# Go语言 标准库 Fmt包

fmt包实现了格式化的I/O函数，主要分为向外输出内容和获取输入内容两大部分。

# fmt 向外输出

## Print

Print系列函数会将内容输出到系统的标准输出

- `Print`函数直接输出内容
- `Printf`函数支持格式化输出字符串
- `Println`函数会在输出内容的结尾添加一个换行符

```
func Print(a ...interface{}) (n int, err error)
func Printf(format string, a ...interface{}) (n int, err error)
func Println(a ...interface{}) (n int, err error)
```

示例：

```
package main

import "fmt"

func printDemo() {
    fmt.Print("包子是个大帅哥")
    name := "包子"
    fmt.Printf("我是：%s\n", name)
    fmt.Println("打印并换行")
}

func main() {
    printDemo()
}
#结果
包子是个大帅哥我是：包子
打印并换行
```

### 格式化输出占位符

`printf`系列函数都支持format格式化参数，在这里我们按照占位符将被替换的变量类型划分，方便查询和记忆。

#### 通用占位符

| 占位符 | 说明 |
| --- | --- |
| %v | 值的默认格式表示 |
| %+v | 类似%v，但输出结构体时会添加字段名 |
| %#v | 值的Go语法表示 |
| %T | 打印值的类型 |
| %% | 百分号 |

**布尔型**

| 占位符 | 说明 |
| --- | --- |
| %t | true或false |

**整型**

| 占位符 | 说明 |
| --- | --- |
| %b | 表示为二进制 |
| %c | 该值对应的unicode码值 |
| %d | 表示为十进制 |
| %o | 表示为八进制 |
| %x | 表示为十六进制，使用a-f |
| %X | 表示为十六进制，使用A-F |
| %U | 表示为Unicode格式：U+1234，等价于”U+%04X” |
| %q | 该值对应的单引号括起来的go语法字符字面值，必要时会采用安全的转义表示 |

**浮点数与复数**

| 占位符 | 说明 |
| --- | --- |
| %b | 无小数部分、二进制指数的科学计数法，如-123456p-78 |
| %e | 科学计数法，如-1234.456e+78 |
| %E | 科学计数法，如-1234.456E+78 |
| %f | 有小数部分但无指数部分，如123.456 |
| %F | 等价于%f |
| %g | 根据实际情况采用%e或%f格式（以获得更简洁、准确的输出） |
| %G | 根据实际情况采用%E或%F格式（以获得更简洁、准确的输出） |

**字符串和[]byte**

| 占位符 | 说明 |
| --- | --- |
| %s | 直接输出字符串或者[]byte |
| %q | 该值对应的双引号括起来的go语法字符串字面值，必要时会采用安全的转义表示 |
| %x | 每个字节用两字符十六进制数表示（使用a-f） |
| %X | 每个字节用两字符十六进制数表示（使用A-F） |

**指针**

| 占位符 | 说明 |
| --- | --- |
| %p | 表示为十六进制，并加上前导的0x |

**宽度标识符**

宽度通过一个紧跟在百分号后面的十进制数指定，如果未指定宽度，则表示值时除必需之外不作填充。精度通过（可选的）宽度后跟点号后跟的十进制数指定。如果未指定精度，会使用默认精度；如果点号后没有跟数字，表示精度为0

| 占位符 | 说明 |
| --- | --- |
| %f | 默认宽度，默认精度 |
| %9f | 宽度9，默认精度 |
| %.2f | 默认宽度，精度2 |
| %9.2f | 宽度9，精度2 |
| %9.f | 宽度9，精度0 |

**其他flag**

| 占位符 | 说明 |
| --- | --- |
| ’+’ | 总是输出数值的正负号；对%q（%+q）会生成全部是ASCII字符的输出（通过转义）； |
| ’ ’ | 对数值，正数前加空格而负数前加负号；对字符串采用%x或%X时（% x或% X）会给各打印的字节之间加空格 |
| ’-’ | 在输出右边填充空白而不是默认的左边（即从默认的右对齐切换为左对齐）； |
| ’#’ | 八进制数前加0（%#o），十六进制数前加0x（%#x）或0X（%#X），指针去掉前面的0x（%#p）对%q（%#q），对%U（%#U）会输出空格和单引号括起来的go字面值； |
| ‘0’ | 使用0而不是空格填充，对于数值类型会把填充的0放在正负号后面； |

## Fprint

Fprint系列函数会将内容输出到一个`io.Writer`接口类型的变量`w`中，我们通常用这个函数往文件中写入内容。

```
func Fprint(w io.Writer, a ...interface{}) (n int, err error)
func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
func Fprintln(w io.Writer, a ...interface{}) (n int, err error)
```

示例

```
func fprintDemo() {
    // 输出写入内容
    fmt.Fprintln(os.Stdout, "输出hello baozi")

    fileObj, err := os.OpenFile("./fprint_test.txt",
os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644)
    if err != nil {
        fmt.Println("打开文件出错，err:", err)
        return
    }
    name := "包子是个大帅哥！"
    // 向打开的文件句柄中写入内容
    fmt.Fprintf(fileObj, "往文件中写如信息：%s", name)
}
```

注意，只要满足`io.Writer`接口的类型都支持写入

## Sprint

Sprint系列函数会把传入的数据生成并返回一个字符串。

```
func Sprint(a ...interface{}) string
func Sprintf(format string, a ...interface{}) string
func Sprintln(a ...interface{}) string
```

示例

```
func sprintDemo() {
    s1 := fmt.Sprint("包子咋这么帅呢！")
    name := "包子"
    age := 18
    s2 := fmt.Sprintf("name:%s,age:%d", name, age)
    s3 := fmt.Sprintln("Hello包子")
    fmt.Println(s1, s2, s3)
}
#结果
包子咋这么帅呢！ name:包子,age:18 Hello包子
```

## Errorf

Errorf函数根据format参数生成格式化字符串并返回一个包含该字符串的错误。

```
func Errorf(format string, a ...interface{}) error
```

通常使用这种方式来自定义错误类型，例如：

```
err := fmt.Errorf("这是一个错误")
```

# fmt 获取输入

Go语言fmt包下有fmt.Scan、fmt.Scanf、fmt.Scanln三个函数，可以在程序运行过程中从标准输入获取用户的输入。

## fmt.Scan

函数定签名如下：

```
func Scan(a ...interface{}) (n int, err error)
```

- Scan从标准输入扫描文本，读取由空白符分隔的值保存到传递给本函数的参数中，换行符视为空白符。
- 本函数返回成功扫描的数据个数和遇到的任何错误。如果读取的数据个数比提供的参数少，会返回一个错误报告原因。

```
func scanDemo() {
    var (
        name  string
        age   int
        email string
    )
    fmt.Scan(&name, &age, &email)
    fmt.Printf("扫描结果 name:%s age:%d email:%s \n", name, age, email)
}
```

将上面的代码编译后在终端执行，在终端按照指定的格式依次输入 包子、18和baozi@baozi.com 使用空格分隔。

```
go run "d:\GoPro\fmt\fmt.go"
baozi 18 baozi@baozi.com
扫描结果 name:baozi age:18 email:baozi@baozi.com
```

fmt.Scan从标准输入中扫描用户输入的数据，将以空白符分隔的数据分别存入指定的参数。

## fmt.Scanf

函数签名如下：

```
func Scanf(format string, a ...interface{}) (n int, err error)
```

- Scanf从标准输入扫描文本，根据format参数指定的格式去读取由空白符分隔的值保存到传递给本函数的参数中。
- 本函数返回成功扫描的数据个数和遇到的任何错误。

```
func scanfDemo() {
    var (
        name  string
        age   int
        about string
    )
    fmt.Scanf("1:%s 2:%d 3:%s", &name, &age, &about)
    fmt.Printf("扫描结果 name:%s age:%d about:%s \n", name, age, about)
}
```

将上面的代码编译后在终端执行，在终端按照指定的格式依次输入1:包子 2:18 3:包子好帅

```
go run "d:\GoPro\fmt\fmt.go"
1:包子 2:18 3:包子好帅
扫描结果 name:包子 age:18 about:包子好帅
```

fmt.Scanf不同于fmt.Scan简单的以空格作为输入数据的分隔符，fmt.Scanf为输入数据指定了具体的输入内容格式，只有按照格式输入数据才会被扫描并存入对应变量。

例如，我们还是按照上个示例中以空格分隔的方式输入，fmt.Scanf就不能正确扫描到输入的数据。

## fmt.Scanln

函数签名如下：

```
func Scanln(a ...interface{}) (n int, err error)
```

- Scanln类似Scan，它在遇到换行时才停止扫描。最后一个数据后面必须有换行或者到达结束位置。
- 本函数返回成功扫描的数据个数和遇到的任何错误。

```
func scanflnDemo() {
    var (
        name  string
        age   int
        about string
    )
    fmt.Scanln(&name, &age, &about)
    fmt.Printf("扫描结果 name:%s age:%d about:%s \n", name, age, about)
}
```

上面的代码编译后在终端执行，在终端依次输入包子 18 包子是个大好人,使用空格分隔。

```
go run "d:\GoPro\fmt\fmt.go"
包子 18 包子是个大好人
扫描结果 name:包子 age:18 about:包子是个大好人
```

fmt.Scanln遇到回车就结束扫描了，这个比较常用。

## bufio.NewReader

有时候我们想完整获取输入的内容，而输入的内容可能包含空格，这种情况下可以使用bufio包来实现。

```
func bufioNewReaderDemo() {
    reader := bufio.NewReader(os.Stdin) // 从标准输入生成读对象
    fmt.Print("请输入内容：")
    text, _ := reader.ReadString('\n') // 读到换行
    text = strings.TrimSpace(text)
    fmt.Printf("%#v\n", text)
}
```

上面的代码编译后在终端执行，在终端依次输入哎呀 包子实在太帅了。

```
go run "d:\GoPro\fmt\fmt.go"
请输入内容：哎呀 包子实在太帅了
"哎呀 包子实在太帅了"
```

## Fscan系列

这几个函数功能分别类似于fmt.Scan、fmt.Scanf、fmt.Scanln三个函数，只不过它们不是从标准输入中读取数据而是从io.Reader中读取数据。

```
func Fscan(r io.Reader, a ...interface{}) (n int, err error)
func Fscanln(r io.Reader, a ...interface{}) (n int, err error)
func Fscanf(r io.Reader, format string, a ...interface{}) (n int, err error)
```

## Sscan系列

这几个函数功能分别类似于fmt.Scan、fmt.Scanf、fmt.Scanln三个函数，只不过它们不是从标准输入中读取数据而是从指定字符串中读取数据。

```
func Sscan(str string, a ...interface{}) (n int, err error)
func Sscanln(str string, a ...interface{}) (n int, err error)
func Sscanf(str string, format string, a ...interface{}) (n int, err error)
```

示例

```
func sscanDemo() {
    var name string = "包子"
    var newName string = ""
    fmt.Sscan(name, &newName) // 相当于把name的值赋值给newName
    fmt.Println(name)
    fmt.Println(newName)
}
#结果
包子
包子
```