* go语言仅支持封装，不支持继承和多态

* go语言没有class，只有struct

1.结构的定义：

```
type TreeNode struct {
    Left, Right *TreeNode
    Value int
}
```

2.结构创建：

* 不论地址还是结构本身，一律使用.来访问成员

```
func initStruct() {
   // 1.初始化struct
   // 方式一
   var root TreeNode

   root = TreeNode{Value:3}
   root.Left = &TreeNode{} // Left 是指针，所以传递给Left是变量地址，需要加&号
   root.Right = &TreeNode{nil, nil, 5}
   root.Right.Left = new(TreeNode)

   // slice
   nodes := []TreeNode {
      {Value :  5},
      {nil, &root, 6},
   }
   fmt.Println(nodes)
}
```

3.工厂函数

* 使用自定义工厂函数
* 注意返回了局部变量地址！！！

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

root.Left.Right = createNode(3)
```

4.结构创建在堆上还是栈上?

* 不需要知道
* 局部变量可能在栈上
* 局部变量地址返回给全局使用，可能变量就分配在堆上

5.为结构定义方法

* 显示定义和命名方法接收者

```
// (node TreeNode) - 接收者
// 参数是值传递
// go语言所有参数传递都是值传递
func (node TreeNode) print() {
   fmt.Print(node.Value)
}

func print(node TreeNode) {
   fmt.Print(node.Value)
}
```

* 调用：

```
root.print()
fmt.Println()
print(root)
```

6.参数类型传递

* 使用指针作为方法的接收者(只有指针才能改变结构内容)
* nil指针也能调用方法！！！

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

// 引用传递
func (node *TreeNode) setValue2 (value int) {
   node.Value = value
}

root.Right.Left = new(TreeNode)


root.Right.Left.setValue1(1)
root.Right.Left.print()
fmt.Println()
root.Right.Left.setValue2(2)
root.Right.Left.print()
```

* 结果：

```
0
2
```

7.中序遍历

```
// 中序遍历
func (node *TreeNode) traverse() {
   if (node == nil) {
      return
   }
   node.Left.traverse()
   node.print()
   node.Right.traverse()
}
```

8.值接收者 vs 指针接收者

* 要改变内容的使用指针接收者
* 结构过大也要考虑使用指针接收者
* 一致性：如果有指针接收者，使用指针接收者

* 值接收者 才是go语言特有的
* 值/指针接收者均可接收值/指针