

1、基本介绍

log包实现了简单的日志服务。本包定义了 `Logger` 类型，该类型提供了一些格式化输出的方法。本包也提供了一个预定义的“标准”Logger，可以通过辅助函数 `Print[f|ln]`、`Fatal[f|ln]` 和 `Panic[f|ln]` 访问，比手工创建一个Logger对象更容易使用。Logger会打印每条日志信息的日期、时间，默认输出到标准错误。Fatal系列函数会在写入日志信息后调用`os.Exit(1)`。Panic系列函数会在写入日志信息后panic。

2、基本日志打印函数

func Printf

```
func Printf(format string, v ...interface{})
```

Printf调用Output将生成的格式化字符串输出到标准logger，参数用和fmt.Printf相同的方法处理。

func Print

```
func Print(v ...interface{})
```

Print调用Output将生成的格式化字符串输出到标准logger，参数用和fmt.Print相同的方法处理。

func Println

```
func Println(v ...interface{})
```

Println调用Output将生成的格式化字符串输出到标准logger，参数用和fmt.Println相同的方法处理。

func Fprintf

```
func Fprintf(format string, v ...interface{})
```

Fprintf等价于{Printf(v...); os.Exit(1)}

func Fatal

```
func Fatal(v ...interface{})
```

Fatal等价于{Print(v...); os.Exit(1)}

func Fprintln

```
func Fprintln(v ...interface{})
```

Fprintln等价于{Println(v...); os.Exit(1)}

func Panicf

```
func Panicf(format string, v ...interface{})
```

Panicf等价于{Printf(v...); panic(...)}

func Panic

```
func Panic(v ...interface{})
```

Panic等价于{Print(v...); panic(...)}

func Panicln

```
func Panicln(v ...interface{})
```

Panicln等价于{Println(v...); panic(...)}

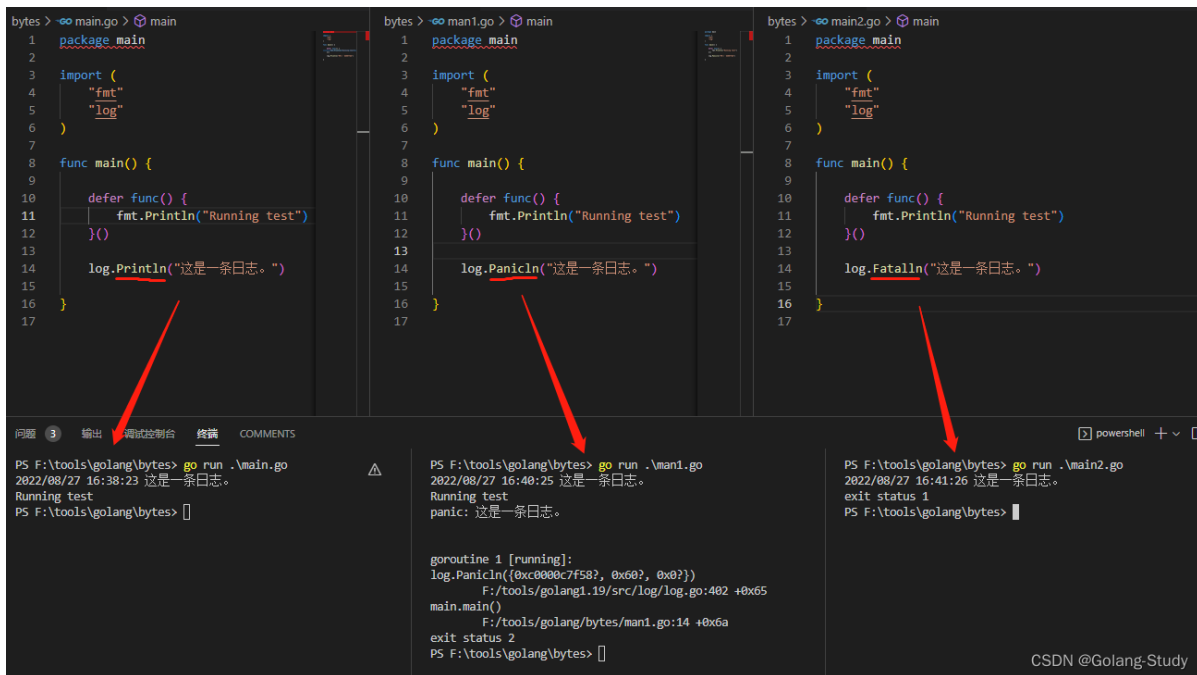
三种打印日志的方式，每种又有三种格式，分别是打印换行、格式化打印以及直接打印。

三种方式：

第一种Print方式将日志按照fmt.Print相同的方法处理。

第二种Fatal方式，先打印日志然后直接os.Exit(1)终止进程，如果有defer，他不会被执行。

第三种Panic方式，先打印日志，然后报恐慌，如果有defer，它会被执行。



3、flag选项

log标准库提供了如下的flag选项，它们是一系列定义好的 常量。这些选项定义Logger类型如何生成用于 每条日志的前缀文本。

```
const (  
    // 数位共同控制输出日志信息的细节。不能控制输出的顺序和格式。  
    // 在所有项目后会有一个冒号：2009/01/23 01:23:23.123123 /a/b/c/d.go:23: message  
    Ldate          = 1 << iota    // 日期：2009/01/23  
    Ltime          // 时间：01:23:23  
    Lmicroseconds  // 微秒分辨率：01:23:23.123123（用于增强Ltime位）  
    Llongfile      // 文件全路径名+行号： /a/b/c/d.go:23  
    Lshortfile     // 文件无路径名+行号： d.go:23（会覆盖掉Llongfile）  
    LstdFlags      = Ldate | Ltime // 标准logger的初始值  
)
```

获取和设置flag

func Flags

```
func Flags() int
```

Flags返回标准logger的输出选项。

func SetFlags

```
func SetFlags(flag int)
```

SetFlags设置标准logger的输出选项。

CSDN @Golang-Study

```
8 func main() {
9
10     defer func() {
11         fmt.Println("Running test")
12     }()
13
14     fmt.Println(log.Flags()) // 3(二进制 1 + 10 = 3)表示Ldate | Ltime
15
16     log.SetFlags(log.Ldate | log.Ltime | log.Lshortfile)
17
18     log.Println("这是一条日志。")
19
20 }
21
```

问题 3 输出 调试控制台 终端 COMMENTS

```
PS F:\tools\golang\bytes> go run .\main.go
3
2022/08/27 16:49:31 main.go:18: 这是一条日志。
Running test
PS F:\tools\golang\bytes> |
```

CSDN @Golang-Study

4、配置日志前缀

func Prefix

```
func Prefix() string
```

Prefix返回标准logger的输出前缀。

func SetPrefix

```
func SetPrefix(prefix string)
```

SetPrefix设置标准logger的输出前缀。

CSDN @Golang-Study

```

8 func main() {
9
10     defer func() {
11         fmt.Println("Running test")
12     }()
13
14     log.SetFlags(log.Llongfile | log.Lshortfile | log.Ldate)
15     log.Println("这是一条很普通的日志。")
16     log.SetPrefix("[XXXXXX]") // 设置前缀
17     log.Println("这是一条很普通的日志11。")
18     log.Println("这是一条很普通的日志12。")
19     log.Println("这是一条很普通的日志13。")
20
21 }
22

```

问题 3 输出 调试控制台 终端 COMMENTS

```

PS F:\tools\golang\bytes> go run .\main.go
2022/08/27 main.go:15: 这是一条很普通的日志。
[XXXXXX]2022/08/27 main.go:17: 这是一条很普通的日志11。
[XXXXXX]2022/08/27 main.go:18: 这是一条很普通的日志12。
[XXXXXX]2022/08/27 main.go:19: 这是一条很普通的日志13。
Running test
PS F:\tools\golang\bytes>

```

CSDN @Golang-Study

5、配置日志输出位置

日志输出默认是往标准输出的，即 `os.Stdout`

func SetOutput

```
func SetOutput(w io.Writer)
```

SetOutput设置标准logger的输出目的地，默认是标准错误输出。

CSDN @Golang-Study

The screenshot shows a Go IDE with a file named `main.go` open. The code defines a `main` function that opens a file `./xx.log` with flags `os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644`. It then calls `log.SetOutput(logFile)` to set the output destination to the file. The code also sets flags `log.Llongfile | log.Lshortfile | log.Ldate` and a prefix `[PREFIX]-->` . It prints two log messages. A red box highlights the `log.SetOutput(logFile)` line, and a red arrow points from it to the terminal output. The terminal shows the output of the program, with the log messages prefixed with `[PREFIX]-->` .

```

main.go 1 x
bytes > main.go > main
4     "fmt"
5     "log"
6     "os"
7 )
8
9 func main() {
10
11     logFile, err := os.OpenFile("./xx.log", os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644)
12     if err != nil {
13         fmt.Println("open log file failed, err:", err)
14         return
15     }
16
17     log.SetOutput(logFile) // 设置输出目的地
18
19     log.SetFlags(log.Llongfile | log.Lshortfile | log.Ldate)
20     log.SetPrefix("[PREFIX]--> ")
21     log.Println("这是一条很普通的日志1。")
22     log.Println("这是一条很普通的日志2。")
23
24     defer func() {
25         logFile.Close()
26     }()
27
28 }
29

```

```

bytes > xx.log
1 [PREFIX]--> 2022/08/27 main.go:21: 这是一条很普通的日志1。
2 [PREFIX]--> 2022/08/27 main.go:22: 这是一条很普通的日志2。
3

```

CSDN @Golang-Study

如果要使用标准的logger，我们通常会把上面的配置操作写到init函数中。

```
func init() {
    logFile, err := os.OpenFile("./xx.log", os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644)
    if err != nil {
        fmt.Println("open log file failed, err:", err)
        return
    }
    log.SetOutput(logFile)
    log.SetFlags(log.Llongfile | log.Lmicroseconds | log.Ldate)
}
```

CSDN @Golang-Study

6、Logger类

上面的使用都是基于预定义的标准Logger的，所以上面的操作都是直接调用包中的函数。

包中还有一个Logger类，这个类也提供了上面的函数，但是这里的函数是方法，需要Logger对象(地址类型)调用。

- type Logger
 - func New(out io.Writer, prefix string, flag int) *Logger
 - func (l *Logger) Flags() int
 - func (l *Logger) SetFlags(flag int)
 - func (l *Logger) Prefix() string
 - func (l *Logger) SetPrefix(prefix string)
 - func (l *Logger) Output(calldepth int, s string) error
 - func (l *Logger) Printf(format string, v ...interface{})
 - func (l *Logger) Print(v ...interface{})
 - func (l *Logger) Println(v ...interface{})
 - func (l *Logger) Fatalf(format string, v ...interface{})
 - func (l *Logger) Fatal(v ...interface{})
 - func (l *Logger) Fatalln(v ...interface{})
 - func (l *Logger) Panic(v ...interface{})
 - func (l *Logger) Panicf(format string, v ...interface{})
 - func (l *Logger) Panicln(v ...interface{})

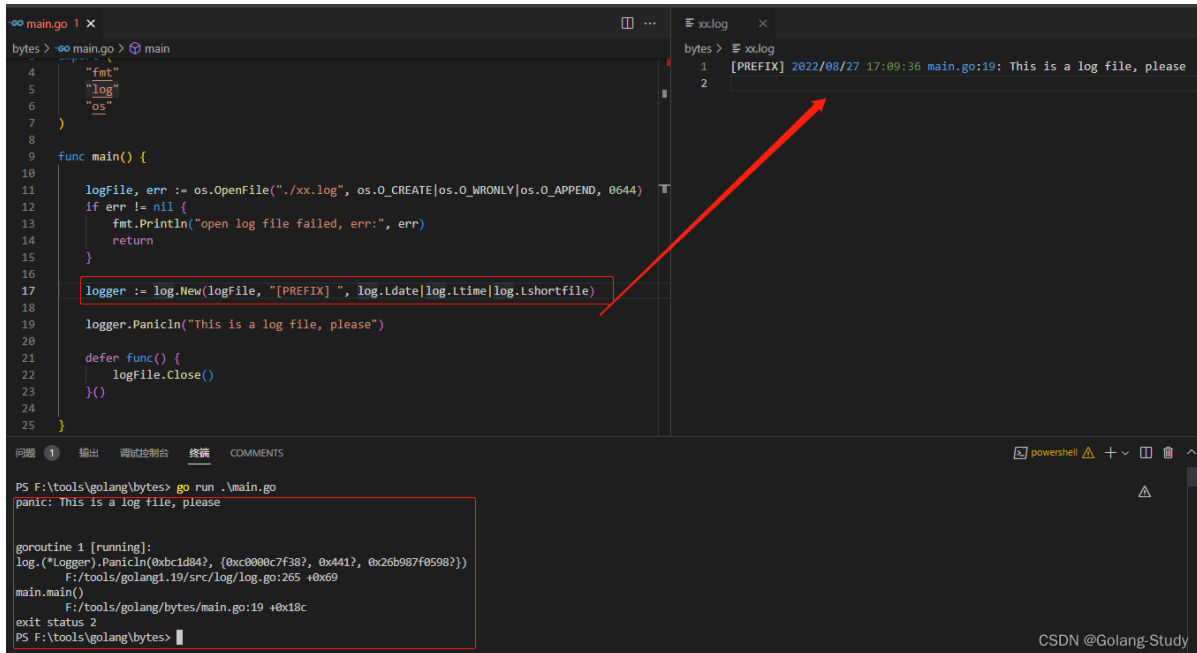
CSDN @Golang-Study

func New

```
func New(out io.Writer, prefix string, flag int) *Logger
```

New创建一个Logger。参数out设置日志信息写入的目的地。参数prefix会添加到生成的每一条日志前面。参数flag定义日志的属性（时间、文件等等）。

CSDN @Golang-Study



```
main.go 1 x
bytes > main.go > main
4   "fmt"
5   "log"
6   "os"
7
8
9
10
11 func main() {
12     logfile, err := os.OpenFile("./xx.log", os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644)
13     if err != nil {
14         fmt.Println("open log file failed, err:", err)
15         return
16     }
17     logger := log.New(logfile, "[PREFIX] ", log.Ldate|log.Ltime|log.Lshortfile)
18
19     logger.Panicln("This is a log file, please")
20
21     defer func() {
22         logfile.Close()
23     }()
24
25 }

xolog
bytes > xolog
1 [PREFIX] 2022/08/27 17:09:36 main.go:19: This is a log file, please
2

PS F:\tools\golang\bytes> go run .\main.go
panic: This is a log file, please

goroutine 1 [running]:
log.(*Logger).Panicln(0xc1d84?, {0xc000c7f38?, 0x441?, 0x26b987f0590?})
    F:/tools/golang1.19/src/log/log.go:265 +0x69
main.main()
    F:/tools/golang/bytes/main.go:19 +0x18c
exit status 2
PS F:\tools\golang\bytes>
```

CSDN @Golang-Study