strings

Package strings go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

Overview

Package strings implements simple functions to manipulate UTF-8 encoded strings.

For information about UTF-8 strings in Go, see https://blog.golang.org/strings.

func Compare

```
func Compare(a, b string) int
```

Compare returns an integer comparing two strings lexicographically. The result will be 0 if a==b, -1 if a < b, and +1 if a > b.

Compare is included only for symmetry with package bytes. It is usually clearer and always faster to use the built-in string comparison operators ==, <, >, and so on.

func Contains

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

Contains reports whether substr is within s.

func ContainsAny

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

ContainsAny reports whether any Unicode code points in chars are within s.

func ContainsRune

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

ContainsRune reports whether the Unicode code point r is within s.

func Count

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

Count counts the number of non-overlapping instances of substr in s. If substr is an empty string, Count returns 1 + the number of Unicode code points in s.

func EqualFold

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

EqualFold reports whether s and t, interpreted as UTF-8 strings, are equal under Unicode case-folding, which is a more general form of case-insensitivity.

func Fields

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

Fields splits the string s around each instance of one or more consecutive white space characters, as defined by unicode.IsSpace, returning a slice of substrings of s or an empty slice if s contains only white space.

func FieldsFunc

```
func FieldsFunc(s string, f func(rune) bool) []string
```

FieldsFunc splits the string s at each run of Unicode code points c satisfying f(c) and returns an array of slices of s. If all code points in s satisfy f(c) or the string is empty, an empty slice is returned.

FieldsFunc makes no guarantees about the order in which it calls f(c) and assumes that f always returns the same value for a given c.

func HasPrefix

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

HasPrefix tests whether the string s begins with prefix.

func HasSuffix

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

HasSuffix tests whether the string s ends with suffix.

func Index

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

Index returns the index of the first instance of substr in s, or -1 if substr is not present in s.

func IndexAny

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

IndexAny returns the index of the first instance of any Unicode code point from chars in s, or -1 if no Unicode code point from chars is present in s.

func IndexByte

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

IndexByte returns the index of the first instance of c in s, or -1 if c is not present in s.

func IndexFunc

```
func IndexFunc(s string, f func(rune) bool) int
```

IndexFunc returns the index into s of the first Unicode code point satisfying f(c), or -1 if none do.

func IndexRune

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

IndexRune returns the index of the first instance of the Unicode code point r, or -1 if rune is not present in s. If r is utf8.RuneError, it returns the first instance of any invalid UTF-8 byte sequence.

func Join

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

Join concatenates the elements of its first argument to create a single string. The separator string sep is placed between elements in the resulting string.

func LastIndex

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

LastIndex returns the index of the last instance of substr in s, or -1 if substr is not present in s.

func LastIndexAny

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

LastIndexAny returns the index of the last instance of any Unicode code point from chars in s, or -1 if no Unicode code point from chars is present in s.

func LastIndexByte

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

LastIndexByte returns the index of the last instance of c in s, or -1 if c is not present in s.

func LastIndexFunc

```
func LastIndexFunc(s string, f func(rune) bool) int
```

LastIndexFunc returns the index into s of the last Unicode code point satisfying f(c), or -1 if none do.

func Map

```
func Map(mapping func(rune) rune, s string) string
```

Map returns a copy of the string s with all its characters modified according to the mapping function. If mapping returns a negative value, the character is dropped from the string with no replacement.

func Repeat

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

Repeat returns a new string consisting of count copies of the string s.

It panics if count is negative or if the result of (len(s) * count) overflows.

func Replace

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

Replace returns a copy of the string s with the first n non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the string and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune string. If n < 0, there is no limit on the number of replacements.

func ReplaceAll

```
func ReplaceAll(s, old, new string) string
```

ReplaceAll returns a copy of the string s with all non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the string and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune string.

func Split

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

Split slices s into all substrings separated by sep and returns a slice of the substrings between those separators.

If s does not contain sep and sep is not empty, Split returns a slice of length 1 whose only element is s.

If sep is empty, Split splits after each UTF-8 sequence. If both s and sep are empty, Split returns an empty slice.

It is equivalent to SplitN with a count of -1.

func SplitAfter

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

SplitAfter slices s into all substrings after each instance of sep and returns a slice of those substrings.

If s does not contain sep and sep is not empty, SplitAfter returns a slice of length 1 whose only element is s.

If sep is empty, SplitAfter splits after each UTF-8 sequence. If both s and sep are empty, SplitAfter returns an empty slice.

It is equivalent to SplitAfterN with a count of -1.

func SplitAfterN

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

SplitAfterN slices s into substrings after each instance of sep and returns a slice of those substrings.

The count determines the number of substrings to return:

```
n > 0: at most n substrings; the last substring will be the unsplit remainder.
n == 0: the result is nil (zero substrings)
n < 0: all substrings
```

Edge cases for s and sep (for example, empty strings) are handled as described in the documentation for SplitAfter.

func SplitN

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

SplitN slices s into substrings separated by sep and returns a slice of the substrings between those separators.

The count determines the number of substrings to return:

```
n > 0: at most n substrings; the last substring will be the unsplit remainder.
n == 0: the result is nil (zero substrings)
n < 0: all substrings
```

Edge cases for s and sep (for example, empty strings) are handled as described in the documentation for Split.

func Title

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

Title returns a copy of the string s with all Unicode letters that begin words mapped to their Unicode title case.

BUG(rsc): The rule Title uses for word boundaries does not handle Unicode punctuation properly.

func ToLower

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

ToLower returns s with all Unicode letters mapped to their lower case.

func ToLowerSpecial

```
func ToLowerSpecial(c unicode.SpecialCase, s string) string
```

ToLowerSpecial returns a copy of the string s with all Unicode letters mapped to their lower case using the case mapping specified by c.

func ToTitle

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

ToTitle returns a copy of the string s with all Unicode letters mapped to their Unicode title case.

func ToTitleSpecial

```
func ToTitleSpecial(c unicode.SpecialCase, s string) string
```

ToTitleSpecial returns a copy of the string s with all Unicode letters mapped to their Unicode title case, giving priority to the special casing rules.

func ToUpper

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

ToUpper returns s with all Unicode letters mapped to their upper case.

func ToUpperSpecial

```
func ToUpperSpecial(c unicode.SpecialCase, s string) string
```

ToUpperSpecial returns a copy of the string s with all Unicode letters mapped to their upper case using the case mapping specified by c.

func ToValidUTF8

```
func ToValidUTF8(s, replacement string) string
```

ToValidUTF8 returns a copy of the string s with each run of invalid UTF-8 byte sequences replaced by the replacement string, which may be empty.

func Trim

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

Trim returns a slice of the string s with all leading and trailing Unicode code points contained in cutset removed.

func TrimFunc

```
func TrimFunc(s string, f func(rune) bool) string
```

TrimFunc returns a slice of the string s with all leading and trailing Unicode code points c satisfying f(c) removed.

func TrimLeft

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

TrimLeft returns a slice of the string s with all leading Unicode code points contained in cutset removed.

To remove a prefix, use TrimPrefix instead.

func TrimLeftFunc

```
func TrimLeftFunc(s string, f func(rune) bool) string
```

TrimLeftFunc returns a slice of the string s with all leading Unicode code points c satisfying f(c) removed.

func TrimPrefix

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

TrimPrefix returns s without the provided leading prefix string. If s doesn't start with prefix, s is returned unchanged.

func TrimRight

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

TrimRight returns a slice of the string s, with all trailing Unicode code points contained in cutset removed.

To remove a suffix, use TrimSuffix instead.

func TrimRightFunc

```
func TrimRightFunc(s string, f func(rune) bool) string
```

TrimRightFunc returns a slice of the string s with all trailing Unicode code points c satisfying f(c) removed.

func TrimSpace

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

TrimSpace returns a slice of the string s, with all leading and trailing white space removed, as defined by Unicode.

func TrimSuffix

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

TrimSuffix returns s without the provided trailing suffix string. If s doesn't end with suffix, s is returned unchanged.

type Builder

```
type Builder struct {
    // contains filtered or unexported fields
}
```

A Builder is used to efficiently build a string using Write methods. It minimizes memory copying. The zero value is ready to use. Do not copy a non-zero Builder.

func (*Builder) Cap

```
func (b *Builder) Cap() int
```

Cap returns the capacity of the builder's underlying byte slice. It is the total space allocated for the string being built and includes any bytes already written.

func (*Builder) Grow

```
func (b *Builder) Grow(n int)
```

Grow grows b's capacity, if necessary, to guarantee space for another n bytes. After Grow(n), at least n bytes can be written to b without another allocation. If n is negative, Grow panics.

func (*Builder) Len

```
func (b *Builder) Len() int
```

Len returns the number of accumulated bytes; b.Len() == len(b.String()).

func (*Builder) Reset

```
func (b *Builder) Reset()
```

Reset resets the Builder to be empty.

func (*Builder) String

```
func (b *Builder) String() string
```

String returns the accumulated string.

func (*Builder) Write

```
func (b *Builder) Write(p []byte) (int, error)
```

Write appends the contents of p to b's buffer. Write always returns len(p), nil.

func (*Builder) WriteByte

```
func (b *Builder) WriteByte(c byte) error
```

WriteByte appends the byte c to b's buffer. The returned error is always nil.

func (*Builder) WriteRune

```
func (b *Builder) WriteRune(r rune) (int, error)
```

WriteRune appends the UTF-8 encoding of Unicode code point r to b's buffer. It returns the length of r and a nil error.

func (*Builder) WriteString

```
func (b *Builder) WriteString(s string) (int, error)
```

WriteString appends the contents of s to b's buffer. It returns the length of s and a nil error.

type Reader

```
type Reader struct {
    // contains filtered or unexported fields
}
```

A Reader implements the io.Reader, io.ReaderAt, io.Seeker, io.WriterTo, io.ByteScanner, and io.RuneScanner interfaces by reading from a string. The zero value for Reader operates like a Reader of an empty string.

func NewReader

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

NewReader returns a new Reader reading from s. It is similar to bytes.NewBufferString but more efficient and read-only.

func (*Reader) Len

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

Len returns the number of bytes of the unread portion of the string.

func (*Reader) Read

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

func (*Reader) ReadAt

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

func (*Reader) ReadByte

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

func (*Reader) ReadRune

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

func (*Reader) Reset

```
func (r *Reader) Reset(s string)
```

Reset resets the Reader to be reading from s.

func (*Reader) Seek

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

Seek implements the io.Seeker interface.

func (*Reader) Size

```
func (r *Reader) Size() int64
```

Size returns the original length of the underlying string. Size is the number of bytes available for reading via ReadAt. The returned value is always the same and is not affected by calls to any other method.

func (*Reader) UnreadByte

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

func (*Reader) UnreadRune

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

func (*Reader) WriteTo

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

WriteTo implements the io.WriterTo interface.

type Replacer

```
type Replacer struct {
    // contains filtered or unexported fields
}
```

Replacer replaces a list of strings with replacements. It is safe for concurrent use by multiple goroutines.

func NewReplacer

```
func NewReplacer(oldnew ...string) *Replacer
```

NewReplacer returns a new Replacer from a list of old, new string pairs. Replacements are performed in the order they appear in the target string, without overlapping matches. The old string comparisons are done in argument order.

NewReplacer panics if given an odd number of arguments.

func (*Replacer) Replace

```
func (r *Replacer) Replace(s string) string
```

Replace returns a copy of s with all replacements performed.

func (*Replacer) WriteString

```
func (r *Replacer) WriteString(w io.Writer, s string) (n int, err error)
```

WriteString writes s to w with all replacements performed.

BUGs

- The rule Title uses for word boundaries does not handle Unicode punctuation properly.