Go语言 标准库 Log包

Go语言内置了`log`包，实现简单的日志服务。通过调用`log`包的函数，可以实现简单的日志打印功能。

使用Logger

Log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，可以通过调用函数Print系列(Print|Printf|Println）、Fatal系列（Fatal|Fatalf|Fatalln）、和Panic系列（Panic|Panicf|Panicln）来使用，比自行创建一个logger对象更容易使用。

函数系列	说明	作用
Print	Print/Printf/Println	单纯打印日志
Panic	Panic/Panicf/Panicln	打印日志，抛出panic异常
Fatal	Fatal/Fatalf/Fatalln	打印日志，强制结束程序(os.Exit(1))，defer函数不会执行

logger会打印每条日志信息的日期、时间，默认输出到系统的标准错误。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。

Print/Println/Printf函数

Print/Println/Printf函数只是单纯打印日志。

```
func printDemo() {
    log.Print("包子的日志")
    log.Printf("包子的第 %d 个日志", 100) // 格式化输出
    name := "包子"
    age := 18
    log.Println(name, " ", age)
}
#结果
2022/12/18 15:50:13 包子的日志
2022/12/18 15:50:13 包子的第 100 个日志
2022/12/18 15:50:13 包子   18
```

基本使用与`fmt`中的函数类似。

Panic/Panicf/Panicln函数

Panic/Panicf/Panicln函数会打印出日志并且抛出panic异常，需要注意的是在panic之后声明的代码将不会执行。

```
func panicDemo() {
    defer fmt.Println("发生了 panic错误！") // panic会执行defer
    log.Print("包子的日志")
    log.Panic("包子的panic日志") // panic会执行异常
    fmt.Println("运行结束。。。") // 这行不会显示
}
```

Fatal/Fatalf/Fatalln函数

对于`Fatal`接口，会将日志内容打印输出，接着调用系统的`os.Exit(1)`接口，强制退出程序并返回状态1，但是有一点需要注意的是，由于直接调用系统os接口退出，defer函数不会调用。

```
func fatalDemo() {
    defer fmt.Println("Fatal defer...") // defer 不会执行
    log.Print("包子的日志")
    log.Fatal("包子的fatal日志") // fatal执行之后终止
    fmt.Println("运行结束。。。")  // fatal之后不会被执行
}
```

标准日志配置

默认情况下log只会打印出时间，但是实际情况下我们还需要获取文件名，行号等信息，log包提供给我们定制的接口。

log包提供两个标准log配置的相关方法：

方法	说明
func Flags() int	返回标准log输出配置
func SetFlags(flag int)	设置标准log输出配置

flag参数

log标准库提供了如下的flag参数，它们是一系列定义好的常量。

注意：只能控制输出日志信息的细节，不能控制输出的顺序和格式，输出的日志在每一项后会有一个冒号分隔

```
const (
    // 控制输出日志信息的细节，不能控制输出的顺序和格式。
    // 输出的日志在每一项后会有一个冒号分隔，例如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                 // 微秒级别的时间，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                    // 文件名+行号，d.go:23（会覆盖掉Llongfile）
    LUTC                          // 使用UTC时间
    LstdFlags     = Ldate | Ltime // 标准logger的初始值
)
```

```
func flagDemo() {
    fmt.Println(log.Flags()) // 打印出3的意思是：Ldate | Ltime 做位运算得到 00000011
    log.SetFlags(log.Llongfile | log.Lmicroseconds | log.Ldate) // 二进制 00001101
  十进制 13  SetFlags 可以传入二进制或者十进制数字，但是不推荐，更推荐使用定义好的常量
    log.Println("这是一条带文件路径和日期时间的日志")
}
```

日志前缀配置

log包提供两个日志前缀配置的相关函数：

方法	说明
func Prefix() string	返回日志的前缀配置
func SetPrefix(prefix string)	设置日志前缀

```
func prefixDemo() {
    log.SetPrefix("包子：")
    fmt.Println(log.Prefix())
    log.SetFlags(log.Llongfile | log.Lmicroseconds | log.Ldate)
    log.Println("这是一条带前缀的日志")
}
```

我们就能够在代码中为我们的日志信息添加指定的前缀，方便之后对日志信息进行检索和处理。

日志输出位置配置

前面介绍的都是将日志输出到控制台上，golang的log包还支持将日志输出到文件中，log包提供了`func SetOutput(w io.Writer)`函数，将日志输出到文件中。

```
func SetOutput(w io.Writer) //设置标准logger的输出目的地，默认是标准错误输出
```

```
func fileDemo() {
    f, err := os.OpenFile("./log/a.log", os.O_CREATE|os.O_WRONLY|os.O_APPEND,
0644)
    if err != nil {
        log.Panic("打开日志文件异常")
    }
    log.SetOutput(f)
    log.Print("包子真帅...")
}
```

自定义logger

log包中提供了`func New(out io.Writer, prefix string, flag int) *Logger`函数来实现自定义logger。从效果上来看，就是标准日志配置、日志前缀配置、日志输出位置配置整合到一个函数中，使日志配置不在那么繁琐。

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

New创建一个Logger对象。其中，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）。

```
func newDemo() {
    logFile, err := os.OpenFile("./log/a.log",
os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644)
    if err != nil {
        log.Panic("打开日志文件异常")
    }
    logger := log.New(logFile, "[包子日志]", log.Ldate|log.Ltime|log.Lshortfile)
    logger.Println("日志演示案例")
}
```

总结

Go内置的log库功能有限，例如无法满足记录不同级别日志的情况，我们在实际的项目中根据自己的需要选择使用第三方的日志库，如logrus、zap等。

在开发中，通常是把日志配置到`init`函数中，使用时直接调用log就行，一个包内写一次`init`即可。