list

package standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 0 |
Imported by: 24,093

Details ✓ Valid go.mod file ? ✓ Redistributable license ? ✓ Tagged version ?
✓ Stable version ?
Learn more

Repository cs.opensource.google/go/go

Links Report a Vulnerability

Documentation

Documentation

Overview

Package list implements a doubly linked list.

To iterate over a list (where l is a *List):

```
for e := l.Front(); e != nil; e = e.Next() {
	// do something with e.Value
}
```

▶ Example

Index

Examples

Constants

This section is empty.

Variables

This section is empty.

Functions

This section is empty.

Types

type Element

```
type Element struct {

	// The value stored with this element.
	Value any
	// contains filtered or unexported fields
}
```

Element is an element of a linked list.

func (*Element) Next

```
func (e *Element) Next() *Element
```

Next returns the next list element or nil.

func (*Element) Prev

```
func (e *Element) Prev() *Element
```

Prev returns the previous list element or nil.

type List

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

List represents a doubly linked list. The zero value for List is an empty list ready to use.

func New

```
func New() *List
```

New returns an initialized list.

func (*List) Back

```
func (l *List) Back() *Element
```

Back returns the last element of list l or nil if the list is empty.

func (*List) Front

```
func (l *List) Front() *Element
```

Front returns the first element of list l or nil if the list is empty.

func (*List) Init

```
func (l *List) Init() *List
```

Init initializes or clears list l.

func (*List) InsertAfter

```
func (l *List) InsertAfter(v any, mark *Element) *Element
```

InsertAfter inserts a new element e with value v immediately after mark and returns e. If mark is not an element of l, the list is not modified. The mark must not be nil.

func (*List) InsertBefore

```
func (l *List) InsertBefore(v any, mark *Element) *Element
```

InsertBefore inserts a new element e with value v immediately before mark and returns e. If mark is not an element of l, the list is not modified. The mark must not be nil.

func (*List) Len

```
func (l *List) Len() int
```

Len returns the number of elements of list l. The complexity is O(1).

func (*List) MoveAfter

added in go1.2

```
func (l *List) MoveAfter(e, mark *Element)
```

MoveAfter moves element e to its new position after mark. If e or mark is not an element of l, or e == mark, the list is not modified. The element and mark must not be nil.

func (*List) MoveBefore

added in go1.2

```
func (l *List) MoveBefore(e, mark *Element)
```

MoveBefore moves element e to its new position before mark. If e or mark is not an element of l, or e == mark, the list is not modified. The element and mark must not be nil.

func (*List) MoveToBack

```
func (l *List) MoveToBack(e *Element)
```

MoveToBack moves element e to the back of list l. If e is not an element of l, the list is not modified. The element must not be nil.

func (*List) MoveToFront

```
func (l *List) MoveToFront(e *Element)
```

MoveToFront moves element e to the front of list l. If e is not an element of l, the list is not modified. The element must not be nil.

func (*List) PushBack

```
func (l *List) PushBack(v any) *Element
```

PushBack inserts a new element e with value v at the back of list l and returns e.

func (*List) PushBackList

```
func (l *List) PushBackList(other *List)
```

PushBackList inserts a copy of another list at the back of list l. The lists l and other may be the same. They must not be nil.

func (*List) PushFront

```
func (l *List) PushFront(v any) *Element
```

PushFront inserts a new element e with value v at the front of list l and returns e.

func (*List) PushFrontList

```
func (l *List) PushFrontList(other *List)
```

PushFrontList inserts a copy of another list at the front of list l. The lists l and other may be the same. They must not be nil.

func (*List) Remove

```
func (l *List) Remove(e *Element) any
```

Remove removes e from l if e is an element of list l. It returns the element value e.Value. The element must not be nil.

Source Files

View all

list.go