parse

package standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 7 | Imported by: 696

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 parse builds parse trees for templates as defined by text/template and html/template. Clients should use those packages to construct templates rather than this one, which provides shared internal data structures not intended for general use.

Index

func IsEmptyTree(n Node) bool
func Parse(name, text, leftDelim, rightDelim string, funcs ...map[string]any) (map[string]*Tree, error)
type ActionNode
    func (a *ActionNode) Copy() Node
    func (a *ActionNode) String() string
type BoolNode
    func (b *BoolNode) Copy() Node
    func (b *BoolNode) String() string
type BranchNode
    func (b *BranchNode) Copy() Node
    func (b *BranchNode) String() string
type BreakNode
    func (b *BreakNode) Copy() Node
    func (b *BreakNode) String() string
type ChainNode
    func (c *ChainNode) Add(field string)
    func (c *ChainNode) Copy() Node

func (c *ChainNode) String() string
type CommandNode
func (c *CommandNode) Copy() Node
func (c *CommandNode) String() string
type CommentNode
func (c *CommentNode) Copy() Node
func (c *CommentNode) String() string
type ContinueNode
func (c *ContinueNode) Copy() Node
func (c *ContinueNode) String() string
type DotNode
func (d *DotNode) Copy() Node
func (d *DotNode) String() string
func (d *DotNode) Type() NodeType
type FieldNode
func (f *FieldNode) Copy() Node
func (f *FieldNode) String() string
type IdentifierNode
func NewIdentifier(ident string) *IdentifierNode
func (i *IdentifierNode) Copy() Node
func (i *IdentifierNode) SetPos(pos Pos) *IdentifierNode
func (i *IdentifierNode) SetTree(t *Tree) *IdentifierNode
func (i *IdentifierNode) String() string
type IfNode
func (i *IfNode) Copy() Node
type ListNode
func (l *ListNode) Copy() Node
func (l *ListNode) CopyList() *ListNode
func (l *ListNode) String() string
type Mode
type NilNode
func (n *NilNode) Copy() Node
func (n *NilNode) String() string
func (n *NilNode) Type() NodeType
type Node
type NodeType
func (t NodeType) Type() NodeType
type NumberNode
func (n *NumberNode) Copy() Node
func (n *NumberNode) String() string
type PipeNode
func (p *PipeNode) Copy() Node
func (p *PipeNode) CopyPipe() *PipeNode
func (p *PipeNode) String() string
type Pos
func (p Pos) Position() Pos

type RangeNode
func (r *RangeNode) Copy() Node
type StringNode
func (s *StringNode) Copy() Node
func (s *StringNode) String() string
type TemplateNode
func (t *TemplateNode) Copy() Node
func (t *TemplateNode) String() string
type TextNode
func (t *TextNode) Copy() Node
func (t *TextNode) String() string
type Tree
func New(name string, funcs ...map[string]any) *Tree
func (t *Tree) Copy() *Tree
func (t *Tree) ErrorContext(n Node) (location, context string)
func (t *Tree) Parse(text, leftDelim, rightDelim string, treeSet map[string]*Tree, ...) (tree *Tree, err error)
type VariableNode
func (v *VariableNode) Copy() Node
func (v *VariableNode) String() string
type WithNode
func (w *WithNode) Copy() Node

Constants

This section is empty.

Variables

This section is empty.

Functions

func IsEmptyTree

```
func IsEmptyTree(n Node) bool
```

IsEmptyTree reports whether this tree (node) is empty of everything but space or comments.

func Parse

```
func Parse(name, text, leftDelim, rightDelim string, funcs ...map[string]any) (map[string]*Tree, error)
```

Parse returns a map from template name to parse.Tree, created by parsing the templates described in the argument string. The top-level template will be given the specified name. If an error is encountered, parsing stops and an empty map is returned with the error.

Types

type ActionNode

```
type ActionNode struct {
    NodeType
    Pos

    Line int       // The line number in the input. Deprecated: Kept for compatibility.
    Pipe *PipeNode // The pipeline in the action.
    // contains filtered or unexported fields
}
```

ActionNode holds an action (something bounded by delimiters). Control actions have their own nodes; ActionNode represents simple ones such as field evaluations and parenthesized pipelines.

func (*ActionNode) Copy

```
func (a *ActionNode) Copy() Node
```

func (*ActionNode) String

```
func (a *ActionNode) String() string
```

type BoolNode

```
type BoolNode struct {
    NodeType
    Pos

    True bool // The value of the boolean constant.
    // contains filtered or unexported fields
}
```

BoolNode holds a boolean constant.

func (*BoolNode) Copy

```
func (b *BoolNode) Copy() Node
```

func (*BoolNode) String

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

type BranchNode

```
type BranchNode struct {
    NodeType
    Pos
```

```
    Line     int       // The line number in the input. Deprecated: Kept for compatibil
    Pipe     *PipeNode // The pipeline to be evaluated.
    List     *ListNode // What to execute if the value is non-empty.
    ElseList *ListNode // What to execute if the value is empty (nil if absent).
    // contains filtered or unexported fields
}
```

BranchNode is the common representation of if, range, and with.

func (*BranchNode) Copy

added in go1.4

```
func (b *BranchNode) Copy() Node
```

func (*BranchNode) String

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

type BreakNode

added in go1.18

```
type BreakNode struct {
    NodeType
    Pos
    Line int
    // contains filtered or unexported fields
}
```

BreakNode represents a {{break}} action.

func (*BreakNode) Copy

added in go1.18

```
func (b *BreakNode) Copy() Node
```

func (*BreakNode) String

added in go1.18

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

type ChainNode

added in go1.1

```
type ChainNode struct {
    NodeType
    Pos

    Node  Node
    Field []string // The identifiers in lexical order.
    // contains filtered or unexported fields
}
```

ChainNode holds a term followed by a chain of field accesses (identifier starting with '.'). The names may be chained ('.x.y'). The periods are dropped from each ident.

func (*ChainNode) Add

added in go1.1

```
func (c *ChainNode) Add(field string)
```

Add adds the named field (which should start with a period) to the end of the chain.

func (*ChainNode) Copy

added in go1.1

```
func (c *ChainNode) Copy() Node
```

func (*ChainNode) String

added in go1.1

```
func (c *ChainNode) String() string
```

type CommandNode

```
type CommandNode struct {
    NodeType
    Pos

    Args []Node // Arguments in lexical order: Identifier, field, or constant.
    // contains filtered or unexported fields
}
```

CommandNode holds a command (a pipeline inside an evaluating action).

func (*CommandNode) Copy

```
func (c *CommandNode) Copy() Node
```

func (*CommandNode) String

```
func (c *CommandNode) String() string
```

type CommentNode

added in go1.16

```
type CommentNode struct {
    NodeType
    Pos

    Text string // Comment text.
    // contains filtered or unexported fields
}
```

CommentNode holds a comment.

func (*CommentNode) Copy

added in go1.16

```
func (c *CommentNode) Copy() Node
```

func (*CommentNode) String

added in go1.16

```
func (c *CommentNode) String() string
```

type ContinueNode

added in go1.18

```
type ContinueNode struct {
    NodeType
    Pos
    Line int
    // contains filtered or unexported fields
}
```

ContinueNode represents a {{continue}} action.

func (*ContinueNode) Copy

added in go1.18

```
func (c *ContinueNode) Copy() Node
```

func (*ContinueNode) String

added in go1.18

```
func (c *ContinueNode) String() string
```

type DotNode

```
type DotNode struct {
    NodeType
    Pos
    // contains filtered or unexported fields
}
```

DotNode holds the special identifier '.'.

func (*DotNode) Copy

```
func (d *DotNode) Copy() Node
```

func (*DotNode) String

```
func (d *DotNode) String() string
```

func (*DotNode) Type

```
func (d *DotNode) Type() NodeType
```

type FieldNode

```
type FieldNode struct {
    NodeType
    Pos

    Ident []string // The identifiers in lexical order.
    // contains filtered or unexported fields
}
```

FieldNode holds a field (identifier starting with '.'). The names may be chained ('.x.y'). The period is dropped from each ident.

func (*FieldNode) Copy

```
func (f *FieldNode) Copy() Node
```

func (*FieldNode) String

```
func (f *FieldNode) String() string
```

type IdentifierNode

```
type IdentifierNode struct {
    NodeType
    Pos

    Ident string // The identifier's name.
    // contains filtered or unexported fields
}
```

IdentifierNode holds an identifier.

func NewIdentifier

```
func NewIdentifier(ident string) *IdentifierNode
```

NewIdentifier returns a new IdentifierNode with the given identifier name.

func (*IdentifierNode) Copy

```
func (i *IdentifierNode) Copy() Node
```

func (*IdentifierNode) SetPos

added in go1.1

```
func (i *IdentifierNode) SetPos(pos Pos) *IdentifierNode
```

SetPos sets the position. NewIdentifier is a public method so we can't modify its signature. Chained for convenience. TODO: fix one day?

func (*IdentifierNode) SetTree

added in go1.4

```
func (i *IdentifierNode) SetTree(t *Tree) *IdentifierNode
```

SetTree sets the parent tree for the node. NewIdentifier is a public method so we can't modify its signature. Chained for convenience. TODO: fix one day?

func (*IdentifierNode) String

```
func (i *IdentifierNode) String() string
```

type IfNode

```
type IfNode struct {
    BranchNode
}
```

IfNode represents an {{if}} action and its commands.

func (*IfNode) Copy

```
func (i *IfNode) Copy() Node
```

type ListNode

```
type ListNode struct {
    NodeType
    Pos

    Nodes []Node // The element nodes in lexical order.
    // contains filtered or unexported fields
}
```

ListNode holds a sequence of nodes.

func (*ListNode) Copy

```
func (l *ListNode) Copy() Node
```

func (*ListNode) CopyList

```
func (l *ListNode) CopyList() *ListNode
```

func (*ListNode) String

```
func (l *ListNode) String() string
```

type Mode

added in go1.16

```
type Mode uint
```

A mode value is a set of flags (or 0). Modes control parser behavior.

```
const (
    ParseComments Mode = 1 << iota // parse comments and add them to AST
    SkipFuncCheck                  // do not check that functions are defined
)
```

type NilNode

added in go1.1

```
type NilNode struct {
    NodeType
    Pos
    // contains filtered or unexported fields
}
```

NilNode holds the special identifier 'nil' representing an untyped nil constant.

func (*NilNode) Copy

added in go1.1

```
func (n *NilNode) Copy() Node
```

func (*NilNode) String

added in go1.1

```
func (n *NilNode) String() string
```

func (*NilNode) Type

added in go1.1

```
func (n *NilNode) Type() NodeType
```

type Node

```
type Node interface {
    Type() NodeType
    String() string
    // Copy does a deep copy of the Node and all its components.
    // To avoid type assertions, some XxxNodes also have specialized
    // CopyXxx methods that return *XxxNode.
    Copy() Node
    Position() Pos // byte position of start of node in full original input string
```

```
    // contains filtered or unexported methods
}
```

A Node is an element in the parse tree. The interface is trivial. The interface contains an unexported method so that only types local to this package can satisfy it.

type NodeType

```
type NodeType int
```

NodeType identifies the type of a parse tree node.

```
const (
    NodeText    NodeType = iota // Plain text.
    NodeAction                  // A non-control action such as a field evaluation.
    NodeBool                    // A boolean constant.
    NodeChain                   // A sequence of field accesses.
    NodeCommand                 // An element of a pipeline.
    NodeDot                     // The cursor, dot.

    NodeField      // A field or method name.
    NodeIdentifier // An identifier; always a function name.
    NodeIf         // An if action.
    NodeList       // A list of Nodes.
    NodeNil        // An untyped nil constant.
    NodeNumber     // A numerical constant.
    NodePipe       // A pipeline of commands.
    NodeRange      // A range action.
    NodeString     // A string constant.
    NodeTemplate   // A template invocation action.
    NodeVariable   // A $ variable.
    NodeWith       // A with action.
    NodeComment    // A comment.
    NodeBreak      // A break action.
    NodeContinue   // A continue action.
)
```

func (NodeType) Type

```
func (t NodeType) Type() NodeType
```

Type returns itself and provides an easy default implementation for embedding in a Node. Embedded in all non-trivial Nodes.

type NumberNode

```
type NumberNode struct {
    NodeType
    Pos
```

```
    IsInt      bool       // Number has an integral value.
    IsUint     bool       // Number has an unsigned integral value.
    IsFloat    bool       // Number has a floating-point value.
    IsComplex  bool       // Number is complex.
    Int64      int64      // The signed integer value.
    Uint64     uint64     // The unsigned integer value.
    Float64    float64    // The floating-point value.
    Complex128 complex128 // The complex value.
    Text       string     // The original textual representation from the input.
    // contains filtered or unexported fields
}
```

NumberNode holds a number: signed or unsigned integer, float, or complex. The value is parsed and stored under all the types that can represent the value. This simulates in a small amount of code the behavior of Go's ideal constants.

func (*NumberNode) Copy

```
func (n *NumberNode) Copy() Node
```

func (*NumberNode) String

```
func (n *NumberNode) String() string
```

type PipeNode

```
type PipeNode struct {
    NodeType
    Pos

    Line     int             // The line number in the input. Deprecated: Kept for compa
    IsAssign bool            // The variables are being assigned, not declared.
    Decl     []*VariableNode // Variables in lexical order.
    Cmds     []*CommandNode  // The commands in lexical order.
    // contains filtered or unexported fields
}
```

PipeNode holds a pipeline with optional declaration

func (*PipeNode) Copy

```
func (p *PipeNode) Copy() Node
```

func (*PipeNode) CopyPipe

```
func (p *PipeNode) CopyPipe() *PipeNode
```

func (*PipeNode) String

```
func (p *PipeNode) String() string
```

type Pos

added in go1.1

```
type Pos int
```

Pos represents a byte position in the original input text from which this template was parsed.

func (Pos) Position

added in go1.1

```
func (p Pos) Position() Pos
```

type RangeNode

```
type RangeNode struct {
    BranchNode
}
```

RangeNode represents a {{range}} action and its commands.

func (*RangeNode) Copy

```
func (r *RangeNode) Copy() Node
```

type StringNode

```
type StringNode struct {
    NodeType
    Pos

    Quoted string // The original text of the string, with quotes.
    Text   string // The string, after quote processing.
    // contains filtered or unexported fields
}
```

StringNode holds a string constant. The value has been "unquoted".

func (*StringNode) Copy

```
func (s *StringNode) Copy() Node
```

func (*StringNode) String

```
func (s *StringNode) String() string
```

type TemplateNode

```
type TemplateNode struct {
    NodeType
    Pos

    Line int       // The line number in the input. Deprecated: Kept for compatibility.
    Name string    // The name of the template (unquoted).
    Pipe *PipeNode // The command to evaluate as dot for the template.
    // contains filtered or unexported fields
}
```

TemplateNode represents a {{template}} action.

func (*TemplateNode) Copy

```
func (t *TemplateNode) Copy() Node
```

func (*TemplateNode) String

```
func (t *TemplateNode) String() string
```

type TextNode

```
type TextNode struct {
    NodeType
    Pos

    Text []byte // The text; may span newlines.
    // contains filtered or unexported fields
}
```

TextNode holds plain text.

func (*TextNode) Copy

```
func (t *TextNode) Copy() Node
```

func (*TextNode) String

```
func (t *TextNode) String() string
```

type Tree

```
type Tree struct {
    Name      string    // name of the template represented by the tree.
    ParseName string    // name of the top-level template during parsing, for error mes
    Root      *ListNode // top-level root of the tree.
    Mode      Mode      // parsing mode.
```

```
    // contains filtered or unexported fields
}
```

Tree is the representation of a single parsed template.

func New

```
func New(name string, funcs ...map[string]any) *Tree
```

New allocates a new parse tree with the given name.

func (*Tree) Copy

added in go1.2

```
func (t *Tree) Copy() *Tree
```

Copy returns a copy of the Tree. Any parsing state is discarded.

func (*Tree) ErrorContext

added in go1.1

```
func (t *Tree) ErrorContext(n Node) (location, context string)
```

ErrorContext returns a textual representation of the location of the node in the input text. The receiver is only used when the node does not have a pointer to the tree inside, which can occur in old code.

func (*Tree) Parse

```
func (t *Tree) Parse(text, leftDelim, rightDelim string, treeSet map[string]*Tree, fun
cs ...map[string]any) (tree *Tree, err error)
```

Parse parses the template definition string to construct a representation of the template for execution. If either action delimiter string is empty, the default ("{{" or "}}") is used. Embedded template definitions are added to the treeSet map.

type VariableNode

```
type VariableNode struct {
    NodeType
    Pos

    Ident []string // Variable name and fields in lexical order.
    // contains filtered or unexported fields
}
```

VariableNode holds a list of variable names, possibly with chained field accesses. The dollar sign is part of the (first) name.

func (*VariableNode) Copy

```
func (v *VariableNode) Copy() Node
```

func (*VariableNode) String

```
func (v *VariableNode) String() string
```

type WithNode

```
type WithNode struct {
    BranchNode
}
```

WithNode represents a {{with}} action and its commands.

func (*WithNode) Copy

```
func (w *WithNode) Copy() Node
```

Source Files

View all

lex.go node.go parse.go