Go语言 标准库 path& path/filepath包

path 实现了对斜杠分隔的路径的实用操作函数，`path/filePath`包实现了兼容各操作系统的文件路径的实用操作函数。path 包中提供的函数，`path/filePath`包都有提供，功能类似，但实现不同。一般应该总是使用`path/filePath`包，而不是 path 包。

path 包

常用方法

path 实现了对斜杠分隔的路径的实用操作函数，常用方法如下：

IsAbs

判断路径是否是一个绝对路径（按 Linux 中路径判断，以 / 开头为根路径）

```
func IsAbs(path string) bool
```

Split

将路径从最后一个斜杠后面位置分隔为两个部分（dir 和 file）并返回，如果路径中没有斜杠，函数返回值 dir 会设为空字符串，file 会设为 path。两个返回值满足 path == dir+file

```
func Split(path string) (dir, file string)
```

Join

Join函数可以将任意数量的路径元素放入一个单一路径里，会根据需要添加斜杠。结果是经过简化的，所有的空字符串元素会被忽略。

```
func Join(elem ...string) string
```

Dir

返回所在的目录。

返回路径除去最后一个路径元素的部分，即该路径最后一个元素所在的目录，内部使用 Split 函数去掉最后一个元素后，会简化路径并去掉末尾的斜杠，如果路径是空字符串，会返回"."；如果路径由 1 到多个斜杠后跟 0 到多个非斜杠字符组成，会返回"/"；其他任何情况下都不会返回以斜杠结尾的路径

```
func Dir(path string) string
```

Base

Base函数返回路径的最后一个元素。

在提取元素前会求掉末尾的斜杠。如果路径是""，会返回"."；如果路径是只有一个斜杆构成，会返回"/"

```
func Base(path string) string
```

Ext

Ext函数返回path文件扩展名。

返回值是路径最后一个斜杠分隔出的路径元素的最后一个'.'起始的后缀（包括'.'）。如果该元素没有'.'会返回空字符串

```
func Ext(path string) string
```

Clean

通过单纯的词法操作返回和 path 代表同一地址的最短路径 。

它会不断的依次应用如下的规则，直到不能再进行任何处理：

1. 将连续的多个斜杠替换为单个斜杠
2. 剔除每一个 . 路径名元素（代表当前目录）
3. 剔除每一个路径内的 .. 路径名元素（代表父目录）和它前面的非 .. 路径名元素
4. 剔除开始一个根路径的 .. 路径名元素，即将路径开始处的"/.."替换为"/"

只有路径代表根地址"/"时才会以斜杠结尾。如果处理的结果是空字符串，Clean 会返回"."

```
func Clean(path string) string
```

Match

匹配路径

```
// 如果 name 匹配 shell 文件名模式匹配字符串，Match 函数返回真。该模式匹配字符串语法为：
// pattern:
//    { term }
// term:
//    '*'                                   匹配 0 或多个非 / 的字符
//    '?'                                   匹配 1 个非 / 的字符
//    '[' [ '^' ] { character-range } ']'   字符组（必须非空）
//    c                                     匹配字符 c（c != '*', '?', '\\', '['）
//    '\\' c                                匹配字符 c
// character-range:
//    c           匹配字符 c（c != '\\', '-', ']'）
//    '\\' c      匹配字符 c
//    lo '-' hi   匹配区间 [lo, hi] 内的字符
// Match 要求匹配整个 name 字符串，而不是它的一部分。只有 pattern 语法错误时，会返回
ErrBadPattern
```

```
func Match(pattern, name string) (matched bool, err error)
```

基本使用

```
package main

import (
    "fmt"
    "path"
```

```
)

func main() {
    fmt.Println(path.IsAbs("C:/aa/bb/test.txt"))         // false，以 Linux 文件路径判断
    fmt.Println(path.Split("C:/aa/bb/test.txt"))         // C:/aa/bb/ test.txt
    fmt.Println(path.Join("C:", "aa", "bb", "test.txt")) // C:/aa/bb/test.txt
    fmt.Println(path.Dir("C:/aa/bb/test.txt"))           // C:/aa/bb
    fmt.Println(path.Base("C:/aa/bb/test.txt"))          // test.txt
    fmt.Println(path.Ext("C:/aa/bb/test.txt"))           // .txt
    fmt.Println(path.Clean("C:/aa/../aa//bb/test.txt"))  // C:/aa/bb/test.txt
    fmt.Println(path.Match("C:/aa/*/", "C:/aa/bb/"))     // true <nil>
}
```

path/filepath 包

常用方法

`path/filePath` 包实现了兼容各操作系统的文件路径的实用操作函数，常用方法如下：

IsAbs

判断路径是否是一个绝对路径

```
func IsAbs(path string) bool
```

Abs

返回 path 代表的绝对路径，如果 path 不是绝对路径，会加入当前工作目录以使之成为绝对路径，因为硬链接的存在，不能保证返回的绝对路径是唯一指向该地址的绝对路径

```
func Abs(path string) (string, error)
```

Rel

返回 targpath 相对于 basepath 的相对路径，如果两个参数一个是相对路径而另一个是绝对路径，或者 targpath 无法表示为相对于 basepath 的路径，将返回错误。

```
func Rel(basepath, targpath string) (string, error)
```

SplitList

将 PATH 或 GOPATH 等环境变量里的多个路径分割开（这些路径被 OS 特定的表分隔符连接起来），与 strings.Split 函数的不同之处是：对 ""，SplitList 返回 []string{}，而 strings.Split 返回 []string{""}。

```
func SplitList(path string) []string
```

Split

将路径从最后一个斜杠后面位置分隔为两个部分（dir 和 file）并返回，如果路径中没有斜杠，函数返回值 dir 会设为空字符串，file 会设为 path。两个返回值满足 path == dir+file

```
func Split(path string) (dir, file string)
```

Join

将任意数量的路径元素放入一个单一路径里，会根据需要添加斜杠。结果是经过简化的，所有的空字符串元素会被忽略

```
func Join(elem ...string) string
```

FromSlash

将 path 中的斜杠（'/'）替换为路径分隔符并返回替换结果，多个斜杠会替换为多个路径分隔符

```
func FromSlash(path string) string
```

ToSlash

将 path 中的路径分隔符替换为斜杠（'/'）并返回替换结果，多个路径分隔符会替换为多个斜杠

```
func ToSlash(path string) string
```

VolumeName

返回最前面的卷名。如 Windows 系统里提供参数"C:\foo\bar" 会返回"C:"；Unix/linux 系统的"\host\share\foo" 会返回"\host\share"；其他平台会返回""

```
func VolumeName(path string) (v string)
```

Dir

返回所在的目录。

返回路径除去最后一个路径元素的部分，即该路径最后一个元素所在的目录，内部使用 Split 函数去掉最后一个元素后，会简化路径并去掉末尾的斜杠，如果路径是空字符串，会返回"."；如果路径由 1 到多个斜杠后跟 0 到多个非斜杠字符组成，会返回"/"；其他任何情况下都不会返回以斜杠结尾的路径

```
func Dir(path string) string
```

Base

Base函数返回路径的最后一个元素。

在提取元素前会求掉末尾的斜杠。如果路径是""，会返回"."；如果路径是只有一个斜杆构成，会返回"/"

```
func Base(path string) string
```

Ext

Ext函数返回path文件扩展名。

返回值是路径最后一个斜杠分隔出的路径元素的最后一个'.'起始的后缀（包括'.'）。如果该元素没有'.'会返回空字符串

```
func Ext(path string) string
```

Clean

通过单纯的词法操作返回和 path 代表同一地址的最短路径 。

它会不断的依次应用如下的规则，直到不能再进行任何处理：

1. 将连续的多个斜杠替换为单个斜杠
2. 剔除每一个 . 路径名元素（代表当前目录）
3. 剔除每一个路径内的 .. 路径名元素（代表父目录）和它前面的非 .. 路径名元素
4. 剔除开始一个根路径的 .. 路径名元素，即将路径开始处的"/.."替换为"/"

只有路径代表根地址"/"时才会以斜杠结尾。如果处理的结果是空字符串，Clean 会返回"."

```
func Clean(path string) string
```

EvalSymlinks

返回 path 指向的符号链接（软链接）所包含的路径。如果 path 和返回值都是相对路径，会相对于当前目录；除非两个路径其中一个是绝对路径

```
func EvalSymlinks(path string) (string, error)
```

Match

匹配路径

```
// 如果 name 匹配 shell 文件名模式匹配字符串，Match 函数返回真。该模式匹配字符串语法为：
// pattern:
//    { term }
// term:
//    '*'                                   匹配 0 或多个非 / 的字符
//    '?'                                   匹配 1 个非 / 的字符
//    '[' [ '^' ] { character-range } ']'   字符组（必须非空）
//    c                                     匹配字符 c（c != '*', '?', '\\', '['）
//    '\\' c                                匹配字符 c
// character-range:
//    c           匹配字符 c（c != '\\', '-', ']'）
//    '\\' c      匹配字符 c
//    lo '-' hi   匹配区间 [lo, hi] 内的字符
// Match 要求匹配整个 name 字符串，而不是它的一部分。只有 pattern 语法错误时，会返回
ErrBadPattern
```

```
func Match(pattern, name string) (matched bool, err error)
```

Glob

返回所有匹配模式匹配字符串 pattern 的文件或者 nil（如果没有匹配的文件），pattern 的语法和 Match 函数相同。pattern 可以描述多层的名字，如 /usr/*/bin/ed（假设路径分隔符是'/'）

```
func Glob(pattern string) (matches []string, err error)
```

Walk

Walk 函数对每一个文件/目录都会调用 WalkFunc 函数类型值。调用时 path 参数会包含 Walk 的 root 参数作为前缀；就是说，如果 Walk 函数的 root 为"dir"，该目录下有文件"a"，将会使用"dir/a"调用 walkFn 参 数。walkFn 参数被调用时的 info 参数是 path 指定的地址（文件/目录）的文件信息，类型为 os.FileInfo。如果遍历 path 指定的文件或目录时出现了问题，传入的参数 err 会描述该问题，WalkFunc 类型函数可以决定如何去处理该错误（Walk 函数将不会深入该目录）；如果该函数返回一个错误，Walk 函数的执行会中止；只有一个例外，如果 Walk 的 walkFn 返回值是 SkipDir，将会跳过该目录的内容而 Walk 函数照常执行处理下一个文件。

```
type WalkFunc func(path string, info os.FileInfo, err error) error
```

遍历 root 指定的目录下的文件树，对每一个该文件树中的目录和文件都会调用 walkFn，包括 root 自身。所有访问文件 / 目录时遇到的错误都会传递给 walkFn 过滤。文件是按词法顺序遍历的，这让输出更漂亮，但也导致处理非常大的目录时效率会降低。Walk函数不会遍历文件树中的符号链接（快捷方式）文件包含的路径。

```
func Walk(root string, walkFn WalkFunc) error
```

基本使用

```
package main

import (
    "fmt"
    "os"
    "path/filepath"
)

func main() {
    // filepath 包的函数会兼容各操作系统，所以相同函数在各个操作系统下返回可能不同
    fmt.Println(string(filepath.Separator))     // \
    fmt.Println(string(filepath.ListSeparator)) // ;

    fmt.Println(filepath.IsAbs("C:/aa/bb/test.txt"))                // true
    fmt.Println(filepath.Abs("./main.go"))                          //
F:\Go\src\testProject\main.go <nil>
    fmt.Println(filepath.Rel("C:/aa", "C:/aa/bb/test.txt"))         //
bb\test.txt <nil>
    fmt.Println(filepath.SplitList("C:/go/bin;D:/Java/jdk;E:/bin")) // [C:/go/bin
D:/Java/jdk E:/bin]
    fmt.Println(filepath.Split("C:/aa/bb/test.txt"))                // C:/aa/bb/
test.txt
    fmt.Println(filepath.Join("C:", "\\aa", "/bb", "\\test.txt"))   //
C:\aa\bb\test.txt
```

```
    // 将路径中的 \\ 更换为 /
    fmt.Println(filepath.FromSlash("C:\\aa//bb//cc\\test.txt")) //
C:\aa\\bb\\cc\test.txt
    // 将路径中的 / 替换为 \\
    fmt.Println(filepath.ToSlash("C:\\aa//bb//cc\\test.txt")) //
C:/aa//bb//cc/test.txt
    fmt.Println(filepath.VolumeName("C:/aa/bb/test.txt"))     // C:
    fmt.Println(filepath.Dir("C:/aa/bb/test.txt"))            // C:\aa\bb
    fmt.Println(filepath.Base("C:/aa/bb/test.txt"))           // test.txt
    fmt.Println(filepath.Ext("C:/aa/bb/test.txt"))            // .txt
    fmt.Println(filepath.Clean("C:/aa/../aa//bb/test.txt"))   //
C:/aa/bb/test.txt
    fmt.Println(filepath.Match("C:/aa/*/", "C:/aa/bb/"))      // true <nil>
    // 获取 C:/aa/bb 目录下所有 .txt 文件
    fmt.Println(filepath.Glob("C:/aa/bb/*.txt")) // [C:\aa\bb\test.txt
C:\aa\bb\test1.txt C:\aa\bb\test2.txt] <nil>

    // 遍历 C:/aa/bb 目录下所有文件
    filepath.Walk("C:/aa/bb", func(path string, info os.FileInfo, err error)
error {
        fmt.Println("Path:", path, "IsDir:", info.IsDir(), "size:", info.Size())
        return nil
    })
    // 变量结果：
    // Path: C:/aa/bb IsDir: true size: 0
    // Path: C:\aa\bb\test.txt IsDir: false size: 5
    // Path: C:\aa\bb\test1.txt IsDir: false size: 6
    // Path: C:\aa\bb\test2.txt IsDir: false size: 69
}
```