2500 行代码实现高性能数值表达式引擎

南京都昌信息科技有限公司 袁永福 2018-9-23
Http://www.dcwriter.cn

◆◆前言

在一些高自由度的软件中，特别是报表之类的软件。需要让用户自定义数值表达式，比如定义“A+B*C-D/E”,然后再实际运行中把具体的 A,B,C,D,E 的值代入表达式运算。这能显著增加软件的运行时的可配置性，是一个值得广泛应用的软件功能。本文就说明了如何使用 2500 行 C#代码实现一种高性能的数值运算表达式引擎。

◆◆ANTLR 引擎

提到数值表达式引擎，不得不提起 Antlr，一个很著名的开源软件，能自动生成源代码来生成语法解析引擎，然后可以在这个语法解析引擎的基础上来实现运算表达式。

笔者此前也在使用 ANTLR 相关的代码来实现了运算表达式引擎，包含了 10000 行 C#源代码。不过代码晦涩难懂，改进更加麻烦。近期在处理一个包含大量表达式的场景时出现了性能问题，需要改进。

◆◆新表达式运算引擎

现在作者抛弃了旧的基于 ANTLR 的引擎，构造了全新的表达式运算引擎，核心模块只有 2500 行 C#代码，生成的程序集文件只有 27KB，但运行速度提升了 10 倍。而且程序简单易懂，扩展性强。当然不再具备 ANTLR 的广泛的通用性，但足够应付作者遇到的应用场景了。

新源代码中定义的主要类型有：

DCConstExpressionItem	常量表达式元素。
DCExpression	表达式对象，为顶级 API 类型。
DCExpressionItem	抽象的表达式元素类型。所有的表达式元素都是从这个类型派生出来的。
DCExpressionItemList	表达式元素列表。
DCFunctionExpressionItem	函数调用表达式元素。
DCGroupExpressionItem	表达式元素组。
DCOperatorExpressionItem	运算操作符表达式元素。
DCToken	表达式语法中的标识符对象。
DCTokenList	标识符列表
DCVariableExpressionItem	变量表达式元素。
IDCExpressionContext	运行表达式时的上下文对象接口。

新引擎工作过程如下：

◆第一步，表达式字符串符号解析

其代码定义在 DCToken.cs 中。在此处，将字符分为以下几种：

标识符	包括 0 到 9 的数字字符，英文字母字符，$符号，其他标识符类型的字符。
操作符	包括“+-*/%\”字符。
逻辑运算符	包括“&^|=><”字符。

分组开始字符	为字符“(”。
分组结束字符	为字符“)”。

所有相邻的同类型的字符合并在一起成为一组符号，此外还识别单引号或双引号作为边界的字符串常量。

以下是符号解析的例子：

原始文本	解析结果
A+B	“A”，”+”，”B”
A+SIN(B)*99	“A”，“+”，“SIN”，”(“，“B”，“)”，“*”，”99”
A+B>98 && C<10	“A”，“+”，”B”，“>”，“98”，“&&”，“C”，“<”，“10”
体重/(身高*身高)	“体重”，“/”，“(“ ，“身高”，“*”，“身高”，“)”

◆第二步，解析表达式单元元素

其代码定义在 DCExpression.cs 中的 Parse()/ParseItem()函数中。将一个个标识符转换为表达式单元元素。目前支持以下元素类型：

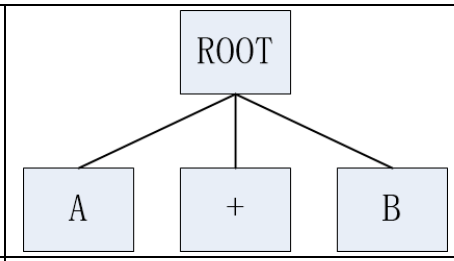
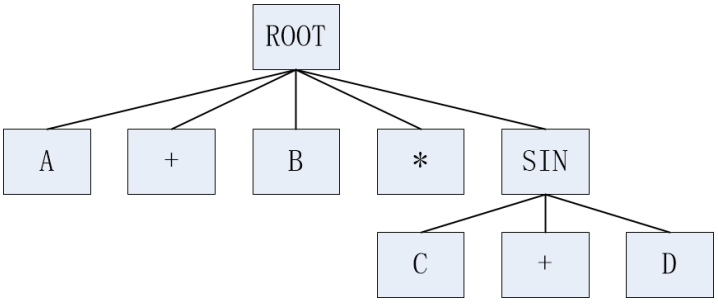
DCConstExpressionItem	常量元素类型。分为字符串/数字/布尔值三种类型。
DCFunctionExpressionItem	函数元素类型。用于调用外部函数。
DCGroupExpressionItem	分组元素类型。由一对圆括号定义的子单元元素组合。
DCOperatorExpressionItem	操作符元素。
DCVariableExpressionItem	变量元素类型。由外部传入具体的变量值。

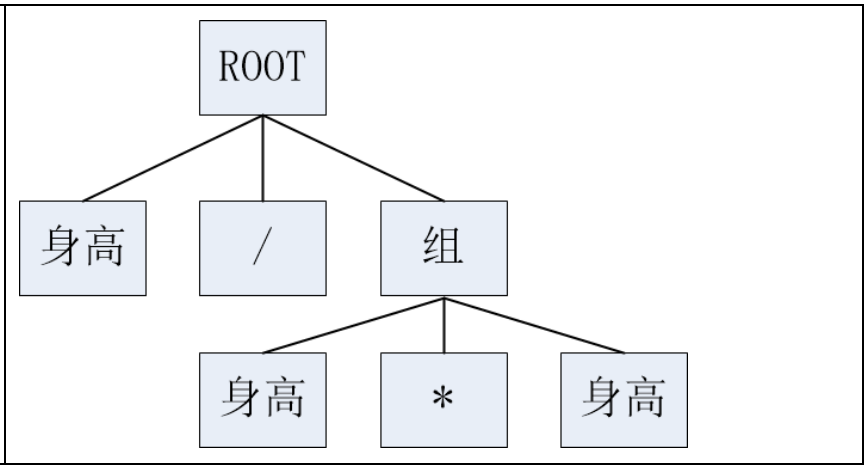
解析过程是一种递归操作,用于构造出一个表达式元素树状结构。

首先定义一个 DCGroupExpressionItem 作为根元素。可以看作把表达式的最外头放了一组圆括号。比如把“A+B”当作“(A+B)”。

在这个过程中，遇到标识符并且紧跟着的是“(”则认为是函数元素；遇到标识符”true”或”false”则认为是布尔类型的常量元素；遇到可以转换为数字的标识符则认为是数字常量元素；遇到其他标识符则认为是变量元素；遇到操作标识符则认为是操作符元素类型；遇到“(”则为分组元素类型并递归解析子元素；遇到“)”则认为结束分组元素类型，结束当前递归。此外还解析出字符串常量元素。

经过解析操作，构造了只有一个根节点的表达式元素树状结构。以下是几个例子：

A+B	
A+B*SIN(C+D)	

体重/(身高*身高)	

◆第三步，优先级调整

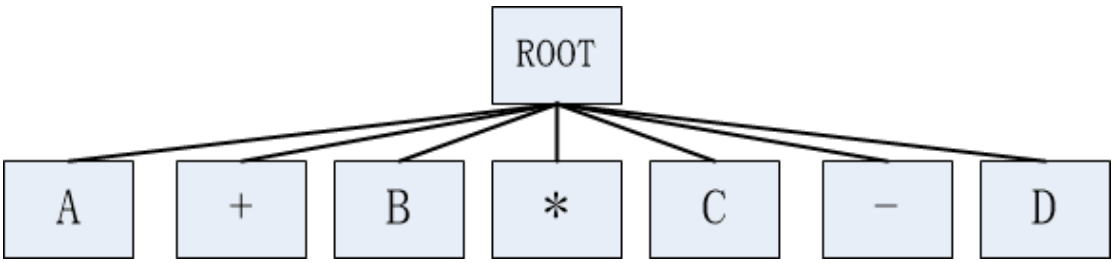
数值运算有优先级。规则是数学运算高于逻辑运算，乘除运算高于加减运算，圆括号可以改变运算优先级。

此时需要将一长串的表达式元素按照优先级在此次整理。主要代码在 DCExpression.cs 中的 CollpaseItems()中，作者称之为表达式元素列表的收缩。其过程如下：

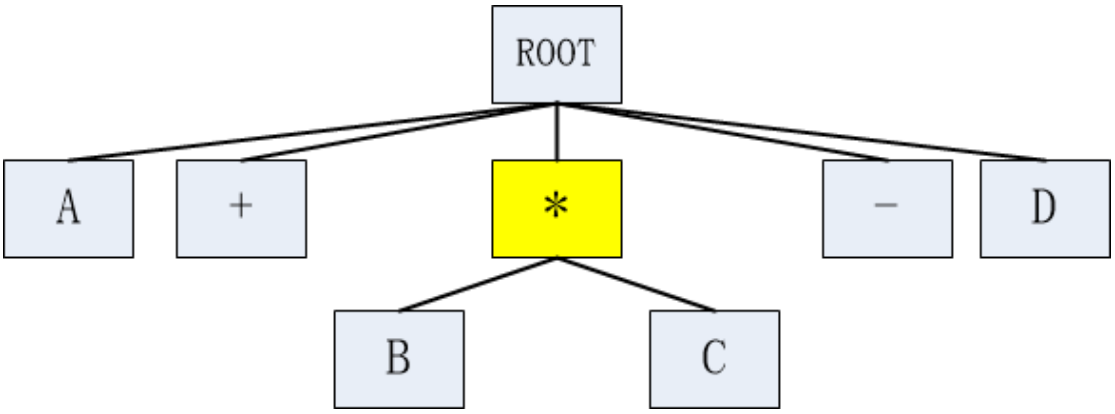
首先是特别处理减号元素的处理。当减号处于当前组的第一的位置，或者其前面是逻辑运算符时，则将减号元素转换为取负元素。

然后对元素列表进行一个不定次数的循环。在循环体中，找到优先级最高的而且未经收缩的操作符元素，然后吞并左边和右边的元素。如此循环直到没法产生吞并操作为止。

举个例子，对于表达式“A+B*C-D”，则生成的表达式元素有

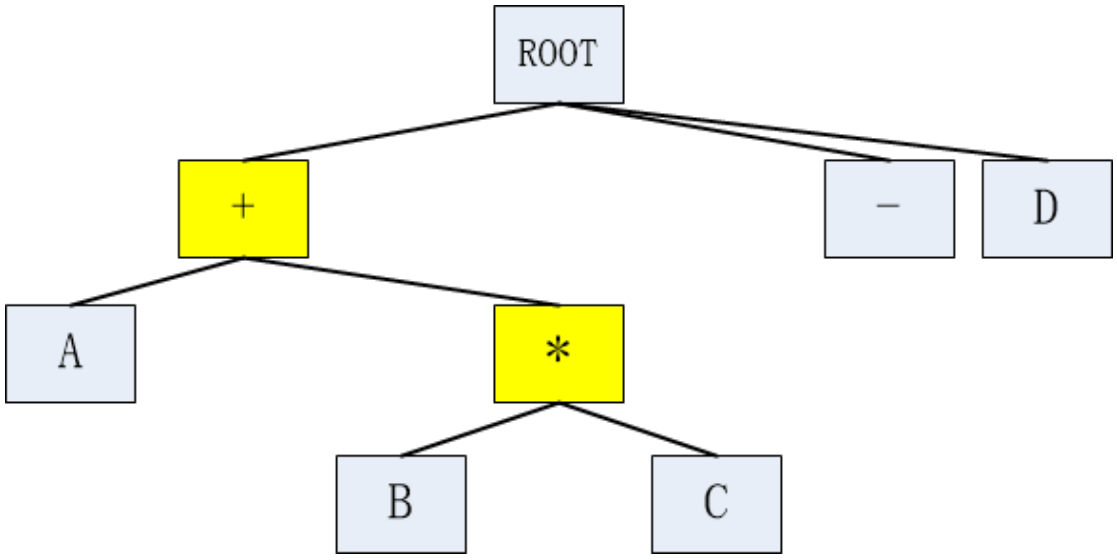


在第一次循环中，*号是优先级最高的，则进行收缩处理，处理后的结构如下：

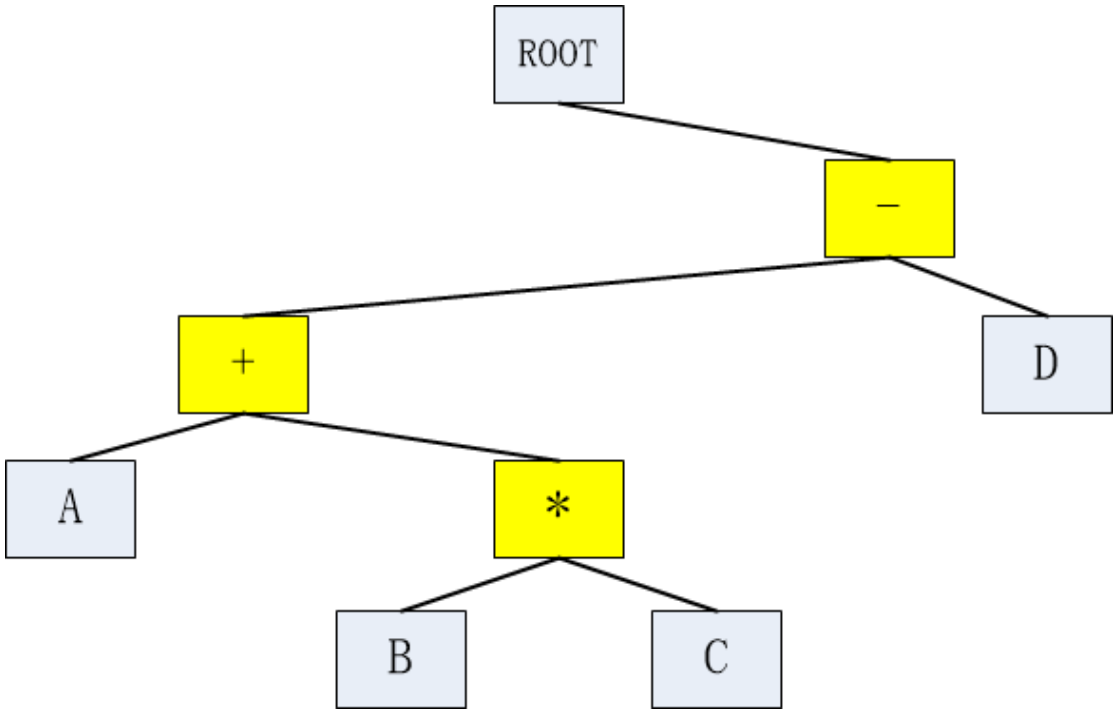


上图中黄色元素表示该元素收缩处理过了，不再参与处理。

在第二次循环中，“+”号优先级最高（后面的“-”和它是同等级的），则对它进行收缩，结果如下：



在第三次循环中，“-”号优先级最高，则进行收缩，结果如下：



之后根节点下再无可以处理的运算符元素，则退出循环。

经过上述步骤，最为关键的表达式元素树状列表产生出来了。

◆第四步，运行表达式

本功能的入口点在 DCExpression.cs 中的 Eval()中，内部调用了根元素的 Eval()，然后递归调用个子元素的 Eval()，获得最终的运算结果。

运算过程需要一个运行环境上下文对象，上下午对象只有 2 个功能：执行外界函数和取变量值。

为此定义了 IDCExpressionContext 接口，其代码如下：

```
namespace DCSoft.Expression
{
    /// <summary>
    /// 表达式执行上下文对象
    /// </summary>
    [System.Runtime.InteropServices.ComVisible( false )]
    public interface IDCExpressionContext
```

```
    {
        /// <summary>
        /// 执行函数
        /// </summary>
        /// <param name="name">函数名</param>
        /// <param name="parameters">参数列表</param>
        /// <returns>函数返回值</returns>
        object ExecuteFunction(string name, object[] parameters);
        /// <summary>
        /// 获得变量值
        /// </summary>
        /// <param name="name">变量名</param>
        /// <returns>变量值</returns>
        object GetVariableValue(string name);
    }
}
```

应用程序只需定义一个实现了该接口的类型即可传递给表达式执行引擎即可进行数值表达式运算了。

◆◆应用

这个表达式引擎完成后，开发者就能很方便的使用这个引擎了。首先是定义实现了 IDCExpressionContext 接口的上下文对象。其代码如下：

```
private class MyContext : IDCExpressionContext
{
    public object ExecuteFunction(string name, object[] parameters)
    {
        name = name.ToUpper();
        try
        {
            switch (name)
            {
                case "SIN": return Math.Sin(Convert.ToDouble(parameters[0]));
                case "MAX":
                    return Math.Max(
                            Convert.ToDouble(parameters[0]),
                            Convert.ToDouble(parameters[1]));
                default:
                    MessageBox.Show("不支持的函数" + name);
                    return 0;
            }
        }
        catch (System.Exception ext)
        {
            MessageBox.Show("执行函数" + name + " 错误:" + ext.Message);
            return 0;
        }
    }

    public object GetVariableValue(string name)
    {
        name = name.ToUpper();
        switch( name )
        {
            case "A":return 1.5;
            case "B": return 2.3;
            case "C": return -99;
            case "D": return 998;
            case "E": return 123.5;
            case "F": return 3.1415;
            default:
                MessageBox.Show("不支持的参数名:" + name);
                return 0;
        }
    }
```

```
}
```

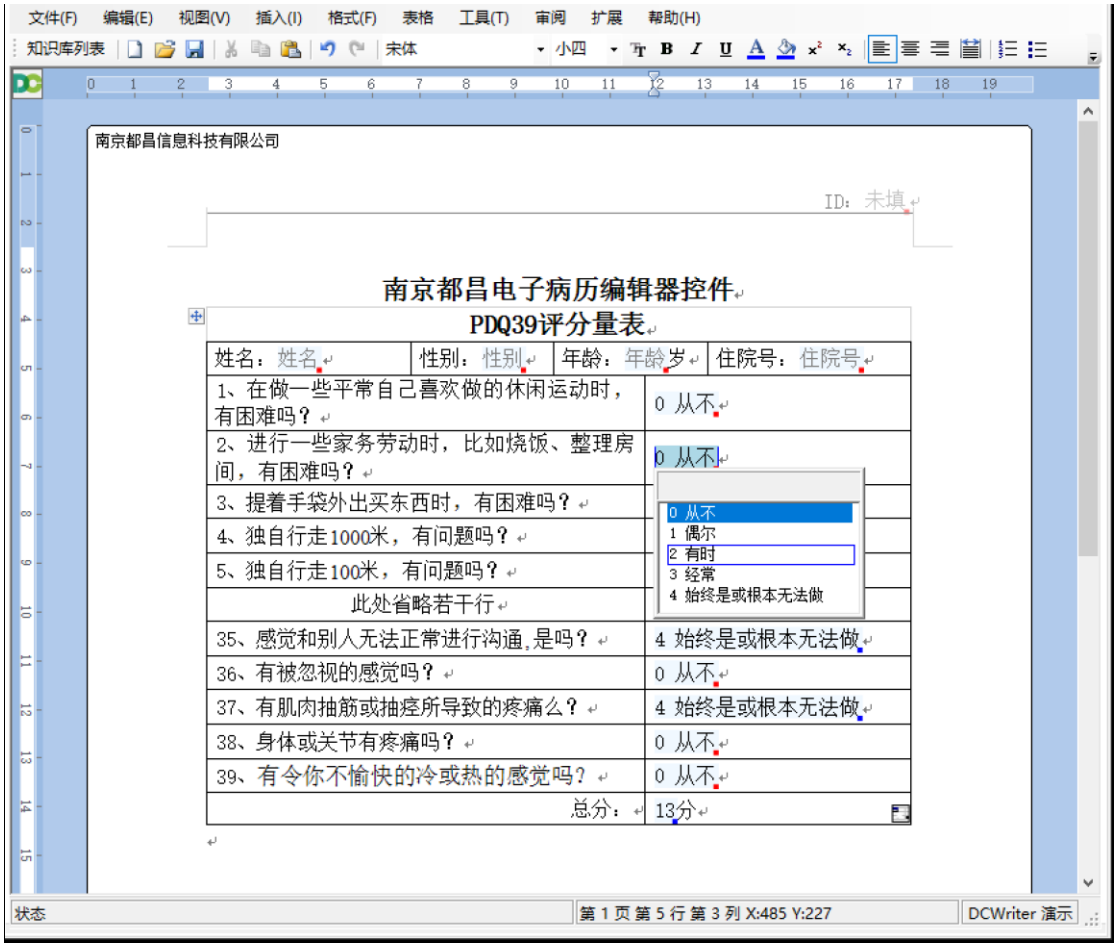
在这个上下文中，通过.NET 的反射机制将表达式对函数的调用转换为对 System.Math 下面的函数的调用；将变量 PI 的取值为 Math.PI 值。

然后我们可以使用以下代码来调用表达式引擎了。

```
private void Form1_Load(object sender, EventArgs e)
{
    cboExpression.Items.Add("SIN(99+123)");
    cboExpression.Items.Add("A*99.1+MAX(-11,-10)");
    cboExpression.Items.Add("FIND('7048dcb034d94ce6bfd750c3f1672096',[zz])>=0");
    cboExpression.Items.Add("SUM(A1:B3)");
    cboExpression.Items.Add("[A1]-[B2]");
    cboExpression.Items.Add("A+B+C+在三+是+213");
    cboExpression.Items.Add("A+B*(C+D*(E-F))-999");
    cboExpression.Items.Add("-A+B+C+D");
    cboExpression.Items.Add("A+B*C-D");
    cboExpression.Items.Add("10+8>-9");
}
private void toolStripButton1_Click(object sender, EventArgs e)
{
    DCExpression exp = new DCExpression(cboExpression.Text);
    MyContext c = new MyContext();
    object vresult = exp.Eval(c);
    txtResult.Text = "运算结果:" + vresult;
}
```

如此这样我们用 2500 行代码实现了一个功能强大、性能卓越的数值表达式引擎。可广泛用于各类软件开发。

在笔者主导开发的电子病历编辑器控件中就使用了这个数值表达式引擎来实现了类似 EXCEL 的公式运算功能，其界面如下：



在这个文档中，我们设置右下角的单元格的数值表达式为“SUM([D3:D13])”，则用户以

下拉列表方式设置 D3:D13 区域的单元格内容时，软件会调用 DCSoft.Expression 的功能，自动执行数值运算，并将运算结果显示在右下角单元格中。

◆◆小结

在本文中使用了 2500 行 C#代码实现了一个结构清晰、易于理解和掌握、性能卓越的数值表达式运算引擎。支持四则数学运算、布尔逻辑运算、能调用外部函数和外部参数值。很容易实现一个类似 EXCEL 的公式运算引擎。

使用该引擎，能显著提高软件的自由度，让软件在运行阶段即可自由改变运行模式，实现了软件的高度的运行时可配置化。

中秋之夜编写此文，诗性大发：玉宇无尘，朗月虚空三千里；架构有道，代码奔腾百万行。献给红尘中这百万代码人共勉。

◆◆关于作者

袁永福，80 后，南京东南大学毕业，中国知名医疗信息技术专家，南京都昌信息科技有限公司（www.dcwriter.cn）的联合创始人，微软 MVP，从事软件研发近 20 年。长期从事电子病历软件底层技术的研发和推广，对医学病历文档技术有着深厚的积累。