# 解释器模式

## 一、基本概念 **1、定义：**

##### 解释器模式（Interpreter），给定一个语言，定义其文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。

##### 优势是可以针对频率较高的词汇进行构建一个解释器。

##### 劣势是难度可能比较大。

### 2、组成环节：

##### AbstractExpression(抽象表达类):定义了一个抽象的解释操作，这个接口为抽象语法树中所有的节点所共享。

##### TerminalExpression(终结符表达类):继承AbstractExpression类，实现文法当中终结符相关联的解释操作。

##### NonTerminalExpression(非终结符表达类):为文法中的非终结符实现解释操纵，对每一条规则都需要一个具体的非终结符表达式。

## 二、go语言中实现

##### 1、首先定义AbstractExpression类，并列出该类型所拥有的方法。

type AbstractExprssion interface {  
 Interpret(s string)  
}

##### 2、再定义其下的两类，并列出该类型所拥有的方法和具体实现方法。

type TerminalExprssion struct{}  
  
type NonTerminalExprssion struct{}  
  
func (t TerminalExprssion) Interpret(s string) {  
 if strings.Contains(s, "Y") {  
 println("TerminalExprssion.")  
 }  
}  
  
func (n NonTerminalExprssion) Interpret(s string) {  
 if strings.Contains(s, "N") {  
 println("NonTerminalExprssion.")  
 }  
}

##### 3、通过main方法进行client端调用。

func main() {  
 str1 := "N"  
 str2 := "Y"  
 t := TerminalExprssion{}  
 n := NonTerminalExprssion{}  
 t.Interpret(str1)  
 t.Interpret(str2)  
  
 n.Interpret(str1)  
 n.Interpret(str2)  
}

##### 4、输出结果

**TerminalExprssion.**

**NonTerminalExprssion.**