## 解释器（Interpreter）

### Intent

为语言创建解释器，通常由语言的语法和语法分析来定义。

### Class Diagram

* TerminalExpression：终结符表达式，每个终结符都需要一个 TerminalExpression。
* Context：上下文，包含解释器之外的一些全局信息。

### Implementation

以下是一个规则检验器实现，具有 and 和 or 规则，通过规则可以构建一颗解析树，用来检验一个文本是否满足解析树定义的规则。

例如一颗解析树为 D And (A Or (B C))，文本 "D A" 满足该解析树定义的规则。

这里的 Context 指的是 String。

public abstract class Expression {
 public abstract boolean interpret(String str);
}

public class TerminalExpression extends Expression {

 private String literal = null;

 public TerminalExpression(String str) {
 literal = str;
 }

 public boolean interpret(String str) {
 StringTokenizer st = new StringTokenizer(str);
 while (st.hasMoreTokens()) {
 String test = st.nextToken();
 if (test.equals(literal)) {
 return true;
 }
 }
 return false;
 }
}

public class AndExpression extends Expression {

 private Expression expression1 = null;
 private Expression expression2 = null;

 public AndExpression(Expression expression1, Expression expression2) {
 this.expression1 = expression1;
 this.expression2 = expression2;
 }

 public boolean interpret(String str) {
 return expression1.interpret(str) && expression2.interpret(str);
 }
}

public class OrExpression extends Expression {
 private Expression expression1 = null;
 private Expression expression2 = null;

 public OrExpression(Expression expression1, Expression expression2) {
 this.expression1 = expression1;
 this.expression2 = expression2;
 }

 public boolean interpret(String str) {
 return expression1.interpret(str) || expression2.interpret(str);
 }
}

public class Client {

 /\*\*
 \* 构建解析树
 \*/
 public static Expression buildInterpreterTree() {
 // Literal
 Expression terminal1 = new TerminalExpression("A");
 Expression terminal2 = new TerminalExpression("B");
 Expression terminal3 = new TerminalExpression("C");
 Expression terminal4 = new TerminalExpression("D");
 // B C
 Expression alternation1 = new OrExpression(terminal2, terminal3);
 // A Or (B C)
 Expression alternation2 = new OrExpression(terminal1, alternation1);
 // D And (A Or (B C))
 return new AndExpression(terminal4, alternation2);
 }

 public static void main(String[] args) {
 Expression define = buildInterpreterTree();
 String context1 = "D A";
 String context2 = "A B";
 System.out.println(define.interpret(context1));
 System.out.println(define.interpret(context2));
 }
}

true
false

### JDK

* [java.util.Pattern](http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)
* [java.text.Normalizer](http://docs.oracle.com/javase/8/docs/api/java/text/Normalizer.html)
* All subclasses of [java.text.Format](http://docs.oracle.com/javase/8/docs/api/java/text/Format.html)
* [javax.el.ELResolver](http://docs.oracle.com/javaee/7/api/javax/el/ELResolver.html)