```
//将日志固定输出到cServer.log文件中，
package main
import (
    "flag"
    "fmt"
    "log"
    "os"
    "runtime"
)

var (
    logFileName = flag.String("log", "cServer.log", "Log file name")
)

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    flag.Parse()
    //set logfile Stdout,log文件的权限位0666（即所有用户可读写），
    //    如果log文件不存在，创建一个新的文件os.O_CREATE
    //    将log信息写入到log文件，是继承到当前log文件，不是覆盖os.O_APPEND
    logFile, logErr := os.OpenFile(*logFileName,
os.O_CREATE|os.O_RDWR|os.O_APPEND, 0666)
    if logErr != nil {
        fmt.Println("Fail to find", *logFile, "cServer start Failed")
        os.Exit(1)
    }
    //    设置输出目的地
    log.SetOutput(logFile)
    //    通过log.SetFlags()自定议你想要表达的格式
    log.SetFlags(log.Ldate | log.Ltime | log.Lshortfile)

    //write log
    for i := 5; i > 0; i-- {
        log.Printf("Server abort! Cause:%v \n", "test log file")
```

```
    }
    log.Println("**************")
    for i := 10; i > 0; i-- {
        log.Fatal("Server abort! Cause:%v \n", "test log file 123")
    }
}
```

**按天分割nginx日志：**

```
package main

import (
    "fmt"
    "io/ioutil"
    "os"
    "os/exec"
    "path/filepath"
    "strings"
    "syscall"
    "time"
)

//install后将脚本加入到crontab定时运行，一天一次，就可以实现nginx日志按天分割
//当然golang也可以自己定时执行，这里加入到crontab运行，是因为golang进程有可能会被kill掉....

func main() {
    //日志目录
    srcDirPath := "/usr/local/nginx/logs"

    //存放切割日志目录
    targetDirPath := "/usr/local/nginx/logs/history"
```

```
//ngixn进程ID文件
nginxPidPath := "/usr/local/nginx/logs/nginx.pid"

//检查存放切割日志目录是否存在，如果不存在则创建
finfo, errFile := os.Stat(targetDirPath)
if errFile != nil {
    errFile := os.MkdirAll(targetDirPath, 0777)
    if errFile != nil {
        fmt.Println("创建日志目录失败：" + errFile.Error())
        return
    }
} else if !finfo.IsDir() {
    fmt.Println(targetDirPath + "已经存在且不是一个目录")
    return
}

//获取当前日期，作为此次切割日志根目录
t := time.Now()
nowDateTime := t.Format("2006-01-02")
logPath := targetDirPath + "/" + nowDateTime
os.MkdirAll(logPath, 0777)

//获取nginx的进程ID
pfile, err := os.Open(nginxPidPath)
defer pfile.Close()
if err != nil {
    fmt.Println("not found nginx pid file")
    return
}
pidData, _ := ioutil.ReadAll(pfile)
pid := string(pidData)
pid = strings.Replace(pid, "\n", "", -1)

//遍历日志目录
```

```
    filepath.Walk(srcDirPath, func(path string, info os.FileInfo, err error) error {
        if info.IsDir() {
            return nil
        } else {
            //获取切割日志路径
            targetfilePath := strings.Replace(path, srcDirPath, logPath, 1)
            if strings.Index(targetfilePath, "nginx.pid") != -1 {
                return nil
            }

            //移动文件
            syscall.Rename(path, targetfilePath)

            //创建原文件,这里不需要了，因为重启nginx后会自动生成一个空的日志文件
            // nFile,errCreate := os.Create(path)
            // if errCreate != nil {
            //     fmt.Println("create file faild:"+errCreate.Error())
            // }
            // defer nFile.Close()
        }
        return nil
    })

    //  平滑重启nginx
    //    USR1亦通常被用来告知应用程序重载配置文件；
    //    例如，向Apache HTTP服务器发送一个USR1信号将导致以下步骤的发生：停止接受新的连接，等待当前连接停止，重新载入配置文件，重新打开日志文件，重启服务器，从而实现相对平滑的不关机的更改。
    cmd := exec.Command("kill", "-USR1", pid)
    _, errCmd := cmd.Output()
    if errCmd != nil {
        fmt.Println("重启nginx失败：" + errCmd.Error())
        return
    }
    fmt.Println("success")
```

```
}
```