公式系统高级篇

一、简介

易得公式系统在兼容传统公式编写语言的基础上进行了大量的扩展和优化，并且引入了一些革命性的新特性，为编写复杂的程序化策略公式提供了强有力的工具。

易得公式系统兼容传统公式厂家的语法，并兼容了飞狐、通达信、分析家、金字塔的 95%以上的函数，在此基础上还支持自定义函数、DLL 调用。在易得公式平台上，您可以轻松将您的交易思路转换为交易策略，并由计算机来高效执行。

易得公式系统具有速度快、准确性高、占用资源少，简洁易用的特点，是您编写公式策略的绝佳助手。

易得公式系统提供了公式编译执行、编译优化、自定义函数、多种运算模式、多维数组等一系列新特性，为用户带来了全新的体验，下面我们赶紧来学习吧。

二、编译执行

在计算机编程语言中，有两种执行方式：解释执行、编译执行。两种执行方式的区别如下:

解释执行是程序的执行过程中，将源代码当场进行解释，每解释一条语句后就立即将此语句分解成一条或几条指令并提交计算机立即执行且将执行结果返回。并不生成目标程序，所以每次执行程序都必须依赖于源代码。因为每次执行都需要进行解释，所以运行效率低下。这类语言一般为一些脚本语言，比如网页程序、JavaScript、Python 等语言。

编译执行是在程序编译的时候一次性编译成可执行的目标程序，产生出机器语言的目标程序，然后再让计算机去执行这个目标可执行程序，得到计算结果。在执行过程中不再依赖于源代码，而是快速地执行目标程序，所以其运算效率高。这类语言一般为静态语言，比如 C/C++等。

易得公式系统采用了编译执行方式，在编写好公式后一次性将代码编译成目标可执

行指令。在执行中快速执行目标可执行指令，不再依赖于公式源代码，无需每次根据源代码进行解释，极大地提高了运行效率。

同时，易得因执行公式不再依赖于源代码，所以能做到公式的执行跟源代码分离，在公式编辑器中支持不保持公式源码到系统，公式源码可由用户自行保管。这也是最好的公式保密方式，公式导出可以只导出可执行代码，而不导出源代码。就跟您使用 Office 等软件，您根本就不需要软件的源代码，只需要运行 exe 程序一样。在众证券软件苦苦为公式加密绞尽脑汁的时候，易得通过技术的创新带来了理念的升华，完美地解决了公式被破解的难题。

如果您需要将公式代码代码分开保存，在公式编辑去中去掉勾选“保存源码到公式”，系统就会提示您自动保存到文本文件中。

值得一提的是，易得公式系统是市面上唯一支持编译执行的软件。

三、运算模式

易得公式系统支持三种运算模式：序列运算模式、逐 K 线运算模式和自动运算模式。其中自动模式是系统在公式编译的过程中对用户编写的公式内容进行分析，自动为用户选择序列模式或者逐 K 线模式，自动运算模式也是系统的推荐模式。

其中在逐 K 线模式下，系统支持盘中仅计算最后一根 K 线，从而提升公式在盘中即时行情的运行效率。

下面我们来介绍一下序列模式和逐 K 线模式的运行机理，只有掌握了其内部运行机理，才能在编写公式的过程中运用自如，一切了然于心。

1. 运算模式的含义

序列运算模式在公式执行时按公式源代码顺序执行，并仅执行一次。在执行每个语句的时候对所有时间序列数据(K 线数据)进行循环计算。

逐 K 线模式对所有时间序列数据(K 线数据)分别执行一次公式的所有源代码，也就是说从第一个周期开始逐个周期执行一次公式，每个周期都会执行公式一遍。

```
V0:=MA(CLOSE,5);
V1:=MA(CLOSE,10);
```

假设在 100 个周期的日线数据上执行如上公式。

在序列模式下，先执行第一条语句计算 100 个周期内每天的 5 日均线值，执行完该

语句后输出结果 V0 中已经包括 100 个周期中每天的 5 日均线值。然后执行第二条语句，计算 100 个周期内每天的 10 日均线值，执行完该语句后输出结果 V1 中已经包括 100 个周期中每天的 10 日均线值。所以公式只被执行了一次，就计算出了 100 天内所有的结果值。

在逐 K 线模式下，先执行整个公式一次计算第一天的 V0 和 V1，然后再执行整个公式一次算第二天的 V0 和 V1，以此类推，重复 100 遍计算完 100 天的 V0 和 V1。所以公式被执行了 100 次，但每次只计算其中 1 天的值。

读到此处，您已经明白两种模式的含义。但聪明的您一定会想：两种模式谁执行起来更快？为什么要有两种模式呢？他们的使用场景又是什么呢？让我们接着学习吧。

2. 运算模式的效率

前面我们已经了解到计算 100 个周期的日线数据，序列运算模式下只需要执行 1 次公式，而逐 K 线模式下需要执行 100 次公式。从这个角度来说，是不是序列模式就比逐 K 线模式更快呢？答案是否定的，因为在序列模式下每条语句的执行都隐含着一个 100 次的循环（计算 100 天每天的值）。所以其实两者的总计算量是一样的，无非是横着来还是竖着来的问题。

那么逐 K 线模式需要执行 100 次，有没有其他的执行开销需要花费更多时间呢？如果每次执行公式都需要将公式源代码解释编译一遍，那么就比序列模式的多了 99 次的代码解释过程，这样的确会慢很多。但幸运的是，我们前面的章节已经讲过，易得的公式系统是采用编译执行技术，在编写好公式后系统已经一次性翻译成了机器可执行语言，后续无论执行多少次都无需再次解释编译。

所以综合来说，两者的总体运行效率是相当的。但在逐 K 线模式下，有个好处是系统支持盘中仅计算最后一根 K 线。也就是说在公式加载的时候进行了一次全部数据的运算，在后续盘中即时行情不断跳动的情况下，逐 K 线模式可以在内存中记录下之前计算的历史数据结果并保持不变，只计算最后一根 K 线的变化值，从而大大地提升了盘中行情的刷新计算效率。

而序列模式因为每条语句执行都需要从第一个周期算到最后一个周期，所以不能支持盘中仅计算最后一根 K 线的功能。

结论：如果是一次性计算所有历史数据的结果(比如回测、选股)，两者的效率相当。但在盘中即时行情刷新计算时，如果逐 K 线模式启用了盘中仅计算最后一根 K 线的功能，那么盘中即时行情的刷新计算会比序列模式快得多。

3. 运算模式的特点

逐 K 线模式执行语法更加灵活，因为它是在每个数据周期上分别执行的，所以公式能根据每个周期的不同情况作出不同的操作选择。比如分支 IF 语句，在每个 K 线周期根据不同给的条件进入不同程序分支执行，去执行不同开仓、减仓等操作。而序列模式只能判断到最后一个周期的条件值。

而序列模式在某些使用未来函数的指标统计场景下，却拥有逐 K 线模式下无法做到的优势。比如 CONST(X)，需要取最后一个周期的 X 值作为常数，逐 K 线模式是从第一根 K 线逐根执行的，在第一根 K 线计算的时候，最后一根 K 线的值还没算出来，肯定是无法取到的。再比如 REFX(X,N)向后引用，需要取当前周期后面 N 周期的值，而在执行当前周期的时候后面的 N 周期的值还没计算出来呢。所以这类引用未来结果的未来函数的逻辑与逐根 K 线模式的运行机理是相矛盾的，只能在序列模式下使用。

4. 运算模式的选择

了解了各个模式的特点，对于选择运算模式就心中有数了。基本原则如下：

1）分支 IF ELSE、循环 FOR WHILE 等语句只适合在逐 K 线模式下使用，不适合在序列模式。

2）未来函数只适合在序列模式下，这类函数包括：

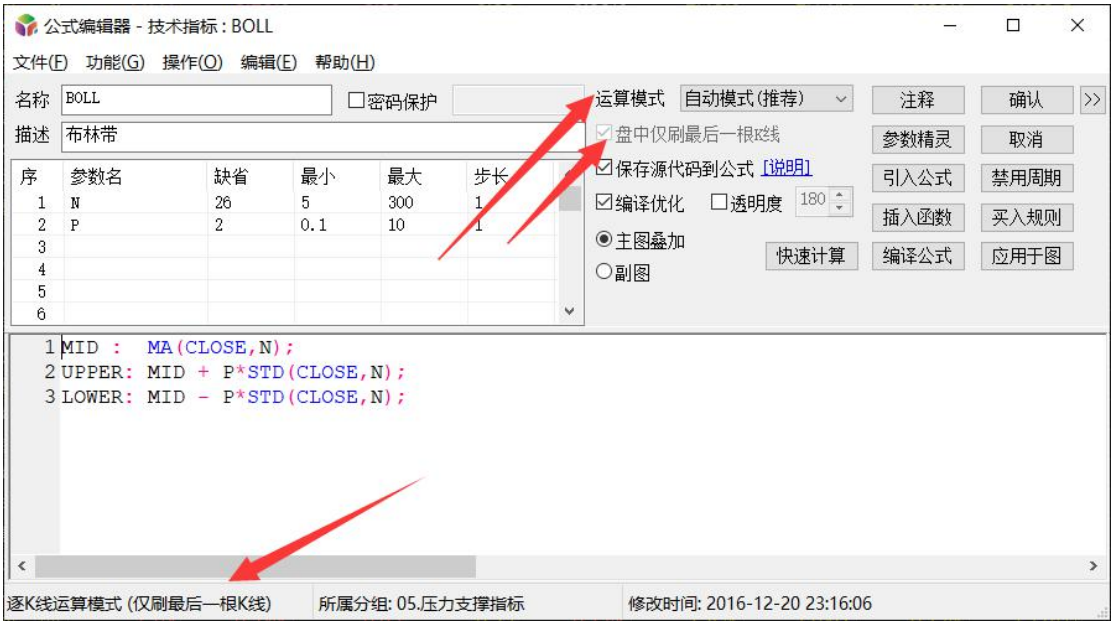
CONST REFX REFXV FILTERX BARSNEXT ALIGNRIGHT BACKSET BACK1SET ZIG FLATZIG ZIGA PEAK PEAKBARS TROUGH TROUGHBARS，

以及 STKINDI2 等小周期引用大周期的跨周期引用函数

3）下列函数在逐 K 线模式下不适合使用仅计算最后一根 K 线功能：

SPLITPX SPLITP SPLITV

这么多原则用起来岂不是很麻烦？不会的，易得提供了自动运算模式(推荐模式)，会根据上面的原则自动为您选择适合的运算模式，在公式编辑器的左下角状态栏会显示您公式所处的运行模式。如果违背了上述原则，在公式编译的时候会给出编译告警信息，当然您可以通过公式编辑器的设置，将此类告警当做编译错误而无法通过编译。在自动运算模式下，如果对于两种模式都符合的公式，自动模式会默认选择逐 K 线模式并仅刷最后一根 K 线。



那如果我的公式想使用 IF 分支或者循环语句，又要用到未来函数，那怎么办呢？如果您理解了其运行机理，您就会知道，函数的逻辑跟模式的运行机理相矛盾，那么您的公式就充满了不确定性，也就是：无解。

或许在阅读本章的开始，您就会想为什么要搞运算模式这么复杂？读到此处，对于这个问题您心中应该已经有了答案。因为有些函数必须在序列模式下才能实现，而有些也只能在逐 K 线模式下才能正确执行。作为最专业的证券软件，当然要给予用户最大的选择权了。

市面一些流行证券软件的的运算模式：
通达信、老飞狐、分析家 V5.0 之前、博弈大师等是使用序列模式；
分析家 V5.0 之后、大智慧、TB 使用逐 K 线模式；
易得既支持序列模式、也支持逐 K 线模式。

四、编译优化

编译优化是在不改变程序运行效果的前提下，对被编译的程序进行等价变换，使之能生成更加高效的目标代码。

易得公式系统的编译优化在公共表达式消除、无用代码消除等方面做出了努力。让我们来举个例子说明吧。

我们非常熟悉的乖离率 BIAS 指标代码如下：

```
BIAS1 : (CLOSE-MA(CLOSE,L1))/MA(CLOSE,L1)*100;
BIAS2 : (CLOSE-MA(CLOSE,L2))/MA(CLOSE,L2)*100;
BIAS3 : (CLOSE-MA(CLOSE,L3))/MA(CLOSE,L3)*100;
```

我们看到第一行代码 BIAS1 的计算方法，前后有 2 个 MA(CLOSE,L1)，都是算 L1 日的均线，其参数和计算函数都是一样的，运算结果自然也是一样的。如按照代码原汁原味来执行，那么意味着要执行计算两次一样的 MA(CLOSE,L1)。 易得公式系统的编译优化会非常聪明地检查出两者的相同，在需要第二次计算 MA(CLOSE,L1)的时候，将前面第一次计算的结果取出直接使用，无需重复计算，从而大大地提升了公式执行效率。该公式的第二行、第三行都出现 MA(CLOSE,L2)等此类相同的公共表达式，也可以一并进行消除。

再比如，如果公式中出现一些中间的计算变量，而后续的公式逻辑根本就没引用这个中间变量。这些都可以作为无用代码消除，跳过这些代码的执行，从而提升运行公式执行效率。

当然，编译优化是一门非常高深的计算机语言研究课题，易得公式系统也是浅尝辄止，在一定条件下尽量帮助用户提升公式的执行效率。同时，用户可以尽量提升自己的公式编写水平，写出更高效简洁的代码，减少无用代码、减少重复调用，而非完全依赖于系统的编译优化功能。

五、自定义函数

开发公式的朋友可能经常会遇到这样一个问题：有个很好的策略想法，但编写的过程发现缺少某些函数，自己的策略思路表达不出来，向软件开发商提出需求却又难以得到回应。

易得公式系统的自定义函数就是为了解决此类问题应运而生，用户可以将自己的想法写成一个函数，在公式中可以自由调用，也可以导出共享给其他人。

自定义函数的推出，使得易得公式系统的可编程能力有了质的飞越，意味着易得公式系统向高级编程语言看齐迈出了实质性的一步。

1. 什么是函数

计算机语言中的函数是一段可以重复使用的代码，用来独立地完成某个功能，它可以接收用户传递的数据，也可以不接收。接收用户数据的函数在定义时要指明参数，不接收用户数据的不需要指明，根据这一点可以将函数分为有参函数和无参函数。

函数可以有个返回值，将计算结果返回给调用者。

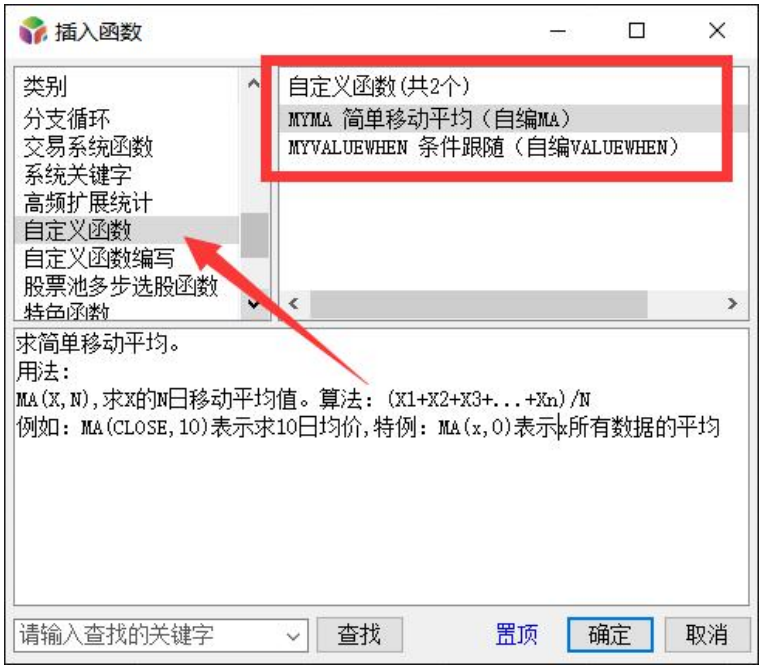
2. 自定义函数的类型

易得公式系统的自定义函数有两种类型：全局自定义函数和局部自定义函数。两者的语法都是一样的，区别在于定义的位置和使用的作用域不同。

1）全局自定义函数

在易得公式系统的“自定义函数”分类下，其跟技术指标、五彩 K 线、交易系统、条件选股等分类是平级的。在这个分类下创建的的自定义函数，可以供任何外部公式调用，但其自身不具备有执行能力，它只是个被调用者的角色。

在开发公式时，编写好的全局自定义函数，还可以在插入函数对话框的自定义函数分类中找出。



2) 局部自定义函数

局部自定义函数的不同之处是它的定义可以出现在任何公式的里面，在本公式出现的自定义函数，只可以在本公式内调用，不能被其他公式调用。当然，局部自定义函数的定义必须出现被调用的前面。

如下例，在技术指标公式中，定义一个 MyFunc 局部自定义函数，并在本公式内调用输出 Mul 的结果值。

```
function MyFunc(x,n)
begin
    a:=x*n;
    return a;
End

Mul:MyFunc(c,5);
```

3. 自定义函数的语法

关键字：function

用来声明定义一个自定义函数

```
function AAAA(X, N)   // 定义名为 AAA 的自定义函数，参数 X、N。
function BBB          // 定义名为 BB 的自定义函数，无参数。
```

关键字：return

自定义函数返回，只能用在自定义函数体中。

语法：

```
return var;     // 自定义函数立即返回，带返回值 var。后面语句不会执行。
return;         // 自定义函数立即返回，不带返回值。后面语句不会执行。
```

1) 有参数自定义函数

```
function AAAA(X, N)  // 定义名为 AAA 的自定义函数，参数 X、N。
begin                // 自定义函数必须以 BEGIN 开始
    res := X * N;
    return res;      // 自定义函数返回，带返回值 res.
end                  // 自定义函数必须以 END 结尾
```

```
A:AAA(c, 5);              // 自定义函数的调用
```

2）无参数自定义函数

```
function BBB              // 定义名为 BB 的自定义函数，没有参数。
begin                     // 自定义函数必须以 BEGIN 开始
    res := c * 5;
    return res;           // 自定义函数返回，带返回值 res.
end                       // 自定义函数必须以 END 结尾

B:BBB;                    // 自定义函数的调用
```

4. 自定义函数的使用规则

1）函数体内不能显式输出指标线数据.

2）函数体内不能使用外部变量，如果需要使用必须以函数参数的方式传入。但函数体内可以使用公式参数和 VARIABLE 变量。

3）函数的参数以及函数体内的变量，都可以跟外部变量同名.

5. 自定义函数的实例

例子 1：求平均线，等价于系统提供的函数 MA

```
function myma(x, n)
begin
    n:=intpart(n);   // 均线天数取整，防止传入浮点数
    if currcalcbar >= n then
    begin
        sums:=0;
        for i=currcalcbar downto currcalcbar-n+1 do
            sums := sums+x[i];
        sums:=sums/n;
          return sums;
    end
end

a:myma(c,5);   // 调用自定义函数 myma。结果与 b:ma(c,5); 一样
```

例子 2：条件跟随函数，等价于系统提供的函数 VALUEWHEN

```
1. function myvaluewhen(condi, x)
2. begin
3.       result:=invaliddata;
4.
5.       if condi > 0 then
6.             result:=x;
7.       else if currcalcbar > 1 then
8.             result:=result[currcalcbar-1];
9.
10.       return result;
11. end
12.
13. a:myvaluewhen(c>o,1); //  调用 myvaluewhen。结果与
    b:valuewhen(c>o,1);一样
```

通过上面的 2 个实例，可以看到通过易得公式系统可以很容易的编写自己想要的函数。其实，系统自带的统计类函数，几乎都可以通过自定义函数自己来实现。

值得一提的是，易得是市面上唯一能通过公式系统编写自定义函数的软件。

六、分支

如果公式的代码是先执行第一条语句，然后是第二条、第三条…一直到最后一条语句，这种称为顺序结构。但是对于很多情况，顺序结构的代码是远远不够的，我们经常需要判断如果满足某个条件就执行某些指令，否则就执行另外一些指令。这就需要分支了，分支也是计算机编程语言最为重要的功能之一。

1. 分支的基本语法

```
IF  判断条件  THEN
BEGIN
      语句块 1
END
```

```
ELSE
BEGIN
    语句块 2
END
```

意思是，如果判断条件成立，那么执行语句块 1，否则执行语句块 2 。所谓语句块，就是由 BEGIN 和 END 两个关键字包围的一个或多个语句的集合。

如果语句块中只有一个语句，也可以省略 BEGIN 和 END，例如：

```
IF 判断条件 THEN
    语句 1
ELSE
    语句 2
```

举例：

```
IF CLOSE>OPEN THEN
   P := P+1;
ELSE
   P := P-1;
```

它表示如果收盘价大于开盘价，P 的数值就增加 1，否则 P 值就减少 1，也就是说统计历史阳线的数量。

2. 只使用 IF 语句

有的时候，我们需要在满足某种条件时进行一些操作，而不满足条件时就不进行任何操作，这个时候我们可以只使用 if 语句。也就是说，if else 不必同时出现。

举例：

```
IF CLOSE>OPEN THEN
    P := P+1;
```

它表示如果收盘价大于开盘价，P 的数值就增加 1，否则 P 值维持不变，也就是说统计历史阳线的数量。

3. 多个 IF ELSE 语句

IF ELSE 语句也可以多个同时使用，构成多个分支，形式如下：

```
IF 判断条件 1 THEN
BEGIN
    语句块 1
END
ELSE IF 判断条件 2 THEN
BEGIN
    语句块 2
END
ELSE IF 判断条件 3 THEN
BEGIN
    语句块 3
END
ELSE
BEGIN
    语句块 4
END
```

意思是，从上到下依次检测判断条件，当某个判断条件成立时，则执行其对应的语句块，然后跳到整个 if else 语句之外继续执行其他代码。如果所有判断条件都不成立，则执行语句块 n，然后继续执行后续代码。也就是说，一旦遇到能够成立的判断条件，则不再执行其他的语句块，所以最终只能有一个语句块被执行。

4. IF 语句的嵌套

IF 语句的嵌套，需要注意 BEGIN 和 END 的配对使用。基本的原则是 END 与之前面最接近的还没有配对的 BEGIN 进行配对。

例如：

```
IF CLOSE>OPEN THEN
BEGIN (1)
    P := P+1;
```

```
    IF CLOSE>10 THEN
    BEGIN (2)
        P := P+1;
        Q := MA(CLOSE,10);
    END (3)
END (4)
```

其中，2-3 是配对的，1-4 是配对的。

5. 分支举例

例 1：计算上市以来上涨天数和下跌天数的比率：

```
VARIABLE: UP=0, DN=0;
IF CLOSE>CLOSE[BARPOS-1]THEN
    UP := UP+1;
ELSE IF CLOSE< CLOSE[BARPOS-1] THEN
    DN := DN+1;
RATIO: IF(DN=0,0,UP/DN);
```

最后一条语句，判断 DN 是否为 0，用来保护结果不被 0 除。

例 2：计算历史上阳线的平均涨幅和阴线的平均跌幅

```
1. VARIABLE:UPR=0,UP=0,DNR=0,DN=0;
2. R := CLOSE/REF(CLOSE,1)-1;
3. IF CLOSE>OPEN AND BARPOS > 1 THEN
4. BEGIN
5.       UPR := UPR + R;
6.       UP := UP+1;
7. END
8. ELSE IF CLOSE< OPEN AND BARPOS > 1    THEN
9. BEGIN
10.       DNR := DNR + R;
11.       DN := DN+1;
12. END
```

```
13. 平均涨幅: IF(UP=0,0,100*UPR/UP);
14. 平均跌幅: IF(DN=0,0,100*DNR/DN);
```

6. IF 分支语句与 IF 函数的区别

我们知道，系统有提供 IF 函数，其原型为：IF(X,A,B)若 X 不为 0 则返回 A，否则返回 B。

IF 分支语句和 IF 函数使用同样的关键字，区分它们的办法是在 IF 语句之后必然存在 THEN 语句，而 IF 函数则没有。

七、循环

所谓循环，就是重复地执行同一段代码，例如要计算 1+2+3+……+99+100 的值，就要重复进行 99 次加法运算。

易得公式系统提供了两种循环语句，分别是 WHILE 循环和 FOR 循环。

1. WHILE 循环

1) 语法

```
WHILE [条件] DO [语句]
```

它表示，如果条件成立则循环执行语句，直到条件不成立为止。

2) 举例

例如我们计算最近多少天完成 100%换手：

```
HR := VOL;
ND := 0;
WHILE HR< CAPITAL/100 DO
BEGIN
    ND := ND+1;
    HR := HR +REF(VOL, ND);
END
RES:ND;
```

ND 就是结果。HR 表示最近成交量累加，设初始值为当日成交量，然后循环直到它大于流通盘为止。循环体中，ND 每次循环加 1，HR 每次循环加上 ND 天前的成交量，也就是说最近 ND 天的成交量累加。

在循环中必须注意的是，循环条件在循环过程中一定要发生变化，并且会变成条件不成立，否则会形成死循环，也就是说循环条件永远成立，计算机不断地进行循环计算。

在上例中，HR 每次递增，当它增大到流通盘以上时，条件变成不成立，从而终止循环。另外一个我们没有注意到的问题是，如果今天是上市第一天，而且换手率没有达到 100，则这个循环会出现问题，因为不论 ND 怎样增大，REF(VOL, ND)总是返回没有数值，也就是说 HR 的不到递增，也就永远无法破坏循环条件而终止循环，它也是一个死循环。因此我们需要改成：

```
HR := VOL;
ND := 1;
WHILE HR < CAPITAL/100 AND ND < BARPOS DO
BEGIN
    HR := HR + REF(VOL, ND);
    ND := ND+1;
END
RES:ND;
```

增加一个 ND < BARPOS 用以阻止超过上市日的向前引用。

从这些例子中我们看到，自己使用循环来实现算法，其功能是强大的，但是需要十分小心，避免死循环的发生。因此，我们能够使用函数来实现的功能，还是尽量使用函数来实现，避免不必要的复杂性。

2. FOR 循环

FOR 循环是指定循环次数的循环，在我们证券计算也大量使用向前引用若干天的数据，所以 FOR 循环非常实用。

1）语法

```
FOR [变量]=[初值] TO [终值] DO [语句]
```

它表示使用变量来控制执行循环语句，首先给变量赋初值，然后判断变量是否小于或等于终值，若满足条件则执行语句，然后将变量加 1，循环判断变量是否小于等于终值并循环执行，直到条件不满足为止。例如：

```
FOR I=1 TO N DO...
表示循环 N 次，循环变量从 1 到 N，类似的

FOR I=0 TO N-1 DO...
也表示循环 N 次，但是循环变量从 0 到 N-1。

循环变量还可以从大循环到小，可以使用
FOR [变量]=[初值] DOWNTO [终值] DO [语句]
此时变量将从大到小变化，直到小于终值为止。
```

我们在使用中需要注意递增还是递减变化，否则将形成死循环。

2) 举例

使用 FOR 循环的一个最大的好处在于其循环次数可以控制，不像 WHILE 循环可能存在潜在的死循环。还以 WHILE 循环中的换手 100%为例：

```
HR := 0;
FOR I=0 TO BARPOS-1 DO
BEGIN
    IF HR< CAPITAL/100 THEN
    BEGIN
        HR := HR+VOL[BARPOS-I];
        IF HR>=CAPITAL/100 THEN
            ND := I+1;
    END
END
RES:ND;
```

我们用 FOR 循环来控制总的循环次数不超过数据总数，从而避免了死循环的

发生。在循环中，如果换手未超过流通盘，则继续累加，当换手刚达到流通盘时，将循环次数赋给结果 ND。

3. 循环的嵌套

我们可以在循环中再套入循环，这就叫做循环嵌套。例如我们想要找到最近 100 天中收盘价相同的天数：

```
ND := 0;
FOR I=0 TO 99 DO
BEGIN
    FOR J=I+1 TO 99 DO
    BEGIN
        IF CLOSE[BARPOS-I]=CLOSE[BARPOS-J] THEN
            ND := ND+1;
    END
END
RES:ND;
```

我们分成内外两个循环，外层循环使用 I 作为循环变量，它从 0 到 99 循环，得到之前每一天的收盘价 CLOSE[BARPOS-I]，而内层循环使用 J 作为循环变量，它 I+1 到 99 循环，表示从第 I+1 天前开始查找等于第 I 天数值的 K 线，若找到（条件 CLOSE[BARPOS-I]=CLOSE[BARPOS-J]满足），则将 ND 加 1。

使用循环嵌套，我们可以做许多事情了。但是使用嵌套一定要注意，不要是循环次数太大，否则运行速度会很慢。

4. BREAK 终止循环

我们看到，循环过程中必须要有一个终止循环的方法，WHILE 语句中使用条件不满足来终止循环，FOR 循环中使用变量递增递减来终止循环，是否还有其它的需要呢？

我们看上面的例子，该循环有一个问题，就是无论是否计算出结果，循环都将继续下去，直到计算到上市第一天，这将大大降低效率。我们通过主动终止循环来解决这个问题：

```
HR := 0;
```

```
FOR I=0 TO BARPOS-1 DO
BEGIN
    HR := HR+VOL[BARPOS-I];
    IF HR>=CAPITAL/100 THEN
    BEGIN
        ND := I+1;
        BREAK;
    END
END
RES:ND;
```

执行 BREAK 语句将终止循环，无论循环中值条件是否达到。在本例中，当计算到结果，就停止循环。一般说来，BREAK 语句总是与 IF 语句配合使用。使用 BREAK 语句可以使公式看起来更加简单。

BREAK 可以用来终止 WHILE 循环和 FOR 循环。

5. CONTINUE 结束本次循环

CONTINUE 语句的作用是跳过循环体中剩余的语句而强制进入下一次循环。CONTINUE 语句只用在 WHILE、FOR 循环中，常与 IF 条件语句一起使用，判断条件是否成立。

我们还以 FOR 循环中的换手 100%为例，这次我们在是使用 CONTINUE 语句在换手未达 100%的时候跳过 ND 赋值语句结束本次循环，立即执行下一次循环。其执行结果与上列中使用 BREAK 语句是相同的。

```
HR := 0;
FOR I=0 TO BARPOS-1 DO
BEGIN
    IF HR< CAPITAL/100 THEN
    BEGIN
        HR := HR+VOL[BARPOS-I];
        IF HR < CAPITAL/100 THEN
            CONTINUE;
        ND := I+1;
```

```
    END
END
RES:ND;
```

6. 循环举例

例 1. 计算 N 日均线

```
SU := 0;
FOR I=0 TO N-1 DO
    SU := SU+REF(CLOSE,I);
RES:SU / MIN(N,BARPOS);
```

该例子中有两个技巧，其一，没有可以去避免向前循环超过上市日的问题，因为发生这样情况时 SU 的数值不会增加，也就是说 SU 等于上市到现在的总和；其二，如果当前位置小于 N，则 SU 的数值表示上市到现在的总和而不是 N 日总和，所以平均价格应该为 SU/BARPOS，因此我们使用了 SU / MIN(N,BARPOS);

八、GOTO 语句

GOTO 语句也称为无条件转移语句，其作用是无条件转向公式内的某一处，公式必须指出转向的目标行，目标行用标号@指明。为防止死循环，不允许向前跳转。

其语法格式为：

```
GOTO  标号;

标号@;
```

例如：

```
IF ISLASTBAR THEN
    GOTO QUITLINE; //  最后一个周期跳过均线计算行
B:MA(C,5);
QUITLINE@;
```

九、多维数据

所谓数组，就是一个容器，它可以存放多个数据，我们可以通过序号来访问这些数据。一般说来我们总是将一些相关的数据组织在一起放到数组中，当我们在使用循环的时候，数组就可以发挥它的优势。易得公式系统目前支持多维数组。

数组在使用之前一定要先声明：

```
VARIABLE: V1[20]=0;
```

表示定义一个数值型一维数组 V，它总共有 20 个元素，这些元素的初始值为 0。

```
VARIABLE: S1[10]=’A’;
```

表示定义一个字符串型一维数组 S，它总共有 10 个元素，这些元素的初始值为’A’。

也可以是任意多维数组，如二维数组:

```
VARIABLE:ar[5][6]=10;
ar[2][3]:=4;
a:ar[2][3];
```

如三维维数组:

```
VARIABLE:ar[5][6][7]=10;
ar[2][3][4]:=4;
a:ar[2][3][4];
```

数组变量声明以后，就可以像普通变量一样使用了。在使用数组变量时，需要在变量名后面带上序号，表示引用数组中的第几个元素，元素的序号从 1 开始。例如：

```
P:= V1[5]*CLOSE;
```

表示 V 的第 5 号元素乘以收盘价。

数组和普通变量有一个重要的区别，数组是不能够引用过去的数值的，对数组进行引用过去数值的操作将会得到它当天的数值，就是说数组只存在当天的数值，从某种意义上来说它更像一个可以重新赋值的常量。因此，

```
REF(V1[3],1);
MA(V1[2],10);
```

等均会返回一个常数。如果你需要引用过去的数值，可以将数组元素赋值给一个普通变量，例如：

```
P:=V1[3];
MA(P,10);
```

使用数组以后，我们可以利用数组的序号来访问数据，这给循环带来了方便，我们通过循环可以遍历整个数组了。

十、使用[]下标快速引用数据

前面的不少例子已经看到，易得公式系统可以使用[]下标快速引用任何序列数据，极大地方便了公式的书写，提升了公式的易读性。

如：

```
CLOSE[BARPOS-1];      // 相当于语句 REF(CLOSE,1);

MA5:MA(C,5);
MA5[BARPOS-5];        // 相当于 REF(MA(C,5), 5)
```

十一、序列变量

我们知道，公式使用的数据类型分为两类：变量和常量。

所谓变量就是一个随着时间变化而变化的数据，例如成交量；常量就是一个永远不变的数据。例如 3，每个函数需要的参数可能是变量也可能是常量，不能随便乱用，函数计算的结果一般是一个变量。

变量有时候在需要引用历史数据时必须是个序列值变量，有时候只需要是个单值变量。比如：

```
MA5:MA(A,5);
```

MA 的第一个参数 A 就必须是个时间序列值，因为计算平均线需要历史的 A 值，只有当根 K 线的 A 值是计算不出来的。

我们可以使用 VARSERIAL 关键字来指定变量必须存储为序列变量。语法如下：

```
varserial: aa, bb;
```

十二、DLL 调用

易得公式系统支持公开的 C++语言编程接口，允许用户通过编写 DLL 扩展更加复

杂的公式逻辑。您可以通过 C++语言接口实现数据库数据存取，实现自己的网络爬虫抓取数据等。

易得公式系统 C++语言接口兼容了分析家/飞狐的使用调用语法，并在数据交互上做了扩展。参数个数提升到 32 个参数，并且支持字符串参数的，使得数据库表面、查询语句等字符串参数能传输到 DLL 模块。

DLL 调用语法：

```
A:"YDFUNC@TESTCLOSEMA"(5);
B:"YDFUNC@TESTMA"(c+o, 5);
DRAWTEXTABS(2,2,"YDFUNC@DEMO"('test'));
```

具体 DLL 编写请参考易得安装目录下自带的 YdFunc 工程的例子程序：**易得公式 DLL 示例.rar**

接口编写注意要点：

1. 如果易得软件是 64 位,DLL 也必须编译成 64 位；如果易得软件是 32 位，DLL 也必须编译成 32 位。
2. 函数名称需全部大写.
3. 对于 C++程序需包括在 extern "C" { } 括号中.
4. 函数入参可以有 32 个，每个入参都可以是单值数值、或者序列数据、或者字符串。
5. 函数数据计算结果由 pData->m_pResultBuf 带回，它是一个序列值。 也可以通过 pData->m_pszResultText 返回一个字符串。
6. 编译好的 dll 必须拷贝到易得客户端的 FmlDll 目录下.
7. 建议使用 Visual C++编程，当前项目编译环境：Visual studio 2010
8. 开发者应该了解易得公式序列运算模式和逐 K 线运算模式的机理。

假设一个公式有 A、B 共 2 个语句，运算的数据有 100 根 K 线。

序列运算模式是：A 语句先对应每根 K 线执行 100 次，全部 K 线执行完成后再 B 语句对应每根 K 线执行 100 次。pData->m_nCurBarPos 标识了当前执行的 K 线位置。

逐 K 线运算模式是：为每根 K 线同时执行 AB 所有语句；先在第一根 K 线上执行 A 语句和 B 语句，然后在第二根 K 线上执行 A 语句和 B 语句，直到第 100 根 K 线。

pData->m_nCurBarPos 标识了当前执行的 K 线位置。

十三、强大灵活的公式编辑器

易得公式系统提供了一个功能强大的公式编辑器，为用户编写公式提供了众多方便和友好的体验。

1. 支持多窗口同时打开编辑器。
2. 支持非模式编辑器，打开编辑器后可以进行主界面操作。
3. 支持函数自动完成功能。
4. 支持参数提示功能。
5. 支持字体颜色自定义。
6. 支持高亮选中变量功能，一眼识别该变量的使用位置。
7. 支持自动缩进排版功能。
8. 支持行号、高亮选中行等高级编辑功能。
9. 未知参数自动识别为参数。
10. 将编译告警当作错误。