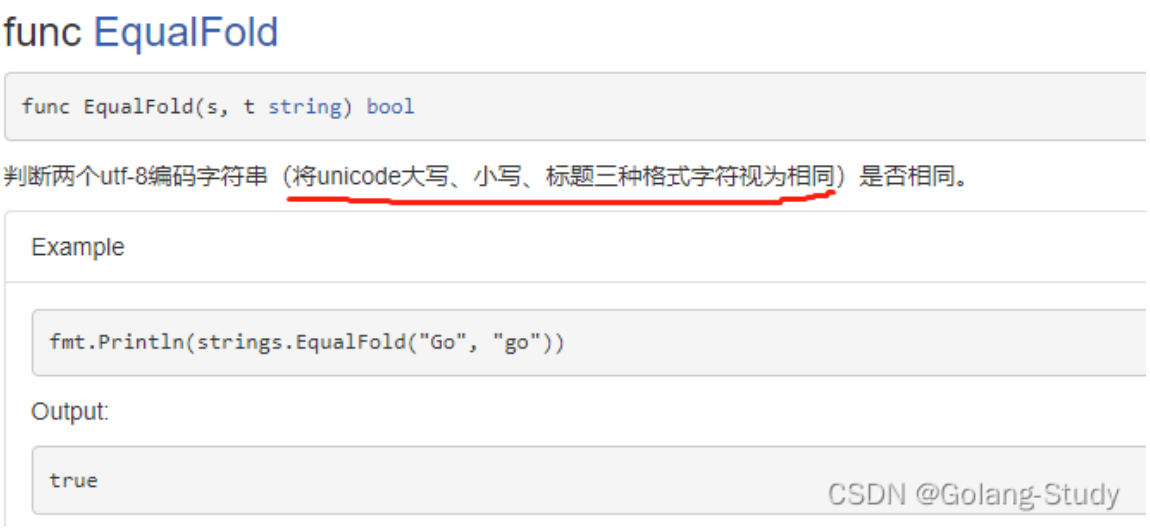
1、基本介绍

strings包实现了用于`操作字符的简单函数`。

2、EqualFold函数



func EqualFold

```
func EqualFold(s, t string) bool
```

判断两个utf-8编码字符串（将unicode大写、小写、标题三种格式字符视为相同）是否相同。

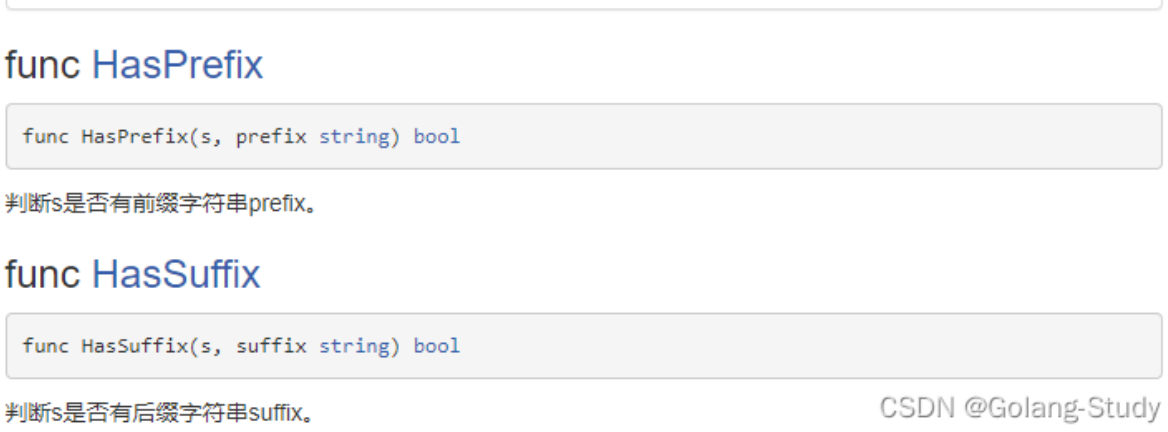
Example

```
fmt.Println(strings.EqualFold("Go", "go"))
```

Output:

```
true
```


3、HasPrefix和HasSuffix(前后缀)



func HasPrefix

```
func HasPrefix(s, prefix string) bool
```

判断s是否有前缀字符串prefix。

func HasSuffix

```
func HasSuffix(s, suffix string) bool
```

判断s是否有后缀字符串suffix。


```go
ret := strings.HasPrefix("index.html", "ind")
fmt.Println(ret)  // true
```

4、Contains系列

func Contains

```
func Contains(s, substr string) bool
```

判断字符串s是否包含子串substr。

Example

```
fmt.Println(strings.Contains("seafood", "foo"))
fmt.Println(strings.Contains("seafood", "bar"))
fmt.Println(strings.Contains("seafood", ""))
fmt.Println(strings.Contains("", ""))
```

Output:

```
true
false
true
true
```

func ContainsRune

```
func ContainsRune(s string, r rune) bool
```

判断字符串s是否包含utf-8码值r。

```
ret := strings.ContainsRune("ni好啊你", '你')
fmt.Println(ret) // true
```

func ContainsAny

```
func ContainsAny(s, chars string) bool
```

判断字符串s是否包含字符串chars中的任一字符。

Example

```
fmt.Println(strings.ContainsAny("team", "i"))
fmt.Println(strings.ContainsAny("failure", "u & i"))
fmt.Println(strings.ContainsAny("foo", ""))
fmt.Println(strings.ContainsAny("", ""))
```

Output:

```
false
true
false
false
```

5、Count(子串数)

func Count

```
func Count(s, sep string) int
```

返回字符串s中有几个不重复的sep子串。

Example

```
fmt.Println(strings.Count("cheese", "e"))
fmt.Println(strings.Count("five", "")) // before & after each rune
```

Output:

```
3
5
```

6、Index系列

func Index

```
func Index(s, sep string) int
```

子串sep在字符串s中第一次出现的位置，不存在则返回-1。

Example

```
fmt.Println(strings.Index("chicken", "ken"))
fmt.Println(strings.Index("chicken", "dmr"))
```

Output:

```
4
-1
```


func IndexByte

```
func IndexByte(s string, c byte) int
```

字符c在s中第一次出现的位置，不存在则返回-1。

func IndexRune

```
func IndexRune(s string, r rune) int
```

unicode码值r在s中第一次出现的位置，不存在则返回-1。

Example

```
fmt.Println(strings.IndexRune("chicken", 'k'))
fmt.Println(strings.IndexRune("chicken", 'd'))
```

Output:

```
4
-1
```



```
func testStrings() {

    // 一个判断英文一个可以判断中文
    index := strings.IndexByte("abcd", 'c')
    fmt.Println(index) // 2

    index = strings.IndexRune("abcd你好", '你')
    fmt.Println(index) // 4
}
```

func IndexAny

```
func IndexAny(s, chars string) int
```

字符串chars中的任一utf-8码值在s中第一次出现的位置，如果不存在或者chars为空字符串则返回-1。

Example

```
fmt.Println(strings.IndexAny("chicken", "aeiouy"))
fmt.Println(strings.IndexAny("crwth", "aeiouy"))
```

Output:

```
2
-1
```


7、LastIndex系列

func LastIndex

```
func LastIndex(s, sep string) int
```

子串sep在字符串s中最后一次出现的位置，不存在则返回-1。

Example

```
fmt.Println(strings.Index("go gopher", "go"))
fmt.Println(strings.LastIndex("go gopher", "go"))
fmt.Println(strings.LastIndex("go gopher", "rodent"))
```

Output:

```
0
3
-1
```

func LastIndexAny

```
func LastIndexAny(s, chars string) int
```

字符串chars中的任一utf-8码值在s中最后一次出现的位置，如不存在或者chars为空字符串则返回-1。

8、Title函数

func Title

```
func Title(s string) string
```

返回s中每个单词的首字母都改为标题格式的字符串拷贝。

BUG: Title用于划分单词的规则不能很好的处理Unicode标点符号。

```
// Her Royal Highness, Java,Python
fmt.Println(strings.Title("her royal highness, java,python"))
```

9、To系列

func ToLower

```
func ToLower(s string) string
```

返回将所有字母都转为对应的小写版本的拷贝。

Example

```
fmt.Println(strings.ToLower("Gopher"))
```

Output:

```
gopher
```


func ToUpper

```
func ToUpper(s string) string
```

返回将所有字母都转为对应的大写版本的拷贝。

Example

```
fmt.Println(strings.ToUpper("Gopher"))
```

Output:

```
GOPHER
```


func ToTitle

```
func ToTitle(s string) string
```

返回将所有字母都转为对应的标题版本的拷贝。

```
fmt.Println(strings.ToTitle("6loud noises你好"))
fmt.Println(strings.ToTitle("хлеб345"))
```

```
6LOUD NOISES你好
ХЛЕБ345
```

10、Repeat和Replace函数

func Repeat

```
func Repeat(s string, count int) string
```

返回count个s串联的字符串。

Example

```
fmt.Println("ba" + strings.Repeat("na", 2))
```

Output:

```
banana
```

func Replace

```
func Replace(s, old, new string, n int) string
```

返回将s中前n个不重叠old子串都替换为new的新字符串，如果n<0会替换所有old子串。

Example

```
fmt.Println(strings.Replace("oink oink oink", "k", "ky", 2))
fmt.Println(strings.Replace("oink oink oink", "oink", "moo", -1))
```

Output:

```
oinky oinky oink
moo moo moo
```


11、Join函数

func Join

```
func Join(a []string, sep string) string
```

将一系列字符串连接为一个字符串，之间用sep来分隔。

Example

```
s := []string{"foo", "bar", "baz"}
fmt.Println(strings.Join(s, ", "))
```

Output:

```
foo, bar, baz
```

12、Trim系列

func Trim

```
func Trim(s string, cutset string) string
```

返回将s前后端所有cutset包含的utf-8码值都去掉的字符串。

Example

```
fmt.Printf("[%q]", strings.Trim(" !!! Achtung! Achtung! !!! ", "! "))
```

Output:

```
["Achtung! Achtung"]
```

如果cutset是"!"，就会没有效果，因为字符串是" "开头和结尾的，会断掉

func TrimSpace

```
func TrimSpace(s string) string
```

返回将s前后端所有空白（unicode.IsSpace指定）都去掉的字符串。

Example

```
fmt.Println(strings.TrimSpace(" \t\n a lone gopher \n\t\r\n"))
```

Output:

```
a lone gopher
```


func TrimLeft

```
func TrimLeft(s string, cutset string) string
```

返回将s前端所有cutset包含的utf-8码值都去掉的字符串。

func TrimRight

```
func TrimRight(s string, cutset string) string
```

返回将s后端所有cutset包含的utf-8码值都去掉的字符串。

```
fmt.Println(strings.TrimLeft("abcdef", "ac"))  // bcdef
fmt.Println(strings.TrimRight("abcdef", "fd")) // abcde
```

func TrimPrefix

```
func TrimPrefix(s, prefix string) string
```

返回去除s可能的前缀prefix的字符串。

Example

```
var s = "Goodbye,, world!"
s = strings.TrimPrefix(s, "Goodbye,")
s = strings.TrimPrefix(s, "Howdy,")
fmt.Print("Hello" + s)
```

Output:

```
Hello, world!
```

func TrimSuffix

```
func TrimSuffix(s, suffix string) string
```

返回去除s可能的后缀suffix的字符串。

Example

```
var s = "Hello, goodbye, etc!"
s = strings.TrimSuffix(s, "goodbye, etc!")
s = strings.TrimSuffix(s, "planet")
fmt.Print(s, "world!")
```

Output:

```
Hello, world!
```


13、Fields函数

func Fields

```
func Fields(s string) []string
```

返回将字符串按照空白（unicode.IsSpace确定，可以是一到多个连续的空白字符）分割的多个字符串。如果字符串全部是空白或者是空字符串的话，会返回空切片。

Example

```
fmt.Printf("Fields are: %q", strings.Fields("  foo bar  baz   "))
```

Output:

```
Fields are: ["foo" "bar" "baz"]
```

14、Split系列

func Split

```
func Split(s, sep string) []string
```

用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（每一个sep都会进行一次切割，即使两个sep相邻，也会进行两次切割）。如果sep为空字符，Split会将s切分成每一个unicode码值一个字符串。

Example

```
fmt.Printf("%q\n", strings.Split("a,b,c", ","))
fmt.Printf("%q\n", strings.Split("a man a plan a canal panama", "a "))
fmt.Printf("%q\n", strings.Split(" xyz ", ""))
fmt.Printf("%q\n", strings.Split("", "Bernardo O'Higgins"))
```

Output:

```
["a" "b" "c"]
["" "man " "plan " "canal panama"]
[" " "x" "y" "z" " "]
[""]
```


func SplitN

```
func SplitN(s, sep string, n int) []string
```

用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（每一个sep都会进行一次切割，即使两个sep相邻，也会进行两次切割）。如果sep为空字符，Split会将s切分成每一个unicode码值一个字符串。参数n决定返回的切片的数目：

```
n > 0 : 返回的切片最多n个子字符串；最后一个子字符串包含未进行切割的部分。
n == 0: 返回nil
n < 0 : 返回所有的子字符串组成的切片
```

Example

```
fmt.Printf("%q\n", strings.SplitN("a,b,c", ",", 2))
z := strings.SplitN("a,b,c", ",", 0)
fmt.Printf("%q (nil = %v)\n", z, z == nil)
```

Output:

```
["a" "b,c"]
[] (nil = true)
```


15、Reader类型

使用`NewReader`函数从字符串获取一个`Reader`读取的流对象，然后就可以根据这个对象进行相关方法的调用。

type Reader

```
type Reader struct {
    // 内含隐藏或非导出字段
}
```

Reader类型通过从一个字符串读取数据，实现了io.Reader、io.Seeker、io.ReaderAt、io.WriterTo、io.ByteScanner、io.RuneScanner接口。

func NewReader

```
func NewReader(s string) *Reader
```

NewReader创建一个从s读取数据的Reader。本函数类似bytes.NewBufferString，但是更有效率，且为只读的。

func (*Reader) Len

```
func (r *Reader) Len() int
```

Len返回r包含的字符串还没有被读取的部分。

func (*Reader) Read

```
func (r *Reader) Read(b []byte) (n int, err error)
```

func (*Reader) ReadByte

```
func (r *Reader) ReadByte() (b byte, err error)
```

func (*Reader) UnreadByte

```
func (r *Reader) UnreadByte() error
```

func (*Reader) ReadRune

```
func (r *Reader) ReadRune() (ch rune, size int, err error)
```

func (*Reader) UnreadRune

```
func (r *Reader) UnreadRune() error
```

func (*Reader) Seek

```
func (r *Reader) Seek(offset int64, whence int) (int64, error)
```

Seek实现了io.Seeker接口。

func (*Reader) ReadAt

```
func (r *Reader) ReadAt(b []byte, off int64) (n int, err error)
```

func (*Reader) WriteTo

```
func (r *Reader) WriteTo(w io.Writer) (n int64, err error)
```

WriteTo实现了io.WriterTo接口。