解释器角色：

1、抽象解释器（AbstractExpression）：具体的解释任务由各个实现类完成。这个接口主要是一个interpreter()方法，称做解释操作。

2、终结符表达式（TerminalExpression）：实现与文法中的元素相关联的解释操作，通常一个解释器模式中只有一个终结表达式，但有多个实例，对应不同的终结符。

3、非终结符表达式（NonterminalExpression）：文法中的每条规则对应于一个非终结表达式，非终结符表达式根据逻辑的复杂程度而增加，原则上每个文法规则都对应一个非终结符表达式。比如公式R=R1+R2中，“+"就是非终结符，解析“+”的解释器就是一个非终结符表达式。

4、上下文（Context）:环境，它的任务一般是用来存放文法中各个终结符所对应的具体值，比如R=R1+R2，R1应该解释为100，R2应该解释为200，这些信息需要存放到环境中。

5、客户类（Client）: 客户端,解析表达式,构建抽象语法树,执行具体的解释操作等.

总结

优点：

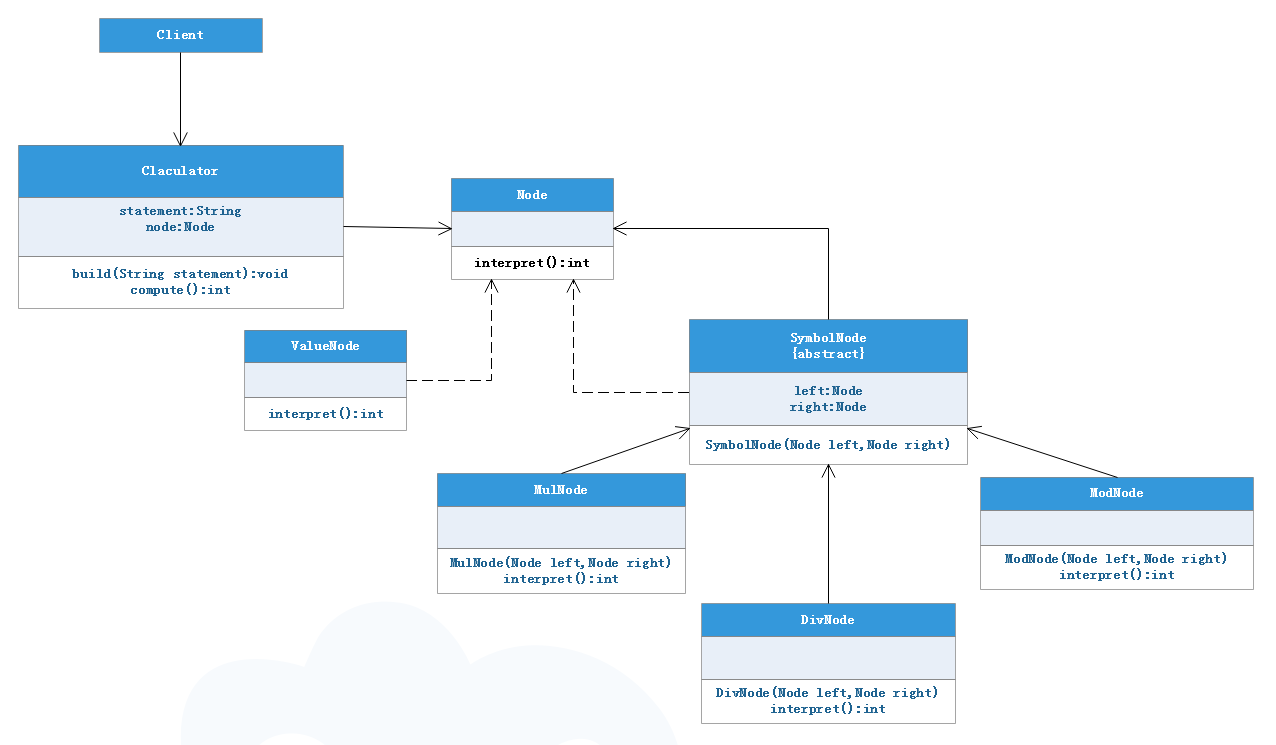
（1）扩展性强，若要新增乘，除，添加相应的非终结表达式，修改计算逻辑即可。

缺点：

（1）需要建大量的类，因为每一种语法都要建一个非终结符的类。

（2）解释的时候采用递归调用方法，导致有时候函数的深度会很深，影响效率。

举例：现在我们用解释器模式来实现一个基本的加、减、乘、除和求模运算。例如用户输入表达式“3 \* 4 / 2 % 4”，输出结果为2。下图为该实例的UML结构图：



抽象语法树：

