命令调用

exec包运行外部命令

- LookPath在PATH环境变量的目录中搜索命令名称
- Command执行命令返回结构体，通过结构体的Output()方法获取输出

```go
package main

import (
    "fmt"
    "os/exec"
)

func main() {
    s, err := exec.LookPath("vue") // 通过PATH找命令
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Printf("%s\n", s) // 返回绝对路径字符串
    // 执行命令，可以跟命令参数
    cmd := exec.Command(s, "-V")
    b, err := cmd.Output() // 获取输出
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    fmt.Println(string(b))
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~~")
}
```

日志

log包

Go标准库中有log包，提供了简单的日志功能。

输出	格式输出	换行输出	
log.Print()	log.Printf()	log.Println()	类似fmt.Print*
log.Fatal()	log.Fatalf()	log.Fatalln()	相当于log.Print* + os.Exit(1)
log.Panic()	log.Panicf()	log.Panicln	相当于log.Print* + panic()

日志输出需要使用日志记录器Logger。

log包提供了一个缺省的Logger即std。std是小写的，包外不可见，所以提供了Default()方法返回std给包外使用。

```
// 大约在源码log.go第90行
var std = New(os.Stderr, "", LstdFlags)

func Default() *Logger { return std }

const (
    Ldate         = 1 << iota // 1 当前时区日期: 2009/01/23
    Ltime                     // 2 当前时区时间: 01:23:23
    Lmicroseconds             // 4 微秒: 01:23:23.123123.  assumes Ltime.
    Llongfile                 // 8 绝对路径和行号: /a/b/c/d.go:23
    Lshortfile                // 16 文件名和行号: d.go:23. overrides Llongfile
    LUTC                      // 32 使用UTC（GMT），而不是本地时区
    Lmsgprefix                // 64 默认前缀放行首，这个标记把前缀prefix放到消息message之前
    LstdFlags     = Ldate | Ltime // 3 initial values for the standard logger
)
```

上表列出的方法底层都使用std.Output输出日志内容。而std本质上是使用了**标准错误输出**、无前缀、**LstdFlags**标准标记的记录器Logger实例。

std使用

```
// 使用缺省Logger
log.Print("abcde\n")
log.Printf("%s\n", "abcd")
log.Println("abc")

log.Fatal("xyz") // 等价于 log.Print("xyz");os.Exit(1)

log.Panicln("Failed") // 等价于 log.Println("Failed");panic()
```

自定义Logger

```
type Logger struct {
    mu        sync.Mutex // ensures atomic writes; protects the following fields
    prefix    string     // prefix on each line to identify the logger (but see Lmsgprefix)
    flag      int        // properties
    out       io.Writer  // destination for output
    buf       []byte     // for accumulating text to write
    isDiscard int32      // atomic boolean: whether out == io.Discard
}

func New(out io.Writer, prefix string, flag int) *Logger {
    l := &Logger{out: out, prefix: prefix, flag: flag}
    if out == io.Discard {
        l.isDiscard = 1
    }
    return l
```

```
}
```

如果觉得缺省Logger std不满意，可以New构建一个自定义Logger并指定前缀、Flags。

```
// 自定义Logger
infoLogger := log.New(os.Stdout, "Info: ", log.LstdFlags|log.Lmsgprefix)
infoLogger.Println("这是一个普通消息") // 使用stdout输出

errLogger := log.New(os.Stderr, "Error: ", log.LstdFlags)
errLogger.Fatal("这是一个错误消息")
```

写日志文件

New方法签名`New(out io.Writer, prefix string, flag int) *Logger`中out参数提供Writer接口即可，那么就可以提供一个可写文件对象。

```
package main

import (
    "fmt"
    "log"
    "os"
)

func main() {
    f, err := os.OpenFile(
        "o:/my.log",
        os.O_WRONLY|os.O_CREATE|os.O_APPEND, // 追加写，文件不存在创建
        os.ModePerm,
    )
    if err != nil {
        log.Panicln(err)
    }
    defer f.Close()

    l := log.New(f, "Info: ", log.LstdFlags)
    l.Println("这是一个写入文件的消息")
}
```

zerolog

log模块太简陋了，实际使用并不方便。

- logrus有日志级别、Hook机制、日志格式输出，很好用
- zap是Uber的开源高性能日志库
- zerolog更注重开发体验，高性能、有日志级别、链式API，json格式日志记录，号称0内存分配

官网 https://zerolog.io/

安装`go get -u github.com/rs/zerolog/log`

缺省Logger

```
import "github.com/rs/zerolog/log"

log.Print("hello") // 使用全局缺省logger
// {"level":"debug","time":"2008-10-14T09:17:50+08:00","message":"hello"}
JSON格式输出
// log.Print产生debug级别消息
```

```
// Logger is the global logger.源码第14行，定义了一个全局导出的缺省Logger
var Logger = zerolog.New(os.Stderr).With().Timestamp().Logger() // 链式调用
// 缺省Logger使用标准错误输出
```

log.Print()、log.Printf()方法使用方式和标准库log模块类似。

级别

zerolog提供以下级别（从高到底）

- panic (zerolog.PanicLevel, 5)
- fatal (zerolog.FatalLevel, 4)
- error (zerolog.ErrorLevel, 3)
- warn (zerolog.WarnLevel, 2)
- info (zerolog.InfoLevel, 1)
- debug (zerolog.DebugLevel, 0)
- trace (zerolog.TraceLevel, -1)

级别有

- gLevel全局级别
  - `zerolog.SetGlobalLevel(级别数字或常量)`来设置全局级别
  - `zerolog.GlobalLevel()`获取当前全局级别
- 每个Logger的级别
- 消息的级别

```
package main

import (
    "fmt"

    "github.com/rs/zerolog"
    "github.com/rs/zerolog/log"
)

func main() {
    fmt.Println("全局级别gLevel为", zerolog.GlobalLevel())
    fmt.Println("缺省logger的级别为", log.Logger.GetLevel())
    log1 := log.Level(zerolog.WarnLevel) // 创建一个子logger
    fmt.Println("log1级别为", log1.GetLevel())
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~")

```

```
    log.Trace().Msg("缺省logger输出trace级别消息") // 输出
    log.Info().Msg("缺省logger输出info级别消息")   // 输出
    log.Warn().Msg("缺省logger输出warn级别消息")   // 输出
    log.Error().Msg("缺省logger输出error级别消息") // 输出
    log1.Debug().Msg("log1的Debug级别消息")      // 不能输出
    log1.Warn().Msg("log1的Warn级别消息")        // 输出
    log1.Error().Msg("log1的Error级别消息")      // 输出
}
```

可以看到，使用缺省logger，全部可以输出日志消息，而log1使22行、23行输出了日志，为什么?

因为，有**消息级别**和**Logger级别**。

log1的级别为warn 2，而log1.Debug()输出的消息级别为debug级别0，消息级别 < log1级别，所以消息不能输出。log1.Warn()、log1.Error()产生warn、error消息消息，消息级别 ≥ log1级别，因此可以输出。

而缺省Logger的级别是trace，任何消息级别都大于等于log1的级别，因此都可以输出。

下面调高全局级别，试试看。

```
package main

import (
    "fmt"

    "github.com/rs/zerolog"
    "github.com/rs/zerolog/log"
)

func main() {
    zerolog.SetGlobalLevel(zerolog.ErrorLevel) // 调高全局级别
    fmt.Println("全局级别gLevel为", zerolog.GlobalLevel())
    fmt.Println("缺省logger的级别为", log.Logger.GetLevel())
    log1 := log.Level(zerolog.WarnLevel) // 创建一个子logger
    fmt.Println("log1级别为", log1.GetLevel())
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~")

    log.Trace().Msg("缺省logger输出trace级别消息") // 不能输出
    log.Info().Msg("缺省logger输出info级别消息")   // 不能输出
    log.Warn().Msg("缺省logger输出warn级别消息")   // 不能输出
    log.Error().Msg("缺省logger输出error级别消息") // 输出
    log1.Debug().Msg("log1的Debug级别消息")      // 不能输出
    log1.Warn().Msg("log1的Warn级别消息")        // 不能输出
    log1.Error().Msg("log1的Error级别消息")      // 输出
}
```

缺省logger和log1都只有error级别的输出，说明将gLevel调整到error级别后，所有logger输出消息都必须大于等于gLevel。

特别注意，zerolog.SetGlobalLevel()设置的是全局变量gLevel，它影响所有Logger。

日志消息是否能够输出，应当满足下面的要求 `消息级别 ≥ max(gLevel, 当前logger级别)`

```
zerolog.SetGlobalLevel(zerolog.Disabled)
// zerolog.Disabled 为7，没有消息级别可以大于等于7，相当于禁用所有Logger，自然不能输出日志了
```

上下文

zerolog是以json对象格式输出的，还可以自定义一些键值对字段增加到上下文中以输出。

```
zerolog.SetGlobalLevel(zerolog.InfoLevel)
log.Info().Bool("Success", false).Str("Rea  son", "File Not Found").Msg("文件没找到")
log.Info().Str("Name", "Tom").Floats32("Scores", []float32{87.5, 90, 59}).Send()
// Send is equivalent to calling Msg("")
```

错误日志

```
package main

import (
    "errors"

    "github.com/rs/zerolog"
    "github.com/rs/zerolog/log" // 全局logger
)

func main() {
    zerolog.TimeFieldFormat = zerolog.TimeFormatUnix // 自定义time字段时间的格式，TimeFormatUnix时间戳
    // zerolog.ErrorFieldName = "err" // 修改日志Json中的缺省字段名error
    // 错误日志
    err := errors.New("自定义的错误")
    log.Error(). // 错误级别消息
            Err(err). // err字段，错误消息内容
            Send()    // 有错误消息了，message可以省略

    log.Fatal(). // fatal级别
            Err(err).
            Send()
}
```

全局Logger

```
// 全局Logger定义如下
var Logger = zerolog.New(os.Stderr).With().Timestamp().Logger()
```

可以覆盖全局Logger

```
zerolog.TimeFieldFormat = zerolog.TimeFormatUnix

// With()创建一个全局Logger的子logger
log.Logger = log.With().Str("School", "Magedu").Logger() // 覆盖了全局Logger
log.Info().Send() // {"level":"info","School":"Magedu","time":1223947070}
```

自定义Logger

```
zerolog.TimeFieldFormat = zerolog.TimeFormatUnix

logger := log.With(). // With()返回基于全局Logger的子logger
            Str("School", "Magedu").
            Caller(). // 增加日志调用的位置信息字段
            Logger()  // 返回Logger

logger.Info().Send() // {"level":"info","School":"Magedu","time":1223947070}
log.Info().Send()    // {"level":"info","time":1223947070} 全局Logger
```

```
zerolog.TimeFieldFormat = zerolog.TimeFormatUnix

logger := zerolog.New(os.Stdout). // 不基于全局Logger，重新构造了一个Logger
                    With().Str("School", "Magedu").
                    Caller().                  // 调用者信息：增加日志函数调用的位置信息字段
                    Logger().                  // 返回Logger
                    Level(zerolog.ErrorLevel) // 重新定义Logger级别为3 error，返回Logger

fmt.Println(logger.GetLevel())
logger.Info().Send() // {"level":"info","School":"Magedu","time":1223947070} 看颜色区别
logger.Error().Send()

log.Info().Send() // {"level":"info","time":1223947070} 全局Logger
```

写日志文件

```
package main

import (
    "os"

    "github.com/rs/zerolog"
    "github.com/rs/zerolog/log" // 全局logger
)

func main() {
    zerolog.TimeFieldFormat = zerolog.TimeFormatUnix

    f, err := os.OpenFile("o:/my.log", os.O_CREATE|os.O_APPEND, os.ModePerm)
    if err != nil {
        log.Panic().Err(err).Send() // 内部调用panic
    }
```

```
    defer f.Close()

    multi := zerolog.MultiLevelWriter(f, os.Stdout) // 多分支写
    // Timestamp()为这个全新的Logger增加时间戳输出
    logger := zerolog.New(multi).With().Timestamp().Logger()
    logger.Info().Msg("日志兵分两路，去控制台stdout，还去日志文件")
}
```

如果只输出到文件可以使用`zerolog.New(f).With().Timestamp().Logger()`