
        return nil
}

// Otherwise, is this a directory we need to recurse into?
// 目录里面还有文件 需要递归处理
// 注意Lstat和stat函数的区别，两个都是返回文件的状态信息
//Lstat多了处理Link文件的功能，会返回Linked文件的信息，而state直接返回的是Link文件所指向的文件的信息
dir, serr := Lstat(path)
if serr != nil {
        if serr, ok := serr.(*PathError); ok && (IsNotExist(serr.Err) || serr.Err == sysca
ll.ENOTDIR) {
                return nil
        }
        return serr
}
//不是目录
if !dir.IsDir() {
        // Not a directory; return the error from Remove.
        return err
}
```

```
// Directory.
fd, err := Open(path)
if err != nil {
        if IsNotExist(err) {
                // Race. It was deleted between the Lstat and Open.
                // Return nil per RemoveAll's docs.
                return nil
        }
        return err
}

// Remove contents & return first error.
err = nil
//递归遍历目录中的文件 如果参数n<=0则将全部的信息存入到一个slice中返回
//如果参数n>0则至多返回n个元素的信息存入到slice当中
//还有一个类似的函数是Readdir 这个返回的是 目录中的内容的Fileinfo信息

for {
        names, err1 := fd.Readdirnames(100)
        for _, name := range names {
                err1 := RemoveAll(path + string(PathSeparator) + name)
                if err == nil {
                        err = err1
                }
        }
        //遍历到最后一个位置
        if err1 == io.EOF {
                break
        }
        // If Readdirnames returned an error, use it.
        if err == nil {
                err = err1
        }
        if len(names) == 0 {
                break
        }
}

// Close directory, because windows won't remove opened directory.
fd.Close()
//递归结束 当前目录下位空 删除当前目录
// Remove directory.
err1 := Remove(path)
if err1 == nil || IsNotExist(err1) {
        return nil
}
if err == nil {
        err = err1
}
return err
}
```

文件状态

从文件中写入写出内容

这一部分较多的涉及I/O的相关操作，系统的介绍放在I/O那部分来整理，大体上向文件中读写内容的时候有三种方式：

1、在使用 f, err := os.Open(file_path) 打开文件之后直接使用 f.read() f.write() 结合自定义的buffer每次从文件中读入/读出固定的内容

2、使用ioutl的readFile和writeFile方法

3、使用bufio采用带有缓存的方式进行读写，比如通过 info:=bufio.NewReader(f) 将实现了io.Reader的接口的实例加载上来之后，就可以使用info.ReadLine（）来每次实现一整行的读取，直到err信息为io.EOF时，读取结束

这个blog (http://david-je.iteye.com/blog/1988940)对三种文件操作的读入速度进行了比较，貌似读取大文件的时候采用ioutil的时候效率要高些。

每种方式都有不同的适用情况，下面是分别用三种方式进行读出操作的例子，对于写入文件的操作，可以参考读出操作来进行：

```
package main

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

func check(e error) {
if e != nil {
        panic(e)
}
}

func main() {
//查看当前的工作目录路径 得到测试文件的绝对路径
current_dir, _ := os.Getwd()
fmt.Println(current_dir)
file_path := current_dir + "/temp.txt"

//方式一：
//通过ioutil直接通过文件名来加载文件
//一次将整个文件加载进来 粒度较大 err返回为nil的时候 文件会被成功加载
dat, err := ioutil.ReadFile(file_path)
//若加载的是一个目录 会返回[]os.FileInfo的信息
//ioutil.ReadDir()
check(err)
//the type of data is []uint
fmt.Println(dat)
//将文件内容转化为string输出
fmt.Println(string(dat))

//方式二：
//通过os.Open的方式得到 *File 类型的变量
//貌似是一个指向这个文件的指针 通过这个指针 可以对文件进行更细粒度的操作
f, err := os.Open(file_path)
check(err)
//手工指定固定大小的buffer 每次通过buffer来 进行对应的操作
buffer1 := make([]byte, 5)
//从文件f中读取len(buffer1)的信息到buffer1中 返回值n1是读取的byte的长度
n1, err := f.Read(buffer1)
check(err)
```

```
fmt.Printf("%d bytes: %s\n", n1, string(buffer1))

//通过f.seek进行更精细的操作 第一个参数表示offset为6 第二个参数表示文件起始的相对位置
//之后再读就从o2位置开始往后读信息了
o2, err := f.Seek(6, 0)
check(err)
buffer2 := make([]byte, 2)
//读入了n2长度的信息到buffer2中
n2, err := f.Read(buffer2)
check(err)
fmt.Printf("%d bytes after %d position : %s\n", n2, o2, string(buffer2))

//通过io包种的函数 也可以实现类似的功能
o3, err := f.Seek(6, 0)
check(err)
buffer3 := make([]byte, 2)
n3, err := io.ReadAtLeast(f, buffer3, len(buffer3))
check(err)
fmt.Printf("%d bytes after %d position : %s\n", n3, o3, string(buffer3))

//方式三
//通过bufio包来进行读取 bufio中又许多比较有用的函数 比如一次读入一整行的内容

//调整文件指针的起始位置到最开始的地方
_, err = f.Seek(10, 0)
check(err)
r4 := bufio.NewReader(f)

//读出从头开始的5个字节
b4, err := r4.Peek(5)
check(err)
//fmt.Println(string(b4))
fmt.Printf("5 bytes : %s\n", string(b4))

//调整文件到另一个地方
_, err = f.Seek(0, 0)
check(err)
r5 := bufio.NewReader(f)
//读出从指针所指位置开始的5个字节
b5, err := r5.Peek(5)
check(err)
//fmt.Println(string(b4))
fmt.Printf("5 bytes : %s\n", string(b5))

//测试bufio的其他函数

for {
        //读出内容保存为string 每次读到以'\n'为标记的位置
        line, err := r5.ReadString('\n')
        fmt.Print(line)
        if err == io.EOF {
                break
        }
}
//ReadLine() ReadByte() 的用法都是类似 一般都是当err为io.EOF的时候
//读入内容就结束
//感觉实际用的时候 还是通过方式三比较好 粒度正合适 还有多种处理输入的方式
```

```
f.Close()

}
```

高级操作

文件打包，文件解压，文件遍历，这些相关的操作基本上都可以参考RemoveAll的方式来进行，就是递归加遍历的方式。
下面是文件压缩的一个实现：

```
//将文件夹中的内容打包成 .gz.tar 文件
package main

import (
"archive/tar"
"compress/gzip"
"fmt"
"io"
"os"
)

//将fi文件的内容 写入到 dir 目录之下 压缩到tar文件之中
func Filecompress(tw *tar.Writer, dir string, fi os.FileInfo) {

//打开文件 open当中是 目录名称/文件名称 构成的组合
filename := dir + "/" + fi.Name()
fmt.Println("the last one:", filename)
fr, err := os.Open(filename)
fmt.Println(fr.Name())
if err != nil {
        panic(err)
}
defer fr.Close()

hdr, err := tar.FileInfoHeader(fi, "")

hdr.Name = fr.Name()
if err = tw.WriteHeader(hdr); err != nil {
        panic(err)
}
//bad way
//      //信息头部 生成tar文件的时候要先写入tar结构体
//      h := new(tar.Header)
//      //fmt.Println(reflect.TypeOf(h))

//      h.Name = fi.Name()
//      h.Size = fi.Size()
//      h.Mode = int64(fi.Mode())
//      h.ModTime = fi.ModTime()

//      //将信息头部的内容写入
//      err = tw.WriteHeader(h)
//      if err != nil {
//              panic(err)
//      }

//copy(dst Writer,src Reader)
_, err = io.Copy(tw, fr)
```

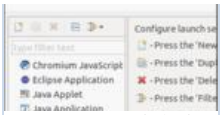
```
if err != nil {
        panic(err)
}
//打印文件名称
fmt.Println("add the file: " + fi.Name())

}

//将目录中的内容递归遍历 写入tar 文件中
func Dircompress(tw *tar.Writer, dir string) {
fmt.Println(dir)
//打开文件夹
dirhandle, err := os.Open(dir + "/")
//fmt.Println(dir.Name())
//fmt.Println(reflect.TypeOf(dir))
if err != nil {
        panic(err)
}
defer dirhandle.Close()

fis, err := dirhandle.Readdir(0)
//fis的类型为 []os.FileInfo

//也可以通过Readdirnames来读入所有子文件的名称
//但是这样 再次判断是否为文件的时候 需要通过Stat来得到文件的信息
//返回的就是os.File的类型

if err != nil {
        panic(err)
}

//遍历文件列表 每一个文件到要写入一个新的*tar.Header
//var fi os.FileInfo
for _, fi := range fis {
        fmt.Println(fi.Name())

        if fi.IsDir() {

                newname := dir + "/" + fi.Name()
                fmt.Println("using dir")
                fmt.Println(newname)
                //这个样直接continue就将所有文件写入到了一起 没有层级结构了
                //Filecompress(tw, dir, fi)
                Dircompress(tw, newname)

        } else {
                //如果是普通文件 直接写入 dir 后面已经有了 /
                Filecompress(tw, dir, fi)
        }

}

}

//在tardir目录中创建一个.tar.gz文件 存放压缩之后的文件
func Dirtotar(sourcedir string, tardir string, tarname string) {
//file write 在tardir目录下创建
fw, err := os.Create(tardir + "/" + tarname + ".tar.gz")
```

```
//type of fw is *os.File
//      fmt.Println(reflect.TypeOf(fw))
if err != nil {
        panic(err)

}
defer fw.Close()

//gzip writer
gw := gzip.NewWriter(fw)
defer gw.Close()

//tar write
tw := tar.NewWriter(gw)

fmt.Println("源目录：", sourcedir)
Dircompress(tw, sourcedir)

//通过控制写入流 也可以控制 目录结构 比如将当前目录下的Dockerfile文件单独写在最外层
fileinfo, err := os.Stat("tarrepo" + "/" + "testDockerfile")
fmt.Println("the file name:", fileinfo.Name())
if err != nil {
        panic(err)

}
//比如这里将Dockerfile放在 tar包中的最外层 会注册到tar包中的 /tarrepo/testDockerfile 中
Filecompress(tw, "tarrepo", fileinfo)
//Filecompress(tw, "systempdir/test_testwar_tar/", fileinfo)

fmt.Println("tar.gz packaging OK")

}

func main() {
//      workdir, _ := os.Getwd()
//      fmt.Println(workdir)
Dirtotar("testdir", "tarrepo", "testtar")

}
```

本文来自：博客园 *(/wr?u=http://www.cnblogs.com)*

感谢作者：*Goden*

查看原文：*Golang*文件操作整理 *(/wr?u=http%3a%2f%2fwww.cnblogs.com%2fGoden%2fp%2f4533908.html)*


Golang Go语言断点续传 | Golang中文社


Golang操作Mysql


golang的json操作


Golang Go语言断点续传


golang 读写文件


golang的package

go语言之bufio 打开文件，读取一行

键-文件存储系统weedfs

« (/articles/3153)上一篇: GO语言基础环境搭建以及HelloWorld (/articles/3153)
» (/articles/3155)下一篇: Golang开发环境搭建（Windows下） (/articles/3155)

文章点评：

（您需要 登录 后才能评论 没有账号（/user/register）？）

编辑 预览

我有话要说.....

- 支持 Markdown 格式, **粗体**、~~删除线~~、 `单行代码`
- 支持 @ 本站用户；支持表情（输入 : 提示），见 Emoji cheat sheet (http://www.emoji-cheat-sheet.com/)

提交

最新主题（/topics） | 最新资源（/resources） | 最新评论

golang 实现的一个类似 Heka 和 Logstash 的轻量级数据采集系统 (/topics/1966)

是否遇到过本站搜索博文出错？ (/topics/1965)

Fuchsia与GoLang有什么联系 (/topics/1964)

编译不成功 (/topics/1963)

go在windows下执行系统命令调用问题？求教 (/topics/1962)

将 TOML 转换为 Go 结构体 TOML-To-Go (/topics/1961)

使用Kubernetes创建 Couchbase 集群 (/topics/1960)

Go最新资料汇总（七十八） (/topics/1959)

哪里可以下载教程 (/topics/1958)

北京互联网公司【招聘高级Go语言开发】【北京朝阳区大望路】 (/topics/1957)

开源项目 (/projects)

(/p/Gosuv)
GO 编写的进程管理工具 Go

(/p/mgweb)
MongoDB Web 管理工具 mgw

(/p/excelize)
操作 Office Excel 文档类

(/p/pengye-go)
Go 实现的社区系统 朋也社

(/p/clevergo)
高性能Web框架 CleverGo (