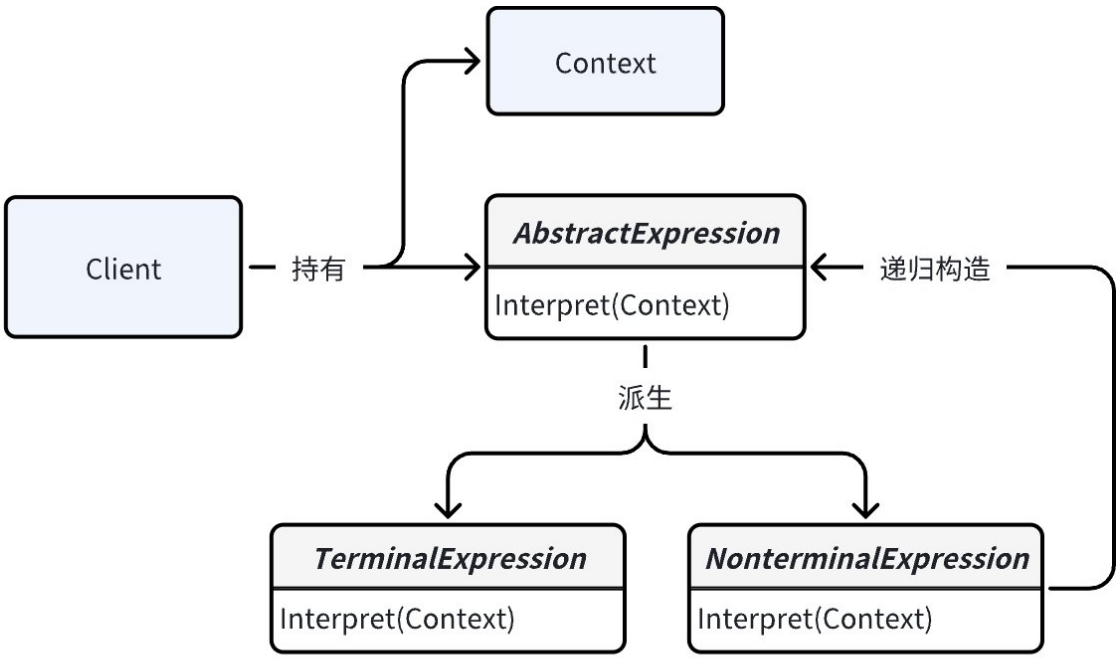
解释器模式

概念

- 解释器模式（Interpreter Pattern）是一种行为设计模式，主要用于定义一种语言的文法表示，它将一种表达式的语义进行抽象和封装，并建立一个解释器来解释该语言中的句子。
- 使用场景
  - 需要解释执行一个特定的语言或语法规则时。
  - 需要实现一个可定制的、可扩展的语法解析器时，如配置文件的解析、SQL查询的解析等。
  - 需要实现一个可动态改变解析规则的系统时，如脚本引擎、模板引擎等。
- 解释器模式包含以下几个组成部分：
  - 抽象表达式（Abstract Expression）：定义解释器的接口，约定解释操作。
  - 终结符表达式（Terminal Expression）：实现与文法中的终结符相关的解释操作。
  - 非终结符表达式（Nonterminal Expression）：实现与文法中的非终结符相关的解释操作。
  - 环境（Context）：包含解释器需要的全局信息。
  - 客户端（Client）：构建抽象语法树，并调用解释操作。
- 优点
  - 扩展性好：可以通过继承等机制来改变或扩展文法。
  - 容易实现：在语法树中的每个表达式节点类都是相似的。
- 缺点
  - 执行效率较低：通常使用大量的循环和递归调用，当要解释的句子较复杂时，运行速度较慢。
  - 类膨胀：每条规则至少需要定义一个类，当包含的文法规则很多时，类的个数将急剧增加。
  - 应用场景少：需要定义语言文法的应用实例非常少。



实例

- 抽象表达类

```
public abstract class Expression
{
    public abstract int Interpret(Dictionary<string, int> context);
}
```

- 终结符表达式

```
public class NumberExpression : Expression
{
    private int _number;

    public NumberExpression(int number)
    {
        _number = number;
    }

    public override int Interpret(Dictionary<string, int> context)
    {
        return _number;
    }
}
```

- 非终结符表达式

```
public class AddExpression : Expression
{
    private Expression _leftExpression;
    private Expression _rightExpression;

    public AddExpression(Expression left, Expression right)
    {
        _leftExpression = left;
        _rightExpression = right;
    }

    public override int Interpret(Dictionary<string, int> context)
    {
        return _leftExpression.Interpret(context) +
_rightExpression.Interpret(context);
    }
}
```

- 客户端

```
static void Main(string[] args)
{
    // 构建抽象语法树
    Expression expression = new AddExpression(
        new NumberExpression(5),
        new AddExpression(
            new NumberExpression(10),
            new NumberExpression(20)
        )
    );

    // 解释并计算表达式
    Dictionary<string, int> context = new Dictionary<string, int>();
    int result = expression.Interpret(context);

    Console.WriteLine($"结果: {result}"); // 输出结果: 35
}
```