[go标准库](https://www.cnblogs.com/itogo/p/8645499.html)

log包是go语言提供的一个简单的日志记录功能，其中定义了一个结构体类型 Logger，是整个包的基础部分，包中的其他方法都是围绕这整个结构体创建的．

**Logger结构**

Logger结构的定义如下:

type Logger struct {

mu sync.Mutex

prefix string

flag int

out io.Writer

buf []byte

}

* mu 是sync.Mutex，它是一个同步互斥锁，用于保证日志记录的原子性．
* prefix 是输入的日志每一行的前缀
* flag 是一个标志，用于设置日志的打印格式
* out 日志的输出目标，需要是一个实现了 io.Writer接口的对象，如: os.Stdout, os.Stderr, os.File等等
* buf 用于缓存数据

与此同时还提供了一个构造方法用于创建 Logger:

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

return &Logger{out: out, prefix: prefix, flag: flag}

}

还有围绕Logger结构的几个参数定义的方法:

func (l \*Logger) SetOutput(w io.Writer) // 用于设置日志输出目标

func (l \*Logger) SetPrefix(prefix string) // 用于设置每一行日志的前缀

func (l \*Logger) Prefix() string // 获取当前使用的前缀

func (l \*Logger) SetFlags(flag int) // 用于设置使用的输出标志

func (l \*Logger) Flags() int // 获取当前使用的标志

这些方法都很简单，只是给我们提供了一个可以修改和获取当前日志器的设置的方式．

**flag可选值**

在 log 包中，定义了一系列的常亮用于表示 flag,如下:

const (

Ldate = 1 << iota // 1 << 0 当地时区的日期: 2009/01/23

Ltime // 1 << 1 当地时区的时间: 01:23:23

Lmicroseconds // 1 << 2 显示精度到微秒: 01:23:23.123123 (应该和Ltime一起使用)

Llongfile // 1 << 3 显示完整文件路径和行号: /a/b/c/d.go:23

Lshortfile // 1 << 4 显示当前文件名和行号: d.go:23 (如果与Llongfile一起出现，此项优先)

LUTC // 1 << 5如果设置了Ldata或者Ltime, 最好使用 UTC 时间而不是当地时区

LstdFlags = Ldate | Ltime // 标准日志器的初始值

)

使用方法:

* 可以单独使用某一个标志，此时只会显示对应的信息
* 可以多个合并使用，只需要将多个标志使用 | 连接即可

例如:

Ldate | Ltime // 2017/07/31 08:01:20

Ldate | Ltime | Lmicroseconds | Llongfile // 2017/07/31 08:01:20.123123 /a/b/c/d.go:23

**常用方法**

在 log 包中，定义了下面几组方法:

func (l \*Logger) Printf(format string, v ...interface{})

func (l \*Logger) Print(v ...interface{})

func (l \*Logger) Println(v ...interface{})

func (l \*Logger) Fatal(v ...interface{})

func (l \*Logger) Fatalf(format string, 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{})

即 Print\*, Fatal\*, Painc\*, 这里方法结尾的 f 或者 ln 就跟 fmt.Print 的含义是相同的，因此上面这九个方法的使用方式其实与 fmt.Print/f/ln 是一样的．我们直接以没有 f 或 ln 的方法为例来看看三组方法的代码：

func (l \*Logger) Print(v ...interface{}) {

l.Output(2, fmt.Sprint(v...))

}

func (l \*Logger) Fatal(v ...interface{}) {

l.Output(2, fmt.Sprint(v...))

os.Exit(1)

}

func (l \*Logger) Panic(v ...interface{}) {

s := fmt.Sprint(v...)

l.Output(2, s)

panic(s)

}

可以看到其实三个方法 都调用了接收者(也就是Logger类型的实例或指针)的 Output 方法，这个方法后面在说，其实就是字面的意思，即用来输出我们传入进去的字符串(fmt.Sprint方法将我们传入的参数转换为字符串后返回)

不同的地方在于:

* Print 仅仅是输出了信息
* Fatal 不仅仅输出了信息，还使程序停止运行
* Painc 不仅仅输出了信息，还调用了 panic 抛出错误

所以这三个方法的用处就显而易见了．

**Output方法**

前面介绍了三组方法的内部都是调用了 Output 方法来实现的，也就是说实际的工作实在 Output 方法中执行的．

func (l \*Logger) Output(calldepth int, s string) error {

now := time.Now()

var file string

var line int

l.mu.Lock()

defer l.mu.Unlock()

if l.flag&(Lshortfile|Llongfile) != 0 {

l.mu.Unlock()

var ok bool

\_, file, line, ok = runtime.Caller(calldepth)

if !ok {

file = "???"

line = 0

}

l.mu.Lock()

}

l.buf = l.buf[:0]

l.formatHeader(&l.buf, now, file, line)

l.buf = append(l.buf, s...)

if len(s) == 0 || s[len(s)-1] != '\n' {

l.buf = append(l.buf, '\n')

}

\_, err := l.out.Write(l.buf)

return err

}

这里需要提前说一下 runtime.Caller 函数，这个函数用于获取调用Go程的栈上的函数调用所在的文件和行号信息。参数为 skip 表示我们需要获取信息的调用层级，返回值为　程序计数器(pc), 文件名，行号以及获取成功与否的标志。

在 Output 方法中，我们做了下面这些事情:

1. 获取当前事件
2. 对 Logger实例进行加锁操作
3. 判断Logger的标志位是否包含 Lshortfile 或 Llongfile,　如果包含进入步骤4, 如果不包含进入步骤5
4. 获取当前函数调用所在的文件和行号信息
5. 格式化数据，并将数据写入到 l.out 中，完成输出
6. 解锁操作

这里我们注意到有一个 callpath 参数，这个参数是用于获取某个指定层级的信息，前面3组方法中，这里使用的都是2, 这是因为，我们真正需要的文件名和行号是 **调用 Print, Fatal, Panic** 这些方法的地方，因此在调用 runtime.Caller 方法时，需要获取栈中当前位置的前两个位置处的信息．

**快捷方式**

log 包除了提供了上述一些需要先创建 Logger 实例才能使用的方法之外，还给我们定义了一些快捷的方法，它的实现方式也很简单，其实就是在 log包内预先定义了一个 Logger 实例叫 std:

var std = New(os.Stderr, "", LstdFlags)

然后定义了一些可以直接使用包来调用的方法:

func Output(calldepth int, s string) error

func Fatal(v ...interface{})

func Fatalf(format string, v ...interface{})

func Fatalln(v ...interface{})

func Panic(v ...interface{})

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

func Panicln(v ...interface{})

func Print(v ...interface{})

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

func Println(v ...interface{})

func SetFlags(flag int)

func Flags() int

func SetOutput(w io.Writer)

func SetPrefix(prefix string)

func Prefix() string

这些方法的内部实际上大部分都是直接调用了 std 的对应的方法来实现的，不过 Print\*, Panic\*, Fatal\* 这些方法的内部还是调用了 std.Output 方法来实现的．

前面已经涵盖了 log 包中的所有方法，除了下面两个:

* func itoa(buf \*[]byte, i int, wid int)
* func (l \*Logger) formatHeader(buf \*[]byte, t time.Time, file string, line int)

这里就不细说了，主要就是用来完成数据的格式化操作的．