DOI:10.13255/j.cnki.jusst.2003.03.017

文章编号：1007-6735(2003)03-0271-06

利用语法树和算符优先法构造的自动计算系统

杨新宇， 丁岳伟， 陈志浩
(上海理工大学 计算机工程学院，上海 200093)

摘要：针对编程过程中常见的公式更改及扩充情况，设计了一种基于语法树和算符优先法的计算系统，可提供公式输入、界面维护以及动态计算的完整功能.

关键词：语法树；算符优先法；自动计算
中图分类号：TP 312 **文献标识码**：A

Automatic computing system based on syntax tree and operator precedence

YANG Xin-yu, DING Yue-wei, CHEN Zhi-hao
(*College of Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China*)

Abstract: In developing information system it is often necessary to compute the result of some given formulas. For formulas that are often altered or extended as demanded, difficulty will appear when implementing the program. In the paper, an information system that supplies the complete functions of accepting formula input, maintaining interface, and computing dynamically is designed and implemented.

Key words: *syntax tree*; *operator precedence*; *automatic computing*

在很多领域，尤其是在一些MIS软件中，常常有这样的需求，即要求根据某一个公式计算出需要的结果. 但是，由于公式经常变化，开发者不能够简单地将公式编码到程序中，这样既不利于程序的实现，也难以进行系统的扩充、维护.

为此，本文作者设计并实现了以下的系统. 该系统解决了上述的公式可变性问题，同时提供了完整的解决方案，包括如下内容：

a. 接受用户公式输入的界面，将其保存到数据库中；

b. 提供编程接口，让开发人员利用此接口选择合适的公式并提供合适的参数，从而计算出结果；

c. 当公式发生变化时，仅仅需要通过图形界面对数据库中的公式做出修改，而不需要修改程序的源代码.

其实现的关键技术在于：

a. 如何保证用户在输入时公式的正确性，而不是在运行时才发现公式根本就是不合法的；

b. 如何根据文本化的公式计算出结果.

在本文中，通过构造 LL(1)文法的语法树来完成第一项任务，用算符优先法完成第二项任务.

1 理 论

以如下形式表达通用公式. 实践证明，这种形式可以满足大部分的需要.

收稿日期：2003-02-18
作者简介：杨新宇(1977-)，男，硕士研究生.

$$\begin{cases} \text{表达式}1\cdots\text{条件}1 \\ \text{表达式}2\cdots\text{条件}2 \\ \quad\vdots \\ \text{表达式}n\cdots\text{条件}n \end{cases}$$

为了计算此公式,从条件 1 开始依次计算各个条件,其中第一个结果为真的条件所对应的表达式的值就是这个公式的值. 如果没有一个条件结果为真,结果为 java.lang.Double.NaN. 其中表达式的运算是常见的四则运算和乘方(含开方)运算. 在条件中还包括了关系运算和逻辑运算,这种表达基本可以满足常见的运算要求.

1.1 描述表达式和条件的文法[1]

描述表达式和条件(表达式)的文法如下:

Condition→{Condition} TMP2	C1
Condition→~{Condition} TMP2	C2
Condition→RelationExpr TMP2	C3
TMP2→LogicOp Condition	T21
TMP2→ϕ	T22
RelationExpr→Expr RelationOp Expr	R
Expr→varialbe TMP1	E1
Expr→value TMP1	E2
Expr→(Expr) TMP1	E3
TMP1→ArithOp Expr	T11
TMP1→ϕ	T12

variable→'['IntNum,IntNum ']'

value→ '['DoubleNum']'

ArithOp→ '+' | '-' | '*' | '/' | '^'

LogicOp→ '&&' | '||'

RelationOp→ '>'|'<'| '>=' | '<=' | '==' | '!='

其中,Condition 为条件,RelationExpr 为关系表达式,Expr 为表达式,TMP1、TMP2 为化简后产生的中间符号,LogicOp 为逻辑算符,ArithOp 为算术算符,RelationOp 为关系算符,variable 为变量,value 为值,IntNum 为整型值,DoubleNum 为双精度值,最右边的符号代表后文该规则对应的类名.

可以证明,这是个 LL(1)文法,即分析的时候从左向右扫描输入串,使用最左推导,并且只需向右看一个符号就可以决定如何推导,即选择哪一个产生式进行推导. 为了简单起见,把对于条件(Condition)的表达(即结果是 boolean 量的)用"{}"包围,以与普通表达式区分(Expr).

1.2 算符优先法[2]

表达式解析器的设计是栈应用的一个典型例子. 本系统采用的是一种简单直观、广为使用的算法——算符优先法.

为了实现"算符优先法",使用两个工作栈,一个称作 OPTR,用以寄存运算符;另一个称作 OPND,用以寄存操作数或运算结果. 算法的基本思想如下.

a. 首先置操作数栈占 OPND 为空栈,表达式的起始符 "#" 为运算符栈 OPTR 的栈底元素.

b. 依次读入表达式中的每个字符,若是操作数则进 OPND 栈;若是运算符,则和 OPTR 栈的栈顶运算符比较优先权后作相应操作,直到整个表达式求值完毕(即 OPTR 栈中的栈顶元素和当前读入的符号都是 "#").

在现有的文献中,一般仅仅给出用于四则混合计算所用到的算符优先表. 本系统因为还要计算逻辑表达式和关系表达式,所以给出本系统所需要的更为全面的算符优先表,见图 1. 图中,$\alpha>\beta$: α 的优先级高于 β; $\alpha=\beta$: α 的优先级和 β 相同;

α \ β	+	-	*	/	^	>	<	>=	<=	==	!=	~	&&	||	(	)	#
+	>	>	<	<	<	>	>	>	>	>	>	E	E	E	<	>	>
-	>	>	<	<	<	>	>	>	>	>	>	E	E	E	<	>	>
*	>	>	>	>	<	>	>	>	>	>	>	E	E	E	<	>	>
/	>	>	>	>	<	>	>	>	>	>	>	E	E	E	<	>	>
^	>	>	>	>	<	>	>	>	>	>	>	E	E	E	<	>	>
>	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
<	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
>=	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
<=	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
==	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
!=	<	<	<	<	<	E	E	E	E	E	E	>	>	>	<	>	>
~	E	E	E	E	E	<	<	<	<	<	<	<	>	>	<	>	>
&&	E	E	E	E	E	<	<	<	<	<	<	<	>	>	<	>	>
||	E	E	E	E	E	<	<	<	<	<	<	<	<	>	<	>	>
(	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	>
)	>	>	>	>	>	>	>	>	>	>	>	>	>	>	E	>	>
#	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	E	=

图 1 算符优先表

Fig.1 Operator precedence table

$\alpha < \beta$: α 的优先级小于β; $\alpha \doteq \beta$: 这种次序出现是非法的,这时候可以报错或抛出异常.

2 实 现

实现可以从 3 个方面描述: a. 编程接口; b. 内部实现; c. 维护界面.

2.1 编程接口

编程接口的工作过程如图 2 所示.

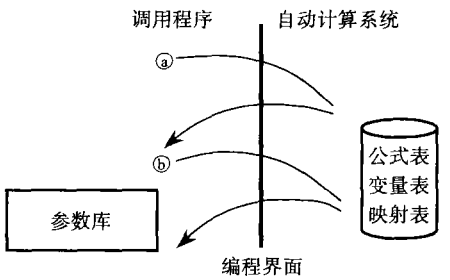

图 2　编程接口的工作过程

Fig.2　Work procedure of programming interface

a. 调用程序通过接口向自动计算系统查询计算某个公式所需要的参数名称,以及为了计算出此参数需要在参数库中调用的方法标识. (在下文中,名词词性的"标识"和 ID 或 id 为同一概念).

其中,参数库是本系统特别设计的模块,属于本系统的一部分,但由调用方的程序员维护. 正是由于参数库的存在,才实现了最大的灵活性. 参数库中存放了获取不同参数值的方法,通常是一个数据库读取语句,它们由方法 ID 区别. 当通过本系统提供的维护界面维护变量时,用户需要指定这个变量是否是需要调用程序传递的参数. 如果是,需要同时指定调用程序为了获得参数的值而需要调用的参数库中的方法的 ID.

在调用程序获得了参数名及计算其值的方法 ID 时,可以很方便地从参数库中取出参数值. 由于参数库极其简单,所以很容易维护.

b. 调用程序传入指定公式所需要的参数,求得结果. 结果用 Object 类型表示,用户可以将其转化为自己需要的类型.

用 Java 语言定义的接口表示如下:

```
public interface AutoCacul
{
    public java.util.Map argsList(int aID,int bID);
    public Object caculate(int aID,int bID,java.util.Map args);
}
```

2.2 内部实现

内部实现要解决的问题有数据库的设计、与数据库的连接、公式在计算机内部的表达、编程实现语法树和算符优先法等. 选择了 Java 语言和 Oracle 8.1.7 来实现.

a. 数据库的设计见表 1 表 2 和表 3(见下页).

表 1 中,维护了变量的详细信息,每一个变量都有一个惟一的 v_id 标识. 要求所有在公式中用到的变量都需要预先在变量表中定义.

表 2 中,保存自动计算系统中的所有公式,公式由 v_id 标识. v_id 的存在是用在维护界面中,单纯的在计算中并没有用到.

表 3 中,维护业务类型、代理商到公式 id 的映射,业务类型用 b_id 表示,代理商用 a_id 表示. 在具体的应用中,它们的数据类型可根据实际需要变化.

表 1　变量表

Tab.1　Variable table

键	名称	数据类型	长度	可否为空	描　述
主键	v_id	Int	4	N	作为主键的整型标识符
	v_name	Char	32	N	易记,可读的标识符
	v_sys	boolean		N	表示此变量是否由调用者传过来
	v_desc	Varchar2	100	Y	对此变量的详细描述
	m_id	Int	4	N	计算方法 id,如果是系统变量则为-1

表 2　公式表

Tab.2　Formula table

键	名称	数据类型	长度	可否为空	描　述
主键	f_id	Int	4	N	作为主键的整型标识符
	v_id	Int	4	N	目标变量在表 1 中的 id,表示此公式是表达那个变量的值(object variable)
	f_expr	Varchar2	4 000	N	用字符串表示的该公式的表达,具体表达办法详细设计文档
	f_desc	Varchar2	100	Y	对此公式的详细描述

表3 业务类型代理商到公式id对照表

Tab.3 Map from business and agent to formula

键	名称	数据类型	长度	可否为空	描 述
主键	b_id	Int	4	N	作为主键的整形标识符业务类型id
主键	a_id	Int	4	N	作为主键的整形标识符代理商id
	f_id	Int	4	N	计算公式

在调用程序接口的时候,自动计算系统先从表3中查找需要计算的公式标识,根据此标识从表2中查到相应公式,然后或者扫描公式,找到需要传递的参数,从表1中找到其方法id;或者根据传递过来的参数,计算出结果,返回给调用程序.

b. 数据库的连接

采用JDBC(Java DataBase Conectivity)技术连接数据库,在数据库的连接上,采取了一些提高性能的做法.

使用JDBC非常简单,通常涉及4个步骤,具体取决于所进行任务的本性: a. 为你使用的DBMS加载一个JDBC驱动程序; b. 使用此驱动打开一个到某个特定数据库的连接; c. 使用这个连接发布SQL语句; d. 处理由SQL操作所返回的结果集.

连接数据库是耗时的操作,为了避免频繁连接到数据库,采用了连接池技术[3]. 创建一个带有Connection对象列表的ConnectionPool对象. 当被请求连接时,连接池找到一个可以使用的连接,把它标记为正在使用,将其传递给请求者;请求者使用这个连接访问被请求的数据,然后将这个连接交给连接池回收. 连接池定期检查是否有任何连接不合法、太陈旧或者过分使用了,并根据需要用新的连接去替代它们. 自动计算系统中,在池中一般维护一个连接. 更好的连接池可以动态根据资源的需求量调整池中连接数量.

由于程序要频繁读取数据库的记录,而且在一段时间内访问比较集中,笔者还设计了本地的缓冲,用于缓存最近访问的一些记录,大大提高了性能. 其代码如下.

```
public class Cache
{
  //容量
  private int capacity=20;
  //存放对象的地方
  private LinkedList repository=new
LinkedList();
  //用于存放对象进来的顺序,以关键字为内容
  private LinkedList list=new LinkedList();
  public Cache(int capacity) {
    this.capacity=capacity;
  }
  public void put(Object key,Object value) {
    if(this.isKeyExist(key))
      return;
    while(list.size()>=capacity) {
      repository.removeFirst();
      list.removeFirst();
    }
    list.addLast(key);
    repository.addLast(value);
  }
  public boolean isKeyExist(Object key) {
    return list.contains(key);
  }
  public Object get(Object key) {
    return repository.get(list.indexOf(key));
  }
}
```

c. 公式在计算机内部的表达

公式表中的f_expr字段存放了公式的计算机内部表示. f_expr 的结构为[表达式1 @ 条件1],[表达式2@条件2],…, [表达式 *n*@条件 *n*]. 其中任何一个“@条件”都可以省去,省去则默认这个条件始终是“真”.

其中的变量表达为如下形式: [v_id, f_id | -1]. v_id 是该变量在表1中的标识,通过此标识可以找到这个变量的变量名v_name及其描述,如果是需要由调用程序传递的参数,还可以知道其方法标识. f_id记录了要把此变量值求出需要使用表2中的哪个公式,当f_id为-1代替的时候,表明此变量是系统传递的参数. 通过这种表达方式,方便地实现了公式之间的嵌套,并且可以方便地将公式转化为用户易理解的形式通过维护界面展示.

d. 语法树的实现

语法树采用设计模式中的解释器(INTERPRETER)

模式实现[4]. 其类图如图 3 所示.

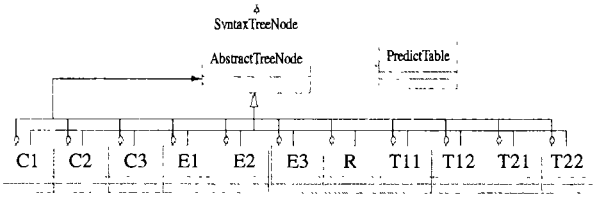

图 3　实现语法树的类图

Fig.3　Class diagram that implements the syntax tree

用解释器模式实用类来表示每一条文法规则，文法规则右边的符号是这些类的实例变量. 类名和产生式的对应关系见本文中对文法的描述.

SyntaxTreeNode是定义的抽象接口，定义如下.

```
public interface SyntaxTreeNode
{
  //添加一个终结符
  public boolean appendTail(String tail);
  //删去最后一个终结符
  public String delTail();
  //判断当前是否允许添加指定类型的终结符
  public boolean isTypeAllowed(int type);
}
```

AbstractTreeNode 是最终类(C1, C2, C3,…, T22)的绝大部分功能的实现，最终类只需要简单地定义产生式即可.

用解释器模式有如下特点：a. 易于改变和扩展文法，也易于实现文法；b. 复杂的文法难以维护；c. 增加了新的解释表达式的方式. 因此，在本系统的条件下，比较适合用解释器模式实现.

e. 算符优先法的实现

算符优先法的原理本文已经叙述，这里给出算法的伪代码：

```
Stack optr=new Stack();
Stack opnd=new Stack(); //两个工作堆栈
/*给出一个符号，如果是常量变量，
*将其值压入操作数堆栈；如果是操作符，
*比较优先级后，作相应处理*/
public void push(String t)
{
  int type=getType(t);
  if(type is VALUE)
    opnd.push(getValue(t));
  else if(type is VARIABLE)
    opnd.push(getVarValue(t));
  else {
    switch(getPriority(optr.peek(),t))
    {
      //栈顶优先级低，直接压栈
      case LT:
        optr.push(t);
        break;
      //优先级相同，直接退栈
      case EQ:
        optr.pop();
        break;
      //栈顶优先级高，用栈顶的算符对
      //操作数栈顶数据操作后，递归调用
      case GT:
        caculateStackTop();
        push(t);
        break;
      default:
        break;
    }
  }
/*主函数，假设表达式已经过词法分
  *析，由一个 Iterator it 接口输出*/
  public Object getExprValue()
  {
    push("#");
    while(it.hasNext())
      push((String)it.next());
    push("#");
    return opnd.peek();
  }
```

2.3　维护界面

采用 C/S 结构实现自动计算系统的维护界面.

单机计算模式已经远远不能适应当前网络发展的需要，更不能满足企业级的计算需求. 当前流行的模式是C/S和B/S结构，也被称为两层计算模式或三层/多层计算模式. 两层计算模式对于单机计算模式是一大进步，但是也有很大的缺点. 由于业务逻辑放在客户端软件中，每当修改业务逻辑的时候，要对所有的客户端重新部署，造成了维护费用高昂；三层计算模式是瘦客户端计算模式，即客户端只需有浏览器就能够运行，而相关的业务逻辑被放到了独立的应用服务器中，这样大

大降低了维护的费用. 由于应用服务器的重要性,服务器产品领域成为软件厂家争夺的焦点,目前主要有两大阵营: 使用自己独有技术的微软和使用 Java 技术的 IBM, BEA, Sun 公司等.

本系统不需要多个客户端,不存在 C/S 模式的缺点,所以本系统采用C/S模式实现. 为了实现界面操作的健壮性而采用的一些技术如下.

a. 避免不合法的表达式

由于用户在编辑公式的时候,笔者在计算机内部为其生成了语法树,所以能够预测用户下一个允许输入的符号. 回顾本文定义的语法树结点的接口 isTypeAllowed(int type)方法,即返回当前是否允许给定类型的终结符,从而可以有选择地禁用一些按钮,以避免用户的误操作.

b. 避免公式的循环依赖

如果为了计算 A 公式必须先计算 B 公式,称 A 依赖于 B. 在较复杂的公式中,会产生循环依赖,例如A 公式依赖于B、C,B 依赖于D、E,而E 又依赖于 A. 这种循环依赖是不允许出现的,也不应该出现,但由于用户的粗心或其他因素,有可能发生. 所以,为了从易用性和健壮性角度出发,要将这种情况检测出来,并发出警告.

这个问题可以很方便地用图论的原理解决,即建模成在有向图中检测环的问题. 这个问题通常用拓扑排序算法[2]完成,用 Java 语言描述如下.

```
boolean hasCycle(){
  //计算有向图每一个顶点的入度
  findInDegree();
  //移走入度为 0 的顶点,
  //并重新计算各顶点的入度,
  //直到没有入度为 0 的顶点
  while(rmVertexWith0Degree()!=0)
    findInDegree();
  return !vertexes.isEmpty();
  }
```

注: 方法rmVertexWith0Degree返回移走顶点个数

3 结 论

通过对自动计算系统的设计,用户可以自行定义需要计算的公式,在一些计算公式需要经常变化的场合,具有很大的灵活性和通用性.

本系统虽然可以基本满足普通的自动计算的需要,但是还有一些可以改进的地方,如界面可以更加人性化,增加其他算符使本系统的功能更加强大等等. 根据本文的思想,读者不难方便地设计出适合自身需要的计算系统.

参考文献:

[1] 吕映芝, 张素琴, 蒋维杜. 编译原理[M]. 北京: 清华大学出版社, 1998.

[2] 严蔚敏, 吴伟民. 数据结构(C 语言版)[M]. 北京: 清华大学出版社, 1992.

[3] [美] Hanna P. 即时应用 *Java Servlets*[M]. 潇湘工作室译. 北京: 人民邮电出版社, 1998, 78~103.

[4] [美] Gamma E, Helm R, Johnson R 等. 设计模式——可复用面向对象软件的基础[M]. 李英军, 马晓星, 蔡敏等译. 北京: 机械工业出版社, 2000, 162~171.

[5] 林勇, 李雪, 黄永宣. 一款表达式解析器在电力统计系统中的应用[J]. 计算机应用, 2002, **22**(8): 103~104.