* 为结构定义的方法必须放在同一个包内
* 可以是不同文件

包

如何扩充系统类型或者别人的类型?

* 定义别名
* 使用组合

1.目录tree内容详解

tree/node.go

```
// 包名
package treeDemo

import "fmt"

// 定义Node结构体
type Node struct {
   Left, Right *Node
   Value int
}

// 初始化结构体
func InitStruct() Node {

   var root Node

   root = Node{Value: 4}
   root.Left = &Node{Value: 2} // Left 是指针, 所以传递给Left是变量地址, 需要加&号
   root.Left.Right = CreateNode(3)
   root.Left.Left = &Node{Value: 1}
   root.Right = &Node{nil, nil, 5}
   root.Right.Right = new(Node)
   root.Right.Right.SetValue2(6)

   return root
}

// 工厂函数
func CreateNode(value int) *Node {
   return &Node{Value: value} // 返回局部变量的地址 给全局使用 C++中程序会挂
}

// (node Node) - 接收者
func (node Node) Print() {
   fmt.Print(node.Value)
}

// 值传递
func (node Node) setValue1 (value int) {
   node.Value = value
}

// 引用传递
func (node *Node) SetValue2 (value int) {
   node.Value = value
}

// 中序遍历
func (node *Node) Traverse() {
   if (node == nil) {
      return
   }
   node.Left.Traverse()
   node.Print()
   node.Right.Traverse()
}
```

1.目录tree内容详解

tree/entry/entry.go

```
package main

import (
   "LearnGo/tree"
   "fmt"
)

type myTreeNode struct {
   node *treeDemo.Node
}

// 后续遍历
func (myNode *myTreeNode) postOrder() {
   if myNode == nil || myNode.node == nil {
      return
   }
   myTreeNodeLeft := myTreeNode{myNode.node.Left}
   myTreeNodeLeft.postOrder()
   myTreeModeRight := myTreeNode{myNode.node.Right}
   myTreeModeRight.postOrder()
   myNode.node.Print()
}

func main() {
   initTree := treeDemo.InitStruct()
   initTree.Traverse()
   fmt.Println()
   myInitTree := myTreeNode{&initTree}
   myInitTree.postOrder()
}
```

* 结果:

```
123456
132654
```

2.目录queue内容详解

queue/queue.go

```
package queue

type Queue []int

func (q *Queue) Push(v int) {
  *q = append(*q, v)
}

func (q *Queue) Pop() int {
   head := (*q)[0]
   *q = (*q)[1:]

   return head
}

func (q *Queue) IsEmpty() bool {
   return len(*q) == 0
}
```

2.目录queue内容详解

queue/entry/entry.go

```
package main

import (
   "LearnGo/queue"
   "fmt"
)

func main() {
   initQueue := queue.Queue{}
   fmt.Println(initQueue.IsEmpty())
   initQueue.Push(1)
   initQueue.Push(3)
   initQueue.Push(4)
   initQueue.Push(5)
   fmt.Println(initQueue.IsEmpty())
   fmt.Println(initQueue.Pop())
}
```

* 结果:

```
true
false
1
```