

# TRADEBLAZER

## 公式开发指南

深圳开拓者科技有限公司

*V4.2 / 2011.8*

## 目录

●前言.....	- 1 -
●TRADEBLAZER 公式概述.....	- 2 -
●TRADEBLAZER 公式的命名.....	- 3 -
●TRADEBLAZER 公式正文规则.....	- 4 -
●TRADEBLAZER 公式体系架构.....	- 5 -
公式的运算步骤.....	- 5 -
历史数据回溯.....	- 6 -
实时数据的运算.....	- 6 -
●基础模块.....	- 8 -
行情报价.....	- 8 -
分时图表.....	- 8 -
超级图表.....	- 11 -
商品设置.....	- 12 -
●数据.....	- 16 -
数据类型.....	- 16 -
BAR 的索引值.....	- 17 -
BAR 的状态值.....	- 18 -
数据周期.....	- 18 -
BAR 数据.....	- 19 -
行情数据.....	- 20 -
属性数据.....	- 20 -
帐户数据.....	- 21 -
数据叠加.....	- 22 -
数据源.....	- 23 -
●TRADEBLAZER 公式应用的建立.....	- 24 -
如何新建公式应用.....	- 24 -
新建公式简称与名称、注释.....	- 24 -
新建公式内容.....	- 25 -
函数说明列表.....	- 25 -

校验保存公式.....	- 25 -
公式在图表上的应用.....	- 28 -
主/副图上的显示.....	- 29 -
●公式管理器 .....	- 30 -
公式编辑器 .....	- 30 -
公式的属性 .....	- 32 -
公式的加密 .....	- 33 -
公式的导入与导出.....	- 34 -
数据窗口.....	- 35 -
公式的报警 .....	- 36 -
●TRADEBLAZER 公式的语法基础.....	- 37 -
保留字.....	- 37 -
标点符号.....	- 39 -
操作符号.....	- 40 -
数学操作符号 .....	- 40 -
关系操作符号 .....	- 41 -
逻辑关系操作符号.....	- 42 -
系统函数.....	- 44 -
●TRADEBLAZER 公式的语句 .....	- 45 -
声明.....	- 45 -
参数.....	- 45 -
参数的类型 .....	- 46 -
参数的默认值 .....	- 47 -
变量.....	- 47 -
变量的类型 .....	- 48 -
变量的声明 .....	- 49 -
变量的赋值 .....	- 49 -
变量的使用 .....	- 50 -
序列变量.....	- 51 -
全局变量.....	- 53 -
赋值语句.....	- 55 -
控制语句.....	- 56 -

条件语句.....	- 56 -
if... 语句 .....	- 56 -
if...else... 语句 .....	- 57 -
if...else...if... 语句 .....	- 58 -
If-Else 的嵌套 .....	- 59 -
循环语句.....	- 61 -
for 循环.....	- 61 -
while 循环 .....	- 63 -
死循环 .....	- 64 -
Break.....	- 64 -
Continue .....	- 65 -
●TRADEBLAZERE 用户函数.....	- 66 -
用户函数的类型 .....	- 67 -
序列函数.....	- 67 -
使用内建用户函数.....	- 67 -
用户函数的参数 .....	- 68 -
如何编写用户函数.....	- 68 -
用户函数的调用 .....	- 69 -
●技术分析类的公式应用.....	- 71 -
技术分析模板.....	- 71 -
输出函数的具体说明.....	- 71 -
公式正文（技术分析类） .....	- 73 -
输出数据的名称.....	- 76 -
输出颜色的选择.....	- 77 -
与 BAR 同齐的数据输出.....	- 77 -
条件 BAR 下的数据输出.....	- 78 -
偏移 N 个 BAR 的输出 .....	- 79 -
UNPLOT 的使用 .....	- 80 -
参数的调整 .....	- 81 -
●交易策略类公式应用.....	- 83 -
交易策略的基本规则.....	- 83 -
交易指令函数.....	- 83 -

交易策略的讯号设置.....	- 85 -
商品叠加的交易策略.....	- 86 -
数据源的叠加.....	- 86 -
公式语句中对数据源的区分.....	- 87 -
交易策略的实现.....	- 88 -
策略的头寸.....	- 88 -
开仓与平仓.....	- 89 -
加仓与减仓.....	- 90 -
策略交易的辅助功能.....	- 91 -
交易助手的应用.....	- 91 -
调试语句的输出.....	- 93 -
自动交易的设置与实现.....	- 94 -
历史性能测试.....	- 96 -
交易策略参数优化.....	- 98 -
优化结果生成分析图表.....	- 100 -
优化结果的选择.....	- 101 -
<b>●TRADEBLAZER 公式策略进阶 .....</b>	<b>- 103 -</b>
止赢止损.....	- 103 -
跟踪止损.....	- 105 -
加仓减仓.....	- 108 -
多品种交易 .....	- 110 -
集合竞价数据过滤.....	- 111 -
收盘平仓.....	- 112 -
A 函数下单撤单和全局变量操作 .....	- 113 -
数据库读写 .....	- 116 -
平仓延迟反手 .....	- 118 -
<b>●策略性能测试与参数优化的具体计算公式.....</b>	<b>- 121 -</b>
交易策略性能测试报告.....	- 121 -
交易策略参数优化报告.....	- 123 -
<b>●公式编写常见问题 .....</b>	<b>- 125 -</b>

## ●前言

《TradeBlazer 公式开发指南》是一本有关 TradeBlazer 公式语言系统编写的工具手册，旨在让读者学习公式的编写或提高公式编写的能力。

《TradeBlazer 公式开发指南》的编写是基于交易开拓者软件平台 V4 版本。

本指南讲解了 TradeBlazer 公式的语法基础、运行机制及如何建立公式应用并建立完整的公式应用等，从而帮助读者建立自己的 TB 公式进行技术分析乃至是可执行全自动委托单发送的交易策略系统。

本指南正文内容中所示的例句、公式代码、公式策略进阶等只为阐述、讲解函数、语法及语句的实现等等学习目的而使用，并非提供商业用途。本指南不保证综上所述示例在实际交易中的有效性及可盈利性。请读者对其正确判断后方可使用，一旦使用综上所述示例而导致的交易结果，均由交易者自己承担。

# ●TradeBlazer 公式概述

交易开拓者公式平台的编辑语言是 TradeBlazer Language，简称“TB 语言”，本手册内容是 TradeBlazer 公式的全面参考手册，详细介绍了 TradeBlazer 公式的结构、语法、特点、使用方法及功能等等。

通过阅读该参考手册，您能够了解 TradeBlazer 公式的基本语法、操作符、表达式及控制语句等，通过手册提供的各种示例程序，掌握各种 TradeBlazer 公式的编写要领，最终达到能够熟练将自己的思想转化为 TradeBlazer 公式，并在交易开拓者软件中应用。

TradeBlazer 公式是一种专为分析金融数据-时间序列而设计的高级语言，它提供直接、强大的框架将交易思想转化为用户函数、公式应用等计算机能够识别的代码。

TradeBlazer 公式是一门语法简单但是功能强大的语言，它能帮助您创建自己的交易和技术分析工具。通过组合普通的公式应用和简单的语句，TradeBlazer 公式使您能够很容易并且直接的用简单语句表达自己的交易规则和行为。

交易开拓者能够读取您开发的 TradeBlazer 公式，在历史价格数据基础上进行评估，并能自动执行特定的交易动作，将您的交易思想转化为实际的交易操作。

通过 TradeBlazer 公式，您能够创建自己的公式应用和用户函数。您也可以拷贝，修改并使用系统内建的几百个函数、公式应用。

TradeBlazer 公式包含的公式类型如下：

**用户函数：**用户函数是能够通过函数名称进行引用的指令集，它执行一系列操作并返回一个值。您可以在其他用户函数或公式应用中调用用户函数进行计算。

**公式应用：**公式应用除了可以实现技术分析功能之外，还可以实现交易策略的功能，您可以把分析功能和交易功能进行有机的组合，更方便快捷的进行分析和自动交易。通过调用公式应用，您可以在交易开拓者中进行技术分析，在历史数据中进行交易策略的性能测试，以及参数的优化从而得出适用于当前的最佳参数、公式报警以及设置实现程序化自动交易等操作。

# ●TradeBlazer 公式的命名

公式简称的命名需要遵守以下规则：

- 不区分大小写；
- 不能超过 32 个英文字符；
- 每一类公式不能出现相同的名称；
- 公式名称不能出现字母、数字、下划线以外的其他字符；
- 不能使用 C++ 关键字；
- 公式名称不能和系统保留字，系统函数等重名。

参数、变量的命名需遵守以下规则：

- 不区分大小写；
- 不能超过 32 个英文字符；
- 每一个公式内部不能重复命名；
- 名称不能出现字母、数字、下划线以外的其他字符；
- 名称不能和系统保留字，系统函数等重名；
- 不能使用 C++ 关键字；
- 不能使用已定义的用户函数名称。



## ●TradeBlazer 公式正文规则

除了使用“...”引用起来的字符串之外，整个公式正文中参加编译部分的语句不可以有中文字符的存在。另外，因为注释语句不参与公式主体的计算，所以允许有中文字符的存在。若需要对单行语句进行注释，可以在句首使用“/”将该行文字注释，若是需要对多行语句进行注释，则可使用“/\* ...\*/”将段文字注释。

公式正文字符的颜色列表：

- 黑色 --- 用户自己声明的变量名或是参数名；
- 红色 --- 数字；
- 蓝色 --- 系统函数；
- 暗红色 --- 已有的用户函数；
- 紫红色 --- 运算符号；
- 果绿色 --- 字符串（可以为中文字符）；
- 翠绿色 --- 注释语句（注释符号后可为中文字符）。

## ●TradeBlazer 公式体系架构

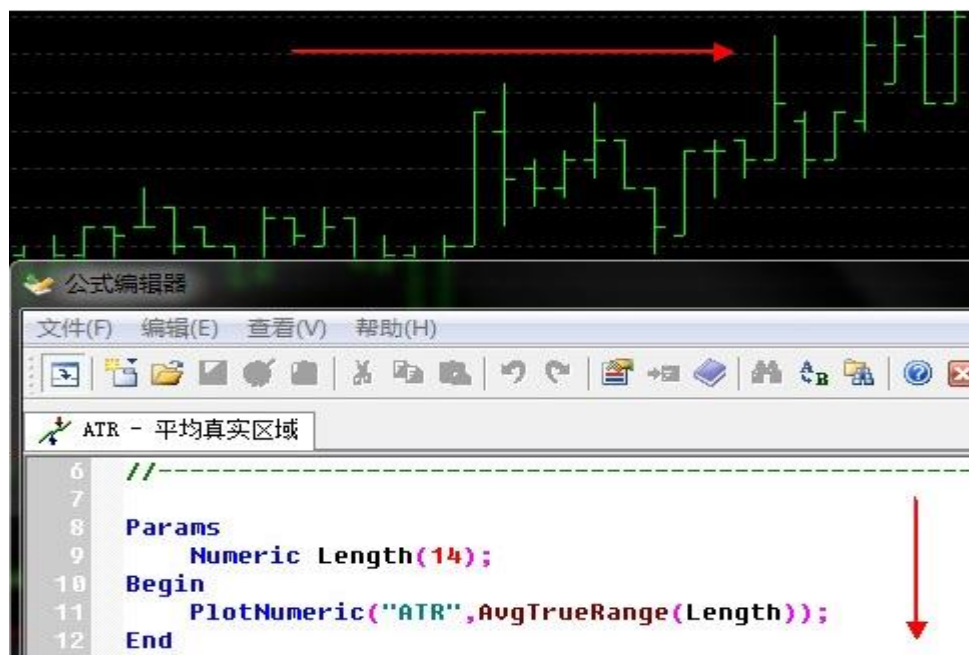
TB 语言做为一个高级语言，其语法却是简单易懂的，其语法类似介于 C++ 与 pascal 之间的语法。其程序语言可以由多重数学、布尔的计算以及逻辑判断等组成。TB 公式属于编译公式，也就是说只有通过编译的公式程序方可被应用于图表，这样使得公式的应用更快更有效率。

### 公式的运算步骤

在公式进行计算时，都是建立在基本数据源----Bar 数据之上。这里所指的 Bar 数据，是指商品在不同时间周期下形成的序列数据，在单独的每个 Bar 上面包含开盘价、收盘价、最高价、最低价、成交量及时间以及持仓量等信息数据。Bar 数据也是我们口头上常常所说的 K 线数据。（有关 Bar 数据后面会有详细介绍）

TradeBlazer 公式在计算时按照 Bar 数据的 Bar 数目，从左边第一个 Bar 到右边最后一个 Bar 的顺序依次进行计算。而公式在单个 Bar 数据的计算方向是从上到下。即每一次公式的运算都是从最上方的公式语句开始逐步向下计算。包括从参数的声明开始、变量的声明直至公式的计算主体 begin 至 end 的结束。

如下图所示：



## 历史数据回溯

在公式的编写中，经常会遇到当前 **Bar** 的数据和上一个 **Bar**，上 **N** 个 **Bar** 数据进行比较，计算的情况，针对这种情况，TradeBlazer 公式提供了一种处理机制：回溯，即对数据的向前引用。我们通常使用[]并在其中填写数值来对回溯所需的 **Bar** 数来进行标识。比如，获取上一个 **Bar** 的收盘价：Close[1]，获取 10 个 **Bar** 前的成交量：Vol[10]。以下提供一个简单的例子来说明如何进行回溯处理。

假定有如下语句：

```
If (Close > Close[1])
{
    Buy(1,Close);
}
```

以上公式执行一个简单的操作，当前 **Bar** 的收盘价大于上一个 **Bar** 的收盘价，即执行按照当前收盘价买入 1 手的动作。根据上表的数据，公式将在 **CurrentBar** 为 2 和 3 的时候调用 Buy 指令。

如果您足够仔细的话，您会发现：对于上面的一段公式的执行，有一个小小的问题，当第一次计算公式时，即 **CurrentBar** = 0 时，这个时候需要获取上一个 **Bar** 的数据，但是当前 **Bar** 已经是第一个 **Bar**，这个时候就存在着问题，如何来获取此时的 Close[1]呢，在这个时候 TradeBlazer 公式将取到当前 Close 来代替，相当于在 **CurrentBar** = 0 时，此条件式是进行了 If (Close > Close)的判断，此时条件不满足。因此，第一个 **Bar** 计算时，Buy 动作是不会被执行的。

假定 **Bar** 数据的总数共有 100，相同的代码将从 **CurrentBar** = 0 到 **CurrentBar** = 99 共执行 100 遍，分别输出公式中的结果值。

## 实时数据的运算

对于历史数据，我们已经知道每一个 **Bar** 上都会计算一次，直至将当前图表上所有的 **Bar** 全部从左到右的计算一遍。

在实时行情中，当前 **Bar** 的数据是随时行情的波动发展而随时变化。而对于实时数据，每当有新的 Tick 进来，公式都会当前的 **Bar** 上对新数据执行一次完整的运算，但不会再回去计算历史数据。

注意：在实时行情中，若当前公式所应的合约交易活跃且公式程序较长、计算较复杂时，则有可能发生当前 **TICK** 到来之后及下一个 **TICK** 到来之前的这段时间之内，无法完成公式代码一遍的计算。如果遇到这样的情况，公式代码不会随着新 **TICK** 的进入而跳转从头开始计算新 **TICK**，而是会将在当前 **TICK** 的计算执行直至代码的最后一行，然后在最新的 **TICK** 进来后执行下一次的运算。这样一来，中间则可能会有部分 **TICK** 没有参与计算。

## ●基础模块

交易开拓者提供三种图表供交易者按自己的习惯与需求来使用，分别是行情报价、分时图表及超级图表。其中可以加载公式应用并执行其指令的是“超级图表”。

### 行情报价

行情报价是交易开拓者专为显示动态行情设计的行情揭示系统，拥有强大的行情订阅和检索功能。

您可以点击标题栏的某一个字段进行排序，再次点击，将会切换升序/降序。

您可以拖拽一条报价记录，改变排列顺序，如果拖拽到超级图表窗口，将会替换原商品代码为当前代码。

您可以选中一条报价记录，右键菜单进行相关操作。

下图为行情报价界面图：

行情报价									
	简称	现价	涨跌C	涨幅C%	涨幅%	现手	总手	持仓量	仓差
1	橡胶指数	32703	97	0.30	-0.30	28	239666	305656	19010
2	股指指数	2984.1	8.2	0.28	0.17	2	12020	46728	1524
3	沪铜指数	68438	205	0.30	0.02	0	40162	336474	-3296
4	白糖指数	6616	6	0.09	-0.19	2	237888	700670	592
5	棉花指数	25340	368	1.47	1.04	78	538752	675468	63578

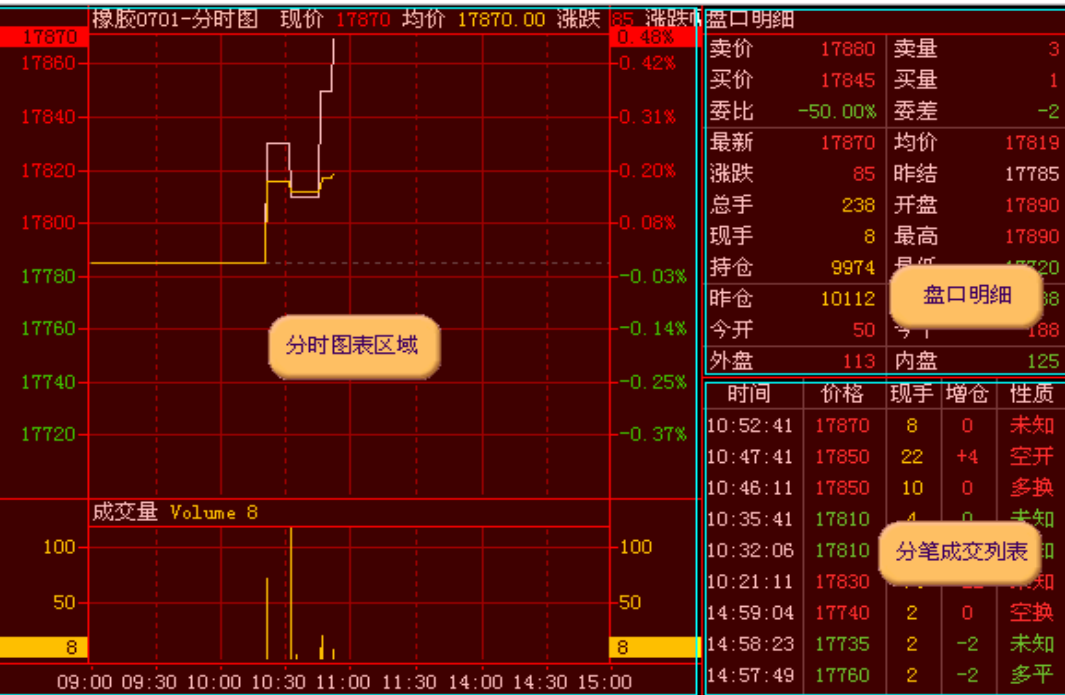
### 分时图表

分时图是交易开拓者专门显示分时图表、盘口明细及分笔成交的模块，拥有更加直观的行情揭示功能。

分时图表采取固定 1 分钟时间坐标的方式，并以昨结算价为纵坐标基准，比较直观的表现出当日行情的走势。使用收盘线的显示方式，白线是 1 分钟的收盘连线(包括无成交量及休息时段)，黄线为实时结算价的连线。

分时图表下部用涨红跌绿的柱状线显示成交量，用黄线显示持仓量，可以单击鼠标右键切换成交量和持仓量的显示模式。

分时图的主界面如下图：



盘口明细通过列表的方式，直观的显示出 22 项实时行情字段，列表如下：

- **卖价**：实时行情的委卖价；
- **卖量**：实时行情的委卖量；
- **买价**：实时行情的委买价；
- **买量**：实时行情的委买量；
- **委比**：用以衡量一段时间内买卖盘相对强度的字段， $(\text{买量}-\text{卖量})/(\text{买量}+\text{卖量})\times 100\%$ ；
- **委差**：反映买卖双方的力量对比，正数为买方较强，负数为抛压较重，等于买量-卖量；
- **总手**：实时行情的当日成交量总数；
- **开盘**：实时行情的当日开盘价；
- **现手**：实时行情的最新成交数量；
- **最高**：实时行情的当日最高价；
- **持仓**：实时行情的最新持仓量；
- **最低**：实时行情的当日最低价；

- **昨仓**：实时行情的昨日最后持仓量；
- **仓差**：反映当日的持仓变化，等于持仓-昨仓；
- **今开**：实时行情的今日开仓量，为每笔成交的开仓量累计之和，每笔成交的开仓量等于(当笔现手+当笔持仓增量)/2；
- **今平**：实时行情的今日平仓量，为每笔成交的平仓量累计之和，每笔成交的平仓量等于(当笔现手-当笔持仓增量)/2；
- **外盘**：实时行情的当日外盘，为每笔成交的外盘累计之和；
- **内盘**：实时行情的当日内盘，为每笔成交的内盘累计之和。

**注意：**内外盘的计算方式为：每笔成交中，以委买价成交，成交量计入内盘，以委卖价成交的，成交量计入外盘，如果当笔最新价既不等于委买价，也不等于委卖价，则各计一半。

分笔成交以列表的形式显示出当日开盘以来所有的成交记录，最新的记录排在最上面。

分笔成交显示如下字段：

- **时间**：当笔成交的产生时间；
- **价格**：当笔成交的成交价格；
- **现手**：当笔成交的成交数量；
- **增仓**：当笔成交的持仓变化量；
- **性质**：当笔成交的开平仓性质，分为九种类型，详细参见下节的开平仓性质。
  - ✓ **双开**：双向开仓的简称；
  - ✓ **双平**：双向平仓的简称；
  - ✓ **多换**：多头换手的简称；
  - ✓ **空换**：空头换手的简称；
  - ✓ **多开**：多头开仓的简称；
  - ✓ **多平**：多头平仓的简称；
  - ✓ **空开**：空头开仓的简称；
  - ✓ **空平**：空头平仓的简称；
  - ✓ **未知**：其他的情况，主要是在合笔及内外盘不确定的情况下产生。与股票不同，国内期货的成交量都是双边计算，因此成交量中即包含了买入量，也包含了卖出量，是单边计算的两倍。每笔成交都伴随着成交量的增加，但持仓量却有可能出现增加、不变和减少三种情况。在没有合笔的情况下，我们定义了双向开仓、双向平仓、多头换手、空头换手四种基本状态。

双向开仓是指某笔成交中，开仓量等于现量，平仓量为零，持仓量增加，差值等于现量，表明多空双方均增仓；双向平仓是指某笔成交中，开仓量等于零，平仓量为现量，持仓量减少，差值等于现量，表明多空双方均减仓；若某笔成交中，开仓量和平仓量均等于现量的一半，持仓量不变，则表明多头仓位和空头仓位都未发生变化，只是部分仓位在多头之间或空头之间发生了转移，结合内外盘的状态，我们定义外盘时该笔成交的状态为多头换手，内盘时为空头换手。

若交易所数据有合笔情况，由于该笔成交有可能是以上四种基本状态的一个组合，持仓量的变化与成交量的变化将会出现不一致。结合内外盘状态，我们又定义了多头开仓、多头平仓、空头开仓、空头平仓四种开平仓状态。多头开仓指持仓量增加，但持仓量的增加值小于现量，且为主动买盘；空头开仓指持仓量增加，但持仓量的增加值小于现量，且为主动卖盘；多头平仓指持仓量减少，但持仓量的增加值小于现量，且为主动卖盘；空头平仓指持仓量减少，但持仓量的增加值小于现量，且为主动买盘。

每笔成交是多空双方力量相互作用的结果，分析每笔成交的开平仓状态，读懂市场语言，是每个短线投资人必修的功课。例如，价格上升，持仓量增加，表明多空双方对后市有分歧，空头并未有认输之意；而价格上升，持仓量却减少，则表明价格上涨是由于空头主动平仓造成。反之，价格下跌，持仓量增加，表明多空双方对后市有分歧，多头并未有认输之意；而价格下跌，持仓量却减少，则表明价格上涨是由于多头主动平仓造成。当然，单笔成交中偶然性因素很多，一定时间周期内的多空头开平仓量统计值将会更有助于投资人进行研判。

## 超级图表

超级图表是交易开拓者的一个重要模块，它提供了商品数据的多种图形查看方式，提供了画线分析、公式应用优化测试及自动交易等功能。

您可以通过以下几个方式新建超级图表窗口：

点击系统菜单“文件”-“新建”-“超级图表”菜单项；

1. 直接按快捷键 **Ctrl+N**，在新建窗口中选择超级图表模块，点击确定；
2. 点击工具栏中的新建按钮，在新建窗口中选择超级图表模块，点击确定；
3. 点击或拖拽面板中的行情报价按钮，将会新建或替换一个超级图表窗口。



超级图表主界面如下图所示：



## 商品设置

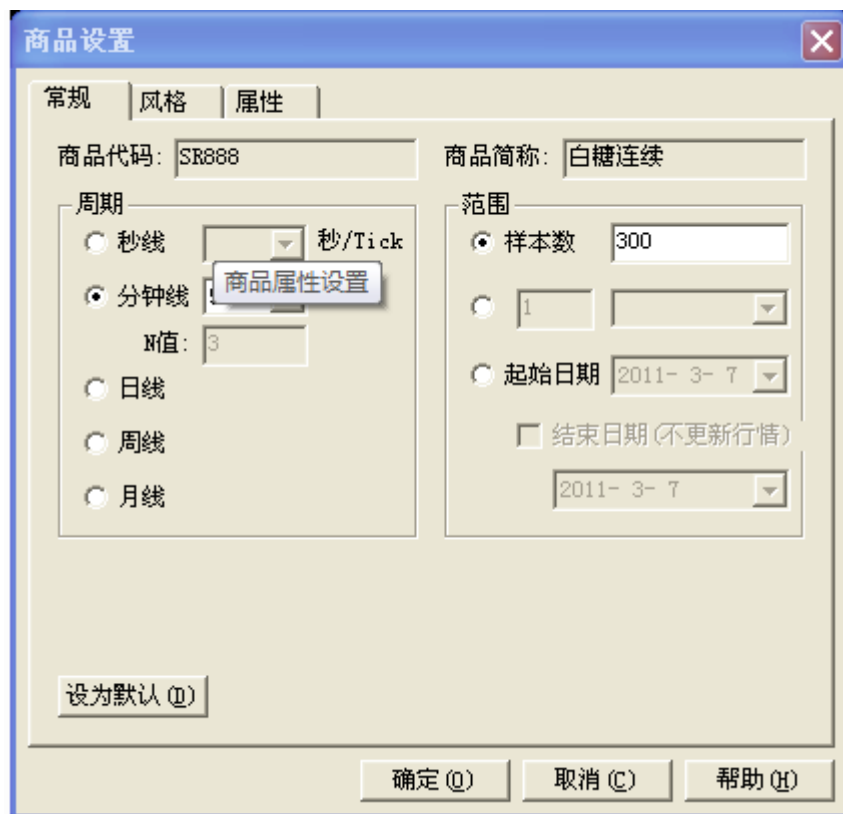
在超级图表窗口被激活之后，点击菜单-“格式”下的“商品设置”，或在超级图表内双击行情数据，可打商品设置对话框。如下图：

The '商品设置' (Commodity Settings) dialog box is shown. It has a title bar with a close button. The main area contains a table with columns: '商品代码' (Commodity Code), '商品名称' (Commodity Name), and '周期' (Period). The first row is filled with 'SR888', '白糖连续', and '5分钟线'. To the right of the table are five buttons: '属性 (P)' (Properties), '交易 (T)' (Trading), '删除 (D)' (Delete), '关闭 (C)' (Close), and '帮助 (H)' (Help).

商品代码	商品名称	周期
SR888	白糖连续	5分钟线

- 属性：打开选中商品的属性设置；
- 费率：打开选中商品的费率设置；
- 删除：从图表中删除选中商品，第一个数据源不能被删除；
- 关闭：关闭当前对话框；
- 帮助：调用当前对话框的联机帮助；

点击“属性”，打开商品属性设置，会弹出如下窗口：



您也可以通过双击主图来打开商品设置对话框。

商品设置包括以下三个页面：常规、风格和属性。

### 常规

**周期：**选择当前商品的数据周期，共有 1Tick、10 秒、1 分钟、5 分钟、15 分钟、30 分钟、N 分钟、1 小时、4 小时、1 日、1 周、1 月共十二种周期可供选择。其中包括的 N 分钟周期取值范围可以从 1-59；

**范围：**数据订阅的长度范围，可按照以下三种方式设置：

样本数：最后 N 个 Bar 的数据，N 的取值范围为(1-80000)；

N 天/周/月/年以来的数据，累计的 Bar 值不能超过 80000；

从某个指定起始日期以来的数据，并可指定数据结束的日期。

**注意：**N 分钟周期是以 1 分钟数据为基础得出的，所以 N 分钟周期的最大取值数量为 80000/N 个 Bar。

## 风格

可在此页面选择当前商品的主图线型及设置显示的颜色，选中某一线型，点击“确认”按钮即可，有以下四种线型可供选择：

**蜡烛线：**蜡烛线又名 K 线图，原来是日本米市商人用来记录米市当中的行情波动，后因其标画方法具有独到之处，因而在股市及期市中被广泛引用。K 线将买卖双方力量的增减与转变过程及实战结果用图形表示出来。经过近百年来使用与改进，K 线理论被投资人广泛接受；

**中空蜡烛线：**中空蜡烛线是在蜡烛线的基础上，用中空蜡烛线表示上涨，此举是为了满足黑白打印图表的需求；

**美国线：**美国线的构造则较 K 线简单。美国线的直线部分，表示了当天行情的最高价与最低价间的波动幅度。左侧横线代表开盘价，右侧横线侧代表收盘价；

**收盘线：**收盘线是将收盘价相连形成的折线，收盘线只关心每日的收盘价格，可简明地展现价格变化趋势。

## 属性

该页显示和当前商品相关的属性，具体解释可参见“数据管理”的商品属性设置。

## 叠加商品设置

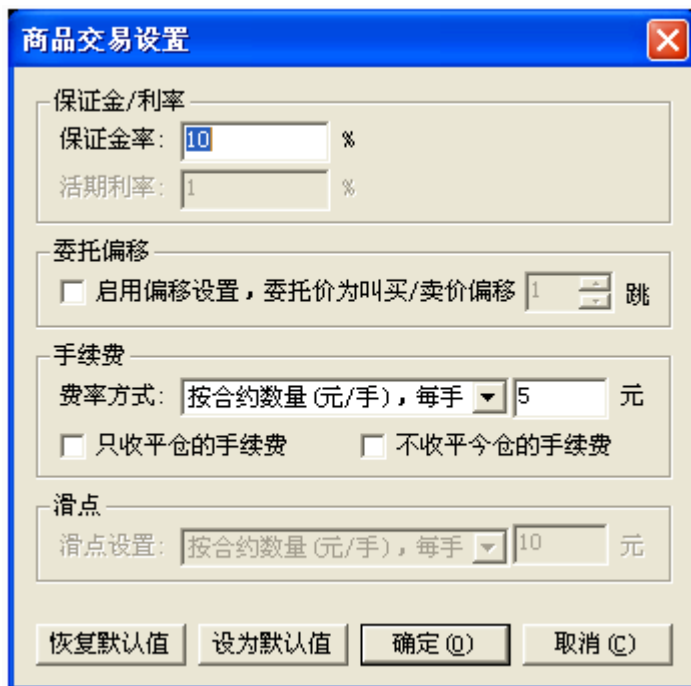
当前图表如果有叠加商品时，从菜单中点击“商品设置”，会弹出如下窗口：



在图表有叠加商品时，不能通过商品设置进行周期切换，此时可以通过工具栏进行周期切换。

**注意：**在常规和风格页面，都有“设为默认”按钮，点击该按钮将该页面的设置保存到配置文件中，以后每次新建超级图表，将会使用该默认配置。

商品交易设置如下图：



商品交易设置对话框，包含以下配置项：

- 保证金/利率**
  - 保证金率：10 %
  - 活期利率：1 %
- 委托偏移**
  - ☐ 启用偏移设置，委托价为叫买/卖价偏移 1 跳
- 手续费**
  - 费率方式：按合约数量 (元/手)，每手 5 元
  - ☐ 只收平仓的手续费 ☐ 不收平今仓的手续费
- 滑点**
  - 滑点设置：按合约数量 (元/手)，每手 10 元

底部按钮：恢复默认值、设为默认值、确定 (O)、取消 (C)

- **保证金利率：**设置为您的交易帐户的保证金比率，不同的商品有不同的保证金比率，系统通过初始资金和保证金率来计算交易的默认数量；
- **手续费：**如图所示，有三种计算手续费的方式可供选择，根据交易商收取佣金的方式分别设置；
- **滑点：**暂不支持，将在以后版本不断完善。

## ●数据

TradeBlazer 公式的应用是基于图表运行的，只有在超级图表上，有 Bar 存在的情况下，才能进行一系列的运算。交易开拓者软件里的“超级图表”也就是平时常说的“K 线图”。交易开拓者对各个合约、不同周期的超级图表分别可提供 8 万个历史 BAR 数据（部分交易者称之为“K 线数据”），且我们正在努力将此数据量更大化。大量的历史数据为交易者对自己的交易策略进行历史性的测试分析提供了便捷。

除了上述的 Bar 数据之外，交易开拓者还有帐户数据、行情数据、属性数据等多种数据可供交易者使用。

目前，交易开拓者开放国内四大交易所的全部交易合约期货数据，在此基础上更有每个品种的指数数据与连续数据可提供。

指数数据是用某品种当前时期全部的交易合约加权平均而计算得到的数据，其中以持仓量与成交仓占较大的权重比例，其代码为 XX000 或 X9000（双字母的商品代码后加 000，单字母的商品代码后加 9000）。

连续数据是用不同时期当时的主力合约数据剪切接拼而组成，其代码为 XX888 或 X9888（双字母的商品代码后加 888，单字母的商品代码后加 9888）。换月的标准为：当新合约的成交量是原主力的 1.1 倍后，则第二天开始以新合约的数据作为当前连续合约的数据。

\*\*\*交易开拓者接入国内股票数据的测试工作已经接近尾声。不久的将来，交易者便可以在交易开拓者的平台上看到国内股票的行情，以及对其进行技术分析、测试、下单交易等操作。届时便可做到股指的期现套利。

## 数据类型

TradeBlazer 公式支持有三种基本数据类型：数值型、字符串、布尔型。

为了通过用户函数返回多个值，我们对三种数据类型进行了扩展，增加了引用数据类型。另外，为了对变量，参数进行回溯，我们增加了序列数据类型。因此，我们的数据类型共有九种，如下表所示：

<b>Bool</b>	布尔型。
<b>BoolRef</b>	布尔型引用。
<b>BoolSeries</b>	和周期长度一致的 Bool 型序列值。
<b>Numeric</b>	数值型。
<b>NumericRef</b>	数值型引用。
<b>NumericSeries</b>	和周期长度一致的 Numeric 型序列值。
<b>String</b>	字符串。
<b>StringRef</b>	字符串引用。
<b>StringSeries</b>	和周期长度一致的 String 型序列值。

## Bar 的索引值

好比一个列队，每一 Bar 在超级图表中都有其相应的位置，为了方便记录与查找，我们为每行编号并将其称为 Bar 的索引值。在 TradeBlazer 公式中用系统函数 **Currentbar** 来表示当前公式应用商品当前 Bar 的索引值。

图表左边第一个 Bar 返回值为 0，其他 Bar 则向右逐个递增。

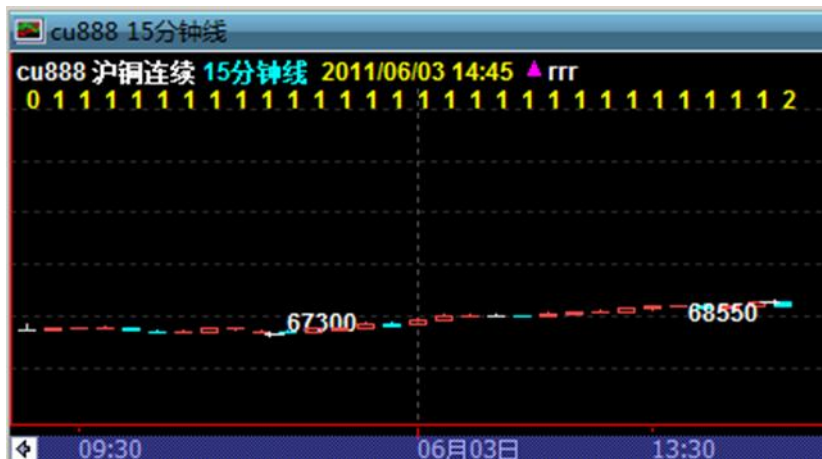
例句：`PlotString("CurrentBar",text(CurrentBar));`



## BAR 的状态值

**Barstatus** 表示当前公式应用商品当前 **Bar** 的状态值，返回值 **0** 表示图表最左边第一个 **Bar**，返回值为 **1** 表示为除去第一个 **Bar** 和最后一个 **Bar** 后所有中间的全部 **Bar**，返回值为 **2** 表示最后一个 **Bar**(即图表上最右边的一个 **Bar**)。

例句: `PlotString("Barstatus",text(Barstatus));`



## 数据周期

交易开拓者对交易所输出的交易数据以时间做为切分，从而生成不同时间周期的 **Bar**。每个合约分别提供有 **Tick**，10 秒，1 分钟，5 分钟，15 分钟，30 分钟，N 分钟，1 小时，4 小时，日，周以及月的 **Bar** 数据。其中的 N 分钟 **Bar** 数据的 N 值是范围从 1-59 之间可由交易者自由设定的值，从而生成自己所需要的分钟周期数据。

**BarType** 表示当前公式应用商品数据的周期类型值，返回值为整型，该值通常和 **BarInterval** 一起使用进行数据周期的判别。

返回值定义如下：

日线

分钟线

## TICK 线

量线

## 周线

月线

## BAR 数据

在公式进行计算时，都是建立在基本数据源(Bar 数据)之上，我们这里所谓的 Bar 数据，是指商品在不同周期下形成的序列数据，在单独的每个 Bar 上面包含开盘价、收盘价、最高价、最低价、成交量及时间。期货等品种还有持仓量等数据。

所有的 Bar 按照不同周期组合，并按照时间从先到后进行排列，由此形成为序列数据，整个序列称之为 Bar 数据。

以下列出所有的 Bar 数据系统函数：

函数名	简写	描述
<b>Date</b>	<b>D</b>	当前 Bar 的日期。
<b>Time</b>	<b>T</b>	当前 Bar 的时间,是以当前 BAR 开始的时间来记录。
<b>Open</b>	<b>O</b>	当前 Bar 的开盘价。
<b>High</b>	<b>H</b>	当前 Bar 的最高价，Tick 时为当时的委卖价。
<b>Low</b>	<b>L</b>	当前 Bar 的最低价，Tick 时为当时的委买价。
<b>Close</b>	<b>C</b>	当前 Bar 的收盘价。
<b>Vol</b>	<b>V</b>	当前 Bar 的成交量。
<b>OpenInt</b>	无	当前 Bar 的持仓量。
<b>CurrentBar</b>	无	当前 Bar 的索引值，从 0 开始计数。
<b>BarStatus</b>	无	当前 Bar 的状态值，0 表示为第一个 Bar，1 表示为中间的普通 Bar，2 表示最后一个 Bar。

**Date:** 在当前公式应用商品在当前 BAR 的日期,格式为 YYYYMMDD 的整数，如：当前 Bar 日期为 2004-1-24，Date 返回值为 20040124。若是在周线或是月线上,返回的则是当周或是当月第一个交易日的日期。

**Time:** 当前公式应用商品在当前 Bar 的时间，格式为 0.HHMMSSmmm 的浮点数,如：当前时间为 11:34:21.356，Time 返回值为 0.113421356。除日线及以上级别的周期，time 返回的都是该 Bar 开始的时间。日线、周线、月线上的 time 均为 0。



**Close:** 当前公式应用商品在当前 **Bar** 的收盘价。当一个 **Bar** 没有结束之前，**close** 的值是一直以最新价来更新变动的，直至此 **Bar** 的结束。

## 行情数据

除了 **Bar** 数据之外，交易开拓者也提供行情数据并支持在 **TB** 公式中对行情数据的调用，行情数据是指当前商品最新的报价数据，行情数据在 **TB** 语言中是以“**Q\_**”打头的一系列函数，该类数据与 **Bar** 无关，不能进行回溯。简单地说就是只能取到最新的数据，不能对历史中的行情数据进行回溯或是调用。

行情数据只在最后一个 **Bar** 是有意义的，其它 **Bar** 会返回无效值（部分行情函数，如：**Q\_UpperLimit**, **Q\_PreSettlePrice** 等是当日内可以取有效值）。因此，在调用行情数据函数时，为了提高效率，最好按照以下方法：

```
if(BarStatus==2)
{
    //调用行情函数
}
```

行情数据函数都按照以下格式命名 **Q\_XXXX**，比如 **Q\_AvgPrice**（当前公式应用商品的实时均价），**Q\_BidPrice**（当前公式应用商品的最新买盘价格）。

在调用行情数据的时候，需要判断当前行情数据是否有效，系统提供函数 **QuoteDataExist** 来对有效性进行判断。如果行情数据已经准备好，返回 **True**，否则，返回 **False**。

## 属性数据

**TradeBlazer** 公式还提供一组重要的属性数据，反映了图表当前该商品的一些基本信息，比如当前的数据周期，买卖盘个数、保证金设置等信息。在所有的 **Bar** 上面获得的市场属性数据都是一样的，属性数据的回溯没有意义。

示例：

**Symbol** 当前公式应用商品的合约代码

**Symbol** 当前公式应用商品的名称

**MarginRatio** 当前公式应用商品的默认保证金比率

## 帐户数据

TradeBlazer 公式可以支持实时帐户数据的调用，帐户数据是指当前交易帐户最新的帐户数据，该数据和 **Bar** 无关。

帐户数据只在最后 **Bar** 是有意义的，其他 **Bar** 会返回无效值。因此，在调用帐户数据函数时，为了提高效率，最好按照以下方法：

```
If(BarStatus==2)
{
    //调用帐户数据函数
}
```

帐户数据函数都按照以下格式命名 **A\_XXXXX**，比如 **A\_BuyPosition**，**A\_FreeMargin**，**A\_OpenOrderContractNo**。在调用行情数据的时候，需要判断当前所应用的公式是否已经关联了交易帐户，只有在已关联交易帐户的情况下方可有效调用此类数据。系统提供函数 **A\_AccountID** 来对有效性进行判断。如果帐户数据已经准备好，返回交易帐户 **ID**，否则返回空的字符串。

其中帐户函数 **A\_SenderOrder()** 配合使用枚举函数可以做到对帐户直接发出指令，而不会在图表上做出任何的信号标识。

需要注意的是，此函数只在最后 **Bar** 上有效，也就是只能在实时行情中对帐户进行发出委托指令的动作，无法在历史数据上做任何的信号标识或是记录。鉴于前面的 **TB** 运行机制我们也有讲述过，公式在实时行情中，会每一个 **TICK** 执行一次，所以该函数在满足条件的每一个 **TICK** 里，每一次的公式执行都会发送一次指令。为此在公式的编写中还需要一些控制语句来防范因此而导致的重复发单。有关 **A\_SenderOrder()** 的具体使用请参考后面章节的“公式进阶”。

## 数据叠加

交易开拓者的超级图表支持商品数据叠加的显示，当叠加的图表调用各项公式时，可能有需要使用叠加的商品对应的基础数据，针对这样的需求，TradeBlazer 公式提供了叠加数据的支持。这样的功能使得价差分析以及和套利交易的实现成为可能。

假定，我们新建一个超级图表模块，其主数据对应的商品为：**cu1109**，在此基础上，我们叠加了 **cu1201** 和 **cu1205**。（叠加商品的操作是在文件菜单的“插入”里选择“插入商品”或是在超级图表上右键菜单里选择“插入商品”）。此时，根据叠加操作的先后顺序，**cu1109** 为 **Data0**，**cu1201** 为 **Data1**，**cu1203** 为 **Data2**，在 TradeBlazer 公式中，我们可以通过 **Data1.Close()**，**Data2.Vol()** 类似方法调用叠加 Bar 数据，叠加 Bar 数据的函数和 Bar 数据一样，只是需要在调用的时候加上数据源。

我们也可以使用 **Data0.Open()** 来调用 Bar 数据，默认情况下，可以省略对主数据源的指定，为了方便，一般直接使用 **Open()** 来代替 **Data0.Open()**。

现有价差计算的公式如下：

Begin

```
PlotNumeric("Spread",Data0.Close - Data1.Close);
```

End

下图所示为两个合约叠加及计算得出的价差指标图：



另可以将两个叠加的品种数据分别设置为主/副图，**data0** 的数据源设为主图显示，**data1** 的数据源设置副图显示。如下：



## 数据源

因数据切片的差异，从而不同行情来源所得到的日内 **Bar** 数据有所不同。比如不同的软件 **ID** 登录会指向不同的数据源，即便是同一个用户 **ID**，选择不同的线路（联通或是电信），也会是不同的数据源，这样同一个策略放在不同的数据源所得到的测试结果会有所不同。交易者需要关注这一点，从测试到交易尽可能固定使用同一个数据源的数据。

# ●TradeBlazer 公式应用的建立

## 如何新建公式应用

在交易开拓者里，无论是技术分析还是历史回溯测试或是自动交易的实现，一切的计算都是基于公式的运用来进行的，那么实现上述操作之前，首先是要建立自己的公式应用。

可以自己通过公式管理器打开新建公式应用，编写代码并进行编译。也可以导入现成的公式应用，直接使用。

本章节里将着重讲解建立一个公式应用的具体步骤。

## 新建公式简称与名称、注释

在打开公式管理器的界面，选择“新建公式应用”，接下来就是对你所想要建立的公式应用命名了。

首先是简称，需要为英文字符串，且不可以与已有同类型公式同名，具体规则详见“命名规则”。

名称是对简称的一个补充命名，可以为中文、英文、数字等，这样便于交易者按自己的习惯来辨别记录。

模板的可选项为“空”、“技术分析”或“交易策略”，您可以根据自己的公式编写需求来进行相应的选择。

最后则是注释框，注释内可以存放交易者设计编写此公式的思路及一些概括性的描述，便以后的使用与修改。

除了第一项的简称是必填的内容，其它的内容都是可选内容，没有强制填写的要求。

## 新建公式内容

在公式编辑器界面里建立公式的内容，编写自己的公式代码或是从其它 TB 公式里复制代码贴粘到当前编辑界面，也可以将现有的 TB 公式代码复制到当前的新公式中，并加以修改从而建立自己所需的代码。

## 函数说明列表

一旦下载安装了交易开拓者平台版软件，软件的公式管理器里已经存放了近百个可直接使用的用户函数，且代码均为开放的。

您可以打开任意的用户函数并可查看其代码，从而了解各函数的计算规则。或是在现有的函数的基础上复制、做出修改从而建立新的用户自定义函数，以便于自己个性化计算需求的使用。具体函数内容及代码请参考交易开拓者软件旗舰版的公式管理器。

除用户函数外另有大量的系统函数，均可在软件帮助文档（F1 键）里的索引里找到具体的说明及用法。

## 校验保存公式

在公式编辑界面，在公式代码已经编写完成后，即可点击“校验保存公式”这个按钮来完成公式的编译。一旦编译成功后，在公式编辑界面的左下方会有“成功保存当前公式信息”，且会有一个绿色小勾图样，这样您便可以对此公式进行调用了。

代码的编写没有全部完成中途需要退出 TB 软件，则可以点击“保存”按钮以将当前已完成的部分代码先保存下来，待编写完成后再点击“校验保存公式”。

**注意：**只保存而未通过校验的公式是不可以被调用的。

但如果公式的代码存在一些错误，会导致公式校验保存不成功，这时就需要根据错误提示的描述及所示的位置进行查找，从而改进公式直至通过校验。

以下例出 TB 公式可能出现的错误代码：

#### 基本编译错误

错误代码	错误描述
<b>C0001</b>	程序体不存在
<b>C0017</b>	参数声明的数据类型和初始值的数据类型不一致
<b>C0018</b>	变量声明的数据类型和初始值的数据类型不一致

#### 语法规义错误

错误代码	错误描述
<b>C0102</b>	变量被重复定义
<b>C0103</b>	函数被重复定义
<b>C0107</b>	变量声明的数据类型错误
<b>C0108</b>	参数声明的数据类型错误
<b>C0109</b>	公式返回的数据类型错误
<b>C0110</b>	命名的第一个字符不能是\$
<b>C0111</b>	向前引用指示必须是数值型变量或常量
<b>C0112</b>	赋值语句左右值必须使用同类数据类型
<b>C0114</b>	赋值语句左值必须是变量而不能为常量
<b>C0115</b>	赋值语句左值变量不可使用向前引用
<b>C0116</b>	逻辑运算语句的左右值的数据类型必须属于 <b>Bool</b> 类
<b>C0117</b>	算术运算语句的左右值的数据类型必须属于 <b>Numeric</b> 类
<b>C0118</b>	If 条件表达式数据类型必须属于 <b>Bool</b> 类
<b>C0119</b>	While 条件表达式数据类型必须属于 <b>Bool</b> 类

<b>C0120</b>	For 语句起步和终止条件表达式数据类型必须属于 Numeric 类
<b>C0121</b>	For 语句的循环变量不能为 NumericRef 类型
<b>C0122</b>	Return 语句的返回值类型与公式定义的返回值类型不符
<b>C0126</b>	关系运算语句的左右值的数据类型必须相同
<b>C0127</b>	参数缺少初始值
<b>C0128</b>	引用参数不应含初始值
<b>C0133</b>	赋值语句的左值只能为变量或者为引用类型的参数
<b>C0135</b>	本参数无初始值,则要求公式体内的前几个参数也不能有初始值

## 公式调用错误

错误代码	错误描述
<b>L0003</b>	函数实现的参数列表和预声明的参数列表不符合
<b>L0004</b>	函数调用时的参数数目与声明时不符合（太少的调用参数）
<b>L0005</b>	函数调用时的参数数目与声明时不符合（太多的调用参数）
<b>L0006</b>	被调用函数的序列参数不能使用默认值
<b>L0007</b>	被调用函数的引用参数不能使用默认值
<b>L0008</b>	只有序列变量和参数才能使用回溯值
<b>L0013</b>	函数的第一个参数必须是字符常量
<b>L0014</b>	被调用公式要求引用参数时，该参数只能以普通变量或引用参数方式传入

## 公式警告

<b>W0201</b>	FOR,WHILE,IF,ELSE 中包含序列函数，可能存在潜在的逻辑错误，请确认代码无误
--------------	---



除上述描述的错误码之外，会有类似以下字样的错误提示：

**fatal error C1001:**  
最终目标文件编译错误

这个是 **TB** 无法识别的、在 **VC** 返回的错误提示。除了 **C1001**,还有可能是其它的号码。遇到此类问题，可参考以下思路进行错误的查找：

1. 查看公式简称及公式正文中是否存在中文字符等非法字符；
2. 公式中参数变量的命名是否与已有函数，C 语言关键字有冲突的；
3. 是否有严重的逻辑错误或公式管理器中未通过编译且存在错误的公式等等；

## 公式在图表上的应用

在公式管理器里对某个公式应用进行“调用”，或是在超级图表右键菜单里选择“插入公式应用”，即可在在图表上实现在对公式的应用。

也可以直接在激活的超级图表上手工输出所需应用公式的简称代码，再按下回车键公式应用就会被图表调用了。

通过公式的应用，可以在超级图表上输出一系列的信息，如指标线、柱状图、字符串、以及交易指令信号等。

也可以在图上不输出任何信息，而依借图表数据从而输出文件或是注释信息，**fileappend** 是在指定的路径输出一个日志文件，**commentary** 是在图表上的注释信息框内输出注释的字符内容，具体请参考后面章节 **fileappend** 和 **commentary** 这两个函数的用法。

## 主/副图上的显示

在公式应用校验保存完成后，可以按自己的看图习惯来设定为主图显示或是副图显示。在公式编辑界面的属性里可选择成“主图显示”或是“副图显示”，再点击一下校验保存键。或者是在对公式代码编写完成后，直接进入属性里选择主或副图显示之后，再进行校验保存。此后每次调用该公式应用，都会显示在所选择的相应主副图位置上。

另外，还可以直接将主/副上的指标进行拖拽互换到指定位置。如果直接从副图拖拽到主图的公式应用，会因为没有经过计算对齐坐标，从而使得指标线等的显示与主图坐标不对应。且会在 K 线进行放大或压缩等操作时，公式应用的指标线等的显示也会进行变化。所以，建议在选定主/副图显示后校验一下公式应用为佳。

若公式应用里含带了 **buy**、**sellshort**、**buytocover**、**sell** 等发单指令的，则其中指令的开平仓讯号以及讯号间连线会始终标识在主图的 K 线上，其它的输出数据如注释信息、指标线、形态等则会根据设置显示在不同的主、副图位置上。

## ●公式管理器

公式管理器是对交易开拓者两类公式进行集中管理的模块，您可以在公式管理器中打开、修改和删除公式。点击面板“TB 公式”分组中“公式管理器”按钮，即可打开公式管理器。（面板可通过快捷“F3”进行显示与隐藏，若在主界面上最左边的位置上没有找到面板的话，可以尝试按下 F3，使其显示。）

公式管理器分页显示各类公式，通过点击标签页进行切换，如用户函数与公式应用。各分页里的字段包括：简称、名称、校验、系统及修改日期。通过校验的为“✓”，没有通过校验的为“✗”。用户自编的公式在系统里显示为“用户”，软件自带的则显示为“系统”。修改日期显示的是该公式最后一次被保存或是校验保存的日期与时间。

公式管理器包含以下功能：

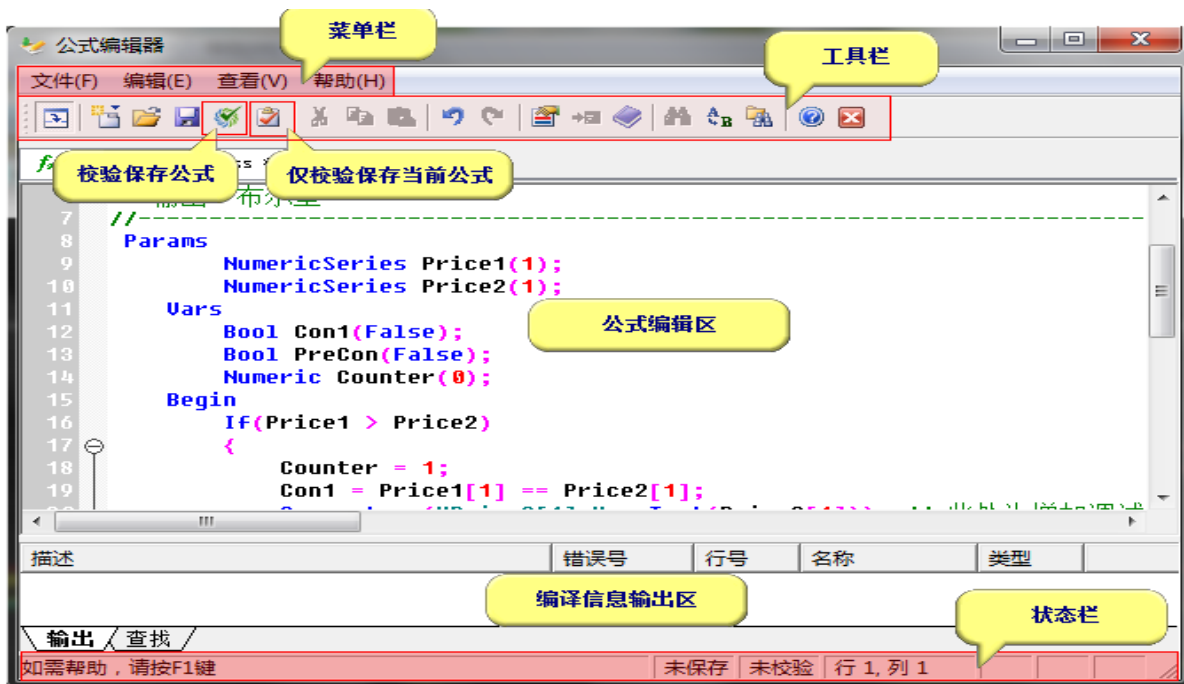
- **打开：**选择列表中一个或多个公式，点击“打开”按钮，可以对所选公式进行编辑；
- **删除：**选择列表中一个或多个公式，点击“删除”按钮，可以删除所选公式，系统公式不能被删除；
- **属性：**选择列表中一个公式，点击“属性”按钮，可以打开公式属性，对其修改；
- **关闭：**关闭公式管理器；
- **帮助：**打开公式管理器帮助页。

公式管理器同时支持按公式名称、简称、分类或修改时间等属性进行排序。

一旦安装好交易开拓者软件，公式管理器里便存放了一定数量的用户函数与公式应用。您可以先尝试打开这些系统自带的公式代码学习其编写方法，或是直接将公式应用调用入图表上，进行指标的分析或是交易策略的测试。

## 公式编辑器

公式编辑器是对 TradeBlazer 进行编辑、编译的模块，公式编辑器的界面如下：



公式编辑器的功能列表如下(按工具栏按钮顺序):

- **显示或隐藏输出窗：**显示或隐藏信息输出区；
- **新建公式：**新建 2 种类型的公式，打开新建向导；
- **打开公式：**打开公式管理器，调入要打开的公式；
- **保存公式：**保存当前公式的代码，没有编译，不可以调用；
- **校验保存公式：**编译保存当前公式，系统公式不能修改或对其进行编译保存；
- **仅校验保存当前公式：**该按钮仅对用户函数有效，编译保存当前公式，系统公式不能修改，也不能对其进行编译保存；
- **剪切：**如标准的 Windows 操作，剪切选中的文本；
- **复制：**如标准的 Windows 操作，复制选中的文本；
- **粘贴：**如标准的 Windows 操作，粘贴剪贴板中的文本；
- **删除：**如标准的 Windows 操作，删除选中的文本；
- **撤销：**如标准的 Windows 操作，撤销上一部操作；
- **重复：**如标准的 Windows 操作，重复撤销的操作；
- **属性设置：**打开该公式的属性设置对话框，详细情况参见公式属性；

- **打开该用户函数：** 如果选中的字符串是已经存在的用户函数，该按钮会变成有效，通过点击该按钮可打开该用户函数；
- **系统函数：** 打开系统函数字典；
- **查找：** 在当前公式中查找；
- **替换：** 替换当前公式的某些内容；
- **全文搜索：** 在全部公式中查找；
- **帮助：** 打开公式编辑器帮助；
- **退出：** 退出公式编辑器

双击信息输出区列表项，可直接定位到对应公式的指定行。

另外，您还可以通过菜单或快捷键实现更多的功能，具体操作参见菜单项。

## 公式的属性

根据公式类型不同，每种公式具有不同的属性页面，详细情况参见下表：

公式类型	属性页面
用户函数	常规，参数，返回类型。
公式应用	常规，参数，线型，讯号，连线，报警。

界面如下图所示：



**常规：**该页面显示公式的基本信息，包括简称、名称、密码信息、分类和注释，所有公式都有这些基本信息。

**参数：**该页面显示公式的参数信息，在公式调用的时候，您可以直接通过修改列表框的参数值进行参数变更。还可以点击“设为默认”将该参数保存到公式内部，供其他地方调用。

**返回类型：**该页面显示用户函数的返回值类型，该信息必须要脚本中返回值类型保持一致，否则不能成功编译。

**讯号：**该页面显示公式的讯号显示设置，更多请查看讯号设置。

**连线：**该页面显示公式的讯号连线设置。

**报警：**该页面显示公式的报警设置，要使公式能够报警，必须勾选“启动报警”复选框，还可选择全局报警设置或用户报警设置，以及是否只报警一次。全局报警设置即消息中心的默认报警设置。

## 线型

该页面是针对技术分析在超级图表中的表现形式进行设置的页面。

对于公式应用，要设置一条输出线的显示风格，首先在线列表中选中该线，然后设置线型，再设置线的风格，最后设置粗细和颜色。如果不想某条线显示，还可以选择线型为隐藏。

您还可以点击“设为默认”将当前的设置保存到公式内部，供其他地方调用。

## 公式的加密

提供公式加密，可以对用户自编的公式代码进行加密设置。在公式编辑界面的属性里勾选“加密公式”并设置好密码，再按下确定，便完成了对该公式代码的加密。

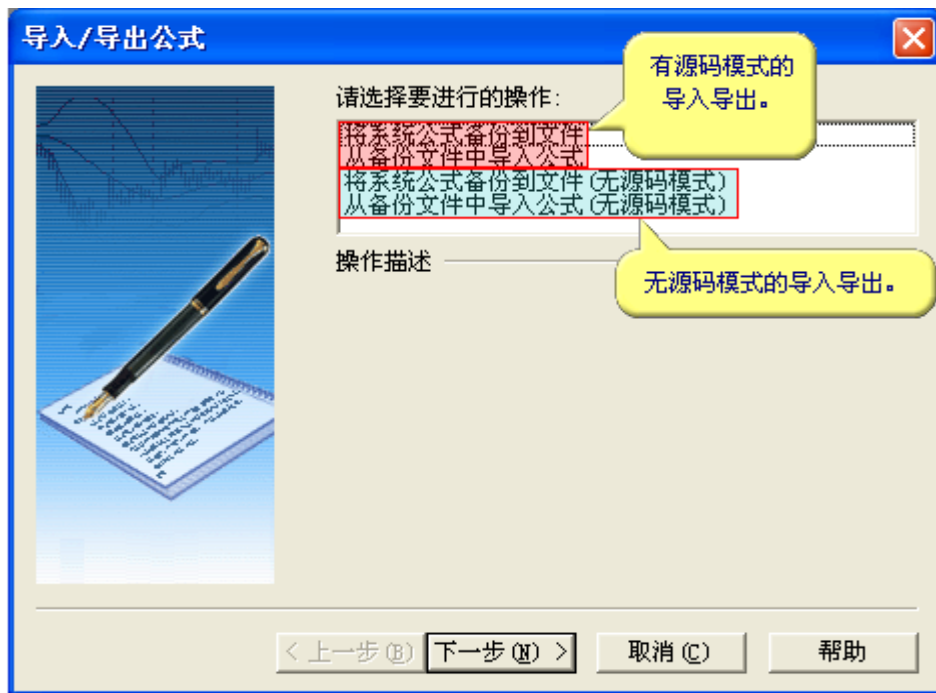
若再次打开此公式，则需输入正确的密码方可看到公式的代码。但是无密码也不影响对公式加载于超级图表上的正常使用。公式加密后导出，其特性与在本地机一样，需要密码方可打开，否则只能在超级图表上调用，而无法看到公式代码。

**注：**公式加密后可导出给另外的用户使用，在没有得知密码详情的情况下必须在导入时选择“导入编译”。否则将无法打开公式编辑界面并进行校验保存的操作，从而影响所导入公式的使用。

## 公式的导入与导出

公式导入导出功能提供将 TradeBlazer 公式打包成文件，并可在此还原；通过该功能，可实现用户之间公式的交互。

公式导入导出界面如下图所示：



公式导入/导出支持以下两种模式：

- 有源码模式：可以导入导出用户函数和公式应用，导入导出时包含源码，该公式包后缀名为(.fbk)。导入的用户可以看到公式应用的源代码，并且还可以对导入的代码进行修改、再编译等操作。
- 无源码模式：只能导入导出公式应用代码，导入导出时不带源码，该公式包后缀名为(.tbf)。

操作方法：

鼠标单击面板“TB 公式”组中的“公式导入/导出”按钮，即可打开公式导入/导出向导窗口：按照提示步骤有选择的分别完成：公式的导入（从备份文件中导入公式）/导出（将系统公式备份到文件）的操作：

## 公式导出

1. 鼠标单击选择要进行的操作：将系统公式备份到文件，然后按照向导提示进行下一步；
2. 从公式类型下拉列表中选择公式类型，有源码模式（所有公式/用户函数/公式应用），无源码模式（只能导出公式应用），通过鼠标单击选择，任选其一；
3. 从可选公式列表中选择要导出的公式添加到已选公式列表中，其中[>]按钮表示：将可选公式列表中被鼠标单击选中的公式添加到已选公式列表中，准备导出；[>>]按钮表示：将可选公式列表中的所有公式全部添加到已选公式列表中，准备导出；[<]按钮表示：将已选公式列表中被鼠标单击选中的公式放回到可选公式列表中，取消导出该公式；[<<]按钮表示：将已选公式列表中的所有公式全部放回到已选公式列表中，取消导出这些公式；您还可以通过双击列表项进行操作；
4. 选择公式文件的保存路径，通过鼠标单击[浏览]按钮打开浏览对话框，选择好公式文件的保存路径，输入公式文件名后单击[确定]按钮即可导出公式。

## 公式导入

1. 鼠标单击选择要进行的操作：从备份文件中导入公式，然后进行按照向导提示进行下一步；
2. 选择公式文件的保存路径，通过鼠标单击“浏览”按钮打开浏览对话框，选择要导入的文件，单击“下一步”按钮；
3. 选择要导入的公式分类，单击“下一步”按钮；
4. 选择要导入的公式列表，单击“完成”按钮，将会进行导入操作，该操作完成之后，整个导入过程完成。

**注意：**使用有源码导入导出时请将关联的用户函数一起选择，否则会导致导入不成功。使用无源码模式导入导出公式应用时，无需依赖用户函数，直接导入导出公式应用即可使用。

## 数据窗口

可选择显示或隐藏数据窗口。显示数据窗口可将当前图表的Bar数据或是指标输出的数据显示在窗口里。窗口内容的显示有三种可选模式，只显示当前K线数据，显示全部K线数据，显示全部K线及指标数据。



如下为数据窗口中的界面图：



时间	开盘价	最高价	最低价	收盘价	成交量	持仓量	MACD:MACD	MACD:MACDAvg
2011/06/01 14:56	3000.6	3000.8	3000.0	3000.6	168	37462	0.22	0.17
2011/06/01 14:57	3000.8	3002.2	3000.6	3001.4	695	37486	0.30	0.20
2011/06/01 14:58	3001.2	3001.6	3000.6	3001.4	227	37503	0.37	0.23
2011/06/01 14:59	3001.4	3001.6	3000.6	3001.0	254	37534	0.38	0.26
2011/06/01 15:00	3001.2	3001.2	3000.0	3000.0	446	37608	0.31	0.27
2011/06/01 15:01	3000.0	3001.2	3000.0	3001.2	328	37558	0.34	0.28

数据窗口还可提供将数据以CSV格式保存出来，以供在excel表格里的统计分析使用。

## 公式的报警

TradeBlazer 公式提供报警功能，您可以在公式应用中通过 **Alert** 函数来实现报警。

您可以在用户函数或公式应用中按照以下方式来编写自己的报警。例如：

Vars

```
Bool    Condition1;
```

Begin

```
Condition1 = ..... // 您设定的条件表达式;
```

```
If(AlertEnabled AND Condition1)
```

```
{
```

```
    Alert("报警信息...");
```

```
}
```

End

当公式编译保存成功之后，您可以将其应用在超级图表中，通过报警属性页启动报警。当条件满足之后，将会产生报警信息，并发送到消息中心。

# ●TradeBlazer 公式的语法基础

## 保留字

保留字都有自己独特的意思或用途，主要是一些功能关键字、系统函数以及数据类型等。在公式的简称以及参数变量等的命名时，要避免使用保留字。

下面分类列举出系统主要的保留字：

操作符

类型	保留字
算术操作符	+ - * / % ^
关系操作符	>>= < <= == != <>
逻辑操作符	AND/&& OR/   NOT/!
括号	(){} []
其它符号	.,;

有关各个操作符的具体说明与用法在后面的章节有详细的介绍。

功能关键字

保留字	说明
<b>Params</b>	用该关键字宣告参数定义的起始，参数必须填写默认值。
<b>Vars</b>	用该关键字宣告变量定义的起始(可以赋初值)，变量不填写初值时，系统将自动为其填充初值。
<b>If</b>	条件语句。
<b>Else</b>	条件语句。

<b>Begin</b>	用该关键字宣告程序主体的起始。
<b>End</b>	用该关键字宣告程序主体的结束。
<b>For</b>	循环语句。
<b>To</b>	循环语句。
<b>DownTo</b>	循环语句。
<b>While</b>	循环语句。
<b>Break</b>	循环语句。
<b>Continue</b>	循环语句。
<b>True</b>	真。
<b>False</b>	假。

## 数据源

保留字	说明
<b>Data0-Data49</b>	支持 50 个数据源。

## 数据输出

保留字	说明
<b>PlotBool</b>	输出布尔型值。
<b>PlotNumeric</b>	输出数值型值。
<b>PlotString</b>	输出字符串值。
<b>UnPlot</b>	取消指定位置的输出。
<b>Alert</b>	报警输出。
<b>Buy</b>	多头建仓操作。

<b>Sell</b>	多头平仓操作。
<b>SellShort</b>	空头建仓操作。
<b>BuyToCover</b>	空头平仓操作。
...	其他系统函数。

## 标点符号

通常，在写语句的过程中，会用到很多的标点符号。可用来定义参数、定义变量、创建规则的优先权。

例如，TradeBlazer 公式用";"来标注一个语句结束。标点符号也是一个保留字，因为符号也是语言结构的一部分，在下表中列出了 TradeBlazer 公式中所用到的标点符号，和该标点符号所表达的意思：

符号	名称	说明
;	分号	语句结束的标志。
,	逗号	当函数带有多个参数时，用于分隔多个参数。
()	小括号	括号之内的表达式有计算的优先权。
""	双引号	字符串常量。
[]	中括号	回溯数据，正数引用以前的数据，负数引用未来数据
{ }	大括号	控制语句的起始与结束。
.	点	扩展数据源的数据调用。

## 操作符号

操作符是一些象征具体操作运算行为的符号，例如操作符"+"代表对两个数求和，这些操作符适用于数值型、字符串、布尔型的数据。

TradeBlazer 公式为您提供了多种操作运算符，便于您对保留字的操作和生成更复杂的数据类型、逻辑型、字符串类型的值。下面有四种不同类型的操作符可用于逻辑表达式、数值表达式、字符串表达式中。

## 数学操作符号

数值型表达式的操作符有几种，如下表所示：

操作符	说明
+	加
-	减
*	乘
/	除
%	求模
()	括号

这些数学操作按其特定的优先级来进行计算，"\*"(乘法)最先，其次是"/"(除法)和"%(求模)，"+"(加法)和"-"(减法)最后，如果有多个乘法/除法(或者是加法/减法)，那么计算顺序是从左边到右边。

例如，在数值型的表达式中：

High+2\*range/2;

它首先计算的是 **range**(此处 **range** 是指 **High-Low**)与 2 的积, 接着计算与 2 的商(除法), 最后求  $2 * \text{range} / 2$  与最高价(**High**)的和。

如果要找到一个 **Bar** 的中间位置, 可以尝试写成如下语句。

例如:  $(\text{High} + \text{Low}) / 2$ ;

## 关系操作符号

逻辑运算符使用下列标准的比较符号, 大于、小于、等于、小于等于、大于等于和不等。

下列的关系操作符号都可以应用到逻辑表达式中。

操作符	说明
<	小于
>	大于
<=	小于等于
>=	大于等于
<>或!=	不等于
==	等于

应用上述的关系运算符, 我们可以对两个数值或字符串表达式进行对比, 在下列的语句中, 我们就是找到一个 **Bar**, 它的当前 **Bar** 收盘价要高于前一个 **Bar** 最高价。

例如: `Close > High[1];`

在字符串的比较运算中, 首先是把每一个字符用它的 **ASCII** 来代替, 其次对两个表达式中的字符逐一比较其 **ASCII** 值, 从第一个开始, 直到两个表达式中的所有字符都已经被计算完为止。

例如: `"abcd" < "zyxw";`

在这个例子中, 我们对把第一个字符串表达式中的字符和第二个表达式中的字符进行比较运算, 字母"a"的 **ASCII** 值是小于"z"的, 同样其它的字符也是一样, 所以该表达式的值为 **True**。

## 逻辑关系操作符号

逻辑运算符常常用于比较两个 True/False 的表达式，共有三个逻辑操作符：AND(&&)，OR(||)，NOT(!)。

下表列出 AND 逻辑操作符的应用情况：

表达式 1	表达式 2	表达式 1 AND 表达式 2
True	True	True
True	False	False
False	True	False
False	False	False

下表列出 OR 逻辑操作符的应用情况：

表达式 1	表达式 2	表达式 1 OR 表达式 2
True	True	True
True	False	True
False	True	True
False	False	False

下表列出 NOT 逻辑操作符的应用情况：

表达式 1	NOT 表达式 1
True	False
False	True

在上面的表格中，应用 OR 可以增加表达式的值为 True 的可能性，仅仅只要两个表达式中，只要有一个的值为 True，那么整个表达式的值就为 True。

其实在应用的过程中，还包含有一些复杂的组合运算。为了获得一个的关键反转 **Bar**，可以使用如下的表达式：

**Low < Low[1] AND Close > High[1];**

在上面的表达式中，我们使用了 **AND** 逻辑运算符，因而要表达式的值为 **True**，那么当前 **Bar** 的最低价一定要小于前一个 **Bar** 的最低价，而且当前 **Bar** 的收盘价还必须高于前一个 **Bar** 的最高价。只有当这两个条件都满足的时候，表达式的值才为 **True**。

再看下面一个例子：

**High > 10 OR Vol > 5000;**

在上面的表达式中，如果要其值为 **True**，那么只需要任意一个条件满足即值为 **True**，那么表达式的值便为 **True**，如果当前 **bar** 的最高价大于 10，或者成交量大于 5000，那么表达式的值便为 **True**。而如果需要表达式的值为 **False** 时，则两个条件都必须为 **False**，表达式的值才为 **False**。

逻辑操作符的优先级低于数学操作符和关系操作符。逻辑操作符也遵循先括号的原则，如果没有括号，那么其运算顺序也是从左边到右边。

对于逻辑表达式中不同条件的先后顺序，可能会产生不同的运算逻辑，执行的效率也会有所不同。

以 **Con1 AND Con2** 这样的表达式举例，系统从左到右进行逻辑判断，当 **Con1** 为 **True** 时，需要继续判断 **Con2** 是否为 **True**，只有当 **Con1**，**Con2** 都为 **True** 时，整个表达式才为 **True**。但是只要当 **Con1** 为 **False** 时，就不再需要判断 **Con2** 的值，而是直接返回 **False**。

因此，以下的两个表达式在执行效率方面是有差异的：

**5 < 4 AND Close > Open;**

**Close > Open AND 5 < 4;**

第一条语句的执行速度大部分情况下都比第二条要快。

对于 **Con1 OR Con2** 表达式，情况也比较类似，当 **Con1** 为 **False** 时，需要继续判断 **Con2** 是否为 **False**，只有当 **Con1**，**Con2** 都为 **False** 时，整个表达式才为 **False**。但是只要当 **Con1** 为 **True** 时，就不再需要判断 **Con2** 的值，而是直接返回 **True**。



以下两条语句的执行效率也是不一样的：

`5 > 4 OR Close > Open;`

`Close > Open OR 5 > 4;`

通过上述的说明，我们应该知道，逻辑表达式的组合时，应该尽可能的把容易判别整个表达式逻辑的条件放在前面，以减少整个表达式的计算时间。

## 系统函数

TradeBlazer 公式的系统函数，可根据使用范围在相应类型的公式中直接调用，计算后返回结果值。

目前的系统函数支持四种数据类型，除了 TradeBlazer 公式中定义的三种基本数据类型：**Bool**，**Numeric**，**String** 之外，新加入 **Long**（长整型）类型，使系统函数能够更加快捷的进行计算，TradeBlazer 公式在处理的时候自动将 **Numeric** 和 **Long** 进行转化，用户无需进行特别的处理。

TradeBlazer 公式现有的系统函数主要分为：数据函数、时间函数、数学函数、其它函数、交易函数、属性函数、帐户函数、颜色函数、字符串函数等。每个系统函数都有自己的适用范围和使用规范，详细说明参见软件联机帮助文档（按键 **F1**）里的附录。

## ●TradeBlazer 公式的语句

### 声明

根据需要在公式的前端，对参数或是变量进行的声明。对于参数与变量的声明同样需要参照命名规则。声明的顺序是先参数后变量。

### 参数

参数是一个预先声明的地址，用来存放输入参数的值，在声明之后，您就可以在接下来的公式中使用该参数的名称来引用其值。

参数的值在公式的内部是不能够被修改，在整个程序中一直保持不变，不能对参数进行赋值操作(引用参数是个特例)。参数的好处在于您可以在调用公式应用的时候才指定相应的参数，而不需要重新编译。

例如，我们常用的移动平均线指标，就是通过不同的 **Length** 来控制移动平均线的周期，在调用指标时可以随意修改各个 **Length** 的值，使之能够计算出相对应的移动平均线。您可以指定 4 个参数为 5,10,20,30 计算出这 4 条移动平均线，也可以修改 4 个参数为 10, 22, 100, 250 计算出另外的 4 条移动平均线。

参数的修改很简单，在超级图表调用指标的过程中，您可以打开指标的属性设置框，切换到参数页面，手动修改各项参数的值，然后应用即可，交易开拓者将根据新的参数设置计算出新的结果，在超级图表中反映出来。

另外，参数的一个额外的优点是，我们可以通过修改公式应用不同的参数，测试交易策略的性能优劣，达到优化参数的目的。交易开拓者提供自动计算优化参数的功能。具体详情请参考后面章节的参数优化。

## 参数的类型

在介绍参数类型之前，我们需要对于 TradeBlazer 公式的公式类型作一些说明，用户函数是公式中比较特殊的类型，它自身不能被超级图表，行情报价这样的模块调用，只能被公式应用或者用户函数调用，因此它的参数类型也和公式应用不一样。

用户函数的参数类型可以包含 TradeBlazer 公式的九种类型，而公式应用只能使用三种简单的基本类型，即数值型（numeric）、布尔型(bool)和字符串型(string)。

三种简单类型参数通过传值的方式将参数值传入公式，公式内部通过使用参数名称，将参数值用来进行计算或赋值。

引用参数是在调用的时候传入一个变量的地址，在用户函数内部会修改参数的值，在函数执行完毕，上层调用的公式会通过变量获得修改后的值，引用参数对于需要通过用户函数返回多个值的情况非常有用。

### 参数的声明

在使用参数之前，必须对参数进行声明，TradeBlazer 公式使用关键字"Params"来进行参数宣告，并指定参数类型。可以选择赋默认值，也可以不赋默认值。如果某个参数没有赋予默认值，则这个参数之前的其他参数的默认值都将被忽略。

参数定义的语法如下：

#### Params

参数类型 参数名 1(初值);

参数类型 参数名 2(初值);

参数类型 参数名 3(初值);

下面是一些参数定义的例子：

#### Params

Bool            bTest(False);        //定义布尔型参数 bTest，默认值为 False;

Numeric        Length(10);        //定义数值型参数 Length，默认值为 10;

NumericSeries   Price(0);        //定义数值型序列参数 Price，默认值为 0;

NumericRef      output(0);        //定义数值型引用参数 output，默认值为 0;

String          strTmp("Hello");    //定义字符串参数 strTmp,默认值为 Hello;

整个公式中只能出现一个 **Params** 宣告，并且要放到公式的开始部分，在变量定义之前。

## 参数的默认值

在声明参数时，通常会赋给参数一个默认值。例如上例中的 **False**，**10**，**0** 等就是参数的默认值。用户函数的默认值是在当用户函数被其他公式调用，省略参数时作为参数的输入值，其他五种公式的默认值是用于图表，报价等模块调用公式时默认的输入值。

参数的默认值的类型在定义的时候指定，默认值在公式调用的时候传入作为参数进行计算。只能够对排列在后面的那些参数提供默认参数。

例如：

**Params**

```
Numeric MyVal1;  
Numeric MyVal2(0);  
Numeric MyVal3(0);
```

不能够使用以下方式对参数的默认值进行设定：

**Params**

```
Numeric MyVal1(0);  
Numeric MyVal2(0);  
Numeric MyVal3;
```

为防止因人为疏忽而导致的参数默认值错误，建议习惯将所有的参数都设上初值。

## 变量

变量是一个存储值的地址，当变量被声明之后，就可以在脚本中使用变量，可以对其赋值，也可以在其他地方引用变量的值进行计算，要对变量进行操作，直接使用变量名称即可。

变量的主要用处在于它可以存放计算或比较的结果，以方便在之后的脚本中直接引用运算的值，而无需重现计算过程。

例如，我们定义一个变量 Y，我们把一个收盘价(Close)乘上 8% 的所得的值存储在 Y 中，即  $Y = \text{Close} * 8\%$ 。那么一旦计算出  $\text{Close} * 8\%$  的值，便赋给变量 Y。而无需在公式中输入计算过程，只需调用变量名称即可引用变量的值。

变量有助于程序的优化，有时 TradeBlazer 公式必须重复调用一些数据，这些数据可能是某些函数（如：Bar 数据，close、vol 等），或通过表达式执行计算和比较的结果。因此，在表达式频繁使用的地方使用变量可提高程序的运行速度和节约内存空间。

使用变量也可以避免输入错误，使程序的可读性提高，示例如下：

```
If(Close > High[1] + Average(Close,10)*0.5)
{
    Buy(100, High[1] + Average(Close,10)*0.5);
}
```

如果使用变量，则整个代码变得简洁：

```
Value1 = High[1] + Average(Close,10)*0.5;
If (Close > Value1)
{
    Buy(100,Value1);
}
```

如果一些表达式的组合经常在不同的公式中被调用，这个时候变量就不能实现功能，变量只能在单个公式的内部使用，这个时候我们需要建立用户函数来完成这些功能，详细说明参见用户函数。

## 变量的类型

TradeBlazer 公式支持九种数据类型，但对于变量定义，引用类型是无效的，剩余六种数据类型中分为简单和序列两大类，简单类型变量是单个的值，不能对其进行回溯，序列类型变量是和 Bar 长度一致的数据排列，我们可以通过回溯来获取当前 Bar 以前的任意值。

## 变量的声明

在使用变量之前，必须对变量进行声明，TradeBlazer 公式使用关键字“Vars”来进行变量宣告，并指定变量类型。可以选择赋默认值，也可以不赋默认值。

变量定义的语法如下：

### Vars

变量类型 变量名 1(初值);

变量类型 变量名 2(初值);

变量类型 变量名 3(初值);

下面是一些变量定义的例子：

### Vars

NumericSeries MyVal1(0); //定义数值型序列变量 MyVal1，默认值为 0;

Numeric MyVal2(0); //定义数值型变量 MyVal2，默认值为 0;

Bool MyVal3(False); //定义布尔型变量 MyVal3，默认值为 False;

String MyVal4("Test"); //定义字符串变量 MyVal4，默认值为 Test。

整个公式中只能出现一个 Vars 宣告，并且要放到公式的开始部分，在参数定义之后，正文之前。

## 变量的赋值

变量声明完成之后，您可以在脚本正文中给变量指定一个值。

语法如下：

Name = Expression;

"Name"是变量的名称，表达式的类型可以是数值型、布尔型、字符串中的任何一种。不过表达式的类型一定要和变量的数据类型相匹配。如果变量被指定为是数值型的，那么表达式一定要是数值型的表达式。

例如：下面的语句将 Close 的 10 周期平均值赋值给变量 Value1：

Value1 = Average(Close , 10);

在下面这个语句中，声明了一个名为"KeyReversal"的逻辑型变量，然后又把计算的值赋给它。

```
Vars
    Bool      KeyReversal(False);
Begin
    KeyReversal = Low < Low[1] AND Close > High[1];
    ...
End
```

## 变量的使用

变量定义、赋值之后，在表达式中直接使用变量名就可以引用变量的值。例如在下面的语句中计算了买入价格后，把值赋给数值型变量 **EntryPrc**，在买入指令中便可直接应用变量名，通过变量名便可引用变量的值：

```
Vars
    Numeric EntryPrc(0);
Begin
    EntryPrc = Highest(High,10);
    If (MarkerPosition <> 1)
    {
        Buy(1,EntryPrc);
    }
End
```

接下来的例子，我们计算最近 10 个 **Bar** 最高价中的最大值(不包括当前 **Bar**)，对比当前 **High**，然后通过 **If** 语句，产生报警信息。

Vars

```
    Bool    Con1(False);
```

Begin

```
    Con1 = High > Highest(High,10)[1];
```

```
    If(Con1)
```

```
    {
```

```
        Alert("New 10-bar high");
```

```
    }
```

End

其实我们并不一定都要应用条件为 **True** 的情况，有时候我们需要判断条件为 **False** 的时候执行某些代码，如下的例子：

Vars

```
    Bool    Con1(False);
```

Begin

```
    Con1 = High < Highest(High,10)[1] AND Low > Lowest(Low,10)[1];
```

```
    If(Con1==False)
```

```
    {
```

```
        Alert("New high or low");
```

```
    }
```

End

## 序列变量

序列变量是变量中的一种，可以对序列变量进行回溯获取以前 **Bar** 的变量数据。序列变量的声明和简单变量一样，只是定义的数据类型不同，您必须选择以下的 **3** 种类型来定义序列变量：

**NumericSeries / BoolSeries / StringSeries。**



例如：

Vars

```
NumericSeries MyNumSVal(0);  
BoolSeries MyBoolVal(False);  
StringSeries MyStrVal("");
```

序列变量和简单变量一样，可以对其赋予默认值。

序列变量定义之后，您可以象简单变量一样的对其使用，不会有任何的不同。除了支持全部简单变量的功能之外，序列变量还可以通过"[nOffset]"来回溯以前的变量值。

对于序列变量，TradeBlazer 公式在内部针对其回溯的特性作了很多的特殊处理，也需要为序列变量保存相应的历史数据，因此，和简单变量相比，执行的速度和占用内存空间方面都作了一些牺牲。因此，尽管您可以定义一个序列变量，把它当作简单变量来使用，但是，我们强烈建议您只将需要进行回溯的变量定义为序列变量。

在指定条件下对某变量赋值，序列变量的会自动传递上一个 Bar 的变量值，直至语句对于进行新的赋值。而如果是普通变量，则只在条件满足的 Bar 上会有指定的赋值，其它 Bar 上则仍会记录初始值。

比如，我们现在分别定义一个变量 **aaa**，以及一个序列变量 **bbb**，再对这两个变量进行同样的赋值。公式代码如下：

```
If(CrossOver(ma1,ma2))  
{  
    aaa = 1;  
    bbb = 1;  
}else if (CrossUnder(ma1,ma2))  
{  
    aaa = -1;  
    bbb = -1;  
}
```



如上图所示 **aaa**、**bbb** 这两个变量的赋值虽然是一样的条件，但其结果有所不同。可以看到，在均线上穿时，**aaa** 与 **bbb** 都是为 1，均线下穿时，**aaa** 与 **bbb** 都是为-1。不同的是，在除去上下穿的 **Bar** 上，**aaa** 与 **bbb** 的值则有所差异。变量 **aaa** 得出的都是默认值“0”，而序列变量 **bbb** 则是传递着上一个 **Bar** 上此变量的值，直到条件改变此值。且，**aaa** 做为普通变量不可以使用回溯取值。而做为序列变量的 **bbb** 则可以回溯取值，如：**bbb[5]**。

## 全局变量

目前系统在单个公式应用中可提供最多 500 个全局变量，全局变量的索引值从 0 开始计数到 499，不能大于 500。

全局变量的初始值为无效值，与普通变量不同，其值不会因为当前 **Bar** 的变化而变化。只有在对全局变量进行赋值之后，其值才会被改变以及保存。

全局变量附着在超级图表上，一旦关掉超级图表之后，所有保存的值将会部被删除。在图表上进行数据刷新的动作，也会导致全局变量的值跟随刷新的计算而重新赋值。

用户自行通过 **SetGlobalVar()** 进行保存以及 **GetGlobalVar()** 来获取数据的操作。

例如：

**SetGlobalVar(1,myprice);** 将第 2 个全局变量设置为自定义的变量“myprice”。

**Val = GetGlobalVar(0);** 将第一个全局变量值取出来赋值给 **Val**。

下面，我们会用一个全局变量所编写的 **Tick** 计数器。掌握此例，有助于交易者理解全局变量的运行机制并能更好的学习及使用全局变量。

新建一个公式应用：

Vars

NumericSeries TickCnt;

Numeric bartime;

Begin

bartime = GetGlobalVar(0);

if (bartime == InvalidNumeric)

{

bartime = 0;

SetGlobalVar(0,bartime);

TickCnt = 1;

SetGlobalVar(1, TickCnt);

FileAppend("d:\\Sample\_13.log", "Bartime = "+DateTimeToString(date+time)  
+"\\t 计数器初始化, Global(0) = "+text(bartime)+"\\t Global(1) = "+Text(TickCnt));

}

if (Date+Time > bartime)

{

bartime = Date + Time;

SetGlobalVar(0,bartime);

TickCnt = 1;

SetGlobalVar(1, TickCnt);

FileAppend("d:\\Sample\_13.log", "Bartime = "+DateTimeToString(date+time)+"\\t 新 K 线  
产生, Global(0) = "+DateTimeToString(bartime)+"\\t Global(1) = "+Text(TickCnt));

} Else If (Date+Time==bartime)

{

TickCnt = GetGlobalVar(1) + 1;

SetGlobalVar(1, TickCnt);

FileAppend("d:\\Sample\_13.log", "Bartime = "+DateTimeToString(date+time)+"\\t 原 K 线  
增加计数, Global(0) = "+DateTimeToString(bartime)+"\\t Global(1) = "+Text(TickCnt));

}

Commentary("TickCnt="+text(TickCnt));

End

## 赋值语句

赋值语句用于给公式变量指定一个具体的值的语句，赋值语句使用赋值操作符(=)进行处理。

以下为赋值语句的一些例子：

Vars

    Bool b;

Begin

    B = true;

    ...

End

Vars

    Numeric Value1;

Begin

    Value1 = (Close + Open)/2;

    ...

End

变量在赋值的时候忽略其扩展数据类型，只考虑其基本数据类型，即 **NumericSeries**，**NumericRef**，**Numeric** 之间可以相互赋值。此时序列数据类型只是对当前 **Bar** 的值进行操作。

以下的写法是错误的：

Vars

    NumericSeries Value2;

Begin

    Value2[1] = (Close + Open)/2;

    ...

End

当我们为某变量进行声明与赋值之后，在公式的后段便可以对该变量进行记录、判断或是输出显示了。

## 控制语句

TradeBlazer 公式支持两大类的控制语句：条件语句和循环语句

### 条件语句

条件语句包括以下四类表达方式：

#### if....语句

If 语句是一个条件语句，当特定的条件满足后执行一部分操作。

语法如下：

If (Condition)

{

TradeBlazer 公式语句;

}

Condition 是一个逻辑表达式，当 Condition 为 True 的时候，TradeBlazer 公式语句将会被执行，Condition 可以是多个条件表达式的逻辑组合，Condition 必须用()括起来。

TradeBlazer 公式语句是一些语句的组合，如果 TradeBlazer 公式语句是单条，您可以省略{}，二条或者二条以上的语句必须使用{}。

例如，您可以计算图表中上升缺口（当前 Bar 的开盘价高于上一个 Bar 的最高价）出现了多少次，只要在图表中使用 If 语句，当找到一个满足条件的 Bar 时，即条件为真时，变量加 1，脚本如下：

Vars

NumericSeries Counter(0);

Begin

If ( Open > High[1])

{

Counter = Counter[1] + 1;

...

}

```
...  
End
```

在 TradeBlazer 公式中，If 语句被广泛使用，当条件满足的时候，在满足条件的 Bar 上面进行标记。例如，下面的语句就是公式应用的例子：

```
If(High > High[1] AND Low < Low[1])  
{  
    PlotNumeric("Outside Bar",High);  
}
```

If 语句在不是用括号的情况，只执行下面的第一条语句，如下的语句，Alert 不会只在条件为 True 时执行，而是每次都执行。

```
If(High > High[1] AND Low < Low[1])  
    PlotNumeric("Outside Bar",High);  
    Alert("Outside Bar");
```

要想 Alert 只在条件为 True 时执行，您需要按照下面的格式编写：

```
If(High > High[1] AND Low < Low[1])  
{  
    PlotNumeric("Outside Bar",High);  
    Alert("Outside Bar");  
}
```

**注意：**编写代码时要注意在 if( )语句的后面一定不要加上“;”，否则会导致公式与所想表达的逻辑不符合，从而影响最终的计算果。

## if...else...语句

If-Else 语句是对指定条件进行判断，如果条件满足执行 If 后的语句。否则执行 Else 后面的语句。

语法如下：

```
If (Condition)  
{
```

```

    TradeBlazer 公式语句 1;
}Else
{
    TradeBlazer 公式语句 2;
}

```

**Condition** 是一个逻辑表达式，当 **Condition** 为 **True** 的时候，**TradeBlazer** 公式语句 1 将会被执行；**Condition** 为 **False** 时，**TradeBlazer** 公式语句 2 将会被执行。**Condition** 可以是多个条件表达式的逻辑组合，**Condition** 必须用()**括起来**。

**TradeBlazer** 公式语句是一些语句的组合，如果 **TradeBlazer** 公式语句是单条，您可以省略**{}**，二条或者二条以上的语句必须使用**{}**。

例如，比较当前 **Bar** 和上一个 **Bar** 的收盘价，如果 **Close > Close[1]**，**Value1 = Value1 + Vol**；否则 **Value1 = Value1 - Vol**，脚本如下：

```

If (Close > Close[1])
    Value1 = Value1 + Vol;
Else
    Value1 = Value1 - Vol;

```

## if...else...if....语句

**If-Else-If** 是在 **If-Else** 的基础上进行扩展，支持条件的多重分支。

语法如下：

```

If (Condition1)
{
    TradeBlazer 公式语句 1;
}Else If(Condition2)
{
    TradeBlazer 公式语句 2;
}Else
{

```

```
TradeBlazer 公式语句 3;  
}
```

**Condition1** 是一个逻辑表达式，当 **Condition1** 为 **True** 的时候，TradeBlazer 公式语句 1 将会被执行，**Condition1** 为 **False** 时，将会继续判断 **Condition2** 的值，当 **Condition2** 为 **True** 时，TradeBlazer 公式语句 2 将会被执行。**Condition2** 为 **False** 时，TradeBlazer 公式语句 3 将会被执行。**Condition1**，**Condition2** 可以是多个条件表达式的逻辑组合，条件表达式必须用()括起来。

TradeBlazer 公式语句是一些语句的组合，如果 TradeBlazer 公式语句是单条，您可以省略{}，二条或者二条以上的语句必须使用{}。

If-Else-If 的语句可以根据需要一直扩展，在最后的 **Else** 之后再加 **If(Condition)**和新的执行代码即可。当然您也可以省略最后的 **Else** 分支，语法如下：

```
If (Condition1)  
{  
    TradeBlazer 公式语句 1;  
}Else If(Condition2)  
{  
    TradeBlazer 公式语句 2;  
}
```

## If-Else 的嵌套

If-Else 的嵌套是在 If-Else 的执行语句中包含新的条件语句，即一个条件被包含在另一个条件中。

语法如下：

```
If (Condition1)  
{  
    If (Condition2)  
    {  
        TradeBlazer 公式语句 1;  
    }Else  
    {
```



```
TradeBlazer 公式语句 2;

}
}Else
{
    If (Condition3)
    {
        TradeBlazer 公式语句 3;
    }Else
    {
        TradeBlazer 公式语句 4;
    }
}
```

Condition1 是一个逻辑表达式, 当 Condition1 为 True 的时候, 将会继续判断 Condition2 的值, 当 Condition2 为 True 时, TradeBlazer 公式语句 1 将会被执行。Condition2 为 False 时, TradeBlazer 公式语句 2 将会被执行。当 Condition1 为 False 的时候, 将会继续判断 Condition3 的值, 当 Condition3 为 True 时, TradeBlazer 公式语句 3 将会被执行。Condition3 为 False 时, TradeBlazer 公式语句 4 将会被执行。Condition1, Condition2, Condition3 可以是多个条件表达式的逻辑组合, 条件表达式必须用()括起来。

TradeBlazer 公式语句是一些语句的组合, 如果 TradeBlazer 公式语句是单条, 您可以省略{}, 二条或者二条以上的语句必须使用{}。

例如, 在一个公式应用中, 条件设置如下: 当前行情上涨的时候, 如果收盘价高于开盘价时, 则产生一个以收盘价买入 1 张合约; 否则产生一个以开盘价买入 1 张合约。当前行情没有上涨的时候, 如果收盘价高于开盘价, 则产生一个以收盘价卖出 1 张合约; 否则产生一个以开盘价卖出 1 张合约。脚本如下:

```
If (Open > High[1])
{
    If (Close>Open)
    {
        Buy(1, Close);
    }Else
    {
        Buy(1, Open);
    }
}
```

```
    }  
  }Else  
{  
    If (Close > Open)  
    {  
      Sell (1,Open);  
    }Else  
    {  
      Sell (1,Close);  
    }  
}
```

## 循环语句

循环语句包括两种表达方式：**For** 和 **While**。

### for 循环

**For** 语句是一个循环语句，重复执行某项操作，直到循环结束。

语法如下：

```
For 循环变量 = 初始值 To 结束值  
{  
    TradeBlazer 公式语句;  
}
```

循环变量为在之前已经定义的一个数值型变量，**For** 循环的执行是从循环变量从初始值到结束值，按照步长为 1 递增，依次执行 **TradeBlazer** 公式语句。结束值必须大于或等于初始值才有意义，初始值和结束值可以使用浮点数，但是在执行过程中会被直接取整。只计算其整数部分。

**TradeBlazer** 公式语句是一些语句的组合，如果 **TradeBlazer** 公式语句是单条，您可以省略{}，二条或者二条以上的语句必须使用{ }。

第一次执行时，首先将循环变量赋值为初始值，然后判断循环变量是否小于等于结束值，如果满足条件，则执行 **TradeBlazer** 公式语句，同时循环变量加 1。接着重新判断循环变量是否小于等于结束值，一直到条件为 **False**，退出循环。

例如，以下的用户计算 **Price** 最近 **Length** 周期的和。

**Params**

NumericSeries Price(1);

Numeric Length(10);

**Vars**

Numeric SumValue(0);

Numeric i;

**Begin**

for i = 0 to Length - 1

{

SumValue = SumValue + Price[i];

}

Return SumValue;

**End**

如果希望 **For** 语句从大到小进行循环，可以使用以下的语法：

**For** 循环变量 = 初始值 **DownTo** 结束值

{

TradeBlazer 公式语句;

}

**For-DownTo** 让循环变量从结束值每次递减 1 直到等于结束值，依次调用 **TradeBlazer** 公式语句执行，初始值必须大于或等于结束值才有意义。

**For** 语句是比较常用的一种循环控制语句，它应用于知道循环次数的地方，很多内建用户函数中都使用 **For** 语句来完成相应的功能，比如 **Summation**, **Highest**, **Lowest**, **LinearReg** 等。

## while 循环

While 语句在条件为真的时候重复执行某一项操作。即，只要条件表达式的值为真(True)时，就重复执行某个动作。直到行情信息改变以致条件为假(False)时，循环才结束。

语法如下：

```
While (Condition)
{
    TradeBlazer 公式语句;
}
```

Condition 是一个逻辑表达式，当 Condition 为 True 的时候，TradeBlazer 公式语句将会被循环执行，Condition 可以是多个条件表达式的逻辑组合，Condition 必须用()括起来。

TradeBlazer 公式语句是一些语句的组合，如果 TradeBlazer 公式语句是单条，您可以省略{}，二条或者二条以上的语句必须使用{}。

例如，以下的公式用来计算要产生大于 100000 成交量需要最近 Bar 的个数：

Vars

```
Numeric    SumVolume(0);
Numeric    Counter (0);
```

Begin

```
While (SumVolume < 100000)
{
    SumVolume = SumVolume + Vol[Counter];
    Counter = Counter + 1;
}
```

End

首先，我们定义两个变量 SumVolume 和 Counter，并将其默认值设为 0。当 SumVolume < 100000 这个表达式为 True 时，While 内的 TradeBlazer 公式语句一直被调用，将前 Counter 个 Bar 的 Vol 加到 SumVolume 中，当 SumVolume 大于等于 100000 时，退出循环。

## 死循环

在使用 **While** 循环的时候，有可能会遇到循环一直执行，永远不能退出的情况，这种情况我们称之为死循环，比如下面的语句：

```
While (True)
{
    TradeBlazer 公式语句;
}
```

在这种情况下，循环将一直执行，导致程序不能继续工作，在这种情况下，我们可以使用 **Break** 来跳出循环，详细情况参加下节。

死循环的出现，将会导致整个公式编译以及运算都会无休止地进行下去，所以公式代码的编写中一定要注意避免此类可能性的发生。

## Break

针对上节的例子，要想从死循环中跳出，我们可以在循环之中添加 **Break** 语句，如下：

```
While (True)
{
    TradeBlazer 公式语句;
    If (Condition)
        Break;
}
```

循环在每次执行后，都将判断 **Condition** 的值，当 **Condition** 为 **True** 时，则执行 **Break** 语句，跳出整个循环。

## Continue

有的时候在循环中，我们可能希望跳过后面的代码，进入下一次循环，在这种情况下，可以使用 **Continue** 语句来达到目的，如下：

```
While (Condition1)
{
    TradeBlazer 公式语句 1;
    If (Condition2)
        Continue;
    TradeBlazer 公式语句 2;
}
```

当 **Condition1** 满足时，循环被执行，在执行完 **TradeBlazer** 公式语句 1 后，将判断 **Condition2** 的值，当 **Condition2** 为 **True**，将跳过 **TradeBlazer** 公式语句 2，重新判断 **Condition1** 的值，进入下一次循环。否则将继续执行 **TradeBlazer** 公式语句 2。

## ●TradeBlazere 用户函数

用户函数是可以通过名称进行调用的一组语句的集合，用户函数返回一个值，这个值可以是 **Numeric**，**Bool**，**String** 三种类型中的任何一种。您可以在需要的任何地方调用用户函数来完成相应的功能。

例如，在 TradeBlazer 公式中经常使用的一个用户函数 **Summation**，**Summation** 通过输入 **Price** 序列数据，以及 **Length** 统计周期数，计算 **Price** 最近 **Length** 周期的和，每次用户需要进行求和计算的时候，都可以调用 **Summation** 代替冗长的求和代码，输入参数并获取返回值。

**Summation** 是 TradeBlazer 公式中一个比较简单的用户函数，TradeBlazer 公式提供了上百个内建用户函数，当然，您也可以编写您自己的用户函数。

用户函数通过参数传递输入数据，通过引用参数或返回值传递输出数据，以上例子中的 **Summation** 函数，在被调用的时候格式如下：

```
Value1 = Summation(Close,10);
```

在调用 **Summation** 的时候，需要根据定义时候的参数列表和顺序，输入相应的输入参数，有默认值的参数可以省略输入参数。

用户函数在交易开拓者中使用有如下规则：

- 支持九种类型的参数定义，支持指定参数默认值；
- 支持使用引用参数，可通过引用参数返回多个数据；
- 支持六种类型的变量定义，支持指定变量的默认值；
- 可以访问 **Data0-Data49** 个数据源的 **Bar** 数据；
- 可以访问行情数据、属性数据；
- 必须通过 **Return** 返回数据，返回数据类型为三种基本类型之一；
- 脚本中的返回数据类型必须和属性界面设置中一致；
- 用户函数之间可以相互调用，用户函数自身也可以递归调用；用户函数可以调用所有的系统函数，包括交易动作和技术分析输出。

## 用户函数的类型

用户函数按照返回值类型不同可以分为数值型(Numeric)，布尔型(Bool)，字符串(String)三种基本类型，三种类型用户函数在调用时需要将返回值赋予类型相同的变量。

按照用户函数属性不同，用户函数可以分为内建用户函数和其他用户函数两种，内建用户函数是交易开拓者提供的，用于支持公式系统运行的预置公式，您可以查看和调用内建用户函数，但是不能删除和修改内建公式。

按照用户函数的实现机制不同，用户函数可分为普通函数和序列函数。普通函数和其他语言的函数类似，输入参数，执行一段程序代码，返回需要的值。序列函数是输入参数或变量中有序列数据类型的数据类型的用户函数。

## 序列函数

序列函数是一种特殊的用户函数，当它的参数或变量中使用了序列数据，我们就称之为序列函数，序列数据作为普通计算机语言和 TB 语言的重要区别，是进行金融序列数据计算的核心。为了保证序列数据的正确计算，序列函数需要每个 Bar 都被调用，如果有些 Bar 没有调用序列函数，序列函数中的序列数据则是上一个 Bar 的值。除非是您的算法需要，否则建议不要在条件语句，条件语句的判断表达式，循环语句中使用序列函数。

## 使用内建用户函数

TradeBlazer 公式中提供上百个内建用户函数，一部分用户函数提供类似于求和，求平均，求线性回归等算法方面的功能，另外一些函数提供技术分析的一些算法，比如：AvgTrueRange，Momentum 等,这些用户函数用户辅助完成技术分析。

在创建自己的技术分析和交易系统时，如果需要自己写一些算法，您可以首先在用户函数中查找是否有相应的内建用户函数，尽可能的多使用内建用户函数，减少出错的可能。您也可以编写自己的算法，以供在技术分析和交易系统中使用。



## 用户函数的参数

大部分用户函数都需要接受输入的信息进行计算，这些输入的信息，我们称之为参数。关于用户函数参数的使用详细说明参见前面章节的“参数”。

## 如何编写用户函数

一个用户函数由三部分组成，参数定义，变量定义，脚本正文。

参数定义和变量定义部分在前面已经详细叙述过，脚本的正文部分将输入参数进行计算，得出函数的返回值，并通过 **Return** 返回。

例如，我们以 **Average** 为例，**Average** 计算 **Price** 在 **Length** 周期内的平均值。**Average** 调用 **Summation** 求和，并计算平均值，然后返回结果，脚本如下：

```
Params
    NumericSeries Price(1);
    Numeric Length(10);
Vars
    Numeric AvgValue;
Begin
    AvgValue = Summation(Price, Length) / Length;
    Return AvgValue;
End
```

对于使用多个输出的情况，即使用引用参数的情况，我们以求 **N** 周期最大值为例进行描述，假定我们需要编写一个用户函数，该函数需要求出序列变量 **Price** 在最近 **Length** 周期内的最大值，并且要求出最大值出现的 **Bar** 和当前 **Bar** 的偏移值。脚本如下：

```
Params
    NumericSeries Price(1);
    Numeric Length(10);
    NumericRef HighestBar(0);
Vars
    Numeric MyVal;
```

```
    Numeric MyBar;
    Numeric i;
Begin
    MyVal = Price;
    MyBar = 0;
    For i = 1 to Length - 1
    {
        If ( Price[i] > MyVal)
        {
            MyVal = Price[i];
            MyBar = i;
        }
    }
    HighestBar = MyBar;
    Return MyVal;
End
```

## 用户函数的调用

用户函数成功创建之后（编译/保存成功），您可以在其他的用户函数、公式应用中调用用户函数，调用用户函数时需要注意保持参数类型的匹配，即用户函数参数的声明数据类型需和调用时传入参数的数据匹配，这是所指的匹配是指基本数据类型：数值型，布尔型，字符串三种类型匹配，并且保持序列参数和传入变量类型的对应。我们可以对用户函数定义为 **Numeric** 或者 **NumericRef** 的参数使用 **Numeric** 类型的变量作为传入参数；但不能将在定义为 **NumericRef** 类型的参数时传入 **NumericSeries**。具体的对应关系如下表：

函数参数声明类型	可传入的变量类型
<b>Numeric</b>	Numeric, NumericRef, NumericSeries
<b>NumericRef</b>	Numeric, NumericRef
<b>NumericSeries</b>	Numeric, NumericRef, NumericSeries

<b>Bool</b>	Bool, BoolRef, BoolSeries
<b>BoolRef</b>	Bool, BoolRef
<b>BoolSeries</b>	Bool, BoolRef, BoolSeries
<b>String</b>	String, StringRef, StringSeries
<b>StringRef</b>	String, StringRef
<b>StringSeries</b>	String, StringRef, StringSeries

对于函数的返回值，您也可以将用户函数的 **Numeric** 返回值赋值给 **NumericSeries** 或 **NumericRef** 变量。即在用户函数的返回值使用时，忽略其扩展数据类型。比如我们在调用 **Average** 求平均值时，可以这样调用：

```
Vars
    Numeric Value1;
Begin
    Value1 = Average(Close,10);
    ...
End
```

我们也可以按照以下方式进行调用：

```
Vars
    NumericSeries Value1;
Begin
    Value1 = Average(Close,10);
    ...
End
```

## ●技术分析类的公式应用

交易开拓者软件的旗舰版可以将指标线的输出等技术分析与交易指令放在同一个公式应用里执行。即一个公式应用，可以在画出所需的分析线型同时也可以图表上的相应指定位置发出买卖开平仓的指令讯号。本章节将针对公式应用中输出技术分析这一块进行详细的讲解。

技术分析是公式应用最常用的功能，它通过计算一系列的数学公式，在每个 **Bar** 都返回值，这些值在图表模块中输出为线条、柱状图、点等表现形式，通过分析图形特点、趋势和曲线帮助客户分析行情走势，得出合理的交易判断。

### 技术分析的模板

当您要新建一个技术分析类的公式应用，可以在新建公式应用的模板里选择“技术分析”。模板里有现成的两个变量的声明、赋值以及输出，以及一个条件输出报警的语句。您可以根据自己的要求来进行添加、修改与完善公式并进行编译。

如果您对交易开拓者的公式比较熟悉之后，也可以在模板处选择为“空”，不需要任何的现成模板，而是自己从头开始一个公式的建立。

### 输出函数的具体说明

TRADEBLAZER 的公式应用可提供在图表上输出数值、布尔值以及字符串，以满足交易者在做技术分析时的各种个性化的输出。分别使用 **PLOTNUMERIC**、**PLOTBOOL**、**PLOTSTRING** 这三个函数进行输出的动作。

以下为三个函数的语法及使用说明：

**PlotNumeric**-----在当前 Bar 输出一个数值；

语法: PlotNumeric(String Name,Numeric Number,Numeric Locator=0,Integer Color=-1,Integer BarsBack=0)

**Name** 输出值的名称的字符串，可以为中、英文、数字或是其它字符；

**Number** 输出的数值；

**Locator** 输出值的定位点，默认时输出单点，否则输出连接两个值线段，用法请看例 2；

**Color** 输出值的显示颜色，默认表示使用属性设置框中的颜色；

**BarsBack** 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar。

例 1: PlotNumeric ("RSI",RSIValue);

输出 RSI 的值。

例 2: PlotNumeric ("OpenToClose",open,close);

输出开盘价与收盘价的连线。（需要在公式属性的输出线形选择柱状图）

例 3: PlotNumeric ("AvgValue",average(close,5),0,blue);

输出一条蓝色的以收盘价计算的五日平均线。

注意：当后面的参数都使用默认值的情况下，可省略不写，如例 1。但如果后面还有其它参数要指明，而只是中间某一个或是多个参数需要默认值的话，则中间参数不可省略，需将默认值一一填写，如例 3。

**PlotBool**-----在当前 Bar 输出一个布尔值。

语法: PlotBool(String Name,Bool bPlot,Numeric Locator=0,Integer Color=-1,Integer BarsBack=0)

**Name** 输出值的名称，不区分大小写；

**bPlot** 输出的布尔值；

**Locator** 输出值的定位点；

**Color** 输出值的显示颜色，默认表示使用属性设置框中的颜色

**BarsBack** 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar。

示例：`PlotBool ("con",con,high);`

在 **bar** 的最高价位置输出条件 **con** 的布尔值。

**PlotString**-----在当前 **Bar** 输出一个字符串。

语法: `PlotString(String Name,String str,Numeric Locator=0,Integer Color=-1,Integer BarsBack=0)`

**Name** 输出值的名称，不区分大小写；

**str** 输出的字符串；

**Locator** 输出值的定位点；

**Color** 输出值的显示颜色，默认表示使用属性设置框中的颜色；

**BarsBack** 从当前 **Bar** 向前回溯的 **Bar** 数，默认值为当前 **Bar**。

示例：

`PlotString ("Bear Market","Bear Bear",high,blue);`

在 **bar** 的最高价位置输出一个蓝色的字符串 **Bear Bear**。

## 公式正文（技术分析类）

参数与变量的声明之后，在公式的正文部分，给所声明的变量进行赋值计算，必要的话还可以将所需的条件语句编写进公式里。最后则是将计算结果以数值、布尔或字符串的形式在图表上的输出。

现在我们来试着建立一个技术分析类的公式应用。首先，需要确立我们的指标里需要什么信息，以及需要通过何种计算从而得到我们想要的结果。比如，现在我们需要得到两条移动平均线，那么首先可以确定，我要输出的是两个变量的值，并且是数值型的变量。然后我们就可以在公式进行定义与赋值了。

有如下代码：

```
Params                                //参数的声明
    numeric length1(10);
    numeric length2(20);
vars                                  //变量的声明
    numeric ma1;
    numeric ma2;
```

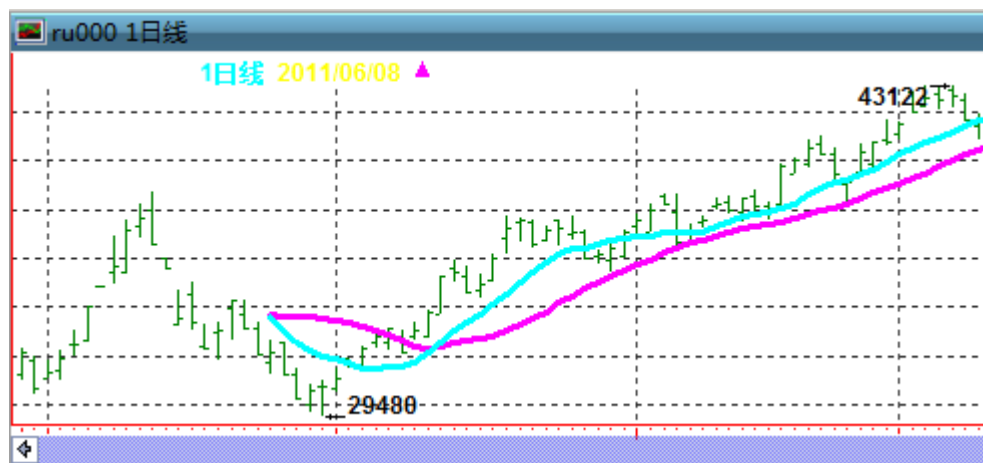
```

begin
    ma1 = averageFC(close,length1);
    ma2 = averageFC(close,length2);           // 给变量的赋值
    PlotNumeric("ma1",ma1,0,cyan);
    PlotNumeric("ma2",ma2,0,Magenta);        //输出计算出来的平均值
End

```

以上公式先把参数与变量的声明了，然后在 **Begin** 与 **End** 对公式的主体进行了赋值与输出。公式编译之后加载在图表上，既可得到预想的两条平均移动线。当然，对于数值的输出可以在图表上设置为线、线段、柱状、点状、十字状或是隐藏等表现形式。以及颜色、粗细的选择。

下图所示为以上代码在图表上的显示：



布尔型的输出在图表上的表现为指定位置显示圆脸图标。返回为真的布尔值显示一个绿的笑脸图标，反回为假的则是显示一个红色的哭脸图标。

有如下代码：

```

Vars
    Bool boolvalue;
Begin
    boolvalue = close>open;
    if(boolvalue)
        PlotBool("text",boolvalue,close+500);
End

```

当布尔变量为真时，在收盘价上方 500 个点位上输出布尔值，如下图：



字符串的输出可以在图表的指定的位置输出相应字符串，可以在公式代码时对字符串的颜色进行指定，也可以在公式应用的属性里对颜色进行选择。

示例：

Vars

```
Bool aa;
```

Begin

```
aa = high >= highest(high,10);
```

```
if(aa==true)
```

```
PlotString("text","高点",high,yellow);
```

End

如果当前 Bar 的高价创 10 新高，则在 Bar 的高价位置输出一个黄色的字符串“高点”，如下图：





## 输出数据的名称

函数 `PlotNumeric`、`PlotBool` 以及 `PlotString` 的第一个参数都是“字符串”，该字符串的意义在于给将要输出的值一个名称。此字符串可以不局限于英文字符。

在同一个公式应用里，同一类型的输出值不可定义相同名称的字符串。

示例：

```
PlotNumeric("tt",close);
```

```
PlotNumeric("tt",open);
```

因为定义了相同的字符串做为名称，以上的写法只会输出一条线，数值“tt”输出的将会是最后一条语句来执行，即以 `open` 价在每个 `Bar` 上画的一条曲线。

在同一个公式应用中，不同的类型的输出值则是可以定义相同的名称字符串的。

示例：

```
PlotNumeric("tt",open);
```

```
PlotBool("tt",false);
```

```
PlotString("tt",1);
```

以上公式虽然第一个参数的名称字符串都是相同的“tt”，但因为输出类型的不同，所以在图表上的表现则为三个值均有输出。

如下图：



## 输出颜色的选择

PlotNumeric、PlotBool、PlotString 这三个输出函数，从右起数第二个参数是均为输出颜色的选项。可以填写交易开拓者系统函数里的任一颜色函数来表示。也可以填写默认值，然后在属性里对颜色进行自定义的设置。系统函数里包括以下颜色函数：

black 黑色 / blue 蓝色 / cyan 青色 / darkbrown 深棕 / darkcyan 深青 / darkgray 深灰 / darkgreen 深绿 / darkmagenta 深紫 / darkred 深红 / defaultcolour 默认颜色 / green 绿色 / lightgray 浅灰 / magenta 紫红 / red 红色 / rgb 自定义颜色 / white 白色 / yellow 黄色

更多个性化颜色选择可在公式里填写默认的颜色参数，然后在公式应用的属性里设置自己喜好的显示颜色。

## 与 Bar 同齐的数据输出

在与数据源同列的每一个 Bar 上均有数值、布尔或字符串的输出。其中数值的输出也可能为无效值。

比如：PlotNumeric("avgvalue",averageFC(close,5));

该公式在图表数据的前 4 个 Bar 上输出的都是无效值，只有从第 5 个 Bar 开始才会输出数值型的计算结果。没有其它额外的条件，所以从第 5 个 Bar 开始之后的每一个 Bar 都会有计算得出的平均数值的输出。

注意：在图表整个 Bar 数据序列上，也会有因为计算、条件等因素，从而在 Bar 数据间中输出无效值。遇到此类情况可能会导致数据输出在图表上的整体图型很奇怪。所以，在输出数据之前可以在公式里加上判断语句，只有在非无效值的情况下才输出数据。

例如：

```
if(aaa!= InvalidNumeric) PlotNumeric("tt",aaa,0,red);
```

## 条件 Bar 下的数据输出

除了在与 Bar 等同的序列上输出数值、布尔或字符串数据，还可以在指定条件下输出数据，而在不符合条件的 Bar 上则不会有任何的值输出。指定条件下的输出可以使得图表上的输出更加简洁清晰。比如，上一个例子里对判断无效值则不输出变量 **aaa** 的语句。再比如，在对某种 K 线形态的标识或是突破点的标注等等。

示例：

Begin

```
if(open>close)
{
    PlotNumeric("tt",high,low,yellow);
}
```

End

指定条件只有为下跌收盘的 Bar 上方可输出一条黄色的高低点柱状连线。如下图：



## 偏移 N 个 BAR 的输出

在三个输出函数中，最后一个参数均是 **BarsBack**，此参数的意义是将值回溯 N 个 Bar 输出。需要注意的是，若此 N 值为正数，则是将输出值向左移 N 个 Bar 输出，若 N 值为负数，则是将输出值向右偏移 N 个 Bar 输出。

示例：

```
if(open>close) PlotNumeric("tt",high,low,yellow,2);
```

将上图所示的数值输出向左偏 2 个 Bar 输出在图表上，其效果如下图所示：



注意：这样的写法在历史分析中相当于用了未来数据，请慎重考虑此写法是否符合您的需求。另，实时行情中，每一次的运算只计算最新的一个 Bar 上的 Tick。所以当最新 K 线数据满足条件后，只有在手工刷新图表，才会对历史 K 线上按条件进行画线。

将前图所示的数值输出向右偏移 1 个 Bar 输出在图表上

示例：

```
if(open>close) PlotNumeric("tt",high,low,yellow,-1);
```

其效果如下图所示：



注意：交易开拓者的图表上，画线的前提条件是 **Bar** 数据。在没有 **Bar** 的情况是不能输出数据并画线的。所以，上例写法在最新 **K** 线出来之前，是没法在图表上画出前出前一个 **Bar** 所输出的数据线的。

## UnPlot 的使用

除上述三个输出函数外，交易开拓者还提供一个函数 **UnPlot**，该函数可以删除输出函数曾经输出的值。

语法：Unplot(String Name,Integer BarsBack =0)

**Name** 要删除输出值的名称，不区分大小写；

**BarsBack** 从当前 **Bar** 向前回溯的 **Bar** 数，默认值为删除当前 **Bar** 曾输出的值。

示例：

Params

Numeric length1(10);

Numeric length2(20);

Vars

Numeric ma1;

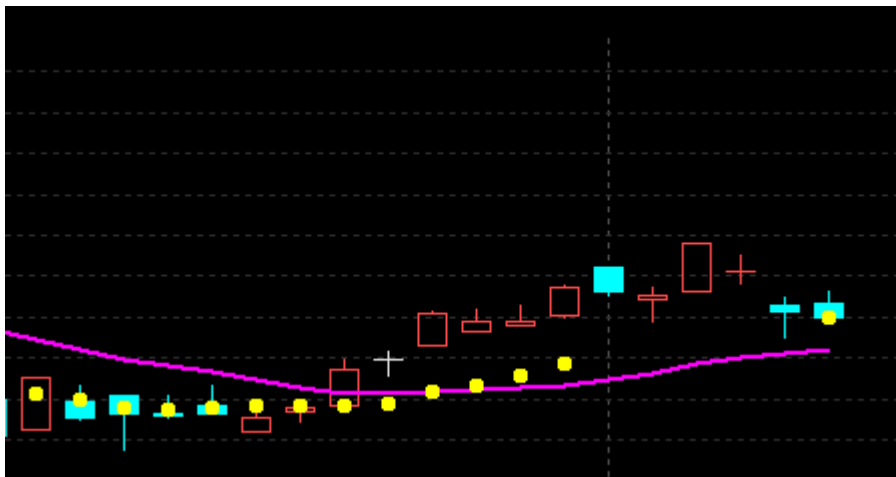
Numeric ma2;

Numeric i;

Begin

```
MA1 = AverageFC(Close,Length1);
MA2 = AverageFC(Close,Length2);
PlotNumeric("MA1",MA1);
PlotNumeric("MA2",MA2);
If(BarStatus==2)
{
    for i =0 to 5
    {
        Unplot("ma1",i);
    }
    PlotNumeric("ma1",close);
}
End
```

以上示例表述的是输出两条移动平均线,把其中 **MA1** 均线的最后 6 个 K 线上的输出值给删除,并在后一个 K 线输出一个新值。结果如下图:



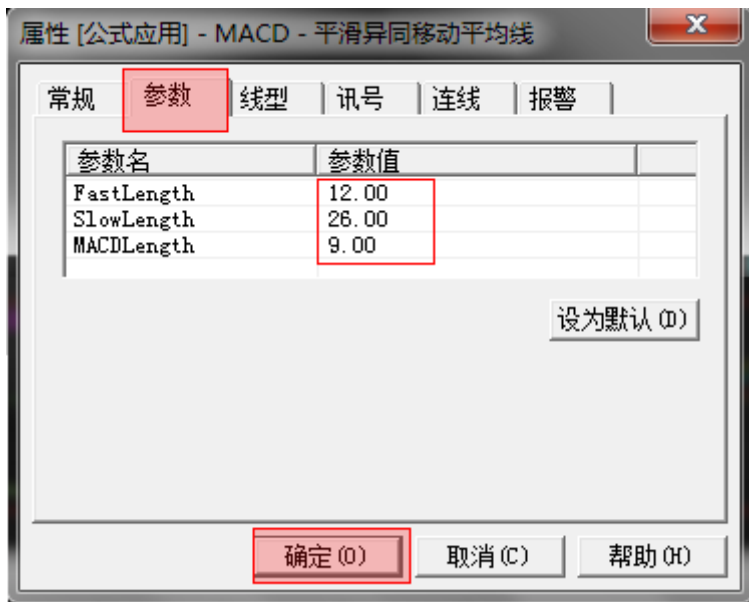
## 参数的调整

在图表上调用公式应用,可以直接在公式应用属性里随时调整、指定相应的参数值而无需重新编译。

进入公式应用的属性,选择“参数”,可以分别对每一个参数进行调整。调整完毕按下“确定”按钮,此时图表上就立刻显示最新参数计算而得的输出结算。

在调整参数之后，如果想将当前此组参数做为该公式应用的固定参数而永久保存，则可以按下“设为默认”按钮。如此一来，系统将以新的参数再次编译公式。之后的公式调用，都以新的参数来参与计算。

如下图：



## ●交易策略类公式应用

上一章节所讲述的是建立技术分析类的公式应用。技术分析是通过计算以及条件语句，在图表上输出一些数据或是线型，通过图型的显示来帮助交易者进行一系列直观的分析。除此之外交易开拓者的公式应用还可以使用更多的条件与计算来判断交易的入场点与出场点，在图表上对买卖点做出标识，也就是我们通常所说的交易信号。

### 交易策略的基本规则

交易，有买有卖方可形成交易。所以一次完整的交易，必须要有开仓以及平仓的动作。如同其它的公式一样，建立一个交易策略也需要先命名公式的名称，然后在编辑界对参数变量进行声明。最后就是在公式主体进行一系列的计算以及条件语句的控制来完成整个交易的过程。

### 交易指令函数

交易开拓者提供 **Buy**, **Sellshort**, **Buytocovert**, **Sell**, **A\_SendOrder** 这五个函数来做下单的动作指令。

**Buy**, **Sellshort**, **Buytocovert**, **Sell** 这四个函数可以针对历史数据在图表上做出讯号标识，同时在实时行情也可以迅速及时地发出下单委托的动作。因为可以对历史数据的图表上做出讯号的标识记录。因此，交易设置的性能测试也是需要由此几个函数参与计算的公式应用方可实现。

**Buy** ---- 对当前合约发出买入开仓的指令，如果图表讯号显示当前持有空仓，则会先平掉空仓，再开多仓；

**Sellshort** ---- 对当前合约发出卖出开仓的指令，如果图表讯号显示当前持有多仓，则会先平掉多仓，再开空仓；



**Buytocover** ---- 对当前合约发出平空仓的指令，当图表信号显示有空头持仓时，方可执行此指令；

**Sell** ---- 对当前合约发出平多仓的指令，当图表信号显示有多头持仓时，方可执行此指令；

当条件判断下，执行买入或是卖出的动作，并在图表上标识出讯号。如下例：

```
if(condition1 && marketposition!=1)    //条件满足且没有持多仓情况下开多
{
    buy();
}
else if(condition2 && marketposition!=-1) //条件满足且没有持空仓时开空
{
    sellshort();
}
```



**A\_SendOrder** 则是通过与枚举函数的配合使用从而直接对帐户进行发送指令的动作。他只能在最后一个 **Bar** 上针对当前帐户进行开仓、平仓以及撤单等委托操作，不可以对图表做出讯号的标识或是进行历史的回溯测试。

前面的章节，我们有说到 **TradeBlazer** 公式在实时行情中是每一个 **Tick** 执行一次公式的。于是，当公式每运行一次，只要在条件满足的情况下，**A\_SendOrder()** 都会执行一次，也就是都会发一次委托。这样一来，势必会因为重复的委托动作导致最后交易次数过多，与原意不符。所以，使用 **A\_SendOrder()** 来进行交易的话，需要全局变量配合使用，从而达到理想的交易效果。

注意：有关全局变量配合使用 **A\_SendOrder()** 来发单的具体实现方法会在后面的“公式进阶”里有示例说明。

## 交易策略的讯号设置

交易开拓者中，使用了 **buy**、**sellshort** 等交易指令函数的公式应用可以在超级图标中相应 **Bar** 的位置标出指令讯号。

在图表中加入公式应用之后，您可以通过选中讯号标记，右键菜单中点击“属性设置”，选择“讯号”选项卡，讯号设置的界面如下图所示：



讯号是公式系统在历史数据中应用产生的买卖指令，在图表中显示为向上或向下的箭头，并在买卖价位处用侧向箭头进行标记。

对话框左边设置四种类型公式应用讯号的显示风格，右边价位标记风格设置。

系统默认的是将多头的开平仓设置为黄色，空头的开平仓设置为紫红色。对于四种公式应用，可分别设置是否显示公式应用名称，是否显示交易数量，以及讯号箭头和价位箭头的类型和颜色。

开仓、平仓之间的连线可清楚地表示出一次交易是盈利还是亏损，我们通过两种不同的颜色将连线区分为获利线和亏损线，您还可以对线条的风格，粗细和颜色进行设置。

右键菜单中“属性设置”中,点击“连线”选项卡，如下图：



系统默认的是将盈利的交易开平之间连红色的线，亏损的交易开平间连绿色的线。您可通过点击“设为默认值”按钮将当前对话框的各项设置保存为系统默认值，之后您可以在其他地方使用该默认设置。

您也可通过点击“恢复默认值”按钮将当前对话框的各项设置修改为系统默认值。修改完讯号设置之后，点击“确定”按钮将当前的设置传递给调用窗口，或者点击“取消”放弃所有修改。

## 商品叠加的交易策略

### 数据源的叠加

交易开拓者支持一个图表上对不同合约数据源的叠加，同时也支持对不同数据源进入同时发单动作。交易者通常所做的价差套利交易就是以此类的方法来实现的。

首先，我们需要在一个超级图表上进行数据源的叠加。打开一个超级图表，在选择所需合约做为主图合约后，可以从菜单栏选择“插入”再选择“插入商品”，并选择指定所需的第二个合约数据。或者是直接在图表上点击右键的菜单中直接选择“插入商品”，进行选择设置。

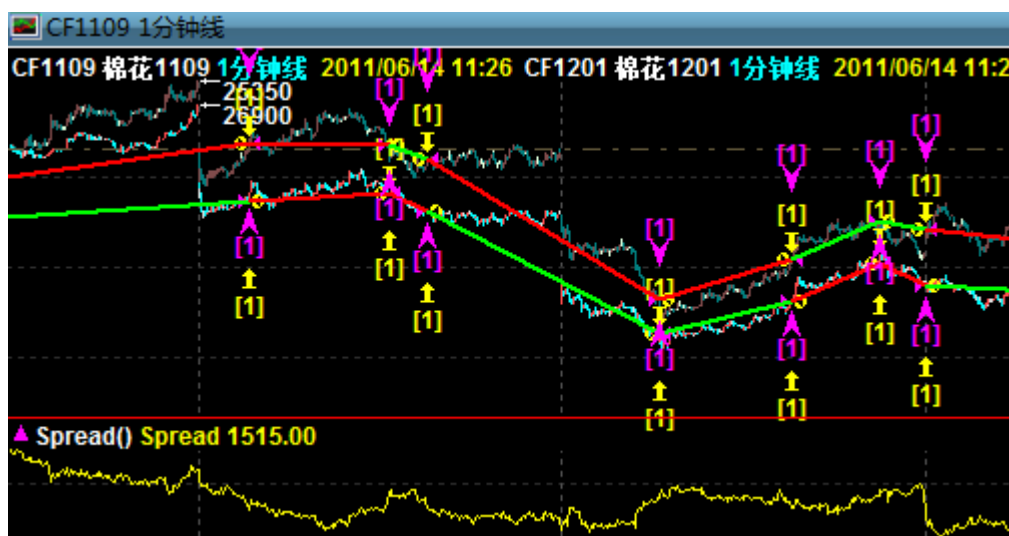
## 公式语句中对数据源的区分

根据插入商品的顺序，使用 `data0.close`、`data1.close`.....等等来进行区分。同样，对于不同数据源的发单指令也是可以使用 `data0.buy()`、`data1.sellshort()`、`data2.A_SendOrder()` 等来表示。

举例如下：

```
if(condition1)
{
    data0.buy(1,open);
    data1.sellshort(1,data1.open);
}else if (condition2)
{
    data0.sellshort(1,open);
    data1.buy(1,data1.open);
}
```

交易讯号如下图：



有关叠加合约的公式具体实现方法可参考后面章节的“公式进阶”里的“多品种交易”。

**注意：**一个超级图表上叠加合约数量是没有限制的。但是叠加多个商品会非常的耗费资源，请交易者根据机器的性能以及交易的需求等来判断一个图表上叠加合约的合理数量。

## 交易策略的实现

### 策略的头寸

交易头寸的设计在一个完整的策略中是必不可少的环节。简单的，您可以指定一个固定的开平仓头寸。更深一层的，您可以根据资产的具体情况以及交易的要求来计算得出所要交易头寸。

可以将交易手数做为一个参数代入公式中，这样可方便随时调整交易的数量。例如：

Params

Numeric Lots(3);

Begin

.....

Buy(lots,price);

.....

可以将直接在公式里直接填写默认的手数，然后在全局交易指令设置里对其具体数量进行设置。例如：

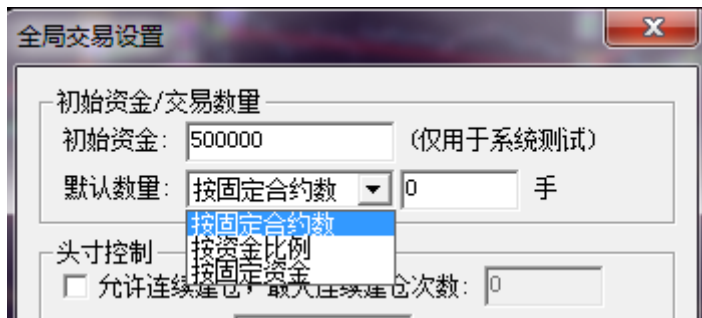
.....

Buy(0,price);

.....

在超级图表已调用的交易策略名称上右键菜单里选择“全局交易设置”的超级图表上。在“默认数量”可以按固定合约数、资金比例及固定资金这三种方法来进行设置。

如下图：



其中资金比例是参照初始资金的设置来计算的，所以需要注意设置一个合理的、符合实际情况的初始资金数。

除了以上简单的设置方式，还可以将交易的手数在公式里定义成为一个变量。通过对资产以及行情等条件的计算以及其它的判断处理从而确定交易的数量。例如交易开拓者软件的系统公式里自带的“海龟交易系统”，就是通过一系统的计算，最后得到的 **TurtleUnits** 便是该交易策略的交易手数。在不同的行情与是资金状态下，计算得出的开仓数量也是不尽相同的。

例如：

```
AvgTR = XAverage(TrueRange, ATRLength);
N = AvgTR[1];
TotalEquity = Portfolio_CurrentCapital() + Portfolio_UsedMargin();
TurtleUnits = (TotalEquity * RiskRatio / 100) / (N * ContractUnit() * BigPointValue());
TurtleUnits = IntPart(TurtleUnits); // 对小数取整
```

## 开仓与平仓

在使用 **buy**、**sellshort**、**buytocover**、**sell** 这几个函数写的公式都是基于图表信号的命令。可以用于历史回溯，也可以及时地对实际交易的帐户发出委托单指令。

只有在图表上已有开仓的讯号的情况下，方可标出平仓讯号。如果没有开仓语句的配合，或是进行手工开仓，或是图表上当前没有开仓的讯号，那样的话是不可以有平仓讯号的标识的。

## 加仓与减仓

在一个趋势走向没有结束之前，交易者可以针对当前行情做一个顺势加仓的交易模式，这样加仓的操用在交易开拓者的公式策略中完全可以实现。

您可以根据自己的想法，在所需要的条件位置进行建仓与加仓，甚至在不同条件下的加仓可以是不同的交易手数。（注：具体详细示例可参考后面章节的“公式进阶”）

当然，也可以在没有持仓限制的公式策略里，可以在图表上右键点击公式名称所出来的菜里“全局交易设置”勾选允许连续连仓，并设定好相应的加仓次数，这样即可完成一个简单加仓的过程。

例如：

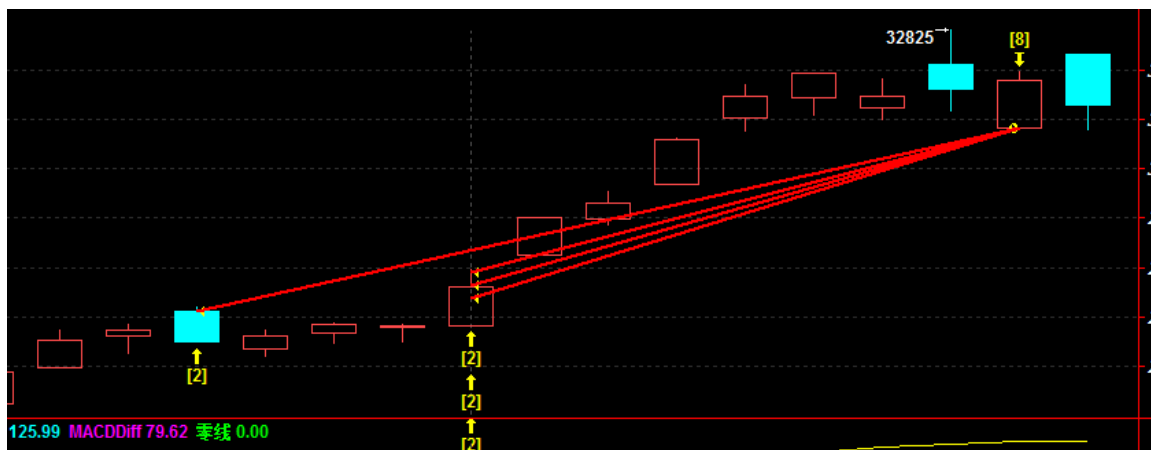
```
if(open>avgvalue[1])
{
    buy(0,open);
}
```

另，在“全局变量设置”里做出如下图所示设置：



上述公式并配合上图所示的设置，则是表示了价格大于平均线之上，每一个满足此条件的 Bar 上都会开建一个多仓，每次建仓的的交易数量为 2 手。同时也限制了最多可以在同一个方向上连续建仓四次，即最多有 3 次的加仓。

其图表信号表现如下：



减仓与加仓相反，是一个逐步而非一次性平掉所有仓位的交易模式。

有关加仓与减仓的更多详细内容请参考后面章节的“公式进阶”中的“加仓与减仓”。

## 策略交易的辅助功能

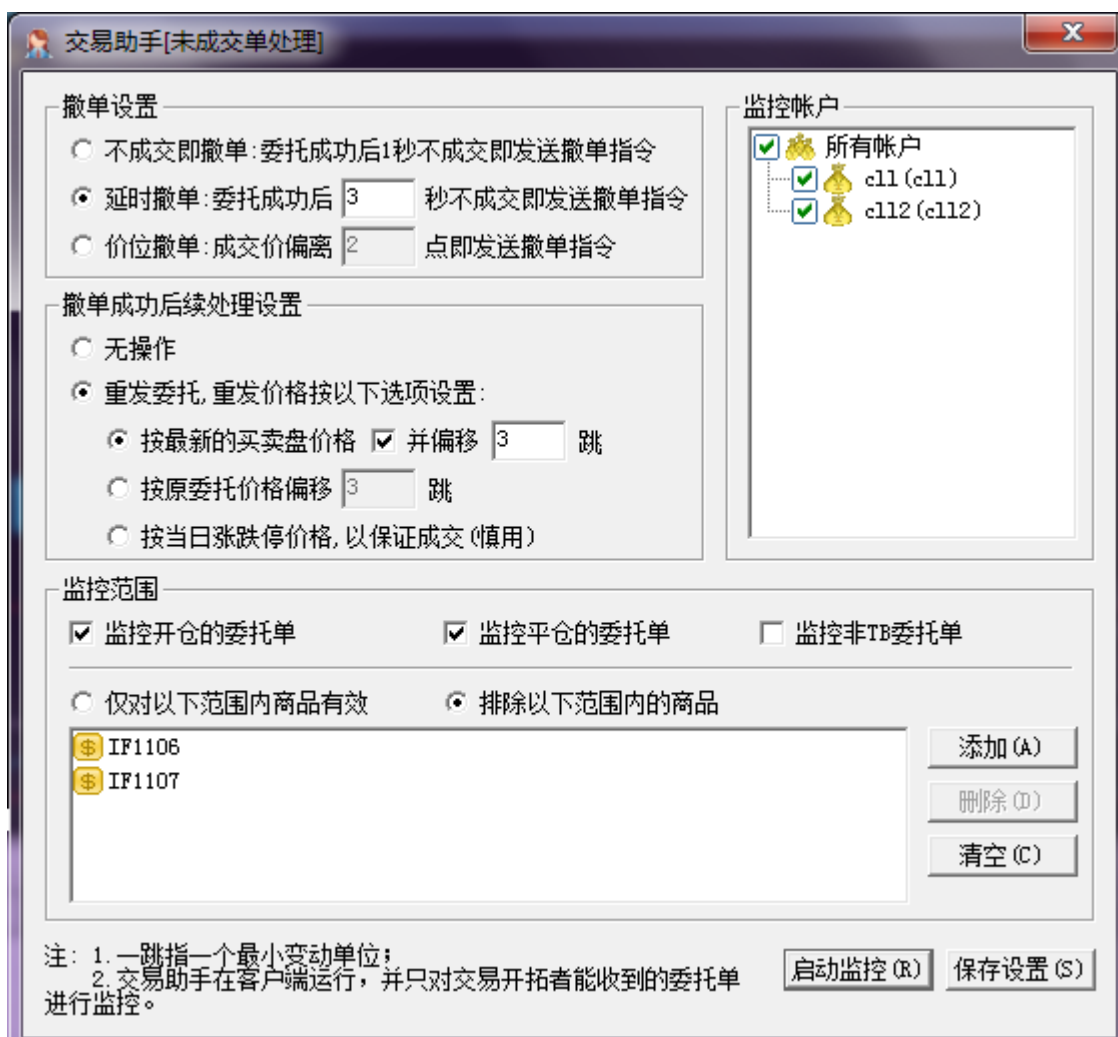
### 交易助手的应用

对于公式策略进行自动开平仓的操作，有时会因为下单价格不合理或是行情波动快速等原因导致委托单不能及时成交。此时可以使用“交易助手”做辅助的处理。交易助手在启动监控后，可以根据之前的设置对未成交单自动进行撤单以及撤后重发委托单等后续的处理。（交易助手是对帐户进行控制的，所以上述操作同样适用于手工下单的未成交委托。）

另外，交易助手还可以配合快车道的自动生成止盈单功能从而达到自动止损的操作。



界面图如下：



撤单设置：分别有不成交易即撤单、延时撤单、价位撤单。

后续处理：可选择不做任何操作或是重发委托单。重发委托单的价格可以有多种选择。可以选择以原委托价加偏移跳数重发、以最新买卖盘或加偏移跳数重发以及使用涨跌停价重发委托。

另外，对监控帐户、监控开仓或平仓单、监控品种等都可自主选择。

**注意：**在选择“延时撤单”时，请设置 2 或 3 秒以上的撤单时间。因为下单从发出委托到单子经过交易所的处理再回报到本地客户端需要一定的时间，如果设置时间过短则会导致不能读取正确的交易状态。

## 调试语句的输出

**Commentary** ---在超级图表当前 Bar 添加一行注释信息。

该函数无返回值，作为系统调试的辅助工具，您可以双击超级图表出现十字光标后，在工具栏选择“显示提示框”，这样便可以看到任意 Bar 上的注释信息了。

示例：

```
if(close>open)
```

```
    commentary("收阳="+Text(close));    \ 在收阳的 K 线上显示收盘价
```



如上左图显示，在信息提示窗口里可以看到 **commentary** 语句里显示的注释内容。上例所写的是条件下输出，只有在满足 **if()** 语句内的条件 Bar 上方可显示此内容。在不满足条件的 Bar 上则不会有注释信息的输出，如上右图所示。

当然，如果没有条件语句来约束 **commentary**，则会在图表上的每一个 Bar 上都输出注释信息。比如在每一个 Bar 上输出一个变量值的注释信息，可方便交易者观察图表上的数据情况从而对自己的公式策略进行相关的调试。

例如： `commentary("开仓价格" + (symbol) + " = " + text(entryprice));`

`commentary("满足多头开仓");`

**FileAppend** ----在指定文件中追加一行字符串，返回值为布尔型。执行成功返回 **True**，执行失败返回 **False**。

语法: **Bool FileAppend(String strPath,String strText);**

参数: **strPath** 指定文件的路径，请使用全路径表示，并使用 \ 做路径分割符，否则会执行失败，**strText** 输出的字符串内容；

与 **commentary** 只在图表上的 **Bar** 上输出注释信息不同。**fileappend** 的调试信息不会在图表上有所标识，而是写入一个日志文件中。每一次执行到此语句，便会在文件中写入一条调试信息，以便交易者的查找调试。

例如: **FileAppend("C:\\Formula.log","Close = "+Text(Close));**

如上述语句，在没有任何条件下输出此调试语句，因为每一个 **Tick** 都会有一次运行，所以同一个 **Bar** 上的则会输出多条调试语句。这样一来，交易者可以通过每一次的运算结算从而查找到自己所需要的信息。

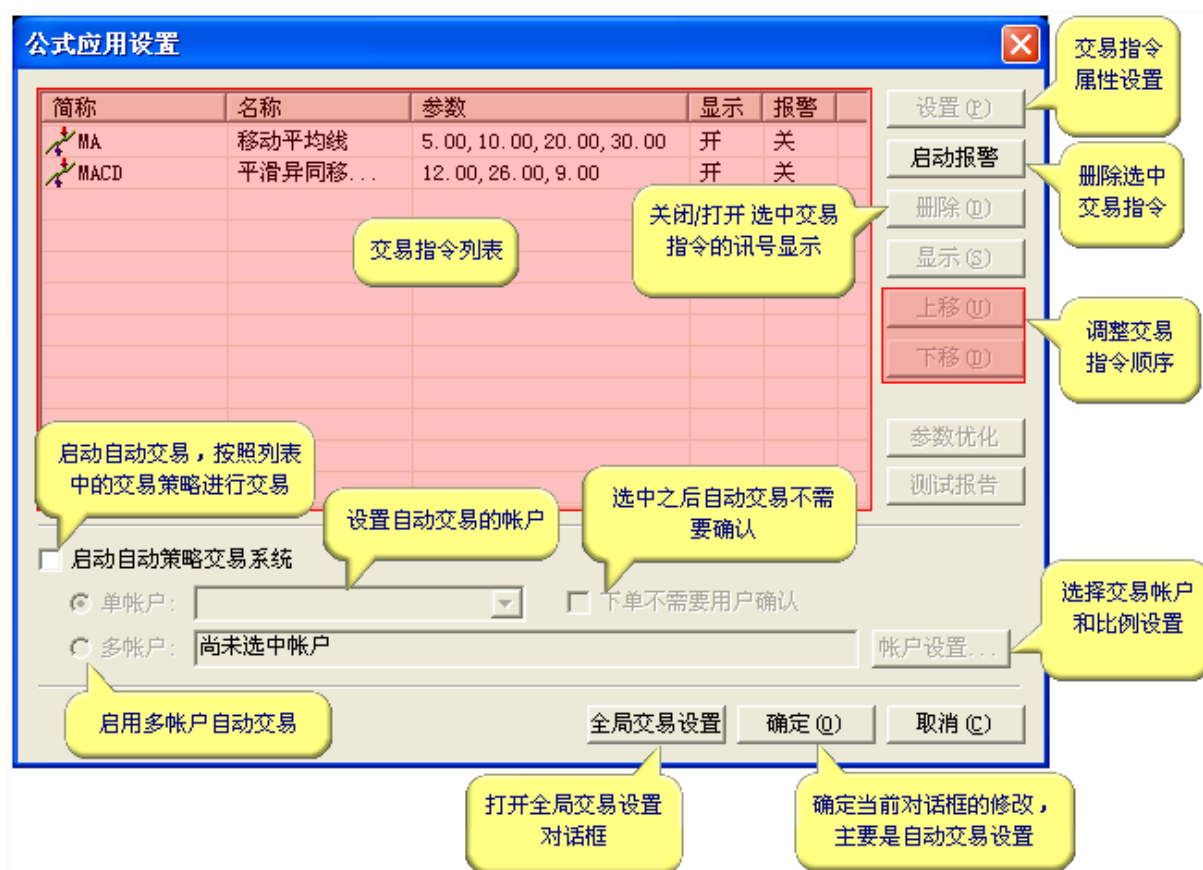
如果是在条件语句下输出 **FileAppend**，则会是在条件满足之后才会有调试语句入日志文件。

## 自动交易的设置与实现

当您的交易策略已经编写完成并通了校验与测试，可以在超级图表上应用并设置为自动交易。

首先，是选定所需交易的品种合约以及时间周期。然后再将公式应用插入在已选定的超级图表上。

您可以通过工具栏上的“启动自动交易”的按钮进行设置，也可以在图表上的右键菜单选择“公式应用设置”里进行全自动交易的设置。会出现 如下设置界面：



在公式应用设置中，可以对当前公式进行参数、显示、以及报警等设置。在设置界面的下方则是启动自动交易的设置选项。

第一步则是要勾选“启动自动策略交易系统”。

接下来是需要根据您的交易需求是否勾选选择上“下单不需要用户确认”。一旦勾选此项目，则会在当前运行的 **BAR** 上产生交易讯号的第一时间就将委托单发送出去。而如果不勾选此项目，则会在讯号出现的同时，在屏幕上跳出一个当时指令委托的确认框，只在交易者手工点击确认后方可发出委托单。

可以选择单个帐户或是多个交易帐户同时执行此策略。多帐户右边的“帐户设置”则提供了不同帐户之间的交易数量的设置复选框。可根据各帐户不同的资金状况不同的交易需求而同时进行不同交易数量的委托。

在公式应用的设置完成后，按下“确定”。此时开始，自动策略交易便进入运行状态了。

为了方便交易者直观清晰地看到当前图表自动策略交易的运行状态。在超级图表的右上角有不同颜色的小圆脸来加以区分判断。



表示当前图表没有加载或执行任何的策略交易；



表示当前图表已加载策略公式并关联了交易帐户，但未启动自动策略交易；



表示当前图表已启动了半自动策略交易，即下单时需要人工确认方可发出委托；



表示当前图表已启动了全自动策略交易，讯号出现会即时发送委托。

**注意：**若公式策略的编写是使用 **A\_SendOrder** 进行发送委托指令的，则不可以设置“多帐户自动策略交易”。交易者可以同时打开多个超级图表，调用同一个策略，分别对应不同交易帐户并启动自动策略交易即可。

## 历史性能测试

交易开拓者提供历史性能测试的功能。

在超级图表中插入一个或一个以上公式应用后，菜单和工具栏中的“投资组合性能测试报告”选项将会有效，点击选项可以打开投资组合数据汇总，如果只有一个公式应用则直接打开交易策略测试报告。投资组合数据汇总的内容和交易策略参数优化报告类似。您可以双击汇总列表中的某项打开对应的性能测试报告，双击汇总行则可以查看投资组合性能测试报告。

您也可以通过点击公式应用设置,选择一个需要测试的交易策略,然后点击按钮“测试报告”或者公式应用名称的右键菜单里“性能测试”直接打开某个公式应用对应的性能测试报告。

单个交易策略的测试报告内容和投资组合的测试报告基本一致。

投资组合性能测试报表界面如下：

交易策略测试报告 [ru000 - 30分钟]

交易汇总交易分析交易记录平仓分析阶段总结资产变化图表分析系统设置

性能概要

统计指标	全部交易	多头	空头
净利润	319749.90	204238.32	115511.57
总盈利	597555.13	329192.08	268363.05
总亏损	(277805.24)	(124953.76)	(152851.48)
总盈利/总亏损	2.15	2.63	1.76
交易手数	277	138	139
盈利比率	37.55%	47.10%	28.06%
盈利手数	104	65	39
亏损手数	173	73	100
持平手数	0	0	0
平均利润	1154.33	1479.99	831.02
平均盈利	5745.72	5064.49	6881.10
平均亏损	(1605.81)	(1711.70)	(1528.51)
平均盈利/平均亏损	3.58	2.96	4.50
最大盈利	52937.86	27093.18	52937.86
最大亏损	(7153.61)	(6013.40)	(7153.61)
最大盈利/总盈利	0.09	0.08	0.20
最大亏损/总亏损	0.03	0.05	0.05
净利润/最大亏损	44.70	33.96	16.15
最大连续盈利手数	3	7	3
最大连续亏损手数	8	5	11

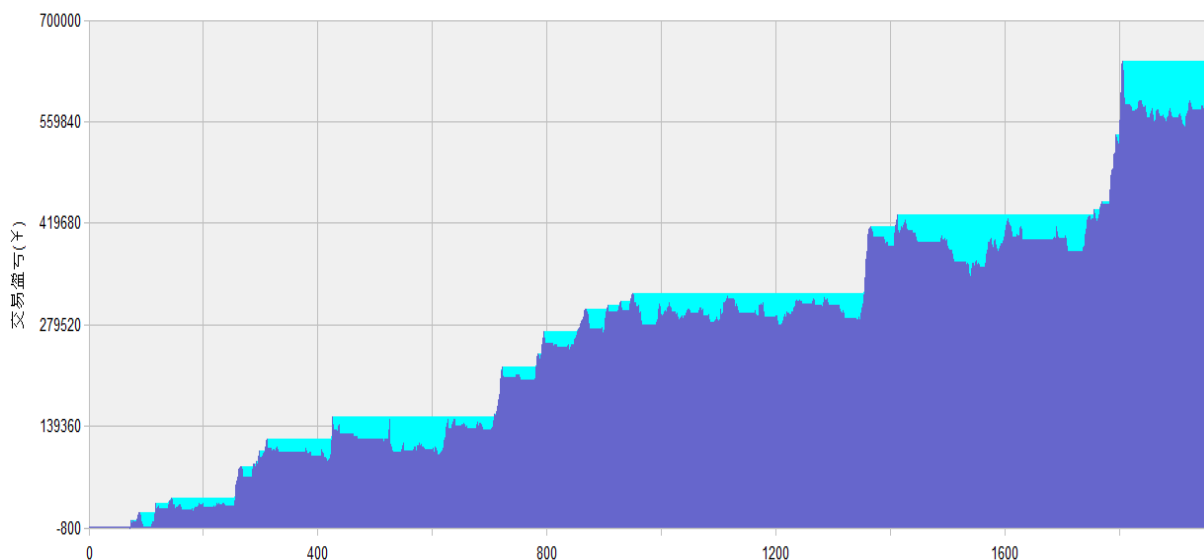
商品[1]个公式应用[1]个时间范围[2005/11/04 14:00 - 2011/04/01 14:30]

性能测试报表按照八个方面对交易策略或投资组合的报表进行分析，包括交易汇总、交易分析、交易记录、平仓分析、阶段总结、资产变化、图表分析和系统设置。

- 交易汇总：按照多头交易、空头交易和全部交易列出当前交易策略的交易统计信息。
- 交易分析：对当前策略的交易情况进行分析，包括交易分析、盈亏和连续盈亏分析。
- 交易记录：按开仓平仓对所有交易进行配对组合，并计算盈亏及累计盈亏。
- 平仓分析：按平仓记录对交易情况进行分析和汇总。
- 阶段总结：按年、月对交易盈亏及次数进行统计。
- 资产变化：列出资产的变化记录及统计信息。
- 图表分析：按资产图表和盈亏图表两大类共十三小类对帐户进行图表分析。
- 系统设置：显示交易策略的参数，设置以及数据等内容。

其中图表分析中的十三小类分别是交易盈亏典线图、交易盈亏典线图（详细）、交易盈亏面积图、交易盈亏面积图（详细）、月度盈亏柱状图、月度累计盈亏点状图、交易总效率分布图、交易建仓效率分布图、交易平仓效率分布图、交易盈亏分布图、浮动盈利分布图以及浮动亏损分布图。

如下例图为“交易盈亏面积图（详细）”：

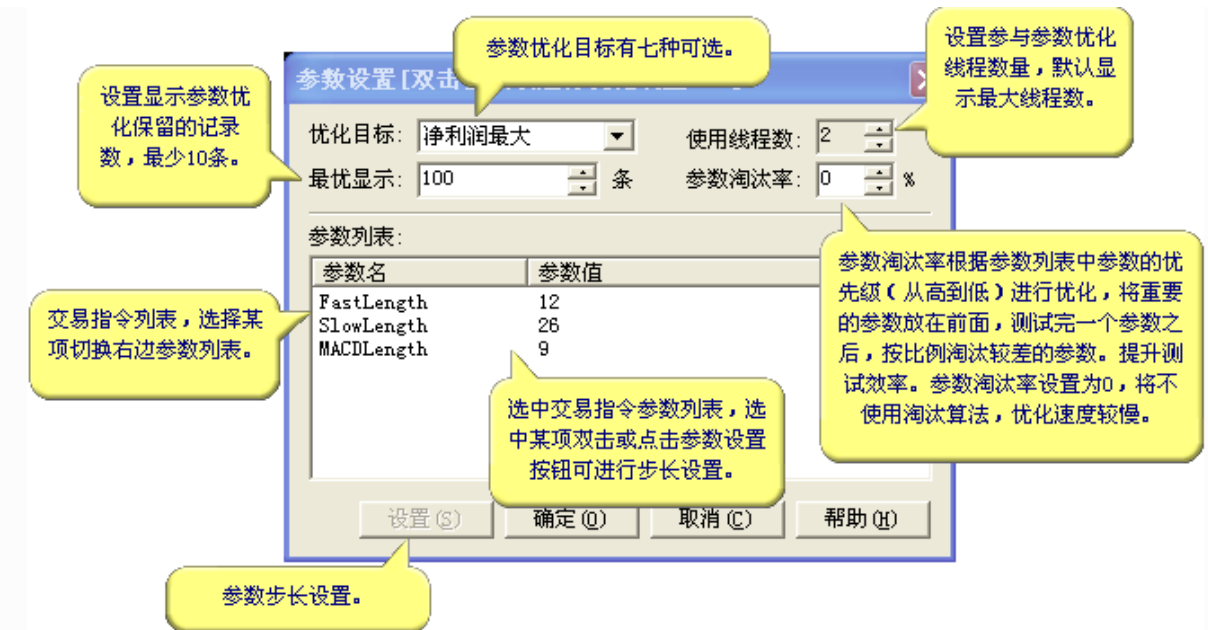


其中蓝色的部分为资金盈亏面积，淡蓝色所画出的部分为前期资金高点的水平线。

## 交易策略参数优化

在超级图表中插入一个或一个以上公式应用后，菜单和工具栏中的交易策略参数优化选项将会有效，您可以通过点击菜单项或工具栏使用该功能。

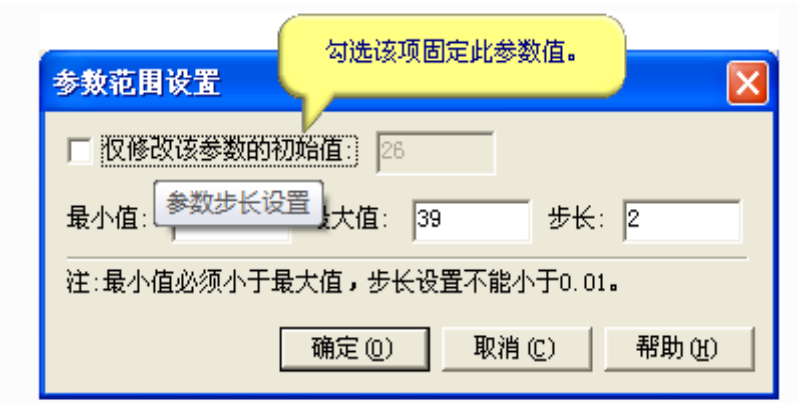
交易策略参数优化模块可对多个公式应用组合的所有参数进行优化，您可以通过参数设置界面对需要优化的参数进行设置，界面如下：



参数优化目标有以下九种选项：

- **净利润最大：**以优化结果中的净利润最大为目标，保留指定数量的记录数；
- **交易手数最大：**以优化结果中的交易手数最大为目标，保留指定数量的记录数；
- **平均净利润最大：**以优化结果中的平均净利润最大为目标，保留指定数量的记录数；
- **盈利因子最大：**以优化结果中的盈利因子最大为目标，保留指定数量的记录数；
- **盈利比率最大：**以优化结果中的盈利比率最大为目标，保留指定数量的记录数；
- **盈亏比率最大：**以优化结果中的盈亏比率最大为目标，保留指定数量的记录数；
- **收益率最大：**以优化结果中的收益率最大为目标，保留指定数量的记录数；
- **头寸系数最大：**以优化结果中的头寸系数最大为目标，保留指定数量的记录数；
- **TB 系数最大：**以优化结果中的 TB 系数最大为目标，保留指定数量的记录数。

通过双击参数项或点击参数设置按钮，对选中的参数项进行步长设置，界面如下：





您可以设置参数的最小值，步长及最大值，系统将会根据设置，产生从最小值到最大值之间按步长分布的参数列表。

在各项参数设置完成之后，点击确定按钮，将会进行参数优化的计算，参数优化计算过程中会显示优化的进程，时间，当前参数，最优参数以及优化目标等信息，您可以通过点击取消按钮终止计算。

**注意：**根据优化参数设置的多少，优化时间可长可短，从数十秒到数十小时均为可能。优化过程中，CPU 将会大量被占用，您可以选择空闲的时间进行大量参数优化计算。

参数优化计算完成之后，将会显示出最优的记录，您可以通过工具栏的交易设置按钮修改交易参数，也可以通过重新优化按钮进行重新计算。

## 优化结果生成分析图表

参数优化结果会以一个报表的形式呈现

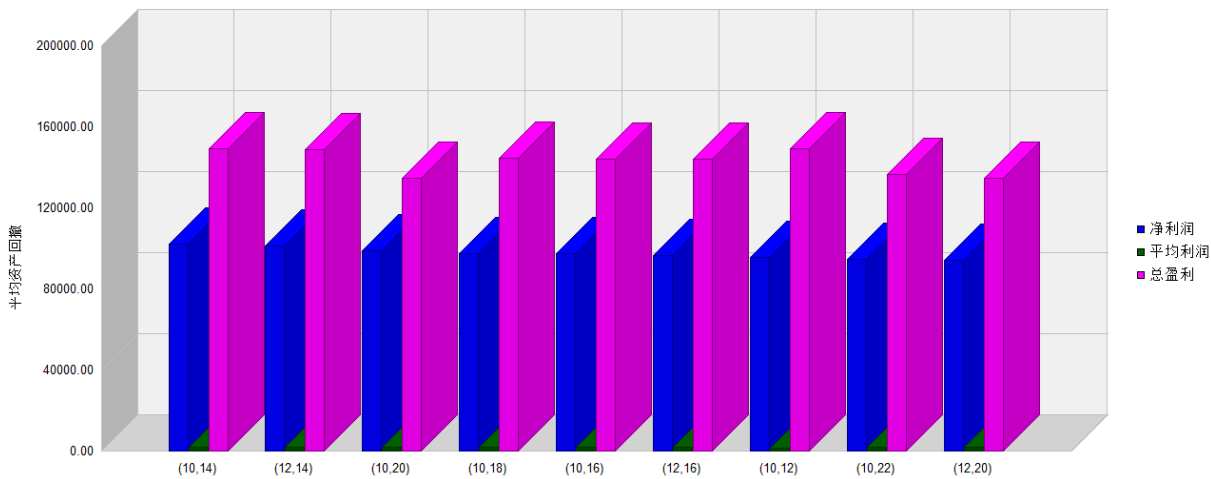


#	P:boLength	P:fsLength	净利润	盈利比率	平均利润	交易手数	最大资产回撤	TB系数	增长系数
1*	22.00	57.00	65360.00	47.54	1071.48	61	-34630.00	14.68	24.07
2	22.00	52.00	65360.00	47.54	1071.48	61	-34630.00	14.68	24.07
3	22.00	47.00	65360.00	47.54	1071.48	61	-34630.00	14.68	24.07
4	14.00	27.00	64220.00	51.35	867.84	74	-34130.00	16.40	22.16
5	22.00	27.00	62180.00	47.95	851.78	73	-34630.00	13.85	18.97
6	30.00	62.00	61990.00	50.00	1192.12	52	-33870.00	20.93	40.26
7	30.00	57.00	61990.00	50.00	1192.12	52	-33870.00	20.93	40.26
8	30.00	52.00	61990.00	50.00	1192.12	52	-33870.00	20.93	40.26
9	30.00	47.00	61990.00	50.00	1192.12	52	-33870.00	20.93	40.26
10	22.00	37.00	60080.00	43.28	896.72	67	-34630.00	11.90	17.76

上图为参数优化报表的一部分。您可以选定某字段上的部分数据点生成报表的按钮，从而生成如下图的个性化报表图型：

交易策略参数优化[SR888 - 1日线]

#	P:boLength	P:fsLength	胜率	盈亏比率	平均利润	交易手数
1*	22.00	57.00	7.54	1071.48	61	



有三种不同类型的图表可选择。

## 优化结果的选择

在优化完成后，系统会把当前计算得到的最优结果直接代入当前图表的系统中，从而以新的一组参数计算得出的讯号标识在图表上。一旦关掉此图表，再次调用此公式应用，仍将执行原来的参数。

如需要将优化参数的结果保存下来的话，可以直接在图表公式名称上右键菜单里，属性里的参数点击“设为默认”。此时会将新的参数代入公式重新进行编译，从而将新的一组参数结果设为默认值了。

在选择参数优化的结果时，我们要注意避免优化结果的偶然性，即要观察最优的一组参数与排位接近的几组参数之间是否平滑。比如，最优一组参数与后面几组参数结果相比有一定的跳跃性，就需要人为去判断一下最优的这组参数是否可取了。

如下图，参数 B 的最优值为 33，但后面一段的排位中，参数 B 基本都是接近 20 的数值。这里我们就要考虑一下这个 33 的结果，是否因为历史里的某段特殊行情从而导致的，这样的偶发性的结果是否应该将其应用于未来的实战交易中。

#	参数 A	参数 B
1	10	33
2	10	22
3	12	22
4	12	23
5	9	20
6	10	21
7	10	21
.....	.....	.....

## ●TradeBlazer 公式策略进阶

交易策略的代码写法会因为交易思想及编程习惯因人而异，没有一个固定的模式可寻。为方便交易者快速地建立自己的交易策略，在此按常用的功能点列出部分策略的代码示例，用户可根据自己的需要选择对应的代码进行组合。

### 止赢止损

模板以止赢 30 跳，止损 20 跳为例，也可以转换为开仓价格的百分比值，或其任何设置的变量进行处理。

#### Vars

```
Numeric MinPoint;           // 一个最小变动单位，也就是一跳
Numeric MyEntryPrice;       // 开仓价格，本例是开仓均价，也可根据需要设置为某次入的价格
Numeric TakeProfitSet(30);  // 止赢设置
Numeric StopLossSet(20);    // 止损设置
Numeric MyExitPrice;        // 平仓价格
```

#### Begin

```
...
MinPoint = MinMove*PriceScale;
MyEntryPrice = AvgEntryPrice;
If(MarketPosition==1) // 有多仓的情况
{
    If(High >= MyEntryPrice + TakeProfitSet*MinPoint) // 止赢条件表达式
    {
        MyExitPrice = MyEntryPrice + TakeProfitSet*MinPoint;
        If(Open > MyExitPrice) MyExitPrice = Open;      // 如果该 Bar 开盘价有跳空触发，
                                                         用开盘价代替
    }
}
```

```

        Sell(0,MyExitPrice);
    }else if(Low <= MyEntryPrice - StopLossSet*MinPoint)// 止损条件表达式
    {
        MyExitPrice = MyEntryPrice - StopLossSet*MinPoint;
        If(Open < MyExitPrice) MyExitPrice = Open;        // 如果该 Bar 开盘价有跳空触发,
                                                         则用开盘价代替

        Sell(0,MyExitPrice);
    }
}
}else if(MarketPosition==1) // 有空仓的情况
{
    If(Low <= MyEntryPrice - TakeProfitSet*MinPoint) // 止赢条件表达式
    {
        MyExitPrice = MyEntryPrice - TakeProfitSet*MinPoint;
        If(Open < MyExitPrice) MyExitPrice = Open;        // 如果该 Bar 开盘价有跳空触发,
                                                         则用开盘价代替

        BuyToCover(0,MyExitPrice);
    }
}
}else if(High >= MyEntryPrice + StopLossSet*MinPoint)// 止损条件表达式
{
    MyExitPrice = MyEntryPrice + StopLossSet*MinPoint;
    If(Open > MyExitPrice) MyExitPrice = Open;        // 如果该 Bar 开盘价有跳空触发,
                                                         则用开盘价代替

    BuyToCover(0,MyExitPrice);
}
}
...
End

```

#### 注意事项:

1. 因无法确认开仓 **Bar** 最高/低价和开仓价的先后顺序, 因此以上写法一般忽略开仓 **Bar** 的处理。
2. 如果某个 **Bar** 最高/低价相差很大, 可能出现止赢止损同时满足的情况, 这种情况下需要切换到更小的周期进行交易, 或者扩大止赢/损幅度。

## 跟踪止损

跟踪止损有很多种方式，本模板的规则如下：当盈利达到 50 跳之后启动第一级跟踪止损，止损的回撤值为 30 跳，当盈利达到 80 跳之后启动第二级的跟踪止损，止损的回撤值为 20 跳。您也可以将这些固定的设置修改为盈利百分比，或者是某个价格的百分比。

### Vars

```
Numeric MinPoint;           // 一个最小变动单位，也就是一跳
Numeric MyEntryPrice;       // 开仓价格，本例是开仓均价，也可根据需要设置为某次入场
                             的价格
Numeric TrailingStart1(50); // 跟踪止损启动设置 1
Numeric TrailingStart2(80); // 跟踪止损启动设置 2
Numeric TrailingStop1(30);  // 跟踪止损设置 1
Numeric TrailingStop2(20);  // 跟踪止损设置 2
Numeric StopLossSet(50);    // 止损设置
Numeric MyExitPrice;        // 平仓价格
NumericSeries HighestAfterEntry; // 开仓后出现的最高价
NumericSeries LowestAfterEntry;  // 开仓后出现的最低价
```

### Begin

```
...
If(BarsSinceentry == 0)
{
    HighestAfterEntry = Close;
    LowestAfterEntry = Close;
    If(MarketPosition <> 0)
    {
        HighestAfterEntry = Max(HighestAfterEntry,AvgEntryPrice); // 开仓的 Bar，将开
                                                                    仓价和当时的收盘价的较大值保留到 HighestAfterEntry
        LowestAfterEntry = Min(LowestAfterEntry,AvgEntryPrice);    // 开仓的 Bar，将开
                                                                    仓价和当时的收盘价的较小值保留到 LowestAfterEntry
    }
}
```

```

    }else
    {
        HighestAfterEntry = Max(HighestAfterEntry,High); // 记录下当前 Bar 的最高点，用于下
                                                    一个 Bar 的跟踪止损判断
        LowestAfterEntry = Min(LowestAfterEntry,Low);    // 记录下当前 Bar 的最低点，用于下
                                                    一个 Bar 的跟踪止损判断
    }
    Commentary("HighestAfterEntry="+Text(HighestAfterEntry));
    Commentary("LowestAfterEntry="+Text(LowestAfterEntry));
    MinPoint = MinMove*PriceScale;
    MyEntryPrice = AvgEntryPrice;
    If(MarketPosition==1) // 有多仓的情况
    {
        If(HighestAfterEntry[1] >= MyEntryPrice + TrailingStart2*MinPoint) // 第二级跟踪止损
                                                    的条件表达式
        {
            If(Low <= HighestAfterEntry[1] - TrailingStop2*MinPoint)
            {
                MyExitPrice = HighestAfterEntry[1] - TrailingStop2*MinPoint;
                If(Open < MyExitPrice) MyExitPrice = Open;    // 如果该 Bar 开盘价有跳空
                                                            触发，则用开盘价代替
                Sell(0,MyExitPrice);
            }
        }
    }else if(HighestAfterEntry[1] >= MyEntryPrice + TrailingStart1*MinPoint)// 第一级跟踪止
                                                    损的条件表达式
    {
        If(Low <= HighestAfterEntry[1] - TrailingStop1*MinPoint)
        {
            MyExitPrice = HighestAfterEntry[1] - TrailingStop1*MinPoint;
            If(Open < MyExitPrice) MyExitPrice = Open;    // 如果该 Bar 开盘价有跳空
                                                            触发，则用开盘价代替
            Sell(0,MyExitPrice);
        }
    }

```

```

    }
}
else if(Low <= MyEntryPrice - StopLossSet*MinPoint)//可以在这里写上初始的止损处理
{
    MyExitPrice = MyEntryPrice - StopLossSet*MinPoint;
    If(Open < MyExitPrice) MyExitPrice = Open;    // 如果该 Bar 开盘价有跳空触发,
    则用开盘价代替
    Sell(0,MyExitPrice);
}
}
else if(MarketPosition== -1) // 有空仓的情况
{
    If(LowestAfterEntry[1] <= MyEntryPrice - TrailingStart2*MinPoint)    // 第二级跟踪止损
    的条件表达式
    {
        If(High >= LowestAfterEntry[1] + TrailingStop2*MinPoint)
        {
            MyExitPrice = LowestAfterEntry[1] + TrailingStop2*MinPoint;
            If(Open > MyExitPrice) MyExitPrice = Open;    // 如果该 Bar 开盘价有跳空
            触发, 则用开盘价代替
            BuyToCover(0,MyExitPrice);
        }
    }
    else if(LowestAfterEntry[1] <= MyEntryPrice + TrailingStart1*MinPoint)// 第一级跟踪止
    损的条件表达式
    {
        If(High >= LowestAfterEntry[1] + TrailingStop1*MinPoint)
        {
            MyExitPrice = LowestAfterEntry[1] - TrailingStop1*MinPoint;
            If(Open > MyExitPrice) MyExitPrice = Open;    // 如果该 Bar 开盘价有跳空
            触发, 则用开盘价代替
            BuyToCover(0,MyExitPrice);
        }
    }
    else If(High >= MyEntryPrice + StopLossSet*MinPoint)    //可以在这里写上初始的
    止损处理

```



```

{
    MyExitPrice = MyEntryPrice + StopLossSet*MinPoint;
    If(Open > MyExitPrice) MyExitPrice = Open;      // 如果该 Bar 开盘价有跳空触发,
                                                    则用开盘价代替

    BuyToCover(0,MyExitPrice);
}
}
...
End

```

注意事项:

1. 因无法确认开仓 **Bar** 最高/低价和开仓价的先后顺序, 因此以上写法一般忽略开仓 **Bar** 的处理。
2. 如果某个 **Bar** 最高/低价相差很大, 可能出现创新高之后跟踪止损的情况, 但系统无法确认最高价和最低价的先后顺序, 因此本模板只用前一个 **Bar** 的最高/低价计算最大赢利位置。

## 加仓减仓

本例仅以做多为例, 做空类似。模板以首次开仓 2 手后每赢利 30 跳加仓一次, 每次 1 手, 最多加仓 3 次; 开仓后每亏损 30 跳减仓 1 手。也可以转换为开仓价格的百分比值, 或波动率的百分比等其任何设置的变量进行处理。

Vars

```

Numeric MinPoint;           // 一个最小变动单位, 也就是一跳
NumericSeries firstPrice;    // 第一次开仓价格
NumericSeries LastPrice;     // 最后一次开仓价格
Numeric AddSet(30);          // 加仓设置
Numeric SubSet(30);          // 减仓设置
Bool FirstEntryCon;          // 首次开仓条件

```

Begin

```
FirstEntryCon = ...
```

```
MinPoint = MinMove*PriceScale;
```

```
If(MarketPosition==0)
```

```
{
```

```
    If(FirstEntryCon)
```

```
    {
```

```
        firstPrice = Open;
```

```
        LastPrice = firstPrice;
```

```
        Buy(2,firstPrice);
```

```
    }
```

```
}else If(MarketPosition==1) // 有多仓的情况
```

```
{
```

```
    While(CurrentEntries < 4 && High >= LastPrice + AddSet*MinPoint) // 加仓
```

```
    {
```

```
        LastPrice = LastPrice + AddSet*MinPoint;
```

```
        if(Open > LastPrice) LastPrice = Open;
```

```
        Buy(1,LastPrice);
```

```
    }
```

```
    While(CurrentEntries > 0 && Low <= firstPrice - SubSet*MinPoint) // 减仓
```

```
    {
```

```
        firstPrice = firstPrice - SubSet*MinPoint;
```

```
        if(Open < firstPrice) firstPrice = Open;
```

```
        Sell(1,firstPrice);
```

```
    }
```

```
}
```

```
...
```

End

注意事项:

1. 因无法确认开仓 Bar 最高/低价和开仓价的先后顺序，忽略开仓 Bar 的加减仓处理。

2. 如果某个 Bar 最高/低价相差很大，可能出现加仓减仓同时满足的情况，这种情况下需要切换到更小的周期进行交易，或者扩大加仓/减仓幅度设置。

## 多品种交易

模板以常用的双均线系统为例，对主图商品和叠加商品分别进行交易。

### Params

```
Numeric FastLength1(5);           // Data0 的短周期参数
Numeric SlowLength1(20);          // Data0 的长周期参数
Numeric FastLength2(5);           // Data1 的短周期参数
Numeric SlowLength2(20);          // Data1 的长周期参数
```

### Vars

```
NumericSeries AvgValue11;
NumericSeries AvgValue12;
NumericSeries AvgValue21;
NumericSeries AvgValue22;
```

### Begin

```
AvgValue11 = AverageFC(Data0.Close, FastLength1);
AvgValue12 = AverageFC(Data0.Close, SlowLength1);
AvgValue21 = AverageFC(Data1.Close, FastLength2);
AvgValue22 = AverageFC(Data1.Close, SlowLength2);
If(Data0.MarketPosition <> 1 && AvgValue11[1] > AvgValue12[1])
{
    Data0.Buy(1, Data0.Open);
}
If(Data0.MarketPosition <> -1 && AvgValue11[1] < AvgValue12[1])
{
    Data0.SellShort(1, Data0.Open);
}
If(Data1.MarketPosition <> 1 && AvgValue21[1] > AvgValue22[1])
{
```

```

        Data1.Buy(1,Data1.Open);
    }
    If(Data1.MarketPosition <>-1 && AvgValue21[1] < AvgValue22[1])
    {
        Data1.SellShort(1,Data1.Open);
    }
End

```

注意事项：

1. 针对不同的商品的数据进行计算或交易，需通过 **Data#** 这样的方式添加前缀，**Data0** 可与省略不写。
2. **Data#** 的顺序和超级图表中商品设置界面的顺序相同，必须要叠加足够的商品才能保证代码正常执行。

## 集合竞价数据过滤

集合竞价时，会产生一个 Tick，这个 Tick 会驱动超级图表计算交易策略，如果条件满足，则会马上发送委托单，但此时交易所并未开市，就会产生废单，为了处理这种情况，可以采取下面方法：

```

Begin
    If(BarStatus==2 && Time==0.090000 && High==Low) return;           // 第一种写法
    If(BarStatus==2 && Time==0.090000 && CurrentTime < 0.090000) return; // 第二种写法
    If(BarStatus==2 && high==low) return;                               // 第三种写法
    If(BarStatus==2 && CurrentTime < 0.090000) return;                 // 第四种写法
    ...
End

```

注意事项：

1. 本例是以国内商品期货交易所开市时间举例，股指期货或其他市场需调整写法。
2. 在日线以上的 **Bar** 数据，前者种写法均不适用，可选择第三或第四种写法。

3.若按第一、三种写法，当商品的高低价不产生变化时，会忽略这些 **Tick**，即使是已经开市；若按第二、四种写法需保证本机的时间准确。

## 收盘平仓

收盘平仓分为两部分，一部分负责处理历史测试，一部分负责处理实时交易。在测试时我们可以以每天的收盘价平仓，在实时交易时我们选择 **14: 59** 分平仓。

Begin

```
...  
If(Date[-1]!=InvalidInteger && Date!=Date[-1])|((Date[-1]==InvalidInteger &&  
Date < CurrentDate))  
{  
    Sell(0,Close);  
    BuyToCover(0,Close);  
}Else If(Date==CurrentDate && Time==0.1455 && CurrentTime>=0.1459)  
{  
    Sell(0,Open);  
    BuyToCover(0,Open);  
}  
...
```

End

注意事项：

1. 本例是以国内商品期货交易所收市时间举例，股指期货或其他市场需调整写法。
2. 本例是针对 5 分钟周期的收盘平仓所写，针对不同的周期需改写为合适的最后 **Bar** 时间。

## A 函数下单撤单和全局变量操作

本例在每天收盘前 N 分钟的时候自动撤掉超级图表中商品的挂单，并全部平仓。通过 A\_SendOrder 进行下单，A\_DeleteOrder 进行撤单，并使用全局变量记录 Tick 计数和撤单标志。

### Params

```
Numeric offSet(1);           // 委托价格偏移，为了保证成交
Numeric BeforeMins(5);       // 收盘前几分钟开始操作
```

### Vars

```
Numeric tempPos; // 仓位
Numeric DeleteOrderTickCount;
Numeric HasSendOrder(0);
```

### Begin

```
If(BarStatus == 0)
{
    DeleteOrderTickCount = 9999;
    HasSendOrder = 0;
    SetGlobalVar(0,DeleteOrderTickCount);
    SetGlobalVar(1,HasSendOrder);
}Else
{
    DeleteOrderTickCount = GetGlobalVar(0);
    HasSendOrder = GetGlobalVar(1);
}

If(CurrentTime > (0.1459 - 0.0001*(BeforeMins-1)) && BarStatus == 2 && HasSendOrder == 0)
{
    If(Data0.Close != InvalidNumeric && Data0.A_GetOpenOrderCount(>0) //商品 0 全部撤单
    {
        Data0.A_DeleteOrder();
        DeleteOrderTickCount = 1;
    }
}
```

```
If(Data1.Close != InvalidNumeric && Data1.A_GetOpenOrderCount(>0)//商品 1 全部撤单
{
    Data1.A_DeleteOrder();
    DeleteOrderTickCount = 1;
}
If(Data2.Close != InvalidNumeric && Data2.A_GetOpenOrderCount(>0) //商品 2 全部撤单
{
    Data2.A_DeleteOrder();
    DeleteOrderTickCount = 1;
}
DeleteOrderTickCount = DeleteOrderTickCount + 1;
SetGlobalVar(0,DeleteOrderTickCount);
If(DeleteOrderTickCount < 5) Return; // 撤单后需要延迟几个 Tick 才平仓
tempPos = Data0.A_BuyPosition();
If(tempPos > 0) // 平多单
{
    Data0.A_SendOrder(Enum_Sell,Enum_Exit,tempPos,Data0.Q_BidPrice
    - offSet* Data0.MinMove*Data0.PriceScale);
}
tempPos = Data0.A_SellPosition();
If(tempPos > 0) //平空单
{
    Data0.A_SendOrder(Enum_Buy,Enum_Exit,tempPos,Data0.Q_AskPrice
    +offSet*Data0.MinMove*Data0.PriceScale);
}
tempPos = Data1.A_BuyPosition;
If(tempPos > 0) // 平多单
{
    Data1.A_SendOrder(Enum_Sell,Enum_Exit,tempPos,Data1.Q_BidPrice
    -offSet*Data1.MinMove*Data1.PriceScale);
}
```

```
tempPos = Data1.A_SellPosition;
If(tempPos > 0) //平空单
{
    Data1.A_SendOrder(Enum_Buy,Enum_Exit,tempPos,Data1.Q_AskPrice
        +offset*Data1.MinMove*Data1.PriceScale);
}
tempPos = Data2.A_BuyPosition;
If(tempPos > 0) // 平多单
{
    Data2.A_SendOrder(Enum_Sell,Enum_Exit,tempPos,Data2.Q_BidPrice
        -offset*Data2.MinMove*Data2.PriceScale);
}
tempPos = Data2.A_SellPosition;
If(tempPos > 0) //平空单
{
    Data2.A_SendOrder(Enum_Buy,Enum_Exit,tempPos,Data2.Q_AskPrice
        +offset*Data2.MinMove*Data2.PriceScale);
}
HasSendOrder = 1;
SetGlobalVar(1,HasSendOrder);
}
End
```

**注意事项:**

1. 本例是以国内商品期货交易所收市时间举例，股指期货或其他市场需调整写法。
2. 本例假设撤单后 5 个 Tick 委托状态能同步成功，实际情况中因网络延时等原因并不一定能保证成功。



## 数据库读写

本例以 5 分钟周期调用日线指标数据举例讲解具体应用。

操作步骤如下：

1. 新建一个工作区，包含上下两个图表窗体，上面选择日线周期，下面选择 5 分钟周期。
2. 新建一个公式应用，命名为 **MyDayMA**。编译成功后插入日线图表中。详细代码如下：

### Params

Numeric length(10);

### Vars

Numeric MA;

string strKey;

string strValue;

### Begin

MA = AverageFC(Close,length);

strKey = DateToString(Date);

strValue = Text(MA);

SetTBProfileString("DayMA",strKey,strValue);

PlotNumeric("MA",MA);

### End

3. 新建一个公式应用，**My5MinMA**。编译成功后插入 5 分钟图表中，详细代码如下：

### Vars

NumericSeries DayMAValue;

string strKey;

string strValue;

Begin

```
    strKey = DateToString(Date);
    strValue = GetTBProfileString("DayMA",strKey);
    If(strValue != InvalidString)
    {
        DayMAValue = Value(strValue);
    }Else
    {
        DayMAValue = DayMAValue[1];
    }
    PlotNumeric("DayMA",DayMAValue);
```

End

4. 上面的公式实际使用了未来数据，用来写技术分析是可以的，但用来进行自动交易就会出问题，为了更准确合理的使用跨周期数据，我们应该稍作修改，代码如下：

Vars

```
    NumericSeries DayMAValue;
    StringSeries strKey;
    string strValue;
```

Begin

```
    If(Date!=Date[1])
    {
        strKey = DateToString(Date[1]);
    }Else
    {
        strKey = strKey[1];
    }
    strValue = GetTBProfileString("DayMA",strKey);
    If(strValue != InvalidString)
    {
        DayMAValue = Value(strValue);
```

```
    }Else
    {
        DayMAValue = DayMAValue[1];
    }
    PlotNumeric("DayMA",DayMAValue);
End
```

注：在“文件---数据管理----配置工具”中可以看到目前已写入数据库的所有数据。您可以通过相对应的块名与键名来查看相对应的存放的数据值，甚至可以手工去添加、删减以及修改里面存放的数据，这样相应公式所读取的值也会随之改变。

## 平仓延迟反手

### Params

```
Numeric FastLength(5);
Numeric SlowLength(20);
Numeric DelayTicks(5);
```

### Vars

```
NumericSeries AvgValue1;
NumericSeries AvgValue2;
Numeric LastBarTime;
Numeric TickCounter;
```

### Begin

```
AvgValue1 = AverageFC(Close,FastLength);
AvgValue2 = AverageFC(Close,SlowLength);
LastBarTime = GetGlobalVar(0);
TickCounter = GetGlobalVar(1);
If(BarStatus==2 && LastBarTime != Time)
```

```
{
    LastBarTime = Time;
    TickCounter = 0;
}
If(MarketPosition <> 1 && AvgValue1[1] > AvgValue2[1])
{
    If(MarketPosition==0 || BarStatus!=2)
    {
        Buy(1,Open);
    }Else
    {
        BuyToCover(1,Open);
        If(TickCounter==0)
        {
            TickCounter = 1;
        }else If(TickCounter < DelayTicks)
        {
            TickCounter = TickCounter + 1;
        }else
        {
            Buy(1,Open);
        }
    }
}
If(MarketPosition <> -1 && AvgValue1[1] < AvgValue2[1])
{
    If(MarketPosition==0 || BarStatus!=2)
    {
        SellShort(1,Open);
    }Else
    {
        Sell(1,Open);
        If(TickCounter==0)
        {
```

```
        TickCounter = 1;
    }else If(TickCounter < DelayTicks)
    {
        TickCounter = TickCounter + 1;
    }else
    {
        SellShort(1,Open);
    }
}
SetGlobalVar(0,LastBarTime);
SetGlobalVar(1,TickCounter);
End
```

## ●策略性能测试与参数优化的具体计算公式

交易开拓者的策略性能测试及参数优化有很多的测试项目与计算结果，并非每一个交易者都会用到其全部的数据，交易者只需选择自己所需的参考数据即可。为方便交易者理解报告上各个数据的含义，现在将其计算公式整理如下：

### 交易策略性能测试报告

**净利润：** 绝对值（总盈利金额 - 总亏损金额）（盈利为黑色数字，亏损为红色）

**总盈利：** 总交易盈利金额 - 手续费

**总亏损：** 绝对值（总交易亏损金额 - 手续费）显示为红色

**总盈利/总亏损：** 绝对值（总盈利 / 总亏损）

**交易手数：** 总的交易数量

**盈利比率：** 盈利手数 / 总交易手数

**盈利手数：** 盈利交易的总手数

**亏损手数：** 亏损交易的总手数

**持平手数：** 持平交易的总手数

**平均利润：** 净利润 / 交易手数

**平均盈利：** 总盈利金额 / 盈利交易手数

**平均亏损：** 总亏损金额 / 亏损交易手数（显示为红色）

**平均盈利/平均亏损：** 绝对值（平均盈利/平均亏损）

**最大盈利：** 盈利最大的单次交易的盈利金额

**最大亏损：** 绝对值（亏损最大的单次交易的亏损金额）

**最大盈利/总盈利：** 如字面所示

**最大亏损/总亏损：** 如字面所示

**净利润/最大亏损：** 如字面所示

**最大连续盈利手数：** 若将手数设为固定一手，则此手数也就是最大连续盈利次数

**大连续亏损手数：** 若将手数设为固定一手，则此手数也就是最大连续亏损次数

**平均持仓周期：** 总交易的 bar 的总数 / 交易次数

**平均盈利周期：** 总盈利交易的 bar 的总数 / 盈利交易次数

**平均亏损周期：** 总亏损交易的 bar 的总数 / 亏损交易次数

**平均持平周期：** 总持平交易的 bar 的总数 / 持平交易次数

**最大使用资金：** 以 bar 的收盘价来计算的最大持仓保证金（组合测试时为多个策略共同 计算的最大使用资金量）

**佣金合计：** 总共的佣金金额

**收益率：** 净利润 / 初始资金

**年度收益率：** 有效收益率 / （总交易的天数 / 365）

**有效收益率：** 净利润 / 最大使用资金

**月度平均盈利：** 净利润 / 总交易的天数 × 30.5

**收益曲线斜率：** 根据交易盈亏曲线拟合的趋势线的斜率

**收益曲线截距：** 根据交易盈亏曲线拟合的趋势线的截距

**收益曲线 R 平方值：** 根据交易盈亏曲线拟合的趋势线与收益曲线之间相关系数的平方（具体计算方式可查阅 EXCLE 表格中 R 平方值的算法）

**总交易时间：** 参与测试的全部 bar 数据的天数

**持仓时间比率：** 持仓 BAR 数量 / 总 bar 数量

**持仓时间：** 总交易时间 \* 持仓时间比率

**最大空仓时间：** 没有持仓的天数

**持仓周期：** 持仓的 BAR 数量

**资产最大升水：** 最高点金额 - 前期低点的金额

**发生时间：** 高点发生的所在 BAR 的日期与时间

**最大升水/前期低点：** 如字面所示

最大资产回撤值（按 Bar 收盘计算）

回撤值： 前期高点 - 低点

发生时间： 低点发生所在 bar 的日期与时间

回撤值/前期高点：（前期高点 - 低点） / 前期高点

净利润/回撤值： 净利润 / 回撤值

最大资产回撤值比例（按 Bar 收盘计算）

回撤值： 前期高点 - 低点

发生时间： 低点发生所在 bar 的日期与时间

回撤值/前期高点：（前期高点 - 低点） / 前期高点

净利润/回撤值： 净利润 / 回撤值

注意：上述两个回撤的区别在于，前者是按回撤金额的最大值来计算，而后者是按回撤比例的最大值来计算的。比如说，一个原始金额为 10 万的帐户，在刚开始交易的一段时间就发生了一个 4 万的回撤。而此帐户在交易一段时间后，总金额增长到了 50 万，此时发生了一个 8 万的回撤。如果以金额回撤来计算，是后面这个 8 万的回撤大。但是以比例来计算，则是前面那个 4 万的回撤大。

## 交易策略参数优化报告

净利润： 绝对值（总盈利金额 - 总亏损金额）

年化收益： 净利润 / 总交易的天数 × 365

盈利比率： 盈利手数 / 总交易手数

平均利润： 净利润 / 交易手数

交易手数： 总交易手数

最大资产回撤： 低点-前期高点

TB 系数： （平均利润×平均利润×交易次数） / （平均盈利×平均亏损）

增长系数： 根据交易盈亏曲线拟合的趋势线的斜率

收益风险比： 年度收益 / 最大资产回撤



**R 平方值：** 根据交易盈亏曲线拟合的趋势线与收益曲线之间相关系数的平方（具体计算方式可查阅 EXCLE 表格中 R 平方值的算法）

**置信度：** 根据测试的交易次数计算的置信水平，计算公式为： $1-1/\text{Sqrt}(\text{交易次数})$ ;

**头寸系数：** 收益风险比 $\times$ R 平方值 $\times$ 置信度 / 最大资产回撤

**资产回撤计数：** 资产回撤发生的次数（是以超过最大回撤基准线以上的回撤来计算的，此基准线在“全局交易设置中”进入设置）

**平均资产回撤：** 资产回撤总金额 / 资产回撤计数（都是以超过最大回撤基准线以上的回撤来计算的，此基准线在“全局交易设置中”进入设置）

**调整收益风险比：** 年度收益 / 平均资产回撤 (年度收益 = 净利润 / 总交易时间 \* 365)

**夏普比率：** 量化收益与风险的比值，具体算法参考互联网上的夏普比率计算方法

**盈亏比：** 平均盈利 / 平均亏损

**总盈利：** 总交易盈利金额 - 手续费

**总亏损：** 总交易亏损金额 - 手续费

**盈利手数：** 盈利交易的总手数

**亏损手数：** 亏损交易的总手数

**连续盈利手数：** 如字面所示

**连续亏损手数：** 如字面所示

**最大盈利：** 盈利最大的单次交易的盈利金额

**最大亏损：** 亏损最大的单次交易的亏损金额

**平均盈利：** 总盈利 / 盈利交易手数

**平均亏损：** 总亏损金额 / 亏损交易手数

**平均盈利周期：** 总盈利交易的 bar 的总数 / 盈利交易手数

**平均亏损周期：** 总亏损交易的 bar 的总数 / 亏损交易手数

**盈利因子：** 总利润 / 总亏损

**最大资产回撤比率%：** 最大资产回撤 / 前期高点

## ●公式编写常见问题

我们把客户在编写公式的过程中常常会出现一些错误，本章节我们把一些出现频率较高的问题集中起来，以 Q & A 的形式整理出来，以供大家参考。

**Q1: 为什么公式的开平仓不受条件的限制，每一个 K 线上都有开、平仓的动作？**

**A1:** 在 `if(.....)` 条件判断语句的后面不小心加上分号，就会导致后面的执行不受条件的控制了。注意一定要把 `if(.....);` 后面这里的“;”去掉。

**Q2: 自己建的公式没有编译，但在公式编辑器中不能编译啊，编译按钮是灰的，该怎么做？**

**A2:** 您只需要在公式代码中任意位置加入一个空格，然后再删掉这个空格，就可以进行编译了。这是因为该公式 以前被保存了过，如果代码没有被修改过，就不能触发进行校验保存的编译。

**Q3: AverageFC 和 Average 有什么区别？**

**A3:** 您可以从以下三个方面了解此两类函数的区别：

- 1、Average 和 AverageFC 都是内建的用户函数，目的都是用来求 N 个 Bar 以来的平均值， 您可以直接看到实现的代码。
- 2、AverageFC 是指 FastCalculate,即快速计算。当这两个函数的第二个参数，即 N 个 Bar 是常量时，使用 AverageFC,提高计算效率。当 N 是不确定的变量时，则必须使用 Average，否则会出现计算问题。
- 3、系统里面类似的用户函数还有 Summation 和 SumamtionFC，Highest 和 HighestFC，Lowest 和 LowestFC 等。

**Q4: 为什么出现最终目标文件编译错误？**

**A4:** 目前发现有以下几种情况会导致这个问题出现：

- 1、有中文字符的存在，特别是一些细节的不容易发现的中文标点符号。
- 1、公式管理器中存在未通过编译的，有严重逻辑错误的公式，需删掉这些有错误的公式。
- 2、用了一些 C++ 的关键字来命名变量，比如 `switch,case,int,Public,protected,class,long,double....` 有好几百个，可以考虑加上一些前缀，比如 `My****`，这样就可以了。

**Q5: 总是报“锁定编译目标文件超时”是什么原因？**

**A5:** 有两种可能：

- 1、已经打开的图表调用了技术指标或交易指令，并且行情更新较快，导致编译时覆盖旧文件失败。这个时候，您可以关闭先所有的图表窗体在试试看。
- 2、可能是公式的写法有问题，是系统现在还不能识别的错误。您可以另外写一个简单的公式看看能不能编译通过，如果能通过，那就证明是这个公式有问题。如果不是，那我也不知道具体原因。（您可以考虑导出您自己的公式，然后删掉安装目录下 **User** 目录下您所在用户名的 **formula**，然后再复制一个干净的 **formula** 进去。）

**Q6: 为什么已经设置了启动多帐户全自动交易，结果只有一个帐户进行了交易的动作？**

**A6:** 如果是 **A\_SendOrder** 函数所写的发单指令，那是是不可以设置进行启动多帐户自动交易的。您可以多开几个图表窗口，同一个应用公式分别单独对应一个交易帐户设置自动交易即可。

**Q7: 同一品种不同超级图表上的交易指令是否相互作用？**

**A7:** 不会相互作用。现在即使同一个超级图表上，指令之间都不会相互作用的。

**Q8: Time 与 CurrentTime 的区别在于哪里？**

**A8:** **Time** 是 Bar 数据的时间，在交易开拓者中，是以当前 Bar 所开始的那个时间做为 bar 的 **Time**，每一个 Bar 上的 **time** 是确定且唯一的。如果是日线的 Bar 上，**Time** 则是为 0。而 **CurrentTime** 则是电脑操作系统的时间，基本上与北京时间是接近一致的。

**Q9: 如何记录开仓价格？**

**A9:** 有多种方法可以尝试，下面例出几种方法以供参考：

1. 使用现有的系统函数：**entryprice**、**lastentryprice**、**aventryprice**；
2. 声明一个序列变量，在开仓时，将开仓价格赋值给此序列变量，并将此值传递下来；
3. 可以使用全局变量记录下开仓价格，以备随时取用。

**Q10: 我启动多帐户的自动交易，为什么只有一个帐户持行了交易？**

**A10:** 首先检查确认是否有关联多个交易帐户且这些帐户都处于联机状态。其次，查看公式代码里发出指令的函数，如果是使用 **A\_sendorder** 所编写的公式，则不可以启动多帐户自动交易。您可以打开多个超级图表，调用同一个合约同一个公式策略，再分别关联不同的交易帐户进行自动交易即可。