🔄 递归调用可视化教程

🎯 核心概念：递归下降解析

关键理解：每个解析函数对应一个语法规则，通过递归调用来处理嵌套结构。

📊 调用栈可视化

示例1：简单表达式 `2 + 3 * 4`

```
表达式: 2 + 3 * 4

parse_expression()                    ← [加减法层]
├── parse_term()                      ← [乘除法层]
│   ├── parse_factor()                ← [基本元素层]
│   │   └── 返回: 数字节点 2
│   ├── 遇到: '+'                   ← [回到加减法层]
│   └── parse_term()                  ← [新的乘除法层]
│       ├── parse_factor()
│       │   └── 返回: 数字节点 3
│       ├── 遇到: '*'
│       └── parse_factor()
│           └── 返回: 数字节点 4
│       └── 返回: 乘法节点 (3 * 4)
└── 返回: 加法节点 (2 + (3 * 4))
```

示例2：括号表达式 `(2 + 3) * 4`

```
# 使用调试查看调用栈
echo "(2 + 3) * 4" | ./minic-new -va

调用过程：
parse_expression()                    ← 顶层
├── parse_term()
│   ├── parse_factor()                ← 遇到'('
│   │   ├── 遇到：'('
│   │   ├── parse_expression()        ← 递归调用！
│   │   │   ├── parse_term()
│   │   │   │   ├── parse_factor() → 2
│   │   │   │   ├── '+'
│   │   │   │   └── parse_factor() → 3
│   │   │   └── 返回：(2 + 3)
│   │   ├── 遇到：')'
│   │   └── 返回：(2 + 3)
│   ├── '*'
│   └── parse_factor() → 4
└── 返回：((2 + 3) * 4)
```

🎓 递归思维训练

关键理解（重要！）：

1. **函数调用层次 = 优先级层次**
2. **低优先级可以调用高优先级**（⚠️纠正之前的表述）
3. **括号通过递归实现优先级提升**

⚠️ 重要澄清：调用方向

```
"低优先级层可以调用高优先级层，但反之不行"

示例：2 + 3 * 4
parse_expression() [+ - 最低] → 可以调用 → parse_term() [* / 中等]
parse_term() [* / 中等] → 可以调用 → parse_factor() [数字/括号 最高]
parse_factor() [最高] → 不能调用 → parse_expression() [会破坏优先级]
```

记忆口诀：

```
"表达式调用项，项调用因子，括号回到表达式"
expr → term → factor → (回到expr)
```

🧪 动手实验：追踪调用

实验1：理解调用层次

```
# 追踪: 2 + 3 * 4
echo "2 + 3 * 4" | ./minic-new -va
# 观察: parse_expression → parse_term → parse_factor
```

实验2：理解括号递归

```
# 追踪: (2 + 3) * 4
echo "(2 + 3) * 4" | ./minic-new -va
# 观察: 括号如何触发递归调用
```

实验3：复杂表达式

```
# 追踪: 1 + 2 * 3 - 4 / 2
echo "1 + 2 * 3 - 4 / 2" | ./minic-new -va
# 观察: 完整的递归下降过程
```

📊 递归调用对比表

表达式	调用深度	主要特点
`2 + 3`	2层	简单加减
`2 + 3 * 4`	3层	优先级体现
`(2 + 3) * 4`	4层	括号递归
`((1 + 2) * 3) - 4`	5层	嵌套括号

🎯 调试技巧

使用调试工具查看调用栈：

```
# 查看每次函数调用的详细信息
./minic-new -va -ve

# 输入：2 + 3 * 4
# 输出示例：
# → parse_expression: 开始处理表达式
#   → parse_term: 开始处理项
#     → parse_factor: 处理数字 2
#   → parse_term: 处理 3 * 4
#     → parse_factor: 处理数字 3
#     → parse_factor: 处理数字 4
# → 结果：14.00
```