前言

目录

第一部分 深入浅出学习 TBL 语言

在第一部分我们将从经典的策略开始一步步了解 TBL 语言的语法和运行机制。

一、均线技术指标

均线是技术分析者经常使用的一个指标。那么在 TBL 语言中如何实现这个指标呢？其实简单一行代码就可以实现：

```
1 Events
2  OnBar(ArrayRef<Integer> indexs)
3  {
4   PlotNumeric("MA1",AverageFC(Close,5));
5   }
```

我们先讲下如何在 K 线图上显示这个指标，然后再讲下 TB 的公式机制，变量、参数和画图。

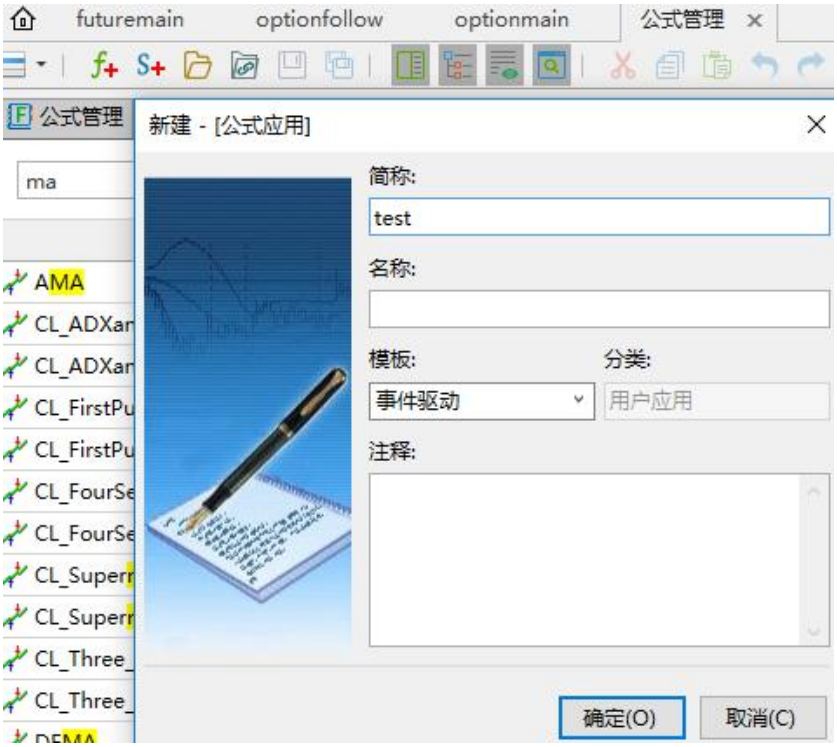
1.1、程序的基本结构

TB 语言是一个编译型语言（如 C 语言），不是解释类型的（如 Matlab，Python）。作为编译型语言的一种，TBL 语言运行的代码必须以公式的形成存在。一个公式必须有完整的必备要素。公式最基本的是程序主体。如上面代码的第 4 行就是程序的主体。而 1、2、3 和 5 行是程序主体的关键字和符号，用于标识程序主体的起始位置。

1.2、使用技术指标

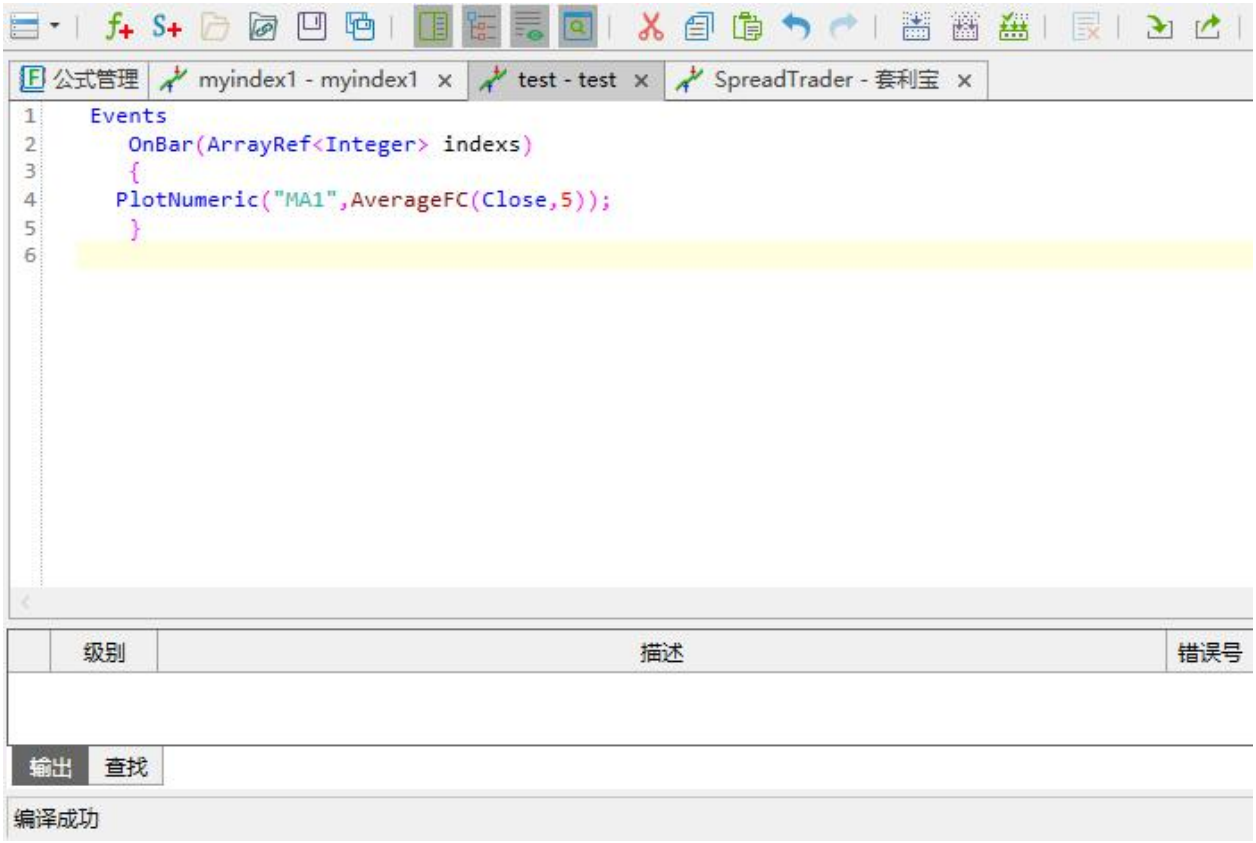
1、新建公式

打开公式管理器，点击 S+，弹出新建公式应用的窗口。输入公式的名称为 test。点确定。



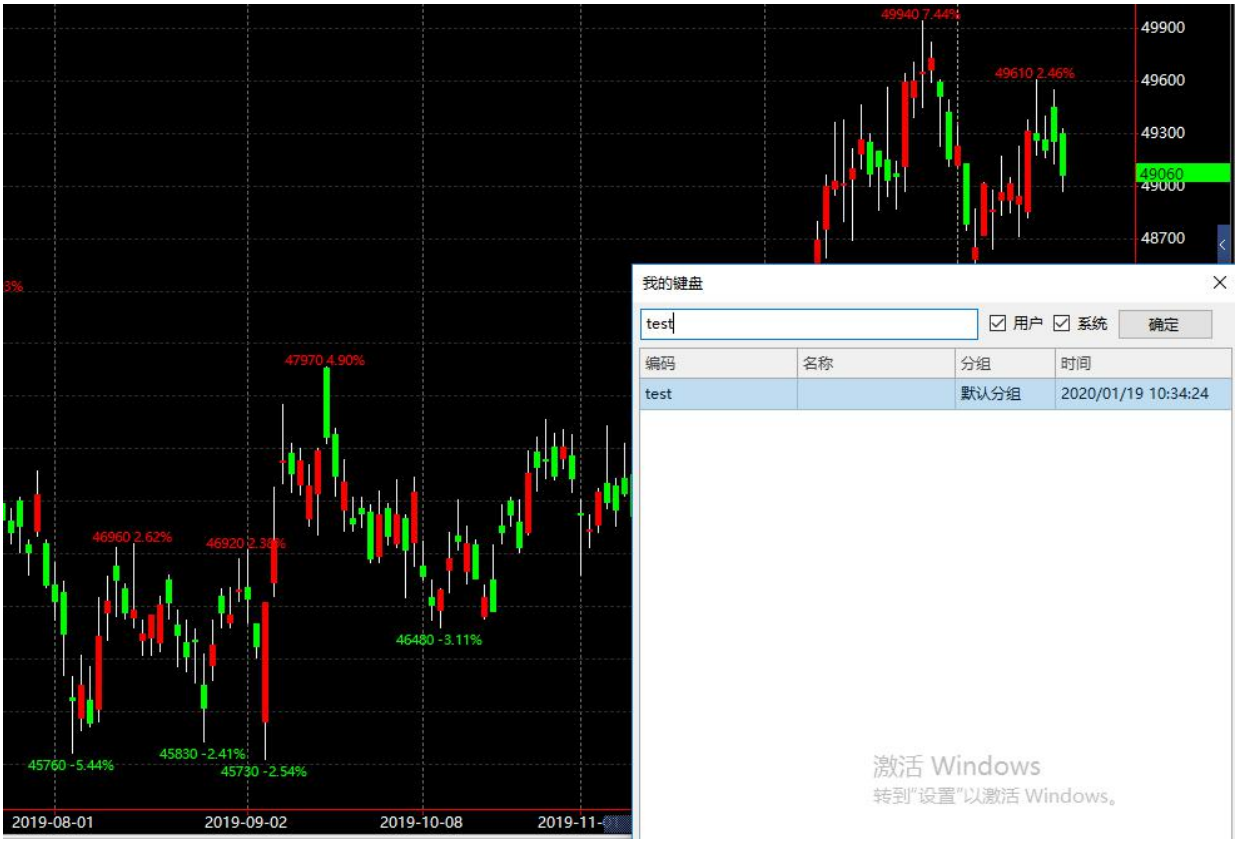
2、编译公式

我们贴入上面的 5 行代码，然后点击工具栏的编译按钮  。然后可以看到左下角有编译通过的提示。



3、在 K 线上显示指标

打开一张 K 线图，小键盘输入公式名 test。在右下角会弹出小键盘的界面，显示已经编译好的公式 test。



4、查看指标

成功加载公式之后，我们可以看到在 K 线图上面有一条线，就是单均线。

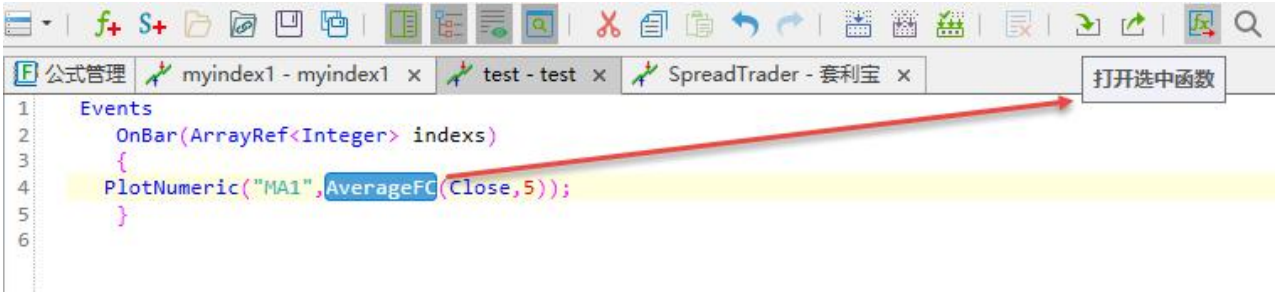
1.3、公式与函数

代码中的均线计算怎么完成的呢？

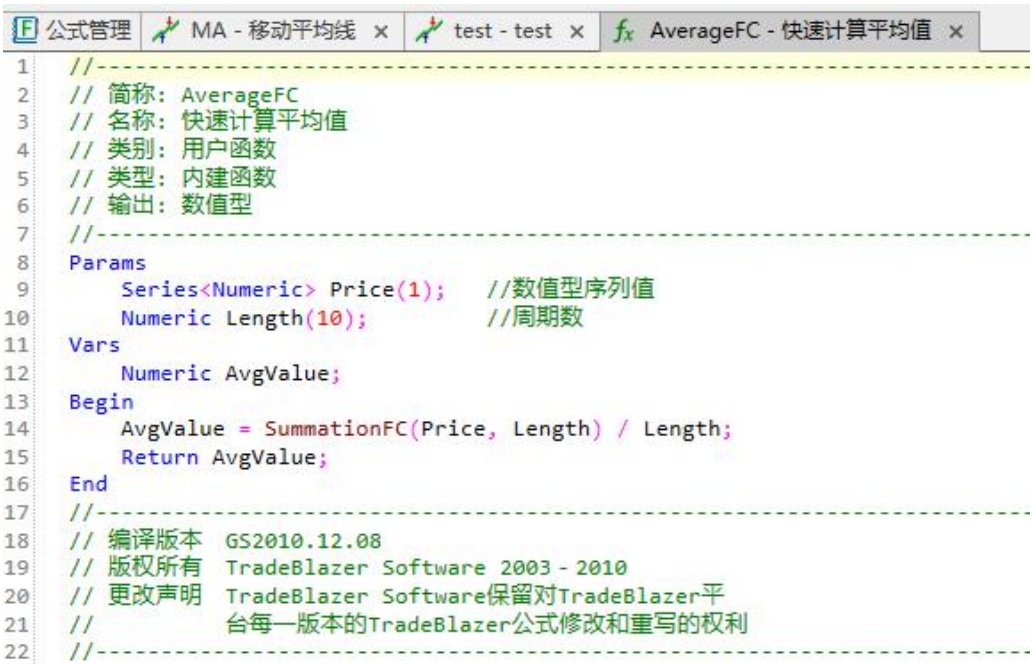
```
PlotNumeric("MA1",AverageFC(Close,5))
```

AverageFC(Close,5)完成了均线的计算，那么 AverageFC 是什么？

我们可以在代码界面选中这个 AverageFC，然后点击工具栏的图标查看。



打开选中函数之后，我们可以看到 AverageFC 是由以下代码组成。



这时候我们发现 test 和 AverageFC，一个实现了均线的展示，一个实现了均线的计算。
仔细看代码我们发现代码的结构也不一样。
Test 在 TBL 中是属于公式类型，AverageFC 在类型中显示是内建函数。

1、代码的类型

TBL 语言中的代码也就只有这两种基本形式公式和函数。公式和函数如果是系统提供的就是内建的，如果是用户自己编写的就是用户类型的。

2、公式和函数的结构

我们可以看到公式是有 events 和{}做标识的。函数是以 begin 和 end 做标识的。
函数有 return,公式没有。
函数本质上只是完成一个运算，把运算结果返回给公式。
细心的读者会发现，AverageFC 代码中还调用了另外一个函数 SummationFC。这个函数实现求和运算，运算完把结果返回给 AverageFC。

1.4、变量与参数

如果仔细看函数中还有两个关键字 params 和 vars。这两个关键字是什么作用？我们接下来讲解。
这两个关键字，params 是参数的标识关键字；vars 是变量的标识关键字。
对于一个计算机语言来说，参数和变量是基础的概念。

1、变量和参数的使用

对于 TBL 语言来说，用户可以在公式和函数中使用参数或者变量，也可以不使用。

2、变量的类型

TBL 的变量支持的基本类型有数值型 numeric、字符串类型 string、布尔类型 bool。以及进一步的扩展比如函数中使用的这个 Series<Numeric>。这是数值型序列。
序列类型和时间序列是对应的，比如收盘价，每根 BAR 有一个数值，随着时间的增加而增加。

3、参数的使用

参数是因为计算过程中有些数值不确定，可能改变，所以用参数代替。方便进一步的优化。
比如我们修改上面的 test 的代码，增加变量和参数。

```
Params
  Numeric FastLength(5);// 短期指数平均线参数
Vars
  Series<Numeric> AvgValue1;
Events
  OnBar(ArrayRef<Integer> indexs)
  {
    AvgValue1 = AverageFC(Close,FastLength);
    PlotNumeric("MA1",AvgValue1);
}
```

可以看到我们把均线长度，定义成了一个参数，fastlength。这样在计算均线的时候，就直接用 fastlength，而不是指定具体的数值。
而

```
Params
  Numeric FastLength(5);// 短期指数平均线参数
```

参数的默认值使用小括号括起来。
同样我们把均线计算的结果也用变量给先存储起来，这样画图的时候，就直接使用变量作为画图函数的参数。

```
Vars
  Series<Numeric> AvgValue1;
```

4、画图函数

PlotNumeric("MA1",AvgValue1);这一句代码完成了在 K 线上的画图。
画图函数只有两个是必须使用的参数。
第一个参数“MA1”指定画线的名称，第二个参数 AvgValue1 给出画线的数值。
画线函数还有其它的 plotstring,plotbool 等等，用户可以查看第二部分的详细介绍。

1.5、多均线指标

在上面的例子中我们讲解了如何画出一条均线。那么如何画出多条均线呢？
其实非常简单，我们把同样的代码复制黏贴多次，修改一下数值，就可以实现多条均线的指标。
比如 4 条均线的指标，我们可以打开系统自带的 MA 指标，查看它的代码如下：

```
Params
  Numeric Length1(5);
  Numeric Length2(10);
  Numeric Length3(20);
  Numeric Length4(30);


Events
  OnBar(ArrayRef<Integer> indexs)
  {
    PlotNumeric("MA1",AverageFC(Close,Length1));
```

```
    PlotNumeric("MA2",AverageFC(Close,Length2));
    PlotNumeric("MA3",AverageFC(Close,Length3));
    PlotNumeric("MA4",AverageFC(Close,Length4));
  }
```

二、四周规则交易策略

2.1、策略说明

量化交易策略很多是在主观交易的基础上发展而来的。主观交易时经常观察的各种趋势类指标比如均线或者通道，各种震荡类指标比如 RSI 或者 KDJ，都可以用来形成一些规则。而这些规则只要逻辑完整，有明确的开仓和平仓的逻辑规则。那么就可以进一步转换为量化交易策略。

四周规则是一个经典的趋势跟踪策略，并且绩效报告经久不衰。我们在本章通过对四周规则的介绍，逐步深入 TBL 语言的跨品种跨周期的强大功能。

交易规则：

当最新价突破过去 20 日的高点，买入；

当最新价突破过去 20 日的低点，卖出。

2.2、策略代码说明

由于四周策略的交易规则非常清晰，它的代码也比较简单，我们先贴出完整的代码然后逐一解释。

```
Vars
  Numeric highline;
  Numeric lowline;
Events
OnBar(ArrayRef<Integer> indexs)
{
  highline=Highest(High[1],20);
  lowline=Lowest(Low[1],20);
  If(MarketPosition<>1 And High>=highline)
    Buy(1,Max(Open,highline));
  If(MarketPosition<>-1 And Low<=lowline)
    SellShort(1,Min(Open,lowline));
}
```

1、交易函数

交易函数是用来执行交易的。它在实盘交易的时候，会完成相应的买卖委托。而为了让用户能明确的看到策略交易的操作，同时 TBQuant 会把这些交易信号标注在 K 线图上。而绩效测试报告也是根据图表上的这些交易信号而产生的。

我们点开 TBQuant 的官网的函数栏目。在左侧函数分类中选择 BAR 数据交易函数，可以看到 TBL 语言里面的交易函数有：

在当前代码我只使用 buy 和 sellshort 这两个指令就可以。

2、状态函数

作为四周规则策略的开多条件其实是最新价突破四周高点通道，就买入开多。而突破这个动作是一瞬间的，突破完之后，最新价可能会持续在四周高点通道之上。如果我们使用 High>=highline 作为开多条件，不加其它限制条件的化，只要突破完以后的 K 线的最新价在四周高点通道之上，那么仍将满足 High>=highline 的条件，那么就仍旧会不断加仓。这种情况不是我们的本意，我们只需要在突破的那一瞬间买入一次即可。所以对于后续的加仓我们要做出限制。我们使用 MarketPosition<>1 作为限制条件。而 MarketPosition 是持仓状态函数，只要我们已经开了多单，那么这个状态就是 1，这时候就不需要再次开仓了。
如果不加 MarketPosition<>1 的效果图如下：黄色的买入箭头不断再买入加仓。

而加上 MarketPosition<>1 的限制之后，就不会在后续的上涨中不断加仓。

这个状态函数就是要读取下目前策略的持仓状态。

我们在 BAR 数据策略函数里面可以找到一个 marketposition 的函数。

可以查看下它的说明：

MarketPosition

说明	获得当前持仓状态。
语法	Integer MarketPosition()
参数	无
备注	获得当前持仓状态，返回值为整型。 返回值定义如下： -1 当前位置为持空仓 0 当前位置为持平 1 当前位置为持多仓
示例	if(MarketPosition==1)判断当前是否持多仓 if(MarketPosition!=0)判断当前是否有持仓，无论持空仓或多仓

这样我们的开多条件就可以这样写：

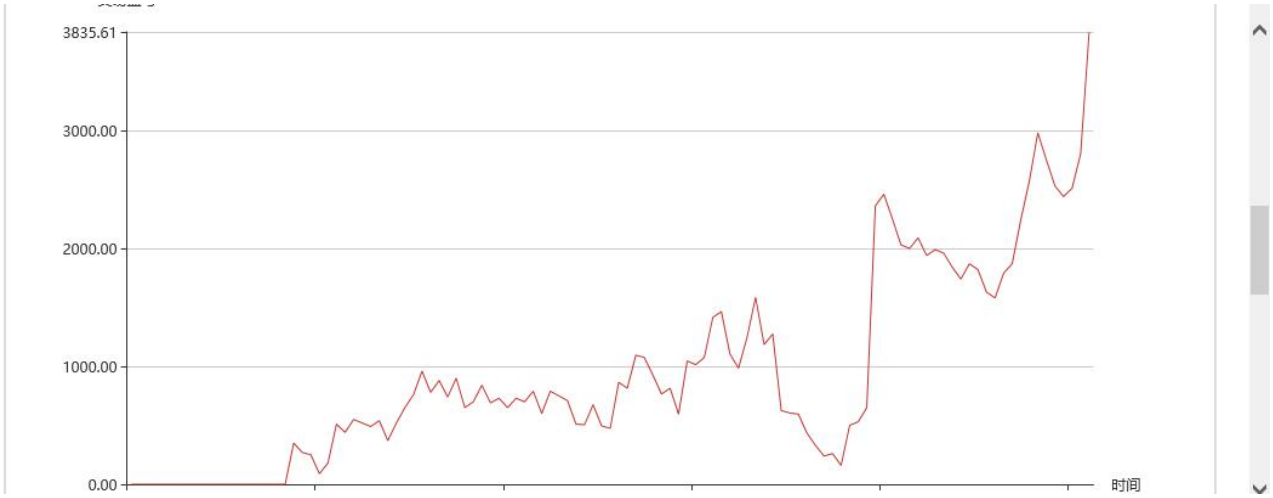
```
If(MarketPosition<>1 And High>=highline)
    Buy(1,Max(Open,highline));
```

3、条件语句

作为计算机语言，控制语句是必备的。TBL 语言同样支持顺序执行、条件语句和循环语句。
TBL 代码的执行顺序是从上到下，实现了顺序执行；
TBL 语言提供了 if else 的关键字实现了条件执行；
TBL 语言提供了 for while 关键字实现了循环执行。
更加详细的介绍可以查看第二篇的控制语句。
在这个代码中，我们使用 if 的条件语句即可。

4、初始资金、滑点、手续费

在四周规则的最简单的版本 FourWeek 中我们对整个代码的交易和测试环境并未做任何设定，比如账户的初始资金、滑点和手续费是多少。
而交易环境对于策略的测试结果影响很大。比如对于高频策略来说，它对交易成本也就是手续费和滑点极其敏感，稍微多一点交易成本，测试结果可能就会由盈转亏。而初始资金如果设置的太小，那么可能资金权益回撤时，不能开出相应仓位。
当四周用在 1mins 图表上，如果没有手续费和滑点，测试报告的累计盈亏曲线是一路新高。



如果设置手续费为 5%%，滑点为 2 跳，那么测试报告的累计盈亏曲线变为了一路亏损。

所以我们需要对这些交易和测试环境的变量做初始设置。

关于这些初始变量的设置都是在策略运行之前的，所以在一个 oninit 的作用域下进行设置。我们对原来的四周规则代码增加交易相关设置的代码，变为如下代码：

```
Vars
   Numeric highline;
   Numeric lowline;
Events
OnInit()
{
   //=========交易相关设置==============
   SetInitCapital(1000000);  //设置初始资金为 100 万
    SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续费率为成交金额的 5%%，不收平今。BitOr 是一个系统函数，当需要同时设置两项内容时，需要用 bitor 把两项内容括起来。
   SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
OnBar(ArrayRef<Integer> indexs)
{
   highline=Highest(High[1],20);
   lowline=Lowest(Low[1],20);
   If(MarketPosition<>1 And High>=highline)
      Buy(1,Max(Open,highline));
   If(MarketPosition<>-1 And Low<=lowline)
      SellShort(1,Min(Open,lowline));
}
```

在代码中我们设置了初始资金为 100 万，手续费为 5%%，不收平今，滑点是 2 跳/手。而 oninit 还有很多交易设置可以进行设置，这里我们不再一一介绍。读者可以在 TBQuant 中新建一个策略，查看下模板中的更多设置。

首先需要注意的三点：

1)、菜单和代码都可以设置这些内容，如果两者设置内容不一样，以哪一个设置为准？

答：以公式设置为准。

2)、这些设置可以对策略单元的不同数据源分开进行设置吗?还是只能所有数据源统一设置？

答：保证金、手续费、滑点和委托偏移委托映射，都可以在函数前面加 data[i]来实现只对 data[i]进行个性设置。忽略自动交易的相关函数目前还不能区分数据源。

3)、可以在哪些事件中设置这些内容？

答：保证金手续费滑点/委托偏移委托映射 /忽略自动，数据源不参与策略报告统计可以放在 Oninit、OnReady 中。

5、交易头寸的计算

在上面的代码中，我们虽然设置了初始资金为 100 万，但是交易手数，却还是使用固定的 1 手。但是不同品种的一手代表的市值可能差距很大，比如股指一手市值 100 多万，豆粕一手市值只有几万。为了不同品种方便对比，需要统一开仓市值。

资金管理其实是交易中重要一个环节，那么如何根据账户的资金来决定下单的手数呢，比如按照固定资金开仓怎么用代码实现呢？

我们直接看下面的代码如何实现固定资金使用比例/固定资金开仓的。

```
Params
  Numeric initcapital(100);  //单位：万
  Numeric moneyrate(10);     //资金使用比例：单位%
  Numeric money(100);        //固定市值开仓：单位万
Vars
  Numeric myprice;     //委托价格
  Numeric lots(1);     //下单手数
Events
OnInit()
{
  SetInitCapital(initcapital*10000);  //设定初始资金
  SetMarginRate(0.1);             //设定保证金比例
  SetBeginBarMaxCount(1);
}
OnBar(ArrayRef<Integer> indexs)
{
  if(MarketPosition<>1)
  {
    myprice=Open; //确定委托价格
    if(money<>0) //固定市值开仓
      lots=IntPart(money*10000/(myprice*contractunit*BigPointValue)); //计算开仓手数

    buy(lots,Open);
  }

}
```

计算开仓手数我们使用了这样一句代码：

```
lots=IntPart(money*10000/(myprice*contractunit*BigPointValue));
```

我们以一个具体的品种比如豆粕为例来解释下。豆粕的合约单位 contractunit 是一手 10 吨，每点价值 BigPointValue 是 1，开仓的委托价格是 myprice。那么一手豆粕对应的开仓市值就是 myprice*contractunit*BigPointValue。那么给出我们开仓的总市值除以这个 1 手的市值，就可以计算出开仓的手数。

我们使用固定资金开仓这种方法，修正四周策略如下：

每次开仓使用 10 万保证金。

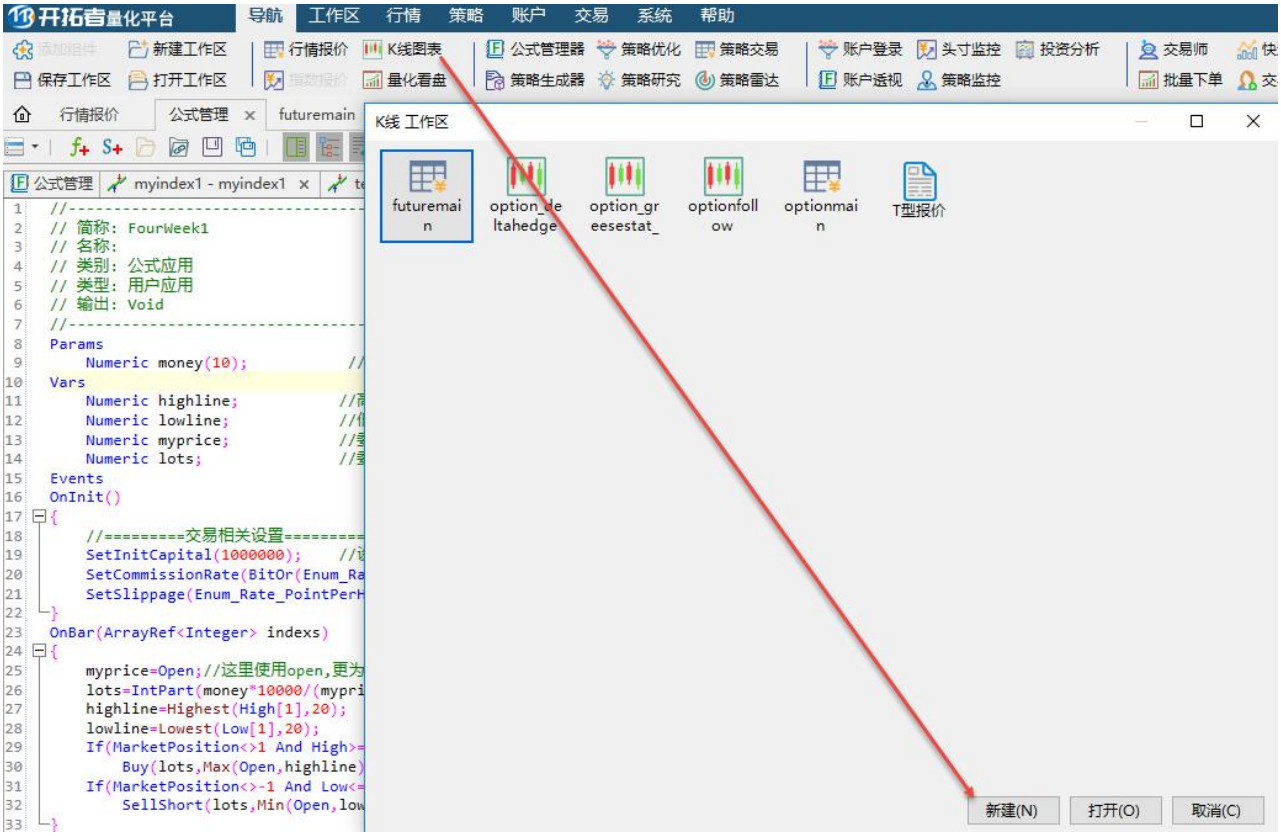
```
Params
  Numeric money(10);        //固定资金开仓：单位万
Vars
  Numeric highline;        //高点连线
  Numeric lowline;        //低点连线
  Numeric myprice;        //委托价格
  Numeric lots;          //委托数量
Events
OnInit()
{
  //=========交易相关设置==============
SetMarginRate(0.1);              //设定保证金比例
SetInitCapital(1000000);  //设置初始资金为 100 万
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续费率为成交金额的 5%%，不收平今， BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
OnBar(ArrayRef<Integer> indexs)
{
  myprice=Open;//这里使用 open,更为精确的是使用委托价格
  lots=IntPart(money*10000/(myprice*contractunit*BigPointValue*MarginRatio)); //计算开仓手数
  highline=Highest(High[1],20);
  lowline=Lowest(Low[1],20);
  If(MarketPosition<>1 And High>=highline)
    Buy(lots,Max(Open,highline));
  If(MarketPosition<>-1 And Low<=lowline)
    SellShort(lots,Min(Open,lowline));
}
```

6、K 线组件加载

我们用上面写好的四周策略来测试一下它在铜期货品种上面的表现。

1）首先我们要新建一个 K 线组件，打开 m9888(豆粕连续)

为了能够测试一个期货品种长期的表现，需要把历史的合约都串起来，形成指数合约，方便测试。在 TBQuant 里面有两种期货指数合约，一种是 000 结尾的，是所有月份合约的加权汇总。另外一种是 888 结尾的，这种是把历史上主力合约串起来。000 比 888 连续，但是 888 的价格都是主力合约的价格，所以测试更真实。



2)加载公式 FourWeek

3)查看历史交易信号

系统默认的 K 线是有 1000 根历史，我们看到 m9888 只是把历史上的主力合约给串了起来，并没有做特别处理。

我们可以看到对于非多即空的四周策略来说，很多持仓都在持仓过程中经历了换月。开平仓都不是一个合约。如果直接这样看测试结果，是不符合是真实交易的。真实交易需要在换月的时候换仓。所以这时候需要有一个换月除权的处理。

首先，我们需要平滑的连续合约，这样指标的计算也是平滑，不会突然跳空。

其次，换月的时候，我们还需要换仓。

再次，统计测试报告的时候，还需要按照真实合约的价格进行结算。

那么我们需要一个系统的方法来处理这些问题。

7、换月除权说明

历史数据回测是量化投资中重要的一环，历史数据回测对数据的连续性要求较高。但是，我们知道，股票有送配股、分红派息，期货合约更是有到期时间，股票除权除息后，价格往往发生了较大的变化，而期货单个合约的数据是较短的，这些都导致了历史数据的不连续。历史数据的不连续，使得策略回测的难度大幅增加。目前，大部分的股票软件都有对股票进行向前或向后除权，这种除权，作为目测观察没有问题，但却因为不是真实价格而无法进行历史测试，很多期货软件也提供了连续的合约指数，但是这种指数也因为不同月份合约之间存在着很大的差异性而无法用做历史测试。

TBQuant（开拓者量化平台）提供了处理股票除权、期货主力合约换月的方法，让量化投资者能极其方便的进行除权换月的数据处理。我们提供了两种方法，一种是基于真实合约数据进行后复权处理，第二种是系统直接提供后复权之后的数据。两种处理方式的底层逻辑是一致的，第二种使用更加方便。

关于除权换月的详细描述读者可以查看 TradeBlazer 官网帮助中心 TBQuant 软件使用的相应章节。在这里我们采用函数来处理这个问题。

我们在 oninit 中增加如下代码，来解决我们面临的一系列问题。

```
//=========除权换月相关设置==============
AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
```

复权有前复权和后复权两种方式。前复权会导致历史价格不断地变化，所以会引起交易信号的前后不一致。所以我们采用后复权的方式。后复权的具体算法在 TradeBlazer 官网的帮助中心 TBQuant 软件使用中可以查看相关章节。

```
AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
```

除权后数据源已经变为连续的后复权的数据了，但是实际交易的发单还是要以真实合约的价格来作为委托价格，所以就需要一个转换。这个转换用映射真实价格的函数来实现。

```
AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
```

在除权换月的时候，老的仓位要平掉新的仓位要建立。这样的一笔换仓操作用户不需要自己写代码处理，只需要调用自动换仓函数即可自动处理。

```
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
```
而对于测试报告来说，除权换月的这笔换仓操作并不是因为交易规则而触发的新的交易，而是因为除权换月而不得不做的换仓操作。所以除权换月前后的两笔分开的交易，应该放在一起统计会更合理。所以在测试报告的层面，我们将这两笔交易合二为一。这个就使用忽略换仓信号计算的函数来实现。

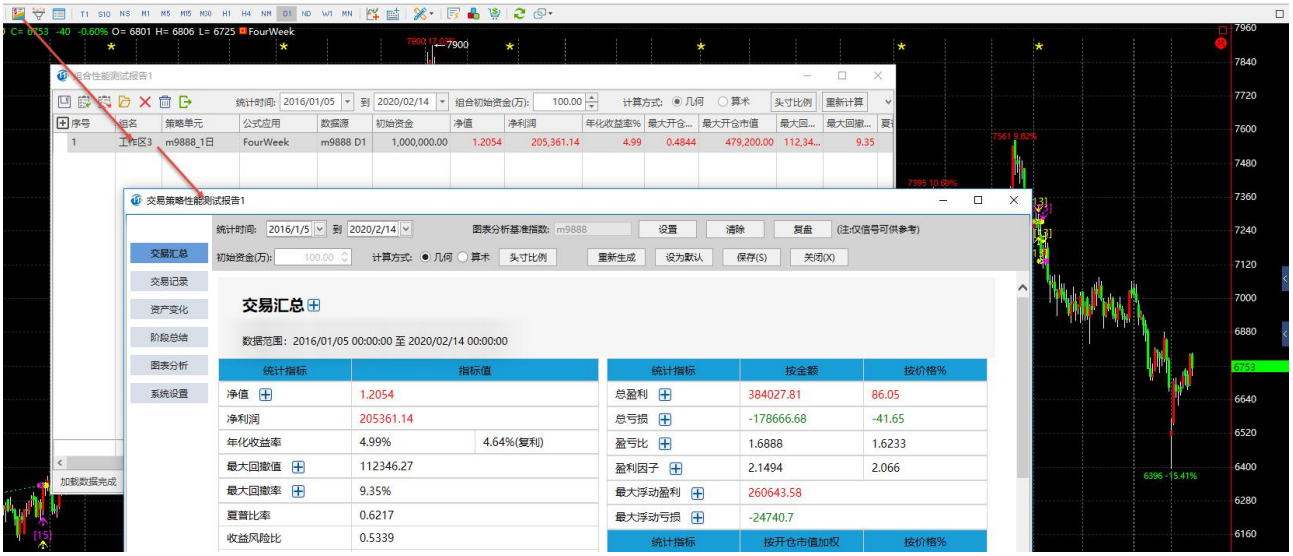
添加之后的四周策略的完整代码如下：

```
//-------------------------------------------------------------------------
// 简称: FourWeek
// 名称:
// 类别: 公式应用
// 类型: 用户应用
// 输出: Void
//-------------------------------------------------------------------------
Params
  Numeric money(10);        //固定资金开仓：单位万
Vars
  Numeric highline;        //高点连线
  Numeric lowline;        //低点连线
  Numeric myprice;        //委托价格
  Numeric lots;          //委托数量
Events
OnInit()
{
  //=========除权换月相关设置==============
  AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
  AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
  AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
  //=========交易相关设置==============
  SetInitCapital(1000000);  //设置初始资金为 100 万
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续费率为成交金额的 5%%，不收平今， BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
OnBar(ArrayRef<Integer> indexs)
{
  myprice=Open/rollover;//这里使用 open/rollover,更为精确的是使用委托价格.
  Rollvoer 是系统提供的复权系数，真实价格= 后复权的价格/rollover。
  lots=IntPart(money*10000/(myprice*contractunit*BigPointValue*MarginRatio)); //计算开仓手数
  highline=Highest(High[1],20);
  lowline=Lowest(Low[1],20);
  If(MarketPosition<>1 And High>=highline)
    Buy(lots,Max(Open,highline));
  If(MarketPosition<>-1 And Low<=lowline)
```

```
    SellShort(lots,Min(Open,lowline));
}

//-------------------------------------------------------------------
// 编译版本:  2020/02/14 112835
// 版权所有   tbyihao
// 更改声明   TradeBlazer Software 保留对 TradeBlazer 平台
//        每一版本的 TradeBlazer 公式修改和重写的权利
//-------------------------------------------------------------------
```

8、测试报告简要说明

处理除权换月的问题之后，我们就可以查看测试报告了。
我们在 K 线图中点击图标，可以弹出如下测试报告。



我们在下面简单介绍下测试报告，具体内容可以在 TradeBlazer 官网的帮助中心 TBQuant 软件使用的相应章节查看。
交易策略性能测试报告显示一个数据源在交易策略上的测试报告，或者显示多个策略单元的所有数据源加载各自的交易策略之后的组合性能测试报告。测试报告分析策略使用的资金成本，交易时间、可能承受的风险及期望的收益。

1）字段简介

统计时间：性能报告统计的时间范围。
初始资金：用于计算净值、算术最大开仓杠杆、动态权益。每个策略单元的初始资金和所有策略单元汇总后的初始资金都使用设置的初始资金。默认值与组合测试报告中的设置一致。
计算方式：计算方式可选择算术或几何。年化收益率、最大开仓杠杆、最大回撤率根据设置的计算方式来计算。计算方式的默认值与组合测试报告中的设置一致。
策略头寸比例：用于加权计算多策略单元组合测试的资金、盈亏、市值等信息。对于没有任何商品数据的策略单元，不参与权重计算。交易记录部分在开仓时计算权重。策略头寸比例的默认值与组合测试报告中的设置一致。
重新计算：根据最新设置重新计算并显示计算后的性能测试结果。

设为默认：可以把用户的初始资金设置、计算方式、策略头寸比例设置保存为默认。

2）详细报告

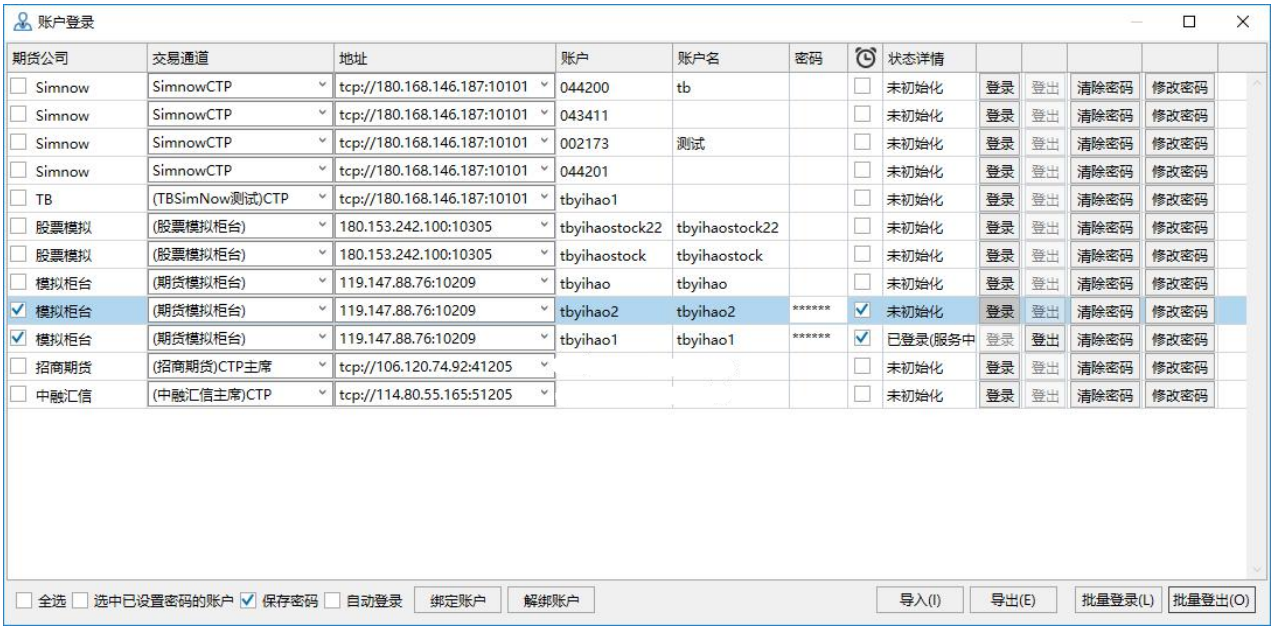
详细报告按照六个方面对交易策略或投资组合的报表进行分析，包括交易汇总、交易记录、资产变化、阶段总结、图表分析和系统设置。
详细报告的六个方面简介：
交易汇总：按照多头交易、空头交易和全部交易列出当前交易策略的交易统计信息。
交易记录：按每个公式应用在每个数据源上的开仓平仓对所有交易进行配对组合，并按策略头寸进行加权计算盈亏及累计盈亏。可以按策略单元筛选交易记录，也可以按公式筛选交易记录。
资产变化：显示资产按策略头寸进行加权后的变化记录及统计信息。
阶段总结：按年、月统计交易净利润、最大回撤、最大开仓市值及手续费。
图表分析：以曲线的形式显示当前测试策略单元的交易盈亏变化情况。图表可以放大缩小，可以进行区间统计。图表下方显示性能测试汇总、相对高低点、最大回撤、周期统计信息。
系统设置：显示交易策略的参数，设置以及数据等内容。

9、账户登录、头寸控制

从测试报告看，这个策略还是可以盈利的，所以下一步是如何进行交易。

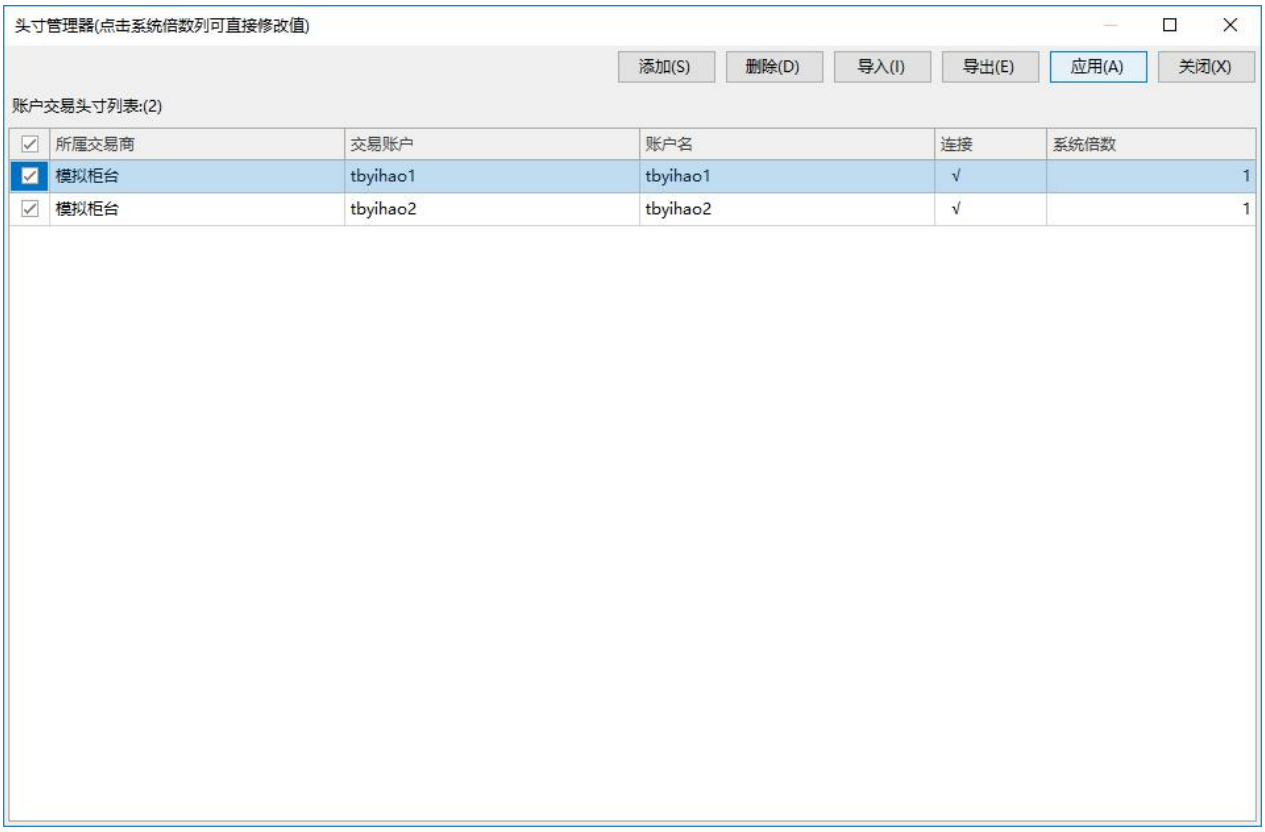
1）登录交易账户

在导航的图标栏中，选择账户登录。打开如下的账户登录界面，选择账户进行登录。



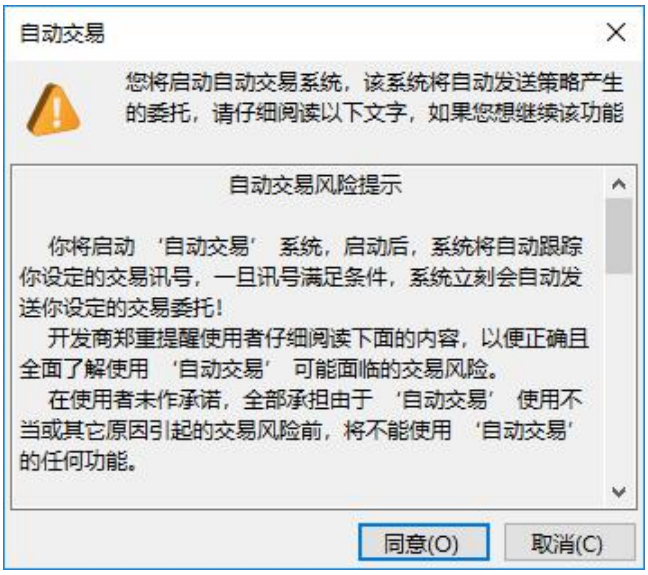
2）设置头寸

对着 K 线图右键，选择头寸管理器，打开如下界面。勾选相应的交易账户，点击应用。然后关闭头寸管理器的窗口。



3）在 K 线图的工具栏，选择启动自动交易的按钮，启动自动交易。

在弹出的风险提示界面，点击同意。



4）成功启动自动交易后，K 线图右上角的头像将会变成绿色笑脸。

10、运行机制

对于专业的读者，在这里必须要了解下 TBL 的代码的运行机制了。TBL 语言的代码驱动机制从原来的面向过程升级到了面向对象，现在的机制叫做事件驱动机制。

1）事件驱动机制

事件驱动基本类型有 12 种，这 12 种对应的基本特征如下表所示：

事件名称	调用语法	基本解释

初始化事件
(OnInit)
	OnInit()
	{
	//用户初始化逻辑，主要用于创建各种数据源
	}
		由策略执行前的初始化事件，只运行一次 只能依赖或使用 Global 等全局变量、全局设置信息 可以订阅数据，数据准备等操作

后初始化事件
(OnReady)
	OnReady()
	{
	//在所有的数据源准备完成后调用，应用在数据源的设置等操作
	}
		在策略执行前的初始化事件（数据源数据准确完成）之后，只运行一次 只能依赖或使用 Global 等全局变量、全局设置信息 可以对各个数据源进行相关设置

Bar 数据更新驱动
(OnBar)
	OnBar(Arrayref<Integer>indexs)
	{
	//用户业务逻辑
	}
		当 Bar 更新变化时驱动，参数 indexs 表示更新的图层编号数组

Bar 数据开始驱动
(OnBaropen)
	OnBaropen(ArrayRef<Integer> indexs)
	{
	//用户业务逻辑
	}
		当 Bar 第一次生成时驱动，参数 indexs 表示更新的图层编号数组

Bar 数据结束驱动
(OnBarclose)
	OnBarclose(ArrayRef<Integer> indexs)
	{
	//用户业务逻辑
	}
		当下一个 Bar 开始之前，最后一次当前 bar 驱动，参数 indexs 表示更新的图层编号数组

持仓更新驱动
(OnPosition)
	OnPosition(PositionRef pos) {
	//用户业务逻辑
	}
		当持仓更新时驱动，参数 pos 表示更新的持仓结构体

委托更新驱动(
OnOrder)
	OnOrder(OrderRef ord)
	{
	//用户业务逻辑
	}
		当委托更新时驱动，参数 ord 表示更新的委托结构体

成交更新驱动
(OnFill)
	OnFill(FillRef ordFill)
	{
	//用户业务逻辑
	}
		当成交更新时驱动，参数 ordFill 表示更新的成交结构体

定时器更新驱动
(OnTimer)
	OnTimer(Integer id,Integer millsecs)
	{
	//用户业务逻辑
	}
		当定时器更新时驱动，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值

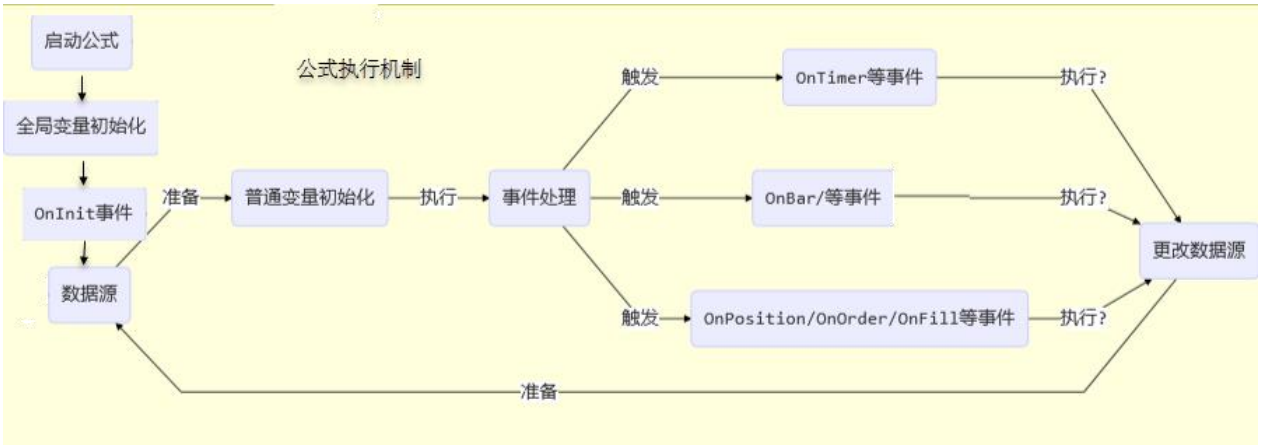
策略账户持仓更新事件 (OnStrategyPosition)	OnStrategyPosition(PositionRef pos) { //用户业务逻辑 }	当策略账户仓更新驱动，策略账户仓是指本策略当日成交累计量，是一个相对持仓，参数 pos 表示持仓结构体
通用事件 (OnEvent)	OnEvent(StringRef name,MapRef<String,String> evtValue) { //用户业务逻辑 }	当订阅了通用事件，有事件是驱动，参数 name 表示事件名称，evtValue 表示事件内容
退出事件 (OnExit)	OnExit() { //用户业务逻辑}	当策略退出时驱动

事件驱动的整个结构运行的顺序是：

1、全局变量初始化。

2、oninit 事件触发。在 oninit 事件中一般完成数据源的订阅、定时器的设置等操作；

3、数据源的准备。
4、普通变量初始化。
5、其它事件 ontimer/onbar/onposition 等的触发。
6、是否更改数据源。



2）onbar 机制

虽然我们已经讲解了代码的执行顺序是从上到下，但是公式加载到 K 线图上的时候它是怎么运行的呢？这时候需要了解下我们的 onbar 机制。因为我们代码中使用了 onbar 的事件驱动。
Onbar 机制，就是之前产品的 begin/end 机制。当策略公式在 onbar 机制下运行时，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、最高价、最低价、收盘价、成交量、持仓量以及时间等信息数据。Bar 数据也是我们口头上常说的 K 线数据。

2.1）历史数据的运行机制：从左到右、从上到下

公式在计算时按照“Bar 数据”的 Bar 数目，从左边第一个 Bar 依次执行到右边最后一个 Bar，在单个 Bar 数据上的公式运算为从上到下完整执行公式中所有语句，即每次公式的运算都是从公式最上方的语句“参数的声明、变量的声明”开始直至公式的计算主体 onbar{}(Begin 至 End)结束。

如图所示，我们定义了一个全局变量，在第一根 BAR 初始值为 0，然后其它 BAR 每次加 1。
TBQuant 的 onbar 机制从左边第一根 BAR 开始运行公式一次，公式代码的运行则从上到下依次执行。然后第二根 BAR 运行公式 num 变为 1,然后第三根 BAR 运行公式，直到倒数第二根 BAR（第 14 根 BAR）运行结束，num 累计为 13。

2.2）实时行情的运行机制：每个 TICK 运行一次

对于实时数据，每当有新的 Tick 进来，公式都会在当前 Bar 上对新数据执行一次完整的运算。比如如果上图中的公式在 2020/1/6 的交易时间运行，那么每来一个新的 TICK，公式会从上到下依次执行，num 会累加 1。盘中在 2020/1/6 这根 bar 上 num 的数值会不断累计变化。如果当天有 100 个 TICK，则 num 在 2020/1/6 这根 BAR 结束时会总共累加 100 次。最后 num 的数值会变成 113。
注意：若当前公式所应用的合约交易非常活跃并且公式程序较长、计算较复杂时，当前 Tick 到来之后与下一个 Tick 到来之前的这段时间之内，可能无法完成公式代码完整执行一遍的计算。 此时，虽然新的 Tick 到来，但是不会触发公式的重新运行，依然继续执行之前的计算直至代码的最后一行。 之后，当最新的 Tick 到来时，才会再次触发新一轮的公式运算。也就是说，在这种情况下，不是 Bar 中每一个 Tick 到来时都触发公式重新计算一次。

下表列出了历史回测和实时交易的区别	历史回测	实时交易
Bar 数据	确定不变	实时更新
公式运行	每根 Bar 一次	每个 Tick 一次
交易信号	固定不变	有可能变化
是否发单	否	是（受公式机制控制）
函数调用	部分函数无效	有效

注意：
实时交易的当前 BAR 是每一个 tick 运行一次，图标上的交易信号会随之改变，实际交易也会随之执行。但是当前 BAR（bar0 标注）走完，新的一根 BAR(bar1 标注)出现时，刚刚的 bar0 就成为了历史。一旦成为历史的 BAR，那么它的运行机制将走的是另一套机制，即历史回测机制。两种机制的不同将可能导致 bar0 上的信号不一致，这样可能导致信号闪烁。

2.3、四周策略的升级——海龟交易系统

资金管理是交易中最重要的一环，甚至比策略本身的交易规则都重要。资金管理对策略的净利润、盈亏比、收益风险比有着重要影响。下面我们简单介绍一个经典交易系统海龟交易系统。海龟交易是在四周规则的基础上加以改进而生成的。它的重要改进就集中在仓位如何管理，包括头寸怎么计算，怎么进行加仓。
我们先贴出完整的代码，然后重点讲解下海龟的头寸怎么计算。

```
//-----------------------------------------------------------------------
// 简称: TurtleTrader
// 名称: 海龟交易系统
// 类别: 公式应用
// 类型: 内建应用
//-----------------------------------------------------------------------
Params
   Numeric nEntries(3);            // 最大建仓次数
   Numeric RiskRatio(1);           // % Risk Per N ( 0 - 100)
   Numeric ATRLength(20);          // 平均波动周期 ATR Length
   Numeric boLength(20);           // 短周期 BreakOut Length
```

```
  Numeric fsLength(55);           // 长周期 FailSafe Length
  Numeric teLength(10);           // 离市周期 Trailing Exit Length
  Bool LastProfitableTradeFilter(True);  // 使用入市过滤条件
Vars
  Numeric MinPoint;             // 最小变动单位
  Series<Numeric> AvgTR;           // ATR
  Numeric N;                  // N 值
   Numeric TotalEquity;           // 按最新收盘价计算出的总资产
  Numeric TurtleUnits;           // 交易单位
  Series<Numeric> DonchianHi;         // 唐奇安通道上轨，延后 1 个 Bar
  Series<Numeric> DonchianLo;         // 唐奇安通道下轨，延后 1 个 Bar
  Series<Numeric> fsDonchianHi;        // 唐奇安通道上轨，延后 1 个 Bar，长周期
  Series<Numeric> fsDonchianLo;        // 唐奇安通道下轨，延后 1 个 Bar，长周期
  Numeric ExitHighestPrice;         // 离市时判断需要的 N 周期最高价
  Numeric ExitLowestPrice;          // 离市时判断需要的 N 周期最低价
  Numeric myEntryPrice;           // 开仓价格
  Numeric myExitPrice;            // 平仓价格
  Bool SendOrderThisBar(False);       // 当前 Bar 有过交易
  Series<Numeric> preEntryPrice(0);      // 前一次开仓的价格
  Series<Bool> PreBreakoutFailure(false);  // 前一次突破是否失败
Events
  OnBar(ArrayRef<Integer> indexs)
  {

    If(BarStatus == 0)
    {
      preEntryPrice = InvalidNumeric;
      PreBreakoutFailure = false;
    }
    MinPoint = MinMove*PriceScale;
    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
    DonchianHi = HighestFC(High[1],boLength);
    DonchianLo = LowestFC(Low[1],boLength);
    fsDonchianHi = HighestFC(High[1],fsLength);
    fsDonchianLo = LowestFC(Low[1],fsLength);
    ExitLowestPrice = LowestFC(Low[1],teLength);
    ExitHighestPrice = HighestFC(High[1],teLength);
    Commentary("N="+Text(N));
    Commentary("preEntryPrice="+Text(preEntryPrice));
    Commentary("PreBreakoutFailure="+IIFString(PreBreakoutFailure,"True","False"));
```

```
    // 当不使用过滤条件，或者使用过滤条件并且条件为 PreBreakoutFailure 为 True 进行后续操作
    If(MarketPosition == 0 && ((!LastProfitableTradeFilter) Or (PreBreakoutFailure)))
    {
      // 突破开仓
      If(High > DonchianHi && TurtleUnits >= 1)
      {
        // 开仓价格取突破上轨+一个价位和最高价之间的较小值，这样能更接近真实情况，并能尽量保证成交
        myEntryPrice = min(high,DonchianHi + MinPoint);
        myEntryPrice = IIF(myEntryPrice < Open, Open,myEntryPrice); // 大跳空的时候用开盘价代替
        preEntryPrice = myEntryPrice;
        Buy(TurtleUnits,myEntryPrice);
        SendOrderThisBar = True;
        PreBreakoutFailure = False;
      }
      If(Low < DonchianLo && TurtleUnits >= 1)
      {
        // 开仓价格取突破下轨-一个价位和最低价之间的较大值，这样能更接近真实情况，并能尽量保证成交
        myEntryPrice = max(low,DonchianLo - MinPoint);
        myEntryPrice = IIF(myEntryPrice > Open, Open,myEntryPrice); // 大跳空的时候用开盘价代替
        preEntryPrice = myEntryPrice;
        SendOrderThisBar = True;
        SellShort(TurtleUnits,myEntryPrice);
        SendOrderThisBar = True;
        PreBreakoutFailure = False;
      }
    }
    // 长周期突破开仓 Failsafe Breakout point
    If(MarketPosition == 0)
    {
      Commentary("fsDonchianHi="+Text(fsDonchianHi));
      If(High > fsDonchianHi && TurtleUnits >= 1)
      {
        // 开仓价格取突破上轨+一个价位和最高价之间的较小值，这样能更接近真实情况，并能尽量保证成交
        myEntryPrice = min(high,fsDonchianHi + MinPoint);
        myEntryPrice = IIF(myEntryPrice < Open, Open,myEntryPrice); // 大跳空的时候用开盘价代替
        preEntryPrice = myEntryPrice;
        Buy(TurtleUnits,myEntryPrice);
        SendOrderThisBar = True;
```

```
            PreBreakoutFailure = False;
        }
        Commentary("fsDonchianLo="+Text(fsDonchianLo));
        If(Low < fsDonchianLo && TurtleUnits >= 1)
        {
            // 开仓价格取突破下轨--一个价位和最低价之间的较大值，这样能更接近真实情况，并能尽量保证成交
            myEntryPrice = max(low,fsDonchianLo - MinPoint);
            myEntryPrice = IIF(myEntryPrice > Open, Open,myEntryPrice); // 大跳空的时候用开盘价代替
            preEntryPrice = myEntryPrice;
            SellShort(TurtleUnits,myEntryPrice);
            SendOrderThisBar = True;
            PreBreakoutFailure = False;
        }
    }
    If(MarketPosition == 1) // 有多仓的情况
    {
        Commentary("ExitLowestPrice="+Text(ExitLowestPrice));
        If(Low < ExitLowestPrice)
        {
            myExitPrice = max(Low,ExitLowestPrice - MinPoint);
            myExitPrice = IIF(myExitPrice > Open, Open,myExitPrice); // 大跳空的时候用开盘价代替
            Sell(0,myExitPrice);  // 数量用 0 的情况下将全部平仓
        }Else
        {
            If(preEntryPrice!=InvalidNumeric && TurtleUnits >= 1)
            {
                If(Open >= preEntryPrice + 0.5*N && CurrentEntries < nEntries) // 如果开盘就超过设定的 1/2N,则直接用开盘价增仓。
                {
                    myEntryPrice = Open;
                    preEntryPrice = myEntryPrice;
                    Buy(TurtleUnits,myEntryPrice);
                    SendOrderThisBar = True;
                }
                while(High >= preEntryPrice + 0.5*N && CurrentEntries < nEntries) // 以最高价为标准，判断能进行几次增仓
                {
                    myEntryPrice = preEntryPrice + 0.5 * N;
                    preEntryPrice = myEntryPrice;
                    Buy(TurtleUnits,myEntryPrice);
                    SendOrderThisBar = True;
                }
```

```
        }
        // 止损指令
        If(Low <= preEntryPrice - 2 * N && SendOrderThisBar == false) // 加仓 Bar 不止损
        {
          myExitPrice = preEntryPrice - 2 * N;
          myExitPrice = IIF(myExitPrice > Open, Open,myExitPrice); // 大跳空的时候用开盘价代
替
          Sell(0,myExitPrice); // 数量用 0 的情况下将全部平仓
          PreBreakoutFailure = True;
        }
      }
    }Else If(MarketPosition ==-1) // 有空仓的情况
    {
      // 求出持空仓时离市的条件比较值
      Commentary("ExitHighestPrice="+Text(ExitHighestPrice));
      If(High > ExitHighestPrice)
      {
        myExitPrice = Min(High,ExitHighestPrice + MinPoint);
        myExitPrice = IIF(myExitPrice < Open, Open,myExitPrice); // 大跳空的时候用开盘价代替
        BuyToCover(0,myExitPrice);  // 数量用 0 的情况下将全部平仓
      }Else
      {
        If(preEntryPrice!=InvalidNumeric && TurtleUnits >= 1)
        {
          If(Open <= preEntryPrice - 0.5*N && CurrentEntries < nEntries) // 如果开盘就超过设
定的 1/2N,则直接用开盘价增仓。
          {
            myEntryPrice = Open;
            preEntryPrice = myEntryPrice;
            SellShort(TurtleUnits,myEntryPrice);
            SendOrderThisBar = True;
          }
          while(Low <= preEntryPrice - 0.5*N && CurrentEntries < nEntries) // 以最低价为标准,
判断能进行几次增仓
          {
            myEntryPrice = preEntryPrice - 0.5 * N;
            preEntryPrice = myEntryPrice;
            SellShort(TurtleUnits,myEntryPrice);
            SendOrderThisBar = True;
          }
        }
        // 止损指令
        If(High >= preEntryPrice + 2 * N &&SendOrderThisBar==false) // 加仓 Bar 不止损
        {
```

```
            myExitPrice = preEntryPrice + 2 * N;
            myExitPrice = IIF(myExitPrice < Open, Open,myExitPrice); // 大跳空的时候用开盘价代
替
            BuyToCover(0,myExitPrice); // 数量用 0 的情况下将全部平仓
            PreBreakoutFailure = True;
          }
        }
      }
      Commentary("CurrentEntries = " + Text(CurrentEntries));
    }
//-------------------------------------------------------------------------
// 编译版本 GS2010.12.08
// 版权所有 TradeBlazer Software 2003－2010
// 更改声明 TradeBlazer Software 保留对 TradeBlazer 平
//       台每一版本的 TradeBlazer 公式修改和重写的权利
//-------------------------------------------------------------------------
```

1、头寸的计算

海龟交易的头寸计算是通过以下几行代码计算的。

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

海龟交易允许加仓，不管是底仓还是加仓都是使用这个 turtleunits 来作为交易数量。
这个 turtleunits 的计算体现着海龟策略的风险管理。
首先它要评估商品的波动性，所以它计算了 atr 平均价格波动幅度。
然后它设定了总资金权益的风险比率 riskratio,然后根据 atr 倒推交易数量。

2、K 线组件加载

我们加载海龟策略到豆粕连续 m9888。
为了对比我们在四周策略的 K 线图工作区的下方加载新的 K 线组件。
1）添加组件

2）拖动组件到当前 K 线图的下方。

3）调整商品和公式。

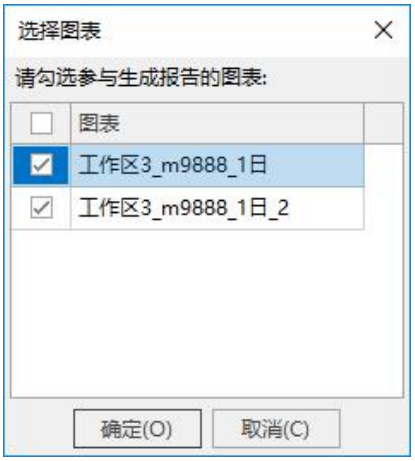
为了方便两个公式进行对比，我们要把初始资金等初始设置设置完全一样，所以需要把四周策略的 oninit 的设置完全 copy 到 turtletrader 的策略上来。所以新的 myturtletrader 需要增加如下代码，重新加载策

略。

```
OnInit()
{
  //=========除权换月相关设置==============
  AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
  AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
  AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
  //=========交易相关设置==============
  SetInitCapital(1000000);  //设置初始资金为 100 万
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续费率为成交金额的 5%%，不收平今，  BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
```

3、测试报告结果

我们点击测试报告按钮，勾选两个图表方便对比。



对比的测试报告如下。

4、如何评估和选择策略

从上面的两个测试报告看，四周的净值是 1.2042，海龟的是 0.9723。最大回撤四周是 9.35，海龟是 29.91。那么是否说海龟策略就不如四周规则呢？

其实我们还不能仅根据这一个商品的对比就下这个结论。

评估策略之前需要问自己几个问题：

1）策略的盈亏周期有多长？

正如经济周期一样，如果样本覆盖的周期还不到一个盈亏周期，那么样本很可能是片面的。而也和经济周期一样，一个策略的盈亏周期也很难界定。
2）测试报告有多少个品种？
这一点更难选择。有些策略是针对特定商品开发的，有些策略是可以适用于多种商品的，甚者多个市场的。我们只能比较那些可应用于同样商品的策略。像上面的对比，我们仅用了豆粕一个品种，是不能下结论的。
3）策略的频率是怎么样的？
就算两个策略交易的同样的商品或者商品组合，但是一个是高频策略，一个是长周期持仓策略。这两个策略还是不能比较优劣的。成功的高频策略明显具备更高的收益风险比，但是也具备容量小的缺陷。
4）账户的资金、手续费、滑点、使用杠杆等等是否一样？
这一点正如四周与海龟的对比，虽然我们统一了账户的初始资金、手续费、滑点。
但是每次交易的使用杠杆不一样，四周我们是按照固定资金开仓的，而海龟每次开仓都是风险倒推，开仓资金不稳定，并且还有加仓。
然后在上面的几个问题都有思考和调整之后，那么如何对比策略的优劣的。我们需要看测试报告的一些字段。
5）主要关注的绩效字段。
首先是净值和净利润，是否盈利。
然后是收益风险比或者夏普比率，这两个指标能部分解决使用杠杆不一样，策略频率不一样的问题。
其次是最大回撤，这关系到用户自身的风险承受，不过也可以调整初始资金来调整这个风险承受。
最后其它的一些指标，比如胜率和盈亏比可以用来评估策略的类别。

2.4、多品种投资组合

在评估策略时，我们的第二个问题是测试报告是有多少个品种的组合。对于那些应用于多个商品的策略，我们需要从多品种组合的角度去评估它。如何方便的做多品种的投资组合呢？实战中又是如何的管理多策略多品种的投资组合呢？

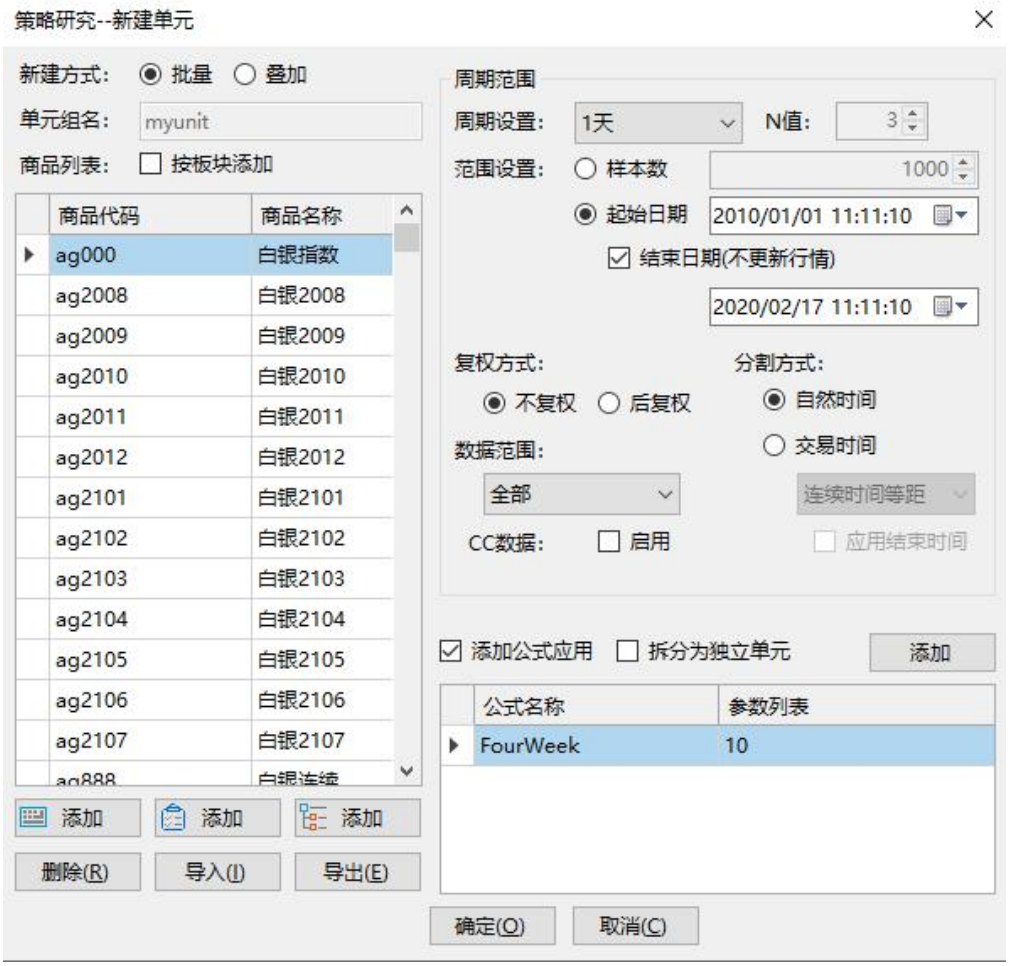
1、策略单元的概念

策略单元是 TBQuant 里面商品和公式组合的最小单位。
在一个策略单元内部，如果加载多个公式，多个公式互相干扰。
如果加载多个商品，多个商品以叠加状态存在。
K 线是一个策略单元，但是更为方便的策略单元是在策略研究和策略交易中设置。
策略研究专门用于历史测试，策略交易专门用于实盘交易。

2、策略研究

策略研究是专门用于历史回测的策略单元的集合。它在参数优化方面有强大的功能。
策略代码编写完毕，如何评判该策略是否可行呢？通过对历史数据加载策略，进行参数优化和回测，查看优化和测试结果，用户可以选择最优的参数。策略研究功能可以使用交易策略和商品数据创建策略单元，并对策略单元进行深入测试、策略优化、策略调试。策略研究完毕的策略单元可以导出到策略交易中交易，也可以导出到策略优化中进行批量优化。
关于策略研究的详细介绍可以在 TradeBlazer 官网的帮助中心\ TBQuant 软件使用的相应章节查看。
我们需要评估下四周策略在多商品组合的表现，所以需要新建一个策略研究， 加载历史数据较长成交比较活跃的商品的连续合约。
多品种组合有两种方式，一种是数据源的批量，另外一种是数据源的叠加。

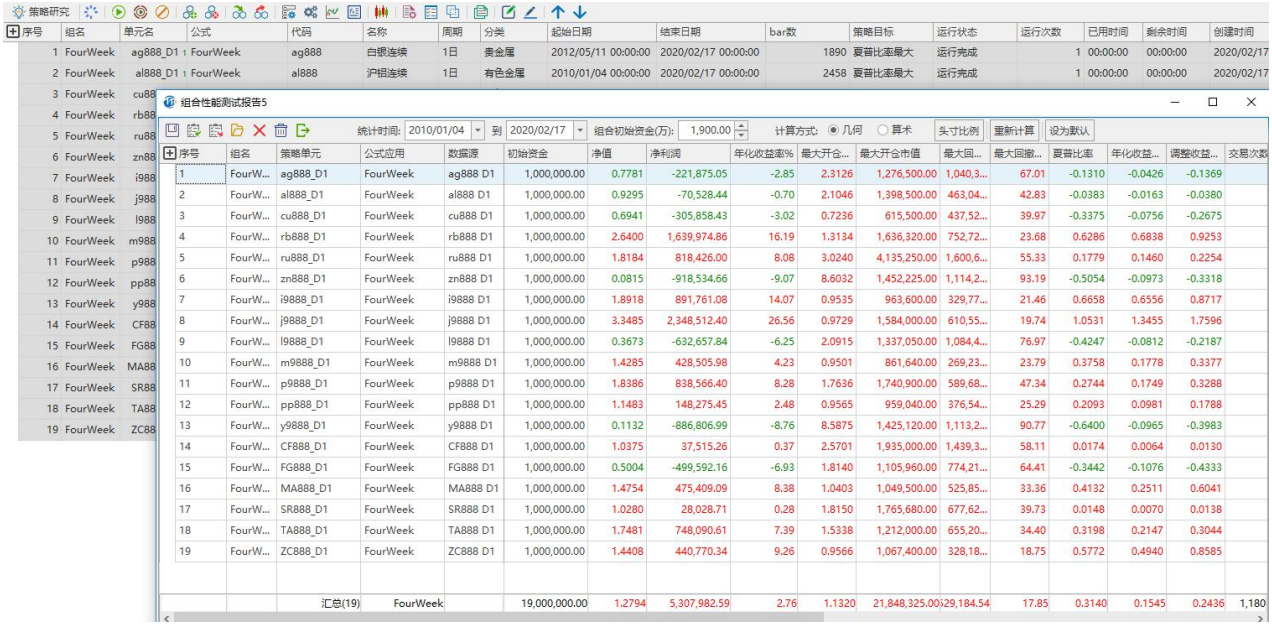
1）批量方式

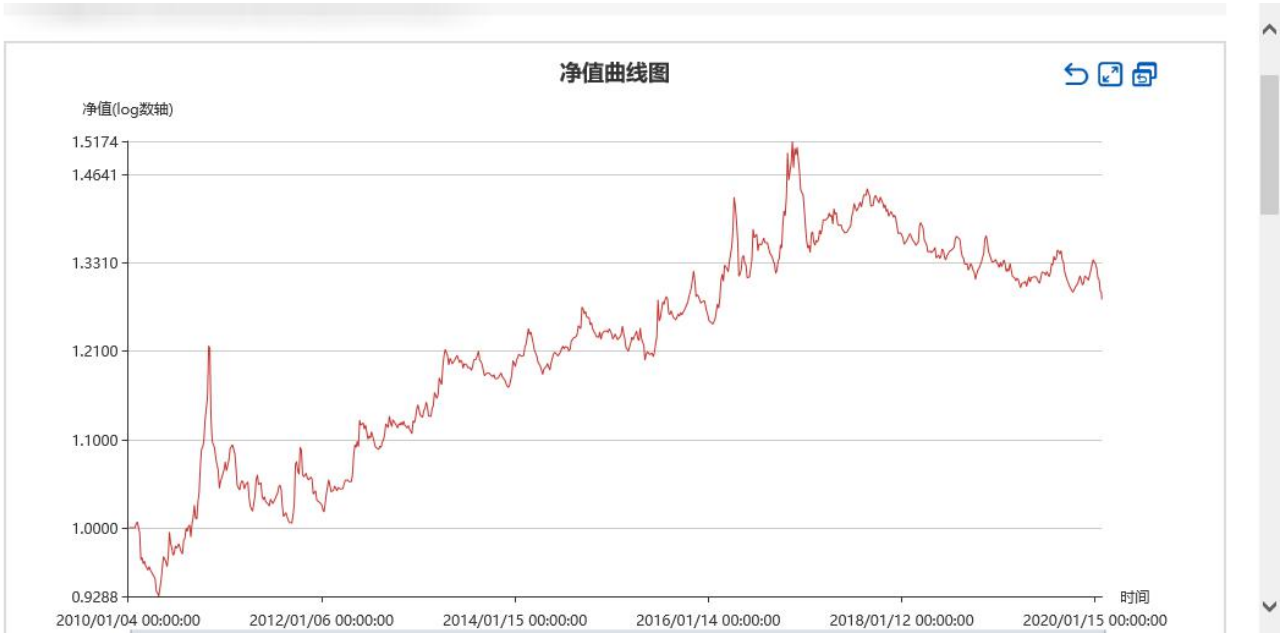


我们点击运行，然后选中这十几个策略单元，查看组合测试报告。

我们可以看到长达 10 年，这个策略整体是盈利的，虽然收益风险比和夏普比率都不是很高。

并且四周策略在 2010 年到 2017 年之前净值曲线都是屡创新高的。



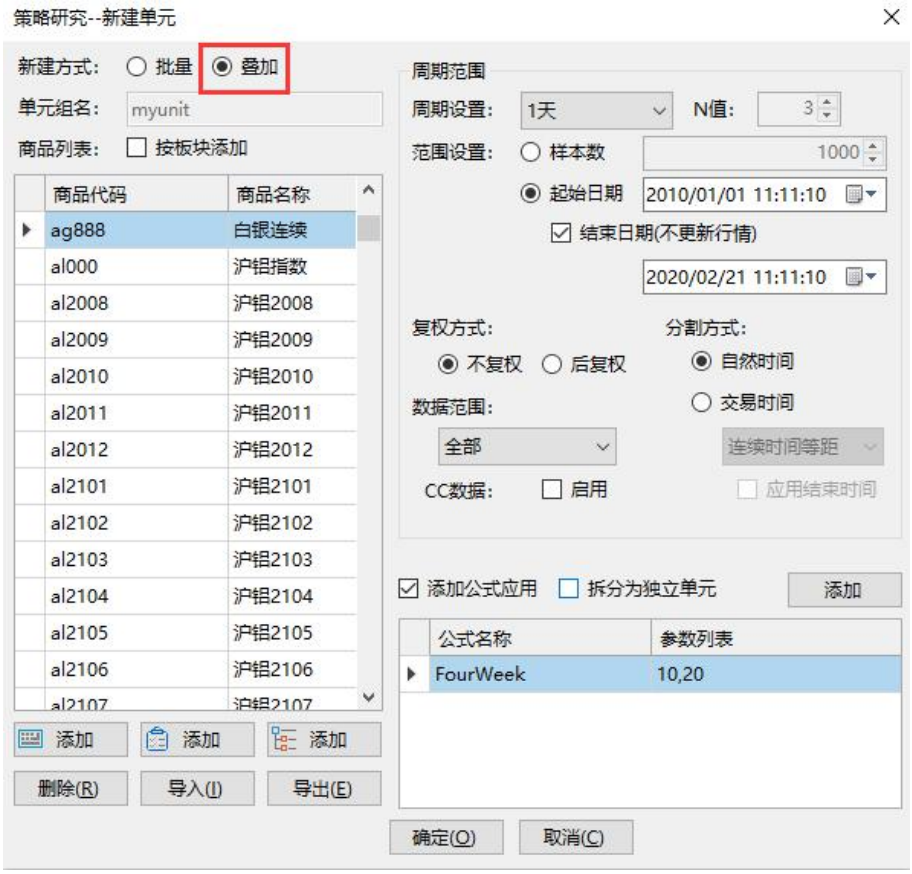


2)叠加方式

多数据源的叠加涉及到数据源的对齐，代码中对数据源的调用，这些细节问题用户可以在【四周规则策略改进】-【跟踪止盈代码写法的缺陷】-【 4、多数据源的 onbar 机制 】中详细查看。这里我们先看一个简单的使用案例。
叠加方式需要我们先修改下四周的公式为：
在 onbar 的运行代码前面加 Range[0:DataCount-1]，这样会对所有数据源都运行交易代码。

```
OnBar(ArrayRef<Integer> indexs)
{
  Range[0:DataCount-1]
  {
    myprice=Open;//这里使用 open,更为精确的是使用委托价格
    lots=IntPart(money*10000/(myprice*contractunit*BigPointValue)); //计算开仓手数
    highline=Highest(High[1],length);
    lowline=Lowest(Low[1],length);
    If(MarketPosition<>1 And High>=highline)
      Buy(lots,Max(Open,highline));
    If(MarketPosition<>-1 And Low<=lowline)
      SellShort(lots,Min(Open,lowline));
  }
}
```

然后我们新建策略研究如下操作：



策略研究展开如下：

序号	组名	单元名	公式	代码	名称	周期	分类	起始日期	结束日期	bar数	策略目标	运行状态	运行次数	已用时间	剩余时间
1	FourWeek	ag888_D1	FourWeek	ag888	白银连续	1日	贵金属	2012/05/11 00:00:00	2020/02/21 00:00:00	1894	夏普比率最大	运行完成	1	00:00:00	00:00:00
				al888	沪铝连续	1日	有色金属	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				cu888	沪铜连续	1日	有色金属	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				rb888	螺纹钢连续	1日	黑色相关	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				ru888	橡胶连续	1日	软商品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				zn888	沪锌连续	1日	有色金属	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				i9888	铁矿石连续	1日	黑色相关	2013/10/18 00:00:00	2020/02/21 00:00:00	1548					
				j9888	焦炭连续	1日	黑色相关	2011/04/18 00:00:00	2020/02/21 00:00:00	2153					
				l9888	塑料连续	1日	能源化工	2010/01/04 00:00:00	2020/02/21 00:00:00	2460					
				m9888	豆粕连续	1日	农产品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				p9888	棕榈油连续	1日	农产品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				pp888	聚丙烯连续	1日	能源化工	2014/02/28 00:00:00	2020/02/21 00:00:00	1457					
				y9888	豆油连续	1日	农产品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				CF888	棉花连续	1日	软商品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				FG888	玻璃连续	1日	软商品	2012/12/04 00:00:00	2020/02/21 00:00:00	1753					
				MA888	甲醇连续	1日	能源化工	2014/06/18 00:00:00	2020/02/21 00:00:00	1385					
				SR888	白糖连续	1日	软商品	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				TA888	PTA连续	1日	能源化工	2010/01/04 00:00:00	2020/02/21 00:00:00	2462					
				ZC888	动力煤连续	1日	黑色相关	2015/05/18 00:00:00	2020/02/21 00:00:00	1161					

我们看到这么多品种起始日期并不一致，为了解决有些数据源起始很早却不出信号的错误，我们还需要在 oninit 里面做一些处理。

```
//数据源叠加处理
   SetBeginBarMaxCount(10);   //设置最大起始 bar 数为 10
```

完整代码如下：

```
//---------------------------------------------------------------------
// 简称: FourWeek
// 名称:
// 类别: 公式应用
// 类型: 用户应用
// 输出: Void
```

```
//-------------------------------------------------------------------
Params
  Numeric money(10);        //固定资金开仓：单位万
  Numeric length(20,10,60,2);   //四周周期参数
Vars
  Numeric highline;        //高点连线
  Numeric lowline;        //低点连线
  Numeric myprice;        //委托价格
  Numeric lots;          //委托数量
Events
OnInit()
{
Range[0:DataCount-1]
  {
 //=========除权换月相关设置==============
  AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
  AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
  AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续
费率为成交金额的 5%%，不收平今， BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
  //=========交易相关设置==============
  SetInitCapital(1000000);  //设置初始资金为 100 万

  //数据源叠加处理
  SetBeginBarMaxCount(10);  //设置最大起始 bar 数为 10
}

OnBar(ArrayRef<Integer> indexs)
{
  Range[0:DataCount-1]
  {
    myprice=Open;//这里使用 open,更为精确的是使用委托价格
    lots=IntPart(money*10000/(myprice*contractunit*BigPointValue)); //计算开仓手数
    highline=Highest(High[1],length);
    lowline=Lowest(Low[1],length);
    If(MarketPosition<>1 And High>=highline)
      Buy(lots,Max(Open,highline));
    If(MarketPosition<>-1 And Low<=lowline)
      SellShort(lots,Min(Open,lowline));
  }
}
```

```
//--------------------------------------------------------------------
// 编译版本：  2020/02/14 112835
// 版权所有    tbyihao
// 更改声明    TradeBlazer Software 保留对 TradeBlazer 平台
//         每一版本的 TradeBlazer 公式修改和重写的权利
//--------------------------------------------------------------------
```

再次加载策略，并打开 K 线，可以看到各个品种的运行情况。

注意：叠加方式只有一个策略单元，一个策略单元只有一份资金账户，所以所有叠加的品种用的是同一个虚拟账户的资金。所以测试报告的组合资金是 100 万。测试报告也会和批量方式有所不一样。

序号	组名	策略单元	公式应用	数据源	初始资金	净值	净利润	年化收益率%	最大开仓...	最大开仓市值	最大回...	最大回撤...	夏普比率	年化收益...	调整收益...	交易次数
1	FourW...	ag888_D1_汇...	FourWeek		1,000,000.00	4.1896	3,189,617.43	31.46	1,960.2356	18,992,525.00	3,668,0...	92.20	0.1536	0.3412	0.5425	
				ag888 D1	1,000,000.00	0.5484	-451,589.43	-5.80	4.0479	1,276,500.00	757,41...	72.62	-0.2432	-0.0798	-0.2857	
				al888 D1	1,000,000.00	1.2431	243,117.45	2.40	1.1629	999,600.00	305,88...	24.30	0.1870	0.0987	0.1764	
				cu888 D1	1,000,000.00	0.6127	-387,349.30	-3.82	1.7321	1,019,200.00	567,41...	51.50	-0.2432	-0.0742	-0.2066	
				rb888 D1	1,000,000.00	1.4779	477,863.53	4.71	1.0510	1,056,790.00	546,51...	27.78	0.2386	0.1696	0.2267	
				ru888 D1	1,000,000.00	1.0307	30,653.61	0.30	1.7431	1,000,650.00	735,87...	56.38	0.0106	0.0054	0.0122	
				zn888 D1	1,000,000.00	0.6582	-341,783.60	-3.37	1.6081	1,006,800.00	683,77...	52.95	-0.1933	-0.0637	-0.1106	
				i9888 D1	1,000,000.00	2.3220	1,321,982.74	20.82	0.9548	1,048,800.00	565,20...	31.59	0.6346	0.6590	1.0420	
				j9888 D1	1,000,000.00	2.5100	1,509,980.84	17.05	0.9315	999,500.00	504,58...	20.32	0.7634	0.8393	1.1648	
				l9888 D1	1,000,000.00	0.5056	-494,371.55	-4.88	2.0659	1,017,000.00	962,05...	67.42	-0.3335	-0.0723	-0.2522	
				m9888 D1	1,000,000.00	1.5010	501,016.50	4.94	1.0858	1,017,130.00	328,76...	22.36	0.3286	0.2210	0.3501	
				p9888 D1	1,000,000.00	1.2234	223,379.78	2.20	1.2775	1,019,840.00	411,93...	31.81	0.1288	0.0693	0.1099	
				pp888 D1	1,000,000.00	1.4117	411,676.57	6.88	0.9506	1,018,380.00	685,10...	33.39	0.4314	0.2060	0.5127	
				y9888 D1	1,000,000.00	0.6296	-370,437.84	-3.65	1.9422	1,016,320.00	774,90...	60.38	-0.2766	-0.0605	-0.2281	
		汇总(19)	FourWeek		1,000,000.00	4.1896	3,189,617.43	31.46	1,960.2356	18,992,525.00	68,075.88	92.20	0.1536	0.3412	0.5425	910

3、策略交易

策略交易是专门用于实盘交易的策略单元的集合。
策略交易相对于策略研究需要设置头寸管理，需要启动交易等。另外在优化方面多了定时优化。
关于策略交易的详细介绍可以在 TradeBlazer 官网的帮助中心\ TBQuant 软件使用的相应章节查看。
如果我们准备使用刚刚的四周组合做实盘交易的话，只需要导出策略单元然后在策略交易的工作区导入即可。

策略交易

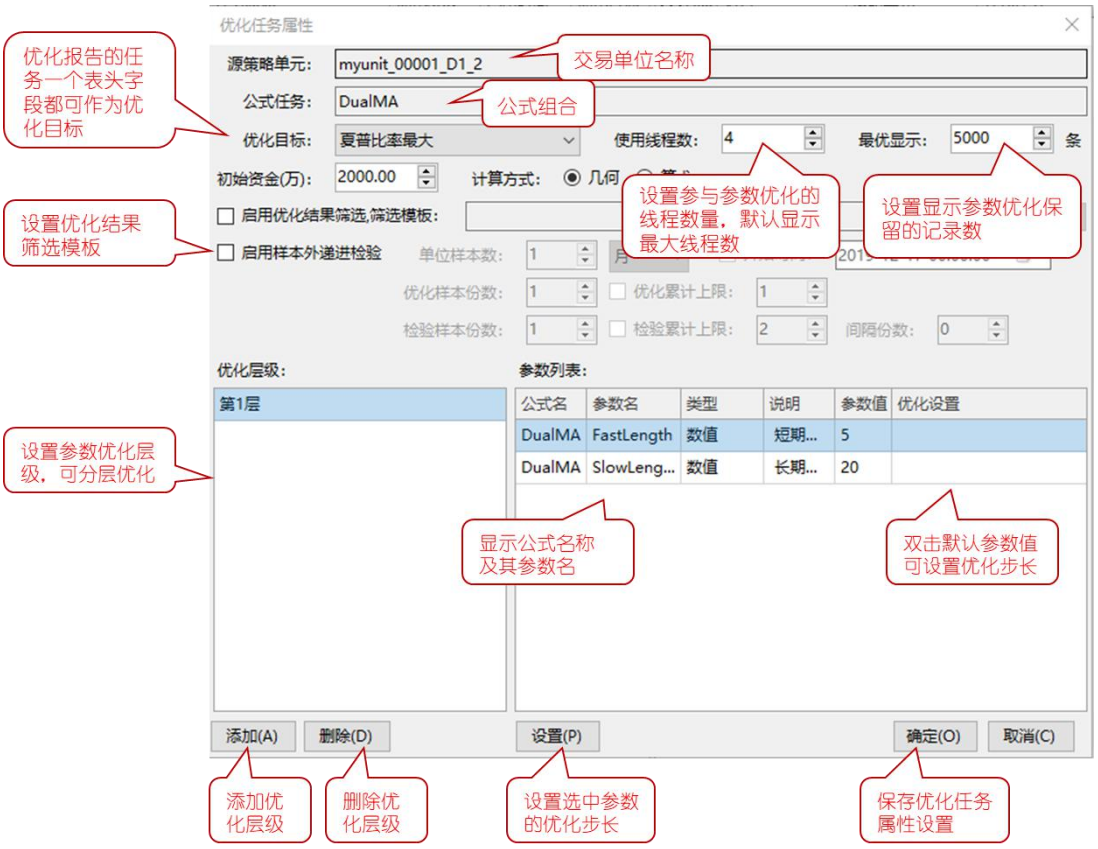
序号	状态	组名	单元名	公式	商品代码	商品简称	周期	分类	算法	起始日期	结束日期	更新时间	bar数	仓位	当日盈亏	持仓盈亏
1	全自动	FourWeek	ag888_D1	FourWeek	ag888	白银连续	1日	贵金属	×	2012/05/11	2020/02/17		1890	-20	-10,500.00	-22,800.00
2	全自动	FourWeek	al888_D1	FourWeek	al888	沪铝连续	1日	有色金属	×	2010/01/04	2020/02/17		2458	-20	1,000.00	-8,000.00
3	全自动	FourWeek	CF888_D1	FourWeek	CF888	棉花连续	1日	软商品	×	2010/01/04	2020/02/17		2458	-30	-45,000.00	-114,000.00
4	全自动	FourWeek	cu888_D1	FourWeek	cu888	沪铜连续	1日	有色金属	×	2010/01/04	2020/02/17		2458	-2	-1,400.00	24,500.00
5	全自动	FourWeek	FG888_D1	FourWeek	FG888	玻璃连续	1日	软商品	×	2012/12/04	2020/02/17		1749	-23	-460.00	-9,660.00
6	全自动	FourWeek	i9888_D1	FourWeek	i9888	铁矿石连续	1日	黑色相关	×	2013/10/18	2020/02/17		1544	-6	0.00	-13,200.00
7	全自动	FourWeek	j9888_D1	FourWeek	j9888	焦炭连续	1日	黑色相关	×	2011/04/18	2020/02/17		2149	-4	3,600.00	-16,600.00
8	全自动	FourWeek	l9888_D1	FourWeek	l9888	塑料连续	1日	能源化工	×	2010/01/04	2020/02/17		2456	-19	-950.00	40,375.00
9	全自动	FourWeek	m9888_D1	FourWeek	m9888	豆粕连续	1日	农产品	×	2010/01/04	2020/02/17		2458	-13	-1,170.00	18,330.00
10	全自动	FourWeek	MA888_D1	FourWeek	MA888	甲醇连续	1日	能源化工	×	2014/06/18	2020/02/17		1381	-50	-4,000.00	-3,000.00
11	全自动	FourWeek	p9888_D1	FourWeek	p9888	棕榈油连续	1日	农产品	×	2010/01/04	2020/02/17		2458	-29	-20,300.00	116,580.00
12	全自动	FourWeek	pp888_D1	FourWeek	pp888	聚丙烯连续	1日	能源化工	×	2014/02/28	2020/02/17		1453	-13	-2,275.00	36,530.00
13	全自动	FourWeek	rb888_D1	FourWeek	rb888	螺纹钢连续	1日	黑色相关	×	2010/01/04	2020/02/17		2458	-28	-3,640.00	-49,840.00
14	全自动	FourWeek	ru888_D1	FourWeek	ru888	橡胶连续	1日	软商品	×	2010/01/04	2020/02/17		2458	-32	-27,200.00	334,400.00
15	全自动	FourWeek	SR888_D1	FourWeek	SR888	白糖连续	1日	软商品	×	2010/01/04	2020/02/17		2458	-30	-29,700.00	-106,500.00
16	全自动	FourWeek	TA888_D1	FourWeek	TA888	PTA连续	1日	能源化工	×	2010/01/04	2020/02/17		2458	-38	-3,420.00	62,320.00
17	全自动	FourWeek	y9888_D1	FourWeek	y9888	豆油连续	1日	农产品	×	2010/01/04	2020/02/17		2458	0	0.00	0.00
18	全自动	FourWeek	ZC888_D1	FourWeek	ZC888	动力煤连续	1日	黑色相关	×	2015/05/18	2020/02/17		1157	-12	8,880.00	-12,960.00
19	全自动	FourWeek	zn888_D1	FourWeek	zn888	沪锌连续	1日	有色金属	×	2010/01/04	2020/02/17		2458	0	0.00	0.00

4、参数优化

虽然四周策略使用默认参数在过去 10 多年是盈利的，但是最近 3 年绩效开始下降，那能否优化下参数，使得最近几年盈利呢？首先在 TBQuant 中如何优化参数呢，我们先简单介绍下 TBQuant 的优化参数可以使用的方法。最后我们选用普通的参数优化来看下是否可以为四周找到更好的参数。

1）普通的参数优化

策略代码编写完毕，如何评判该策略是否可行呢？通过对历史数据加载策略，进行回测，查看回测结果，用户可以了解策略的期望收益和可能承受的风险，从而了解策略的可行性。策略可行性可从策略本身的测试效果、以及和其他策略的组合效果等多个方面来进行评判。
这是参数优化任务设置的主界面。



2）样本外递进检验

样本外递进检验是严格意义上的样本外检验。这种优化算法，使用一段历史数据进行参数优化，选择参数，应用参数到优化样本外的检验样本中。不断推进样本递进优化，并提取检验样本的测试结果形成测试报告。样本外递进检验比起单纯的历史测试，能更真实的反应参数选取对未来绩效的影响。参数优化的重心更加偏向于如何确定参数选取的方法，而不是简单的历史绩效的比较。

3）定时优化

定时优化应用就是对策略交易单元的优化任务设置定时器，周期性的自动优化参数并应用。比如设置了起始时点和执行周期，那么从起始时点开始每隔一个执行周期，都会触发一次自动优化并应用参数到交易单元。

4）批量优化

除了单策略优化、按照策略研究中对选中优化任务进行参数优化外，还可以通过“策略优化”对所有的策略单元优化任务进行批量优化。在策略优化管理器可以查看运行中和已完成的任务，并进行相应管理。

5）策略组合优化报告

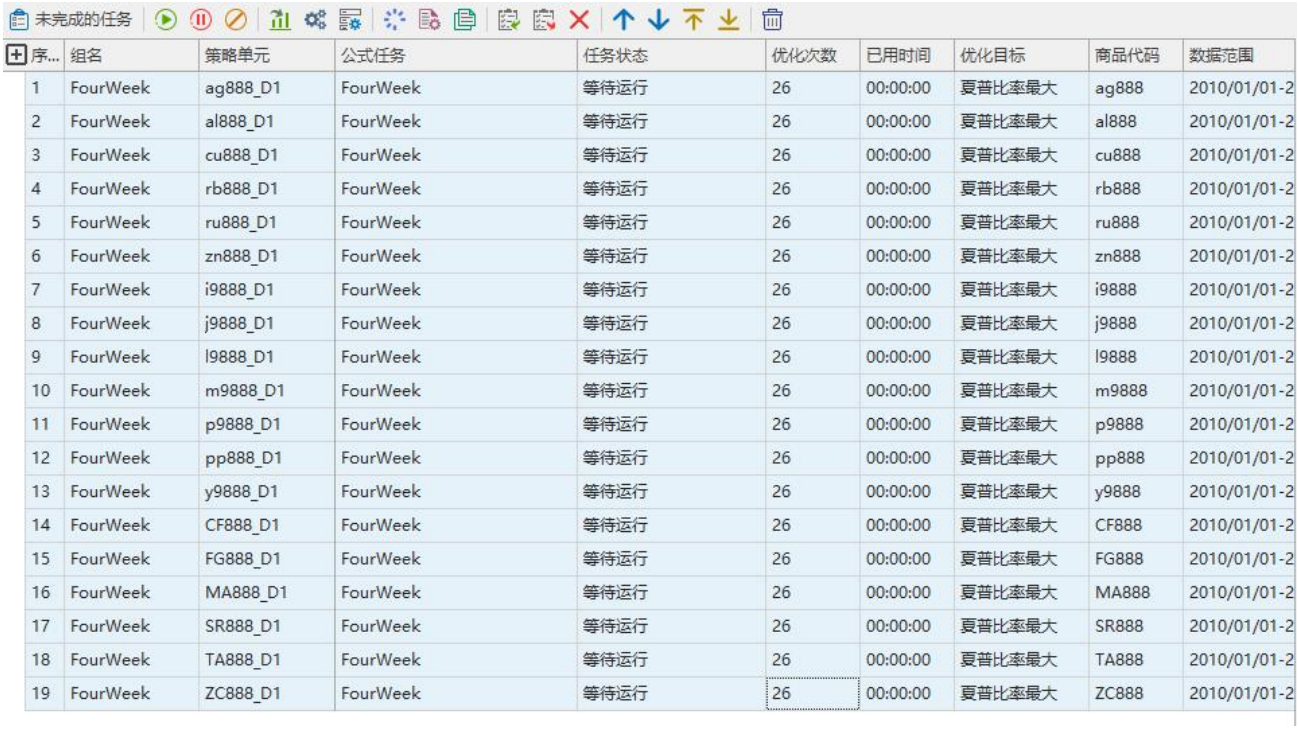
策略优化生成了对于每一个策略单元的优化结果。用户可以知道每一个策略单元应该选择哪个参数的报告绩效最好。如果用户想知道多个策略单元组合选择什么参数，使得组合的报告绩效最好，那么用户可以使用策略组合优化报告的功能。策略组合优化报告有两种组合方式，一种是多个策略单元都必须用一样的参数，前提是多个策略单元使用的也是一样的策略，并且按照同样的参数优化设置；另外一种是每个策略单元可以使用各自的参数，这种对多个策略单元的策略没有限制。第一种方式叫做同策略，第二种方式叫做交叉组合。

6）优化四周的参数

首先，我们修改下四周策略，把周期 20 设置为一个参数。

```
Params
   Numeric money(10);          //固定资金开仓：单位万
   Numeric length(20,10,60,2);    //四周周期参数
```

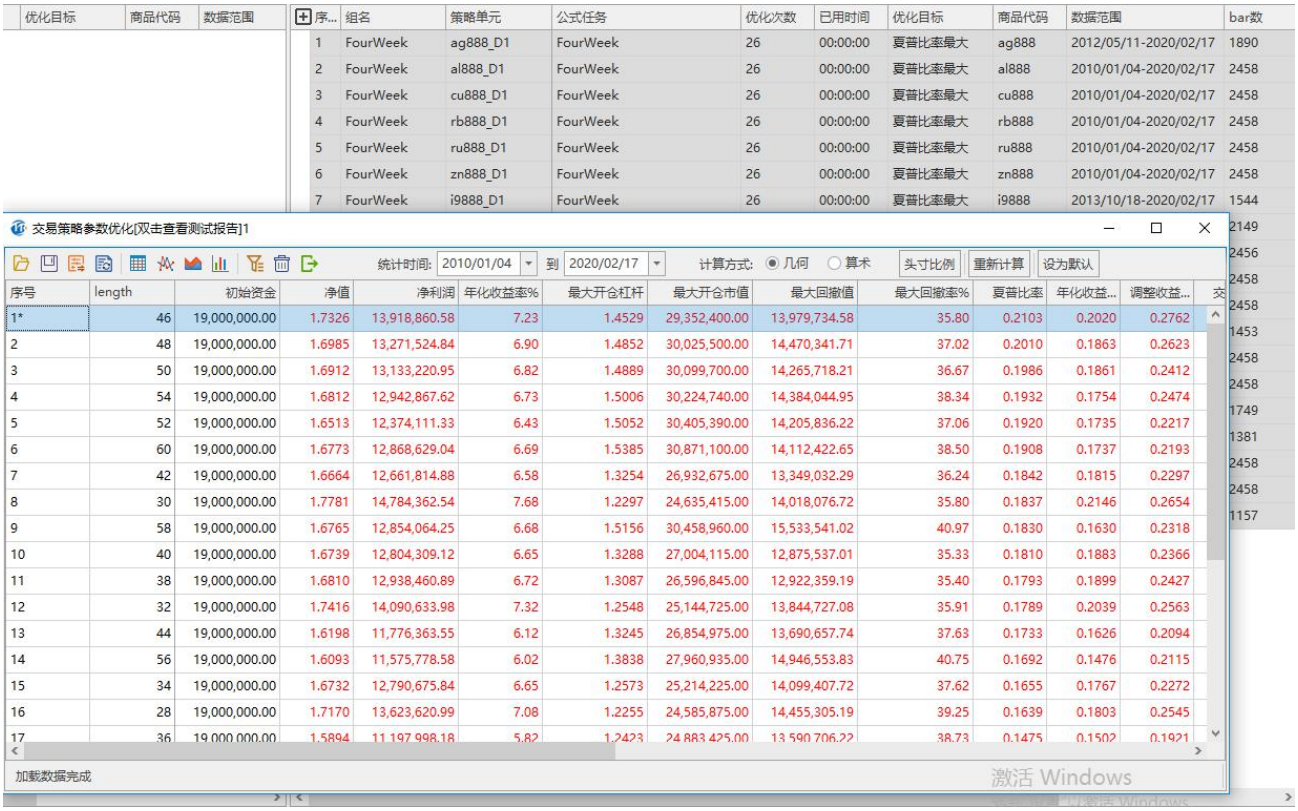
在这里我们需要优化的参数是 length。我们需要设定了 length 的默认值、最小值、最大值、步长。
而这些可以在优化的界面进行设置，也可以在定义这个参数的时候直接在参数名字后面指定。比如 Numeric length(20,10,60,2)，这个就说明 length 的默认值是 20，最小值是 10，最大值是 60，步长是 2。
我们在策略研究，勾选所有的策略单元，右键新建优化任务。然后到策略优化界面，设置优化参数。



未完成的任务

序...	组名	策略单元	公式任务	任务状态	优化次数	已用时间	优化目标	商品代码	数据范围
1	FourWeek	ag888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	ag888	2010/01/01-2
2	FourWeek	al888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	al888	2010/01/01-2
3	FourWeek	cu888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	cu888	2010/01/01-2
4	FourWeek	rb888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	rb888	2010/01/01-2
5	FourWeek	ru888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	ru888	2010/01/01-2
6	FourWeek	zn888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	zn888	2010/01/01-2
7	FourWeek	i9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	i9888	2010/01/01-2
8	FourWeek	j9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	j9888	2010/01/01-2
9	FourWeek	l9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	l9888	2010/01/01-2
10	FourWeek	m9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	m9888	2010/01/01-2
11	FourWeek	p9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	p9888	2010/01/01-2
12	FourWeek	pp888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	pp888	2010/01/01-2
13	FourWeek	y9888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	y9888	2010/01/01-2
14	FourWeek	CF888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	CF888	2010/01/01-2
15	FourWeek	FG888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	FG888	2010/01/01-2
16	FourWeek	MA888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	MA888	2010/01/01-2
17	FourWeek	SR888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	SR888	2010/01/01-2
18	FourWeek	TA888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	TA888	2010/01/01-2
19	FourWeek	ZC888_D1	FourWeek	等待运行	26	00:00:00	夏普比率最大	ZC888	2010/01/01-2

优化后的结果中，我们右键选择同策略。查看最优的参数是 46。
我们可以看到净值/夏普比率都有所提升。



优化目标	商品代码	数据范围

序...	组名	策略单元	公式任务	优化次数	已用时间	优化目标	商品代码	数据范围	bar数
1	FourWeek	ag888_D1	FourWeek	26	00:00:00	夏普比率最大	ag888	2012/05/11-2020/02/17	1890
2	FourWeek	al888_D1	FourWeek	26	00:00:00	夏普比率最大	al888	2010/01/04-2020/02/17	2458
3	FourWeek	cu888_D1	FourWeek	26	00:00:00	夏普比率最大	cu888	2010/01/04-2020/02/17	2458
4	FourWeek	rb888_D1	FourWeek	26	00:00:00	夏普比率最大	rb888	2010/01/04-2020/02/17	2458
5	FourWeek	ru888_D1	FourWeek	26	00:00:00	夏普比率最大	ru888	2010/01/04-2020/02/17	2458
6	FourWeek	zn888_D1	FourWeek	26	00:00:00	夏普比率最大	zn888	2010/01/04-2020/02/17	2458
7	FourWeek	i9888_D1	FourWeek	26	00:00:00	夏普比率最大	i9888	2013/10/18-2020/02/17	1544

交易策略参数优化[双击查看测试报告]1

统计时间: 2010/01/04 到 2020/02/17 计算方式: 几何 算术 头寸比例 重新计算 设为默认

序号	length	初始资金	净值	净利润	年化收益率%	最大开仓杠杆	最大开仓市值	最大回撤值	最大回撤率%	夏普比率	年化收益...	调整收益...	交
1*	46	19,000,000.00	1.7326	13,918,860.58	7.23	1.4529	29,352,400.00	13,979,734.58	35.80	0.2103	0.2020	0.2762	
2	48	19,000,000.00	1.6985	13,271,524.84	6.90	1.4852	30,025,500.00	14,470,341.71	37.02	0.2010	0.1863	0.2623	
3	50	19,000,000.00	1.6912	13,133,220.95	6.82	1.4889	30,099,700.00	14,265,718.21	36.67	0.1986	0.1861	0.2412	
4	54	19,000,000.00	1.6812	12,942,867.62	6.73	1.5006	30,224,740.00	14,384,044.95	38.34	0.1932	0.1754	0.2474	
5	52	19,000,000.00	1.6513	12,374,111.33	6.43	1.5052	30,405,390.00	14,205,836.22	37.06	0.1920	0.1735	0.2217	
6	60	19,000,000.00	1.6773	12,868,629.04	6.69	1.5385	30,871,100.00	14,112,422.65	38.50	0.1908	0.1737	0.2193	
7	42	19,000,000.00	1.6664	12,661,814.88	6.58	1.3254	26,932,675.00	13,349,032.29	36.24	0.1842	0.1815	0.2297	
8	30	19,000,000.00	1.7781	14,784,362.54	7.68	1.2297	24,635,415.00	14,018,076.72	35.80	0.1837	0.2146	0.2654	
9	58	19,000,000.00	1.6765	12,854,064.25	6.68	1.5156	30,458,960.00	15,533,541.02	40.97	0.1830	0.1630	0.2318	
10	40	19,000,000.00	1.6739	12,804,309.12	6.65	1.3288	27,004,115.00	12,875,537.01	35.33	0.1810	0.1883	0.2366	
11	38	19,000,000.00	1.6810	12,938,460.89	6.72	1.3087	26,596,845.00	12,922,359.19	35.40	0.1793	0.1899	0.2427	
12	32	19,000,000.00	1.7416	14,090,633.98	7.32	1.2548	25,144,725.00	13,844,727.08	35.91	0.1789	0.2039	0.2563	
13	44	19,000,000.00	1.6198	11,776,363.55	6.12	1.3245	26,854,975.00	13,690,657.74	37.63	0.1733	0.1626	0.2094	
14	56	19,000,000.00	1.6093	11,575,778.58	6.02	1.3838	27,960,935.00	14,946,553.83	40.75	0.1692	0.1476	0.2115	
15	34	19,000,000.00	1.6732	12,790,675.84	6.65	1.2573	25,214,225.00	14,099,407.72	37.62	0.1655	0.1767	0.2272	
16	28	19,000,000.00	1.7170	13,623,620.99	7.08	1.2255	24,585,875.00	14,455,305.19	39.25	0.1639	0.1803	0.2545	
17	36	19,000,000.00	1.5894	11,197,998.18	5.82	1.2423	24,883,425.00	13,590,706.22	38.73	0.1475	0.1502	0.1921	

加载数据完成

激活 Windows

如果对测试报告选择交叉组合，则能得到更优的组合效果。我们可以看到净值更高，并且 2020 年还创了新高。

但是交叉组合相比于同参数，历史拟合的嫌疑更大。

7）优化的误区

虽然使用交叉组合比同参数的净值曲线好，但是在实际当中我们要注意避免陷入参数拟合的误区。

当可优化的参数越多，可优化的范围越大，那么就越有可能找到更好的交易结果。但是这个交易结果在未

来实盘能否重现，这个是读者需要思考的。所以选择参数不是一定要选择历史效果最好的，而是要选择预期未来效果最好的。

三、四周规则策略改进

3.1 改进一：增加跟踪止盈

1、基本方法说明

趋势跟踪是基于对未来行情没有预期的情况下， 根据技术规则对未来走势进行的应对。四周本身就是一种趋势跟踪的手段，但也有在盈利很多的情况下，没能及时出场，导致回撤较大。这种情况下，我们可以增加一种盈利跟踪的手段，跟踪止盈。

跟踪止盈就是在盈利幅度较大的情况下，回撤一定幅度就及时出场的技术手段。

回撤幅度可以用价格百分比、ATR 倍数等等，这里我们用价格百分比作为例子。默认盈利超过进场价格的 6%作为盈利幅度较大的条件，回撤默认 2%作为出场条件。

2、基本代码说明

所以我们需要增加两个变量来记录盈利的高点。

```
If(MarketPosition==1)
  {
    If(BarsSinceEntry==0)
      buylasthigh=EntryPrice;
    Else
      buylasthigh=Max(High,buylasthigh);
  }
  If(MarketPosition==-1)
  {
    If(BarsSinceEntry==0)
      selllastlow=EntryPrice;
    Else
      selllastlow=Min(Low,selllastlow);
  }
```

同样我们也要增加相应的代码，来触发跟踪止盈。

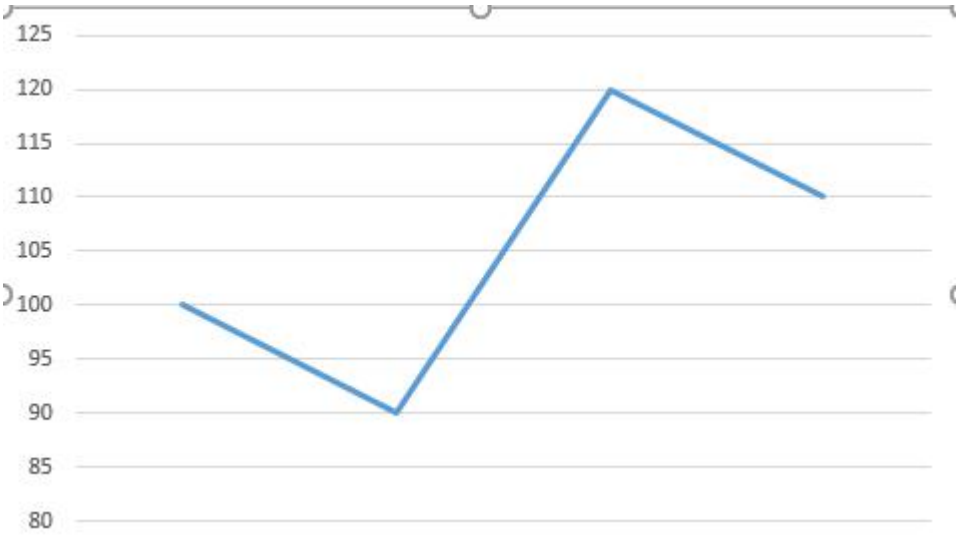
```
f(MarketPosition==1 and Low<=buylasthigh*(1-0.01*hcrate) and buylasthigh>EntryPrice*(1+0.01*6))
    Sell(0,Min(Open,buylasthigh*(1-0.01*hcrate)));
                if(MarketPosition==-1     and     High>=selllastlow*(1+0.01*hcrate)     and
selllastlow<EntryPrice*(1-0.01*6))
    BuyToCover(0,Max(Open,selllastlow*(1+0.01*hcrate)));
```

而在 TBL 语言中代码的运行顺序是从上到下的。所以我们不能把更新盈利高低点的代码放在跟踪止盈代码的前面。因为如果这样写，在历史 K 线中，我们可能直接就读取了未来数据。

比如这样一个情形，有这么一根 BAR。开盘价是 100，收盘价是 110，最高价是 120，收盘价是 110。我们上一根 BAR 收盘买入，入场价格是 100。我们准备从盈利高点回落 20 个点就止损。

如果当前 BAR 先突破新高达到了 120，然后回落到 90。那么按理说应该止损的。

如果当前根 BAR 先跌落到 90，然后一直上涨到 120。那么按理说不应该止损的。

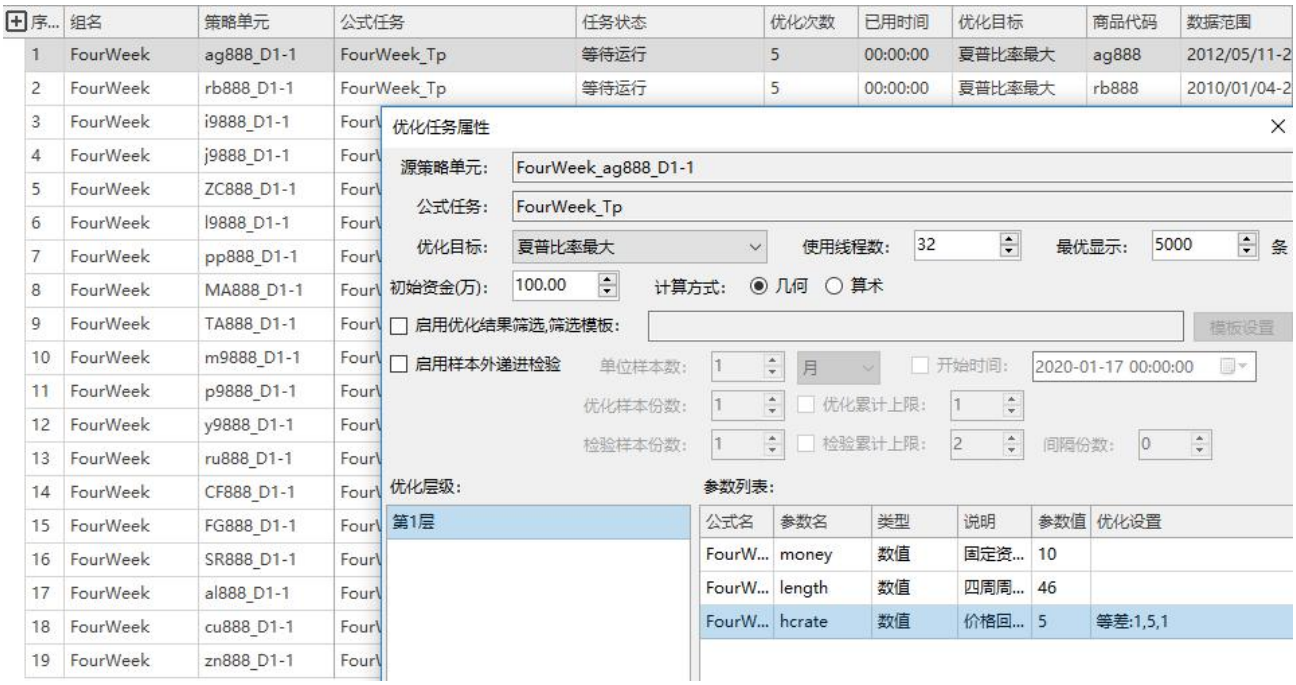


但是作为历史的 BAR 不会显示这个过程的， 我们看到的只有开高低收这四个价格，所以我们不知道该不该平仓。

为了避免这样的一种不确定，所以我们不能使用当根 BAR 的高低点来更新盈利峰值，然后再进行跟踪止盈。我们只能把跟踪止盈的代码放在更新盈利峰值的前面。这样我们使用的盈利峰值其实是根据上一根 BAR 计算出来的。

3、优化止盈参数

因为不同品种的波动率是不一样的，使用默认的 2%并不适合所有品种，所以还需要启动一次参数优化。我们在之前四周的策略单元里面，换掉公式为 fourweek_Tp,然后启动参数优化，优化设置为下图。我们把四周周期参数使用之前优化好的结果 46，这一次只优化追踪止盈的参数。



4、测试报告

优化完之后，我们打开优化后的测试报告可以看出，增加了追踪止盈之后，组后的净值下降为了 1.37 但是夏普比率从 0.21 提升到了 0.51。这说明增加追踪止盈是有意义的。

5、完整代码

```
Params
   Numeric money(100);          //固定市值开仓：单位万
   Numeric length(20,10,60,2);    //四周周期参数
   Numeric hcrate(2,1,5,1);    //价格回撤幅度
Vars
```

```
  Numeric highline;       //高点连线
  Numeric lowline;       //低点连线
  Numeric myprice;       //委托价格
  Numeric lots;         //委托数量
  Series<Numeric> buylasthigh(0,2);  //买持仓价格峰值
  Series<Numeric> selllastlow(0,2);  //卖持仓价格低谷
Events
OnInit()
{
  //=========除权换月相关设置==============
  AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
  AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
  AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
  //=========交易相关设置==============
  SetInitCapital(1000000);  //设置初始资金为 100 万
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续
费率为成交金额的 5%%，不收平今， BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
}
OnBar(ArrayRef<Integer> indexs)
{
  myprice=Open;//这里使用 open,更为精确的是使用委托价格
  lots=IntPart(money*10000/(myprice*contractunit*BigPointValue)); //计算开仓手数
  highline=Highest(High[1],length);
  lowline=Lowest(Low[1],length);
If(MarketPosition<>1 And High>=highline)
    Buy(lots,Max(Open,highline));
  If(MarketPosition<>-1 And Low<=lowline)
    SellShort(lots,Min(Open,lowline));
                  if(MarketPosition==1      and      Low<=buylasthigh*(1-0.01*hcrate)      and
buylasthigh>EntryPrice*(1+0.01*6))
    Sell(0,Min(Open,buylasthigh*(1-0.01*hcrate)));
                 if(MarketPosition==-1      and      High>=selllastlow*(1+0.01*hcrate)      and
selllastlow<EntryPrice*(1-0.01*6))
    BuyToCover(0,Max(Open,selllastlow*(1+0.01*hcrate)));
  If(MarketPosition==1)
  {
    If(BarsSinceEntry==0)
      buylasthigh=EntryPrice;
    Else
      buylasthigh=Max(High,buylasthigh);
  }
  If(MarketPosition==-1)
```

```
  {
    If(BarsSinceEntry==0)
      selllastlow=EntryPrice;
    Else
      selllastlow=Min(Low,selllastlow);
  }
}
```

3.2 跟踪止盈代码写法的缺陷

1、如何改进？大周期信号，小周期交易

在上面的写法中，我们盈利的最高点，只能用上一根 BAR 的。因为如果用当根 BAR 的高低点，那么可能不知道高点和低点谁先出来的，因为一根 BAR 提供的信息是汇总的，并没有提供过程。
这样写可以保证当前 BAR 在实时运行和历史回测运行时，运行结果的一致。但是如果使用上一根 BAR 的高点，而在我们的案例中一根 BAR 是一天，一天之内行情可能剧烈波动，而我们使用上面的代码则对日内行情波动无动于衷。如果个别读者可能还使用月 K 线和年 K 线做交易，那么他承受的一根 BAR 之内的波动就更大了。
所以有些读者就在想，跟踪止盈能不能在更详细的小周期上进行跟踪，比如 5mins 图表上进行跟踪止盈。其实这就涉及到跨周期，在大的周期上出信号，小的周期上进行跟踪止盈。

2、实现的代码及说明

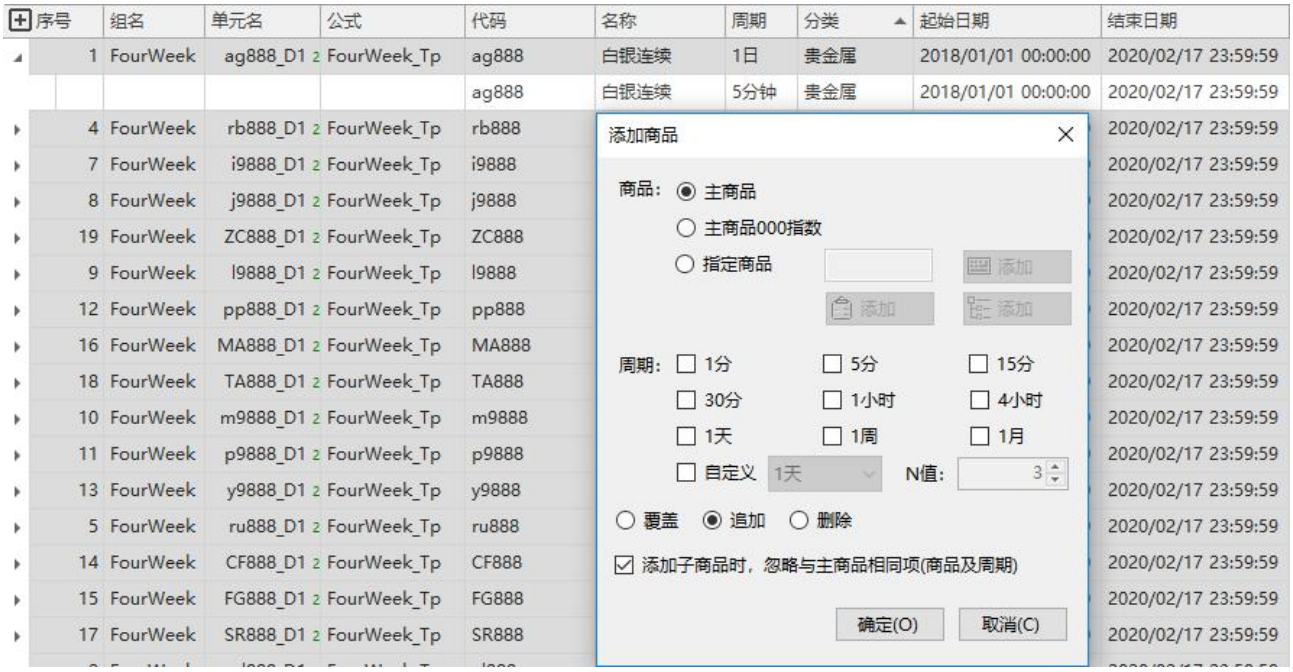
关于跨周期因为 TBQuant 已经提供了不同数据源的自动对齐机制，所以如果我们叠加一个日线和一个 5mins 数据源，那么只需要 5mins 数据源上能读到日线上的指标就可以实现大周期信号，小周期追踪止盈了。
关于实现的代码如下：
其实很简单，只需要把哪些语句在 data0 上运行用 Range[0:0]括起来，哪些语句在 data0 上运行用 Range[1:1]括起来。Data1 数据源上使用 data0 的变量则前面要加 data0.的前缀。

```
OnBar(ArrayRef<Integer> indexs)
{
  Range[0:0]
  {
    myprice=Open/rollover;//这里使用 open,更为精确的是使用委托价格
    lots=IntPart(money*10000/(myprice*contractunit*BigPointValue)); //计算开仓手数
    highline=Highest(High[1],length);
    lowline=Lowest(Low[1],length);
  }
  Range[1:1]
  {
                        if(MarketPosition==1    and    Low<=buylasthigh*(1-0.01*hcrate)    and
buylasthigh>EntryPrice*(1+0.01*6))
      Sell(0,Min(Open,buylasthigh*(1-0.01*hcrate)));
                      if(MarketPosition==-1    and    High>=selllastlow*(1+0.01*hcrate)    and
selllastlow<EntryPrice*(1-0.01*6))
```

```
        BuyToCover(0,Max(Open,selllastlow*(1+0.01*hcrate)));
    If(MarketPosition<>1 And High>=data0.highline)      //来自 data0 的变量，前面都要加 data0.
        Buy(data0.lots,Max(Open,data0.highline));
    If(MarketPosition<>-1 And Low<=data0.lowline)
        SellShort(data0.lots,Min(Open,data0.lowline));
    If(MarketPosition==1)
    {
        If(BarsSinceEntry==0)
            buylasthigh=EntryPrice;
        Else
            buylasthigh=Max(High,buylasthigh);
    }
    If(MarketPosition==-1)
    {
        If(BarsSinceEntry==0)
            selllastlow=EntryPrice;
        Else
            selllastlow=Min(Low,selllastlow);
    }
  }
}
```

3、数据源的多周期的操作

我们已经改写了策略为跨周期操作，那么现在我们需要到原来的策略单元中把数据源也变成大小周期叠加。这时候，我们只需要打开原来的策略研究，选中所有的策略单元，这些策略单元已经是日线的了，所以我们追加一个 5mins 的就可以。这个操作通过右键，勾选数据源，周期勾选 5mins，选择追加可以便捷实现。

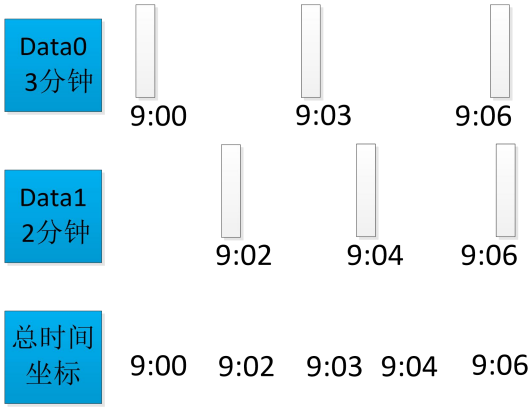


4、多数据源的 onbar 机制

跨周期的实现需要读者了解 onbar 的多数据源的运行机制。有心的读者可以认真阅读下面的内容。
Onbar 在多数据源上的运行的机制虽然基本特征依旧满足单数据源的 onbar 机制的特征：1、历史数据是从左到右从上到下的运行。2、实时行情每个 tick 驱动一次。但是多数据要比在一个数据源上的机制复杂，需要考虑各个数据源的时间问题，这里面既有不同周期的数据对齐，也有不同交易时间的特殊处理。这里我们重点讲解多数据源 onbar 机制的 4 个方面：1、数据的对齐机制 ；2、数据的编号；3、公式的运行机制；4、特殊情况的补充机制。

1）数据的对齐机制

这里简单介绍下数据对齐机制，关于对齐之后的运行机制在本文后面章节详细介绍。
比如 data0 是 3 分钟，data1 是 2 分钟的，将这两个数据源叠加在一起，会怎样对齐？



1）将各个数据源的时间坐标求出并集，这样多数据源可以参照总坐标对齐。如上图所示。
2）公式在一个数据源上运行时，可以获取其它数据源当前时间对应的数据，无值则向前取值。这样能保证任何一个数据源读取其它数据没有使用未来数据。
3）所有数据源的最后一根 BAR 肯定是对齐的。

2）数据源的编号

当有多个数据源时，系统会给这些数据源进行编号，编号从 0 依次累加。如果总共有三个合约，那么三个数据源的序号是 0，1，2。如果需要调用相应数据源的数据或者公式运算，则直接用 data[i].的前缀就可以调用第 i+1 个（因为编号从 0 开始）数据源的数据或者公式运算。

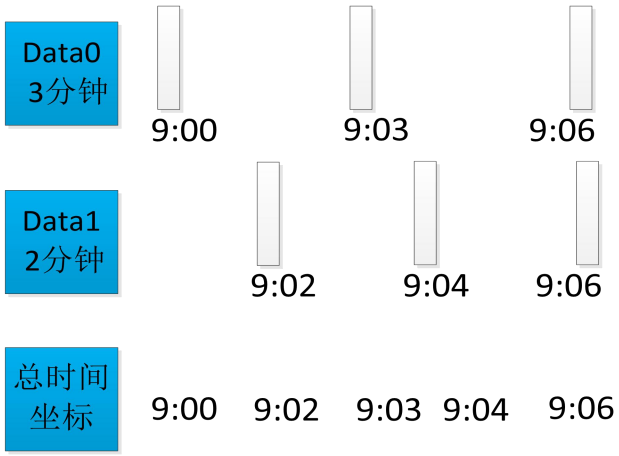
3）公式运行的机制

3.1）公式运行机制的建立：在每个数据源上建立局部变量的备份

当只有一个数据源时，公式依赖于 data0 这唯一的数据源运行。这体现在除了全局变量其余的局部变量都是依赖于 data0 的数据源的。
当有多个数据源时，公式会在每个数据源都建立一套机制，对应的除了全局变量其余的局部变量在每个数据源都有自己独立的存在。
比如公式定义一个普通的数值变量 numeric num(0)；如果公式应用于有三个数据源的交易单元，则实际上存在三个独立的 num，分别是 data0.num,data1.num,data2.num。

3.2）多数据源的运行机制什么时候被触发？

对于历史数据，公式会每个总时间坐标上运行一次。
对于实盘数据，任何一个数据源的 tick 更新都会触发公式运行一次。运行的数据源是基于每个数据源上的最后一根 BAR。但是公式执行过程中，如有多次更新会合并触发下一次运行。
比如下面的例子：数据和代码分别如下：




```
if (Data0. Open - Data1. Open) >10    (1)
{
  Data0.Buy(1, Open);                 (2)
  Data1.Sellshort(1, Open);           (3)
}
```

a）对于历史数据，公式会在每个时间坐标（9:00，9:02，9:03 9:04，9:06）上都会运行一次。上述公式运行时，每条语句的执行情况如下表所示：

时间坐标	（1）执行	（2）执行	（3）执行
9:00	✓	✓	
9:02	✓		✓
9:03	✓	✓	
9:04	✓		✓
9:06	✓	✓	✓

b)对于实时数据，上述公式运行时，每条语句的执行情况如下表所示：

新 tick	（1）执行	（2）执行	（3）执行
Data0	✓	✓	
Data1	✓		✓

各数据源在计算指标时不补缺失的数据，直接取最原始的数据进行计算。
如 Data0.var1 = Data1.averageFC(Close，10)，取 Data1 最近 10 根有效的 bar 的 Close 计算均价。

3.3）哪些代码会在哪个数据源上被执行

Onbar{}之间的代码在那个数据源上运行则依赖 range[i:j]和 data[i].这两个机制控制。Range 和 data 中的下标 i 与数据源的编号对应。
3.3.1）data[i].对单条代码运行的控制
Data[i].可以加在局部变量前面，这代表这个局部变量是属于 data[i]的那个局部变量；

Data[i].可以加在函数的前面,这代表这个函数运算只在 data[i]上运行,函数中的局部变量如果没 data[j]的前缀，都默认是 data[i]的局部变量。
注意:
a)没有加 data[i].前缀的局部变量默认是 data0 的（除了 data[i].函数中的局部变量）。
b)Data[i]和 datai 等价。
c)没有指定数据源的函数系统默认添加数据源前缀 Data0。
d)函数入参可以是其它数据源的数据，如 Data0.Average（Data1. Close，3）。
e)函数入参不显示数据源则仍默认其数据源为父域，如 Data1.Average（Close，3）中的 Close 指 Data1.Close。
我们举例说明:
比如下面求均值的简单语句

Code 语句	Avg 是哪个图层	Average 是那个图层	Close 是哪个图层
Avg=average(close,5);	Data0	Data0	Data0
Avg=data1.average(close,5);	Data0	Data1	Data1
Avg=average(data1.close,5);	Data0	Data0	Data1
Avg=data1.average(data0.close,5);	Data0	Data1	Data0
Data1.Avg=average(close,5);	Data1	Data0	Data0
Data1.Avg=data1.average(close,5);	Data1	Data1	Data1
Data1.Avg=average(data1.close,5);	Data1	Data0	Data1
Data1.Avg=data1.average(data0.close,5);	Data1	Data1	Data0

3.3.2）RANGE 对代码段的控制

```
Range[i:j]
{
    代码段 codes1;
}
```

如果使用 range[i:j]把代码段 codes1 括起来,则代表这一段代码只在数据源 datai 到 dataj 的数据源运行。I 要小于 j,比如 range[1:2]，则代表代码段 codes1 只在数据源 data1 和 data2 上运行。
具体例子如下:

```
Range[0:1]
  {
    AvgValue1 = AverageFC(Close,FastLength);
    AvgValue2 = AverageFC(Close,SlowLength);
    PlotNumeric("MA1",AvgValue1);
    PlotNumeric("MA2",AvgValue2);
  }
```

上面代码会在 Data[0]和 Data[1]上都运行，分别画出 Data[0]和 Data[1]的双均线。
注意: Range[i:j]内的代码运行说明如下:
a)没有指定数据源的代码，依次在 Data [i]到 Data [j]的 bar 上运行。
b)有指定数据源的代码，则在指定的数据源的 bar 上运行。
c)同一个公式内，可有多个并列的 Range 代码块，Range 内不支持 Range 嵌套。
d)Range[m:n]中，数据源起始位置 m 及结束位置 n 可以指定，也可以为变量和参数声明，类型为全局变量。

3.3 改进二：大周期信号过滤

上一节我们介绍了大周期出信号，小周期交易的做法。并引入了 onbar 的多数据源机制。有了这个多数据源的机制，其实跨品种跨周期都是非常方便的可以实现。本节我们准备尝试另外一种经典的思路来改进四周策略，那就是滤网策略。滤网策略就是用大周期去决定交易方向，过滤小周期上频繁的交易。
作为例子，我们准备大周期日线用均线过滤，小周期 15mins 使用四周规则。

1、均线过滤

我们在 fourweek_Tp 的基础上加上均线过滤，代码的改进其实非常简单，在大周期 data0 上计算均线，然后在小周期 data1 上调用即可。
完整策略代码如下：

```
Params
  Numeric money(10);        //固定资金开仓：单位万
  Numeric length(20,10,60,2);   //四周周期参数
  Numeric hcrate(2,1,5,1);   //价格回撤幅度
  Numeric malength(60,20,120,10); //均线周期参数
Vars
  Numeric highline;        //高点连线
  Numeric lowline;        //低点连线
  Numeric myprice;        //委托价格
  Numeric lots;          //委托数量
  Series<Numeric> buylasthigh(0,2);  //买持仓价格峰值
  Series<Numeric> selllastlow(0,2);  //卖持仓价格低谷
  Numeric ma60;          //均线
Events
OnInit()
{
  Range[0:DataCount-1]
  {
  //=========除权换月相关设置==============
  AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
  AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
  AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
  AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算
  //=========交易相关设置==============
  SetInitCapital(1000000);  //设置初始资金为 100 万
   SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday,Enum_Rate_ByFillAmount),5);   //设置手续费率为成交金额的 5%%，不收平今， BitOr 进行位或运算即设置属性和
  SetSlippage(Enum_Rate_PointPerHand,2);  //设置滑点为 2 跳/手
  SetBeginBarMaxCount(1);
  }
}
OnBar(ArrayRef<Integer> indexs)
{
  Range[0:0]
  {
```

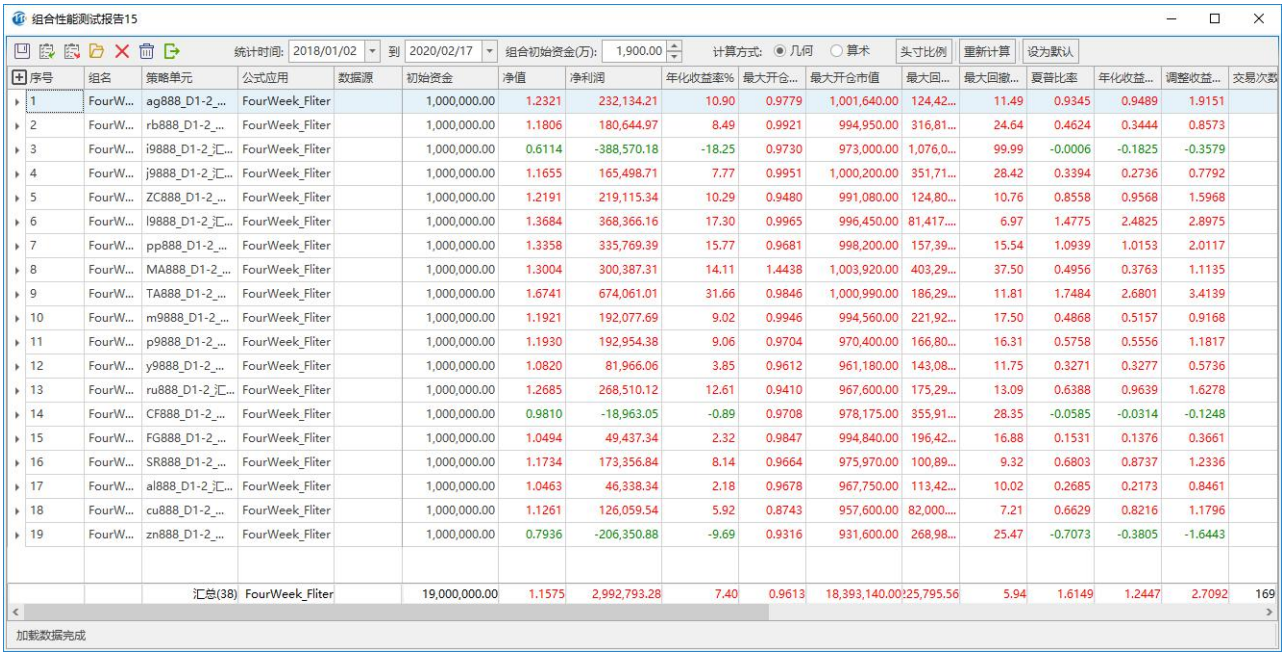
```
    myprice=Open;//这里使用 open,更为精确的是使用委托价格
    lots=IntPart(money*10000/(myprice*contractunit*BigPointValue*MarginRatio)); //计算开仓手
数
    ma60=Average(Close[1],malength);
  }
  Range[1:1]
  {
    myprice=Open;//这里使用 open,更为精确的是使用委托价格
    lots=IntPart(money*10000/(myprice*contractunit*BigPointValue*MarginRatio)); //计算开仓手
数
    highline=Highest(High[1],length);
    lowline=Lowest(Low[1],length);
    PlotNumeric("highline",highline);
    PlotNumeric("lowline",lowline);
    PlotNumeric("maline",data0.ma60);
                    if(MarketPosition==1    and    Low<=buylasthigh*(1-0.01*hcrate)    and
buylasthigh>EntryPrice*(1+0.01*6))
      Sell(0,Min(Open,buylasthigh*(1-0.01*hcrate)));
                   if(MarketPosition==-1    and    High>=selllastlow*(1+0.01*hcrate)    and
selllastlow<EntryPrice*(1-0.01*6))
      BuyToCover(0,Max(Open,selllastlow*(1+0.01*hcrate)));
    If(MarketPosition<>1 And High>=highline And Close[1]>data0.ma60)
      Buy(lots,Max(Open,highline));
    If(MarketPosition<>-1 And Low<=lowline And Close[1]<data0.ma60)
      SellShort(lots,Min(Open,lowline));
    If(MarketPosition==1)
    {
      If(BarsSinceEntry==0)
        buylasthigh=EntryPrice;
      Else
        buylasthigh=Max(High,buylasthigh);
    }
    If(MarketPosition==-1)
    {
      If(BarsSinceEntry==0)
        selllastlow=EntryPrice;
      Else
        selllastlow=Min(Low,selllastlow);
    }
  }
}
```

2、对比效果

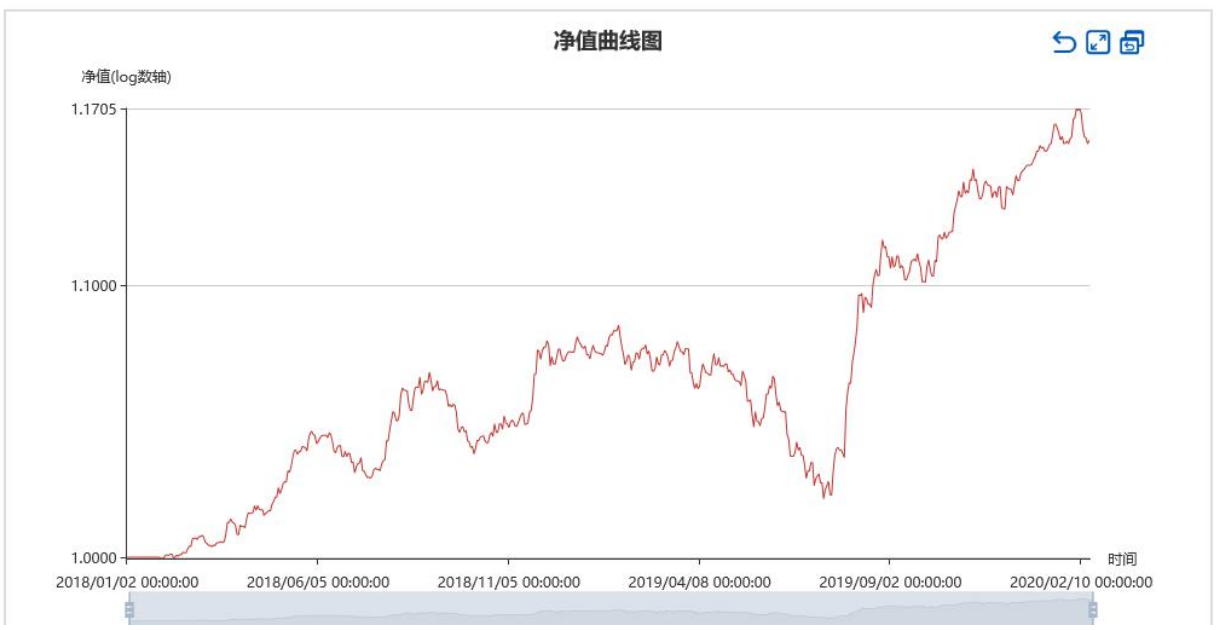
现在我们的参数有四周、均线、追踪止盈，我们同时优化三个参数，看下能够在近三年取得较好的业绩。策略研究我们使用原来的四周，叠加 15mins 的周期，然后启动参数优化。

序号	组名	单元名	公式	代码	名称	周期	分类	起始日期	结束日期	bar数	策略目标	运行状态	运行次数
1	FourWeek	ag888_D1	FourWeek_Fliter	ag888	白银连续	1日	贵金属	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
				ag888	白银连续	15分钟	贵金属	2018/01/02 09:00:00	2020/02/17 14:45:00	18440			
4	FourWeek	rb888_D1	FourWeek_Fliter	rb888	螺纹钢连续	1日	黑色相关	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
7	FourWeek	i9888_D1	FourWeek_Fliter	i9888	铁矿石连续	1日	黑色相关	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
8	FourWeek	j9888_D1	FourWeek_Fliter	j9888	焦炭连续	1日	黑色相关	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
19	FourWeek	ZC888_D1	FourWeek_Fliter	ZC888	动力煤连续	1日	黑色相关	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
9	FourWeek	l9888_D1	FourWeek_Fliter	l9888	塑料连续	1日	能源化工	2018/01/02 00:00:00	2020/02/17 00:00:00	512	夏普比率最大	运行完成	1
12	FourWeek	pp888_D1	FourWeek_Fliter	pp888	聚丙烯连续	1日	能源化工	2018/01/02 00:00:00	2020/02/17 00:00:00	512	夏普比率最大	运行完成	1
16	FourWeek	MA888_D1	FourWeek_Fliter	MA888	甲醇连续	1日	能源化工	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
18	FourWeek	TA888_D1	FourWeek_Fliter	TA888	PTA连续	1日	能源化工	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
10	FourWeek	m9888_D1	FourWeek_Fliter	m9888	豆粕连续	1日	农产品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
11	FourWeek	p9888_D1	FourWeek_Fliter	p9888	棕榈油连续	1日	农产品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
13	FourWeek	y9888_D1	FourWeek_Fliter	y9888	豆油连续	1日	农产品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
5	FourWeek	ru888_D1	FourWeek_Fliter	ru888	橡胶连续	1日	软商品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
14	FourWeek	CF888_D1	FourWeek_Fliter	CF888	棉花连续	1日	软商品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
15	FourWeek	FG888_D1	FourWeek_Fliter	FG888	玻璃连续	1日	软商品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
17	FourWeek	SR888_D1	FourWeek_Fliter	SR888	白糖连续	1日	软商品	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
2	FourWeek	al888_D1	FourWeek_Fliter	al888	沪铝连续	1日	有色金属	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
3	FourWeek	cu888_D1	FourWeek_Fliter	cu888	沪铜连续	1日	有色金属	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1
6	FourWeek	zn888_D1	FourWeek_Fliter	zn888	沪锌连续	1日	有色金属	2018/01/02 00:00:00	2020/02/17 00:00:00	514	夏普比率最大	运行完成	1

我们再优化界面，查看优化结果。



序号	组名	策略单元	公式应用	数据源	初始资金	净值	净利润	年化收益率%	最大开仓...	最大开仓市值	最大回...	最大回撤...	夏普比率	年化收益...	调整收益...	交易次数
1	FourW...	ag888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.2321	232,134.21	10.90	0.9779	1,001,640.00	124,42...	11.49	0.9345	0.9489	1.9151	
2	FourW...	rb888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.1806	180,644.97	8.49	0.9921	994,950.00	316,81...	24.64	0.4624	0.3444	0.8573	
3	FourW...	i9888_D1-2_汇...	FourWeek_Fliter		1,000,000.00	0.6114	-388,570.18	-18.25	0.9730	973,000.00	1,076,0...	99.99	-0.0006	-0.1825	-0.3579	
4	FourW...	j9888_D1-2_汇...	FourWeek_Fliter		1,000,000.00	1.1655	165,498.71	7.77	0.9951	1,000,200.00	351,71...	28.42	0.3394	0.2736	0.7792	
5	FourW...	ZC888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.2191	219,115.34	10.29	0.9480	991,080.00	124,80...	10.76	0.8558	0.9568	1.5968	
6	FourW...	l9888_D1-2_汇...	FourWeek_Fliter		1,000,000.00	1.3684	368,366.16	17.30	0.9965	996,450.00	81,417....	6.97	1.4775	2.4825	2.8975	
7	FourW...	pp888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.3358	335,769.39	15.77	0.9681	998,200.00	157,39...	15.54	1.0939	1.0153	2.0117	
8	FourW...	MA888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.3004	300,387.31	14.11	1.4438	1,003,920.00	403,29...	37.50	0.4956	0.3763	1.1135	
9	FourW...	TA888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.6741	674,061.01	31.66	0.9846	1,000,990.00	186,29...	11.81	1.7484	2.6801	3.4139	
10	FourW...	m9888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.1921	192,077.69	9.02	0.9946	994,560.00	221,92...	17.50	0.4868	0.5157	0.9168	
11	FourW...	p9888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.1930	192,954.38	9.06	0.9704	970,400.00	166,80...	16.31	0.5758	0.5556	1.1817	
12	FourW...	y9888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.0820	81,966.06	3.85	0.9612	961,180.00	143,08...	11.75	0.3271	0.3277	0.5736	
13	FourW...	ru888_D1-2_汇...	FourWeek_Fliter		1,000,000.00	1.2685	268,510.12	12.61	0.9410	967,600.00	175,29...	13.09	0.6388	0.9639	1.6278	
14	FourW...	CF888_D1-2_...	FourWeek_Fliter		1,000,000.00	0.9810	-18,963.05	-0.89	0.9708	978,175.00	355,91...	28.35	-0.0585	-0.0314	-0.1248	
15	FourW...	FG888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.0494	49,437.34	2.32	0.9847	994,840.00	196,42...	16.88	0.1531	0.1376	0.3661	
16	FourW...	SR888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.1734	173,356.84	8.14	0.9664	975,970.00	100,89...	9.32	0.6803	0.8737	1.2336	
17	FourW...	al888_D1-2_汇...	FourWeek_Fliter		1,000,000.00	1.0463	46,338.34	2.18	0.9678	967,750.00	113,42...	10.02	0.2685	0.2173	0.8461	
18	FourW...	cu888_D1-2_...	FourWeek_Fliter		1,000,000.00	1.1261	126,059.54	5.92	0.8743	957,600.00	82,000....	7.21	0.6629	0.8216	1.1796	
19	FourW...	zn888_D1-2_...	FourWeek_Fliter		1,000,000.00	0.7936	-206,350.88	-9.69	0.9316	931,600.00	268,98...	25.47	-0.7073	-0.3805	-1.6443	
		汇总(38)	FourWeek_Fliter		19,000,000.00	1.1575	2,992,793.28	7.40	0.9613	18,393,140.00	25,795.56	5.94	1.6149	1.2447	2.7092	169



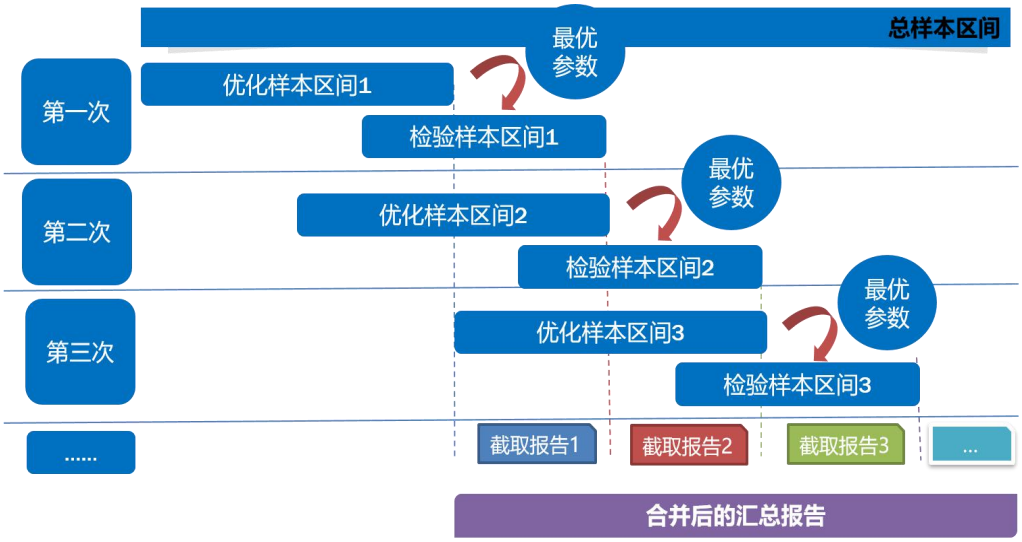
我们可以看到，最初的四周哪怕是在交叉优化的情况下，最后三年也仅仅是维持不亏损，并没有盈利。
而现在我们增加了两个参数之后，最后三年的净值曲线是不断新高。夏普比率也高达 1.61。
这说明滤网是可以让历史测试取得更好的绩效的。

3、更多的参数，更多自由度，拟合度更高，策略未来表现更不确定

回顾我们从最简单的四周规则只有一个四周周期参数，到后来我们增加止盈参数，再增加滤网的均线参数之后，通过三个参数同时优化，我们竟然可以使得原来难以盈利的情况扭转为一个净值不断新高的策略。
这难道说明策略应该越多参数越好吗？优化处理的结果在未来可以复现吗？
其实答案是明显的。
越多的参数，对历史的拟合度就越高，历史测试取得好的业绩的可能就越高。然而未来业绩的落差可能就越大。
那么如何进行优化才能保持历史测试和未来实盘绩效的一致性呢？

4、递进参数优化简要说明

我们可以回顾前面章节我们提到过一种参数优化的方法叫做样本外递进检验。
样本外递进检验是严格意义上的样本外检验。这种优化算法，使用一段历史数据进行参数优化，选择参数，应用参数到优化样本外的检验样本中。不断推进样本递进优化，并提取检验样本的测试结果形成测试报告。
样本外递进检验比起单纯的历史测试，能更真实的反应参数选取对未来绩效的影响。参数优化的重心更加偏向于如何确定参数选取的方法，而不是简单的历史绩效的比较。
下面是样本外递进检验的算法流程图。需要进一步了解的用户可以到 TradeBlazer 官网的帮助中心\ TBQuant 软件使用查看相应章节。



四、策略生成器

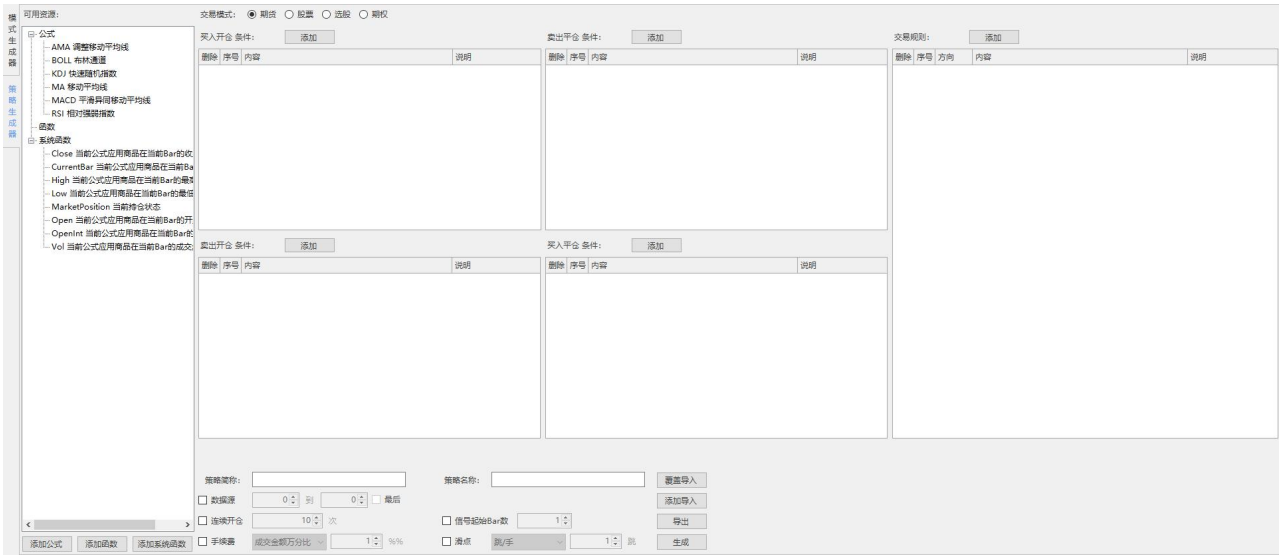
很多读者有非常清晰的交易逻辑，但是短期内还未掌握一种计算机语言，导致量化之路搁浅。TBQuant 充分考虑到客户的多样性，所以推出了一项重磅功能。少量编写代码，然后鼠标操作即可实现策略编写。这项功能就是 TBQuant 的策略生成器。

下面我们使用策略生成器来实现四周规则的基础策略。

4.1、打开策略生成器

我们在工具栏点击生成器的按钮，打开如下界面，在左侧选择策略生成器：

从左到右依次是可用资源、交易条件、策略属性和交易规则。

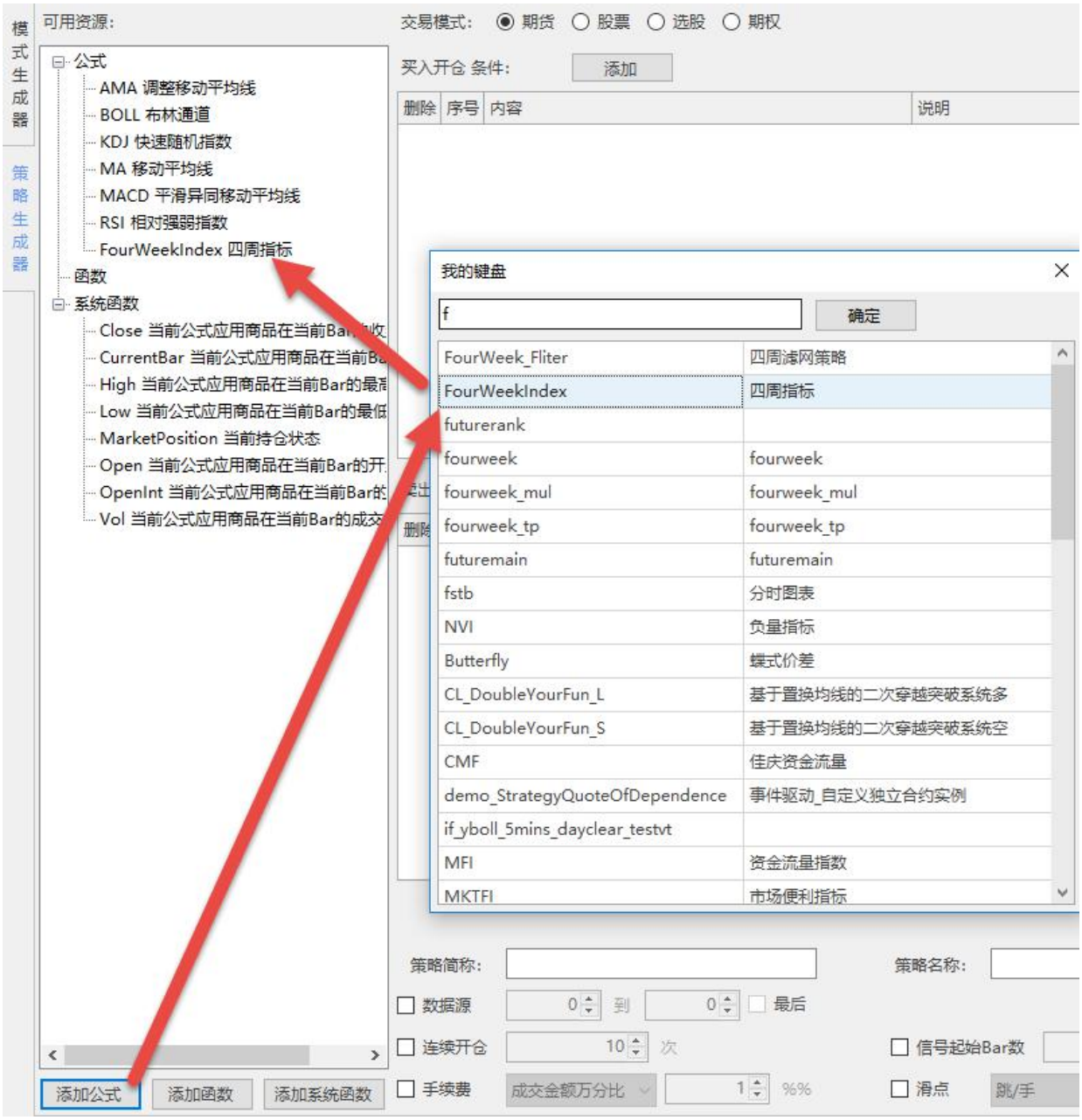


4.2、添加公式和函数

策略生成器的第一步是要设置准备使用的指标和函数资源。我们需要先编辑一个 FourWeekIndex。添加完之后会在可选资源里面看到这个公式。所有函数的返回值和公式当中的 plot 变量都是可以选择的指标。则这两个指标都可以调用。不过在公式界面只显示添加的公式名，不显示具体的指标。具体的指标在条件设置界面显示。

Params

```
  Numeric length(20,10,60,2);   //四周周期参数
Vars
  Numeric highline;        //高点连线
  Numeric lowline;        //低点连线
Events
OnBar(ArrayRef<Integer> indexs)
{
  highline=Highest(High[1],length);
  lowline=Lowest(Low[1],length);
  PlotNumeric("highline",highline);
  PlotNumeric("lowline",lowline);
}
```



4.3、添加条件

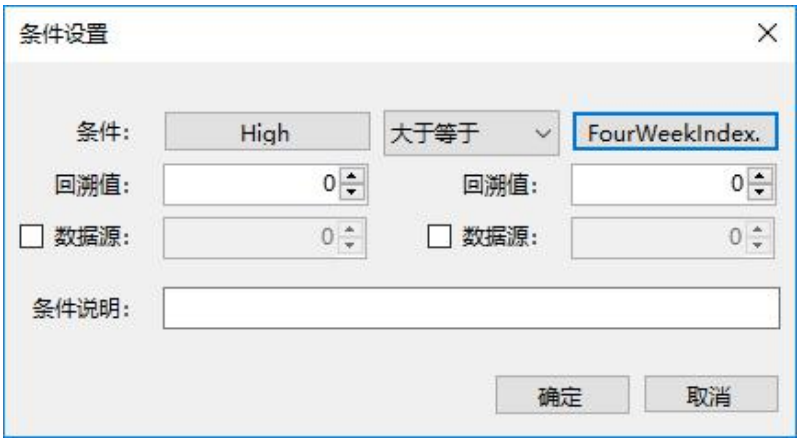
添加条件是设置交易触发条件。我们先设置买入条件，最新价突破四周高点，买入开仓。

1、我们需要选择交易的市场

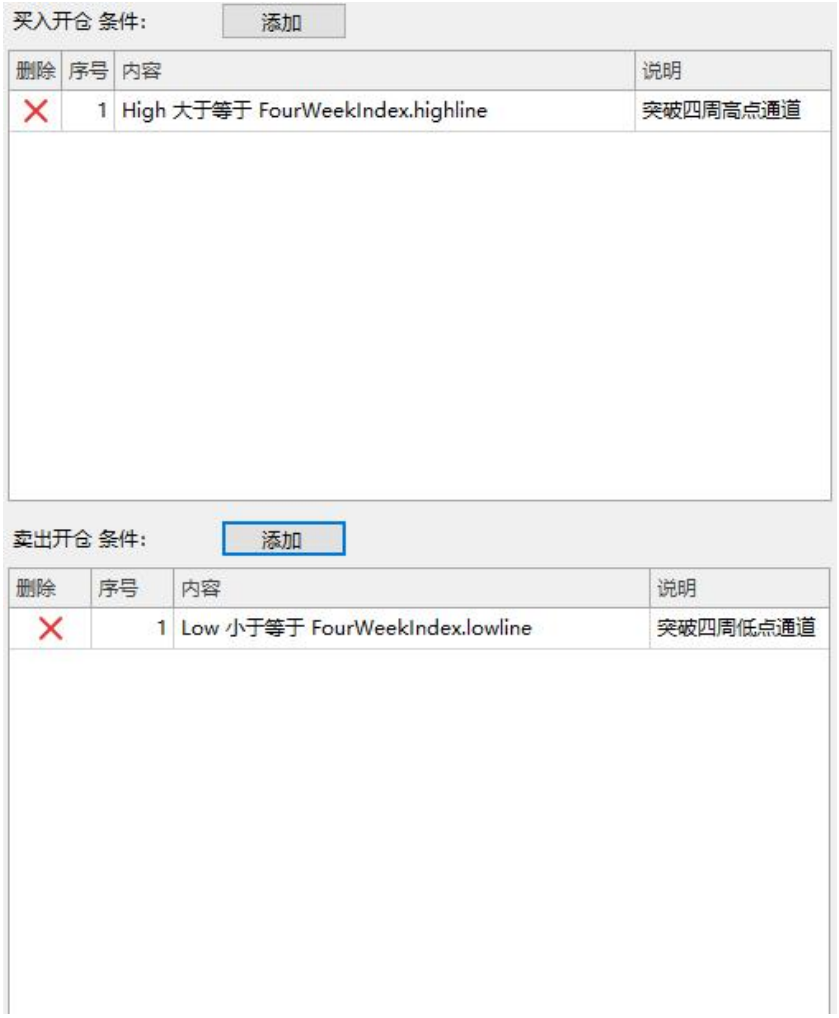
假设选择期货，这时候条件设置有 4 个，买入开仓、卖出平仓、卖出开仓、买入平仓。如果选择股票，有买入和卖出两种。如果选择选股，只有一个选股。如果选择期权，也是有 4 个，买入开仓、卖出平仓、卖出开仓、买入平仓。

2、对买入开仓添加条件

在买入开仓界面，点击添加按钮，弹出条件设置界面。
条件需要用户自己选择指标进行比较生成条件。
我们要实现的是当根 bar 的最高价突破四周高点通道，具体设置如下图所示：
点击指标 1 设置，弹出自设条件界面，从里面选择指标。
我们选择系统函数 High，为指标 1，设置这个指标的回溯值为 0；
点击指标 2 设置，弹出自设条件界面，从里面选择指标。
我们选择 FourWeekIndex 公式下面的指标 highline，为指标 2，设置这个指标的回溯值为 0；



设置完成之后，我们可以看到在买入开仓下面就有这样一个条件。
同样，我们设置一下卖出开仓的条件。

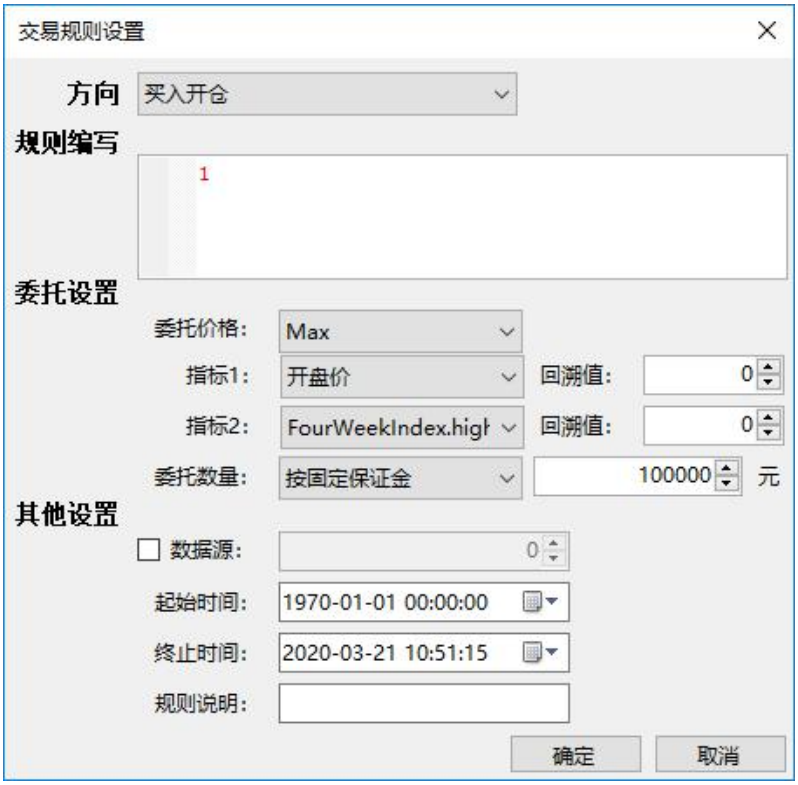


备注：数据源这里不用勾选。TBQuant 的数据源支持多品种叠加，默认是对所有叠加的品种都执行这个条件。如果想值针对具体的某个数据源则可以勾选数据源，进行选择制定。如果对于设置的条件想要删除，直接点击条件所在行的删除标志。

4.4、添加规则

我们设置好条件之后，需要进一步指定满足条件之后的具体交易。这时候需要设置规则。比如四周高点突破买入。

1、交易规则界面



2、选择交易方向

这里可以选择的有四个，我们选择买入开仓。

3、编写交易规则

交易规则的编写是用逻辑运算符把条件组合起来。
支持的逻辑运算符有 and or 还有辅助的（）。
而条件则直接使用条件前面的序号表示。
我们这里条件比较简单，就直接用 1。这代表我们的交易规则就是满足 1 就可以。

4、勾选委托设置

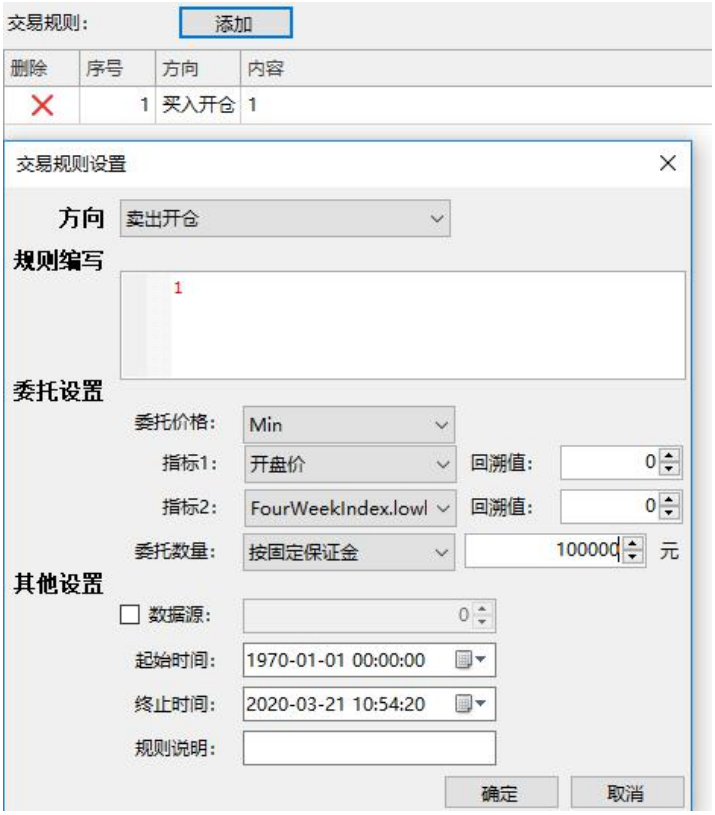
委托设置有委托价格和委托数量的设置。
我们选择 max,然后再确定两个价格为开盘价和 highline。委托数量我们用固定保证金 10 万。

5、设置数据源

数据源主要确定下数据源的起始和终止时间。方便测试和运行。

6、查看和修改交易规则

设置完交易规则之后，我们双击交易规则，则弹出具体设置的窗口，可以查看细节，也可以修改。



4.5、设置策略属性

我们需要设置策略属性。策略名称、策略注释、初始资金、数据源、连续开仓、最大开仓、手续费、滑点等设置。

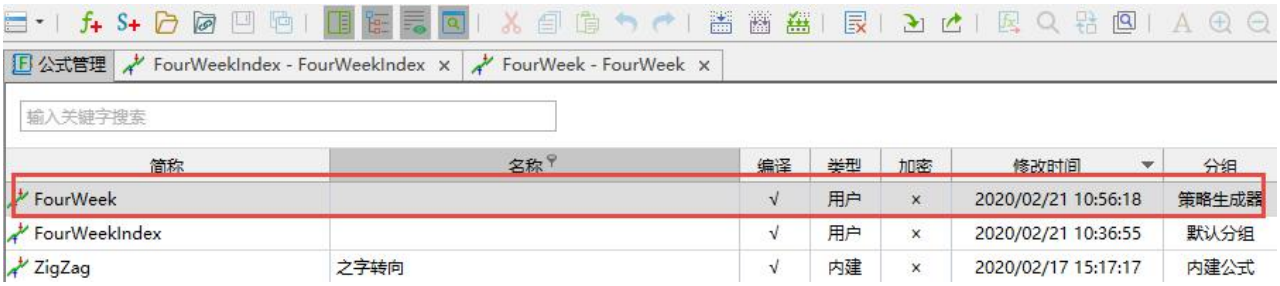
4.6、生成和导入导出

最后一步，我们需要把这些设置生成一个策略。

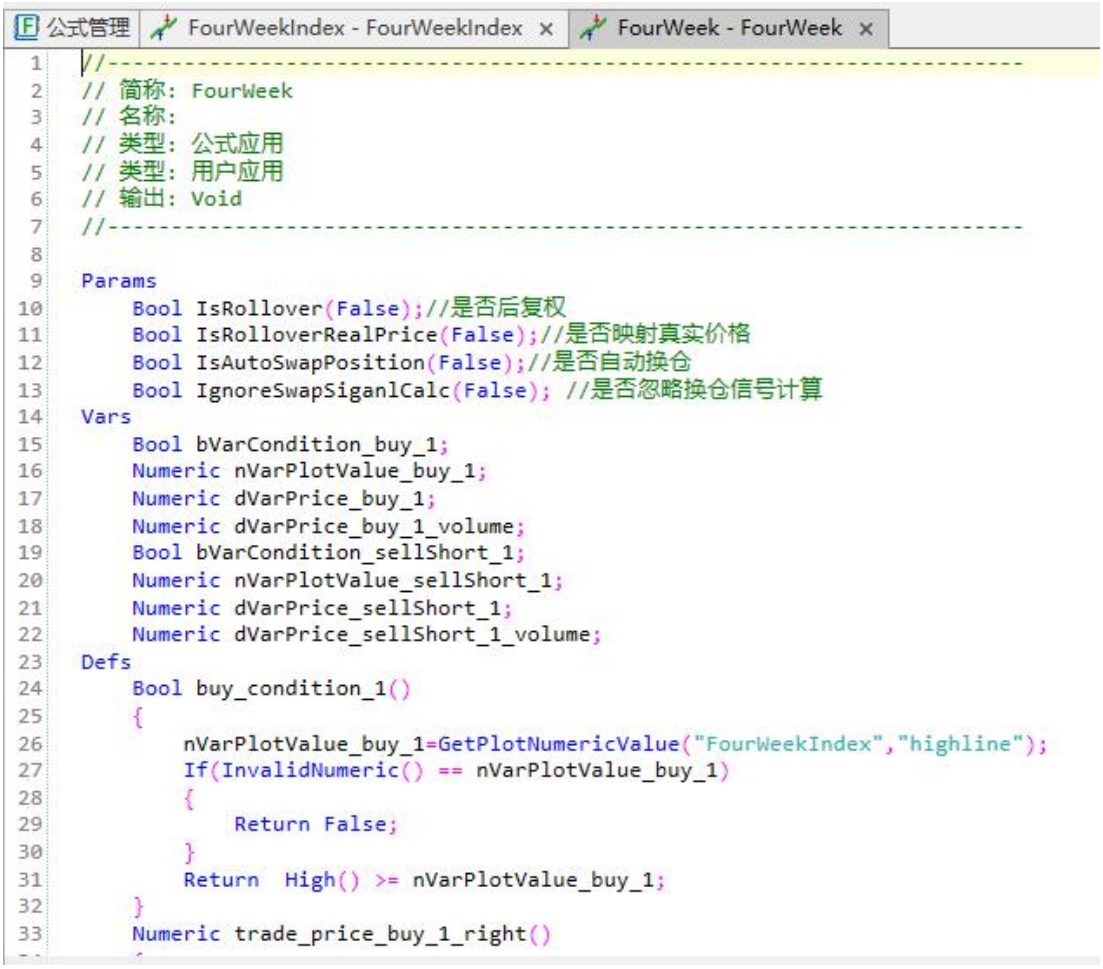
1、生成策略

直接点击生成按钮。

在公式管理器中可以看到对应的公式。

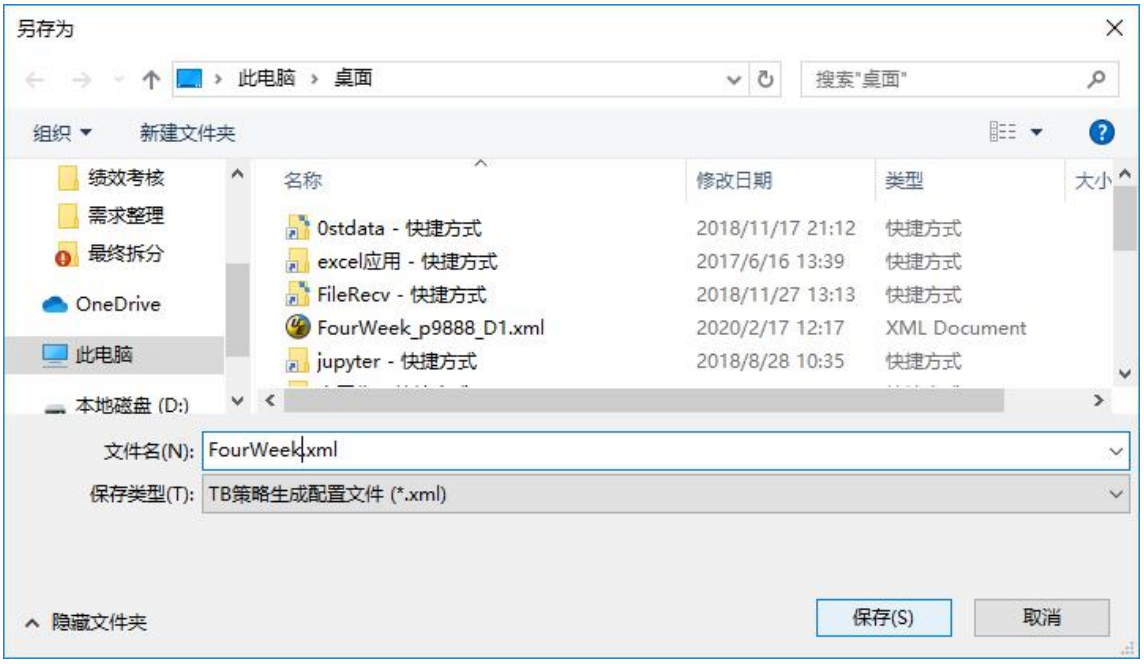


双击公式可以查看代码。



2、导入和导出

为了下次更方便地设置策略，我们可以把这次设置的条件和规则保存下来。我们点击导出按钮，弹出导出的界面框，保存。



下次打开策略生成器，可以直接选择导入。导入有两种：覆盖和增加。覆盖则使用保存的设置，增加是把保存的条件和规则追加到现在的界面中。

3、设为默认

设为默认则下次打开就是这个界面。同时也会保存一个默认的配置文件。

4.7、调用策略

对于我们生成的策略在策略生成器有显示，用户可以调用，但是要注意生成策略依赖的公式也需要同时调用，并且要首先加载。否则生成的策略失去了基础，无法运行。

比如新建一个工作区，加载 FourWeekIndex 和 FourWeek 这两个公式。

FourWeekIndex 必须在 FourWeek 前面，否则 FourWeek 读不到指标数值。

打开 K 线，我们可以看到两个公式运行的交易信号。

五、截面模型的实现

期货市场交易标的相对较少，而股票市场标的众多，如果每一只股票都加入投资组合。管理起来十分不方

便，同时交易者也想实现 alpha 的收益。所以，当交易股票市场的时候，一个绕不开的话题就是选股。那么选股就牵涉到截面的比较。而股票交易的前期准备工作比较多。我们从基础数据、数据类型、函数、截面模型的编写四个部分进行讲解。

5.1、基础数据

股票不仅需要有行情数据，更重要而且数量更为庞大的是基本面的数据。TBL 语言为了方便调用基本面的数据，设计了基础数据的类型。
基础数据从广义上理解是交易所行情之外的其它数据。比如股票的财务报表数据，比如商品的分类数据，比如连续合约的合约切换等信息。
而 TBQuant 不仅系统提供了丰富的基础数据方便用户使用，用户也可以自己定义基础数据。

1、基础数据的读取

1）通过变量定义的方式

用户需要根据基础数据的数据库中的键名来定义一个基础数据的变量。这样的话，读取这个变量就是读取基础数据相应位置的数值。给这个变量赋值就是写入数值到基础数据数据库的相应位置。
变量定义的格式如下：
Dic<数据类型> 变量名（String 键名，bool 是否持久化，String 关联标的），后面两个可以不写
1)键名和关联标的可以从系统一数据中心查询，特别注意要区分大小写。比如"TB_FinanceInd_F014N" 不能改为"tb_FinanceInd_F014N","system",不能改为"System"。
2）如果不持久化，软件重启，这个基础数据就清除了。
3)string 关联标的如果不写，那么就自动关联读取时候的数据图层，如果这个数据图层有基础数据就返回，否则无效值。
比如下面的代码：

```
Dic<Array<Numeric>> pershare("TB_FINANALYSIS_PERSHARE");  //读取每股财务指标"
```

TB_FINANALYSIS_PERSHARE "是 TB 基础数据库中每股财务指标的数组对应的的数据库的键名，后面两个参数 string 关联标的，省略没有给。

2）直接读取数据库

直接读取基础数据数据库的函数是 GetDicValue()；
void GetDicValue(String name, String symbol, Numeric time, 接收基础数据的变量 rValue)
name 是基础数据的键名，symbol 是关联标的，time 是时间，最后一个 rvalue 是接收基础数据的变量。
如果用户对于要读取的基础数据的参数的具体数值不是很清楚，可以点开系统一数据中心进行搜索查看。

2、基础数据的写入

首先如果是系统提供的基础数据，是不允许用户写入的。用户只能对自己定义的基础数据进行写入。用户对自定义基础数据写入也有两种方式。一种是变量赋值，一种是直接写数据库。

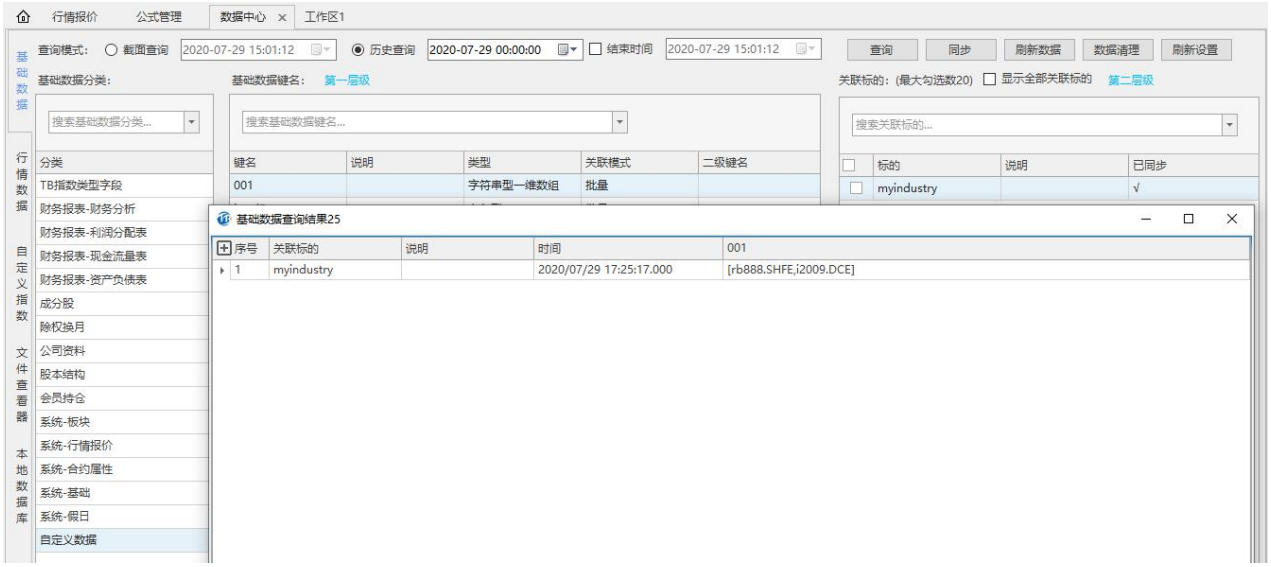
1）通过变量赋值的方式

用户可以自己定义一个基础数据的变量，这个变量通过键名参数的设定对应基础数据数据库的相应位置。然后对这个变量赋值，即相当于给基础数据数据库的相应位置写入数值。
比如下面的 demo:

```
Params
Vars
  Dic<Bool> boolDs("boolDs",False);
  Dic<Integer> integerDs("integerDs",False);
  Dic<Numeric> numericDs("numericDs",False);
  Dic<String> stringDs("stringDs",True);
  Integer mybarcount;
Events
OnBar(ArrayRef<Integer> indexs)
{
  mybarCount = CurrentBar();
  if(barCount % 2 == 0)
  {
    boolDs[0] = True;
    integerDs[0] = barCount;
    numericDs[0] = barCount / 10;
    stringDs[0] = "str_" + Text(barCount);
  }
}
```

2）直接写数据库

bool SetDicValue(String name, String symbol, Numeric time, Numeric rValue, Bool isPersistence = false)
name 是基础数据的键名，symbol 是关联标的，time 是时间，最后一个 rvalue 是写入基础数据的变量。
特别的如果用户想定义一个不受关联标的限制的基础数据，也就是可以在所有数据图层都能用的基础数据。那么 symbol 这一项最好写成是一个用户自定义的字符串。
比如，SetDicValue("001","myindustry",SystemDateTime(),subarray,True);subarray 的内容是一个字符串数组。
subarray(["rb888.SHFE","i2009.DCE"])。
那么在基础数据中心，我们可以查询到如下结果。



3、基础数据键值的参数传递

1）定义

为了方便的调用基础数据，省去用户抄写基础数据键值的麻烦，TBQuant 设计了这个键值的参数传递。在编写代码时，只需要把变量定义为如下格式：

```
Params
    Array<String> pabcDicKey([]);    //手动选择基础数据
```

其中 Array<String> p*** DicKey([])是固定格式，不能做出任何修改。星号部分用户可自定义。

2）具体案例

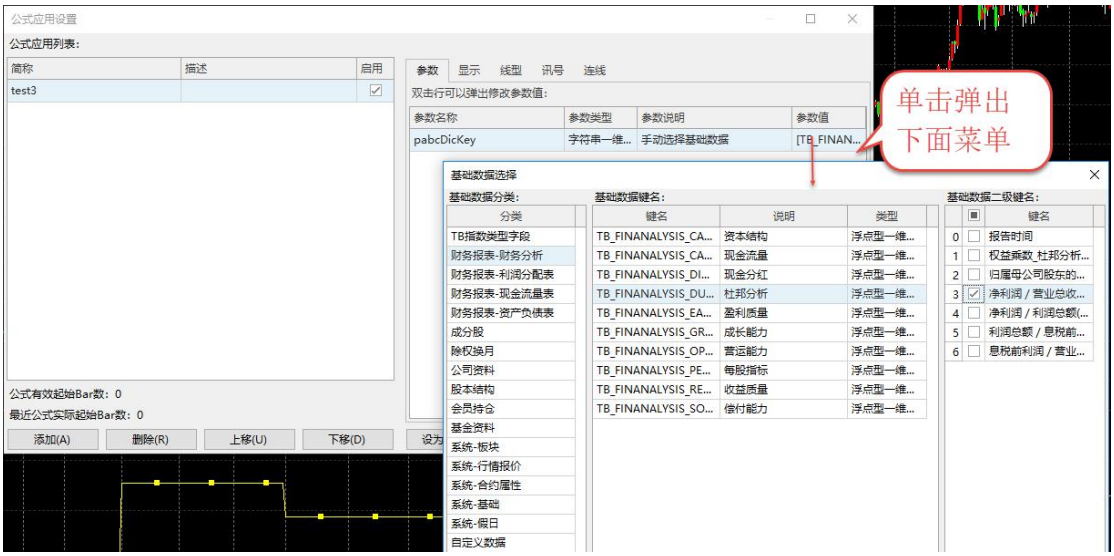
我们用一个简单的例子来看下，这个如何使用。

1、编写一个公式 test3。代码如下，设置显示方式为子图。

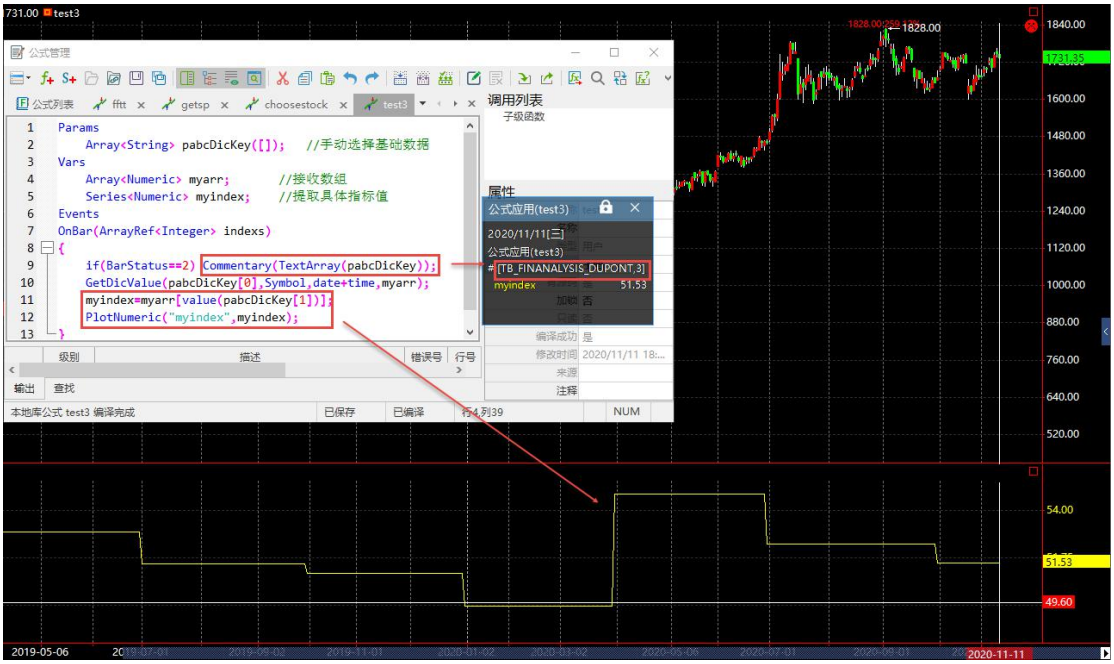
```
Vars
    Array<Numeric> myarr;        //接收数组
    Series<Numeric> myindex;     //提取具体指标值
Events
OnBar(ArrayRef<Integer> indexs)
{
    if(BarStatus==2) Commentary(TextArray(pabcDicKey));
    GetDicValue(pabcDicKey[0],Symbol,date+time,myarr);
    myindex=myarr[value(pabcDicKey[1])];
    PlotNumeric("myindex",myindex);
}
```

2、加载到一个 K 线图上，设置公式参数。

从弹出的菜单中，我们选择一个财务指标。



3、查看结果



5.2、数据类型

截面的处理数据类型提出了新的需求，基本的数据类型已经不能满足截面的需求。所以基本的数据类型需要扩展，尤其是对数组的支持。

1、TBL 目前支持的数据类型

TBL 支持的数据类型有基本类型和扩展类型。总共可细分为 6 类，可以划分为 4 个级别。

数据类型	数据类型的级别	数据类型的内容
基本类型：普通类型	0	Numeric/String/Bool/Integer
基本类型：内嵌类型	0	Position, Order, Fill, Tick, Bar, CodeProperty

容器类型一级	1	Array,array<array>
容器类型二级	2	Dic,map,series
类型引用	3	Ref
类型修饰	3	Global

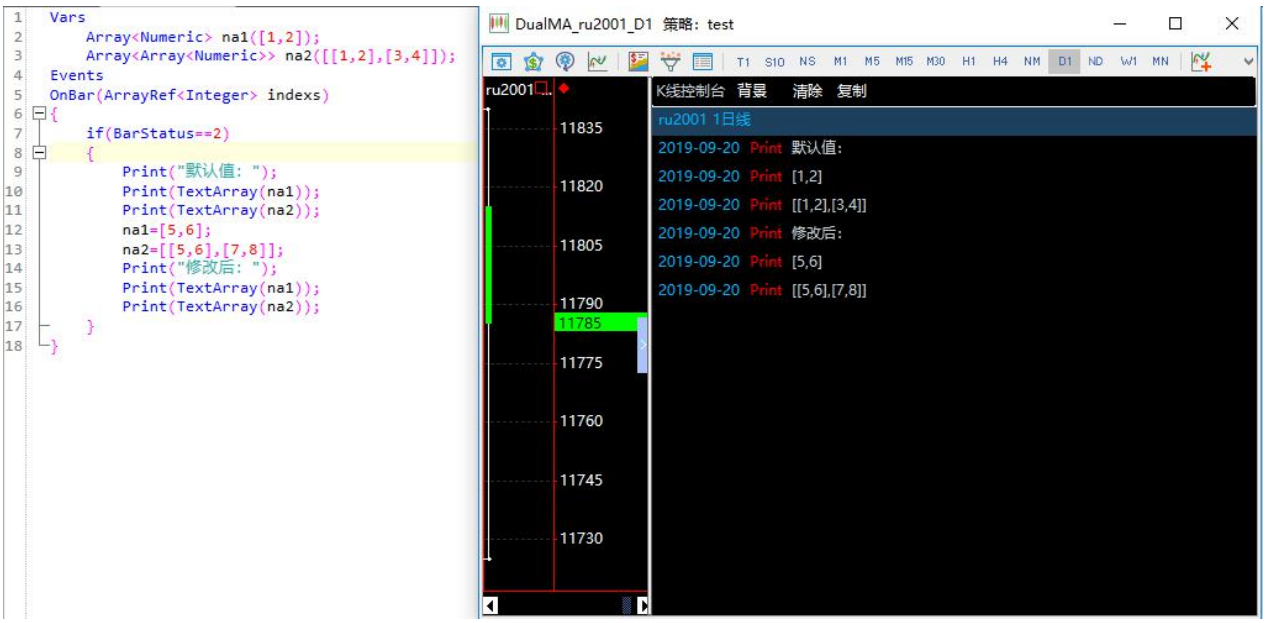
2、数组类型

在 TBQuant 里面我们支持一维数组和二维数组对基础数据类型的容器扩展。

1)数组的定义和赋值

比如下面的例子，定义了一个一维数组和一个全局的二维数组。并且给其中的元素进行了赋值。数组参数的默认值可以直接使用列表形式赋值。

```
Vars
   Array<Numeric> na1([1,2]);
   Array<Array<Numeric>> na2([[1,2],[3,4]]);
Events
OnBar(ArrayRef<Integer> indexs)
{
   if(BarStatus==2)
   {
      Print("默认值: ");
      Print(TextArray(na1));
      Print(TextArray(na2));
      na1=[5,6];
      na2=[[5,6],[7,8]];
      Print("修改后: ");
      Print(TextArray(na1));
      Print(TextArray(na2));
   }
}
```



注意：数组的定义不需要指定数组的大小，数组大小会自动根据代码对数据元素的使用而扩容。

2)数组的运算

数组的运算可以由函数来实现。而基本的四则运算则可以直接使用运算符实现。

3)数组的相关函数

ArrayClear：数组的全部删除。
ArrayClear：删除二维数组的全部内容。
ArrayCompare：对两个数组进行比较。
ArrayCompare：对两个二维数组进行比较。
ArrayCopy：复制数组的内容。
ArrayCopy：复制二维数组内容。
ArrayEqual：检验两个数组是否相等。

ArrayEqual：检验两个二维数组是否相等。
ArrayErase：数组的元素删除。
ArrayErase：二维数组的指定元素删除。
ArrayInsert：数组的单个数据插入。
ArrayInsert：二维数组插入单个元素。
ArrayInsertRange：数组的多个数据插入。
ArrayInsertRange：二维数组插入多个元素。
ArraySort：对数组进行排序。
ArraySort：二维数组排序。
ArraySwap：交换数组的内容。
ArraySwap：交换二维数组的内容。
GetArraySize：获取数组的大小。
Na1Sort：一维数组排序带下标。
Na1Sort2：一维数组排序带下标，原数组不变。
Na1Max：一维数组的最大值。
Na1Min：一维数组的最小值。
Na2Add：二维数组的加法。
Na2Dev：二维数组的减法。
Na2Mul：二维数组的乘法。
Na2Inv：二维数组的求逆。
Na2Trans：二维数组的转置。
Na2Abs：二维数组的绝对值。
Na2Sum：二维数组的元素求和。
SetArraySize：设置数组的大小和初始值。
SetArraySize：设置二维数组的大小和初始值。
SortIds：一维数组排序带双重下标。

3、数组的行列操作


```
Vars
    Array<Array<Numeric>> arr([[1,2],[3,4]]);  //二维数组赋值
Events
OnBar(ArrayRef<Integer> indexs)
```

```
{
    if(BarStatus==2)
    {
        commentary("整个数组="+TextArray(arr));
        commentary("数组第一行="+TextArray(arr[0]));  //行下标从 0 开始
        commentary("数组第一列="+TextArray(arr[:,0])); //列下标从 0 开始
    }
}
```

5.3、股票多因子函数

多因子模型是股票交易的一个常用基本模型。所以多因子函数的支持也是 TBL 语言的一个基本功能。
在数学统计的函数里面，系统支持了因子标准化、因子提纯、多元线性回归等函数来实现多因子模型。

5.4、神奇公式模型的编写

在做好了上面数据和函数的准备之后，我们开始编写一个截面模型，神奇公式。

1、神奇公式的简介

神奇公式是格林布拉特提出的选股的经典法则，它综合了盈利和估值两个方面。选取盈利好估值低的股票纳入股票池。我们在本例当中使用 ROE 衡量盈利，PE 衡量估值。
特别注意：
1、本案例策略单元的数据源为 hs300 的个股+if 股指期货，最后一个品种为股指可以对冲。所以股票的数据源序号为 0 到 DataCount-2。股指的为 DataCount-1。
2、本案例使用后复权的数据，所以计算 PE 需要还原真实价格。

2、代码的编写

2.1 基础数据的获取

PE 和 ROE 在 TBQuant 里面属于来自财务报表的基本面数据，这些数据都存储于基础数据的数据库中，客户端需要调用的时候只需要在公式代码中添加如下语句，即可调用。

```
Dic<Array<Numeric>> pershare("TB_FINANALYSIS_PERSHARE");  //读取每股财务指标
Dic<Array<Numeric>> earning("TB_FINANALYSIS_EARNINGSQUALITY");  //读取盈利质量财务指标
Series<Numeric> me;              //财务指标每股收益
Series<Numeric> roe;             //财务指标 roe
```

FeData 读取了整张财务数据的一维数组，其中每股收益是数组的第一个元素，roe 是数组的第五个元素。

2.2 PE/ROE 指标的排序

我们分别对 PE 和 ROE 进行排序，PE 希望越小越好，ROE 越大越好。分别排序之后我们对两个排名进行加权，再综合排序进行选择股票。

最后的综合排名存储在 id32 的数组当中。

```
SortIds(na1pe,id11,id12,0,DataCount-2,True);      //PE 越小越好
SortIds(na1roe,id21,id22,0,DataCount-2,False);    //ROE 越大越好
for i=0 to DataCount-2
{
  id[i]=id12[i]*pe_rate+id22[i]*roe_rate;
}
SortIds(id,id31,id32,0,DataCount-2,True);       //综合指标靠前
```

2.3 选择排名靠前的股票进行买入

首先是选股节奏的确立是通过如下语句，每周四和周一才会选股。

```
//控制选股节奏
If(Weekday<>4 And Weekday<>1)
  Return;
```

其次换仓的时候，是先平掉历史仓位中被淘汰的，再对本次选入的股票调整持仓。

最后，如果需要对冲股指，可以在股票池中加入股指期货，启动后面的代码进行卖出操作。

3、完整代码

完整代码如下：

```
Params
  Numeric pe_rate(0.8);          //pe 指标的权重
  Numeric roe_rate(0.2);          //roe 指标的权重
  Numeric buy_rate(60);          //选取买入股票的排名分位数/支数
  Numeric money(5000000);        //单支股票的买入市值
Vars
  Dic<Array<Numeric>> pershare("TB_FINANALYSIS_PERSHARE");  //读取每股财务指标
  Dic<Array<Numeric>> earning("TB_FINANALYSIS_EARNINGSQUALITY");  //读取盈利质量财务指标
  Series<Numeric> me;              //财务指标每股收益
  Series<Numeric> roe;             //财务指标 roe
  Global Array<Numeric> na1pe;        //财务指标 PE 的数组
  Global Array<Numeric> na1roe;       //财务指标 ROE 的数组
  Global Numeric back(1);            //财务指标的回溯周期
  Global Array<Integer> id11;          //排序下标数组
```

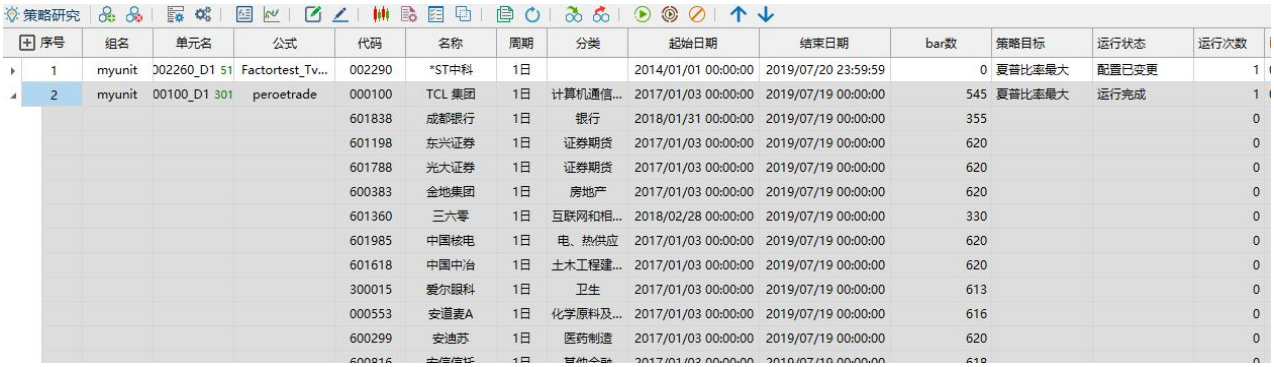
```
  Global Array<Integer> id12;          //排序下标数组
  Global Array<Integer> id21;          //排序下标数组
  Global Array<Integer> id22;          //排序下标数组
  Global Array<Integer> id31;          //排序下标数组
  Global Array<Integer> id32;          //排序下标数组
  Global Array<Numeric> id;            //排序下标数组
  Global Numeric n;              //买入股票的支数
  Global Numeric i;             //循环变量
  Numeric lots;                //下单手数
Defs
  Integer LogFile(String str)
  {
    FileAppend("d:\\my"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
  }
Events
  OnInit()
  {
    LogFile("init......");
    //=========除权换月相关设置==============
    AddDataFlag(Enum_Data_RolloverBackWard());  //设置后复权
    AddDataFlag(Enum_Data_RolloverRealPrice());  //设置映射真实价格
    AddDataFlag(Enum_Data_AutoSwapPosition());  //设置自动换仓
    AddDataFlag(Enum_Data_IgnoreSwapSignalCalc());  //设置忽略换仓信号计算

  }
  onBar(ArrayRef<Integer> indexs)
  {
  //读取财务指标
  Range[i=0:DataCount-2]
  {
    if(pershare[0][4]<>InvalidNumeric)
      me=pershare[0][4];
    if(earning[0][6]<>InvalidNumeric)
      roe=earning[0][6];
    Commentary("me="+Text(me));
    Commentary("roe="+Text(roe));
    //LogFile(DateToString(Date)+"_"+Text(i)+"_"+Text(me)+"_"+Text(roe));
    nalpe[i]=Close[back]/Rollover/me[back];  //从后复权取得真实价格，然后再计算 PE
    nalroe[i]=roe[back];
  }
  //控制选股节奏
  If(Weekday<>4 And Weekday<>1)
    Return;
```

```
//对财务指标进行排序选股
for i=0 to DataCount-2  //对下标数组进行赋值
{
  id11[i]=i+1;
  id12[i]=i+1;
  id21[i]=i+1;
  id22[i]=i+1;
  id31[i]=i;
  id32[i]=i;
}
SortIds(na1pe,id11,id12,0,DataCount-2,True);      //PE 越小越好
SortIds(na1roe,id21,id22,0,DataCount-2,False);    //ROE 越大越好
for i=0 to DataCount-2
{
  id[i]=id12[i]*pe_rate+id22[i]*roe_rate;
}
SortIds(id,id31,id32,0,DataCount-2,True);        //综合指标靠前
//选择前面 n 支股票进行买入，同时对冲股指
if(buy_rate<1)
  n=IntPart(DataCount*buy_rate);
Else
  n=buy_rate;
//没有选中的，平掉历史持仓
for i=n to DataCount-2
{
  if(Data[id31[i]].MarketPosition==1)
    Data[id31[i]].Sell(0,Data[id31[i]].Open);
}
//选中的，没有持仓的进行建仓
for i=0 to n-1
{
   Data[id31[i]].lots=IntPart(money/(Data[id31[i]].Open*Data[id31[i]].BigPointValue*Data[id
31[i]].ContractUnit)/100)*100; //股数为 100 的整数倍
   If(Data[id31[i]].MarketPosition<>1)
   {
     Data[id31[i]].Buy(Data[id31[i]].lots,Data[id31[i]].Open);
   }
   Else
   {
     If(Data[id31[i]].lots<Data[id31[i]].CurrentContracts)
        Data[id31[i]].Sell(Data[id31[i]].CurrentContracts-Data[id31[i]].lots,Data[id31[i]].
Open);
     If(Data[id31[i]].lots>Data[id31[i]].CurrentContracts)
        Data[id31[i]].Buy(Data[id31[i]].lots-Data[id31[i]].CurrentContracts,Data[id31[i]].O
```

```
pen);
        }
    }
    /* //股指期货对冲操作
    Range[DataCount-1:DataCount-1]
    {
        lots=IntPart(n*money/(Open*BigPointValue*ContractUnit));
        BuyToCover(0,Open);
        SellShort(lots,Open);
    } */
}
```

4、具体应用

具体应用当中，我们选取了 HS300 的个股作为股票池，并添加了股指期货作为可选的空头配置。
交易单元如图设置



序号	组名	单元名	公式	代码	名称	周期	分类	起始日期	结束日期	bar数	策略目标	运行状态	运行次数
1	myunit	002260_D1 51	Factortest_Tv...	002290	*ST中科	1日		2014/01/01 00:00:00	2019/07/20 23:59:59	0	夏普比率最大	配置已变更	1
2	myunit	00100_D1 301	peroetrade	000100	TCL 集团	1日	计算机通信...	2017/01/03 00:00:00	2019/07/19 00:00:00	545	夏普比率最大	运行完成	1
				601838	成都银行	1日	银行	2018/01/31 00:00:00	2019/07/19 00:00:00	355			0
				601198	东兴证券	1日	证券期货	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0
				601788	光大证券	1日	证券期货	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0
				600383	金地集团	1日	房地产	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0
				601360	三六零	1日	互联网和相...	2018/02/28 00:00:00	2019/07/19 00:00:00	330			0
				601985	中国核电	1日	电、热供应	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0
				601618	中国中冶	1日	土木工程建...	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0
				300015	爱尔眼科	1日	卫生	2017/01/03 00:00:00	2019/07/19 00:00:00	613			0
				000553	安道麦A	1日	化学原料及...	2017/01/03 00:00:00	2019/07/19 00:00:00	616			0
				600299	安迪苏	1日	医药制造	2017/01/03 00:00:00	2019/07/19 00:00:00	620			0

运行结果：
运行交易单元，右键测试报告，可以看到盈亏曲线图。

六、套利宝

高频交易以稳定的盈利令人羡慕，然而很多交易者缺乏相应的平台而不能很方便地实现策略的研发。
TBQuant 推出了事件驱动机制，解决了这一难题。本章我们来讲解一个经典的价差套利如何用 TBL 实现。

6.1、价差套利的基本思路

本章节的价差套利的交易思路是很简洁的，就是价差低了买入，价差高了卖出。一般一个程序只实现一个方向的交易。
比如买入开仓这个价差等待卖出平仓，或者卖出开仓这个价差等待买入平仓。
然而价差套利代码实现上却需要考虑诸多细节。
比如行情的接收、数据和参数的检查、委托单的状态处理，实时仓位的跟踪和异常的处理等等。

6.2、主要代码讲解

1、算法的主要流程

价差交易的算法大致分为四个阶段，初始化阶段、准备阶段、交易阶段和委托后续处理阶段。

1.1 在初始化阶段（oninit）完成了 3 个事情

1）行情的接收。订阅了 tick 级别的行情数据。
2）委托单的订阅。通过 A_SubscribeTradeByCreateId 函数订阅了所有委托单。
3）确定公式参数的正确性。通过 CheckParams 函数实现。

1.2 在准备阶段(onready)完成了 1 件事情

确定初始仓位是多少。初始化或者读取之前写在数据库中的仓位。

1.3 在交易阶段（onbar）是程序的核心部分。它完成了行情接收和委托单的处理

1）接收行情并检查数据是否准备好。
2）开仓交易。它通过 RunSpreadOrderEntry 函数实现。
3）平仓交易。它通过 RunSpreadOrderExit 函数实现。

1.4 后续委托处理阶段（onorder）是算法的一个重要阶段

因为高频交易涉及到盘口的流动性和实际成交的多变，委托单的后续处理是必不可缺的。
1）监控开仓单 OnNotifyOrder(ord, curEntryStatus)。
2）监控平仓单 OnNotifyOrder(ord, curExitStatus)。

2、重要函数的介绍

2.1 初始化和准备阶段的相关函数

这个阶段有 5 个函数。

1) bool CheckParams()
检查参数设置的是否合理。基本原则是买的价差要低于卖的价差。
2) bool CheckTickData(TickRef tickData)
检查接收的 Tick 数据是否准备好。主要检查最新价和成交量是否大于 0。
3) Numeric CalcSpreadPrice(Numeric p1, Numeric p2)
计算两个合约的价差。最简单的减法。
4) Integer CalcSendOrderPrice(Bool entry,NumericRef myPrice)
计算委托单的发单价格。
买入价格 =合约 A 的卖一价*合约 A 的单位- 合约 B 的买一价*合约 B 的单位。
CalcSpreadPrice(lastTickData[0].bidask1.askP*SpreadUnitA,lastTickData[1].bidask1.bidP*SpreadUnitB)
卖出价格 = 合约 A 的买一价*合约 A 的单位- 合约 B 的卖一价*合约 B 的单位。
CalcSpreadPrice(lastTickData[0].bidask1.bidP*SpreadUnitA,lastTickData[1].bidask1.askP*SpreadUnitB)
5) Numeric GetSendedOrderPrice(Bool entry,Bool pairB)
获取合约的委托单价格。委托单分为四种类型，分别存储在下面的四个全局数组里面。

```
Global Array<Integer> entryPairAOrderIds;  // 缓存的 Data0 开仓 OrderIds
Global Array<Integer> entryPairBOrderIds;  // 缓存的 Data1 开仓 OrderIds
Global Array<Integer> exitPairAOrderIds;  // 缓存的 Data0 平仓 OrderIds
Global Array<Integer> exitPairBOrderIds;  // 缓存的 Data1 平仓 OrderIds
```

2.2 开仓函数

开仓有六个函数。其中最后一个 RunSpreadOrderEntry 是主体函数。前面 5 个函数被最后一个函数调用。需要注意的是开仓先开 B 这一个合约 data1，然后再开 A 这个合约 data0。所以对 data1 的处理要稍微复杂一些，data0 的处理相对简单。Data0 这一腿成交后，套利合约就完成了一对开仓。
开仓状态(curEntryStatus)总共有 8 种：

状态序号	所需操作	函数名称
0	开启开仓交易,开仓 data1	OnEntryStatusNone
1/3/5/7	等待委托单到交易所	无
2	是否需要撤单重发 data1	OnEntryStatusSendedData1
4	Data1 开仓成交，开仓 data0	OnEntryStatusFilledData1
6	是否需要撤单重发 data0	OnEntryStatusSendedData0
8	更改数据库的仓位状态	OnEntryStatusFilledData0

1) Integer OnEntryStatusNone()
当开仓状态是 0 时，也就是套利单交易还未开启时，启用这个函数。这个函数把第二腿合约 data1 的开仓委托单先发送交易所。
2) Integer OnEntryStatusSendedData1()
时刻监控 data1 是否撤单重发，行情数据错误或者行情已经偏离委托太远，都需要撤单。
3) Integer OnEntryStatusFilledData1()
当 data1 这一腿开仓委托已成交，则发送 data0 这一腿的开仓。
4) Integer OnEntryStatusSendedData0()
时刻监控 data0 是否需要撤单重发，当行情已经偏离委托太远，就需要撤单。

5) Integer OnEntryStatusFilledData0()
当 data0 这一腿开仓也成交了，那么就更改数据库的仓位数量。
6) Integer RunSpreadOrderEntry()
这个是开仓交易的主函数，根据不同的开仓状态调用前面 5 个函数。

2.3 平仓函数

平仓函数也有 6 个函数，结构上基本和开仓函数的一致。第六个函数 RunSpreadOrderExit 是主函数，根据不同的平仓状态来调用前 5 个函数来处理平仓。
开仓状态(curExitStatus)总共有 8 种：

状态序号	所需操作	函数名称
0	开启平仓交易，平仓 data1	OnExitStatusNone
1/3/5/7	等待委托单到交易所	无
2	是否需要撤单重发 data1	OnExitStatusSendedData1
4	Data1 平仓成交，平仓 data0	OnExitStatusFilledData1
6	是否需要撤单重发 data0	OnExitStatusSendedData0
8	更改数据库的仓位状态	OnExitStatusFilledData0

1) Integer OnExitStatusNone()
当平仓状态是 0 时，也就是套利单平仓还未开启时，启用这个函数。这个函数把第二腿合约 data1 的平仓委托单先发送交易所。
2) Integer OnExitStatusSendedData1()
时刻监控 data1 是否撤单重发，行情数据错误或者行情已经偏离委托太远，都需要撤单。
3) Integer OnExitStatusFilledData1()
当 data1 这一腿平仓委托已成交，则发送 data0 这一腿的平仓。
4) Integer OnExitStatusSendedData0()
时刻监控 data0 是否需要撤单重发，当行情已经偏离委托太远，就需要撤单。
5) Integer OnExitStatusFilledData0()
当 data0 这一腿平仓也成交了，那么就更改数据库的仓位数量。
6) Integer RunSpreadOrderExit()
这个是平仓交易的主函数，根据不同的平仓状态调用前面 5 个函数。

2.4 委托单处理函数

套利委托单的处理函数有 5 个，其中第 5 个函数 OnNotifyOrder 不仅是委托单处理的主函数，也是开仓和平仓主函数的主函数。
也就是是 OnNotifyOrder 这个函数控制着所有的委托单处理。它调用下面的前 4 个函数，也调用开仓和平仓处理的主函数。
套利委托单的状态总共有 8 种。

状态序号	状态含义	函数名称
1	委托 B 发送中	OnNotifySending(ord, True)
2	委托 B 发送成功	OnNotifySended(ord, True)
3	委托 B 撤单中	OnNotifyCanceling(ord, True)

4/8	委托 A 或委托 B 已经成交	RunSpreadOrderEntry(); RunSpreadOrderExit();
5	委托 A 发送中	OnNotifySending(ord,False)
6	委托 A 发送成功	OnNotifySended(ord,False)
7	委托 A 撤单中	OnNotifyCanceling(ord,False)

1) Bool IsMyOrder(OrderRef ord,ArrayRef<Integer> orderIds)
检查委托单是否是这个套利交易的委托单
2) Integer OnNotifySending(OrderRef ord,bool pairB)
处理委托发送中的状态
3) Integer OnNotifySended(OrderRef ord,bool pairB)
处理委托发送成功的状态
4) Integer OnNotifyCanceling(OrderRef ord,bool pairB)
处理委托撤单中的状态
5) Integer OnNotifyOrder(OrderRef ord,Integer myStatus)
委托单处理的主函数
在理解上面套利委托单处理时，也需要知道 TBL 语言对委托单的状态标识。
委托单总共有 7 种状态，通常分为完成和未完成两大类。
其中 enum_deleted,enum_filled,enum_canceled 是完成的，其它是未完成的。

枚举值	基本含义	对应整数值
Enum_delcare	申报中	1
Enum_declared	申报完成	2
Enum_deleted	交易所废单	3
Enum_fillpart	部分成交	4
Enum_filled	全部成交	5
Enum_canceled	已经撤销	6
Enum_canceling	撤销中	7

6.3、完整代码

```
//--------------------------------------------------------------------
// 简称: SpreadTrader
// 名称: 套利宝
// 类别: 公式应用
// 类型: 用户应用
// 输出: Void
//--------------------------------------------------------------------
Params
  String SpreadSection("SpreadTrader");  // 用于数据库存储的 Section 设置
  String SymbolA("y2005.DCE");       // DataO 商品信息，为盘口活跃品种
```

```
  String SymbolB("y2009.DCE");       // Data1 商品信息，为盘口不活跃品种
  Bool ResetPosition(False);         // 重设持仓数据
  Integer InitPosition(0);          // 初始持仓值
  Integer SpreadUnitA(1);           // Data0 的价差计算单位
  Integer SpreadUnitB(1);           // Data1 的价差计算单位
  Bool SpreadLong(True);            // True 对价差做多，False 对价差做空
  Numeric EntryLine(0);            // 价差开仓位置
  Numeric ExitLine(100);            // 价差平仓位置
  Integer MaxPosition(1);           // 最大持仓手数
  Integer DeleteOrderSet(2);        // 挂单后盘口变化点数设置(1-N)，大于等于这个点数才撤单，
设置太小会导致撤单次数太多。
  Integer OrderOffset(0);           // 挂单成交后补单的委托价格偏移点数
Vars
  Global Integer i;               // 函数内使用临时变量
  Global String tmpStr;             // 函数内使用临时变量
  Global Integer tmpInteger;         // 函数内使用临时变量
  Global Numeric tmpNumeric;         // 函数内使用临时变量
  Global Tick tmpTickData;          // 函数内使用临时变量
  Global Order tmpOrder;            // 函数内使用临时变量
  Global Bool needCancelOrder;       // 是否需要撤单的临时变量

  Global Bool dataReady;            // 检查数据是否正确的标识
  Global Bool paramReady;           // 检查参数是否正确的标识

  Global Numeric priceTickA;         // Data0 的价格变动单位
  Global Numeric priceTickB;         // Data1 的价格变动单位

  Global Array<Tick> lastTickData;     // 缓存的最新 Tick 数据
  Global Array<bool> tickDataValid;     // Tick 数据是否有效标识

  Global Numeric spreadBidPrice;       // 价差的买盘价
  Global Numeric spreadAskPrice;       // 价差的卖盘价
  Global Numeric orderPrice;          // 委托单价格
  Global Numeric upPrice;            // 价格上限
  Global Numeric dnPrice;            // 价格下限

  Global String curPosKey;          // 根据商品信息生成的 Key，使用该 Key 来保存持仓数据到数据库
中
  Global Integer curPosition(0);       // 当前持仓手数，正数为多头持仓，负数为空头持仓
  Global Integer curEntryStatus(0);     // 当前套利下单的开仓处理状态
  Global Integer curExitStatus(0);     // 当前套利下单的平仓处理状态
                        // 0   -   初始状态
                        // 1   -   Data1 挂单委托发送中
                        // 2   -   Data1 挂单委托成功
```

```
                        // 3   -   Data1 挂单委托撤单中
                        // 4   -   Data1 挂单委托成交
                        // 5   -   Data0 直接委托发送中
                        // 6   -   Data0 委托发送成功
                        // 7   -    Data0 委托撤单中
                        // 8   -    Data0 委托成交
                        // -99    -    出错状态，需人工干预

  Global Array<Integer> entryPairAOrderIds;   // 缓存的 Data0 开仓 OrderIds
  Global Array<Integer> entryPairBOrderIds;   // 缓存的 Data1 开仓 OrderIds
  Global Array<Integer> exitPairAOrderIds;   // 缓存的 Data0 平仓 OrderIds
  Global Array<Integer> exitPairBOrderIds;   // 缓存的 Data1 平仓 OrderIds
Defs
  bool CheckParams()
  {
    If(SpreadLong)
    {
      If(EntryLine >= ExitLine) Return False;
    }Else
    {
      If(EntryLine <= ExitLine) Return False;
    }
    Return True;
  }
  bool CheckTickData(TickRef tickData)
  {
    /*
    Commentary("symbol="+tickData.symbol);                  //合约名
    Commentary("time="+text(tickData.datetime));             //时间
    Commentary("AskPrice="+text(tickData.bidask1.askP));    //申卖价 1
    Commentary("AskVol="+text(tickData.bidask1.askV));       //申卖量 1
    Commentary("BidPrice="+text(tickData.bidask1.bidP));    //申买价 1
    Commentary("BidVol="+text(tickData.bidask1.bidV));      //申买量 1
    Commentary("Last="+text(tickData.last));                 //现价
    Commentary("Open="+text(tickData.open));                //开盘价
    Commentary("High="+text(tickData.high));                 //最高价
    Commentary("Low="+text(tickData.low));                   //最低价
    Commentary("Volume="+text(tickData.volume));            //现手
    Commentary("TotalVolume="+text(tickData.totalVolume));       //总成交
    Commentary("---------------separator---------------");
    */
    If(tickData.last > 0 && tickData.totalVolume > 0)
      Return True;
     Else
```

```
        Return False;
    }
    Numeric CalcSpreadPrice(Numeric p1, Numeric p2)
    {
      return p1-p2;
    }
    Integer CalcSendOrderPrice(Bool entry,NumericRef myPrice)
    {
      If((SpreadLong&&entry)||(!SpreadLong&&!entry)) // 多头开仓或空头平仓
      {
        if (lastTickData[0].bidask1.askP < 0.000001 || lastTickData[1].bidask1.bidP < 0.000001)
return -1;
          spreadAskPrice=CalcSpreadPrice(lastTickData[0].bidask1.askP*SpreadUnitA,lastTickDa
ta[1].bidask1.bidP*SpreadUnitB);
        If(spreadAskPrice > IIF(entry,EntryLine,ExitLine)) return -2;

        orderPrice = lastTickData[1].bidask1.bidP;
        if (lastTickData[1].limitUp > 0.000001 && orderPrice > lastTickData[1].limitUp) return -3;
      }Else // 多头平仓或空头开仓
      {
        if (lastTickData[0].bidask1.bidP < 0.000001 || lastTickData[1].bidask1.askP < 0.000001)
return -1;
          spreadBidPrice=CalcSpreadPrice(lastTickData[0].bidask1.bidP*SpreadUnitA,lastTickDa
ta[1].bidask1.askP*SpreadUnitB);
        If(spreadBidPrice < IIF(entry,EntryLine,ExitLine)) return -2;

        orderPrice = lastTickData[1].bidask1.askP;
        if (lastTickData[1].limitDown > 0.000001 && orderPrice < lastTickData[1].limitDown) return
-3;
      }
      Return 0;
    }
    Numeric GetSendedOrderPrice(Bool entry,Bool pairB)
    {
      tmpNumeric = InvalidNumeric;
      tmpInteger = InvalidInteger;
      If(entry)
      {
        if(pairB)
        {
          tmpInteger = entryPairBOrderIds[0];
        }Else
        {
          tmpInteger = entryPairAOrderIds[0];
```

```
      }
    }Else
    {
      if(pairB)
      {
        tmpInteger = exitPairBOrderIds[0];
      }Else
      {
        tmpInteger = exitPairAOrderIds[0];
      }
    }

    If(tmpInteger!=InvalidInteger && A_GetOrder(tmpInteger,tmpOrder))
    {
      tmpNumeric = tmpOrder.price;
    }
    Return tmpNumeric;
  }

  //-------------------------------------------------------------------------
  // 以下为开仓操作处理函数
  Integer OnEntryStatusNone()
  {
    tmpInteger = CalcSendOrderPrice(True,orderPrice);
    If(tmpInteger != 0) Return tmpInteger;

    ArrayClear(entryPairBOrderIds);
     If(!Data1.A_SendOrderEx(IIF(SpreadLong,Enum_Sell,Enum_Buy),Enum_Entry,1,orderPrice,entry
PairBOrderIds)) Return -4;
    curEntryStatus = 1;
    Return 0;
  }
  Integer OnEntryStatusSendedData1()
  {
    needCancelOrder = False;
    If(SpreadLong)
    {
      If (lastTickData[0].bidask1.askP > 0.000001 && lastTickData[1].bidask1.bidP > 0.000001)
      {
        orderPrice = (SpreadUnitA*lastTickData[0].bidask1.askP-EntryLine)/SpreadUnitB;
         if(lastTickData[1].bidask1.bidP > orderPrice) // 价格有利，则用叫买价发单
          orderPrice = lastTickData[1].bidask1.bidP;
      }Else
      {
```

```
        needCancelOrder = True;
      }
    }Else
    {
      If (lastTickData[0].bidask1.bidP > 0.000001 && lastTickData[1].bidask1.askP > 0.000001)
      {
        orderPrice = (SpreadUnitA*lastTickData[0].bidask1.bidP-EntryLine)/SpreadUnitB;
        if(lastTickData[1].bidask1.askP < orderPrice) // 价格有利，则用叫卖价发单
          orderPrice = lastTickData[1].bidask1.askP;
      }Else
      {
        needCancelOrder = True;
      }
    }

    If(!needCancelOrder)
    {
      tmpNumeric = GetSendedOrderPrice(true,true);
       upPrice = tmpNumeric+priceTickB*DeleteOrderSet;
      dnPrice = tmpNumeric-priceTickB*DeleteOrderSet;
      if (orderPrice > dnPrice && orderPrice < upPrice)
        needCancelOrder = False; // 不需要撤单重发
       Else
        needCancelOrder = True;
    }

    If(needCancelOrder)
    {
      For i=0 To GetArraySize(entryPairBOrderIds)-1
      {
        Data1.A_DeleteOrderEx(entryPairBOrderIds[i]);
      }
      curEntryStatus = 3;
    }
    Return 0;
}
Integer OnEntryStatusFilledData1()
{
    If(SpreadLong)
    {
      orderPrice = lastTickData[0].limitUp;
      If(lastTickData[0].bidask1.askP > 0.000001)
      {
        orderPrice = Min(orderPrice,lastTickData[0].bidask1.askP+priceTickA*OrderOffset);
```

```
    }
  }Else
  {
    orderPrice = lastTickData[0].limitDown;
    If(lastTickData[0].bidask1.bidP > 0.000001)
    {
      orderPrice = Max(orderPrice,lastTickData[0].bidask1.bidP+priceTickA*OrderOffset);
    }
  }

  ArrayClear(entryPairAOrderIds);
   If(!Data0.A_SendOrderEx(IIF(SpreadLong,Enum_Buy,Enum_Sell),Enum_Entry,1,orderPrice,entryPairAOrderIds)) Return -1;

  curEntryStatus = 5;
  Return 0;
}
Integer OnEntryStatusSendedData0()
{
  If(SpreadLong)
  {
    orderPrice = lastTickData[0].bidask1.askP+priceTickA*OrderOffset;
  }Else
  {
    orderPrice = lastTickData[0].bidask1.bidP-priceTickA*OrderOffset;
  }

  tmpNumeric = GetSendedOrderPrice(true,False);
  if (orderPrice > tmpNumeric+0.000001 || orderPrice <= tmpNumeric-0.000001) // 新的委托价格和原委托价格不同即撤单重发
  {
    For i=0 To GetArraySize(entryPairAOrderIds)-1
    {
      Data0.A_DeleteOrderEx(entryPairAOrderIds[i]);
    }
    curEntryStatus = 7;
  }
  Return 0;
}
Integer OnEntryStatusFilledData0()
{
  curEntryStatus = 0;
  curPosition = curPosition + 1;
  SetTBProfileString(SpreadSection,curPosKey,Text(curPosition));
```

```
    Return 0;
  }
  Integer RunSpreadOrderEntry()
  {
    if (curPosition < MaxPosition && curEntryStatus != -99) // 还可以开仓
    {
      If(curEntryStatus == 0)
      {
        OnEntryStatusNone();
      }Else if(curEntryStatus == 1 || curEntryStatus == 5)
      {
        // 等待委托单发送到交易所
      }Else if(curEntryStatus == 2)
      {
        OnEntryStatusSendedData1();
      }Else if(curEntryStatus == 3 || curEntryStatus == 7)
      {
        // 等待撤单发送到交易所
      }Else if(curEntryStatus == 4)
      {
        OnEntryStatusFilledData1();
      }Else if(curEntryStatus == 6)
      {
        OnEntryStatusSendedData0();
      }Else if(curEntryStatus == 8)
      {
        OnEntryStatusFilledData0();
      }
    }

              spreadAskPrice=CalcSpreadPrice(lastTickData[0].bidask1.askP*SpreadUnitA,lastTi
ckData[1].bidask1.bidP*SpreadUnitB);
              spreadBidPrice=CalcSpreadPrice(lastTickData[0].bidask1.bidP*SpreadUnitA,lastTi
ckData[1].bidask1.askP*SpreadUnitB);
    Commentary("SpreadPrice="+Text(spreadAskPrice));
    Commentary("SpreadBid="+Text(spreadBidPrice));

    FileAppend("D:\\SpreadTrader.tbf","CurPosition="+Text(curPosition));
    Return 0;
  }

  //--------------------------------------------------------------------
  // 以下为平仓操作处理函数
  Integer OnExitStatusNone()
```

```
  {
    tmpInteger = CalcSendOrderPrice(False,orderPrice);
    If(tmpInteger != 0) Return tmpInteger;

    ArrayClear(exitPairBOrderIds);
    If(!Data1.A_SendOrderEx(IIF(SpreadLong,Enum_Buy,Enum_Sell),Enum_Exit,1,orderPrice,exitPa
irBOrderIds)) Return -4;
    curExitStatus = 1;
    Return 0;
  }
  Integer OnExitStatusSendedData1()
  {
    needCancelOrder = False;
    If(SpreadLong)
    {
      If (lastTickData[0].bidask1.bidP > 0.000001 && lastTickData[1].bidask1.askP > 0.000001)
      {
        orderPrice = (SpreadUnitA*lastTickData[0].bidask1.bidP-ExitLine)/SpreadUnitB;
        if(lastTickData[1].bidask1.askP < orderPrice) // 价格有利，则用叫卖价发单
          orderPrice = lastTickData[1].bidask1.askP;
      }Else
      {
        needCancelOrder = True;
      }
    }Else
    {
      If (lastTickData[0].bidask1.askP > 0.000001 && lastTickData[1].bidask1.bidP > 0.000001)
      {
        orderPrice = (SpreadUnitA*lastTickData[0].bidask1.askP-ExitLine)/SpreadUnitB;
        if(lastTickData[1].bidask1.bidP > orderPrice) // 价格有利，则用叫买价发单
          orderPrice = lastTickData[1].bidask1.bidP;
      }Else
      {
        needCancelOrder = True;
      }
    }

    If(!needCancelOrder)
    {
      tmpNumeric = GetSendedOrderPrice(False,true);
      upPrice = tmpNumeric+priceTickB*DeleteOrderSet;
      dnPrice = tmpNumeric-priceTickB*DeleteOrderSet;
      if (orderPrice > dnPrice && orderPrice < upPrice)
        needCancelOrder = False; // 不需要撤单重发
```

```
        Else
          needCancelOrder = True;
      }

      If(needCancelOrder)
      {
        For i=0 To GetArraySize(exitPairBOrderIds)-1
        {
          Data1.A_DeleteOrderEx(exitPairBOrderIds[i]);
        }
        curExitStatus = 3;
      }
      Return 0;
    }
    Integer OnExitStatusFilledData1()
    {
      If(SpreadLong)
      {
        orderPrice = lastTickData[0].limitDown;
        If(lastTickData[0].bidask1.bidP > 0.000001)
        {
          orderPrice = Max(orderPrice,lastTickData[0].bidask1.bidP-priceTickA*OrderOffset);
        }
      }Else
      {
        orderPrice = lastTickData[0].limitUp;
        If(lastTickData[0].bidask1.askP > 0.000001)
        {
          orderPrice = Min(orderPrice,lastTickData[0].bidask1.askP-priceTickA*OrderOffset);
        }
      }
      ArrayClear(exitPairAOrderIds);
       If(!Data0.A_SendOrderEx(IIF(SpreadLong,Enum_Sell,Enum_Buy),Enum_Exit,1,orderPrice,exitPa
irAOrderIds)) Return -1;

      curExitStatus = 5;
      Return 0;
    }
    Integer OnExitStatusSendedData0()
    {
      If(SpreadLong)
      {
        orderPrice = lastTickData[0].bidask1.bidP-priceTickA*OrderOffset;
      }Else
```

```
    {
      orderPrice = lastTickData[0].bidask1.askP+priceTickA*OrderOffset;
    }

    tmpNumeric = GetSendedOrderPrice(False,False);
    if (orderPrice > tmpNumeric+0.000001 || orderPrice < tmpNumeric-0.000001) // 新的委托价格和原委托价格不同即撤单重发
    {
      For i=0 To GetArraySize(exitPairAOrderIds)-1
      {
        Data0.A_DeleteOrderEx(exitPairAOrderIds[i]);
      }
      curExitStatus = 7;
    }
    Return 0;
  }
  Integer OnExitStatusFilledData0()
  {
    curExitStatus = 0;
    curPosition = curPosition - 1;
    SetTBProfileString(SpreadSection,curPosKey,Text(curPosition));
    Return 0;
  }
  Integer RunSpreadOrderExit()
  {
    if (curPosition > 0 && curExitStatus != -99) // 还可以开仓
    {
      If(curExitStatus == 0)
      {
        OnExitStatusNone();
      }Else if(curExitStatus == 1 || curExitStatus == 5)
      {
        // 等待委托单发送到交易所
      }Else if(curExitStatus == 2)
      {
        OnExitStatusSendedData1();
      }Else if(curExitStatus == 3 || curExitStatus == 7)
      {
        // 等待撤单发送到交易所
      }Else if(curExitStatus == 4)
      {
        OnExitStatusFilledData1();
      }Else if(curExitStatus == 6)
      {
```

```
        OnExitStatusSendedData0();
      }Else if(curExitStatus == 8)
      {
        OnExitStatusFilledData0();
      }
    }
    Return 0;
}

//--------------------------------------------------------------------------
// 以下为委托状态变化处理函数
Bool IsMyOrder(OrderRef ord,ArrayRef<Integer> orderIds)
{
    for i=0 To GetArraySize(orderIds)-1
    {
      If(ord.orderId==orderIds[i])
        Return True;
    }
    Return False;
}
Integer OnNotifySending(OrderRef ord,bool pairB)
{
    tmpInteger = IIF(pairB,1,5);      // 默认为发送中
    If(ord.status==Enum_Declared)   // 已报
    {
      If(ord.exchOrderId!="")
      {
        tmpInteger = IIF(pairB,2,6);
      }
    }else If(ord.status==Enum_Deleted || ord.status==Enum_Canceled) // 废单或交易所自动撤单
    {
      tmpInteger = IIF(pairB,0,4);      // 修改为初始状态或 Pair B 已成交状态
    }Else If(ord.status==Enum_Filled)   // 跳过已报状态直接收到成交状态
    {
      tmpInteger = IIF(pairB,4,8);      // 修改为已成交状态
    }Else If(ord.status==Enum_Declare)
    {
      // 等待发送成功
    }Else
    {
      tmpInteger = -99;    // 错误的状态
    }
    Return tmpInteger;
}
```

```
Integer OnNotifySended(OrderRef ord,bool pairB)
{
  tmpInteger = IIF(pairB,2,6);  // 默认为已报
  If(ord.status==Enum_Filled)  // 已成交
  {
    tmpInteger = IIF(pairB,4,8);  // 修改为已成交状态
  }Else If(ord.status==Enum_Canceled) // 已撤单
  {
    tmpInteger = IIF(pairB,0,4);    // 修改为初始状态或 Pair B 已成交状态
  }Else If(ord.status==Enum_FillPart)  // 部分成交
  {
    // 等待全部成交
  }Else If(ord.status!=Enum_Declared)
  {
    tmpInteger = -99;   // 错误的状态
  }
  Return tmpInteger;
}
Integer OnNotifyCanceling(OrderRef ord,bool pairB)
{
  tmpInteger = IIF(pairB,3,7);  // 默认为撤单中
  If(ord.status==Enum_Canceled)  // 已撤单
  {
    tmpInteger = IIF(pairB,0,4); // 修改为初始状态或 Pair B 已成交状态
  }Else If(ord.status==Enum_Filled) // 撤单不及时，已经成交
  {
    tmpInteger = IIF(pairB,4,8);  // 修改为已成交状态
  }Else If(ord.status==Enum_Declared)  // 已报
  {
    // 等待撤单成功
  }Else If(ord.status!=Enum_Canceling)
  {
    tmpInteger = -99;   // 错误的状态
  }
  Return tmpInteger;
}

Integer OnNotifyOrder(OrderRef ord,Integer myStatus)
{
  If(myStatus==1) // 委托 B 发送中
  {
    myStatus = OnNotifySending(ord,True);
  }else If(myStatus==2) // 委托 B 发送成功
  {
```

```
      myStatus = OnNotifySended(ord,True);
    }else If(myStatus==3) // 委托 B 撤单中
    {
      myStatus = OnNotifyCanceling(ord,True);
    }else If(myStatus==5) // 委托 A 发送中
    {
      myStatus = OnNotifySending(ord,False);
    }else If(myStatus==6) // 委托 A 发送成功
    {
      myStatus = OnNotifySended(ord,False);
    }Else If(myStatus==7) // 委托 A 撤单中
    {
      myStatus = OnNotifyCanceling(ord,False);
    }

    If(myStatus==4||myStatus==8) // 委托 A 或委托 B 已经成交
    {
      RunSpreadOrderEntry();
       RunSpreadOrderExit();
    }
    Return myStatus;
  }
Events
  OnInit()
  {
    SubscribeBar(SymbolA,"tick");
    SubscribeBar(SymbolB,"tick");
    A_SubscribeTradeByCreateId(Enum_Trade_Source_ALL);
    paramReady = CheckParams();
  }

  //在所有的数据源准备完成后调用，应用在数据源的设置等操作
  OnReady()
  {
    curPosKey = SymbolA+"-"+SymbolB+IIFString(SpreadLong,".Long",".Short")+"Pos";
    If(ResetPosition)  // 重置持仓数据
    {
      curPosition = InitPosition;
      SetTBProfileString(SpreadSection,curPosKey,Text(curPosition));
    }Else // 从数据库中读取持仓数据
    {
      tmpStr = GetTBProfileString(SpreadSection,curPosKey);
      If(tmpStr!=InvalidString && tmpStr !="")
        curPosition = Value(tmpStr);
```

```
    }

    priceTickA = Data0.MinMove*Data0.PriceScale;
    priceTickB = Data1.MinMove*Data1.PriceScale;
}

//在新 bar 的第一次执行之前调用一次，参数为新 bar 的图层数组
OnBarOpen(ArrayRef<Integer> indexs)
{
}

OnBar(ArrayRef<Integer> indexs)
{
    For i=0 To GetArraySize(indexs)-1
    {
        Data[indexs[i]].GetTick(tmpTickData);
        if(CheckTickData(tmpTickData))
        {
            tickDataValid[indexs[i]] = True;
            lastTickData[indexs[i]] = tmpTickData;
        }
    }

    dataReady = True;
    For i=0 To DataCount-1
    {
        If(!tickDataValid[indexs[i]])
        {
            dataReady = False;
            Break;
        }
    }

    If(paramReady && dataReady)
    {
        RunSpreadOrderEntry();
        RunSpreadOrderExit();
    }
}

//持仓更新事件函数，参数 pos 表示更新的持仓结构体
OnPosition(PositionRef pos)
{
}
```

```
  //委托更新事件函数，参数 ord 表示更新的委托结构体
  OnOrder(OrderRef ord)
  {
    If(IsMyOrder(ord,entryPairAOrderIds)||IsMyOrder(ord,entryPairBOrderIds))    // 判断是否为该系统发出的开仓单
    {
      curEntryStatus = OnNotifyOrder(ord, curEntryStatus);
    }else if(IsMyOrder(ord,exitPairAOrderIds)||IsMyOrder(ord,exitPairBOrderIds))   // 判断是否为该系统发出的平仓单
    {
      curExitStatus = OnNotifyOrder(ord, curExitStatus);
    }
  }

  //成交更新事件函数，参数 ordFill 表示更新的成交结构体
  OnFill(FillRef ordFill)
  {
  }

  //定时器更新事件函数，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值
  OnTimer(Integer id,Integer intervalMillsecs)
  {
  }

//-------------------------------------------------------------------
// 编译版本:  2020/01/06 130422
// 版权所有  nopain
// 更改声明  TradeBlazer Software 保留对 TradeBlazer 平台
//       每一版本的 TradeBlazer 公式修改和重写的权利
//-------------------------------------------------------------------
```

七、条件选股

7.1、应用背景

股票投资中选股是一个很重要的功能。而每个投资者的选股方法又不同，所以选股既要便捷实现，又要个性化。这个对于一般的软件确实是个难题。但是对于 TBQuant 来说，有底层的 TBL 语言作为支撑，又有模式运行作为应用，很好的解决了这个难题。

下面我们以技术选股和财务选股两个经典案例来讲解代码的实现。

基本的流程是，我们编写代码，然后生成模式，然后运行模式选股。

本文中我们重点讲解编写代码部分。至于模式的使用可以查看 TradeBlazer 官网的帮助中心 TBQuant 软件使用的相应章节。

7.2、技术选股

应用说明

我们以双均线和 KDJ 为例来编写这样的模式。
如果选择双均线选股，短期均线>长期均线，选出；
如果选择 KDJ 选股，K>D，选出。

编写要点

1、参数的分类

我们对照着代码和效果图

```
Params
    Enum<String> index(["ma","kdj"]);       //指标选择
    //@\group {index.ma}
    Numeric mashortlen(5);                  //短期均线
    //@\group {index.ma}
    Numeric malonglen(20);                  //长期均线
    //@\group {index.kdj}
    Numeric Length(14);                     //KDJ的第一个参数
    //@\group {index.kdj}
    Numeric SlowLength(3);                  //KDJ的第二个参数
    //@\group {index.kdj}
    Numeric SmoothLength(3);                //KDJ的第三个参数
```

参数的分组的代码：

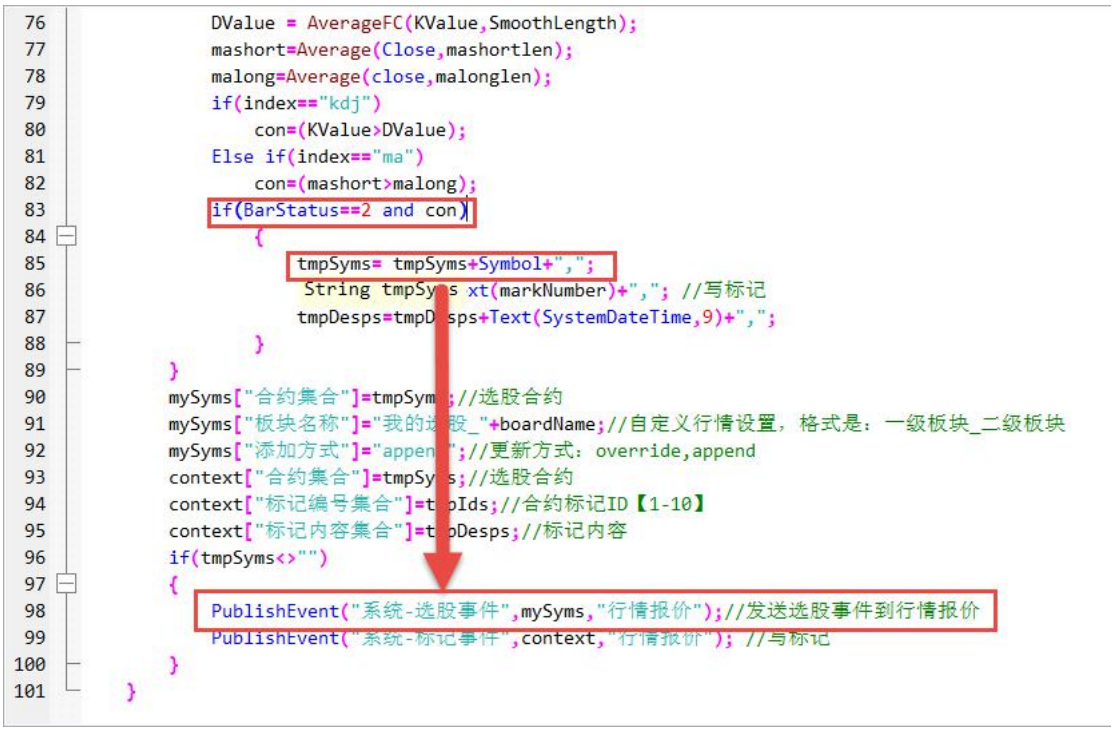
```
//@\group {index.ma}
    Numeric mashortlen(5);                  //短期均线
```

//@\group {}是基本结构，不能改变。大括号内是参数值的集合。
这一句话表示的是下一行的参数，从属于大括号内的参数值。可以从属于多个参数值。
比如上面的两句话，代码短期均线这个参数从属于 index.ma 这个参数值。

2、股票推送行情报价

如何把代码运行选出的股票推送到行情报价呢？

在代码中需要用到发送事件的功能，具体实现如下图所示。



```
            DValue = AverageFC(KValue,SmoothLength);
            mashort=Average(Close,mashortlen);
            malong=Average(close,malonglen);
            if(index=="kdj")
                con=(KValue>DValue);
            Else if(index=="ma")
                con=(mashort>malong);
            if(BarStatus==2 and con)
                {
                    tmpSyms= tmpSyms+Symbol+",";
                     String tmpSyms xt(markNumber)+","; //写标记
                    tmpDesps=tmpDesps+Text(SystemDateTime,9)+",";
                }
        }
        mySyms["合约集合"]=tmpSyms;//选股合约
        mySyms["板块名称"]="我的选股_"+boardName;//自定义行情设置，格式是：一级板块_二级板块
        mySyms["添加方式"]="append";//更新方式：override,append
        context["合约集合"]=tmpSyms;//选股合约
        context["标记编号集合"]=tmpIds;//合约标记ID【1-10】
        context["标记内容集合"]=tmpDesps;//标记内容
        if(tmpSyms<>"")
        {
            PublishEvent("系统-选股事件",mySyms,"行情报价");//发送选股事件到行情报价
            PublishEvent("系统-标记事件",context,"行情报价"); //写标记
        }
    }
```

完整代码

Params

```
    Enum<String> index(["ma","kdj"]);        //指标选择
    //@\group {index.ma}
    Numeric mashortlen(5);                   //短期均线
    //@\group {index.ma}
    Numeric malonglen(20);                   //长期均线
    //@\group {index.kdj}
    Numeric Length(14);                      //KDJ 的第一个参数
    //@\group {index.kdj}
    Numeric SlowLength(3);                   //KDJ 的第二个参数
    //@\group {index.kdj}
    Numeric SmoothLength(3);                 //KDJ 的第三个参数
    //订阅数据和推送选股的参数
    Array<String> pStkIndustryID([]);    //手动选择板块
    Numeric pStartDateTime(20200901);    //查询起始时间
    Numeric pEndDateTime(22000101);      //查询结束时间
    String boardName("技术选股");         //推送到行情报价的三级板块名
    Numeric markNumber(1);               //标记的号码
```

Vars

```
    //基础变量
    Global Integer i;
    Global Integer j;
    Bool con;
    //推送选股变量
```

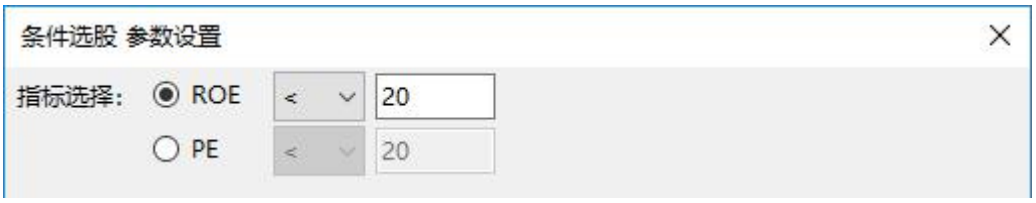
```
    Global String tmpSyms;       //自定义合约的股票
    Global String tmpIds;
    Global String tmpDesps;
    Map<String,String> mySyms;   //推送自定义合约的 MAP
    Map<String,String> context; //写标记用
    Array<String> subSymbols; //追加的数据源
    //技术指标
    Series<Numeric> mashort;
    Series<Numeric> malong;
    Series<Numeric> HighestValue;
    Series<Numeric> LowestValue;
    Series<Numeric> KValue;
    Numeric SumHLValue;
    Numeric SumCLValue;
    Numeric DValue;
Defs
    Integer LogFile(String str)
    {
        FileAppend("d:\\my"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
        Return 0;
    }
Events
    //订阅行情
    OnInit()
    {
        For j=0 To GetArraySize(pStkIndustryID)/2-1
        {
            ArrayClear(subSymbols);

    GetDicValue(pStkIndustryID[j*2],pStkIndustryID[j*2+1],SystemDateTime(),subSymbols);
            for i=0 To GetArraySize(subSymbols)-1
            {

    SubscribeBar(subSymbols[i],"1d",pStartDateTime,pEndDateTime);//,Enum_Data_NoQuoteData);
            }
        }
    }

    OnBar(ArrayRef<Integer> indexs)
    {
        Range[0:DataCount-1]
        {
            HighestValue = HighestFC(High, Length);
            LowestValue = LowestFC(Low, Length);
```

```
            SumHLValue = SummationFC(HighestValue-LowestValue,SlowLength);
            SumCLValue = SummationFC(Close - LowestValue,SlowLength);
            If(SumHLValue <> 0)
            {
                KValue = SumCLValue/SumHLValue*100;
            }Else
            {
                KValue = 0;
            }
            DValue = AverageFC(KValue,SmoothLength);
            mashort=Average(Close,mashortlen);
            malong=Average(close,malonglen);
            if(index=="kdj")
                con=(KValue>DValue);
            Else if(index=="ma")
                con=(mashort>malong);
            if(BarStatus==2 and con)
                {
                    tmpSyms= tmpSyms+Symbol+",";
                    tmpIds=tmpIds+Text(markNumber)+","; //写标记
                    tmpDesps=tmpDesps+Text(SystemDateTime,9)+",";
                }
        }
        mySyms["合约集合"]=tmpSyms;//选股合约
        mySyms["板块名称"]="我的选股_"+boardName;//自定义行情设置，格式是：一级板块_二级板块
        mySyms["添加方式"]="append";//更新方式：override,append
        context["合约集合"]=tmpSyms;//选股合约
        context["标记编号集合"]=tmpIds;//合约标记 ID【1-10】
        context["标记内容集合"]=tmpDesps;//标记内容
        if(tmpSyms<>"")
        {
            PublishEvent("系统-选股事件",mySyms,"行情报价");//发送选股事件到行情报价
            PublishEvent("系统-标记事件",context,"行情报价"); //写标记
        }
    }
```

7.3、财务选股

应用说明

我们以 PE/ROE 为例来编写这样的模式。
用户选择任何一个财务指标，可以接下来设置>=或者<目标数值，进行选股。
比如用户选择 PE<20，则把 PE 小于 20 的股票选出。



编写要点

1、如何调用基础数据

获取基础数据，可以先定义基础数据变量，然后在公式中使用读取基础数据的函数来获取。

```
Vars
    Dic<Array<Numeric>> pershare("TB_FINANALYSIS_PERSHARE");    //读取每股财务指标
    Dic<Array<Numeric>> earning("TB_FINANALYSIS_EARNINGSQUALITY");    //读取盈利质量财务指标
    Series<Numeric> me;                    //财务指标每股收益
    Series<Numeric> roe;                   //财务指标roe
    Series<Numeric> pe;                    //财务指标pe
```

```
            //获取基础数据
            if(index=="PE")
            {
                GetDicValue(pershare,SystemDateTime,dicValues);
                indexvalue=lastclose/dicValues[4];
            }
            Else if(index=="ROE")
            {
                GetDicValue(earning,SystemDateTime,dicValues);
                indexvalue=dicValues[6];
            }
```

2、如何在没有订阅行情的情况下取得最新价

对于 PE 选股，我们从基础数据可以读到每股盈利，但是最新价需要从行情中取。如果使用 onbar 那么所有历史 BAR 都会运行一遍。这对于当前应用没有必要，我们只是读一个最新价。所以可以在 onready 里面，直接读取。

```
    OnReady()
    {
        Range[i=0:DataCount-1]
        {
            //获取最新价
            GetReadyBar(lastbar,BarCount-1);
            lastclose=lastbar.close;
```

完整代码

```
Params
    Enum<String> index(["PE","ROE"]);        //指标选择
```

```
    //@\group {index.PE;index.ROE}
    Enum<String> relationcal(["<",">="]);       //关系运算
    //@\group {index.PE;index.ROE}
    Numeric tindexvalue(20);                        //指标的目标值
    Array<String> pStkIndustryID([]);   //手动选择板块
    Numeric pStartDateTime(20200901);   //查询起始时间
    Numeric pEndDateTime(22000101);     //查询结束时间
    String boardName("财务选股");       //推送到行情报价的三级板块名
    Numeric markNumber(1);              //标记的号码

Vars
    Dic<Array<Numeric>> pershare("TB_FINANALYSIS_PERSHARE");    //读取每股财务指标
    Dic<Array<Numeric>> earning("TB_FINANALYSIS_EARNINGSQUALITY");    //读取盈利质量财务指标
    Series<Numeric> me;                         //财务指标每股收益
    Series<Numeric> roe;                        //财务指标 roe
    Series<Numeric> pe;                         //财务指标 pe

    Global String tmpSyms;       //自定义合约的股票
    Global String tmpIds;
    Global String tmpDesps;

    Global Integer i;
    Global Integer j;

    Array<Numeric> dicValues;
    //Array<Numeric> dicTimes;

    Map<String,String> mySyms;   //推送自定义合约的 MAP
    Map<String,String> context; //写标记用
    Array<String> subSymbols; //追加的数据源

    Numeric lastclose(0);
    Bar lastbar;
    Bool con;
    Numeric indexvalue;
Defs
    Integer LogFile(String str)
    {
        FileAppend("d:\\my"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
        Return 0;
    }
Events
    //订阅行情
    OnInit()
```

```
{
    For j=0 To GetArraySize(pStkIndustryID)/2-1
    {
        ArrayClear(subSymbols);

GetDicValue(pStkIndustryID[j*2],pStkIndustryID[j*2+1],SystemDateTime(),subSymbols);
        for i=0 To GetArraySize(subSymbols)-1
        {

SubscribeBar(subSymbols[i],"1d",pStartDateTime,pEndDateTime);//,Enum_Data_NoQuoteData);
        }
    }
}

OnReady()
{
    Range[i=0:DataCount-1]
    {
        //获取最新价
        GetReadyBar(lastbar,BarCount-1);
        lastclose=lastbar.close;
        //获取基础数据
        if(index=="PE")
        {
            GetDicValue(pershare,SystemDateTime,dicValues);
            indexvalue=lastclose/dicValues[4];
        }
        Else if(index=="ROE")
        {
            GetDicValue(earning,SystemDateTime,dicValues);
            indexvalue=dicValues[6];
        }
        LogFile(Text(i)+"_"+Text(lastclose)+"_"+Text(indexvalue));
        if(relationcal=="<")
            con=(indexvalue<tindexvalue);
        else if(relationcal==">=")
            con=(indexvalue>=tindexvalue);
        if(con)
        {
            tmpSyms= tmpSyms+Symbol+",";
            tmpIds=tmpIds+Text(markNumber)+","; //写标记
            tmpDesps=tmpDesps+Text(SystemDateTime,9)+",";
        }
    }
```

```
    mySyms["合约集合"]=tmpSyms;//选股合约
    mySyms["板块名称"]="我的选股_"+boardName;//自定义行情设置，格式是：一级板块_二级板块
    mySyms["添加方式"]="append";//更新方式：override,append
    context["合约集合"]=tmpSyms;//选股合约
    context["标记编号集合"]=tmpIds;//合约标记 ID【1-10】
    context["标记内容集合"]=tmpDesps;//标记内容
    if(tmpSyms<>"")
    {
        PublishEvent("系统-选股事件",mySyms,"行情报价");//发送选股事件到行情报价
        PublishEvent("系统-标记事件",context,"行情报价"); //写标记
    }
    AddStrategyFlag(Enum_Strategy_Finished);//完成
}
```

第二部分 TBL 语法精要

一、TBL 语言

1.1 公式运行机制

TBL 语言是基于 TBQuant 金融量化平台的一种专用计算机语言。其语法简洁易懂，介于 C++与 Pascal 之间。其程序语言可以由多重数学、布尔值的计算以及逻辑判断等组成。TB 公式属于编译型公式，即只有通过编译的公式程序方可被应用于图表，这样使得公式的执行更有效率。

TBQuant 作为一款金融量化的软件，针对金融市场的数据特性，它提供了很多使用方便的机制用于金融量化交易。针对金融时间序列数据的特性，从 TB 软件最初的 begin/end 的机制，到现在的 onbar 等事件驱动机制，这都给时间序列建模提供了极大方便。而 TBQuant 的多品种多周期的自动对齐机制，使得对截面数据的处理也变得方便高效。

下面我们介绍下 TBQuant 的 3 个重要运行机制。

单数据源的 onbar 机制

Onbar 机制，就是之前产品的 begin/end 机制。当策略公式在 onbar 机制下运行时，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、最高价、最低价、收盘价、成交量、持仓量以及时间等信息数据。Bar 数据也是我们口头上常说的 K 线数据。

1）历史数据的运行机制：从左到右、从上到下

公式在计算时按照“Bar 数据”的 Bar 数目，从左边第一个 Bar 依次执行到右边最后一个 Bar，在单个 Bar 数据上的公式运算为从上到下完整执行公式中所有语句，即每次公式的运算都是从公式最上方的语句“参数的声明、变量的声明”开始直至公式的计算主体 onbar{}(Begin 至 End)结束。

如果所示，我们定义了一个全局变量，在第一根 BAR 初始值为 0，然后其它 BAR 每次加 1。
TBQuant 的 onbar 机制从左边第一根 BAR 开始运行公式一次，公式代码的运行则从上到下依次执行。然后第二根 BAR 运行公式 num 变为 1,然后第三根 BAR 运行公式，直到倒数第二根 BAR（第 14 根 BAR）运行结束，num 累计为 13。

2）实时行情的运行机制：每个 TICK 运行一次

对于实时数据，每当有新的 Tick 进来，公式都会在当前 Bar 上对新数据执行一次完整的运算。比如如果上图中的公式在 2019/7/19 的交易时间运行，那么每来一个新的 TICK，公式会从上到下执行依次，num 会累加 1。盘中在 2019/7/19 这根 bar 上 num 的数值会不断累计变化。如果当天有 100 个 TICK,则 num 在 2019/7/19 这根 BAR 结束时会总共累加 100 次。最后 num 的数值会变成 113。
注意：若当前公式所应用的合约交易非常活跃并且公式程序较长、计算较复杂时，当前 Tick 到来之后与下一个 Tick 到来之前的这段时间之内，可能无法完成公式代码完整执行一遍的计算。 此时，虽然新的 Tick 到来，但是不会触发公式的重新运行，依然继续执行之前的计算直至代码的最后一行。 之后，当最新的 Tick 到来时，才会再次触发新一轮的公式运算。也就是说，在这种情况下，不是 Bar 中每一个 Tick 到来时都触发公式重新计算一次。下表列出了历史回测和实时交易的区别。

	历史回测	实时交易
Bar 数据	确定不变	实时更新
公式运行	每根 Bar 一次	每个 Tick 一次
交易信号	固定不变	有可能变化
是否发单	否	是（受公式机制控制）
函数调用	部分函数无效	有效

多数据源的 onbar 机制

Onbar 在多数据源上的运行的机制虽然基本特征依旧满足单数据源的 onbar 机制的特征：1、历史数据是从左到右从上到下的运行。2、实时行情每个 tick 驱动一次。但是多数据要比在一个数据源上的机制复杂，需要考虑各个数据源的时间问题，这里面既有不同周期的数据对齐，也有不同交易时间的特殊处理。这里我们重点讲解多数据源 onbar 机制的 4 个方面：1、数据的对齐机制 ；2、数据的编号；3、公式的运行机制；4、特殊情况的补充机制。

1）数据的对齐机制

关于多数据源的对齐机制我们在本文开始的章节已有介绍，重点在于总时间坐标的确定，这里不再复述。需要注意的是各数据源的最后一根 BAR 肯定是对齐在一起的。

2）数据源的编号

当有多个数据源时，系统会给这些数据源进行编号，编号从 0 依次累加。如果总共有三个合约，那么三个数据源的序号是 0，1，2。如果需要调用相应数据源的数据或者公式运算，则直接用 data[i].的前缀就可以调用第 i+1 个（因为编号从 0 开始）数据源的数据或者公式运算。

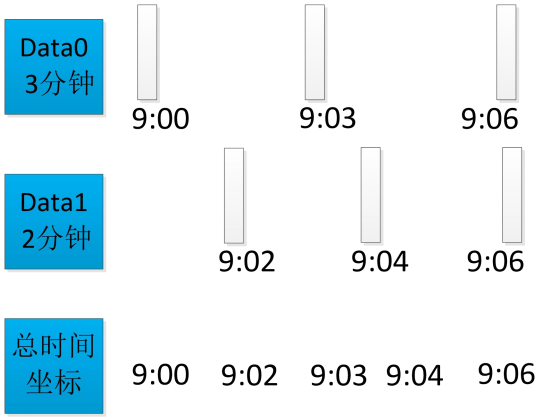
3）公式运行的机制

3.1）公式运行机制的建立：在每个数据源上建立局部变量的备份

当只有一个数据源时，公式依赖于 data0 这唯一的数据源运行。这体现在除了全局变量其余的局部变量都是依赖于 data0 的数据源的。
当有多个数据源时，公式会在每个数据源都建立一套机制，对应的除了全局变量其余的局部变量在每个数据源都有自己独立的存在。
比如公式定义一个普通的数值变量 numeric num(0)；如果公式应用于有三个数据源的交易单元，则实际上存在三个独立的 num，分别是 data0.num,data1.num,data2.num。

3.2）多数据源的运行机制什么时候被触发？

对于历史数据，公式会每个总时间坐标上运行一次。
对于实盘数据，任何一个数据源的 tick 更新都会触发公式运行一次。运行的数据源是基于每个数据源上的最后一根 BAR。但是公式执行过程中，如有多次更新会合并触发下一次运行。
比如下面的例子：数据和代码分别如下：




```
if (Data0. Open - Data1. Open) >10    (1)
{
  Data0.Buy(1, Open);                 (2)
```

```
  Data1.Sellshort(1, Open);          (3)
}
```

a)对于历史数据，公式会在每个时间坐标（9:00，9:02，9:03 9:04，9:06）上都会运行一次。上述公式运行时，每条语句的执行情况如下表所示：

时间坐标	（1）执行	（2）执行	（3）执行
9:00	✓	✓	
9:02	✓		✓
9:03	✓	✓	
9:04	✓		✓
9:06	✓	✓	✓

b)对于实时数据，上述公式运行时，每条语句的执行情况如下表所示：

新 tick	（1）执行	（2）执行	（3）执行
Data0	✓	✓	
Data1	✓		✓

各数据源在计算指标时不补缺失的数据，直接取最原始的数据进行计算。

如 Data0.var1 = Data1.averageFC(Close，10)，取 Data1 最近 10 根有效的 bar 的 Close 计算均价。

（注：通过 Commentary，在相应数据源的输出，会合并输出，如在 30Min(Data[0])输出 15Min(Data[1])，会有相应的两次输出）

3.3）哪些代码会在哪个数据源上被执行

Onbar{}之间的代码在那个数据源上运行则依赖 range[i:j]和 data[i].这两个机制控制。Range 和 data 中的下标 i 与数据源的编号对应。

data[i].对单条代码运行的控制

Data[i].可以加在局部变量前面，这代表这个局部变量是属于 data[i]的那个局部变量；

Data[i].可以加在函数的前面,这代表这个函数运算只在 data[i]上运行,函数中的局部变量如果没 data[j]的前缀，都默认是 data[i]的局部变量。

注意：

a)没有加 data[i].前缀的局部变量默认是 data0 的（除了 data[i].函数中的局部变量）。

b)Data[i]和 datai 等价。

c)没有指定数据源的函数系统默认添加数据源前缀 Data0。

d)函数入参可以是其它数据源的数据，如 Data0.Average（Data1. Close，3）。

e)函数入参不显示数据源则仍默认其数据源为父域，如 Data1.Average（Close，3）中的 Close 指 Data1.Close。

我们举例说明：

比如下面求均值的简单语句

Code 语句	Avg 是哪个图层	Average 是那个图层	Close 是哪个图层
Avg=average(close,5);	Data0	Data0	Data0
Avg=data1.average(close,5);	Data0	Data1	Data1
Avg=average(data1.close,5);	Data0	Data0	Data1
Avg=data1.average(data0.close,5);	Data0	Data1	Data0
Data1.Avg=average(close,5);	Data1	Data0	Data0

Data1.Avg=data1.average(close,5);	Data1	Data1	Data1
Data1.Avg=average(data1.close,5);	Data1	Data0	Data1
Data1.Avg=data1.average(data0.close,5);	Data1	Data1	Data0

3.4）特殊情况的处理机制

基于 3.1）中所描述的数据对齐机制和 3.2）中描述的实时行情的运行机制，会出现下列情况：一个有实时行情的数据源的 TICK 触发了一个处于非交易时间的数据源上的执行机制。比如 data0 是 rb1910，data1 是 jd1909 这样的两个数据源，如果处于夜盘交易时段，这就会发生 rb1910 的 tick 会驱动 jd1909 的执行机制运行。这时候运行机制需要做特别处理。

处理规则是：

当一个数据源的数据时间(rb1910 的夜盘时段)已经不在另一个数据源的数据交易时间段时（比如 jd1909 没有夜盘），该数据源（rb1910）那就不能修改另一个数据源(jd1909)的信号、序列变量、commentery、plot、基础数据，即使修改也被忽略。

比如下面这个简单的例子：

```
Events
OnBar(ArrayRef<Integer> indexs)
{    If(time==0.1455)
         Data1.buy(1,open);
}
```

如果加载在一个策略单元，这个策略单元第一个数据源是 rb1910 的 5min，第二个数据源是 jd1909 的 5min. 那么夜盘时间 rb1910 的每一个根 BAR 在运行时都对应这 jd1909 的最后一根 BAR,也就是 14.55 的那根 BAR。我们比较下特殊处理前后的执行过程。

时间	Time==0.1455	Data1.buy（无特殊处理）	Data1.buy（有特殊处理）
14.55	成立	图上标信号并买入	图上标信号并买入
21.00	不成立	图上信号消失	图上信号依旧存在

事件驱动机制 onorder 等

为了满足交易策略的多样化，TBQuant 扩展了 onbar 运行机制到事件驱动机制。事件驱动顾名思义就是以事件的发生驱动某些代码的执行。Onbar 也是事件驱动的一种。这里简单介绍下事件驱动的基本类型和机制。事件驱动的详细介绍可以看学习资料的文档《事件驱动案例讲解》。

1、事件驱动的基本类型

事件驱动基本类型有 12 种，对应的基本特征如下表所示：

事件名称	调用语法	基本解释

初始化事件
(OnInit)
	OnInit()
	{
	//用户初始化逻辑，主要用于创建各种数据源
	}
		由策略执行前的初始化事件，只运行一次 只能依赖或使用 Global 等全局变量、全局设置信息 可以订阅数据，数据准备等操作

后初始化事件
(OnReady)
	OnReady()
	{
	//在所有的数据源准备完成后调用，应用在数据源的设置等操作
	}
		在策略执行前的初始化事件（数据源数据准确完成）之后，只运行一次 只能依赖或使用 Global 等全局变量、全局设置信息 可以对各个数据源进行相关设置

Bar 数据更新驱动
(OnBar)
	OnBar(Arrayref<Integer>indexs)
	{
	//用户业务逻辑
	}
		当 Bar 更新变化时驱动，参数 indexs 表示更新的图层编号数组

Bar 数据开始驱动
(OnBaropen)
	OnBaropen(ArrayRef<Integer> indexs)
	{
	//用户业务逻辑
	}
		当 Bar 第一次生成时驱动，参数 indexs 表示更新的图层编号数组

Bar 数据结束驱动
(OnBarclose)
	OnBarclose(ArrayRef<Integer> indexs)
	{
	//用户业务逻辑
	}
		当下一个 Bar 开始之前，最后一次当前 bar 驱动，参数 indexs 表示更新的图层编号数组

持仓更新驱动
(OnPosition)
	OnPosition(PositionRef pos) {
	//用户业务逻辑
	}
		当持仓更新时驱动，参数 pos 表示更新的持仓结构体

委托更新驱动(
OnOrder)
	OnOrder(OrderRef ord)
	{
	//用户业务逻辑
	}
		当委托更新时驱动，参数 ord 表示更新的委托结构体

成交更新驱动
(OnFill)
	OnFill(FillRef ordFill)
	{
	//用户业务逻辑
	}
		当成交更新时驱动，参数 ordFill 表示更新的成交结构体

定时器更新驱动
(OnTimer)
	OnTimer(Integer id,Integer millsecs)
	{
	//用户业务逻辑
	}
		当定时器更新时驱动，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值

策略账户持仓更新事件 (OnStrategyPosition)	OnStrategyPosition(PositionRef pos) { //用户业务逻辑 }	当策略账户仓更新驱动，策略账户仓是指本策略当日成交累计量，是一个相对持仓，参数 pos 表示持仓结构体
通用事件 (OnEvent)	OnEvent(StringRef name,MapRef<String,String> evtValue) { //用户业务逻辑 }	当订阅了通用事件，有事件是驱动，参数 name 表示事件名称，evtValue 表示事件内容
退出事件 (OnExit)	OnExit() { //用户业务逻辑}	当策略退出时驱动

2、事件驱动的运行机制

事件驱动的整个结构运行的顺序是:

1、全局变量初始化。

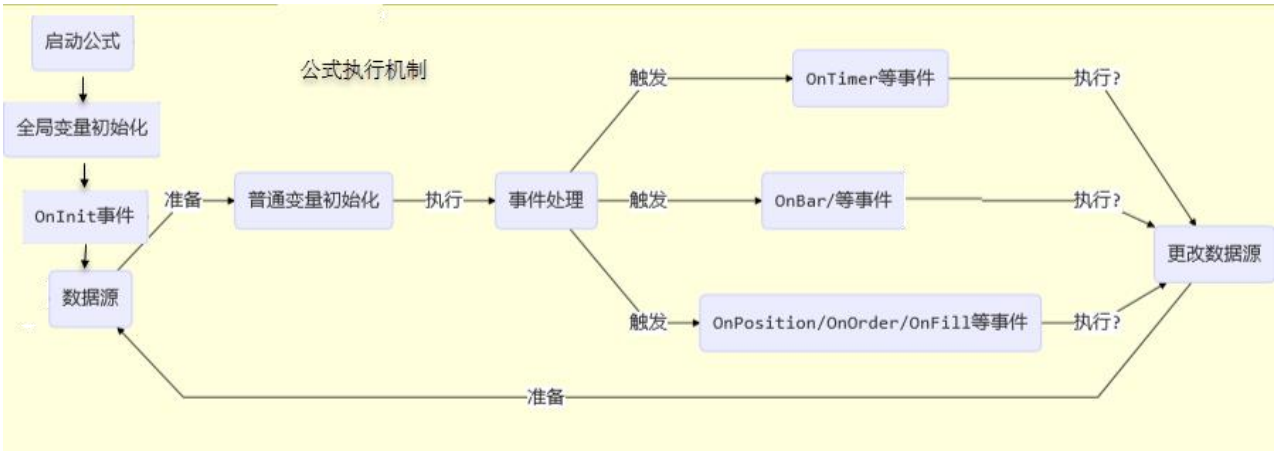
2、oninit 事件触发。在 oninit 事件中一般完成数据源的订阅、定时器的设置等操作。

3、数据源的准备。

4、普通变量初始化。

5、其它事件 ontimer/onbar/onposition 等的触发。

6、是否更改数据源。



多公式组合策略在数据源上的运算

为了使用的方便，以及降低使用门槛，TB 公式系统支持将现有的多个公式应用按照一定的顺序组合为一个交易策略。
交易策略执行时按照公式设置的顺序依次串行执行，公式间持仓共享，公式信号单独显示。
相当于把每个公式看成是一个函数，在交易策略内依次执行各个函数，整个策略共享一个 MarketPosition。

TB 提供了很多常用策略的公式应用代码，如交易策略进阶中的止盈止损、跟踪止损、收盘平仓、撤单、平仓延迟反手等。
用户可以根据需要将自己的公式组合 TB 提供的上述公式应用，形成一个完整的交易策略。
用户也可以将多个入场方式和出场方式分别写成单独的公式，然后根据需要将这些入场公式和出场公式按照一定的顺序组合成一个交易策略。实际执行中先满足条件的公式先执行。
示例：将下面的 3 个开仓条件和 2 个止损条件分别编写为 5 个独立的公式应用，并以下面的顺序组成一个策略。策略可以连续建仓，最大头寸为 2。
公式 1：收盘价大于上一根收盘价 0.2%，下一根 K 线开盘开多仓 1 手
公式 2：5、10 日均线金叉开多仓 1 手
公式 3：收盘价大于上一根 K 线最高价 2 跳，下一根 K 线开盘开多仓 1 手
公式 4：低于上一根 K 线收盘价 0.5%，多头全部平仓
公式 5：5、10 日均线死叉，多头全部平仓
分析：
上面 3 个开仓公式，任何 2 个先到达开仓条件，则由于仓位达到最大头寸 2，另外一个开仓公式将不会执行。
上面任何 1 个平仓公式达到平仓条件平掉所有的仓位，则另外一个平仓公式不会执行。
实现：
1.编写上面 5 个公式应用的代码。
公式应用 1：Average_Buy

```
Params
  Numeric percent(0.002); //大于收盘价幅度
Vars
Events
  OnBar(ArrayRef<Integer> indexs)
  {
    If(CurrentContracts<2 && (Close[1] > Close[2] *(1 + percent)))
```

```
    {
      Buy(1,Open);
    }
  }
```

公式应用 2: Dual_Buy

```
Params
  Numeric FastLength(5);
  Numeric SlowLength(20);
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
Events
  OnBar(ArrayRef<Integer> indexs)
  {
    AvgValue1 = AverageFC(Close,FastLength);
    AvgValue2 = AverageFC(Close,SlowLength);
    PlotNumeric("MA1",AvgValue1);
    PlotNumeric("MA2",AvgValue2);

    If(CurrentContracts<2 && AvgValue1[1] > AvgValue2[1])
    {
      Buy(1,Open);
    }
  }
```

公式应用 3: High_Buy

```
Vars
  Numeric MinPoint;          // 一个最小变动单位，也就是一跳
  Numeric MyEntryPrice;       // 开仓价格，
  Numeric AddSet(2);          // 大于最高价跳数
Events
  OnBar(ArrayRef<Integer> indexs)
  {
    MinPoint = MinMove*PriceScale;
    If(CurrentContracts<2 && (Close[1] > High[2] + AddSet*MinPoint))
    {
      Buy(1,Open);
    }
  }
```

公式应用 4: Average_Sell

```
Params
  Numeric percent(0.005); //小于收盘价幅度
Vars
Events
  OnBar(ArrayRef<Integer> indexs)
```

```
  {
    If(MarketPosition==1 && (Low < Close[1] *(1 - percent)))
    {
      Sell(0, Min(Close[1] *(1 - percent),Open));
    }
  }
```

公式应用 5：Dual_Sell

```
Params
  Numeric FastLength(5);
  Numeric SlowLength(20);
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
Events
OnBar(ArrayRef<Integer> indexs)
{
  AvgValue1 = AverageFC(Close,FastLength);
  AvgValue2 = AverageFC(Close,SlowLength);
  PlotNumeric("MA1",AvgValue1);
  PlotNumeric("MA2",AvgValue2);

  If(MarketPosition == 1 && AvgValue1[1] < AvgValue2[1])
  {
    Sell(0,Open);
  }
}
```

2. 在程序化交易工作区，创建并设置策略单元，按照上面的顺序添加以上 5 个公式应用。
3. 启动策略单元的程序化运行。

【注意】

用户需注意实盘测试与历史测试不一致的情况。

如果公式 4 和公式 5，在同一根 BAR 都能满足平仓条件。那么历史测试上因为公式顺序，先执行的是公式 4。而实际情况中，可能是公式 5 的平仓条件先满足，而先执行了公式 5。

1.2 公式编码规则

语言元素

交易开拓者公式平台的语言是 TradeBlazer Language，简称“TB 语言”。 它是一种高级语言，语法简洁易懂，介于 C++与 Pascal 之间，其语言元素包括保留字、函数、表达式和语句。

保留字：TB 公式中保留字有自己独特的意义或用途，主要是指一些功能关键字、系统函数以及数据类型等，例如：Params，Vars ， For……

函数：函数实现一定的功能，分为系统函数和用户函数，例如：sqr，abs……
表达式：由变量和操作符组成的运算式，例如：x=a+b;
语句：赋值语句，分支语句，控制语句等等，例如：if…else 语句

命名规则

1. 公式名称规则

不区分大小写;
不能超过 64 个英文字符;
公式之间不得重名;
公式名称不能出现字母、数字、下划线以外的其他字符，且不能以数字开头;
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。

2. 变量，参数规则

不区分大小写;
不能超过 64 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，且不能以数字开头;
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。

公式正文规则

公式中不区分字符的大小写，例如 Ab 与 ab 等效;
公式语句以“;”结束;
公式中代码正文不允许出现中文字符，使用英文双引号""引用起来的字符串除外;
公式正文字符因含义不同，以不同的颜色区分:
黑色 --- 用户自己声明的变量名或者参数名;
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）。

注释方式

对单行语句进行注释，可以在句首使用“//”将该行文字注释，也可直接在公式语句之后使用 “// ”对语句的意思或者想要标记的内容注释;

对多行语句进行注释，则可使用“/* ...*/”将整段文字进行注释。
也可以使用快捷键 Ctrl + B 进行块注释。
使用快捷键 Ctrl+U 对选中行取消行注释，Ctrl+L 对选中行添加行注释。
注释语句不参与公式主体的计算，允许出现中文字符。

保留字

TB 公式中保留字主要指功能关键字、系统函数以及数据类型等。命名公式简称以及参数、变量时，要避免使用保留字。
下面分类列举出系统主要的保留字。

运算符

类型	保留字
算术运算符	+ - * / % ^
关系运算符	> >= < <= == != <>
逻辑运算符	AND/&& OR/|| NOT/!

功能关键字

保留字	说明
Params	用该关键字宣告参数定义的起始，参数必须填写默认值。
Vars	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。
If	条件语句。
Else	条件语句。
Begin	用该关键字宣告程序主体的起始。
End	用该关键字宣告程序主体的结束。
For	循环语句。
To	循环语句。
DownTo	循环语句。
While	循环语句。
Break	循环语句。

Continue	循环语句。
True	真。
False	假。
Return	返回函数结果

数据源

保留字	说明
Data0-Data99**	支持多个数据源（具体数目及支持效率与本地电脑配置相关）。
Data	数据源数组(格式为：Data[i]，i 为整数，用法同上)。

函数：数据输出和交易指令

保留字	说明
PlotBool	输出布尔型值。
PlotNumeric	输出数值型值。
PlotString	输出字符串值。
UnPlot	取消指定位置的输出。
PlotDic	在当前 Bar 的输出信息中添加基础数据信息。
Alert	报警输出。
Buy	多头建仓操作。
Sell	多头平仓操作。
SellShort	空头建仓操作。
BuyToCover	空头平仓操作。
...	其他系统函数。

标点符号

TB 公式的语句支持使用标点符号，例如，";"标注一个语句结束；()创建规则的优先权等等。标点符号也是系统保留字，下表中列出了 TB 公式中所用到的标点符号及其含义：

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，引用以前的数据，或者数组中的元素。
{}	大括号	控制语句的起始。
.	点	扩展数据源的数据调用。

操作符

操作符是象征具体操作运算行为的符号，根据其作用的不同，分为三类：

数学操作符

操作符	说明
+	加
-	减
*	乘
/	除
%	求模
()	括号

这些数学操作符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至右。
例：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*Range/2 与最高价(High)的和。
例： (High+Low)/2;
先计算()中 High 与 Low 的和，然后将计算得到的和值除以 2，找到一个 Bar 的中间位置

关系操作符

操作符	说明
<	小于
>	大于
<=	小于等于

>=	大于等于
<>或!=	不等于
==	等于

应用关系操作符，可以对两个数值或字符串表达式进行比较。
如果两个操作数都是数值型，则按其大小比较
如果两个操作数都是字符型，则按字符的 ASCII 码值从左到右一一比较
汉字字符大于西文字符
关系操作符的优先级相同
例：Close>High[1];
判断当前 Bar 收盘价是否比前一个 Bar 最高价高
例："abcd" < "zyxw";
字符串的比较运算中，按照顺序依次比较的每个字符的 ASCII 码值，直至计算出结果即可。在这个例子中，首先比较“a”和“z”，字母"a"的 ASCII 值是小于"z"，所以该表达式的值为 True。

逻辑操作符

TB 语言支持三类逻辑操作符：与 AND(&&)，或 OR(||)，非 NOT(!)。
AND 逻辑操作符运算规则：

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

简单记忆法则：AND 运算只有两个条件都为 True 时，结果才为 True。
OR 逻辑操作符运算规则：

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

简单记忆法则：OR 运算只要有一个条件为 True，结果即为 True。
NOT 逻辑操作符运算规则：

表达式 1	NOT 表达式 1
True	False
False	True

例：Low < Low[1] AND Close > High[1];

当前 Bar 的最低价小于前一个 Bar 的最低价，并且当前 Bar 的收盘价高于前一个 Bar 的最高价时，这个表达式的计算结果才为 Ture。
例：High > 10 OR Vol > 5000;
当前 Bar 的最高价大于 10，或者成交量大于 5000，表达式的值都为 True。

逻辑操作符的优先级低于数学操作符和关系操作符。逻辑操作符也遵循括号优先的原则，如果没有括号，那么其运算顺序是从左至右。因此逻辑表达式中不同条件的前后顺序，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 为例，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;
因此，逻辑表达式的组合应该尽可能地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。

表达式

表达式的组成

TB 表达式由常量、变量、操作符、函数和圆括号按一定的规则组成，通过运算能得到结果，运算结果的类型由数据和操作符共同决定。

表达式的书写规则

乘号不能省略。
括号必须成对出现，均使用圆括号，可以嵌套，但必须配对。
表达式从左到右在同一基准上书写，无高低、大小之分。
例：sqr((3*x+y)-z)/(x*y)^4

不同数据类型的转换

操作数的数据类型应该符合要求，不同的数据应该转换成同一类型。

例：Commentary("Close = "+Text(Close));
Commentary 函数只能显示字符串类型的注释信息，因此，使用 Text()函数将 Close 的数值转换成字符再显示

优先级

同一表达式中，不同运算符的优先级是：
算术运算符 > 字符运算符 > 关系运算符 > 逻辑运算符
【注意】对于存在多种运算符的表达式，可增加圆括号改变优先级或使表达式更清晰。
例：MarketPosition == 1 And BarsSinceEntry >= 1
先进行关系运算 MarketPosition 是否等于 1，BarsSinceEntry 是否大于等于 1；然后再根据逻辑运算符 AND 计算两个表达式的的结果（此条件的含义为是否持有多仓且不是开仓 bar）
例：Low <= MyEntryPrice - StopLossSet*MinPoint
先计算 StopLossSet*MinPoint，然后计算 MyEntryPrice - StopLossSet*MinPoint，最后比较 Low 是否小于 MyEntryPrice - StopLossSet*MinPoint（此条件的含义为判断止损条件是否满足）

赋值语句

一个语句代表一个完全的指示或描述，语句中包含有保留字、操作符、符号。并且语句总是以";"作为语句结束的标志。
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;
Events
OnBar(ArrayRef<Integer> indexs)
{
  B := true;
  ...
}

Vars
  Numeric Value1;
Events
OnBar(ArrayRef<Integer> indexs)
{
  Value1 := (Close + Open)/2;
  ...
}

Vars
  String str;
Events
OnBar(ArrayRef<Integer> indexs)
{
```

```
  str ="It Is A Test!";
  ...
}
```

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 Series<Numeric>，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
  Series<Numeric> Value2;
Events
OnBar(ArrayRef<Integer> indexs)
{
  Value2[1] = (Close + Open)/2;
}
```

控制语句

计算机语言一般都有有三个基本的控制语句，顺序、选择和循环。

顺序类型直接以代码的顺序决定。选择类型和循环类型则需要有关键字来进行操作。

一、选择类型

选择类型的关键字有 if 和 else。

1.1 一个分支

```
If(逻辑条件)
{
    执行语句组;
}
```

注意：如果执行语句组只有一条语句时，{}可以省略。

举例：

```
Vars
  Numeric a;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  if(a>0 and a<=1)
    b=1;
}
```

1.2 两个互斥分支

```
If (逻辑条件)
{
    执行语句组 A;
}
Else
{
    执行语句组 B;
}
```

举例:

```
Vars
  Numeric a;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  if(a>0 and a<=1)
    b=1;
  Else
    b=2;
}
```

1.3 多个互斥分支

```
If (逻辑条件 1)
{
    执行语句组 A;
}
Else if(逻辑条件 2)
{
    执行语句组 B;
}
Else if(逻辑条件 3)
{
    执行语句组 C;
}
...
Else
{

}
```

举例:

```
Vars
  Numeric a;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  if(a>0 and a<=1)
    b=1;
  else if(a>1 and a<=2)
    b=2;
  else if(a>2 and a<=3)
    b=3;
  Else
    b=4;
}
```

1.4 条件语句的嵌套

```
If(逻辑条件 1)
{
    If(逻辑条件 2)
    {
        执行语句组;
    }
}
```

举例:

```
Vars
  Numeric a;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  if(a>0)
    {
    if(a<1)
      b=1;
    }
}
```

二、循环类型

循环类型的关键字有 while 和 for。配合的还有 to downto break continue。

2.1 while 类型

```
While (逻辑条件)
{
    执行语句组;
}
```

举例:

```
Vars
  Numeric a;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  While(b<3)
  {
    b=b+1;
  }
}
```

2.2 for 增序类型

```
for i=min to max
{
    执行语句组;
}
```

举例:

```
Vars
  Integer i;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  for i=0 to 3
  {
    b=b+1;
  }
}
```

2.3 for 降序类型

```
for i=max downto min
{
    执行语句组;
```

```
}
```
举例：
```
Vars
  Integer i;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  for i=3 DownTo 0
  {
    b=b+1;
  }
}
```

2.4 break

Break 是触发语句所在的最内层循环的终止。
举例：
```
Vars
  Integer i;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  for i=3 DownTo 0
  {
    b=b+1;
    if(b>=2)
      break;
  }
}
```
注释：当 b>=2 时，这个循环就提前终止了。

2.5 continue

Continue 是跳过语句所在循环剩下的执行语句，进入下一轮的执行。
举例：
```
Vars
  Integer i;
  Numeric b;
Events
OnBar(ArrayRef<Integer> indexs)
{
  for i=3 DownTo 0
```

```
  {
    if(b>=2)
      Continue;
    b=b+1;
  }
}
```

注释：当 b>=2 时，b=b+1 始终得不到执行，所以 b 最终=2。

2.6 循环的嵌套

```
For i=min to max
{
    For j=min to max
    {
        执行语句组;
    }
}
```

举例：

```
Vars
  Integer i;
  Integer j;
  Array<Array<Numeric>> a;
Events
OnBar(ArrayRef<Integer> indexs)
{
  for i=3 DownTo 0
  {
    for j=0 to 3
    {
      a[i][j]=i+j;
    }
  }
}
```

三、RANGE 对代码段的控制

```
Range[i:j]
{
    代码段 codes1;
}
```

如果使用 range[i:j]把代码段 codes1 括起来,则代表这一段代码只在数据源 datai 到 dataj 的数据源运行。I 要小于等于 j,比如 range[1:2]，则代表代码段 codes1 只在数据源 data1 和 data2 上运行。

具体例子如下：

```
Range[0:1]
```

```
  {
    AvgValue1 = AverageFC(Close,FastLength);
    AvgValue2 = AverageFC(Close,SlowLength);
    PlotNumeric("MA1",AvgValue1);
    PlotNumeric("MA2",AvgValue2);
  }
```

上面代码会在 Data[0]和 Data[1]上都运行，分别画出 Data[0]和 Data[1]的双均线。

注意：Range[i:j]内的代码运行说明如下：

a）没有指定数据源的代码，依次在 Data [i]到 Data [j]的 bar 上运行。

b）有指定数据源的代码，则在指定的数据源的 bar 上运行。

c）同一个公式内，可有多个并列的 Range 代码块，Range 内不支持 Range 嵌套。

d）Range[m:n]中，数据源起始位置 m 及结束位置 n 可以指定，也可以为变量和参数声明，类型为简单变量。

e）如果想要得到数据源的编号，则可以这样使用

```
Range[i=0:1]
{
    Commentary(“i=”+text(i));
}
```

1.3 数据类型

TBQuant 公式支持的数据类型有基本类型和扩展类型。总共可细分为 6 类，可以划分为 4 个级别。

数据类型	数据类型的级别	数据类型的内容
基本类型：普通类型	0	Numeric/String/Bool/Integer
基本类型：内嵌类型	0	Position,Order,Fill,Tick,Bar, CodeProperty
容器类型一级	1	Array,array<array>
容器类型二级	2	Dic,map,series
类型引用	3	Ref
类型修饰	3	Global，natural

数据类型的含义

1.1 基本类型：普通类型

Bool：布尔型，只包含 True 和 False 两个值。

Numeric：浮点型，包括数字、小数点、正负号。TB 语言没有日期、时间类型，也是用 Numeric 类型表示。

Integer：长整型。

String：字符串，包括所有西文字符、汉字、数字字符，必须用英文双引号""引用，例如"ab123”。

1.2 基本类型：内嵌类型

1） Tick-行情快照：对象类型

属性	类型	说明
dateTime	Numeric	时间，表示日期时间
symbol	String	合约代码
open	Numeric	开盘价
high	Numeric	最高价
low	Numeric	最低价
last	Numeric	最新价
limitUp	Numeric	涨停价
limitDown	Numeric	跌停价
preClose	Numeric	昨收价
preSettlement	Numeric	昨结价
volume	Integer	最新成交量
totalVolume	Integer	总成交量
openInt	Integer	持仓量
avgPrice	Numeric	实时均价(今日结算价)
totalTurnOver	Numeric	总成交金额(单位：元)
insideVolume	Integer	内盘
outsideVolume	Integer	外盘
todayClose	Numeric	今日收盘价
settlePrice	Numeric	当日结算价
status	Integer	状态(暂未启用)
preOpenInt	Integer	昨持仓
turnOver	Numeric	成交金额
rollover	Numeric	除权系数
bidask1	BidAsk	1 档行情

bidask2	BidAsk	2 档行情
bidask3	BidAsk	3 档行情
bidask4	BidAsk	4 档行情
bidask5	BidAsk	5 档行情

BidAsk: 对象类型

属性	类型	说明
bidP	Numeric	买价
bidV	Integer	买量
askP	Numeric	卖价
askV	Integer	卖量

2) Bar-Bar 数据 对象类型

属性	**类型**	**说明**
datetime	Numeric	开始时间
open	Numeric	开盘价
high	Numeric	最高价
low	Numeric	最低价
close	Numeric	收盘价
turnOver	Numeric	成交金额
volume	Integer	成交量
openInt	Integer	持仓量
rollover	Numeric	除权系数
lastDateTime	Numeric	最后一个 tick 的时间戳，表示日期时间

3）Position-持仓：对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
accountIndex	Integer	账户下标索引
symbol	String	合约代码
longCurrentVolume	Integer	多头当前持仓
longYesterdayVolume	Integer	多头剩余昨仓
longPreYesterdayVolume	Integer	多头期初持仓
longActiveVolume	Integer	多头未成交的报单净委托量
longActiveCloseVolume	Integer	多头未成交的报单平仓委托量
longCloseVolume	Integer	多头当前平仓量
longCanSellVolume	Integer	多头可平量
longMarketValue	Numeric	多头持仓市值
longAvgPrice	Numeric	多头均价（买均价）
longFloatPorfit	Numeric	多头浮动盈亏（买盈亏）
longUseMarginAmount	Numeric	多头占用的保证金
shortCurrentVolume	Integer	空头当前持仓
shortYesterdayVolume	Integer	空头剩余昨仓
shortPreYesterdayVolume	Integer	空头期初持仓
shortActiveVolume	Integer	空头未成交的报单净委托量
shortActiveCloseVolume	Integer	空头未成交的报单平仓委托量
shortCloseVolume	Integer	空头当前平仓量
shortCanCoverVolume	Integer	空头可平量
shortMarketValue	Numeric	空头持仓市值
shortAvgPrice	Numeric	空头均价（买均价）
shortFloatPorfit	Numeric	空头浮动盈亏（买盈亏）

shortUseMarginAmount	Numeric	空头占用的保证金
cancelTimes	Integer	合约的撤单次数

4) Order-委托 对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
accountIndex	Integer	账户下标索引
symbol	String	合约代码
orderId	Integer	报单索引
exchOrderId	String	报单编号（交易所）
createDateTime	Numeric	报单委托时间
volume	Integer	委托量
price	Numeric	委托价
fillVolume	Integer	成交量
fillAmount	Integer	成交金额
side	Integer	买卖方向，如: Enum_Buy、Enum_Sell
combOffset	Integer	开平标志，如：Enum_Entry、Enum_Exit、Enum_ExitToday
priceType	Integer	价格类型，如：Enum_PriceType_Limit、Enum_PriceType_Market、Enum_PriceType_OwnBest、Enum_PriceType_OpponentBest、Enum_PriceType_BestToCancel、Enum_PriceType_BestToLimit、Enum_PriceType_TotalFilledOrCancel、Enum_PriceType_FixPrice
hedge	Integer	投机套保，如: Enum_HedgeType_Speculatio、Enum_HedgeType_Arbitrage、Enum_HedgeType_Hedge、

		Enum_HedgeType_MarketMaker
status	Integer	报单状态，如：Enum_Declare、Enum_Declared、Enum_FillPart、Enum_Filled、Enum_Canceling、Enum_Canceled、Enum_Deleted
createId	Integer	报单源 ID，如：Enum_Trade_Source_Extra、Enum_Trade_Source_Manual、Enum_Trade_Source_Program、Enum_Trade_Source_TBPY、Enum_Trade_Source_Algo、Enum_Trade_Source_Helper、Enum_Trade_Source_Monitor、Enum_Trade_Source_ALL
createSource	String	报单源
cancelSource	String	撤单源
note	String	详细信息

5) Fill-成交 对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
accountIndex	Integer	账户下标索引
symbol	String	合约代码
fillId	String	成交索引
fillDateTime	Numeric	交易所成交时间
fillVolume	Integer	成交量
fillPrice	Numeric	成交价
orderId	Integer	报单索引
exchOrderId	String	报单编号（交易所）
createDateTime	Numeric	报单委托时间
volume	Integer	委托量

price	Numeric	委托价
side	Integer	买卖方向，如: Enum_Buy、Enum_Sell
combOffset	Integer	开平标志，如：Enum_Entry、Enum_Exit、Enum_ExitToday
priceType	Integer	价格类型，如：Enum_PriceType_Limit、Enum_PriceType_Market、Enum_PriceType_OwnBest、Enum_PriceType_OpponentBest、Enum_PriceType_BestToCancel、Enum_PriceType_BestToLimit、Enum_PriceType_TotalFilledOrCancel、Enum_PriceType_FixPrice
hedge	Integer	投机套保，如：Enum_HedgeType_Speculatio、Enum_HedgeType_Arbitrage、Enum_HedgeType_Hedge、Enum_HedgeType_MarketMaker
createId	Integer	报单源ID，如：Enum_Trade_Source_Extra、Enum_Trade_Source_Manual、Enum_Trade_Source_Program、Enum_Trade_Source_TBPY、Enum_Trade_Source_Algo、Enum_Trade_Source_Helper、Enum_Trade_Source_Monitor、Enum_Trade_Source_ALL
createSource	String	报单源

6) CodeProperty - 合约属性

属性	类型	说明
symbol	String	合约代码
bigCategory	Integer	商品大类，如: Enum_CategoryStocks、Enum_CategoryFutures、Enum_CategoryForexs、Enum_CategoryBonds、Enum_CategoryFunds、Enum_CategoryOptions、Enum_CategorySpotTrans、Enum_CategoryForeignFutures、Enum_CategoryForeignStocks
symbolName	String	商品名称

dealTimes	String	交易时段，格式为：09:00-10:15;10:30-11:30;13:30-14:15;14:30-15:00;
decDigits	Integer	小数点位数
priceScale	Numeric	最小变动加价(1/100,1/1000)
minMove	Integer	变动最小单位(1,20,50) 最小价格变动 = fPriceScale * nMinMove;默认为 1
contractUnit	Integer	合约单位 期货中 1 张合约包含 N 吨铜，小麦等
utcOffset	Integer	交易所世界标准时间偏移
marginRate	Numeric	保证金比率
marginMode	Integer	保证金计算方式
baseShares	Integer	最小委托量
handShares	Integer	一手量
incrementalShares	Integer	最小交易增量
currencyID	Integer	币种，如：Enum_CORMB、Enum_COUSA、Enum_COHK、Enum_COGBP、Enum_COJPY、Enum_COCAD、Enum_COAUD、Enum_COEUR、Enum_COCHF、Enum_COKOR
bigPointValue	Numeric	每个整数点的价值

1.3 容器类型一级

容器类型一级主要是增加一维数组 array 和二维数组 array<array>。

1.4 容器类型二级

容器类型二级有三种类型 series,map,dic。关于这三种类型会在本章的第三节《变量》部分进行详细介绍。这里只做简介。

1）series 序列变量

序列变量可以进行回溯得到当前 Bar 之前的 Bar 上的变量数据。

2）map 类型

Map 类型的基本结构是 Map<keys,values>。Map 是一种字典结构，每一个 key 对应一个内容。

Key 必须是不重复的。否则 Map[Key] = Value，原来的 value 会被覆盖。

3）dic 基础数据

Dic 是基础数据类型，为了方便使用基础数据而新增的数据类型。Dic 的使用也必须有 Key。

比如 Dic<numeric> dicvar1(key,True/False),key 是基础数据的关键字，True/False 基础数据是否持久

化数据库 True-持久化数据库 False-内存读写。（这个参数的默认值是 False）。Numeric 表明 dicvar1 是个数值型的基础变量。

1.5 ref 引用

Ref 一般用在函数的参数。具体应用场景分两类：1 是函数需要返回多个变量；2 是有些数据类型不支持函数的 return 返回，比如数组。

1.6 global 修饰

Global 修饰代表这个变量被全局化成为了全局变量。公式内全局数据类型定义的变量为公式内所有数据源所共享，并且不会每次运行都重置（即可延续之前的赋值）。全局变量在 oninit 之前就不会被重置。

1.7 natural 修饰

Natual 修饰和 global 修饰不能共用。Natural 修饰是针对数据源变量，变量加了 natural 修饰之后在 onready 事件之后就不会被重置。
Global 和 natural 的区别
Global 修饰
1.非数据源变量的不重置
2.oninit 开始一直不重置
Natural 修饰
1.数据源变量的不重置
2.onready 之后一直不重置
3.序列变量比较特殊：新 bar 来的时候，序列变量在当前 bar 还没有被用户赋值前，访问时是从上一根复制的。

数据类型组合的基本原则

1)、基本类型必须指定，是否扩展、引用、修饰，根据具体应用自行决定。

基本类型的 9 种，必须要选择其中一种。

2)、基本类型如果需要扩展、引用、修饰，按照级别优先级排序，级别高的在前面。

比如 global map<string,array<numeric>> var1;

3)、同级别不可以组合；

比如 ref 和 global 不会同时出现，numeric 和 order 也不会同时出现

4)、特殊情况说明

4.1 Series 不支持引用 ref 扩展,不支持 global 修饰。
4.2 Dic 不支持内嵌类型，不支持 global 修饰。
4.3 Map<keys,values>。key 的类型只能是 integer/string,value 的类型是基本类型和容器类型一级的组合，values 的可选类型总共 27 种。Map 整体有 54 种可选类型。
4.4 MapRef 只能用在 Defs 函数参数中用户函数暂不支持。
4.5 array<array<>>不支持 series 扩展。

数据类型的使用场景

数据类型	数据类型的级别	数据类型的内容	用户函数参数	Defs 函数参数	公式应用参数	变量
基本类型:普通类型	0	Numeric/String/Bool/Integer	可以	可以	可以	可以
基本类型:内嵌类型	0	Position,Order,Fill,Tick,Bar,CodeProperty	可以	可以	不可以	可以
容器类型一级	1	Array,array<array>	可以	可以	可以	可以
容器类型二级	2	Dic,map,series	除了 dic	除了 dic	不可以	可以
类型引用	3	Ref	除了 mapref	可以	不可以	不可以
类型修饰	3	Global, natural	不可以	不可以	不可以	可以

1.4 特殊数据

Bar 数据

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。Bar 数据也是我们口头上常说的 K 线数据。
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
Date	D	当前 Bar 的日期。
Time	T	当前 Bar 的时间,即当前 Bar 开始生成的时间

Open	O	当前 Bar 的开盘价。
High	H	当前 Bar 的最高价，Tick 时为当时的委卖价。
Low	L	当前 Bar 的最低价，Tick 时为当时的委买价。
Close	C	当前 Bar 的收盘价。
Vol	V	当前 Bar 的成交量。
OpenInt	无	当前 Bar 的持仓量。
CurrentBar	无	当前 Bar 的索引值，从 0 开始计数。
BarStatus	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。

Bar 的索引值

超级图表中的 Bar 类似于队列，每一个 Bar 都有其相应的位置，为了方便记录与查找，将其编号称为 Bar 的索引值，在 TB 公式中用系统函数 CurrentBar 引用该值。
图表左边第一个 Bar 的 CurrentBar 返回值为 0，向右其他 Bar 的则逐个递增。
注：可通过 PlotString("CurrentBar",Text(CurrentBar))；验证结果。

Bar 的状态值

TB 公式中用系统函数 BarStatus 表示当前公式所应用商品当前 Bar 的状态值，分为三种状态：图表最左边第一个 Bar,返回值 0;图表最右边最后一个 Bar,返回值 2;中间其他 Bar,返回值 1。注意:只有在 BarStatus 的返回值为 2 的情况下，公式指令才会对合约进行委托发单。

数据回溯

在 TradeBlazer 公式中有三种类型的数据回溯：变量回溯、参数回溯和函数回溯。
MaxBarsBack
在讨论回溯之前，我们需要了解公式中一个和数据回溯相关的系统函数 MaxBarsBack，该函数返回公式应用在当前商品执行所需的最大回溯 Bar 数目，即 CurrentBar 处序列变量回溯所缺少的 Bar 数。该函数初始值为 0，在公式应用从左到右每个 Bar 执行的过程中，会随着代码中对系统函数，序列变量等值的回溯调用而增加，所有 Bar 索引小于 MaxBarsBack 的输出都是不准确的，会被忽略掉。
当策略应用在多图层数据上时，任何一个图层遇到最大回溯，都会清除所有图层上的信号。

变量回溯

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。
要使用变量回溯，需要在变量的后面，使用中括号"[nOffset]"，nOffset 是要回溯引用的 Bar 相对于当前 Bar 的偏移值，该值必须大于等于 0，当 nOffset = 0 时，即为获取当前 Bar 的变量值。
例如，我们定义如下公式应用：

```
Vars
  Series<Numeric> MyVal;
Events
OnBar(ArrayRef<Integer> indexs)
{
  MyVal = Average(Close,10);
  PlotNumeric("MyVal",MyVal[3]);
}
```

以上公式定义数值型序列变量 MyVal，MyVal 等于收盘价的 10 个周期的平均值，然后将序列变量 MyVal 的前 3 个 Bar 数据输出。

以上公式 MyVal 的前 9 个数据因为需要计算的 Bar 数据不足，返回的值无效，从第 10 个 Bar 开始，MyVal 获取到正确的平均值，但是我们需要输出的数据是 MyVal[3]，即前 3 个 Bar 的数据，因此，直到第 13 个 Bar，有效的数据才会被输出。以上公式的 13 是该公式需要的最少引用周期数，如果将输出信息画到超级图表中，前 12 个 Bar 是没有图形显示的。建议用户在公式的最前面加上语句 if (CurrentBar < MaxBarsBack +) return，以避免使用越界的回溯数据。

参数回溯

TradeBlazer 公式支持的九种基本类型，在用户函数的参数定义中全部支持，在其他的公式中参数定义只支持三种简单类型。因此，关于参数的回溯问题，只对用户函数有效，以下我们举例说明用户函数序列参数的使用。

要使用参数回溯，需要在参数的后面，使用中括号"[nOffset]"，nOffset 是要回溯引用的 Bar 相对于当前 Bar 的偏移值，该值必须大于等于 0，当 nOffset = 0 时，即为获取当前 Bar 的参数值。

例如，我们定义一个用户函数 MyFunc，脚本如下：

```
Params
  Series<Numeric>  Price(0);
  Numeric      Length(10);
Vars
  Numeric      MyAvg;
  Numeric      MyDeviation;
Events
OnBar(ArrayRef<Integer> indexs)
{
  MyAvg = Summation(Price,Length)/Length;
  MyDeviation = MyAvg - Price[Length];
  Return MyDeviation;
}
```

以上的例子，对输入的 Price 我们求其 10 个周期的平均值，然后求出该平均值和 Price 的前 Length 个 Bar 的值之间的差值，将其返回。对于 Price[Length]这样的参数回溯引用，其实现原理和上节所描述的变量回溯引用基本一致。

函数回溯

函数回溯分为系统函数的回溯和用户函数的回溯。

系统函数中回溯的使用主要是针对 Bar 数据。比如我们需要获取两个 Bar 前的收盘价，脚本为 Close[2]；又或者我们需要获取 10 个 Bar 前的成交量，脚本为 Vol[10]。对于 Bar 数据的回溯是系统函数中最常用的，虽然也可以对行情数据和交易数据等进行回溯，但是大部分并无实质的意义，返回的结果和不回溯是一样的，因此，不推荐如此使用。

要对系统函数回溯引用，我们可以通过在函数名称后面添加"[nOffset]"获取其回溯值，nOffset 是要回溯引用的 Bar 相对于当前 Bar 的偏移值，该值必须大于等于 0，当 nOffset = 0 时，即为获取当前 Bar 的参数值。

带有参数的函数回溯，需要将"[nOffset]"放到参数之后，另外，无参数和使用默认参数的情况下，函数调用的括号可以省略。例如:Close[2]等同于 Close()[2]。

用户函数的回溯和系统函数原理基本一致，但考虑到系统的执行速度和效率等因素，目前，TradeBlazer 公式不支持对用户函数的回溯，如果您想要获取用户函数的回溯值，建议您将函数返回值赋值给一个序列变量，通过对序列变量的回溯来达到相同的目的。

如下面的脚本所示，取 Close 的 10 个 Bar 平均值的 4 个周期前的回溯值:

```
Vars
  Series<Numeric> AvgValue;
  Numeric     TmpValue;
Events
OnBar(ArrayRef<Integer> indexs)
{
  AvgValue = Average(Close,10);
  TmpValue = AvgValue[4];
  ...
}
```

叠加数据

交易开拓者的超级图表支持多周期多品种商品叠加的显示，当叠加的超级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。

假定，我们新建一个超级图表模块，其叠加的商品为：cu1810，cu1811 和 cu1812。此时，根据叠加操作的先后顺序，cu1810 为 Data0，cu1811 为 Data1，cu1811 为 Data2，在 TradeBlazer 公式中，我们可以通过 Data0.Open()，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。

如果省略对数据源的指定，则默认数据源为 Data0，直接使用 Open()表示 Data0.Open()。

叠加数据源的个数没有限制。100 个内的数据源可以使用保留字 Data0-Data99，更多的数据源可以使用 Data 数组来表示，譬如 Data[2000]。

行情数据

除了 Bar 数据之外，TradeBlazer 公式还可以支持实时行情数据的调用，行情数据是指当前商品最新的报价数据，该数据和 Bar 无关，行情数据的回溯没有意义。

行情数据只在最后 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)
{
    //调用行情数据函数
```

```
}
```
行情数据函数都按照以下格式命名 Q_XXXXX，比如 Q_Close，Q_BidPrice。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。

账户数据

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_OpenOrderContractNo。在调用行情数据的时候，需要判断当前是否已经启动自动化交易，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。

属性数据

TradeBlazer 公式还提供一组重要的属性数据，反映了该商品的一些基本信息，比如当前数据周期，买卖盘个数、保证金设置等信息。

1.5 参数与变量

参数

参数是一个预先声明的地址，用来存放输入参数的值，声明之后可以在公式中使用该参数名称引用其值。参数的值在公式的内部不能被修改，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，切换到参数工作区，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。

参数的类型

参数类型和用户函数、Defs 函数和公式应用有关。
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。
序列参数可以通过回溯获取以前 Bar 的值，具体介绍可参见参数回溯。

参数的声明

使用参数之前，必须对参数进行声明，TB 公式使用关键字"Params"进行参数声明，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);
```

例：

```
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;
  Series<Numeric>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;
  Array<Numeric> nArray(0);       //定义一维数组，默认为 0;
  Array<Array<Numeric>> nArrayParam(1);   //定义二维数组，默认值都为 1;
```

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

注：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。

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);
```

您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;
```

参数使用

在声明参数之后，我们可以在脚本正文中通过参数名称使用该参数，在使用的过程中要注意保持数据类型的匹配，示例如下：

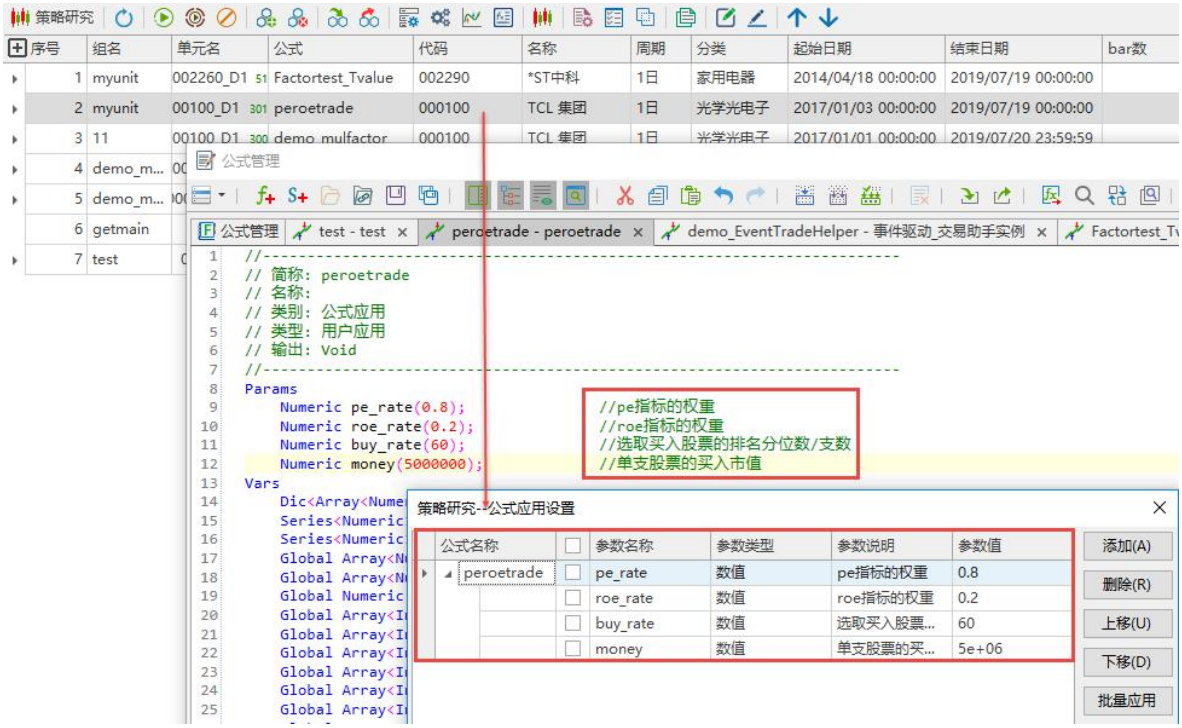
```
Params
  Series<Numeric> Price(1);
Vars
  Series<Numeric> CumValue(0);
begin
  CumValue = CumValue[1] + Price;
  Return CumValue;
end
```

在以上的公式中，首先定义了一个数值型序列参数 Price，并将其默认值设置为 1。接着定义了一个变量 CumValue。脚本正文中，将 CumValue 的上一个 Bar 值加上 Price，并将值赋给 CumValue，最后返回 CumValue。通过上述的公式可以看到，我们只需要调用参数名，就可以使用参数的值进行计算了，如果要对序列参数进行回溯，请参见参数回溯。

参数说明

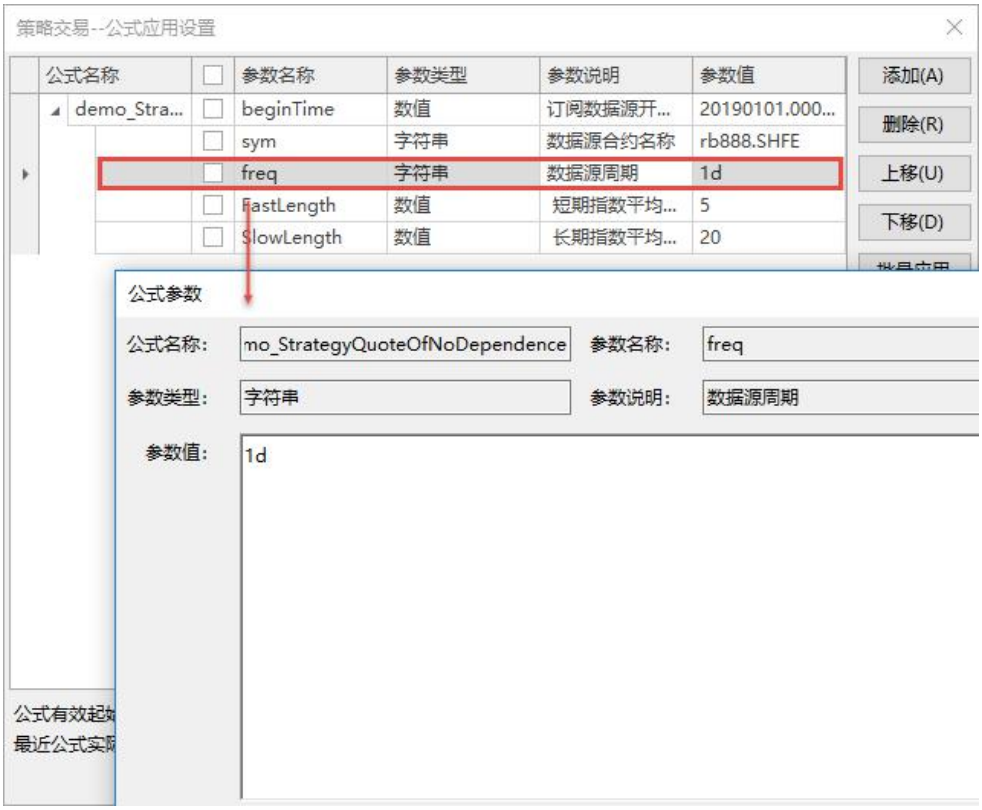
在定义参数变量的时候，可以用添加注释。而公式系统中参数的注释在打开公式应用设置的时候会进行展示。

比如下图 peroetrade 的公式中声明了 4 个变量，并且在代码中对 4 个变量进行了注释。那么在公式应用设置界面就有一列参数说明里面可以看到这些注释。



公式参数的修改

公式参数的修改，可以直接双击参数值，进行编辑。也可以对着参数所在行的其它字段进行双击，弹出参数编辑框，进行修改。



引用参数

TradeBlazer 公式的用户函数可以通过返回值，返回函数的计算结果，返回值只能是四种简单类型。当我们需要通过函数进行计算，返回多个值的时候，单个的返回值就不能满足需求了。在这种情况下，我们提出了引用参数的概念，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。因为引用参数的使用是没有个数限制，因此，我们可以通过引用参数返回任意多个值。引用参数不应含有初始值。

例如，用户函数 MyFunc 如下：

```
Params
   Series<Numeric> Price(0);
   NumericRef oHigher;
   NumericRef oLower;
Vars
   Numeric Tmp(0);
Begin
   Tmp = Average(Price,10);
   oHigher = IIf(Tmp > High,Tmp,High);
   oLower = IIf(Tmp < Low,Tmp,Low);
   Return Tmp;
End
```

以上代码通过两个数值型引用参数返回 10 个周期的 Price 平均值和最高价的较大值 oHigher，以及 10 个周期的 Price 平均值和最低价的较小值 oLower，并且通过函数返回值输出 10 个周期的 Price 平均值。在调用该用户函数的公式中，可以通过调用该函数获得 3 个计算返回值，示例如下：

```
Vars
  Numeric AvgValue;
  Numeric HigherValue;
  Numeric LowerValue;
Events
OnBar(ArrayRef<Integer> indexs)
{
  AvgValue = MyFunc(Close,HigherValue,LowerValue);
  ...
}
```

变量与常量

TB 公式中常量一般指的是常数，直接参与公式运算；变量指的是在公式运算过程中值会发生改变的量。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，可以对其赋值、计算。要对变量进行操作，直接使用变量名称即可。
变量的主要作用是存放计算或比较的结果，在之后的脚本中可直接引用，而无需重新计算过程。
在使用变量之前，必须对变量进行声明，TB 公式使用关键字“Vars”来进行变量声明，并指定变量类型。
变量声明对默认值没有硬性规定，可以赋值，也可以不赋值。变量的命名需要满足变量的命名规则。

变量的类型

TradeBlazer 公式支持上百种数据类型，但对于变量定义，引用类型是无效的。其余数据类型比如基本数据类型、序列数据类型、数组数据类型、公式内全局数据类型、基础数据类型可以用于公式变量类型。

变量的声明

在使用变量之前，必须对变量进行声明，TBQuant 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(默认值);
  变量类型 变量名 2(默认值);
  变量类型 变量名 3(默认值);
```

例：

```
Vars
  Series<Numeric> 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;
  Global Numeric MyVal3(0); //定义公式内全局数值型变量 MyVal，默认值为 0;
  Array<Integer> MyVal4; //定义数值型变量 MyVal4;
  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 声明，并且要放到公式的开始部分，在参数定义之后，正文之前。
【说明】
1.公式应用中，数据源数据类型的变量每个数据源一份，数据源可以直接引用该类变量，如下例所示。

```
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
  Global Numeric dataIndex;
events
OnBar(ArrayRef<Integer> indexs)
{
  For dataIndex=0 To DataSourceSize -1
  {
          Data[dataIndex].AvgValue1=Data[dataIndex].AverageFC(Data[dataIndex].Close,FastLeng
th);
          Data[dataIndex].AvgValue2=Data[dataIndex].AverageFC(Data[dataIndex].Close,SlowLeng
th);
  }
}
```

2.在函数中，变量只有一份，为公式内所有数据源所共享。

3.公式内全局数据类型 Global 的变量为公式内所有数据源所共享，不会在公式每次运行都重置（即可延续之前的赋值）。

```
Global Integer nVar;
Global Numeric dVar;
Global Bool bVar;
Global String sVar;
```

变量的默认值

在声明变量时，通常会赋给变量一个默认值。例如上例中的 0，False，"Test"等就是变量的默认值。如果某个变量没有赋予默认值，系统将会自动给该变量赋予默认值。数值型变量的默认值为 0，布尔型变量的默认值为 False，字符串的默认值为空串。

变量的默认值是在当公式在执行时，给该变量赋予的初值，使该变量在引用时存在着有效的值。在该公式每个 Bar 的执行过程中，该变量的默认值都会被重新赋值。

变量的赋值

变量声明完成之后，用户可以直接使用默认值，也可以根据实际需要在脚本中重新为变量赋值。

语法如下：

```
Name = Expression;
```

"Name"是变量的名称，“Expression”是表达式，该语句的意思是将表达式的结果赋值给变量，其中表达式的类型必须与变量的数据类型匹配，即若变量声明为数值型，则表达式必须为数值型的表达式。

例：

```
Value1 = Average(Close , 10); //将 Close 的 10 周期平均值赋值给变量 Value1
```

例：

```
Vars
  Bool KeyReversal(False);
```

```
events
OnBar(ArrayRef<Integer> indexs)
{
  KeyReversal = Low < Low[1] AND Close > High[1];
  ...
}//声明名为"KeyReversal"的逻辑型变量，然后又把计算的值赋给它。
```

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);
events
OnBar(ArrayRef<Integer> indexs)
{
  EntryPrc = Highest(High,10);
  If (MarkerPosition <> 1)
  {
    Buy(1,EntryPrc);
  }
}
```

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);
events
OnBar(ArrayRef<Integer> indexs)
{
  Con1 = High > Highest(High,10)[1];
  If(Con1)
  {
    Alert("New 10-bar high");
  }
}
```

【说明】

1. 公式应用中的变量需显式调用，如 Data0.Close。如不显示数据源则默认其数据源为 Data0。

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得到当前 Bar 之前的 Bar 上的变量数据。序列变量的

声明与简单变量一样，仅数据类型不同。

序列变量的定义

例:

```
Vars
  Series<Numeric> MyNum (0); //定义数值型序列变量，默认值为 0
  Series<Numeric> MyNum1 (0, 20); //定义数值型序列变量，第一个为默认值 0，后一个为最大回溯范围
20
  Series<Integer> MyNum (0); //定义整数型序列变量，默认值为 0
  Series<Bool>  MyBool (False); //定义布尔型序列变量，默认值为 False
  Series<String>  MyStr (""); //定义字符串型序列变量，默认值为空字符串序列变量
```

序列变量的赋值

和简单变量一样，可以对其赋予默认值。
序列变量定义之后，使用方式类似于简单变量。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，TB 公式在内部针对其回溯的特性作了很多特殊处理，并需要保存相应的历史数据。因此，和简单变量相比，序列变量的执行速度、占用内存空间等方面都作了一些牺牲。尽管可定义序列变量把它当作简单变量来使用，但强烈建议仅将需要回溯的变量定义为序列变量。
在指定条件下对某变量赋值，如果该变量是序列变量，它的值会传递下去，直至语句对其进行新的赋值。如果该变量是普通变量，则赋值仅针对条件满足的这个 Bar，其它 Bar 上的变量记录的仍是初始值，这是由 TB 公式的运行机制决定的。
通过 MyNum1 (0, 20)格式声明，第一个为序列变量默认值，第二个为设置的序列变量的回溯范围。相对于在公式菜单栏“序列变量范围”，有更高优先级。

例: 分别定义简单变量 aaa，序列变量 bbb，对其进行同样的赋值。部分公式代码如下:

```
Con1= CrossOver(ma1,ma2);
Con2= CrossUnder(ma1,ma2);
If(Con1)
{
  aaa = 1;
  bbb = 1;
}
If (Con2)
{
  aaa = -1;
  bbb = -1;
}
```

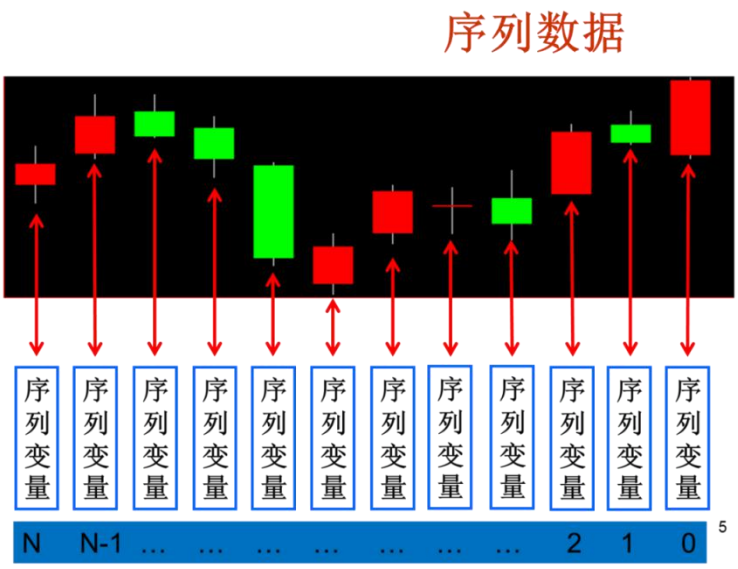
(ma1，ma2 分别为短期均线、长期均线值)

上图是简单变量和序列变量赋值示例，如图所示 aaa、bbb 这两个变量的赋值虽然条件相同，但其结果大不相同。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 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。 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]。

序列变量的赋值过程与逻辑

1. 序列变量在中间 Bar 上的赋值

序列变量是和图表中 K 线数据等长的变量，它的赋值过程是当程序在某个 Bar 运行时，仅仅将序列变量对应该 Bar 的值赋上，然后传递下去，直到变量再次被改变。例：序列变量 bbb[5]上曾经赋值 bbb[4]=1，之后 bbb 的值一直没有新的改变，直到当前 Bar，bbb=-1，即 bbb[0]=-1，bbb[3]、bbb[2]、bbb[1]的值为 1。
序列变量赋值之后，不允许再重新为之前对应 Bar 变量赋值。
为避免序列数据混乱，建议不要在条件语句和循环语句中为序列变量赋值。



2.序列变量或 Bar 数据取值时前面没有序列值，则返回无效值。

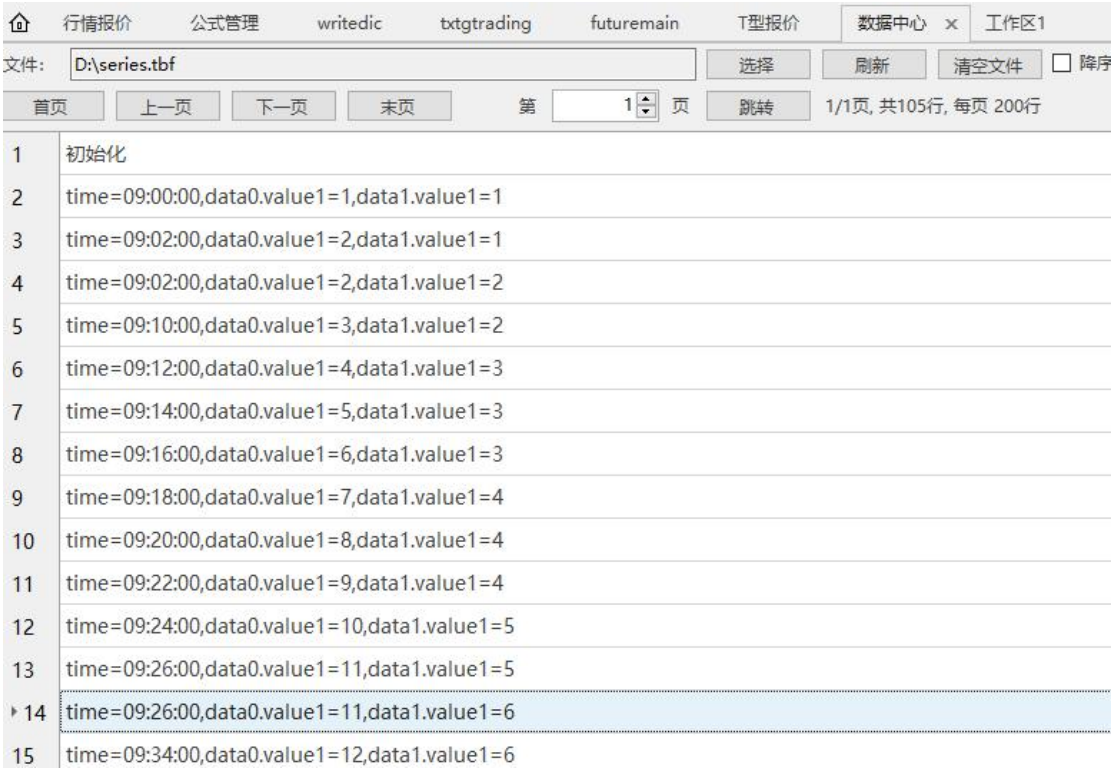
如下图，在 T1 时刻取 data0 的序列变量，则返回无效值。

3.数据源在当前时间上没有 bar 数据时

其序列变量不被赋值，但调用该序列变量时会返回数值。
比如我们叠加一个 2 分钟和 6 分钟的两个数据源的策略研究，加载如下策略：
Params

```
Vars
  Series<Numeric> value1;
events
OnBar(ArrayRef<Integer> indexs)
{
  If(CurrentBar == 0)
  {
    FileDelete("d:/series.tbf");
    FileAppend("d:/series.tbf","初始化");
  }
  data0.Value1 = data0.Value1 + 1;
  data1.Value1 = data1.Value1 + 1;
   FileAppend("d:/series.tbf","time="+TimeToString(Time)+",data0.value1="+Text(data0.Value1)+
",data1.value1="+Text(data1.value1));
}
```

运行之后，我们查看文档。图表中第一根 BAR 的时间是 9：00：00，第一个品种是 2 分钟图表，第二个品种是 6 分钟图表。
在 9：02：00 和 9：04：00，第一个品种有对应 K 线，第二个品种没有对应的 K 线。这时候第二个品种的序列变量是不会被改变的。只维持原来数值，可以被第一个品种调用。



4.当序列变量定义为下列格式时

序列变量会变成一种特殊的数据类型。Series<Numeric> MyNum1 (0, 1)这样的序列变量只能保留一个数值不能回溯。所以它在实时运行时没有办法同时保留上一根 BAR 最后的状态和每个实时 TICK 的运行状态。系统选择保留的是每个实时 TICK 的运行状态。这个时候序列变量已经是随着每个 TICK 更新，它的初始状态并不是来自于上一根 BAR，而是来自于上一个 TICK 的运行情况。可以把这样的序列变量理解为一个只在当前数据图层起作用的全局变量。

比如现在叠加了 3 个品种，那么每一个图层 i 都有一个序列变量 data[i].mynum1,它是一个只在第 i 个图层才会存在的全局变量。

```
Begin
    Mynum1=mynum1+1;
End
```

如果实盘运行经历了三根 BAR，那么每根 BAR 运行完的 mynum1 的数值是跟 TICK 相关的。

	初始值	BAR1	BAR2	BAR3
三个品种的 TICK 数		(2, 5, 6)	(3, 2, 4)	(1, 1, 1)
三个品种的 mynum1	(0, 0, 0)	(2, 5, 6)	(5, 7, 10)	(6, 8, 11)

全局变量

全局变量是一类较为特殊的数值型变量，保存的变量值不会因为 Bar 的改变而消失，它的作用范围是策略单元，策略单元内不同的公式应用和函数都可以访问。关掉策略单元后，所有保存的值才会消失。

1、全局变量和普通变量（数据源变量和公式内全局变量）的区别

TB 公式的执行机制是行情驱动，从左至右、从上至下的，即程序在图表中执行不仅仅运行一次，而是多次执行（历史行情中每一个 Bar 都会触发程序，实时行情中每一个 Tick 都会触发）。此时，程序每运行一次，都会对普通变量重新分配内存，进行初始化操作，所以普通变量无法保存上一个 Bar 中程序运行的结果；而全局变量是附着于策略单元的，它的内存全局分配，程序运行时不重新分配，它的值不会因为某次程序运行结束而消失，只有关闭公式加载的策略单元时，全局变量才清除。

2、全局变量的使用方式

全局变量有两种方式，一种是在变量定义里使用 global 来定义全局变量，另外一种是 TBQuant 继承原有产品的四个系统函数来实现的全局变量。

1）在 VARS 变量定义里使用 global 定义全局变量

```
Vars
  Global Numeric a;
  Global array<numeric> a;
```

使用这种方式定义全局变量，可以在 numEric，string，bool，integer 前面加 global 就可以实现。可以定义全局的普通变量，全局的数组，但是不能定义全局的序列变量。

Global Bool	Global Numeric	Global Integer	Global String
Global BoolArray	Global NumericArray	Global IntegerArray	Global StringrArray

2）使用系统函数的全局变量

这样的全局变量的初始值为 InvalidNumeric。使用之前需要为其赋初值，赋值之后全局变量的值不会随当前 Bar 变化而变化，只有再次对全局变量赋值才会改变。

通过系统函数 SetGlobalVar 和 SetGlobalVar2 设置某个数值索引或字符串索引的全局变量值。暂时只支持存储数字。

通过系统函数 GetGlobalVar 和 GetGlobalVar2 获取指定数值索引或字符串索引的全局变量值。

函数名	含义
SetGlobalVar	设置某个索引的全局变量值
SetGlobalVar2	设置某个字符串索引的全局变量值
GetGlobalVar	获取某个索引的全局变量值
GetGlobalVar2	获取某个字符串索引的全局变量值

【说明】在并行计算中不能使用全局变量。

局部变量

定义：

局部变量是指除了在 VARS 下面直接定义变量以外的地方，都可以定义变量，这些变量叫局部变量。

使用规则：

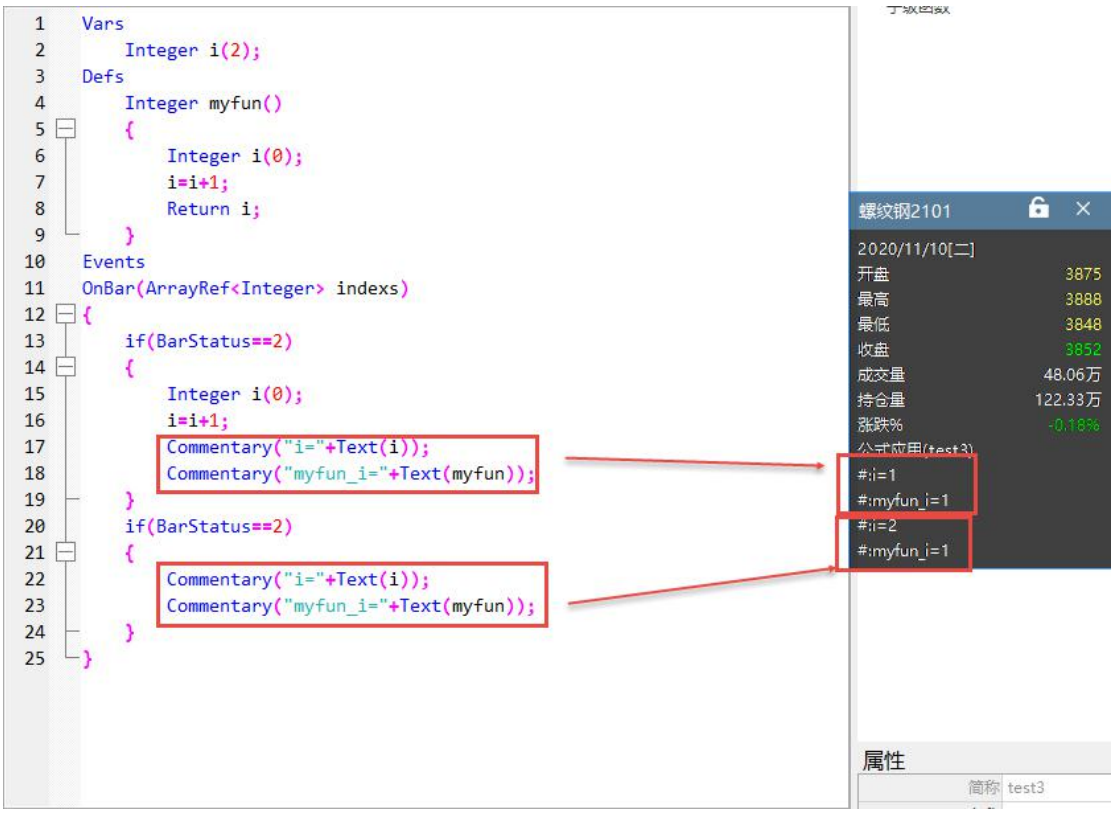
局部变量的作用域只在于最近的一个{}内。

局部变量和 VARS 变量，以及局部变量之间都可以重名。

当代码中遇到重名的变量时，先确定{}内有没有局部变量定义，没有的话这个变量就是 VARS 变量。

使用案例：

具体的使用案例可以参照下面的案例，定义了三个变量 i。一个 VARS 变量，一个函数内的局部变量，另外一个是正文{}内的局部变量。



完整代码：

```
Vars
    Integer i(2);
Defs
    Integer myfun()
    {
        Integer i(0);
        i=i+1;
        Return i;
    }
Events
OnBar(ArrayRef<Integer> indexs)
{
    if(BarStatus==2)
    {
        Integer i(0);
        i=i+1;
```

```
        Commentary("i="+Text(i));
        Commentary("myfun_i="+Text(myfun));
    }
    if(BarStatus==2)
    {
        Commentary("i="+Text(i));
        Commentary("myfun_i="+Text(myfun));
    }
}
```

基础数据

基础数据从广义上理解是交易所行情之外的其它数据。比如股票的财务报表数据，比如商品的分类数据，比如连续合约的合约切换等信息。
对应基础数据需要定义基础数据的变量。

1、变量定义方式

Dic<数据类型> 变量名 (String 键名，bool 是否持久化，String 关联标的)，后面两个可以不写
1）键名和关联标的可以从系统一数据中心查询，特别注意要区分大小写。比如"TB_FinanceInd_F014N" 不能改为"tb_FinanceInd_F014N","symtem",不能改为"Symtem"。
2）如果不持久化，软件重启，这个基础数据就清除了。
3)string 关联标的如果不写，那么就自动关联读取时候的数据图层，如果这个数据图层有基础数据就返回，否则无效值。
4）基础数据是系统类型的，关联标的是"symtem",必须要填写。
比如下面的代码:

```
Vars
  Dic<Numeric> roe("TB_FinanceInd_F014N",False); //读取财务指标 ROE
```

Dic<Numeric>是变量的类型，roe 是用户自己指定的变量名，"TB_FinanceInd_F014N"是 TB 基础数据库中财务指标 ROE 的内部键名，False 代表不持久化，最后一个参数 string 关联标的，省略没有给.

2、函数调取基础数据

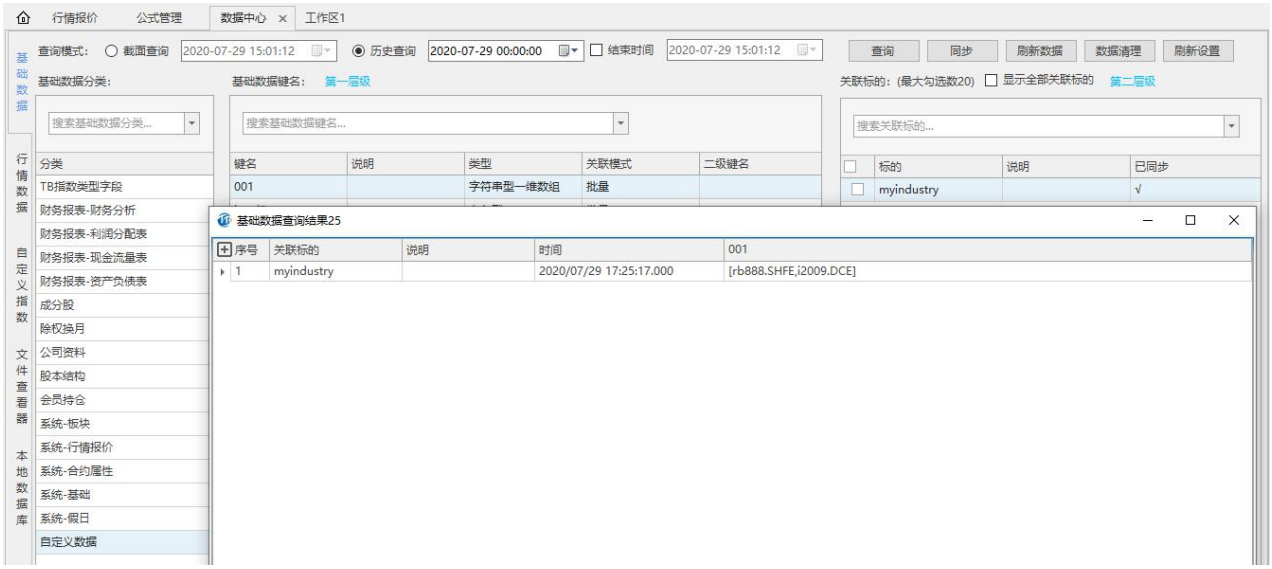
如果在 TBQuant 里面则使用 GetDicValue();
void GetDicValue(String name, String symbol, Numeric time, 接收基础数据的变量 rValue)
name 是基础数据的键名，symbol 是关联标的，time 是时间，最后一个 rvalue 是接收基础数据的变量。
如果读取基础数据不清楚这些参数的具体数值，可以点开系统一一数据中心进行搜索查看。具体使用可以看《TBQuant 软件使用》的【数据查询】章节。

3、基础数据写入

如果用户想自定义基础数据，数据怎么写入。

1）如果在 TBQuant 里面则使用 SetDicValue()；

bool SetDicValue(String name, String symbol, Numeric time, Numeric rValue, Bool isPersistence = false)
name 是基础数据的键名，symbol 是关联标的，time 是时间，最后一个 rvalue 是写入基础数据的变量。
特别的如果用户向定义一个不受关联标的限制的基础数据，也就是可以在所有数据图层都能用的基础数据。
那么 symbol 这一项最好写成是一个用户自定义的字符串。
比如，SetDicValue("001","myindustry",SystemDateTime(),subarray,True)；subarray 的内容是一个字符串数组。
那么在基础数据中心，我们可以查询到如下结果。



2） 在 TBQuant 里面数据写入，也可以通过直接赋值的方式。

给定义好的基础数据进行写入。
比如下面的 demo:

```
Params
Vars
  Dic<Bool> boolDs("boolDs",False);
  Dic<Integer> integerDs("integerDs",False);
  Dic<Numeric> numericDs("numericDs",False);
  Dic<String> stringDs("stringDs",True);
  Integer mybarcount;
Events
OnBar(ArrayRef<Integer> indexs)
{
  mybarCount = CurrentBar();
  if(barCount % 2 == 0)
  {
    boolDs[0] = True;
```

```
    integerDs[0] = barCount;
    numericDs[0] = barCount / 10;
    stringDs[0] = "str_" + Text(barCount);
  }
}
```

数组

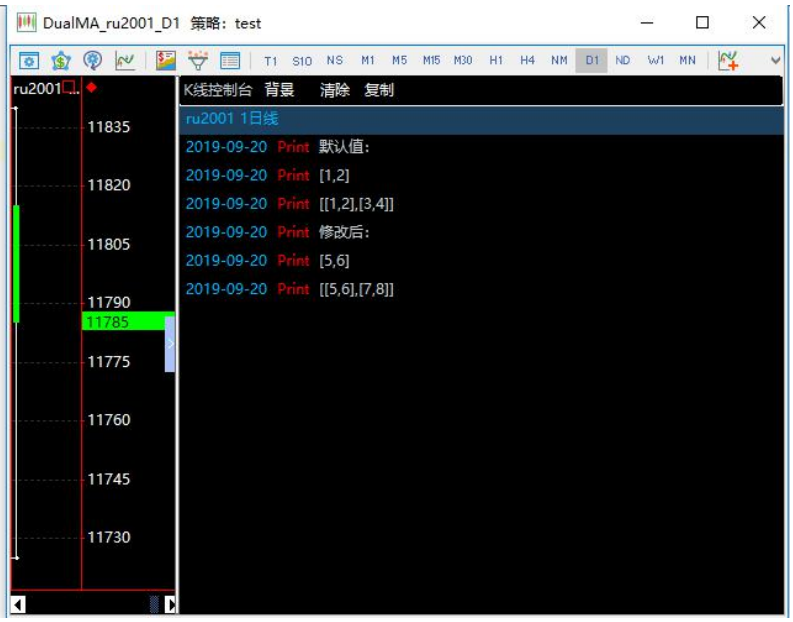
在 TBQuant 里面我们支持一维数组和二维数组对基础数据类型的容器扩展。

1、数组的定义和赋值

比如下面的例子，定义了一个一维数组和一个全局的二维数组。并且给其中的元素进行了赋值。数组参数的默认值可以直接使用列表形式赋值。

```
Vars
  Array<Numeric> na1([1,2]);
  Array<Array<Numeric>> na2([[1,2],[3,4]]);
Events
OnBar(ArrayRef<Integer> indexs)
{
  if(BarStatus==2)
  {
    Print("默认值: ");
    Print(TextArray(na1));
    Print(TextArray(na2));
    na1=[5,6];
    na2=[[5,6],[7,8]];
    Print("修改后: ");
    Print(TextArray(na1));
    Print(TextArray(na2));
  }
}
```

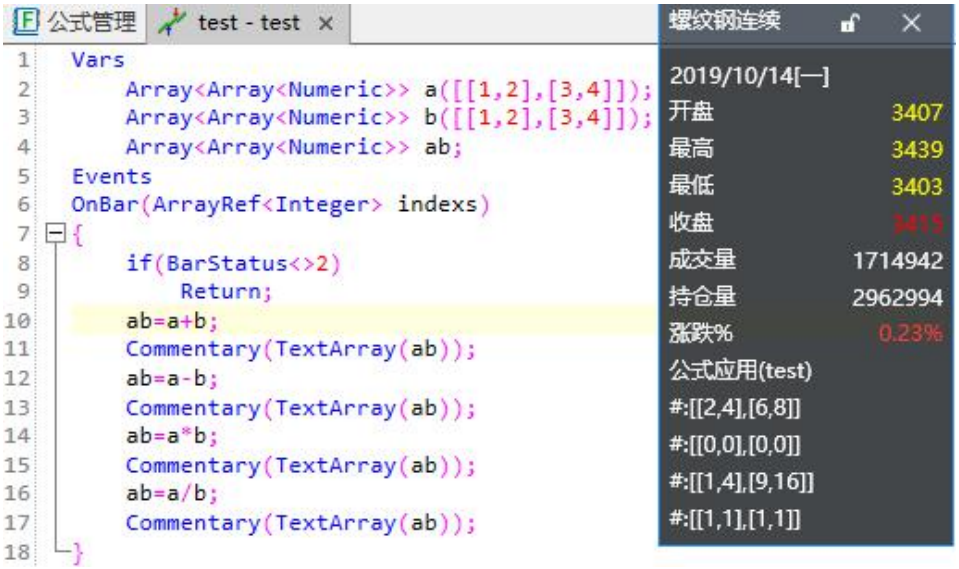
```
Vars
    Array<Numeric> na1([1,2]);
    Array<Array<Numeric>> na2([[1,2],[3,4]]);
Events
OnBar(ArrayRef<Integer> indexs)
{
    if(BarStatus==2)
    {
        Print("默认值: ");
        Print(TextArray(na1));
        Print(TextArray(na2));
        na1=[5,6];
        na2=[[5,6],[7,8]];
        Print("修改后: ");
        Print(TextArray(na1));
        Print(TextArray(na2));
    }
}
```



注意：数组的定义不需要指定数组的大小，数组大小会自动根据代码对数据元素的使用而扩容。

2、数组的运算

数组的运算可以有函数来实现。而基本的四则运算则可以直接使用运算符实现。




```
Vars
    Array<Array<Numeric>> a([[1,2],[3,4]]);
    Array<Array<Numeric>> b([[1,2],[3,4]]);
    Array<Array<Numeric>> ab;
Events
OnBar(ArrayRef<Integer> indexs)
{
    if(BarStatus<>2)
        Return;
    ab=a+b;
    Commentary(TextArray(ab));
    ab=a-b;
    Commentary(TextArray(ab));
    ab=a*b;
    Commentary(TextArray(ab));
    ab=a/b;
    Commentary(TextArray(ab));
}
```

3、数组的相关函数

ArrayClear：数组的全部删除。
ArrayClear：删除二维数组的全部内容。
ArrayCompare：对两个数组进行比较。
ArrayCompare：对两个二维数组进行比较。
ArrayCopy：复制数组的内容。
ArrayCopy：复制二维数组内容。
ArrayEqual：检验两个数组是否相等。

ArrayEqual: 检验两个二维数组是否相等。
ArrayErase: 数组的元素删除。
ArrayErase: 二维数组的指定元素删除。
ArrayInsert: 数组的单个数据插入。
ArrayInsert: 二维数组插入单个元素。
ArrayInsertRange: 数组的多个数据插入。
ArrayInsertRange: 二维数组插入多个元素。
ArraySort: 对数组进行排序。
ArraySort: 二维数组排序。
ArraySwap: 交换数组的内容。
ArraySwap: 交换二维数组的内容。
GetArraySize: 获取数组的大小。
Na1Sort: 一维数组排序带下标。
Na1Sort2: 一维数组排序带下标，原数组不变。
Na1Max: 一维数组的最大值。
Na1Min: 一维数组的最小值。
Na2Add: 二维数组的加法。
Na2Dev: 二维数组的减法。
Na2Mul: 二维数组的乘法。
Na2Inv: 二维数组的求逆。
Na2Trans: 二维数组的转置。
Na2Abs: 二维数组的绝对值。
Na2Sum: 二维数组的元素求和。
SetArraySize: 设置数组的大小和初始值。
SetArraySize: 设置二维数组的大小和初始值。
SortIds: 一维数组排序带双重下标。

Map

Map 是一种字典类型的数据。Map<keys,values>的结构提供了不重复的 key 与数值的一一对应的存储。
Key 支持的类型只能是 integer/string,value 的类型是基本类型和容器类型一级的组合， value 的可选类型总共 27 种。Map 整体有 54 种可选类型。

1）Map 的定义

Map<keytype,valuetype> mapvar;
比如 Map<integer,string> mapvar;这样就定义了一个 key 为整数类型,value 为 string 类型。

2）Map 的增加元素

Map[newkey]=newvalue;
比如使用上面定义的 Map Mapvar[1]=“china”; Mapvar[2]=“usa”;

3）Map 的读取元素

Map[key]
比如 Commenta (Mapvar[1]) ; 内容为“china”。

4）Map 的删除元素

MapErase(Map,delkey)
比如 MapErase(Mapvar1,2),就删除了 Map 的内容 Mapvar[2]=“usa”。

5）Map 的大小获取

GetMapsize(Map)
比如沿用上面 2）的结果，GetMapsize(mapvar1),这时候长度为 2。

6）Map 的所有 key 的读取

GetMapkeys(Map,array<keytype>)
比如先定义一个整形数组 vars array<integer> arr1;
沿用上面 2）的结果，GetMapkeys(Mapvar1,arr1)。那么就读取了 Mapvar1 的 keys,存储到了 arr1 的数组里面。Arr1[0]=1;arr1[1]=2;

7）Map 的关键字是否存在的查询

Mapcontain(Map,findkey)
比如沿用上面 4）,Mapcontain(mapvar1,2)这是返回的结果是 False。因为已经删除了这个内容。

8）Map 的指定元素查找。

Bool MapFind(MapRef mapValue, Key, value)
比如 MapFind(mapvar1,1,var2)。可以查找 mapvar1 中是否有键值为 1 的元素,并把对应的数值赋值给 var2。

9）Map 删除映射表的全部内容。

MapClear(MapRef mapValue)

1.6 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 Integer（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 Integer 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、账户函数、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，详细说明参见附录。

输出函数介绍

TB 的公式应用提供 PlotNumeric、PlotBool、PlotString 三个函数在图表上输出数值、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。TBQuant 又增加了 plotauto, plotdic, plotKline, upplot, print 等函数的输出功能。
例：分析系统提供的 MA 移动平均线（公式代码如下图所示）

公式管理 MA - 移动平均线 ×

```
1  //---------------------------------------------------------------------------
2  // 简称: MA
3  // 名称: 移动平均线
4  // 类别: 公式应用
5  // 类型: 内建应用
6  //---------------------------------------------------------------------------
7  Params
8      Numeric Length1(5);
9      Numeric Length2(10);
10     Numeric Length3(20);
11     Numeric Length4(30);
12 Begin
13     PlotNumeric("MA1",AverageFC(Close,Length1));
14     PlotNumeric("MA2",AverageFC(Close,Length2));
15     PlotNumeric("MA3",AverageFC(Close,Length3));
16     PlotNumeric("MA4",AverageFC(Close,Length4));
17 End
18 //---------------------------------------------------------------------------
19 // 编译版本 GS2010.12.08
20 // 版权所有 TradeBlazer Software 2003 - 2010
21 // 更改声明 TradeBlazer Software保留对TradeBlazer平
22 //      台每一版本的TradeBlazer公式修改和重写的权利
23 //---------------------------------------------------------------------------
```

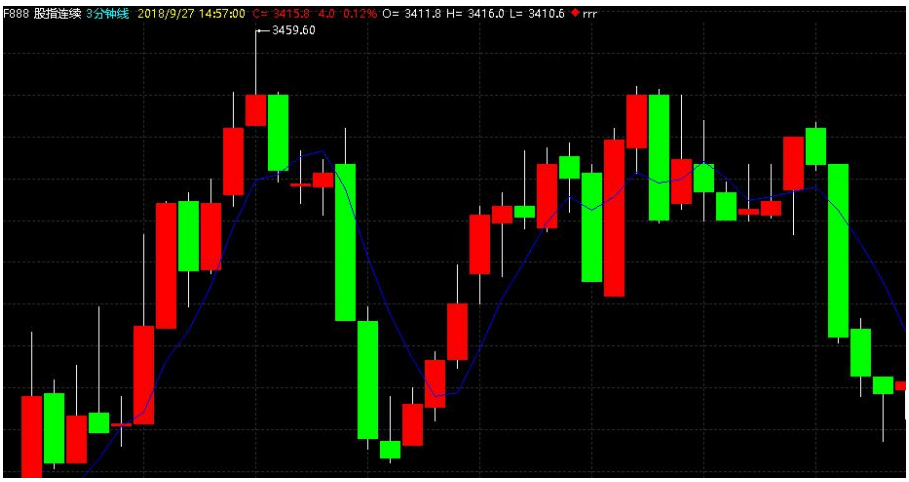
MA 移动平均线公式定义了 4 个参数，存储不同的周期值，分别赋默认值 5, 10, 20, 30。公式的代码段只有 4 条输出不同周期均线的语句，使用了 PlotNumeric 函数进行数值的输出，AverageFC 函数求出不同周期的平均值。公式执行效果如下图所示：

PlotNumeric

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)
参数说明：
Name 输出值的名称字符串，可以为中文、英文、数字或者其它字符；
Number 输出的数值；
Locator 输出值的定位点，默认值为 0，表示输出单点，否则输出连接两个值线段；
Color 输出值的显示颜色，默认值为-1，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 0，表示当前 Bar。
例 1：PlotNumeric ("Close",Close);
输出 Close 的值，如下图所示。

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），如下图所示。

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，如下图所示。



【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者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

PlotBool-----在当前 Bar 输出一个布尔值。在图表上的表现为指定位置显示圆脸图标，布尔值为真显示一个绿色的笑脸图标，布尔值为假则显示一个红色的哭脸图标。

语法：

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

参数说明：

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，默认值为-1，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 0，表示当前 Bar。

例 ：PlotBool ("con",con,High)；

在 Bar 的最高价位置输出条件 con 的布尔值，如下图所示。

PlotString

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)
参数说明:
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，默认值为-1，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 0，表示当前 Bar。
例 :
PlotString ("Bear Market","Bear Bear",High,Yellow);
在 Bar 的最高价位置输出一个黄色的字符串 Bear Bear，如下图所示。

【注意事项】
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”，该字符串的意义是给将要输出的值命名，支持中、英文字符。在同一个公式应用里，同一类型的输出值需要分别命名，不能重复使用相同的名称。
输出颜色的选择
PlotNumeric、 PlotBool、 PlotString 这三个输出函数，从右向左数第二个参数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，也可以使用默认值，然后在公式属性里对颜色进行自定义设置。
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、dakmagenta 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
数据输出与图表中 Bar 的数量同齐
图表中每个 Bar 均有数值、布尔或字符串的输出。数值的输出涉及到计算，有时在某些 Bar 上可能为无效值。如: PlotNumeric(“avgvalue”,averageFC(close,5));该公式在图表数据的前 4 个 Bar 上都是无效值，第 5 个 Bar 之后每个 Bar 才会有计算得出的有效平均数值的输出。
注意: 在图表整个 Bar 数据序列上，可能因为计算、条件等因素，从而在 Bar 数据中输出无效值，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，输出数据之前可在公式中加上判断语句，保证在非无效值的情况下才输出数据。

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

条件 Bar 下的数据输出

在指定条件下输出数据，不符合条件的 Bar 上则不会有任何输出。

例：判断无效值则不输出

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

例：对某种 K 线型态的标识或者突破点的标注。

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

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。如下图所示：

偏移 N 个 Bar 的输出

三个输出函数最后一个参数均是 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。

例：比较两种不同偏移的输出结果：

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

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

注意：第一种写法在历史分析中相当于使用了未来数据，请慎重考虑此写法是否符合实际需求，此处仅作为比较偏移取值不同的显示效果示例。实时行情中，每一次的运算只计算最新 Bar 上的 Tick，所以当最新 K 线数据满足条件后，只有再手工刷新图表，才会对历史 K 线上按条件进行画线。

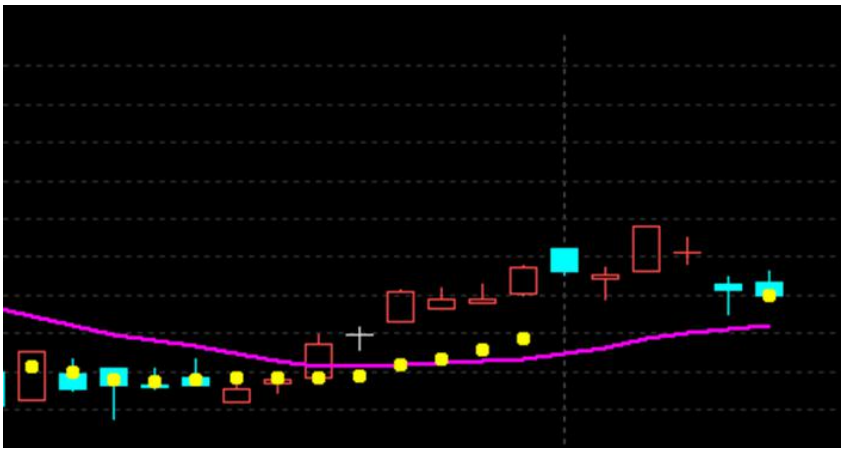
第二种写法在最新 K 线出来之前，无法在图表上画出前一个 Bar 所输出的数据线。因为交易开拓者的图表上，画线的前提条件是有 Bar 数据。

UnPlot

UnPlot-----删除输出函数曾经输出的值。
语法：
Unplot(String Name,Integer BarsBack =0)
参数说明：
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 0，表示删除当前 Bar 曾输出的值。
例：

```
Params
    Numeric length1(10);
    Numeric length2(20);
Vars
    Numeric ma1;
    Numeric ma2;
    Numeric i;
events
OnBar(ArrayRef<Integer> indexs)
{
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);
    }
}
```

本例的功能是输出两条移动平均线，把其中 MA1 均线的最后 6 个 K 线上的输出值删除，并在后一个 K 线输出一个新值。结果如下图所示：



【问题】为什么使用 PlotNumeric 输出线条，在属性中设置了线型设置，实际显示却是混乱的

例: If (condition1)

{Plotnumeric("t1",L[3],L[3],-1,3);//公式属性里线型选柱状图

}

If (condition2)

{Plotnumeric ("t2","2",0-50);//公式属性里线型选线

}

表现: Bar 满足 condition1 但不满足 condition2，图表中在 L[3]显示黄粗圆点；如果 t2 公式属性里线型选点，黄粗圆点变成了红圈方白点。 *(线型中还有其他设置，这里的意思就是线型没有按照设置的显示，具体表现情况不尽相同)*

【回答】这个问题的关键在于线型不是直接输出，而是满足某种条件输出。这样，在图表上输出线型的顺序与属性里显示的线型设置的序列可能不同，于是出现了混乱情况。遇到这种问题，建议将一定会在每个 bar 上都出现的线型写在前面,需要判断条件才输出的写在最后。如果条件输出的结果不能保障出现的顺序，可以改成几个不同的公式来执行，实现不同线型的需求。

PlotKline

PlotKline-----在当前 Bar 输出一个 K 线。可用于在子图中输出价差 K 线(需在公式属性中，设置“显示方式”为子图)。

语法:

PlotKline (Numeric Open, Numeric High, Numeric Low, Numeric Close)

参数说明:

Open 输出 K 线的开盘价的值;

High 输出 K 线的最高价的值;

Low 输出 K 线的最低价的值;

Close 输出 K 线的收盘价的值。

例:

例 1: PlotKline(100, 110, 93, 107);

输出 RSI 的值。

例 2: PlotKline(Open-100,High-100,Low-100,Close-100);

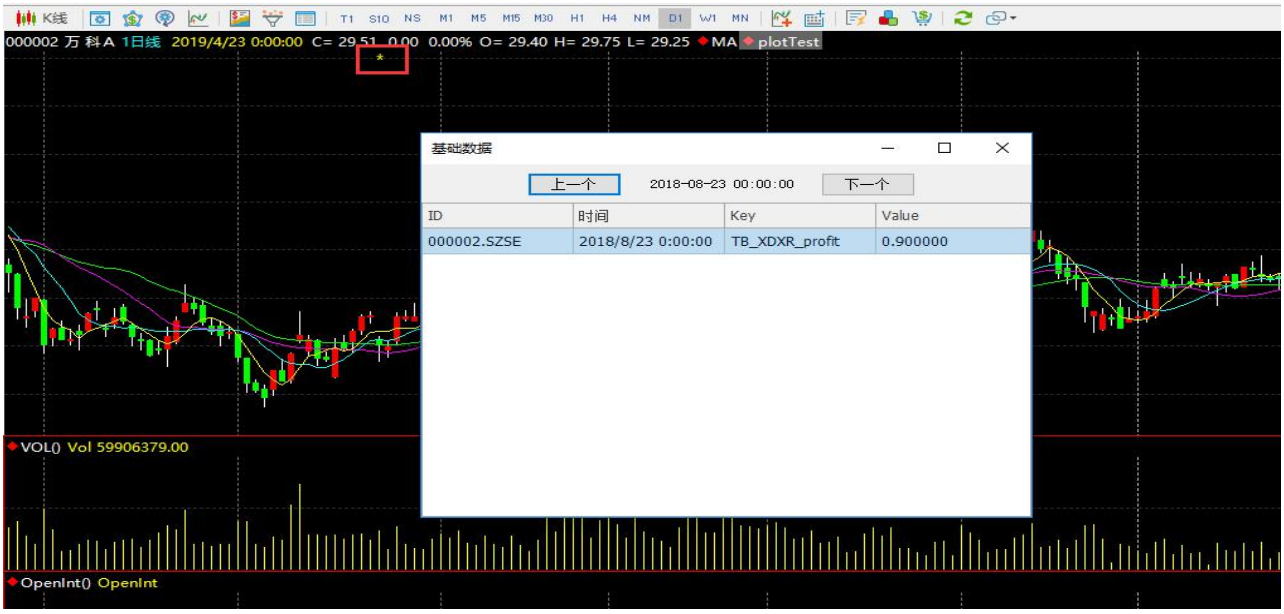
在当个 k 线 100 个点下，输出相应 K 线。

PlotDic

PlotDic-----在当前 Bar 的输出信息中添加基础数据信息。
参数说明:
DicRef<Numeric>、DicRef<Array<Numeric>>、DicRef<Array<Array<Numeric>>>、 DicRef<Integer>、DicRef<Array< Integer >>、DicRef<Array<Array< Integer >>>、DicRef<Bool>、DicRef<Array< Bool >>、DicRef<Array<Array< Bool >>>、DicRef<String>、DicRef<Array< String >>、DicRef<Array<Array< String >>>需要输出的基础数据引用
Integer dIconType(-1); //输出值的图形类型
Numeric dLocator(0); //输出值的定位点
Integer lColor(-1); //输出值的显示颜色，默认表示使用属性设置框中的颜色
Integer lBack(0); //当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
例:

```
Vars
    Dic<Numeric> xdxr_price("TB_XDXR_price");
    Dic<Numeric> xdxr_profit("TB_XDXR_profit");
    Dic<Numeric> xdxr_ration("TB_XDXR_ration");
    Dic<Numeric> xdxr_scrip_issue("TB_XDXR_scrip_issue");
events
    OnBar(ArrayRef<Integer> indexs)
    {

       if(BarStatus == 1)
       {
          xdxr_price[0];
          xdxr_profit[0];
          xdxr_ration[0];
          xdxr_scrip_issue[0];
       }
      PlotDic(xdxr_price);
      PlotDic(xdxr_profit);
      PlotDic(xdxr_ration);
      PlotDic(xdxr_scrip_issue);
    }
```

在合约的 k 线图表上加载该公式，有基础数据的 bar 上显示*标记，点击*，弹出该 bar 对应的基础数据窗。



Plotauto

在 TBQuant 里面为了使得输出画图函数设置更加方面，也为了使得输出函数名字统一，所以使用新的输出函数 plotauto 替代原来三个函数。并且在原来三个函数的参数基础上增加了 plotauto 函数的参数（类型，风格，宽度等）。但是不同的数据类型必须使用不同的参数，这个需要特别注意。

对应 plotnumeric 类型的输出：plotauto 的参数

参数说明：

String strName; //输出值的名称，不区分大小写

Numeric dVal; //输出的数值

Numeric dLocator(0); //输出值的定位点

Integer lLineType(-1);//设定画线类型 Enum_Dot、 Enum_Line、 Enum_Bar、Enum_Cross 枚举函数指定 默认为属性框中设定

Integer lLineStyle(-1); //设定画线风格 Enum_Solid、Enum_Dash、Enum_Broken、Enum_Dash_Dot 枚举函数指定 默认为属性框中设定

Integer lLineWidth(-1); //设定画线线宽 Enum_1Pix、Enum_2Pix、Enum_3Pix、Enum_4Pix、Enum_5Pix、Enum_6Pix、Enum_7Pix 枚举函数指定 默认为属性框中设定

Integer lColor(-1); //输出值的显示颜色，默认表示使用属性设置框中的颜色

Integer lBack(0); //当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar

使用案例：

```
events
  OnBar(ArrayRef<Integer> indexs)
  {
     PlotAuto("OpenToClose",open,close,Blue,Enum_Line,Enum_Solid,Enum_2Pix);
  }
```

对应 plotbool 类型的输出：plotauto 的参数

参数说明：

String strName; //输出值的名称，不区分大小写

Bool bVal; //输出的布尔值

```
Numeric dLocator(0);  //输出值的定位点
Integer lColor(-1);  //输出值的显示颜色，默认表示使用属性设置框中的颜色
Integer lIconType(-1);  //输出值的显示图形，默认显示系统笑脸
Integer lBack(0);  //当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
```

使用案例：

```
events
OnBar(ArrayRef<Integer> indexs)
{
    PlotAuto ("con",con,high); //在 bar 的最高价位置输出条件 con 的布尔值。
}
```

对应 plotstring 类型的输出：plotauto 的参数

参数说明：

```
String strName;  //输出值的名称，不区分大小写
String strVal;  //输出的字符串
Numeric dLocator(0);  //输出值的定位点
Integer lColor(-1);  //输出值的显示颜色，默认表示使用属性设置框中的颜色
Integer lBack(0);  //当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
```

使用案例：

```
events
OnBar(ArrayRef<Integer> indexs)
{
    PlotAuto("Bear Market","Bear Bear",high,blue); //在 bar 的最高价位置输出一个字符串，并且显示为蓝色。
}
```

Print 函数在控制台的输出

在 TBQuant 里面 K 线右侧增加了控制台，可以使用 Print(string)函数输出内容到控制台。

比如写这样一个公式：print 二维数组的内容到右侧控制台。

```
Vars
  Global Array<Array<Numeric>> a;
  Global Numeric i;
  Global Numeric j;
Events
OnInit()
{
  for i=0 to 10
  {
    for j=0 to 10
    {
      a[i][j]=i+j;
    }
  }
}
```

```
OnBar(ArrayRef<Integer> indexs)
{
  Print(TextArray(a));
}
```

1.7 用户函数

函数是 TB 语言中一类非常重要的语言元素，实现数据的加工计算。函数一般有参数，数据的传入通过函数声明的各个参数，把函数的参数值代入函数供计算机处理；函数必须有返回值，函数计算得出结果之后，返回数值到调用它的程序。

函数分为系统函数和用户函数，顾名思义，系统函数是由系统提供的，而用户函数则是用户根据需求自己编制的。

系统函数详细说明参见系统函数。

用户函数是通过名称进行调用的一组语句的集合，它具有特定的功能，执行结束后有返回值。在公式中如果需要使用某个函数相应的功能，调用函数即可，无需重新编写代码。

1、用户函数的类型

1、按照返回值类型分类

数值型(Numeric)

布尔型(Bool)

字符串(String)

整数型(Integer)

用户函数在调用时需要将返回值赋予类型相同的变量。

2、按照用户函数的实现机制分类

普通函数：输入参数，执行一段程序代码，返回需要的值

序列函数：输入参数或变量中有序列数据的用户函数

序列函数是一种特殊的用户函数，它的参数或变量中使用了序列数据。引入序列数据是 TB 语言和普通计算

机语言的重要区别，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沿用上一个 Bar 的值。因此，除非是算法需要，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。

2、TB 软件中用户函数使用规则

参数支持基本数据类型和容器扩展类型，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；
变量支持类型，支持指定变量的默认值；
可以访问 Data0-Data99 个数据源的 Bar 数据，通过数据源数组 data 访问的数据源个数不受限制；
可以访问行情数据、属性数据；
必须通过 Return 返回数据，返回数据类型为四种基本类型之一；
脚本中的返回数据类型必须和函数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。

3、函数的编写

用户函数同公式应用一样，其实体由三部分组成：参数声明，变量声明，脚本正文。
参数声明和变量声明参照 TB 语言基础相关内容；
脚本正文部分是编写实现函数功能的代码部分，它将函数的参数值代入代码进行计算，得出函数的返回值，通过 Return 返回。
例：编写 Average1 函数，Average1 函数的功能是计算 Price 在 Length 周期内的平均值。
编写步骤：

1. 新建用户函数，定义函数的名称 Average1；

在导航页上打开【公式管理】应用工作区“公式管理”，单击新建用户函数按钮“”打开新建函数的窗口，进行下图所示的设置：

2. 单击“确定”按钮，打开公式编辑器，输入代码：

a) 声明参数：Price，Length；
b) 声明变量 AvgValue 保存函数的计算结果，类型为 Numeric（与返回值类型一致）；
c) 编写具体实现功能的代码：调用 Summation 求和，并计算平均值，返回结果。

3. 点击工具栏的编译公式按钮“  ”编译保存当前函数。

Average1 函数脚本如下：

```
Params
  Series<Numeric> Price(1);
  Numeric Length(10);
Vars
  Numeric AvgValue;
Events
OnBar(ArrayRef<Integer> indexs)
{
  AvgValue = Summation(Price, Length) / Length;
  Return AvgValue;
}
```

本例只有一个返回值，即最后求得的平均值，在函数中直接使用 Return 语句返回即可。

注意：如果函数需要多个返回值，不可使用多条 Return 语句，可以将其他需要返回值的变量定义为引用型参数，即***Ref 类型，这种类型的参数变量可以将其在函数内部中的改变直接传递出去。

例：求 N 周期最大值。假定需要编写的用户函数功能需求为：求出序列变量 Price 在最近 Length 周期内的最大值，并且求出最大值出现的 Bar 与当前 Bar 的偏移量。

函数脚本如下：

```
Params
  Series<Numeric> Price(1);
  Numeric Length(10);
  NumericRef HighestBar; //设置引用型的变量
Vars
  Numeric MyVal;
  Numeric MyBar;
  Numeric i;
Events
OnBar(ArrayRef<Integer> indexs)
{
  MyVal = Price;
  MyBar = 0;
  For i = 1 to Length-1
  {
    If ( Price[i] > MyVal)
     {
      MyVal = Price[i];
      MyBar = i; //记录最大值 Bar 与当前 Bar 的偏移量
    }
  }
  HighestBar = MyBar; //将偏移量赋值给引用型变量，将该值传递回去
  Return MyVal; //返回计算得到的最大值
}
```

4、用户函数的调用

语法格式：
变量名=函数名(<参数列表>)；
说明：
用户函数成功创建之后（编译/保存成功），可以在其他的用户函数、公式应用中调用；
函数调用时，函数如果有参数一定要加()，参数列表中的参数个数、类型要一一对应、匹配；如果没有参数，()可以省略；
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。
函数参数声明为引用类型时，可传入的变量类型不包括序列类型。譬如函数参数声明类型为 NumericRef 时，不可传入 Series<Numeric>数据类型。
函数参数声明为其它类型时，可传入同类型的所有扩展类型。
公式中变量定义的数据类型和函数的返回值类型一致
其他公式、函数中调用函数时，可将获得返回值的变量的数据类型定义为函数返回值同种类型的扩展类型。
例如：函数返回值为 Numeric，可以赋给公式中类型为 Series<Numeric>或 NumericRef 的变量。
函数内可以调用其它数据源的内嵌系统函数，如在函数 Data0.Average 的代码中可以使用 Data1.Close。
例：在公式中调用 Average1 函数，求最近 10 个周期的 Close 的平均值。
方法一：

```
Vars
  Numeric Value1;
Begin
  Value1 = Average1(Close,10);
End
```

方法二：

```
Vars
  Series<Numeric> Value1;
Begin
  Value1 = Average1(Close,10);
End
```

1.8 公式应用

公式应用是用户编写交易策略主体代码的模块。本文从公式函数、技术分析类、交易策略类和公式报警四个方面介绍下公式应用的使用。

1、公式函数 Defs

TBQuant 升级了公式运行机制为事件驱动机制以后，除了可以定义用户函数外，在用户公式里面，也可以定义函数，这里的函数是公式函数，统一定义在 Defs 的域内。
对于公式函数可以使用的参数类型是最为广泛的，可以参照数据类型的说明。
关于公式函数的使用有几点需要注意：
1）公式函数的参数不需要事先定义；
2）公式函数只可以在当前公式中使用，在其它公式中需要重新定义。
3）defs 同样支持数据源的调用比如 data[i].func()，func 是 defs 类型的函数。

2、技术分析类

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。

模板介绍

新建技术分析类的公式应用，在新建公式应用的模板里选择“技术分析”，打开之后如下图所示。

公式管理 | get2v - get2v | test - test

```
Params
    Numeric FastLength(5);
    Numeric SlowLength(20);
Vars
    Series<Numeric> AvgValue1;
    Series<Numeric> AvgValue2;
Events
OnBar(ArrayRef<Integer> indexs)
{
    Range[0:1]
    {
        AvgValue1 = AverageFC(Close,FastLength);
        AvgValue2 = AverageFC(Close,SlowLength);
        PlotNumeric("MA1",AvgValue1);
        PlotNumeric("MA2",AvgValue2);
        if (AlertEnabled && CrossOver(AvgValue1, AvgValue2))
        {
            Alert("Do Alert Now");
        }
    }
}
```

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。用户可以根据自己的需求来进行添加、修改与完善公式并进行编译。

编写思路

首先，确定指标里需要的信息及计算方式；
然后，在公式中进行相应的定义和计算；
最后，将计算结果以数值、布尔或字符串的形式在图表上输出。

注意事项

1） 指标的输出属性可以在属性设置中进行相应设置；
2） 指标是在主图显示还是在子图显示；
3） 指标的线型；
4） 如果公式在多数据源上运行，公式中与数据源有关的变量和函数需显式调用。若不显示数据源则默认其数据源为 Data0。如技术分析模板中的代码在多数据源上运行时，需写成如下的形式：

```
Params
    Numeric FastLength(5);
    Numeric SlowLength(20);
Vars
    Series<Numeric> AvgValue1;
    Series<Numeric> AvgValue2;
Events
OnBar(ArrayRef<Integer> indexs)
{
    Range[0:1]
    {
     AvgValue1 = AverageFC(Close,FastLength);
     AvgValue2 = AverageFC(Close,SlowLength);
     PlotNumeric("MA1",AvgValue1);
     PlotNumeric("MA2",AvgValue2);
     if (AlertEnabled && CrossOver(AvgValue1, AvgValue2)
     {
       Alert("Do Alert Now");
     }
   }
  }
```

3、交易策略类

当我们在公式应用中编写了完整的开平仓规则以及、头寸控制、风险控制等代码，我们称之为交易策略，交易策略是我们一个独立交易思想的完整体现。

模板介绍

新建交易策略类的公式应用，在新建公式应用的模板里选择“交易策略”，打开之后如下图。

公式管理 | get2v - get2v | test - test*

```
1  Params
2      Numeric para1(10);
3      Numeric para2(5);
4  Vars
5      Bool condition1;
6      Bool condition2;
7  Events
8  OnBar(ArrayRef<Integer> indexs)
9  {
10     condition1={条件表达式1};
11     condition2={条件表达式2};
12     if(condition1 && marketposition<>1)      //条件满足且没有持多仓情况下开多
13     {
14         buy(1,Open);
15     }else if(condition2 && marketposition!=-1)  //条件满足且没有持空仓时开空
16     {
17         sellshort(1,Open);
18     }
19 }
```

模板里已经设置好了两个参数，两个布尔变量（用于保存开平仓的条件），两个条件语句（用于开平仓）。用户可以根据实际策略的需求，调整变量、条件与计算的数量，判断交易的入场点与出场点，在图表上对

买卖点做出标识。
如果用户已经熟悉了 TB 公式编写的方法，新建公式应用时可不选择任何模板，直接选择“空”模板，打开空白的编辑器，自行编写语句。

交易函数

1) Buy, SellShort, BuyToCover, Sell

语法格式:

```
Bool  Buy(Numeric Share=0,Numeric Price=0)
Bool  BuyToCover(Numeric Share=0,Numeric Price=0)
Bool  Sell(Numeric Share=0,Numeric Price=0)
Bool  SellShort(Numeric Share=0,Numeric Price=0)
```

参数说明:

Share 数量，为整型值，默认值为 0 表示使用系统设置参数;

Price 价格，为浮点数，默认值为 0 表示使用现价(非最后 Bar 为 Close);

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，当图表讯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，当图表讯号显示有多头持仓时，方可执行此指令;

例: 判断条件，执行买入或者卖出的动作，并在图表上标识出讯号

代码如下:

```
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();
    }
```

2) MarketPosition

MarketPosition-----获得当前持仓状态。

语法格式:

```
Integer MarketPosition()
```

返回值说明:

-1 当前位置为持空仓

0 当前位置为持平

1 当前位置为持多仓

3) A_SendOrder

A_SendOrder()与枚举函数配合使用，直接对帐户进行发送指令的动作（开空、开多、平空、平多）。A_SendOrder()仅在实时行情中最后一个 Bar 上针对当前帐户操作，不能在图表中标识买卖讯号，也不能进行历史回溯测试。

语法:

Bool A_SendOrder(Integer BuyOrSell,Integer EntryOrExit,Numeric fLot,Numeric fPrice)

针对当前公式应用的帐户、商品发送委托单，发送成功返回 True,发送失败返回 False。

参数:

BuyOrSell 发送委托单的买卖类型，取值为 Enum_Buy(买入)或 Enum_Sell(卖出)之一;

EntryOrExit 发送委托单的开平仓类型，取值为 Enum_Entry(开仓),Enum_Exit(平仓),Enum_ExitToday(平今仓)之一;

fLot 委托单的交易数量;

fPrice 委托单的交易价格。

下表为 A_SendOrder 函数使用示例:

参数 1: 买卖	参数 2: 开平	参数 3: 数量	参数 4: 价格	示 例
1.开多 Enum_Buy	Enum_Entry	5	Q_AskPrice()	A_SendOrder(Enum_Buy,Enum_Entry,5,Q_AskPrice());
2.平多 Enum_Sell	Enum_Exit, Enum_ExitToday	5 or A_BuyPosition()	Q_BidPrice()	A_SendOrder(Enum_Sell,Enum_Exit,5,Q_BidPrice());
3.开空 Enum_Sell	Enum_Entry	开空仓单 5 手	Q_BidPrice()	A_SendOrder(Enum_Sell,Enum_Entry,5,Q_BidPrice());
4.平空 Enum_Buy	Enum_Exit Enum_ExitToday	5 or A_SellPosition()	Q_AskPrice()	A_SendOrder(Enum_Buy,Enum_Exit,5,Q_AskPrice());

A_SendOrder()函数直接发单，无需任何确认，在实时行情中，每个 Tick 都会触发程序的运行，因此，使用 A_SendOrder()函数需要根据仓位头寸并配合全局变量进行条件处理，避免造成重复发单。

【问题】buy、sellshort 与 A_sendorder 有什么区别？

【回答】buy、sellshort 在图表上标识买卖信号，与 k 线、行情数据有关，可用于历史回测及实时交易，该函数同一个 bar 不会重复发单（TB 底层保证）。marketposition 依据图表信号判断当前持仓情况，该值不与账户关联，不能反应账户实际的持仓情况。

A_sendorder 与账户关联，交易不在图表上产生信号，不能用于历史回测，仅对实时行情有效。使用此函数进行交易，需要额外增加头寸判断的条件语句，避免重复发单。

【补充】交易函数匹配

- MarketPosition 与 Buy，SellShort， BuyToCover，Sell 匹配图表持仓情况：根据图表持仓情况判断开仓及控制仓位
- A_sendorder 与 A_TotalPosition，A_BuyPosition，A_SellPosition 匹配账户持仓情况：根据账户持仓情况，判断开仓及控制仓位

编写思路

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。

首先，理顺思路，将交易规则转化为相关的条件语句表达式，确立策略中的交易头寸及要采用的参数和变量;

然后，确定需要使用的交易命令;

最后，在公式中进行相应的定义，完成策略实现代码。

示例，以下是一个双均线交易策略的代码：

```
Params
  Numeric FastLength(5);
  Numeric SlowLength(20);
  Numeric BuyLots(1);
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
events
OnBar(ArrayRef<Integer> indexs)
{
  AvgValue1 = AverageFC(Close,FastLength);
  AvgValue2 = AverageFC(Close,SlowLength);
  If(MarketPosition!=1 And (AvgValue1[1] > AvgValue2[1]))
  {
    Buy(BuyLots,Open);
  }
  If(MarketPosition!=-1 And (AvgValue1[1] < AvgValue2[1]))
  {
    SellShort(BuyLots,Open);
  }
}
```

为了在上面交易策略在超级图表中执行同时看到两条均线的数值，我们也可以在交易策略中输出指标线条，只需要增加以下两行代码：

```
    PlotNumeric("MA1",AvgValue1);
    PlotNumeric("MA2",AvgValue2);
```

除了希望看到两条均线值之外，我们还希望能够在超级图表中看到交易策略的盈亏曲线，这时我们需要再增加一条指标线：

```
    PlotNumeric("OpenEquity",Portfolio_TotalProfit);
```

策略的头寸

直接指定交易头寸，如果指定头寸为 0，则系统自动按 1 手处理。

使用参数（如上述 DualMA 双均线案例）

根据资产具体情况以及交易要求计算得出

定义变量，通过对资产、行情等条件的计算以及其它的判断条件确定交易的数量。

例：交易开拓者软件的系统公式里自带的“海龟交易系统”

```
    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
```

计算公式

头寸规模 = 账户净值的 1%/价值量波动性

账户净值 = 可用资金 + 持仓保证金（全局交易里设置）

价值量波动性 = N*每点价值量
N = 真实波动幅度的 20 周期指数移动平均值（XAverage）
通过一系列的计算，在不同的行情与资金状态下，计算得出的开仓数量也不尽相同。

程序调试

公式代码编写完毕，单击编译保存按钮对公式进行编译。简单语法错误，可以根据编译提示信息进行修改，如果出现逻辑错误，导致公式不能得到所需的运行结果，则需要在代码中添加调试语句，检查出错的原因。常用的调试语句有 Commentary 和 FileAppend。

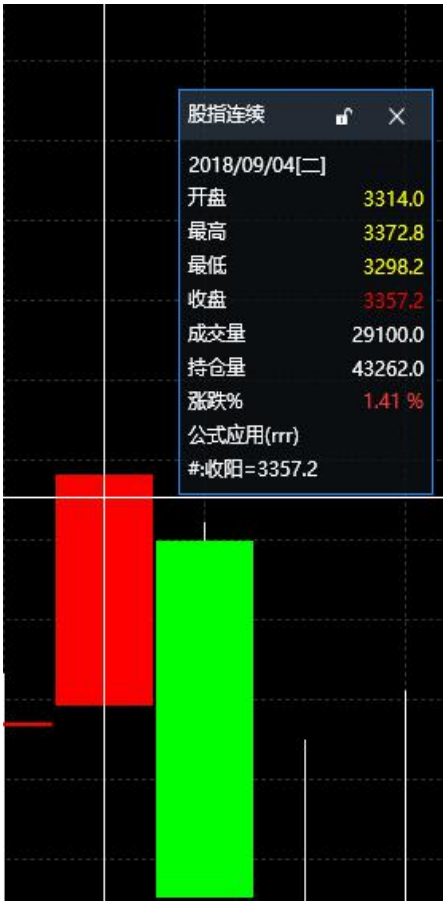
Commentary

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，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。

例:

```
If(close>open)
    Commentary("收阳="+Text(close));  //在收阳的 K 线上显示收盘价
```



如上图显示，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，不满足条件的 Bar 上则不会有注释信息的输出。

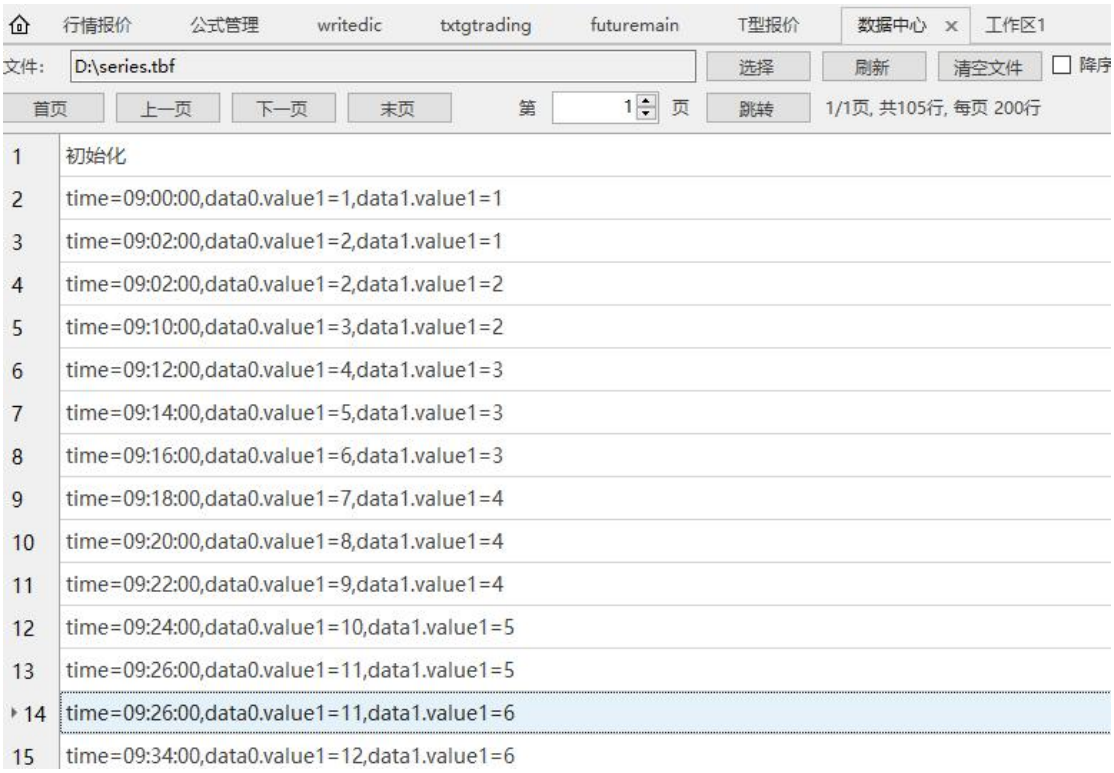
当然，如果没有条件语句来约束 Commentary，则在图表上的每个 Bar 上都输出注释信息。采用这种方式可以方便交易者观察图表上的数据情况，对公式策略进行适当的调试。

例如：
```
Commentary(“开仓价格”+(symbol)+”=”+text(entryprice));
Commentary(“满足多头开仓”);
```

Commentary 显示的注释信息仅在图表中标识，如果需要生成记录文件，则要使用函数。FileAppend

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。
FileAppend 的调试信息不会在图表上标识，而是写入一个文件中。每一次执行到这个语句，便会在文件中写入一条调试信息，方便交易者查找调试。
注意：文件名后缀一般使用 tbf 格式。文件的内容在数据中心进行查看。
例如：

```
FileAppend("d:/series.tbf","time="+TimeToString(Time)+",data0.value1="+Text(data0.Value1)+",data1.value1="+Text(data1.value1));
```



1	初始化
2	time=09:00:00,data0.value1=1,data1.value1=1
3	time=09:02:00,data0.value1=2,data1.value1=1
4	time=09:02:00,data0.value1=2,data1.value1=2
5	time=09:10:00,data0.value1=3,data1.value1=2
6	time=09:12:00,data0.value1=4,data1.value1=3
7	time=09:14:00,data0.value1=5,data1.value1=3
8	time=09:16:00,data0.value1=6,data1.value1=3
9	time=09:18:00,data0.value1=7,data1.value1=4
10	time=09:20:00,data0.value1=8,data1.value1=4
11	time=09:22:00,data0.value1=9,data1.value1=4
12	time=09:24:00,data0.value1=10,data1.value1=5
13	time=09:26:00,data0.value1=11,data1.value1=5
14	time=09:26:00,data0.value1=11,data1.value1=6
15	time=09:34:00,data0.value1=12,data1.value1=6

上述语句，没有添加任何条件直接输出，实时行情中因为每个 Tick 程序都会运行一次，所以同一个 Bar 上会输出多条调试语句的结果。用户可以通过查看这些结果找到自己所需要的信息。
如果在条件语句下输出 FileAppend，则条件满足之后才会有调试语句写入文件。
注意事项：

- 开、平仓指令成对出现；
- 加仓时注意在全局交易设置中勾选允许连续建仓；
- 使用 A_SendOrder 函数发单，注意重复发单的机制。

4、公式报警

TB 的公式提供 Alert、AlertEnabled 函数实现公式报警功能。
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：

```
Vars
   Bool  Condition1;
Events
```

```
OnBar(ArrayRef<Integer> indexs)
{

  //Condition1 = 您设定的条件表达式;
  If(AlertEnabled AND Condition1)
  {
    Alert("报警信息...");
  }
}
```

AlertEnabled
参见 AlertEnabled。
Alert
参见 Alert。

1.9 Plot 帮助文档

1、Plot 定义

1.1 背景

- **画布**

a) 有多个画板组成
b) 画板之间相互独立
c) 画板之间只有布局关系
d) 画板之间可以分组，相同分组 X 坐标轴可以联动

- **画板**

a) 有多个图形组成，分别是一个主图，多个附图
b) 有共同的一个坐标轴
c) 图形之间是叠加关系
d) 生成主坐标的图形，称为主图(或父图)
e) 叠加在主图上的图形，称为附图(或子图)

1.2 定义

- Plot 表示画图对象
- 可以重复使用画图、设置坐标、属性等
- 可以定义画板内的叠加关系
- 可以定义画布内的布局关系
- 公式定义

```
Vars
  Plot plt;
```

2、画板定义

2.1 定义

- Figure 表示一个画板

```
figure();
```

- 参数

参数	类型	说明
groupid	整型	画板分组 ID

```
plt.figure();
```

2.2 重定向

- Redirect

```
redirect(String component,String name);
```

- 参数

参数	类型	说明
component	String	组件名称，目前只支持 table 可以重定向'量化看盘'
name	String	组件实例名称

```
plt.redirect('量化看盘');
```

2.3 主图

- Plot 本身不代表是否为主图或附图
- Plot 可以重复使用
- 主图：画板第一个输出的图形，且决定 X、Y 轴，即为主图

2.4 附图

- 附图：叠加到主图上的图形，不能定义 X、Y，这也是与主图的最大差别
- 实现：
  a) subplot：把附图对象叠加到主图对象上
  b) 使用同一个 Plot，也可以叠加
  c) childPlt 不能是 figure，即不是独立画板

```
childPlt.subplot(parentPlt);
```

- 参数

参数	类型	说明
parentPlt	Plot 对象	主图对象

2.5 示例

```
Vars
    Plot parentPlt1;//主图
    Plot parentPlt2;//主图
    Plot childPlt;//附图
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
```

```
Events
    OnInit()
    {
        parentPlt1.figure(); //独立画板
        parentPlt2.figure(); //独立画板
        parentPlt1.setOption("MA1","x-format","time");//设置 X 为时间轴
        parentPlt2.setOption("MA2","x-format","time");//设置 X 为时间轴
        childPlt.subplot(parentPlt1);//childPlt 叠加到 parentPlt1 主图上显示
        childPlt.subplot(parentPlt2);//childPlt 叠加到 parentPlt2 主图上显示
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        parentPlt1.line("MA1",xData,yData);
        parentPlt2.line("MA2",xData,yData);
        yData[0] = AverageFC(Close,10);
        parentPlt1.line("MA11",xData,yData);//同一个 Plot 叠加
        childPlt.line("MA3",date+time,AverageFC(Close,20));//subplot 叠加
    }
```

2.6 K 线

- **叠加**
    a) K 线默认为主图，不需要显示设置 Figure()
    b) 附图如果没有显示调用 subplot，则默认叠加到 k 线所在主图
- **示例**

```
Vars
    Plot plt1;//主图
    Plot plt2;//附图
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        yData[0] = AverageFC(Close,10);
        plt2.line("MA2",xData,yData);
    }
```

3、属性设置

3.1 属性函数

plt.setOption(name,option,value);

- 参数

参数	类型	说明
name	String	图形名称，建议只要只用字符、数字、下划线组成
option	String	属性名称
value	String/Numeric/Integer	属性值，根据不同的属性对应相应的值类型

3.2 设置颜色

- 定义
  a) 属性名: "color"
  b) 属性表

属性值	类型	说明
Black	枚举	黑色
Blue	枚举	蓝色
Red	枚举	红色
White	枚举	白色
Yellow	枚举	黄色
N	数值	ARGB 值

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt1.setOption("MA1","color",White);//将指标“MA1”的输出颜色设置为白色
        plt2.setOption("MA2","color",Yellow);//将指标“MA2”的输出颜色设置为黄色
        plt2.figure();//独立画板
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        plt2.line("MA2",date+time,AverageFC(Close,5));
    }
```

3.3 设置宽度

- 定义
    a) 属性名: "width"
    b) 属性表

属性值	类型	说明	应用场景
Enum_1Pix	枚举	1 个像素	线、柱状图、字符串、图标
Enum_2Pix	枚举	2 个像素	线、柱状图、字符串、图标
Enum_3Pix	枚举	3 个像素	线、柱状图、字符串、图标
Enum_4Pix	枚举	4 个像素	线、柱状图、字符串、图标
Enum_5Pix	枚举	5 个像素	线、柱状图、字符串、图标
Enum_6Pix	枚举	6 个像素	线、柱状图、字符串、图标
Enum_7Pix	枚举	7 个像素	线、柱状图、字符串、图标

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt1.setOption("MA1","width",Enum_2Pix);//将指标“MA1”的输出线宽设置为 2 像素
        plt2.setOption("MA2","width",Enum_5Pix);//将指标“MA2”的输出线宽设置为 5 像素
        plt2.figure();//独立画板
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        plt2.line("MA2",date+time,AverageFC(Close,5));
    }
```

3.4 设置风格

- 定义
    a) 属性名: "style"
    b) 属性表

属性值	类型	说明	应用场景
num_Solid	枚举	实线	线
Enum_Dash	枚举	虚线	线
Enum_Broken	枚举	破折线	线
Enum_Dash_Dot	枚举	点划线	线

“candel”	字符串	蜡烛图	K 线
“hollow”	字符串	中空蜡烛图	K 线
“amer”	字符串	美国线	K 线
“close”	字符串	收盘线	K 线

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
    Array<Bar> klines;
    Bar myKline;
Events
    OnInit()
    {
        plt1.setOption("MA1","style",Enum_Dash); //将指标“MA1”的输出线风格设置为虚线
        plt2.setOption("Kline","style","hollow"); //将“Kline”的风格设置为中空蜡烛图
        plt2.setOption("Kline","x-format","time"); //定义为时间轴
        plt2.setOption("MA2","style",Enum_Broken); //将指标“MA2”的输出线风格设置为破折线
        plt2.figure();//独立画板
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        GetBar(myKline);
        klines[0]=myKline;
        yData[0] = AverageFC(Close,10);
        plt2.kline("Kline",klines);
        plt2.line("MA2",xData[0],yData[0]);
    }
```

3.5 设置线型

- 定义
    a) 属性名: "line-type"
    b) 属性表

属性值	类型	说明	应用场景
Enum_Dot	枚举	点	画线
Enum_Line	枚举	线	画线
Enum_Cross	枚举	十字线	画线

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt1.setOption("MA1","line-type",Enum_Dot);//将指标“MA1”的线型设置为点
        plt2.setOption("MA2","line-type",Enum_Cross);//将指标“MA2”的线型设置十字线
        plt2.figure();//独立画板
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        plt2.line("MA2",date+time,AverageFC(Close,5));
    }
```

3.6 设置布局

- 设置上边界
    a) 属性名: "margin-top"
    b) 属性表

属性值	类型	说明	应用场景
“-10”	字符串	以主图层下限向下 10 等数值个单位为输出图形的上限	所有图形
“10”	字符串	以主图层下限向上 10 等数值个单位为输出图形的上限	所有图形
“-10%”	字符串	以主图层下限向下 10%等数值为输出图形的上限	所有图形
“10%”	字符串	以主图层下限向上 10%等数值为输出图形的上限	所有图形
“high”	字符串	以主图层上限为输出图形的上限	所有图形
“low”	字符串	以主图层下限为输出图形的上限	所有图形

- 设置下边界
    a) 属性名: "margin-bottom"
    b) 属性表

属性值	类型	说明	应用场景
“-10”	字符串	以主图层下限向下 10 等数值个单位为输出图形的下限	所有图形
“10”	字符串	以主图层下限向下 10 等数值个单位为输出图形的下限	所有图形
“-10%”	字符串	以主图层下限向下 10%等数值个单位为输出图形的下限	所有图形
“10%”	字符串	以主图层下限向下 10%等数值个单位为输出图形的下限	所有图形

“high”	字符串	以主图层下限向下 10 个单位为输出图形的下限	所有图形
“low”	字符串	以主图层下限向下 10 个单位为输出图形的下限	所有图形

- 示例

```
Vars
    Plot plt;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        //指标 MA1 在主图的[100%,120%]内显示
        plt.setOption("MA1","margin-top","120%");
        plt.setOption("MA1","margin-bottom","100%");
        //指标 MA2 不设置边界
        //指标 MA3 在主图的[0%,20%]内显示
        plt.setOption("MA3","margin-top","20%");
        plt.setOption("MA3","margin-bottom","0%");
        //指标 Vol 在主图的[-20%,0%]内显示
        plt.setOption("Vol","margin-top","0%");
        plt.setOption("Vol","margin-bottom","-20%");
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt.line("MA1",xData,yData);
        plt.line("MA2",date+time,AverageFC(Close,5));
        plt.line("MA3",xData,yData);
        plt.barv("Vol",date+time,Vol);
      }
```

3.7 设置坐标轴格式

- 定义
  a) X 轴属性名: "x-format"
  b) Y 轴属性名: "y-format"
  c) 属性表

属性值	类型	说明	应用场景
numberic	String	指定主图 x 轴数值，也是默认值	所有图形
numberic_s	String	指定主图 x 轴数值，数值是会缩进的(万/亿等)	所有图形
time	String	指定主图 x 轴为时间轴	所有图形

d) 说明：k 线所在图形，默认值是时间轴，不能修改

- **示例**

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt1.figure();//独立画板
        plt2.figure();//独立画板
        plt1.setOption("MA1","x-format","time");//时间格式
        plt1.setOption("MA1","y-format","numeric");//数值
        plt2.setOption("Vol","y-format","numeric_s");//格式化缩进数值
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        plt2.line("Vol",CurrentBar(),Vol);
    }
```

3.8 设置柱状图属性

- **填充属性**

a) 属性名：“fill”
b) 属性表

属性值	类型	说明	应用场景
true	bool	柱状图填充	柱状图
false	bool	柱状图不填充	柱状图

- **描述信息**
  a) 属性名：“bar-descript”
  b) 属性值：类型为字符串数组，对应每一个柱状图。
- **示例**

```
Vars
    Plot plt1;
```

```
    Plot plt2;
    Global Array<Numeric> xData;//x 轴数据集
    Global Array<Numeric> yData;//y 轴数据集
    Global Array<Numeric> lData;//定位点集
Events
    OnInit()
    {
        //指标 Vol 在主图的[-20%,0%]内显示
        plt1.setOption("Vol","margin-top","0%");
        plt1.setOption("Vol","margin-bottom","-20%");
        plt2.figure();//独立画板
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        ArrayInsert(xData,GetArraySize(xData),date+time);
        ArrayInsert(yData,GetArraySize(yData),Vol);
        ArrayInsert(lData,GetArraySize(lData),Vol/2+CurrentBar%1000);
        If(CurrentBar%5==0)
        {
              plt2.barv("Vol2",xData,yData,lData);//批量
              ArrayClear(xData);
              ArrayClear(yData);
              ArrayClear(lData);
        }
        plt1.barv("Vol",date+time,Vol,Vol/2+CurrentBar%1000);//单点
    }
```

3.9 设置表格属性

- **列表头**
  - a) 属性名: "x-title"
  - b) 属性值: 类型为字符串数组，对应表格的列表头名称
- **表格属性**

属性值	类型	说明
column-index	String	设置列位置，0 表示索引键值
column-precision	String	设置列精度
column-font-size	String	设置列字体大小，默认 10.5
row-index	String	设置行位置，从 1 开始
title-color	Integer	置表头颜色

- **示例**

a) 批量更新

```
Vars
    Plot plt1;
```

```
    Plot plt2;
    Array<String> titleX;
    Array<Array<Numeric>> tableData;
Events
    OnInit()
    {
        titleX = ["开盘","高价","低价","最新","成交量","持仓量","时间"];
        plt1.figure();
        plt2.figure();
        plt1.setOption("table1","x-title",titleX); //设置列头
        plt2.setOption("table2","x-title",titleX); //设置列头
        plt2.setOption("table2","title-color",Yellow);
        plt2.setOption("table2","key-column","时间");//设置时间所在列为行头，否则自动添加
序号
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        tableData[0][0] = Open;
        tableData[0][1] = High;
        tableData[0][2] = Low;
        tableData[0][3] = Close;
        tableData[0][4] = Vol;
        tableData[0][5] = OpenInt;
        tableData[0][6] = Date+time;
        plt1.table("table1",tableData);
        plt2.table("table2",tableData);
    }
```

b) 动态更新

```
Vars
    Plot plt;
    Array<String> titleX;
    Array<Array<Numeric>> tableData;
    Array<Array<String>> tableData2;
Events
    OnInit()
    {
        titleX = ["开盘","高价","低价","最新","成交量","持仓量","时间"];
        plt.figure();
        plt.redirect("量化看盘");
        //plt.setOption("table","x-title",titleX);//如果设置即定了顺序，否则根据动态更新
创建顺序
        plt.setOption("table","title-color",Yellow);
        plt.setOption("table","column-index","时间=0");//设置时间所在列为行头，否则自动添
```

加序号

```
        plt.setOption("table","column-index","最新=4");//
        plt.setOption("table","column-precision","高价=3");//
        plt.setOption("table","column-font-size","低价=7");//默认 10.5}
    OnBar(ArrayRef<Integer> indexs)
    {
        plt.table("table","开盘",Text(Open),DateTimeToString(Date+Time));
        plt.table("table","高价",High,DateTimeToString(Date+Time));
        plt.table("table","低价",Low,DateTimeToString(Date+Time));
        plt.table("table","成交量",Text(Vol),DateTimeToString(Date+Time));
        plt.table("table","持仓量",Text(OpenInt),DateTimeToString(Date+Time));
        plt.table("table","最新",Text(Close),DateTimeToString(Date+Time));}
```

4、图形输出

4.1 线形

- **线形输出**

plt.line(name,x-axis,y-axis,locator);

参数

参数	类型	说明
name	String	线形名称
X-axis	Array<Numeric>或 Numeric	X 轴坐标数组
Y-axis	Array<Numeric>或 Numeric	y 轴坐标数组
locator	Array<Numeric>或 Numeric	画图定位点

- **单次输出**

```
Vars
    Plot plt1;
    Plot plt2;
Events
    OnInit()
    {
        plt2.figure();//独立画板
        plt2.setOption("line2","x-format","time");//设置 x 轴为时间轴
        plt2.setOption("line2","color",Red());//设置颜色
        plt2.setOption("line2","style",Enum_Dash_Dot());//设置线型
        plt2.setOption("line2","width",Enum_2Pix());//设置线宽
        plt2.setOption("line2","y-format","numeric");//数值格式
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        if(open > close)
        {
```

```
            plt1.setOption("line1","color",green);
            plt2.setOption("line2","color",green);
    }
        else
        {
            plt1.setOption("line1","color",Red);
            plt2.setOption("line2","color",green);
        }
        plt1.line("line1",date+time,(High+Low)/2);
        plt2.line("line2",date+time,(High+Low)/2);
      }
```

- **批量输出**

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> x1Data;//x 轴数据集
    Array<Numeric> y1Data;//y 轴数据集
    Global Array<Numeric> x2Data;//x 轴数据集
    Global Array<Numeric> y2Data;//y 轴数据集
    Integer i(0);
Events
    OnInit()
    {
        plt1.figure();//独立画板
        plt2.figure();//独立画板
        plt2.setOption("line2","x-format","time");//设置 x 轴为时间轴
        plt2.setOption("line2","color",Red());//设置颜色
        plt2.setOption("line2","style",Enum_Dash_Dot());//设置线型
        plt2.setOption("line2","width",Enum_2Pix());//设置线宽
    }
    OnReady()
    {
        For i=0 To 1000
        {
              x1Data[i]= i;
              y1Data[i]= Rand(100,150);
        }
        plt1.line("line1",x1Data,y1Data);//批量画图
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        ArrayInsert(x2Data,GetArraySize(x2Data),date+time);
        ArrayInsert(y2Data,GetArraySize(y2Data),(High+Low)/2);
        If(CurrentBar%5==0)
```

```
        {
            plt2.line("line2",x2Data,y2Data);//批量
            ArrayClear(x2Data);
            ArrayClear(y2Data);
        }
    }
```

- **画线段-不连续 Bar**

```
plt1.setOption("MA1","line-display","interval");
```

参数	类型	说明
line-display	String	设置画图类型属性， (interval(不连续) 、series(连续)

```
Params
    //此处添加参数
Vars
    //此处添加变量
    Plot plt1;

Events
    //此处实现事件函数

    //初始化事件函数，策略运行期间，首先运行且只有一次，应用在订阅数据等操作
    OnInit()
    {
        plt1.setOption("MA1","color",Yellow());
        plt1.setOption("MA1","width",Enum_4Pix());
        plt1.setOption("MA1","line-display","interval");
    }


    //Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
    OnBar(ArrayRef<Integer> indexs)
    {
        if( time > 0.090000 && time < 0.110000)
        {
            plt1.line("MA1",Date+Time,High);
        }
    }
```

- **画线段-独立图层和 k 线联动**

```
Params
        //此处添加参数
Vars
        //此处添加变量
```

```
        Plot plt1;

Events
        //此处实现事件函数

        //初始化事件函数，策略运行期间，首先运行且只有一次，应用在订阅数据等操作
        OnInit()
        {
                plt1.figure(0);
                plt1.setOption("MA1","x-format","time");
                plt1.setOption("MA1","color",Yellow());
                plt1.setOption("MA1","width",Enum_4Pix());
                plt1.setOption("MA1","line-display","interval");
        }
        //Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
        OnBar(ArrayRef<Integer> indexs)
        {
                if( time > 0.090000 && time < 0.110000)
                {
                        plt1.line("MA1",Date+Time,High);
                }
        }
```

4.2 柱状图

- **柱状图输出**

```
pl.barv(name,x-axis,y-axis,locator);
```

参数

参数	类型	说明
name	String	柱状图名称
x-axis	Array<Numeric>或 Numeric	X 轴坐标数组
y-axis	Array<Numeric>或 Numeric	Y 轴坐标数组
locator	Array<Numeric>或 Numeric	定位点数组

- **横向柱状图输出(待开发)**

```
pl.barh(name,x-axis,y-axis);
```

参数

参数	类型	说明
name	String	柱状图名称
X-axis	Array<Numeric>或 Numeric	X 轴坐标数组
Y-axis	Array<Numeric>或 Numeric	Y 轴坐标数组

- **单次输出**

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> x1Data;//x 轴数据集
    Array<Numeric> y1Data;//y 轴数据集
Events
    OnInit()
    {
        plt1.setOption("bar1","margin-top","0%");
        plt1.setOption("bar1","margin-bottom","-20%");
        plt1.setOption("bar1","color",Yellow);
        plt2.figure();//独立画板
        plt2.setOption("bar2","x-format","time");//设置 x 轴为时间轴
        plt2.setOption("bar2","color",Red());//设置颜色
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        x1Data[0] = date+time;
        y1Data[0] = Vol;
        plt1.barv("bar1",x1Data,y1Data);
        plt2.barv("bar2",x1Data,y1Data);
    }
```

- **批量输出**

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> x1Data;//x 轴数据集
    Array<Numeric> y1Data;//y 轴数据集

    Global Array<Numeric> x2Data;//x 轴数据集
    Global Array<Numeric> y2Data;//y 轴数据集
    Integer i(0);
Events
    OnInit()
    {
        plt1.figure();//独立画板
        plt1.setOption("bar1","color",Yellow());//设置颜色

        plt2.figure();//独立画板
        plt2.setOption("bar2","x-format","time");//设置 x 轴为时间轴
        plt2.setOption("bar2","color",Red());//设置颜色
    }
    OnReady()
```

```
    {
        For i=0 To 1000
        {
            x1Data[i]= i;
            y1Data[i]= Rand(100,150);
        }
        plt1.barv("bar1",x1Data,y1Data);//批量画图
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        ArrayInsert(x2Data,GetArraySize(x2Data),date+time);
        ArrayInsert(y2Data,GetArraySize(y2Data),(High+Low)/2);
        If(CurrentBar%5==0)
        {
            plt2.barv("bar2",x2Data,y2Data);//批量画图
            ArrayClear(x2Data);
            ArrayClear(y2Data);
        }
    }
```

4.3 文本

- **输出**

```
plt.text(name,x-axis,y-axis, texts);
```

- **参数**

参数	类型	说明
name	String	输出名称
X-axis	Array<Numeric>或 Numeric	X 轴坐标数组
Y-axis	Array<Numeric>或 Numeric	y 轴坐标数组
texts	Array<String> 或 String	输出的字符串数组

- **示例**

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
    Array<String> txtData;//字符串集
Events
    OnInit()
    {
        plt2.figure();
        plt2.setOption("txt2","x-format","time");
    }
```

```
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = Low;
        txtData[0]=Text(CurrentBar);
        plt1.text("txt1",xData,yData,txtData);
        plt2.text("txt2",xData,yData,txtData);
        plt2.setOption("txt2","color",Yellow-CurrentBar);//动态颜色
    }
```

4.4 表格

- **默认是独立画板，叠加无效**
- **表格输出**

```
plt.table(name,context);
```

- **参数**

参数	类型	说明
name	String	表格名称
context	Array<Array<numeric>>	表格内容

- **示例：** 如前面 3.9 的示例

4.5 K 线

- **K 线输出**

```
pl.kline(name,bars);
```

- **参数**

参数	类型	说明
name	String	K 线名称
bars	Array<Bar>或 Bar	K 线数据数组

- **示例**

```
Vars
    Plot plt;
    Bar myBar;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnReady()
    {
    plt.figure();
    plt.setOption("kline","x-format","time");//K 线不需要显示设置
    }
    OnBar(ArrayRef<Integer> indexs)
    {
```

```
        myBar.dateTime=date+time;
        myBar.open=Open()/10;
        myBar.high=High()/10;
        myBar.low=Low()/10;
        myBar.close=Close()/10;
        myBar.volume=Vol()/10;
        myBar.openInt=OpenInt();
        myBar.turnOver=turnOver();
        plt.kline("kline",myBar);
    }
```

4.6 图标

- Icon 输出

```
pl.icon(name,x-axis,y-axis, icons);
```

- 参数

参数	类型	说明
name	String	输出名称
x-axis	Array<Numeric>或 Numeric	K 线数据数组
y-axis	Array<Numeric>或 Numeric	Y 轴坐标数组
icons	Array<String> 或 String	图标字符串数组

- 映射表

参数	说明
default	默认
cuowu	错误
dengpao	灯泡
jiesuo	解锁
guanbi	关闭
yuanquan	圆圈
suoding	锁定
jiage	价格
jianshao	减少
jinzhi	禁止
mubiao	目标
nixiang	逆向
shoucang	收藏
xiangshang	向上
sousuo	搜索
zhengque	正确
yiwen	疑问
renminbi	人民币

zengjia	增加
xiangxia	向下
zhengxiang	正向
jingshi	警示

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
    Global Array<String> iconDatas;//字符串集
    Array<String> iconData;//字符串集
Events
    OnInit()
    {
        plt2.figure();//独立画板
        plt2.setOption("icon2","x-format","time");
        iconDatas =
["default","cuowu","dengpao","jiesuo","guanbi","yuanquan","suoding","jiage","jianshao","jinz
hi",
"mubiao","nixiang","shoucang","xiangshang","sousuo","zhengque","yiwen","renminbi","zengjia",
"xia
ngxia","zhengxiang","jingshi"];
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = Low-50;
        iconData[0]=iconDatas[CurrentBar()%GetArraySize(iconDatas)];
        Commentary("icon="+iconData[0]);
        plt1.icon("icon1",xData,yData,iconData);
        plt2.icon("icon2",xData[0],yData[0],iconData[0]);
    }
```

5、图形隐藏

5.1 K 线

- K 线隐藏

```
data0.hideKline();
```

- 隐藏图层上的 K 线，不影响 Plot 图形的输出。
- 示例

```
Vars
```

```
    Plot plt;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnReady()
    {
        Data0.HideKline();
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt.line("MA1",xData,yData);
    }
```

5.2 其他图形

- **图表隐藏**

```
plt.hide(name);
```

- **参数**

参数	类型	说明
name	String	图表名称，如果是空则 Plot 所画的图全部隐藏

- **示例 1-图表隐藏**

```
Vars
    Plot plt1;
    Plot plt2;
    Plot plt3;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt2.figure(); //独立画板
        plt2.setOption("MA2","x-format","time");
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);

        yData[0] = AverageFC(Close,5);
        plt2.line("MA2",xData,yData);
```

```
        yData[0] = AverageFC(Close,10);
        plt2.line("MA3",xData,yData);
        plt2.hide("MA2");

        plt3.line("MA4",xData,yData);
        plt3.hide("MA4");
    }
```

- 示例 2-画板隐藏

```
Vars
    Plot plt1;
    Plot plt2;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt2.figure(); //独立画板
        plt2.setOption("MA2","x-format","time");
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);

        yData[0] = AverageFC(Close,5);
        plt2.line("MA2",xData,yData);

        yData[0] = AverageFC(Close,10);
        plt2.line("MA3",xData,yData);

        plt2.line("MA4",xData,yData);
        plt2.hide();
    }
```

6、画板布局

6.1 分组

- 定义

```
figure(groupid);
```

- 参数

参数	类型	说明

groupid	整型	画板分组 ID，0 代表 K 线所在分组

- 示例

```
Vars
    Plot plt1;
    Plot plt2;
    Plot plt3;
    Array<Numeric> xData;//x 轴数据集
    Array<Numeric> yData;//y 轴数据集
Events
    OnInit()
    {
        plt1.figure(0); //独立画板，与 k 线同一分支
        plt2.figure(0); //独立画板，与 k 线同一分支
        plt3.figure(); //独立画板，没有分组
        plt1.setOption("MA1","x-format","time");//设置 X 为时间轴
        plt2.setOption("MA2","x-format","time");//设置 X 为时间轴
        plt3.setOption("MA3","x-format","time");//设置 X 为时间轴
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        xData[0] = date+time;
        yData[0] = AverageFC(Close,5);
        plt1.line("MA1",xData,yData);
        yData[0] = AverageFC(Close,10);
        plt2.line("MA2",xData,yData);
        plt3.line("MA3",date+time,AverageFC(Close,20));
    }
```

6.2 布局

- 定义

a) 表格布局

setLayout(row,column);//k 线默认(0,0)

参数

参数	类型	说明
row	Integer	行号，从 0 开始，k 线默认 0
column	Integer	列号，从 0 开始，k 线默认 0

b) 单元格合并

setCrossCells(rows,columns);

参数

参数	类型	说明

rows	Integer	跨行数，默认 1
columns	Integer	跨列数，默认 1

c) 大小占比

setSizeWeight(rowWeight,columnWeight);

参数

参数	类型	说明
rowWeight	Integer	行宽占比，默认是 1
columnWeight	Integer	列高占比，默认是 1

- **示例**

```
Vars
    Plot plt1;
    Plot plt2;
    Plot plt3;
    Plot plt4;
    Plot plt5;
Events
    OnInit()
    {
        /*目标图形,K 线默认(0,0)
        |K 线 |MA1|
        |M2 |MA3 |
        | |MA4|MA5|
        */
        plt1.figure();
        plt2.figure();
        plt3.figure();
        plt4.figure();
        plt5.figure();
        plt1.setLayout(0,2);//布局在(0,2)
        plt2.setLayout(1,0);//布局在(1,0)
        plt3.setLayout(1,1);//布局在(1,0)
        plt3.setCrossCells(1,2);//跨 2 列
        plt4.setLayout(2,1);//布局在(2,1)
        plt5.setLayout(2,2);//布局在(2,2)
        plt1.setOption("MA1","x-format","time");//设置 X 为时间轴
        plt2.setOption("MA2","x-format","time");//设置 X 为时间轴
        plt3.setOption("MA3","x-format","time");//设置 X 为时间轴
        plt4.setOption("MA4","x-format","time");//设置 X 为时间轴
        plt5.setOption("MA5","x-format","time");//设置 X 为时间轴}
    OnBar(ArrayRef<Integer> indexs)
```

```
    {
        plt1.line("MA1",date+time,AverageFC(Close,5));
        plt2.line("MA2",date+time,AverageFC(Close,5));
        plt3.line("MA3",date+time,AverageFC(Close,5));
        plt4.line("MA4",date+time,AverageFC(Close,5));
        plt5.line("MA5",date+time,AverageFC(Close,5));
    }
```

7 清空画图信息

7.1 清空画线指标

```
plt.clear("MA2",x1,x2);
```

- **参数**

参数	类型	说明
x1	Numeric	起始 x 坐标点
x2	Numeric	结束 x 坐标点

- **示例**

```
Params
    Numeric FastLength(5);// 短期指数平均线参数
    Numeric SlowLength(20);// 长期指数平均线参数
Vars
    Series<Numeric> AvgValue1;
    Series<Numeric> AvgValue2;

    Plot plt;
    Global Numeric x1;
Events
    OnInit()
    {
        plt.setOption("MA1","line-display","interval");
        plt.setOption("MA2","line-display","interval");
        plt.setOption("MA1","color",Blue());
    }
    OnBar(ArrayRef<Integer> indexs)
    {
        AvgValue1 = AverageFC(Close,FastLength);
        AvgValue2 = AverageFC(Close,SlowLength);
        plt.line("MA1",AvgValue1);
        plt.line("MA2",AvgValue2);
```

```
        Commentary(Text(CurrentBar));
        if(CurrentBar == 100)
        {
            x1 = Date+Time;
        }
        if(CurrentBar == 200)
        {
            plt.clear("MA1",x1,Date+Time);
            plt.clear("MA2",x1,Date+Time);
            //plt.clear("MA2",x1);
        }

        If(MarketPosition <>1 && AvgValue1[1] > AvgValue2[1])
        {
            Buy(0,Open);
        }

        If(MarketPosition <>-1 && AvgValue1[1] < AvgValue2[1])
        {
            SellShort(0,Open);
        }
        //PlotNumeric("PL",Portfolio_TotalProfit);
    }
```

1.10 事件驱动帮助文档

1、新增域

1.1 公式函数域(Defs)

除了可以定义用户函数外，在用户公式里面，也可以定义函数，这里的函数是公式函数，统一定义在 Defs 的域内。

1.2 事件函数域(Events)

事件驱动功能增加了很多事件的函数，用户根据需求，把需要处理的事件函数的实现放在 Events 域内。事件函数用户无法定义，只能实现。

1.3 域兼容

Begin/End 是业务代码域，相当于现在的 OnBar 事件。Begin/End 与 Events 不能同时共存，只能选择一种方式。

2、事件函数

2.1 初始化事件(OnInit)

a)由策略执行前的初始化事件，只运行一次
b)只能依赖或使用 Global 等全局变量、全局设置信息
c)可以订阅数据，数据准备等操作

```
OnReady()
{
    //用户初始化逻辑，主要用于数据源的相关设置，后面可以支持数据预处理
}
```

2.3 Bar 数据更新事件(OnBar)

当 Bar 更新变化时驱动，参数 indexs 表示更新的图层编号数组

```
OnBar(IntegerArrayRef indexs)
{
    //用户业务逻辑
}
```

2.4 Bar 数据开始事件(OnBarOpen)

当 Bar 第一次生成时驱动，参数 indexs 表示更新的图层编号数组

```
    OnBarOpen(IntegerArrayRef indexs)
    {
        //用户业务逻辑
    }
```

2.5 Bar 数据结束事件(OnBarClose)

当下一个 Bar 开始之前，最后一次当前 bar 驱动，参数 indexs 表示更新的图层编号数组

```
    OnBarOpen(IntegerArrayRef indexs)
    {
        //用户业务逻辑
    }
```

2.6 账户持仓更新事件(OnPosition)

当账户持仓更新时驱动，参数 pos 表示更新的持仓结构体

```
OnPosition(PositionRef pos)
{
    //用户业务逻辑
}
```

2.7 账户委托更新事件(OnOrder)

当委托更新时驱动，参数 ord 表示更新的委托结构体

```
OnOrder(OrderRef ord)
{
    //用户业务逻辑
}
```

2.8 账户成交更新事件(OnFill)

当成交更新时驱动，参数 ordFill 表示更新的成交结构体

```
OnFill(FillRef ordFill)
{
    //用户业务逻辑
}
```

2.9 定时器更新事件(OnTimer)

当定时器更新时驱动，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值

```
OnTimer(Integer id,Integer millsecs)
{
    //用户业务逻辑
}
```

2.10 策略账户持仓更新事件(OnStrategyPosition)

当策略账户仓更新驱动，策略账户仓是指本策略当日成交累计量，是一个相对持仓，参数 pos 表示持仓结构体

```
OnStrategyPosition(PositionRef pos)
{
    //用户业务逻辑
}
```

2.11 通用事件(OnEvent)

当订阅了通用事件，有事件是驱动，参数 name 表示事件名称，evtValue 表示事件内容

```
OnEvent(StringRef name,MapRef<String,String> evtValue)
{
    //用户业务逻辑
}
```

2.12 退出事件(OnExit)

当策略退出时驱动

```
OnExit()
{
    //用户业务逻辑
}
```

4、新增函数

3.1 订阅/退订 Bar

- **函数：**

```
//添加图层
Integer SubscribeBar(String symbol, String frequency, Numeric beginTime = 0);
//删除图层（只能删除公式添加的图层）
Bool UnsubscribeBar(Integer layerId);
```

- **参数：**

参数名	类型	说明
symbol	String	合约代码
frequency	String	周期
beginTime	Numeric	开始时间，<=0 从当前时间开始
layerId	Integer	图层序号

- **返回值类型：**

SubscribeBar：返回图层序号，>=0 表示成功，否则表示失败
UnsubscribeBar：表示删除是否成功

- **语法使用：**

```
Integer layerId;
layerId = SubscribeBar("rb000.SHFE", "10s", 20190501.0930);
```

```
UnsubscribeBar(layerId);
```

3.2 创建/销毁 Timer

- **函数:**

```
//创建定时器
Integer CreateTimer(Integer intervalMillSecs, Numeric beginTime=0,Integer triggerCount=-1);
//销毁定时器
Bool StopTimer(Integer id);
```

- **参数:**

参数名	类型	说明
intervalMillSecs	Integer	时间间隔，单位毫秒
begin_time	Numeric	开始时间，默认当前时间
triggerCount	Integer	触发次数，默认无限次

- **返回值类型:**

id 代表定时器的变化，如果 id>=0 表示成功，否则表示失败

- **语法使用:**

```
id = CreateTimer(5*1000);
StopTimer(id);
```

3.3 交易订阅

- **函数:**

```
//订阅交易数据（委托和成交），根据操作源 ID 订阅
//如果没有订阅，默认只收当前策略的产生交易数据
Bool A_SubscribeTradeByCreateId(Integer createId);
```

- **参数:**

参数名	类型	说明
createId	Integer	操作源 ID

- **返回值类型:**

表示订阅是否成功

- **语法使用:**

```
A_SubscribeTradeByCreateId(Enum_ALLOrder);
```

3.4 查询 Bar

- **函数:**

```
//查询当前图层指定回溯的 Bar
Bool GetBar(BarRef myBar,Integer back=0);
```

- **参数:**

参数名	类型	说明
myBar	Bar	Bar 结构体对象
back	Integer	回溯值
triggerCount	Integer	触发次数，默认无限次

- **返回值类型:**

表示是否查询成功

- **语法使用:**

```
Bar myBar;
id = GetBar(myBar,0);
```

3.5 查询 Tick

- **函数:**

```
//查询当前图层合约的最新 Tick
Bool GetTick(TickRef myTick);
//查询指定合约的最新 Tick
Bool GetTick(String symbol,TickRef myTick);
```

- **参数:**

参数名	类型	说明
symbol	String	合约代码
myTick	Tick	Tick 结构体对象

- **返回值类型:**

表示是否查询成功

- **语法使用:**

```
Tick myTick;
```

```
id = GetBar(myTick,0);
```

3.6 查询持仓

- **函数:**

```
//查询指定账户在当前图层合约上的持仓
Bool A_GetPosition(PositionRef mypos, Integer accountIndex = 0);
//查询指定账户指定合约的持仓
Bool A_GetPosition(String symbol,PositionRef mypos, Integer accountIndex = 0);
```

- **参数:**

参数名	类型	说明
symbol	String	合约代码
mypos	Pos	通过参数返回的持仓
accountIndex	Integer	资金账户索引

- **返回值类型:**

表示是否查询成功

- **语法使用:**

```
Position mypos;
A_GetPosition(mypos);
```

3.7 查询委托

- **函数:**

```
//根据报单索引查询委托
Bool A_GetOrder(Integer orderId,OrderRef myorder, Integer accountIndex = 0);
//查询指定账户的所有未完成委托的报单索引
Bool A_GetUnFillOrderIDs(IntegerArrayRef orderIds,String filterByCreateSource=String(), Integer accountIndex = 0) = 0;
//查询指定账户指定合约的所有未完成委托的报单索引
Bool A_GetUnFillOrderIDs(String symbol, IntegerArrayRef orderIds,String filterByCreateSource=String(),Integer accountIndex = 0) = 0;
```

- **参数:**

参数名	类型	说明
orderId	Integer	报单索引
myorder	Order	通过参数返回的委托

参数名	类型	说明
accountIndex	Integer	资金账户索引
orderIds	IntegerArrayRef	通过参数返回的报单索引
symbol	String	合约代码
filterByCreateSource	String	根据操作源过滤，空不过滤

- **返回值类型：**

表示是否查询成功

- **语法使用：**

```
Order myorder;
A_GetOrder(1323412, myorder);
```

3.8 查询成交

- **函数：**

```
//查询指定委托的成交单个数
Integer A_GetFillCount(Integer orderId);
//查询指定委托的第 seq 个成交
Bool A_GetFill(Integer orderId, Integer seq, FillRef myfill);
```

- **参数：**

参数名	类型	说明
orderId	Integer	报单索引
seq	Integer	当前成交在委托单序号
myfill	Fill	通过参数返回的成交

- **返回值类型：**

A_GetFillCount：返回指定委托的成交单个数

A_GetFill：表示是否查询成功

- **语法使用：**

```
Fill myfill;
A_GetFill(1323412, 0 , myfill);
```

3.9 委托发单

- **函数：**

//指定账户指定图层下单

Bool A_SendOrderEx(Integer BuyOrSell, Integer EntryOrExit, Numeric lot, Numeric price, IntegerArrayRef orderIds, Integer accountIndex = 0);

//指定账户指定合约下单

Bool A_SendOrderEx(const String& symbol, Integer BuyOrSell, Integer EntryOrExit, Numeric lot, Numeric price, IntegerArrayRef orderIds, Integer accountIndex = 0);

- **参数:**

参数名	类型	说明
BuyOrSell	Integer	买卖方向
EntryOrExit	Integer	开平标志
lot	Numeric	手数
price	Numeric	价格
orderIds	IntegerArrayRef	通过参数返回的报单索引
accountIndex	Integer	资金账户索引
symbol	String	合约代码

- **返回值类型:**

表示下单请求是否发送成功

- **语法使用:**

```
IntegerArray orderIds;
A_SendOrderEx("rb1910.SHFE", Enum_Buy, Enum_Entry, 1, 3700, orderIds);
```

3.10 委托撤单

- **函数:**

//撤指定报单索引的委托单

Bool A_DeleteOrderEx(Integer orderId);

//撤指定账户指定图层对应合约的所有委托

Bool A_DeleteAccountOrder(Integer accountIndex = 0);

//撤指定账户指定合约的所有委托单

Bool A_DeleteAccountOrder(const String& symbol, Integer accountIndex = 0);

- **参数:**

参数名	类型	说明
orderId	Integer	报单索引

参数名	类型	说明
accountIndex	Integer	资金账户索引
symbol	String	合约代码

- **返回值类型:**

表示撤单请求是否发送成功

- **语法使用:**

```
A_DeleteAccountOrder("rb1910.SHFE");
```

3.11 通用事件

3.11.1 订阅/退订

- **函数:**

```
//订阅通用事件，该对象为事件订阅者
Bool SubscribeEvent(String name);
Bool SubscribeEvent(String name,ArrayRef<String> keys);
//退订通用事件
Bool UnsubscribeEvent(String name);
```

- **参数:**

参数名	类型	说明
name	String	通用事件名称
keys	ArrayRef<String>	事件键名列表，如果指定方便事件发送器参考展示

- **语法使用:**

```
Vars

OnInit()
{
    SubscribeEvent("系统-选股事件");
}

OnEvent(StringRef name,MapRef<String,String> evtValue)
{
  //用户业务逻辑
}
```

3.11.2 事件发布

- **函数:**

```
//发布通用事件，发布事件的对象为事件生产者
Bool PublishEvent(String name,Map<String,String> values,String target);
```

- **参数:**

参数名	类型	说明
name	String	通用事件名称
values	Map	通用事件内容
target	String	通用事件接收目标

3.11.3 示例--公式与系统组件交互

```
//事件生产者（选股事件-自定义板块合约），事件订阅者是行情报价或移动端
Vars
  Map<String,String> context;
  Global String tmpSyms;
Events
    OnExit()
    {
        Range[0:DataSourceSize -1]
        {
            tmpSyms = tmpSyms+Symbol+",";
        }
        context["合约集合"]=tmpSyms;//选股合约
        context["板块名称"]="选股板块_自定义 1";//自定义行情设置，格式是：一级板块_二级板块
        context["添加方式"]="override";//更新方式：override,append
        PublishEvent("系统-选股事件",context,"行情报价");//发送选股事件到行情报价
        PublishEvent("系统-选股事件",context,"移动端");//发送选股事件到移动端
    }

//事件生产者（选股事件-记录标记），事件订阅者是移动端
Vars
  Map<String,String> context;
  Global String tmpSyms;
  Global String tmpIds;
  Global String tmpDesps;
Events
```

```
OnExit()
{
    Range[0:DataSourceSize -1]
    {
        tmpSyms = tmpSyms+Symbol+",";
        tmpIds=tmpIds+Text(IntPart(Rand(1,10)))+",";
        tmpDesps=tmpDesps+Text(SystemDateTime,9)+",";
    }
    context["合约集合"]=tmpSyms;//选股合约
    context["标记编号集合"]=tmpIds;//合约标记 ID【1-10】
    context["标记内容集合"]=tmpDesps;//标记内容
    PublishEvent("系统-标记事件",context,"行情报价");
}
```

3.11.4 示例--公式与公式交互

```
//事件订阅者
Vars
    Array<String> keys;
    Array<Integer> ids;
    String mySide;
    String mySym;
    Numeric myVolume;
    Numeric myPrice;
Events
    //此处实现事件函数

    //初始化事件函数，策略运行期间，首先运行且只有一次，应用在订阅数据等操作
    OnInit()
    {
        keys[0]="合约代码";
        keys[1]="信号方向";
        keys[2]="信号数量";
        keys[3]="信号价格";
        SubscribeEvent("信号代理",keys);
    }

    //通用事件触发函数，参数 evtName 为事件名称，参数 evtValue 为事件内容
    OnEvent(StringRef evtName,MapRef<String,String> evtValue)
    {
        mySide=evtValue["信号方向"];
        mySym=evtValue["合约代码"];
        myVolume=Value(evtValue["信号数量"]);
        myPrice=Value(evtValue["信号价格"]);
```

```
        if(mySide=="买入开仓")
        {
            A_Buy(mySym,myVolume,myPrice,ids);
        }
        Else if(mySide=="卖出平仓")
        {
            A_Sell(mySym,myVolume,myPrice,ids);
        }Else if(mySide=="卖出开仓")
        {
            A_SellShort(mySym,myVolume,myPrice,ids);
        }Else if(mySide=="买入平仓")
        {
            A_BuyToCover(mySym,myVolume,myPrice,ids);
        }
        //Print2Quote(evtName,TextMap(evtValue));
    }

//事件生产者
Params
    Numeric FastLength(5);// 短期指数平均线参数
    Numeric SlowLength(20);// 长期指数平均线参数
    Integer pTradeVolume(1);//开仓数量
Vars
    Series<Numeric> AvgValue1;
    Series<Numeric> AvgValue2;
    Map<String,String> evt;
    Global Integer lastSignalBar;
Events
    OnBar(ArrayRef<Integer> indexs)
    {
        AvgValue1 = AverageFC(Close,FastLength);
        AvgValue2 = AverageFC(Close,SlowLength);
        PlotNumeric("MA1",AvgValue1);
        PlotNumeric("MA2",AvgValue2);

        If(MarketPosition ==-1 && AvgValue1[1] > AvgValue2[1])
        {
            evt["信号方向"]="买入平仓";
            evt["信号数量"]=Text(abs(CurrentContracts));
            BuyToCover(abs(CurrentContracts),Open,Enum_Signal_NotSend);
        }
        Else if(MarketPosition ==1 && AvgValue1[1] < AvgValue2[1])
        {
            evt["信号方向"]="卖出平仓";
```

```
            evt["信号数量"]=Text(CurrentContracts);
            Sell(CurrentContracts,Open,Enum_Signal_NotSend);
        }
        Else If(MarketPosition==0)
        {
            If(AvgValue1[1] > AvgValue2[1])
            {
                Buy(pTradeVolume,Open,Enum_Signal_NotSend);
                evt["信号方向"]="买入开仓";
                evt["信号数量"]=Text(pTradeVolume);
            }

            If(AvgValue1[1] < AvgValue2[1])
            {
                SellShort(pTradeVolume,Open,Enum_Signal_NotSend);
                evt["信号方向"]="卖出开仓";
                evt["信号数量"]=Text(pTradeVolume);
            }
        }
        If(BarStatus==2 and IsTradeEnabled and lastSignalBar!= CurrentBar and evt["信号方向"]<>InvalidString)
        {
            evt["合约代码"]=Symbol;
            evt["信号价格"]=Text(Open);
            PublishEvent("信号代理",evt,"所有订阅者");
            lastSignalBar=CurrentBar;
        }
    }
```

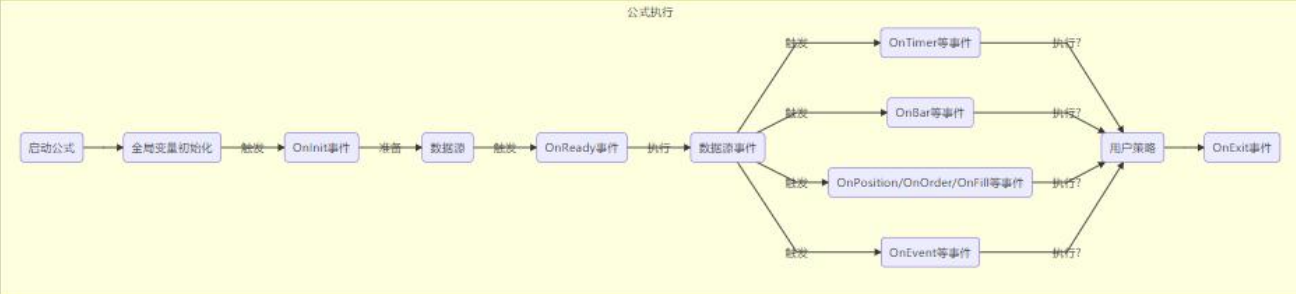
3.11.5 示例--事件发送器与公式交互

运行例 2 的事件订阅者策略，可以在事件发送器找到该策略单元发送事件，即这里事件发送器是事件生产者。

5、要点说明

5.1 公式执行流程图

公式执行



5.2 数据源变化影响

a) SubscribeBar/UnsubscribeBar，数据源会重新执行，如果在 OnInit 事件函数外使用，一定要谨慎控制，否则可能出现死循环。

b) 数据源重新执行时，全局变量不会重置，只有公式重新执行才会重置。

5.3 回测说明

回测的数据源只包括用户添加和公式 OnInit 添加，其他事件添加不能回测

5.4 交易数据更新顺序

a) 更新原则

持仓-->委托-->成交

b) 默认推送

- 策略单元启动时且已经绑定账户：推送持仓快照、未完成的委托
- 策略单元启动后，当绑定账户时：推送持仓快照、未完成的委托
- 账户断线重连后：推送断线期间的持仓快照变化、委托变化

c) 具体场景

序号	场景	事件顺序	说明
1	委托发送失败	无事件	账户未登录
2	委托发送成功	持仓事件更新 委托事件更新	持仓：未成交的委托量统计，如可平仓量等 委托：信息更新，如状态、报单索引等
3	委托被拒绝	持仓事件更新 委托事件更新	持仓：恢复到委托发送时的状态 委托：信息更新，如状态等
4	委托成功	委托事件更新	委托：信息更新，如状态、报单编号等
5	撤单发送失败	无事件	账户未登录、无效委托单 ID

序号	场景	事件顺序	说明
6	撤单发送成功	持仓事件更新 委托事件更新	持仓：撤单计数等 委托：信息更新，如状态等
7	撤单被拒绝	持仓事件更新 委托事件更新	持仓：恢复到撤单发送时的状态 委托：信息更新，如状态等
8	撤单成功	持仓事件更新 委托事件更新	持仓：未成交的委托量统计，如可平仓量等 委托：信息更新，如状态、撤单时间等
9	成交	持仓事件更新 委托事件更新 成交事件更新	持仓：未成交的委托量统计、持仓量等 委托：信息更新，如状态、成交量、成交时间等 成交：记录更新

6、Demo

6.1 事件驱动模板

```
//-----------------------------------------------------------------
// 简称: MyEventStrategy
// 名称: 事件驱动模板
// 类别: 公式应用
// 类型: 用户应用
// 输出: Void
//-----------------------------------------------------------------
Params
    //此处添加参数
    Numeric p;
Vars
    //此处添加变量
    Numeric avg;
    Global Integer timerId;

Defs
    //此处添加公式函数
    Numeric calcAvg(Numeric a,Numeric b)
    {
        return (a+b)/2;
    }

Events
    //此处实现事件函数
```

```
//初始化事件函数，策略运行期间，首先运行且只有一次
OnInit()
{
    timerId=createTimer(1000);
}

//在所有的数据源准备完成后调用，应用在数据源的设置等操作
OnReady()
{

}

//在新 bar 的第一次执行之前调用一次，参数为新 bar 的图层数组
OnBarOpen(ArrayRef<Integer> indexs)
{

}
//Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
OnBar(IntegerArrayRef indexs)
{
    avg=calcAvg(high,low);
}

//持仓更新事件函数，参数 pos 表示更新的持仓结构体
OnPosition(PositionRef pos)
{

}

//委托更新事件函数，参数 ord 表示更新的委托结构体
OnOrder(OrderRef ord)
{

}

//成交更新事件函数，参数 ordFill 表示更新的成交结构体
OnFill(FillRef ordFill)
{

}

//定时器更新事件函数，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值
OnTimer(Integer id,Integer millsecs)
```

```
        {

        }
    //------------------------------------------------------------------
    // 编译版本:    2019/06/10 090433
    // 版权所有 riv
    // 更改声明 TradeBlazer Software 保留对 TradeBlazer 平台
    //          每一版本的 TradeBlazer 公式修改和重写的权利
    //------------------------------------------------------------------
```

6.2 其他 demo

请查看版本目录 Doc

7、附录

7.1 frequency-周期

String 类型，如"3h"表示 3 小时周期。大于等于"d"时，只支持"1d"、"1w"、"1mon"。

取值	说明
"s"	秒
"m"	分钟
"h"	小时
"d"	天
"w"	周
"mon"	月

7.2 exchange-交易所代码

String 类型

取值	说明
’SHFE’	上海期货交易所
’DCE’	大连商品交易所
’CZCE’	郑州商品交易所
’CFFEX’	中国金融期货交易所

取值	说明
'INE'	上海国际能源交易中心
'SSE'	上海证券交易所
'SZSE'	深圳证券交易所

7.3 Tick-行情快照

对象类型

属性	类型	说明
datetime	Numeric	时间
symbol	String	合约代码
open	Numeric	开盘价
high	Numeric	最高价
low	Numeric	最低价
last	Numeric	最新价
limitUp	Numeric	涨停价
limitDown	Numeric	跌停价
preClose	Numeric	昨收价
preSettlement	Numeric	昨结价
volume	Integer	最新成交量
totalVolume	Integer	总成交量
bidask1	[BidAsk]	1 档行情
bidask2	[BidAsk]	2 档行情
bidask3	[BidAsk]	3 档行情
bidask4	[BidAsk]	4 档行情
bidask5	[BidAsk]	5 档行情

BidAsk

对象类型

属性	类型	说明
bidP	Numeric	买价
bidV	Integer	买量
askP	Numeric	卖价
askV	Integer	卖量

7.4 Bar-Bar 数据

对象类型

属性	类型	说明
datetime	Numeric	开始时间
open	Numeric	开盘价
high	Numeric	最高价
low	Numeric	最低价
close	Numeric	收盘价
turnOver	Numeric	成交金额
volume	Integer	成交量
openInt	Integer	持仓量

7.5 Position-持仓

对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
symbol	String	合约代码
longCurrentVolume	Integer	多头当前持仓

属性	类型	说明
longYesterdayVolume	Integer	多头剩余昨仓
longActiveVolume	Integer	多头未成交的报单净委托量
longActiveCloseVolume	Integer	多头未成交的报单平仓委托量
longCanSellVolume	Integer	多头可平量
longMarketValue	Numeric	多头持仓市值
longAvgPrice	Numeric	多头均价（买均价）
longFloatPorfit	Numeric	多头浮动盈亏（买盈亏）
longUseMarginAmount	Numeric	多头占用的保证金
shortCurrentVolume	Integer	空头当前持仓
shortYesterdayVolume	Integer	空头剩余昨仓
shortActiveVolume	Integer	空头未成交的报单净委托量
shortActiveCloseVolume	Integer	空头未成交的报单平仓委托量
shortCanCoverVolume	Integer	空头可平量
shortMarketValue	Numeric	空头持仓市值
shortAvgPrice	Numeric	空头均价（买均价）
shortFloatPorfit	Numeric	空头浮动盈亏（买盈亏）
shortUseMarginAmount	Numeric	空头占用的保证金

7.6 Order-委托

对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
symbol	String	合约代码
orderId	Integer	报单索引
exchOrderId	String	报单编号（交易所）

属性	类型	说明
createDateTime	Numeric	报单委托时间
volume	Integer	委托量
price	Numeric	委托价
fillVolume	Integer	成交量
fillAmount	Integer	成交金额
side	Integer	买卖方向
combOffset	Integer	开平标志
priceType	Integer	价格类型
hedge	Integer	投机套保
status	Integer	报单状态
createId	Integer	报单源 ID
createSource	String	报单源
cancelSource	String	撤单源
note	String	详细信息

7.7 Fill-成交

对象类型

属性	类型	说明
brokerId	Integer	经济公司 ID
accountId	String	资金账户 ID
symbol	String	合约代码
fillId	String	成交索引
fillVolume	Integer	成交量
fillPrice	Integer	成交价
orderId	Integer	报单索引
exchOrderId	String	报单编号（交易所）

属性	类型	说明
createDateTime	Numeric	报单委托时间
volume	Integer	委托量
price	Numeric	委托价
side	Integer	买卖方向
combOffset	Integer	开平标志
priceType	Integer	价格类型
hedge	Integer	投机套保
createId	Integer	报单源 ID
createSource	String	报单源

7.8 CreateId-操作 ID

Integer 类型

取值	说明
Enum_ExtraOrder	非本地单操作源 ID
Enum_ManualOrder	手工单操作源 ID
Enum_AutoTradeOrder	程序化单操作源 ID
Enum_TBPYTradeOrder	TBPY 单操作源 ID
Enum_TradeHelperOrder	交易助手单操作源 ID
Enum_PosMonitorOrder	头寸监控单操作源 ID
Enum_ALLOrder	所有委托单操作源 ID

7.9 系统事件名称

String 类型

取值	说明
'系统-选股事件'	行情报价或移动端的选股事件
'系统-标记事件'	行情报价的标记事件

7.10 系统事件目标

String 类型

取值	说明
''	空表示当前策略单元
'行情报价'	系统行情报价板的自定义板块
'移动端'	移动端
'所有订阅者'	发布到所有订阅者
公式操作源	公式操作源组成：公式名@工作区名@组名@单元名

1.11 模式交易规范

1、背景

- 为了降低软件使用门槛
- 模式开发与模式使用分离
- 模式开发者通过模式生成器产生模式
- 用户通过模式交易完成复杂场景的应用

2、模式生成

2.1 背景

- 模式由部件(公式)组成，可以灵活组成多种模式
- 根据约定，可以完成与系统功能无缝对接
  如对账户持仓进行平仓，模式如何自动识别仓位数量等

2.2 参数规范

- 参数不分区大小写
- 用户公式参数须有参数说明

- 尽量使用 Enum 类型，方便用户选择
- 约定参数以 p 开头，自定义参数尽量不要以 p 开头

2.3 约定参数

与下单方向有关

- 若从持仓入口，会根据平仓方向，自动导入可平量。如是解锁，导入多空中最小可平量。
- 一般应用在开仓部件，如果是开仓与平仓部件分开的情况，平仓部件无须有方向

参数名	类型	说明
pTradeSide	Enum<String>	买入开仓、卖出平仓，卖出开仓，买入平仓，解锁

与数量有关

- 若参数一致，会自动导入相关数值
- Numeric 或 Integer

参数名	类型	说明
pTradeVolume	Numeric	委托数量(只限批量模式)
pTradeCapitalRatio	Numeric	风险度
pTradeAmount	Numeric	保证金
pTradeMarketValue	Numeric	市值
pTradeLever	Numeric	杠杆
pTradeRatio	Numeric	比例

与价格有关

- 若参数一致，会自动导入相关价格
- 只应用在批量模式，叠加无效，若违背规则，不允许下单

参数名	类型	说明
pLastPrice	Numeric	当前 tick 的最新价
pAskPrice	Numeric	当前 tick 的卖价
pBidPrice	Numeric	当前 tick 的买价
pOpenPrice	Numeric	当前 tick 的开盘价
pHighPrice	Numeric	当前 tick 的最高价
pLowPrice	Numeric	当前 tick 的最低价

参数名	类型	说明
pPreClosePrice	Numeric	当前 tick 的昨收价

与合约有关

- 若以 p 开头，以 Symbol 结束的 String 型变量，填充时提供小键盘输入
- 合约是 code.exch 格式属性，如：rb888.SHFE

参数名	类型	说明
pXXXSymbol	String	具体合约
pXXXIndustry	Array<String>	板块合约集合
pXXXIndustryId	Array<String>	板块合约 Id 集合

与时间有关

- 若以 p 开头，以 DateTime 结束的 Numeric 型变量，填充时提供时间组件，格式“年月日.时分秒”，如 20200724.215959

参数名	类型	说明
pXXXDateTime	Numeric	日期时间

与基础数据有关

- 若以 p 开头，以 DicKey 结束的 String 一位数组型变量，填充时提供基础数据组件，可以通过分类，选择键名和二级键名，生成格式“键名，二级键名序号 1，二级键名序号 N...”

参数名	类型	说明
	pXXXDicKey	Array<String>

2.4 生成流程

- **添加部件(公式)**
  - ✧ 从常用公式添加
  - ✧ 从小键盘添加，来自公式管理器
- **修改参数**
  - ✧ 根据模式需要，修改公式参数
- **设置必填参数**
  - ✧ 必填参数会默认提醒用户

- **填写模式名称与说明**
- **选择适用周期**
  - ✧ 这个是默认的，在下单时可以再修改
- **选择数据源方式**
  - ✧ 一个合约一个模式单
- **填写历史样本 Bar 数**
  - ✧ 基于下单时间，增加一定量历史的样本，估算出样本的起始时间
  - ✧ 模式单一旦生成，样本的起始时间已确定，无法修改
- **填写信号起始 Bar 数**
  - ✧ 控制信号的起始位置
  - ✧ 避免因最大回溯变化影响信号
  - ✧ 判断启动时间点 IsStartBarTime

```
//判断为启动时间
If(IsStartBarTime())
{
//逻辑处理
}
```

- **信号开仓次数**
  - ✧ 控制开仓次数，满足条件后自动退出

2.5 模式管理

可见范围

- 列表可见：是默认行为，列表是模式交易的列表选项
- 菜单可见：非模式交易组件的菜单入口
- 无：模式交易不可见

3、系统部件

3.1 虚拟开仓 VirtualOpen

- 背景：由于模式可以是只平仓，没有开仓。但模式的执行是基于程序化基础上完成的，所以没有仓位是无法平仓的。
- 作用：该公式是协助用户默认开仓，可以作为开仓部件，但该信号不会有实质发单行为

- 用户也可以参考该公式自行实现

3.2 模式代理 PatternProxy

- **背景：**
  - ✧ 由于程序化信号不能确保委托单是否执行成功，需要借助于头寸监控或交易助手。
  - ✧ 由于程序化的历史信号不会发单，需要借助于头寸监控同步持仓。
  - ✧ 为了减少模式交易对其他部件的依赖，借助于 PatternProxy 部件，可以确保用户公式信号的实质执行完成或出错停止提醒，也确保可以选择当天的信号或历史信号要不要到执行
- **原理：**
  - ✧ 模式单的信号部件只需要完成信号发送，PatternProxy 会代理模式单所有信号，如果没有 PatternProxy 不保证信号执行成功与否
  - ✧ PatternProxy 会动态合并信号，执行当前仓位与昨日仓位的差异部分
  - ✧ 代理的价格：对手价
  - ✧ 代理其他参数

参数名	类型	说明
maxVolume	Integer	单次最大委托量，默认是仓差部分
cancelPrcOffset	Integer	撤单价格的偏移条件
cancelMinSecs	Integer	撤单最小时间间隔
cancelMaxSecs	Integer	撤单最大时间间隔
cancelLimitCnt	Integer	撤单总次数限制

4、模式机制

4.1 原理背景

模式是基于程序化基础上完成，模式的运行机制即程序化运行机制

4.2 信号产生

信号委托

通过 Buy、Sell、SellShort、BuyToCover 等发出的委托为信号委托

账户委托

通过 A_SendOrder、A_SendOrderEx、A_Buy、A_Sell、A_SellShort、A_BuyToCover 等账户函数发出的委托为账户委托

4.3 信号执行

信号委托

如果没有代理公式，只执行启动后的信号

如果有代理公式，则执行理论仓位与该模式成交仓位（不是账户仓位）的差异

- 忽略历史信号：忽略下单时间点前的信号
- 不忽略历史信号：所有信号都有效

账户委托

自行实现控制委托

自行实现管理持仓

4.4 状态恢复

信号委托

由于基于程序化，每次信号都是确定，信号闪烁除外

账户委托

每次启动时，可以查询该模式委托、成交、模式账户仓位、账户仓位等信息，自行实现恢复缓存、控制委托等

4.5 有效性

信号委托

信号一直有效

如果使用代理，一旦交易，只能在当前交易日内有效，不能跨天

账户委托

账户的委托仅限单个交易日有效

5、退出机制

5.1 背景

- 模式运行是在程序化基础上完成的，是否完成需要明确的终止条件
- 自动终止：分为模式单设置控制、公式控制，而且任一满足即可终止
- 人工终止：可以随时停止与运行

5.2 模式单设置退出

设置信号开仓次数

- 为-1 不控制
- 只有达到开仓限制，且仓位为 0 时，才退出
- 如果只是开仓部件，建议用公式公式
- A 函数不受影响

5.3 公式控制退出

部件终止

- AddStrategyFlag(Enum_Strategy_Finish)可以控制部件终止

- 部件退出，只是忽略后面的信号，该部件在模式没有退出之前仍然继续运行

//根据信号控制部件退出

```
if(SignalSize() >= 1 /*开仓次数*/ && ExitSize() >= 1 /*平仓次数*/)
{
   AddStrategyFlag(Enum_Strategy_Finish); //退出部件
}
```

模式终止

- 若模式的每一个部件，都有退出机制，则该模式也自然满足退出机制。
- 若非 PatternProxy 都有退出机制，PatternProxy 也会自行退出
- 若某一个部件出错或其他情况，以下两个任一均可以使模式退出

a) AddStrategyFlag(Enum_Strategy_Error,"错误信息") 可以控制部件终止

b) Stop()

```
//1.根据委托状态被拒是，退出该模式
If(order.status==Enum_Deleted)
{
   AddStrategyFlag(Enum_Strategy_Error,order.note); //退出模式
}
//2.根据委托状态被拒是，退出该模式
If(order.status==Enum_Deleted)
{
   Stop();//退出模式
}
```

6、模式监控

关系

不受头寸监控器监控

状态

- **待运行**
  未启动，或软件重启前没有完成的模式单状态
- **运行中**
  正在运行的模式单状态
- **运行完成**
  正常运行完成模式单状态
- **出错停止**
  异常退出的模式单状态

权限

部分系统模式为模式下单权限

其他系统模式和用户模式均为程序化权限

创建时间

即模式单生成时间

样本范围

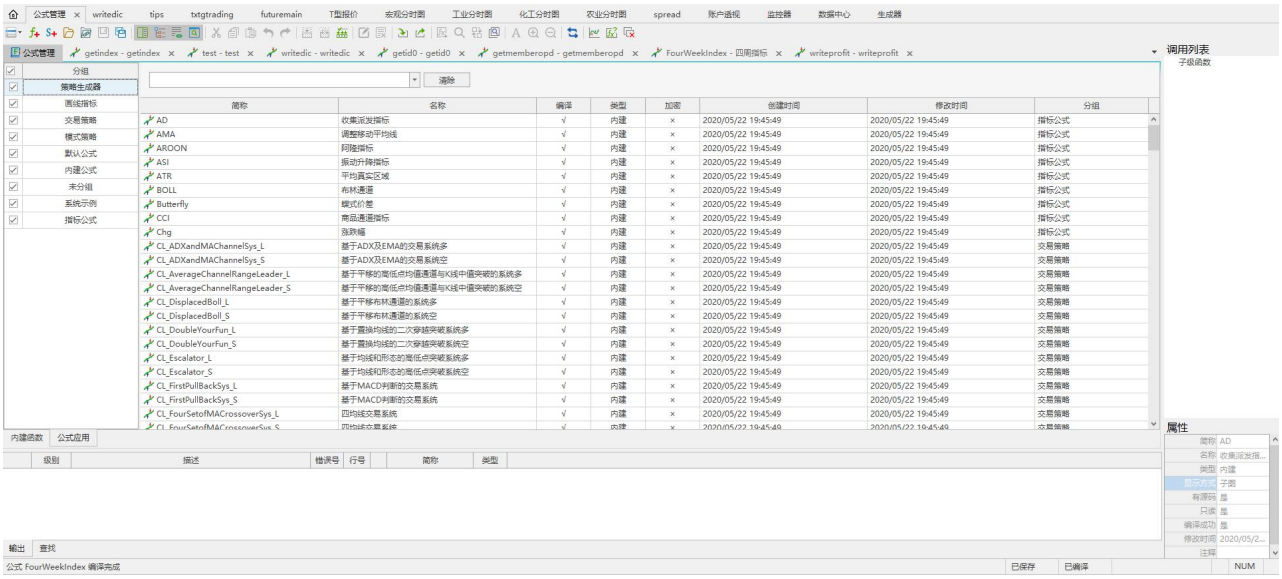
样本量在下单确定，基于下单时间，增加设置历史样本 Bar 数，估算出样本的起始时间，不能修改；周期范围不允许修改。

二、公式管理

2.1 公式管理器

公式管理是对交易开拓者公式和函数进行集中管理的模块，可以在公式管理器中新建、打开、编辑、编译、导入导出公式等功能。公式管理分页显示各类公式，通过点击标签页进行切换。点击工具栏或在导航页点击“ 

”，即打开公式管理应用。



公式管理的工具栏可用功能如下：

注意：1108 版本之后公式系统需要泛型转换，如果在公式导入的时候没有选择泛型转换，也可以在工具栏点击泛型转换的图标实现。

公式管理的菜单功能列表如下：

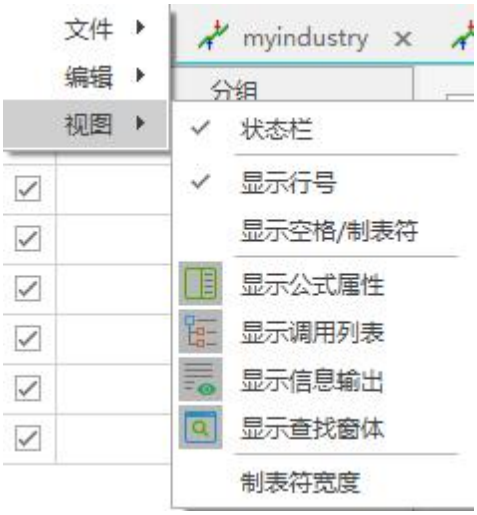
文件：


文件
编辑
视图
新建用户函数
新建公式应用
打开公式
打开相关
保存公式
另存公式为
保存全部公式
编译公式
批量编译
全部编译
公式分组管理
批量删除
导入
导出
序列变量范围
关闭所有编辑页

编辑：


文件
编辑
视图
myindustry
撤消
重复
剪切
复制
粘贴
自动完成
行注释 Ctrl+L
取消行注释 Ctrl+U
块注释
跳到行
选中文本小写
选中文本大写
删除
全选
查找
替换
全文搜索

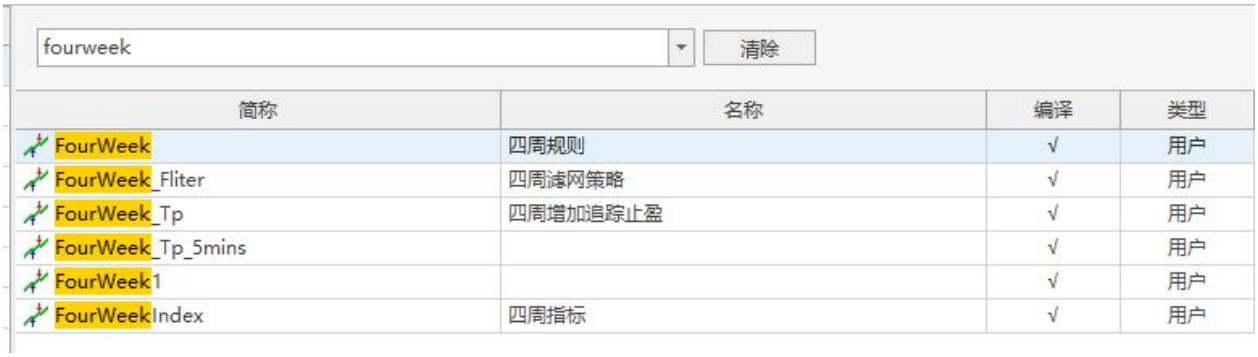
视图：



新建用户函数：打开新建用户函数窗口；
新建公式应用：打开新建公式应用窗口；
打开公式：打开选中公式的编辑窗口；
打开相关：打开选中公式和公式调用列表中所有函数的编辑窗口；
显示公式属性：在窗口右边显示或隐藏选中公式的属性窗口；
显示调用列表：在窗口右边显示或隐藏选中公式调用的所有用户函数编码，以及这些用户函数的下级用户函数级联调用；
显示信息输出：在窗口下边显示或隐藏信息输出窗口；
显示查找窗体：在窗口下边显示或隐藏查找窗口；
批量编译：批量编译公式管理列表中选中的用户函数或公式；
全部编译：全部编译公式管理列表中的所有用户函数和公式；
批量删除：批量删除公式管理列表中选中的用户函数或公式；
导入:将备份文件中的用户公式/函数的源码信息导入到系统中,支持 TradeBlazer 的有源码公式格式(*.fbk)和 TB 极速版的有源码用户公式格式(.xml)；
导出：将用户公式/函数的源码信息导出到指定文件中(*.fbk)；
序列变量范围：限制序列变量的最大回溯范围，以节约系统资源（也可在公式中声明）。序列变量的最大回溯范围设置为公式中最大引用比较合理，譬如公式中有语句 AvgValue2[8] ，Data1.AvgValue1[30]，则序列变量最大回溯范围设置为 30 比较合理。
关闭：关闭所有打开的公式/函数编辑页；
另存公式为：将当前选中的公式另存为其它公式名称；
排序：支持按公式名称、简称、类型或修改时间等属性进行排序；
全文搜索：在所有的系统函数或公式应用文件中搜索关键字；
筛选：将鼠标放置在标题栏上的，点击标题右边的“”图标，在弹出的下拉框中选择筛选条件，即可进行筛选。

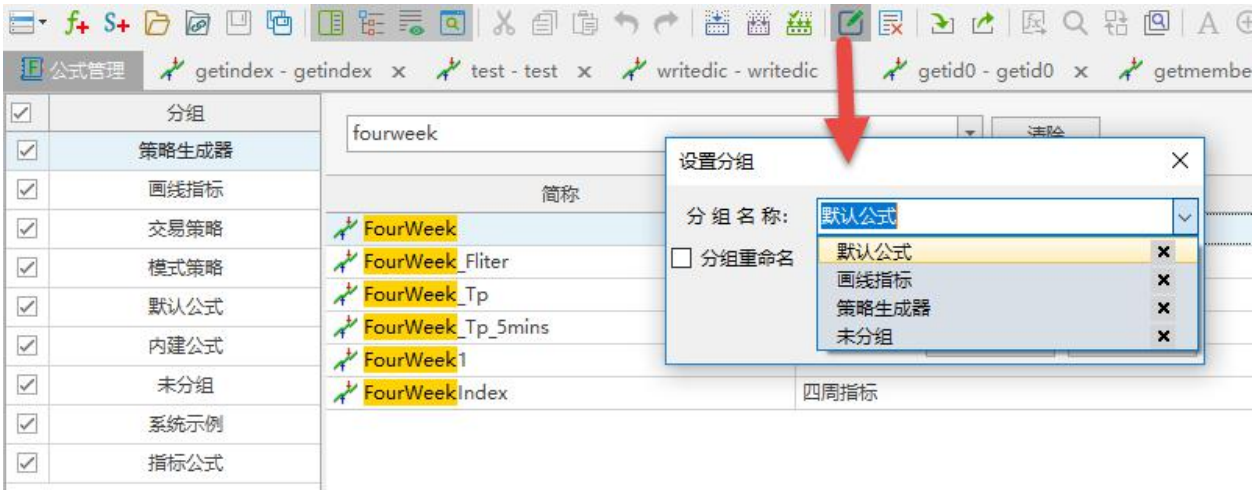
查找：

从公式应用/用户函数中模糊搜索查找内容，如果搜索到了则展示在列表中，并且查找内容黄色显示；



公式分组管理：

点击工具栏公式分组管理，可以弹出如下界面，对公式进行分组。

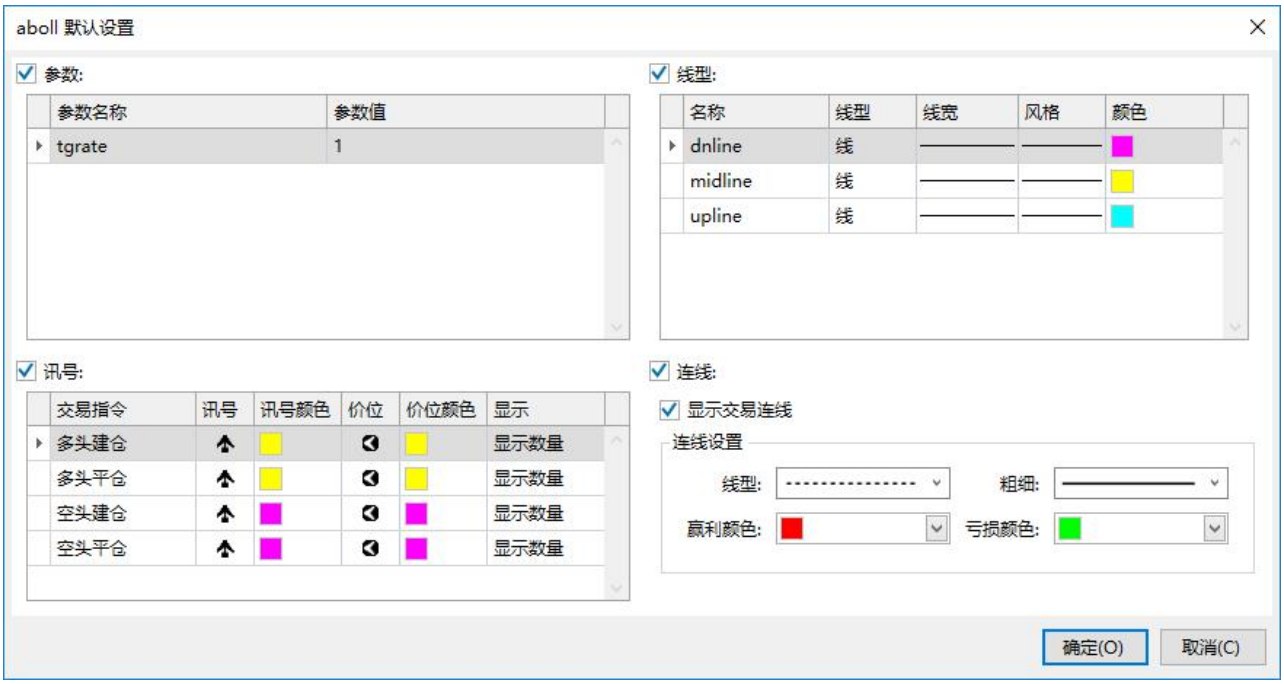


打开系统函数帮助：

导航到系统函数的索引页。

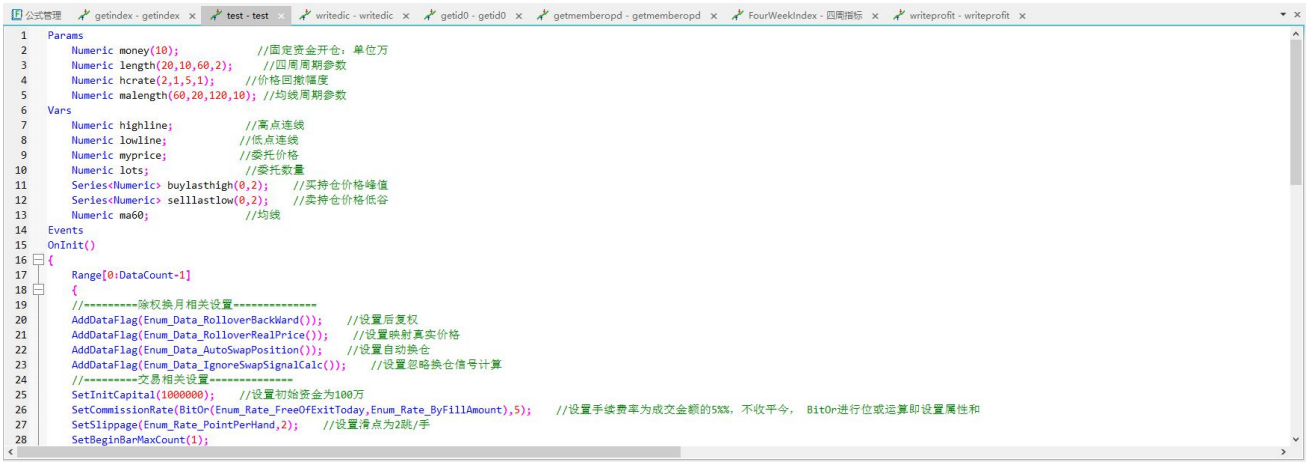
公式设置：

打开公式设置，可以对参数/线型/讯号/连线进行设置。



2.2 公式编辑器

公式编辑器是对 TradeBlazer 进行编辑、编译的模块，公式编辑器与公式管理共用主界面，当切换到打开的公式代码工作区式，系统进入公式编辑器模式，公式编辑器界面如下：



公式编辑器的工具栏和菜单栏功能列表如下：

新建用户函数：打开新建用户函数窗口；

新建公式应用：打开新建公式应用窗口；

打开相关：打开当前公式调用列表中所有调用函数的编辑窗口；

保存公式：保存当前编辑公式和函数。

保存全部公式：保存所有打开的编辑公式和函数。

显示公式属性：在窗口右边显示或隐藏选中公式的属性窗口；

显示调用列表：在窗口右边显示或隐藏选中公式调用的所有用户函数编码，以及这些用户函数的下级用户函数级联调用；

显示信息输出：在窗口下边显示或隐藏信息输出窗口；

剪切：如 Windows 标准剪切操作，剪切选中的文本；

复制：如 Windows 标准复制操作，复制选中的文本；

可以使用 Ctrl + D 复制本行内容并添加到下一行；

粘贴：如 Windows 标准粘贴操作，粘贴剪贴板中的文本；

编译公式：对于用户公式，编译并保存当前公式；对于用户函数，编译并保存当前　函数并编译其相关用户函数；

批量编译：编译编辑器中所有打开的用户函数和公式，并编译用户函数影响到的公式；

导入:将备份文件中的用户公式/函数的源码信息导入到系统中,支持 TradeBlazer 的有源码公式格式(*.fbk)和 TB 极速版的有源码用户公式格式(.xml)；

导出：将用户公式/函数的源码信息导出到指定文件中(*.fbk)；

序列变量范围：限制序列变量的最大回溯范围，以节约系统资源（也可在公式中声明），序列变量的最大回溯范围设置为公式中最大引用比较合理；

另存公式为：将当前选中的公式或函数另存为其它名称；

打开选中函数：如果在公式中选中的字符串是已存在的用户函数名称，则可打开该用户函数；

查找：在当前公式中查找；

替换：替换当前公式中的某些内容；

全文搜索：在所有的系统函数或公式应用文件中搜索关键字；

字体：设置当前公式字体；

放大：放大当前公式窗口字体；或者按 ctrl+鼠标滚轮向上滚动放大当前公式窗口字体；

缩小：缩小当前公式字体；或者按 ctrl+鼠标滚轮向下滚动缩小当前公式窗口字体；

关闭：关闭所有打开的公式/函数编辑页；

删除：如 Windows 标准删除操作，删除选中的文本；

全选：选择当前公式编辑信息；
打开系统函数帮助：打开公式编辑区中选中系统函数的帮助文件；
双击信息输出区列表项，可直接定位到对应公式的指定行。

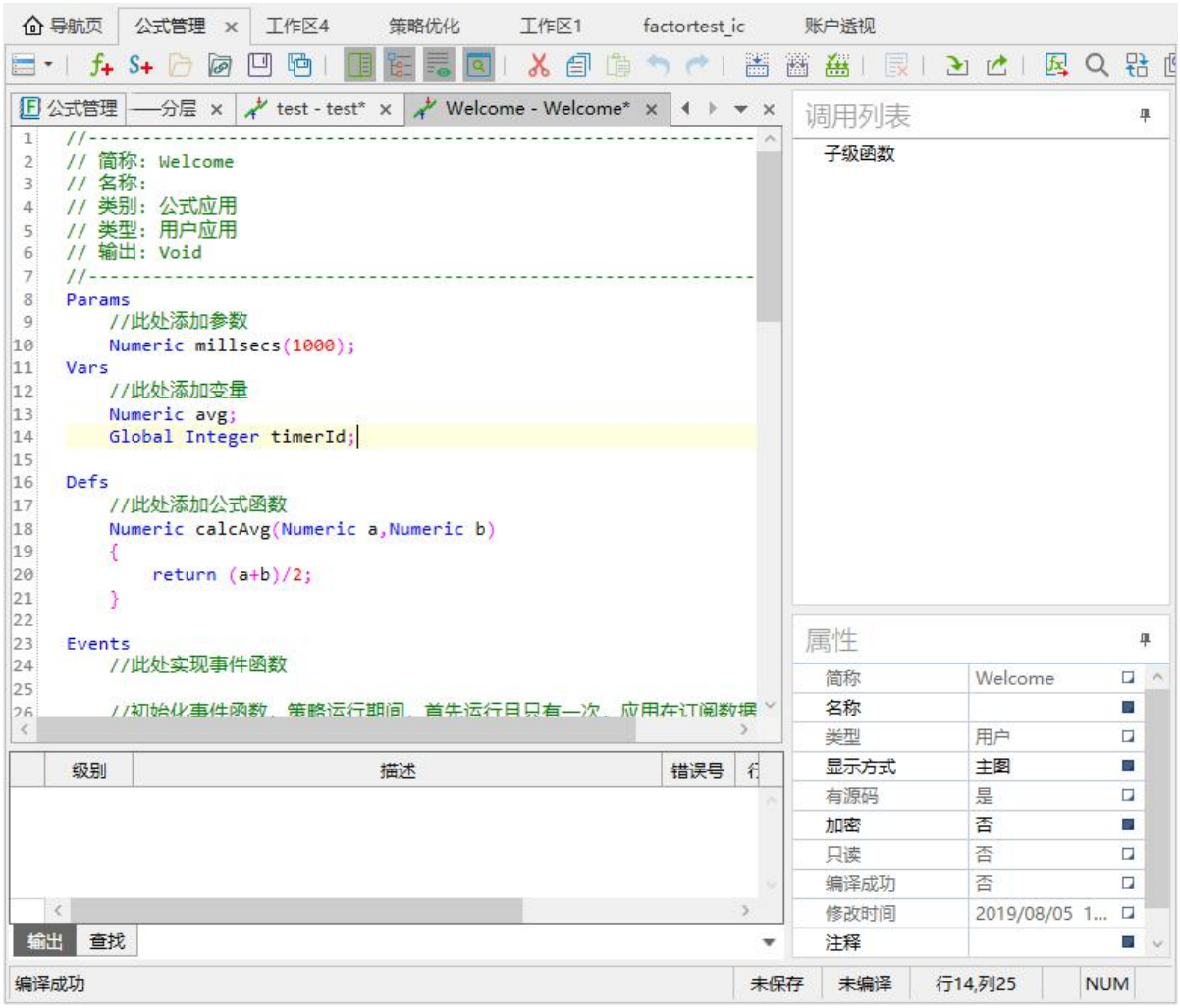
2.3 TB 公式

下面是编写公式“Welcome”，公式内容为显示“TB！”的操作步骤：

新建公式

1.点击工具栏的新建公式应用按钮“ ”，打开新建公式应用窗口。

如图所示，输入公式的简称、名称、注释；
【说明】新建公式应用窗口的填写
公式简称为必填项，命名需要符合“命名规则”的要求；
名称（选填项），它是对简称的补充命名，可以为中文、英文、数字等，便于交易者按自己的习惯来辨别；
选择模板（选填项），模板的可选项为“空”、“技术分析”或“交易策略”，可根据公式编写需求选择；
输入注释（选填项），注释框内中可填写交易者设计编写此公式的思路以及一些概括性的描述，方便以后使用与修改。
2.单击新建公式应用窗口中的“确定”按钮，打开公式编辑器，进入公式编辑，如下图所示；



3、输入如下公式内容，然后单击工具栏上的编译公式按钮 ；

```
Params
Vars
    Series<Numeric> atr;
Events
    onBar(ArrayRef<Integer> indexs)
    {
    PlotString("hi","TBL");
    }
```

【说明】

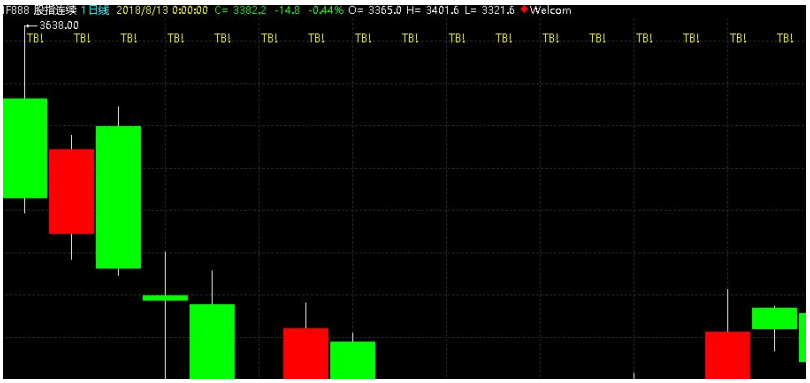
公式编译成功之后，会在状态栏显示“编译成功”，“已保存”，“已编译”信息。编译成功的公式可以被加载运行了；

若公式代码较多，编写没有全部完成，中途需要退出 TB 软件，可以单击“保存”按钮将当前已完成的部分代码先保存下来，待编写完成后再单击“编译保存公式”；

【注意】只保存而未通过编译的公式是不能够被调用的。

若公式代码存在错误，公式编译保存不成功，此时编译显示信息区域会出现错误提示信息，可根据该信息修改公式直至通过编译。

4、打开超级图表，加载“Welcome”公式，运行结果如下图所示。



【提问】公式里只写了一条语句，加载到图表上为什么显示出很多？

【回答】这和 TB 公式的运行机制有关。

TB 公式的分类

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，它执行一系列操作实现一定的功能，返回一个结果值。用户函数不能直接加载在图表上，只能在其他用户函数或公式应用中调用。

公式应用

公式应用可直接应用于图表，分为技术分析和交易策略两类。其中技术分析类公式通过语句的控制在图表上输出数据、线型，借助这些图型的显示交易者可进行一系列直观的分析；交易策略类公式通过条件语句判断交易的入场点与出场点，在图表上对买卖点做出标识（交易信号）。

TB 公式的结构

TB 公式一般分为三段：公式参数段，公式变量段，公式脚本段。

示例：在公式编辑器中打开系统公式 DualMA，如下图所示：

```
//-------------------------------------------------------------------------
// 简称: DualMA
// 名称: 双均线交易系统
// 类别: 公式应用
// 类型: 内建应用
//-------------------------------------------------------------------------
Params
    Numeric FastLength(5);
    Numeric SlowLength(20);
Vars
    Series<Numeric> AvgValue1;
    Series<Numeric> AvgValue2;
Events
    OnBar(ArrayRef<Integer> indexs)
    {
        AvgValue1 = AverageFC(Close,FastLength);
        AvgValue2 = AverageFC(Close,SlowLength);
        PlotNumeric("MA1",AvgValue1);
        PlotNumeric("MA2",AvgValue2);

        // 集合竞价和小节休息过滤
        If(!CallAuctionFilter()) Return;

        If(MarketPosition <>1 && AvgValue1[1] > AvgValue2[1])
        {
            Buy(1,Open);
        }

        If(MarketPosition <>-1 && AvgValue1[1] < AvgValue2[1])
        {
            SellShort(1,Open);
        }
        //PlotNumeric("PL",Portfolio_TotalProfit);
    }
//-------------------------------------------------------------------------
// 编译版本  GS2010.12.08
```

参数

变量

事件驱动

公式参数段

定义公式中使用的参数，在 Params 之后。如果不定义公式参数则该段不存在。
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，可以在公式代码段使用该参数的名称来引用其值。
参数的值在公式的内部不能被修改，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)。
使用参数的优点:

- 调用公式应用时，根据需要指定相应的参数参与公式的计算，而不需要修改公式，重新编译;
- 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。

公式变量段

定义公式应用中使用的变量，在 Vars 之后。如果不定义公式变量则该段不存在。
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，而无需重现计算过程。
使用变量的优点:

- 变量有助于程序的优化，有时 TB 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)，也可能是通过表达式执行计算和比较的结果。因此，不重复运算，直接使用变量可以提高程序的运行速度，节约内存空间;
- 使用变量也可以避免输入错误，提高程序的可读性。

公式脚本段

编写公式应用的代码，包含在 Begin 与 End 之间。
公式脚本段是用户自由书写代码的位置，可以赋值、调用函数、使用控制语句……将自己的思路转换为符合 TB 语言语法的语句，通过编译之后加载到图表自动运行。

2.4 公式属性

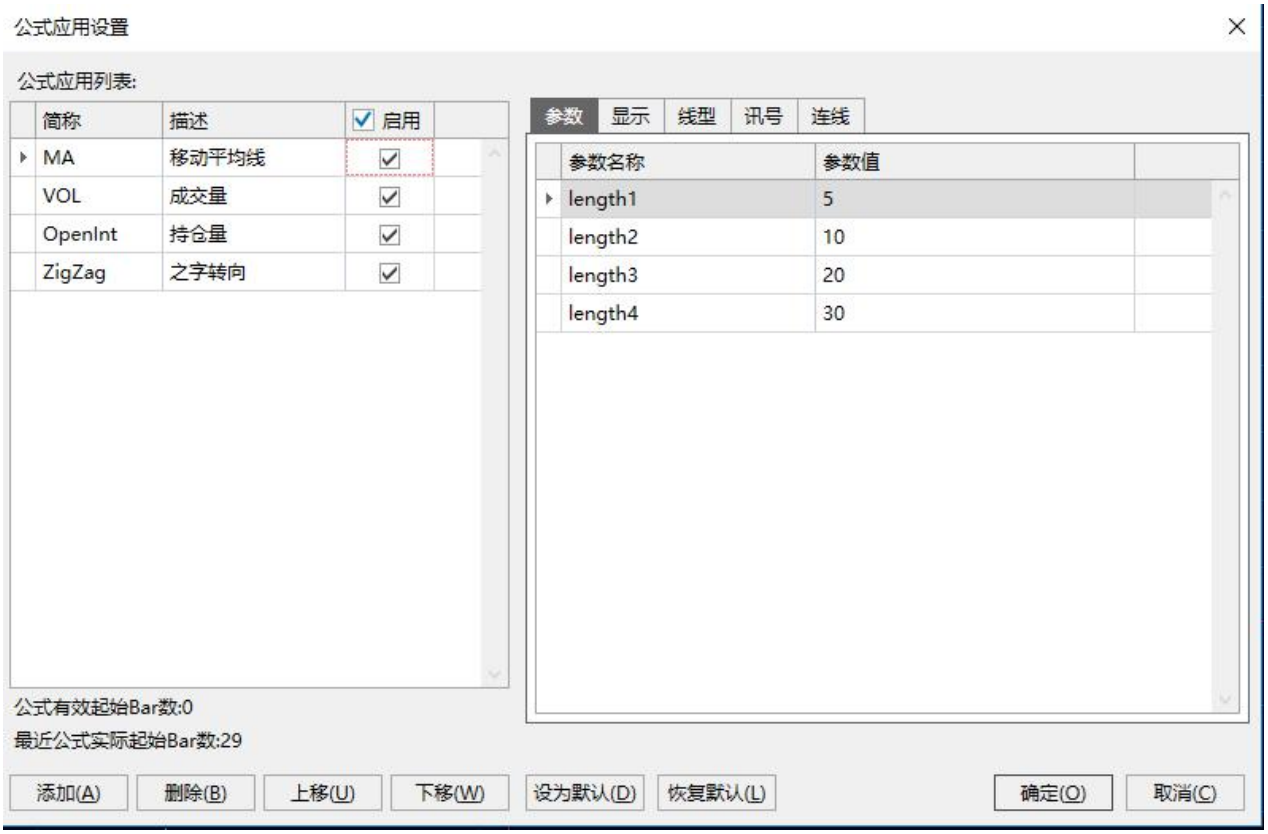
根据公式类型不同，每种公式具有不同的属性工作区。
公式应用的简要属性窗口如下图所示。

用户函数的简要属性窗口如下图所示：

显示方式：主图与副图

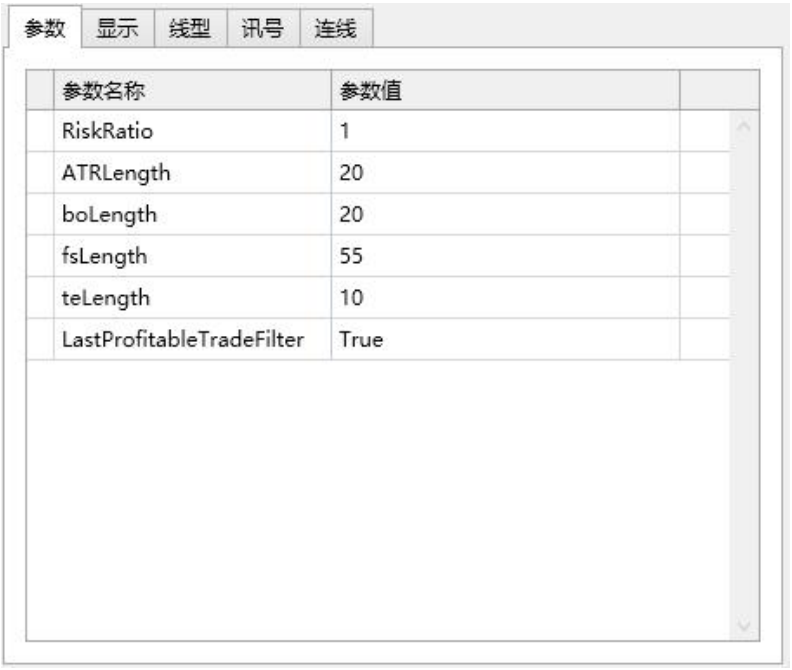
如果选择主图则公式的信息输出都在主图显示，包括注释和画线。
如果选择副图则公式的信息输出都在副图显示，包括注释和画线。
在 K 线图界面可以拖动主图上的指标画线到副图显示，但这个动作不会被记忆。切换品种之后，公式属性的显示方式是主图的指标仍旧只在主图显示。

可以在图表中打开公式应用设置，查看公式应用的详细属性设置，公式应用设置界面如下图所示：



公式参数

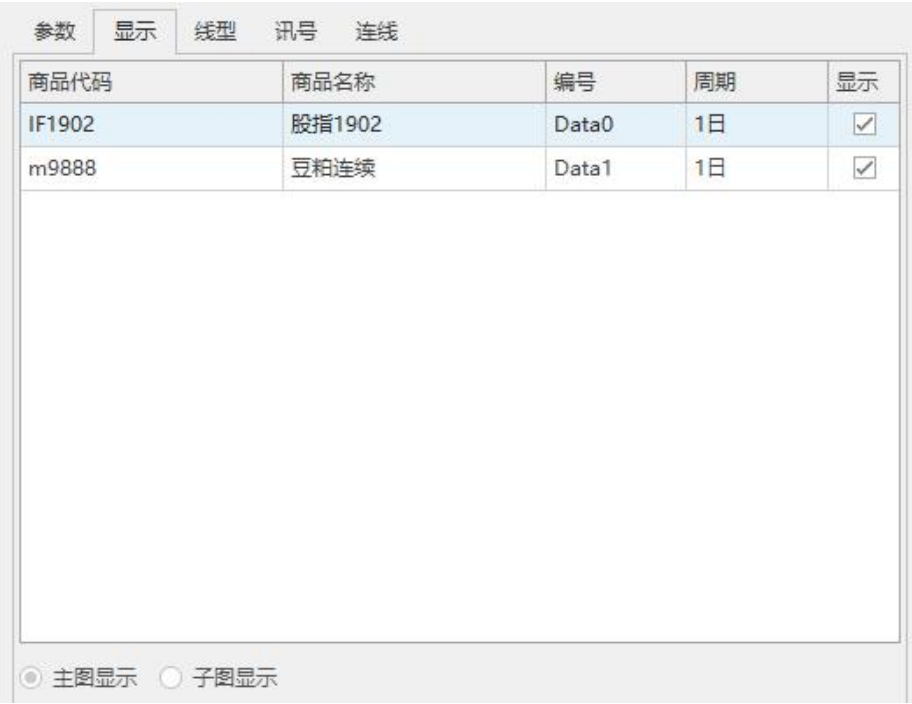
在左边公式列表选中一个公式，点击右边的线型工作区，可以设置公式的参数。
该工作区显示公式的参数信息，在公式调用的时候，您可以直接通过修改列表框的参数值进行参数变更。
还可以点击[设为默认]将该参数保存到公式内部，供其他地方调用。



公式显示

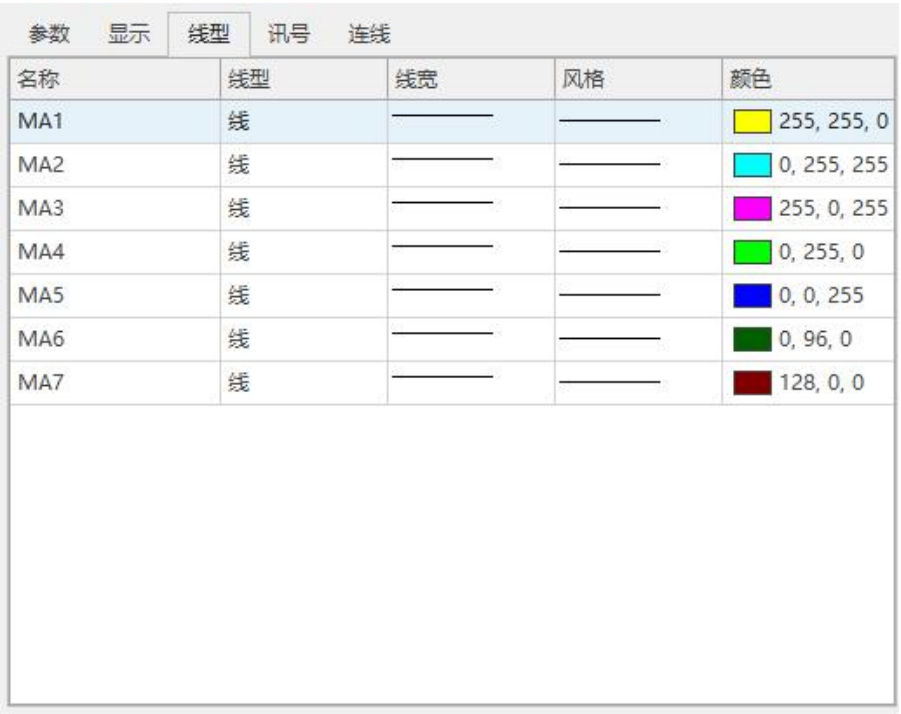
在左边公式列表选中一个公式，点击右边的显示工作区，勾选期望显示该公式的商品，勾选后商品图层将

显示该公式，否则不显示该公式。



公式线型

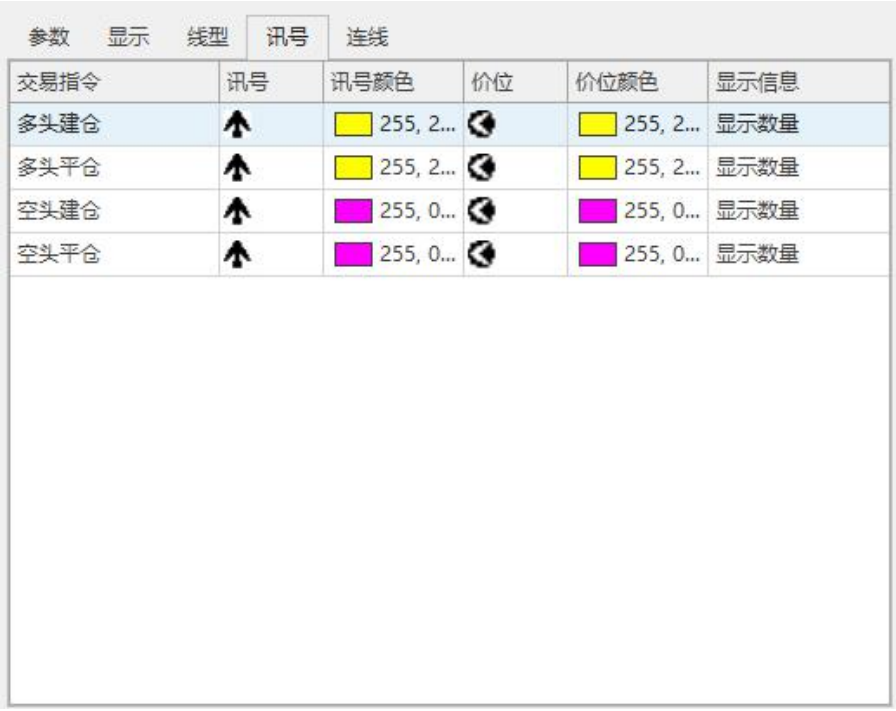
在左边公式列表选中一个公式，点击右边的线型工作区，可以设置公式每条输出线的线型、线宽、风格、颜色。如果不想显示输出线，可以在输出线的线型下拉框中选择隐藏项。



公式讯号

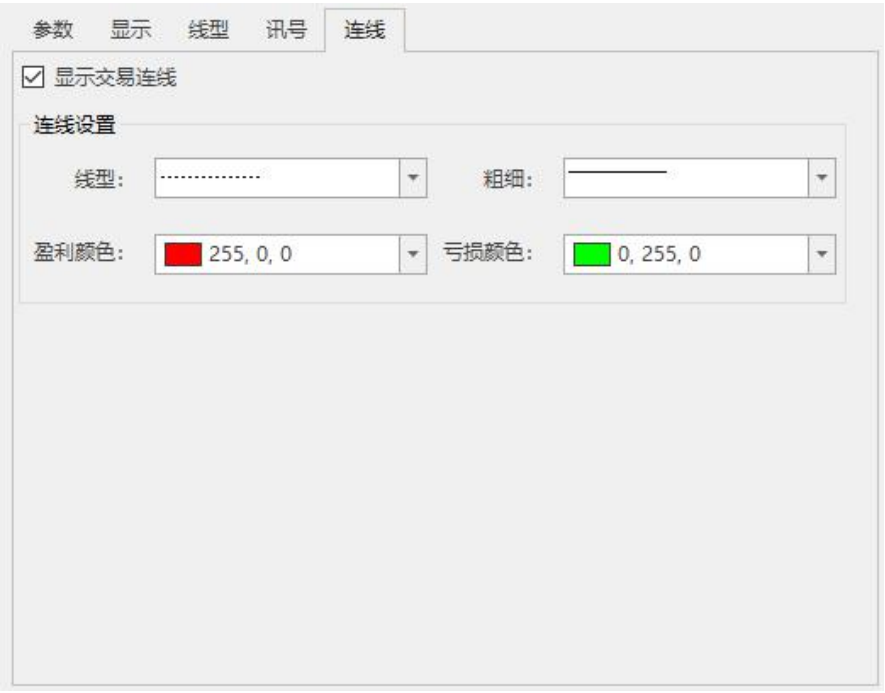
讯号是公式系统在历史数据中应用产生的买卖指令，在图表中显示为向上或向下的箭头，并在买卖价位处用侧向箭头进行标记。

在左边公式列表选中一个公式，点击右边的讯号工作区，可以设置四种类型公式应用讯号箭头的显示风格、颜色，价位箭头的显示风格、颜色，以及是否显示交易数量。



公式连线

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盈利线和亏损线，还可以对线条的线型、粗细和颜色进行设置。
如果开仓信号与平仓信号是不同公式产生的，则连接使用平仓信号的设置。

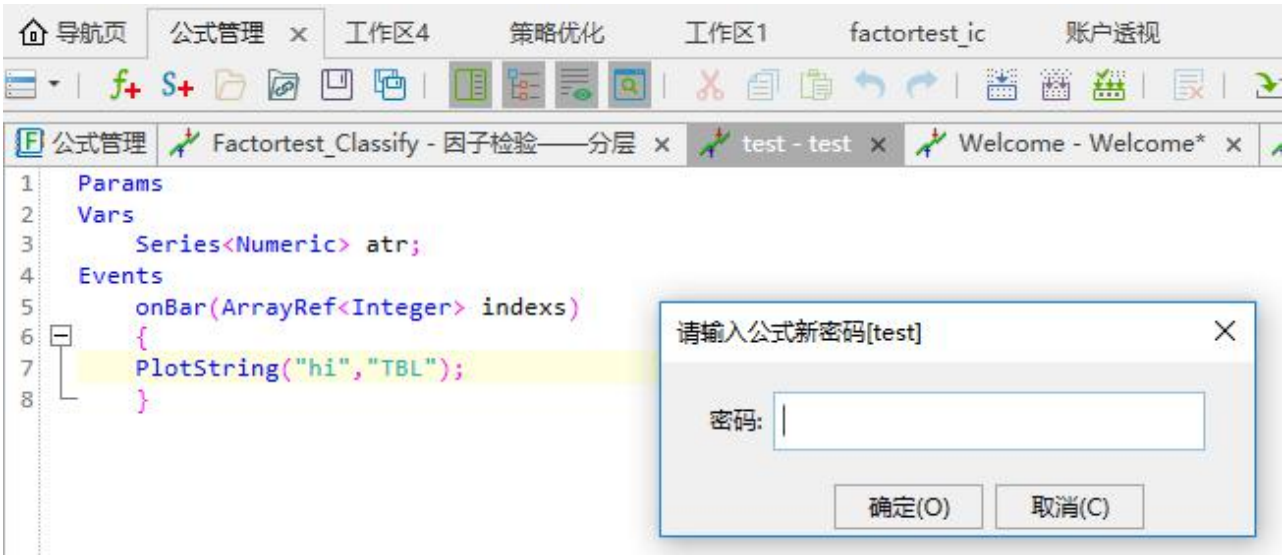


可通过点击[设为默认值]按钮将当前公式应用右边的参数、显示、线型、讯号、连线设置保存为系统默认值，之后可以在其他地方使用该默认设置。也可通过点击[恢复默认值]按钮将当前对话框的各项设置修改

为系统默认值。

公式加密

公式编辑器提供对用户自编的公式代码进行加密，在公式属性中，加密项选择“是”，同时设置密码，确定之后，便完成了对该公式代码的加密操作。



再次打开已加密公式时，需要输入正确的密码方可打开公式的代码。对于已经通过编译的公式，是否知道密码不影响公式的加载使用。导出已加密公式，其特性与在本地机一样，需要密码方可在公式编辑器中打开查看源代码，不影响在超级图表上的调用。

【说明】：公式加密后可导出给其他用户使用，如果该用户不知道密码，导入时必须选择“导入编译”，否则导入的公式需要进入编辑界面进行编译，本机才能执行。如前所述，打开已加密公式需要验证密码，由于不能打开及编译将会影响所导入公式的使用。

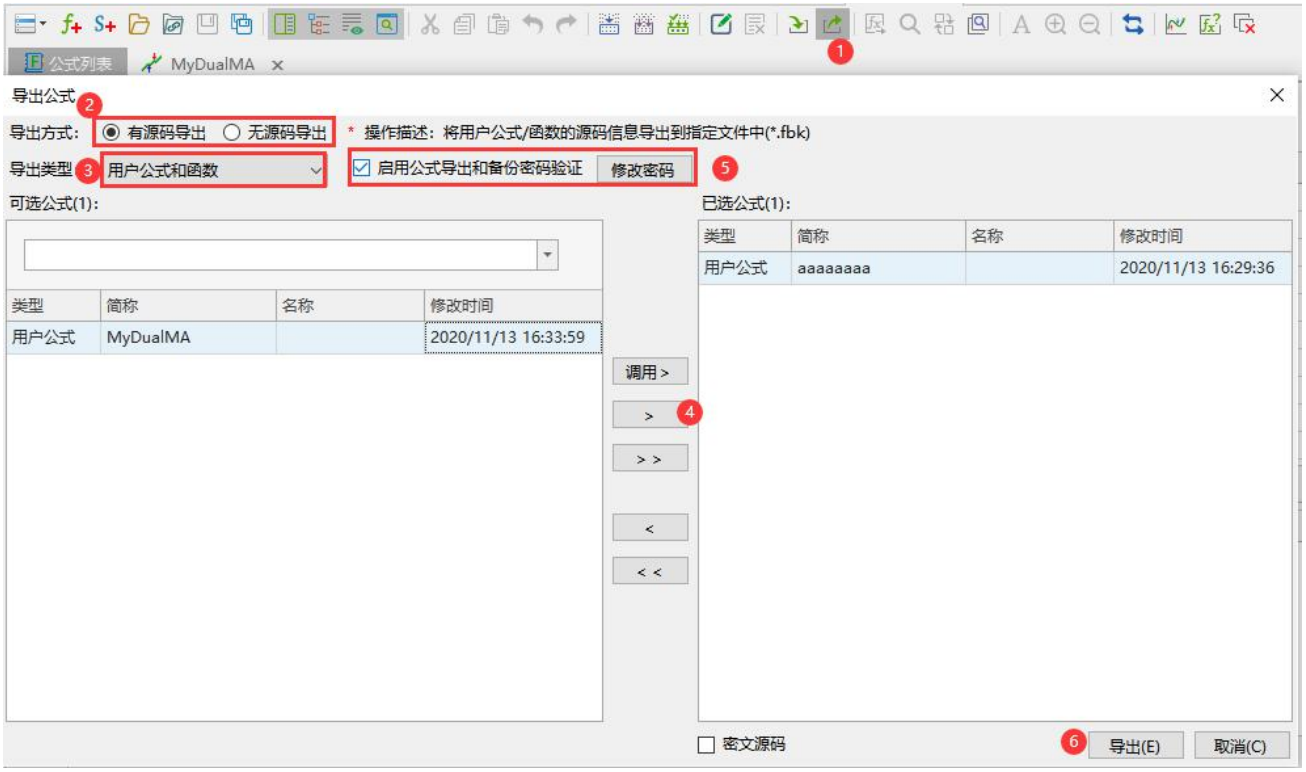
2.5 公式导入导出

公式导入导出功能能够将 TB 公式打包成文件，或者将已打包的公式文件导入；通过该功能，可方便用户之间公式的交互。

公式导出

下面的操作步骤将“Welcome”公式分别以有源码方式和无源码方式备份到文件 w1、w2 中，文件存放在 D 盘根目录下。

1.单击公式管理工具栏的公式导出按钮“”，打开“导出公式”窗口，如下图所示：



公式导出可以启用密码验证，输入密码加密文件；也可能不输入，无密码导出。

2．选择有源码导入模式。

有源码模式：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)。用户导入有源码文件后，可以在编辑器中看到公式应用的源代码，并且可以对导入的代码进行修改、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(.nef)。

3.显示可选公式列表，选择显示用户公式和函数。

也可以选择只显示用户公式，或只显示用户函数。

4.从可选公式列表中选择要导出的公式“Welcome”添加到已选公式列表中；

[>]按钮：将可选公式列表中选中的公式添加到已选公式列表中，准备导出；

[>>]按钮：将可选公式列表中的所有公式添加到已选公式列表中，准备导出；

调用函数>:将已选公式列表的调用的用户函数添加到已选公式列表中，准备导出；

[<]按钮：将已选公式列表中选中的公式放回到可选公式列表中，取消导出该公式；

[<<]按钮：将已选公式列表中的所有公式放回到已选公式列表中，取消导出这些公式；

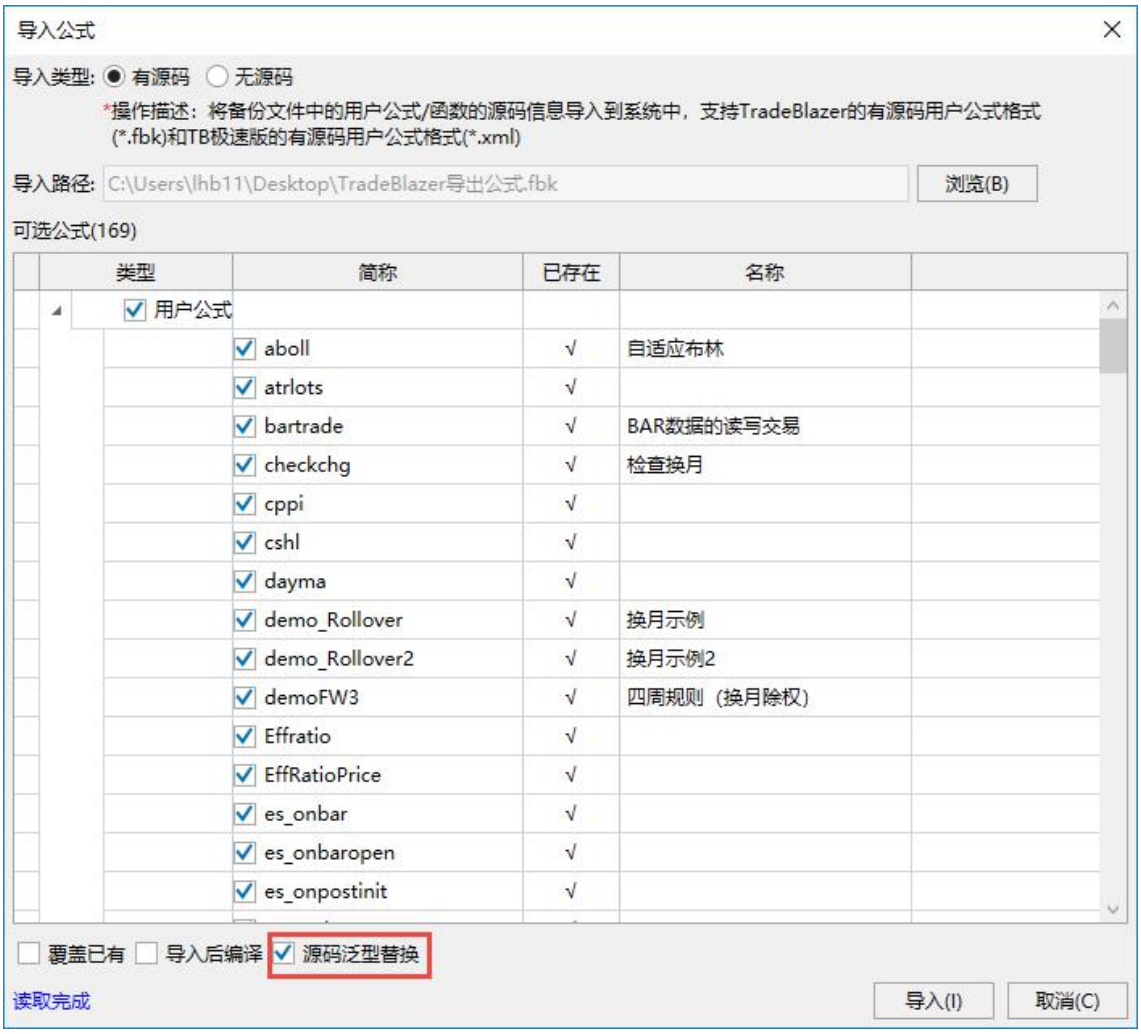
可以通过双击列表项进行操作。

5．选择公式文件的保存路径。

6．单击“导出”按钮，系统提示导出成功，公式导出完成。

公式导入

1．单击公式管理工具栏的公式导入按钮“”，打开“导出公入”窗口，如下图所示：



2．选择有源码导入模式。

3. 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。

4. 选择要导入的用户公式和用函数。

5. 勾选“覆盖已有”将覆盖本地同名的公式源码。

6. 点击导入按钮。

【注意】

1）注意数据类型升级了泛型，所以源码泛型转换需要勾选。

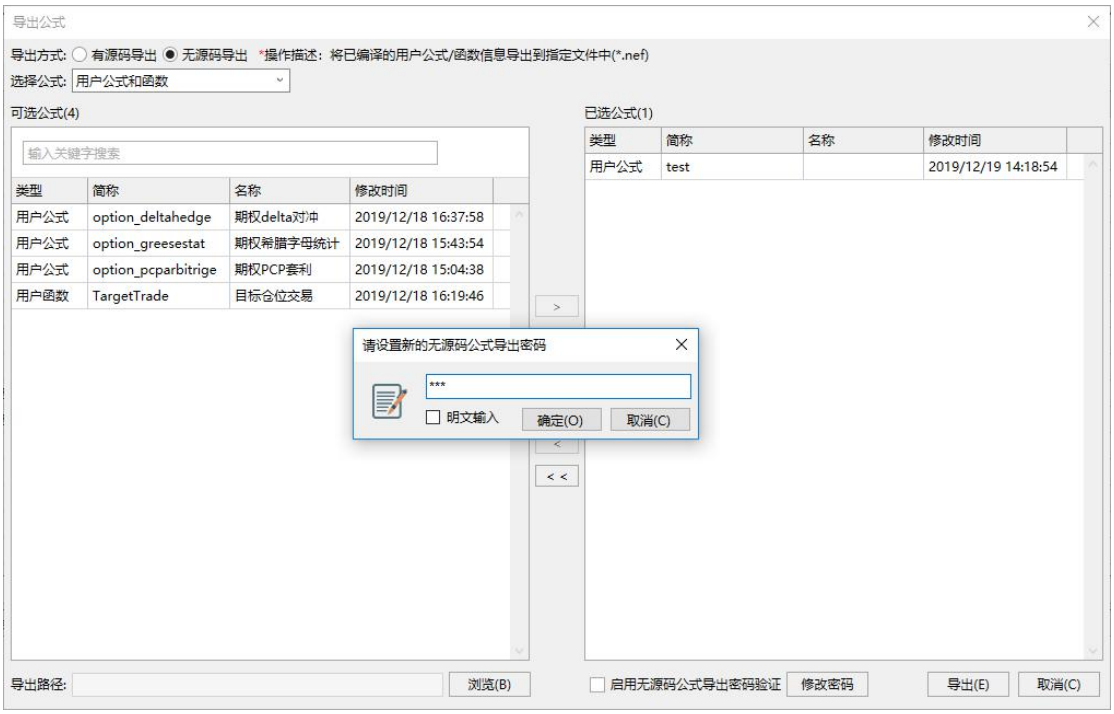
2）使用有源码导入导出时请将该公式关联的用户函数一起选择导入，否则会导致所导入公式编译不成功。使用无源码模式导入导出公式应用时，无需依赖用户函数，直接导入导出公式应用即可使用。

3）导入公式后，可以选择全部编译。系统会自动先编译函数， 再编译公式。

无源码公式导出的加密功能

无源码公式导出的密码验证是指用户导出无源码公式时需要输入密码才能导出。

密码设置在右下角，启用无源码公式导出密码验证。用户也可以点击修改密码按钮修改密码。



三、策略案例

3.1 公式进阶案例讲解

案例一：止盈止损

本例中模板以止赢 30 跳，止损 20 跳为例，实际使用时可以转换为开仓价格的百分比或其它任何设置的变量进行止盈止损。

分析

要编写止盈止损的代码，关键是找准基准价格（本例以建仓价格 MyEntryPrice 为基准）和止盈止损价格

1、止盈 30 跳

多仓情况，止盈条件可写为当最高价比建仓价格高 30 跳，表达式为：
最高价>= 建仓价格 + 止盈值（30 跳表示的点位）
High>= MyEntryPrice + 30* MinMove*PriceScale
空仓情况，止盈条件可写为当最低价比建仓价格低 30 跳，表达式为：
最低价<= 建仓价格 - 止盈值（30 跳表示的点位）
Low<= MyEntryPrice - 30* MinMove*PriceScale

2、止损 20 跳

多仓情况，止损条件可写为当最低价比建仓价格低 20 跳，表达式为：
最低价>= 建仓价格 - 止损值（20 跳表示的点位）
Low <= MyEntryPrice - 20* MinMove*PriceScale
空仓情况，止损条件可写为当最高价比建仓价格高 20 跳，表达式为：
最高价>= 建仓价格 + 止损值（20 跳表示的点位）
high >= MyEntryPrice + 20* MinMove*PriceScale
【说明】实际编写时，为了保证代码的可重复性，表达式中不直接使用数字 30,20，而是定义数值型变量 TakeProfitSet 存放止盈设置，StopLossSet 存放止损设置。这样当需要更改止盈止损的点位时，只需对变量重新赋值即可，代码中止盈止损的表达式则无需修改。
如果止盈条件修改为价格上涨 30%止盈，表达式可以写为：（以多仓为例）
High>= MyEntryPrice + 0.3* MyEntryPrice

3、止盈止损的价格，考虑开盘价格跳空的情况

止盈止损的价格，通常情况下直接按照止盈止损的要求即可，如果遇到开盘跳空的时候，需要进行相应的处理。
以多仓止盈，跳空为例，如果开盘价>止盈价，则使用开盘价进行止盈。
If(Open > MyExitPrice) MyExitPrice = Open
【说明】变量 MyExitPrice 也用作保存止盈价格

4、公式主体结构设计

由于止盈止损分为多仓和空仓两类不同的情况，所以采用嵌套的 if……else 语句

代码

```
Vars
  Numeric MinPoint; // 一个最小变动单位，也就是一跳
  Numeric MyEntryPrice; // 开仓价格，例中为开仓均价，可设置为某次入场价
  Numeric TakeProfitSet(30); // 止赢设置
  Numeric StopLossSet(20); // 止损设置
  Numeric MyExitPrice; // 平仓价格
Events
OnBar(ArrayRef<Integer> indexs)
{
  //...
  MinPoint = MinMove*PriceScale;
  MyEntryPrice = AvgEntryPrice;
  If(MarketPosition == 1 And BarsSinceEntry >= 1) // 有多仓的情况
  {
    If(High >= MyEntryPrice + TakeProfitSet*MinPoint) // 止赢条件表达式
```

```
    {
      MyExitPrice = MyEntryPrice + TakeProfitSet*MinPoint;
      // 如果该 Bar 开盘价即跳空触发，用开盘价代替
      If(Open > MyExitPrice) MyExitPrice = Open;
      Sell(0,MyExitPrice);
    }Else If(Low <= MyEntryPrice - StopLossSet*MinPoint) // 止损条件表达式
    {
      MyExitPrice = MyEntryPrice - StopLossSet*MinPoint;
      // 如果该 Bar 开盘价即跳空触发，则用开盘价代替
      If(Open < MyExitPrice) MyExitPrice = Open;
      Sell(0,MyExitPrice);
    }
  }
  Else If(MarketPosition == -1 And BarsSinceEntry >= 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;
      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;
      BuyToCover(0,MyExitPrice);
    }
  }
  //...
}
```

注意事项

1. 因无法确定开仓 Bar 最高价、最低价和开仓价的先后顺序，因此以上写法忽略开仓 Bar 的处理。
2. 如果某个 Bar 最高价、最低价相差很大，可能会出现止盈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，或者扩大止盈、止损幅度。

案例二：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。原始止损条件为回撤 50 跳。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。

分析（以多仓为例）

1、第一个建仓位置到当前位置的 Bar 计数用以下函数表示。

BarsSinceEntry

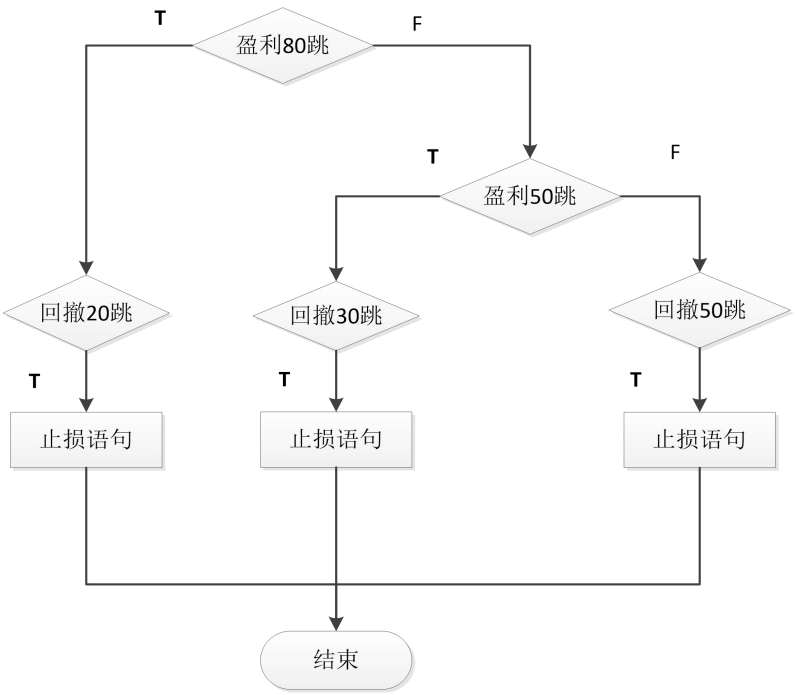
说明	获得当前持仓的第一个建仓位置到当前位置的 Bar 计数。
语法	Integer BarsSinceEntry()
参数	无
备注	获得当前持仓的第一个建仓位置到当前位置的 Bar 计数，返回值为整型。 只有当 MarketPosition != 0 时，即有持仓的状况下，该函数才有意义，否则返回 0。 注意：在开仓 Bar 上为 0。

记录建仓以来最高最低价格。跟踪止损条件是采用这个最高最低价格和止损点之间进行比较判断。以多仓为例，需要记录开仓以来最高价格，使用序列变量 HighestAfterEntry 保存此值，在开仓 bar 上，比较开仓价和最新价格，记录较大的值；其他 bar 上，比较该 bar 的 high 和 HighestAfterEntry 的值，记录较大值。空仓反之，记录开仓以来的最低价格。

```
If(BarsSinceEntry == 0)
{
  HighestAfterEntry = Close;
  LowestAfterEntry = Close;
  If(MarketPosition <> 0)
  {
    HighestAfterEntry = Max(HighestAfterEntry,AvgEntryPrice);
    LowestAfterEntry = Min(LowestAfterEntry,AvgEntryPrice);
  }
}Else
{
  HighestAfterEntry = Max(HighestAfterEntry,High);
  LowestAfterEntry = Min(LowestAfterEntry,Low);
}
```

2、公式结构

跟踪止损和直接止损的区别是随着盈利的提高，突破某个值时，相应止损回撤值减少，即止损点不是固定的。因此需要使用多分支的嵌套语句分别进行判断。



3、止损价格需考虑开盘跳空的情况

```
If(Open < MyExitPrice) MyExitPrice = Open;
```

【说明】MyExitPrice 为止损价格

代码：

```
Vars
   Numeric MinPoint;       // 一个最小变动单位，也就是一跳
   Numeric MyEntryPrice;    // 开仓价格，本例为开仓均价，可设置为某次入场价
   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
   Series<Numeric> HighestAfterEntry;     // 开仓后出现的最高价
   Series<Numeric> LowestAfterEntry;      // 开仓后出现的最低价
events
OnBar(ArrayRef<Integer> indexs)
{

   //...
   If(BarsSinceEntry == 0)          // 条件满足：开仓 Bar
   {
      HighestAfterEntry = Close;
```

```
    LowestAfterEntry = Close;      // 赋初值为当前最新价格
    If(MarketPosition <> 0)        // 有持仓时执行以下代码
    {
      // 开仓 Bar，将开仓价和当时的收盘价的较大值保留到 HighestAfterEntry
      HighestAfterEntry = Max(HighestAfterEntry,AvgEntryPrice);
      // 开仓 Bar，将开仓价和当时的收盘价的较小值保留到 LowestAfterEntry
      LowestAfterEntry = Min(LowestAfterEntry,AvgEntryPrice);
    }
  }Else    // 非开仓 Bar 时进行以下运算
  {
    // 记录下当前 Bar 的最高点，用于下一个 Bar 的跟踪止损判断
    HighestAfterEntry = Max(HighestAfterEntry,High);
    // 记录下当前 Bar 的最低点，用于下一个 Bar 的跟踪止损判断
    LowestAfterEntry = Min(LowestAfterEntry,Low);
  }

  Commentary("HighestAfterEntry = "+Text(HighestAfterEntry));
  Commentary("LowestAfterEntry = "+Text(LowestAfterEntry));

  MinPoint = MinMove*PriceScale;
  MyEntryPrice = AvgEntryPrice;

  If(MarketPosition == 1 And BarsSinceEntry >= 1) // 有多仓的情况
  {
    // 第二级跟踪止损的条件表达式
    If(HighestAfterEntry[1] >= MyEntryPrice + TrailingStart2*MinPoint)
    {
      If(Low <= HighestAfterEntry[1] - TrailingStop2*MinPoint)
      {
        MyExitPrice = HighestAfterEntry[1] - TrailingStop2*MinPoint;// 如果该 Bar 开盘价即跳空触发，则用开盘价代替
        If(Open < MyExitPrice) MyExitPrice = Open;
        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;
        // 如果该 Bar 开盘价即跳空触发，则用开盘价代替
        If(Open < MyExitPrice) MyExitPrice = Open;
        Sell(0,MyExitPrice);
      }
```

```
    }Else If(Low <= MyEntryPrice - StopLossSet*MinPoint) //可在此写初始止损处理
    {
      MyExitPrice = MyEntryPrice - StopLossSet*MinPoint;
      // 如果该 Bar 开盘价即跳空触发，则用开盘价代替
      If(Open < MyExitPrice) MyExitPrice = Open;
      Sell(0,MyExitPrice);
    }
  }Else If(MarketPosition == -1 And BarsSinceEntry >= 1)  // 有空仓的情况
  {
    // 第二级跟踪止损的条件表达式
    If(LowestAfterEntry[1] <= MyEntryPrice - TrailingStart2*MinPoint)
    {
      If(High >= LowestAfterEntry[1] + TrailingStop2*MinPoint)
      {
        MyExitPrice = LowestAfterEntry[1] + TrailingStop2*MinPoint;
        If(Open > MyExitPrice) MyExitPrice = Open;
        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;
        BuyToCover(0,MyExitPrice);
      }
    }Else If(High >= MyEntryPrice + StopLossSet*MinPoint) //可在此写初始止损处理
     {
      MyExitPrice = MyEntryPrice + StopLossSet*MinPoint;
      If(Open > MyExitPrice) MyExitPrice = Open;
      BuyToCover(0,MyExitPrice);
    }
  }
  //...
}
```

注意事项：

1、因无法确认开仓 Bar 最高价、最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 最高价、最低价相差很大，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 的最高价、最低价计算最大盈利位置。

案例三：加仓减仓

本例仅以做多为例，做空类似。模板以首次开仓 2 手后每盈利 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它任何设置的变量进行处理。

分析：

1、加仓三次如何表示，需要用到 CurrentEntries 函数

CurrentEntries

说明	获得当前持仓的建仓次数。
语法	Integer CurrentEntries()
参数	无
备注	获得当前持仓的建仓次数，返回值为整型。 只有当 MarketPosition != 0 时，即有持仓的状况下，该函数才有意义，否则返回 0。

通过判断条件 CurrentEntries<4 得知是否加仓三次

2、条件设置，以盈利为例，每赢利 30 跳加仓一次

最高价格>= 最后一次建仓价格 + 30 跳盈利
High >= LastPrice + AddSet*MinPoint
【说明】为了方便调整盈利设置，这里不直接使用常数 30，而是用变量 AddSet 来保存。

3、公式结构

因为有仓位时需要一直判断是否满足加仓减仓的条件，所以要使用循环语句。总的来说是分支结构，其中嵌套循环语句。

代码：

```
Vars
   Numeric MinPoint;        // 一个最小变动单位，也就是一跳
   Series<Numeric> FirstPrice;   // 第一次开仓价格
   Series<Numeric> 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
Events
OnBar(ArrayRef<Integer> indexs)
```

```
{

  //FirstEntryCon = ...        // 设置首次开仓条件
  MinPoint = MinMove*PriceScale;
  If(MarketPosition == 0)     // 空仓时
  {
    If(FirstEntryCon)
    {
      FirstPrice = Open;
      LastPrice = FirstPrice;
      Buy(2,FirstPrice);     // 条件满足，首次开仓 2 手
    }
  }Else If(MarketPosition == 1 And BarsSinceEntry >= 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);
    }
  }
  //...
}
```

注意事项

1. 因无法确认开仓 Bar 最高价、最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 最高价、最低价相差很大，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。

案例四：多品种交易

多品种交易，指的是图表中叠加多个商品，公式中对多个商品同时进行交易。通常情况下，如果公式中没有指明交易的商品，而且也没有在商品交易属性中设置委托偏移，仅针对第一个商品 Data0 进行交易。
模板以常用的双均线系统为例，对叠加的各个商品分别进行交易。

代码：

```
Params
  Numeric FastLength(5);
  Numeric SlowLength(20);
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
Events
OnBar(ArrayRef<Integer> indexs)
{

  Range[0:1]
  {
    AvgValue1 = AverageFC(Close,FastLength);
    AvgValue2 = AverageFC(Close,SlowLength);
    PlotNumeric("MA1",AvgValue1);
    PlotNumeric("MA2",AvgValue2);
  }

  Range[0:1]
  {
    If(MarketPosition <>1 && AvgValue1[1] > AvgValue2[1])
    {
      Buy(1,Open);
    }

    If(MarketPosition <>-1 && AvgValue1[1] < AvgValue2[1])
    {
      SellShort(1,Open);
    }
  }
}
```

案例五：跨周期

案例实现在 5 分钟 K 线上调用日线指标。

分析

1. 本案例的核心在于如何得到日线指标，TB 新公式系统支持同一策略单元内跨周期数据源调用。只需将 5 分钟周期数据源和日周期数据源加入同一策略单元，便可非常方便地在 5 分钟 K 线上调用日线指标。
2. 考虑引用日线指标时引用未来数据问题，对于当日来讲，日线指标还没有计算出来就要引用，这种方式

写技术分析指标是可以的，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，日线指标最多引用到前一日。

代码

新建一个公式应用 My5MinMA，得到日线指标并显示，详细代码如下：

```
Params
   Numeric Length(10);
Vars
   Series<Numeric> MA;
events
OnBar(ArrayRef<Integer> indexs)
{

   Data[1].MA = Data[1].AverageFC(Data[1].Close,Length);  //计算日线指标
   Data[0].PlotNumeric("MA", Data[1].MA[1]);  //在 5 分钟周期中引用日线指标
   Data[1].PlotNumeric("MAday", Data[1].MA[1]);  //在日线上同时显示指标
}
```

在超级图表中依次插入 5 分钟商品和日线商品，将编译成功的公式 My5MinMA 插入超级图表，为了对比方便，将支持多数据源显示的 10 周期均线公式 MA_Mlayer_10 插入超级图表中，显示如下：

注意事项

为了更准确合理的使用跨周期数据，建议引用数据源上一个 bar 的数据或变量，以避免使用未来数据或变化的数据。

案例六：收盘平仓

分析

收盘平仓是指每个交易日收盘前平掉持有的仓位。由于真正收盘后，交易指令是无法成交的，而太早发出平仓指令，实际平仓价又会和真实的收盘价有一定的差异。因此，实际交易时只能通过在真正收盘之前，提前一定的时间，发出平仓指令，近似地达到收盘平仓的目标。因此，平仓信号分为两种情况，一种是（非当天）的收盘平仓信号，另一种是当天的收盘平仓信号。

1、历史的收盘平仓信号：以该交易日的收盘价平仓（近似效果）

历史收盘平仓信号，一定产生于过去某个交易日的最后一根 Bar。所以，编程的主要任务是找到该交易日的最后一根 bar，然后在此 bar 上执行平仓代码。这里又分为两种情况：
1）非最后交易日的收盘平仓
找到两个 bar 日期都为有效值，并且前一根 Bar 日期与后一根 bar 的日期不同，由此可得出前一根 Bar 是那天的最后一个 Bar，应该进行收盘平仓。
`Date[-1] != InvalidInteger && Date != Date[-1]`
2）最后交易日（但并不是今日）的收盘平仓
既然是最后交易日，那收盘 BAR 一定是最后一根。因此，可通过后一个 bar 是否为无效值来判断。但是最后 Bar 的日期是小于系统当前日期的值，说明已经不是当天了，也要进行收盘平仓。（但要注意和下面一种情况的区别是，不是交易当天，所以不需要对时间进行判断）
`Date[-1] == InvalidInteger && Date < CurrentDate`

2、当天的收盘平仓（选择 14：59 分平仓）

当天的收盘平仓和历史的收盘平仓信号的不同之处在于，当天的收盘平仓是在接近收盘时才产生，所以需要增加一个时间的判断。以商品期货 15:00 收盘，5 分钟周期为例，找到当日最后一个 bar，即 time 为 14:55 的那个 bar，再判断系统当前时间是否超过 14:59 分，满足条件之后做收盘平仓操作。
`Date == CurrentDate && Time == 0.1455 && CurrentTime >= 0.1459`

代码

```
Events
OnBar(ArrayRef<Integer> indexs)
{

  //...
  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, Close);
    BuyToCover(0, Close);
  }
  //...
}
```

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。

案例七：A 函数下单、撤单以及全局变量操作

案例实现的功能是每天收盘前 N 分钟时自动撤掉超级图表中多个商品的挂单，并全部平仓。代码中通过 A_SendOrder 进行下单，A_DeleteOrder 进行撤单。

分析

1、A 函数仅对实时行情有效，所以，需要使用之前用“BarStatus == 2”进行限定；
2、根据 TB 程序运行机制，实时行情时，当前 bar 每个 tick 都会触发程序的执行，为了避免 A 函数每个 tick 重复撤单平仓，公式中使用全局变量 HasSendOrder 来保存上一次程序运行之后撤单平仓标志。
3、公式中的平仓是根据 A_BuyPosition()的返回值来确定的，但是在收盘平仓之前有撤挂单的操作，或者因为成交回报不及时而不能得到准确的 A_BuyPosition()值，因此，设置撤单之后延时 5 个 tick 再平仓。（注意：这里假定撤单后 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够成功，因此实际策略中请根据情况调整）
本例中用公式中使用全局变量 DeleteOrderTickCounter 来记录延时的 tick 数。当撤单执行后，DeleteOrderTickCounter 赋值为 1，开始计数，每次累加 1，判断 DeleteOrderTickCounter 小于 5，没有达到 5 个 tick 的时候直接 return，直到 5 个 tick 之后再继续后续操作，平仓。
值得注意的是 DeleteOrderTickCounter 的初值如何赋值，代码中给出的是 999，实际上只要是比 5 大的数都可以，这是为了保证到了收盘平仓时间，如果没有挂单可以直接进行平仓，即：使得条件 DeleteOrderTickCounter< 5 不成立，程序可执行平仓操作。

代码

```
Params
  Numeric offSet(1);           // 委托价格偏移
  Numeric BeforeMins(10);      // 收盘前几分钟开始操作
Vars
  Numeric tempPos;             // 仓位
  Global Numeric dataIndex;    // 商品索引
  Global Integer DeleteOrderTickCounter;
```

```
  Global Integer HasSendOrder;
Events
OnBar(ArrayRef<Integer> indexs)
{

  If(BarStatus == 0)  // 第一个 Bar，初始化 Tick 计数器、撤单标志，存于全局变量中
  {
    DeleteOrderTickCounter = 9999;
    HasSendOrder = 0;
    SetGlobalVar(0,DeleteOrderTickCounter);
    SetGlobalVar(1,HasSendOrder);
  }Else     // 其他 Bar，从全局变量中读取撤单 Tick 计数器、撤单标志的值
  {
    DeleteOrderTickCounter = GetGlobalVar(0);
    HasSendOrder = GetGlobalVar(1);
  }


  // 收盘前 N 分钟，且撤单标志为 0，即还未撤单时
  If(CurrentTime > (0.1459 - 0.0001*(BeforeMins - 1)) && BarStatus == 2 && gValue[0] == 0)
  {
    For dataIndex=0 To DataSourceSize -1
    {
      If(Data[dataIndex].Close != InvalidNumeric && Data[dataIndex].A_GetOpenOrderCount() > 0)
      {
        Data[dataIndex].A_DeleteOrder();
        // Tick 开始计数，为了延迟 5 个 Tick 后做平仓用的
       DeleteOrderTickCounter = 1;
      }

      DeleteOrderTickCounter = DeleteOrderTickCounter + 1;
      SetGlobalVar(0,DeleteOrderTickCounter);
      If(DeleteOrderTickCounter < 5) Return; // 撤单后需要延迟几个 Tick 才平仓

      Data[dataIndex].tempPos = Data[dataIndex].A_BuyPosition();
      If(Data[dataIndex].tempPos > 0) // 平多单
      {
                  Data[dataIndex].A_SendOrder(Enum_Sell,Enum_Exit,  Data[dataIndex].tempPos,
Data[dataIndex].Q_BidPrice - offSet* Data[dataIndex].MinMove* Data[dataIndex].PriceScale);
      }
      Data[dataIndex].tempPos = Data[dataIndex].A_SellPosition();
      If(Data[dataIndex].tempPos > 0) // 平空单
      {
                   Data[dataIndex].A_SendOrder(Enum_Buy,Enum_Exit,  Data[dataIndex].tempPos,
```

```
Data[dataIndex].Q_AskPrice + offSet* Data[dataIndex].MinMove* Data[dataIndex].PriceScale);
        }

        HasSendOrder = 1;
        SetGlobalVar(1,HasSendOrder);

    }
  }
}
```

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够成功。

案例八：平仓延迟反手

本案例实现平仓之后，再反手开仓。

分析

使用 buy、sellshort 这类反手交易指令建仓时，如果持有反向仓位，将会先平再开。实际交易过程中，平仓和开仓两个指令会同时发出，交易所不一定先撮合成交哪个指令，所以一般使用这种反手交易指令时，要求客户的账户上有交易数量 2 倍的资金。如果客户资金有限，希望先执行平仓操作，资金返回账户之后再反手开仓，那么编写公式时需要注意将平仓和开仓操作分开，保证平仓成交之后再进行开仓。本例实现的思路是先平仓，平仓之后延时 N 个 tick 再开仓。
实现方法：（*以平空仓反手开多为例*）
1、定义参数 DelayTicks，保存延时的 tick 数；
2、定义变量 TickCounter 记录 tick 数，初始值为 0，且最新 Bar 第一次生成时，重新开始计数；
3、持空仓时如果需要平仓开多，不直接采用 buy，而是先执行 BuyToCover 平仓，同时 TickCounter 开始 tick 计数；
4、比较 TickCounter 与 DelayTicks，达到了延时时间，再执行 buy 开仓。

代码(只应用在单数据源)

```
Params
  Numeric FastLength(5);
  Numeric SlowLength(20);
  Numeric DelayTicks(5);
Vars
  Series<Numeric> AvgValue1;
  Series<Numeric> AvgValue2;
```

```
  Numeric LastBarTime;
  Numeric TickCounter;
  Numeric dataIndex;
Events
OnBar(ArrayRef<Integer> indexs)
{

  AvgValue1 = AverageFC(Close,FastLength);
  AvgValue2 = AverageFC(Close,SlowLength);
  LastBarTime = GetGlobalVar(0);
  TickCounter = GetGlobalVar(1);

  // 最新 Bar 第一次生成时，Tick 重新开始计数
  If(BarStatus == 2 && gValue[0]!= Time)
  {
    LastBarTime = Time;
    TickCounter = 0;
  }
  If(MarketPosition <> 1 && AvgValue1[1] > AvgValue2[1])
  {
    If(MarketPosition == 0 || BarStatus != 2)
    // 无持仓，直接买多仓
    // 持空仓且 Bar 不是实时行情，平空仓，买多仓
    {
      Buy(1,Open);
    }Else  // 持空仓，Bar 实时行情，平空仓，通过 TickCounter 计数，延迟反手
    {
      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   // 持多仓且 Bar 为实时行情，平多，延迟反手
    {
      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);
}
```

注意事项

1、TB 程序每个 tick 触发，所以可以实现延时几个 tick 在反手开仓，但是用户要考虑 gValue[1]重置为 0 的时机，本例采用的是新 bar 生成的时候重置，这样设置对于一些长线周期没问题，但是对于短周期（tick 级，秒级），会出现延时 tick 还未达到的，新 bar 生成时将 gValue[1]的情况。
2、平仓延迟反手除了延时的办法，还可以直接判断平仓是否成交的方式，用户可根据实际情况选择。

3.2 事件驱动基本使用案例

案例一：事件驱动的基本框架

学习焦点：

通过这个案例主要展示一下事件驱动的基本框架，也就是各个代码模块的顺序。
如果对比 begin/end 的语法，我们可以看到事件驱动的代码模块是分为四大部分：
Params 作为标识的参数段；
Vars 作为标识的变量段；
Defs 作为标识的自定义函数段；
Events 作为标识的事件函数段。 8 大类的事件函数都可以调用。

代码如下：

```
Params
//此处添加参数
  Numeric timenum(5);       //定时器计数上限
Vars
//此处添加变量
  Global Integer timerId;     //定时器的 ID
  Global Integer i;       //定时器的计数
  Global Integer layer1;     //数据源的图层 1
  Global Integer layer2;     //数据源的图层 2
Defs
//此处添加公式函数
Numeric calcAvg(Numeric a,Numeric b)
{
  return (a+b)/2;
}

Events
//此处实现事件函数

//初始化事件函数，策略运行期间，首先运行且只有一次
OnInit()
{
  timerId=createTimer(1000);
  layer1 = SubscribeBar("rb000.SHFE", "1m", 20190701.0930);
  layer2 = SubscribeBar("rb000.SHFE", "15m", 20190601.0930);
}
//初始化事件函数，策略运行期间，首先运行且只有一次
OnReady()
{

}
//Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
OnBar(ArrayRef<Integer> indexs)
{

}
//Barbegin 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
Onbaropen(ArrayRef<Integer> indexs)
{

}
```

```
//持仓更新事件函数，参数 pos 表示更新的持仓结构体
OnPosition(PositionRef pos)
{

}

//委托更新事件函数，参数 ord 表示更新的委托结构体
OnOrder(OrderRef ord)
{

}

//成交更新事件函数，参数 ordFill 表示更新的成交结构体
OnFill(FillRef ordFill)
{

}

//定时器更新事件函数，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值
OnTimer(Integer id,Integer millsecs)
{

}
```

案例二：BAR 数据的订阅和退订

学习焦点：

通过这个案例主要展示一下 BAR 数据的订阅和退订，并解释下数据源和程序之间的交互机制。

数据源的生成方式：

先简单阐述 TBQuant 订阅 BAR 数据的几种方式：第一种手动添加，手动添加就是在定义策略研究或者策略交易的时候，添加数据源添加进来的；第二种初始化代码添加，这就是在 oninit 事件中用 subscribebar 函数订阅；第三种是在其它事件中比如 onbar 事件中用 subscribebar 函数订阅。

以上三种方式不管任何一种方式添加的数据源都会参与数据源集合的生成。如果对策略单元右键打开 K 线我们将看到前四个数据源的 K 线，并且是按照时间点进行对齐的。

不同方式的区别：

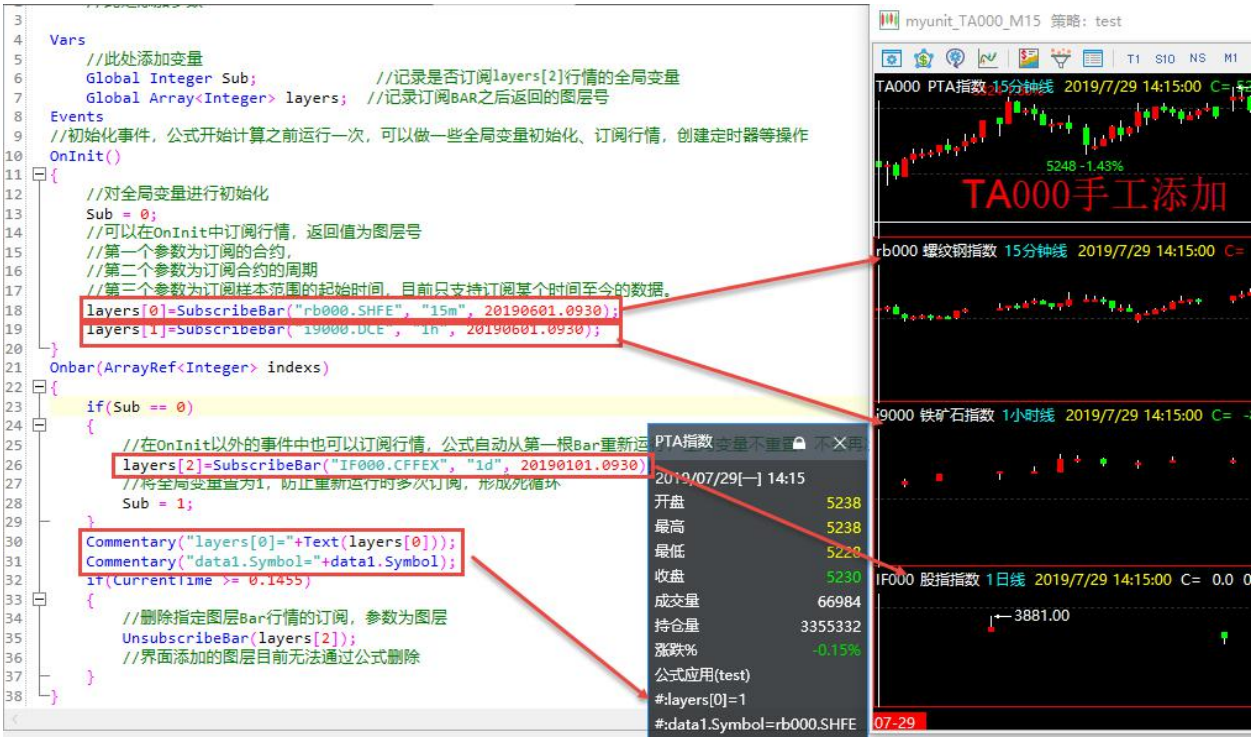
1）手工添加的数据源只能手动删除；

2）代码生成的数据源只能代码删除；

3）手工添加的数据源序号从 0 开始排列，代码生成的数据源序号按照代码执行顺序依次排列在手工数据源后面；比如在下图所示，TA000 是手动添加的，数据源序号是 0，其它三个品种按照代码添加顺序依次是 1 到 3；

4）代码添加数据源如果不是在 oninit 事件中添加的，需要注意避免多次添加。因为 oninit 事件只会执行一次，而其它事件可能会执行多次；

5）代码添加数据源如果不是在 oninit 事件中添加的，公式运行到满足条件的时候，会添加数据源进来。这时候公式重新从第一根 BAR 运行。但是全局变量不重置，oninit()也不会再次运行；
6）回测的数据源只包括用户添加和公式 OnInit 添加，其他事件添加不能回测。
7)代码添加数据源一定要注意不同交易所，品种代码的大小写。如果写错大小写，会报获取数据失败。中金所和郑商所是大写，上交所和大商所是小写。
8）onbar 的参数 indexs 和 subscribebar 订阅 K 线返回的序号值，都对应于数据源的序号。需要注意的是 indexs 在实时行情中只返回当时有 tick 的数据源序号。
9）目前的 TBQuant 运行机制支持没有手工添加数据源的情况下可以执行程序，也就是数据源可以全部从代码中订阅。但是要注意不订阅 K 线时不要使用依赖 K 线的局部变量。
10）所有 bar 数据的调用可以直接用 data[i].方式读取，也可以用 getbar 函数读取。



代码如下：

```
Params
  //此处添加参数

Vars
  //此处添加变量
  Global Integer Sub;        //记录是否订阅 layers[2]行情的全局变量
  Global Array<Integer> layers; //记录订阅 BAR 之后返回的图层号
Events
//初始化事件，公式开始计算之前运行一次，可以做一些全局变量初始化、订阅行情，创建定时器等操作
OnInit()
{
  //对全局变量进行初始化
  Sub = 0;
```

```
    //可以在 OnInit 中订阅行情，返回值为图层号
    //第一个参数为订阅的合约，
    //第二个参数为订阅合约的周期
    //第三个参数为订阅样本范围的起始时间，目前只支持订阅某个时间至今的数据。
    layers[0]=SubscribeBar("rb000.SHFE", "15m", 20190601.0930);
    layers[1]=SubscribeBar("i9000.DCE", "1h", 20190601.0930);
}
Onbar(ArrayRef<Integer> indexs)
{
    if(Sub == 0)
    {
        //在 OnInit 以外的事件中也可以订阅行情，公式自动从第一根 Bar 重新运行，全局变量不重置，不会再次执行 OnInit
        layers[2]=SubscribeBar("IF000.CFFEX", "1d", 20190101.0930);
        //将全局变量置为 1，防止重新运行时多次订阅，形成死循环
        Sub = 1;
    }
    Commentary("layers[0]="+Text(layers[0]));
    Commentary("data1.Symbol="+data1.Symbol);
    if(CurrentTime >= 0.1455)
    {
        //删除指定图层 Bar 行情的订阅，参数为图层
        UnsubscribeBar(layers[2]);
        //界面添加的图层目前无法通过公式删除
    }
}
```

案例三：TICK 数据的订阅和退订

学习焦点：

通过这个案例主要了解一下 TICK 数据如何获取和 TICK 数据有哪些属性。
TICK 数据获取的方式有两种：
第一种是先定义一个 TICK 类型的变量 a，对于数据源中的合约使用 datai.gettick(a)，这样获取到 datai 的 TICK；
第二种是也是用 gettick()，不过在参数中指定了合约代码。
注意：不管哪一种方式获取 TICK 数据，首先都要先订阅 tick 级别的 BAR 数据，然后使用 onbar 机制来接收 TICK 行情。

代码如下：

Params

```
Vars
    Tick a;
    Global Integer i;
    Global Integer maxCount;
    Tick tickdata;
    Global Integer layer1(-1);
Events
    //初始化事件函数，策略运行期间，首先运行且只有一次
    OnInit()
    {
        maxCount = 0;
        //订阅深度行情，目前只有公式订阅的深度行情在 OnBar 函数中才可以 GetTick 获取结果
        //返回当前订阅成功的图层信息
        layer1 = SubscribeBar("IF888.CFFEX","tick",20201030.000000);
        CreateTimer(10000);
    }

    //Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
    OnBar(ArrayRef<Integer> indexs)
    {
        For i=0 To GetArraySize(indexs)-1
        {
            //只有订阅深度行情的图层才可以 GetTick 获取信息 否则 GetTick 返回 false
            if(layer1 == indexs[i])
            {
                //可以指定图层获取 Tick 结构 注意对应图层信息
                Data[indexs[i]].GetTick(tickdata);
                Data[indexs[i]].Commentary("申买价 1="+text(tickdata.bidask1.askP));
    //申买价 1
                Data[indexs[i]].Commentary("申买量 1="+text(tickdata.bidask1.askV));
    //申买量 1
                Data[indexs[i]].Commentary("申卖价 1="+text(tickdata.bidask1.bidP));
    //申卖价 1
                Data[indexs[i]].Commentary("申卖量 1="+text(tickdata.bidask1.bidV));
    //申卖量 1
                Data[indexs[i]].Commentary("申买价 5="+text(tickdata.bidask5.askP));
    //申买价 5
                Data[indexs[i]].Commentary("申买量 5="+text(tickdata.bidask5.askV));
    //申买量 5
                Data[indexs[i]].Commentary("申卖价 5="+text(tickdata.bidask5.bidP));
    //申卖价 5
                Data[indexs[i]].Commentary("申卖量 5="+text(tickdata.bidask5.bidV));
    //申卖量 5
                Data[indexs[i]].Commentary("last="+text(tickdata.last));
```

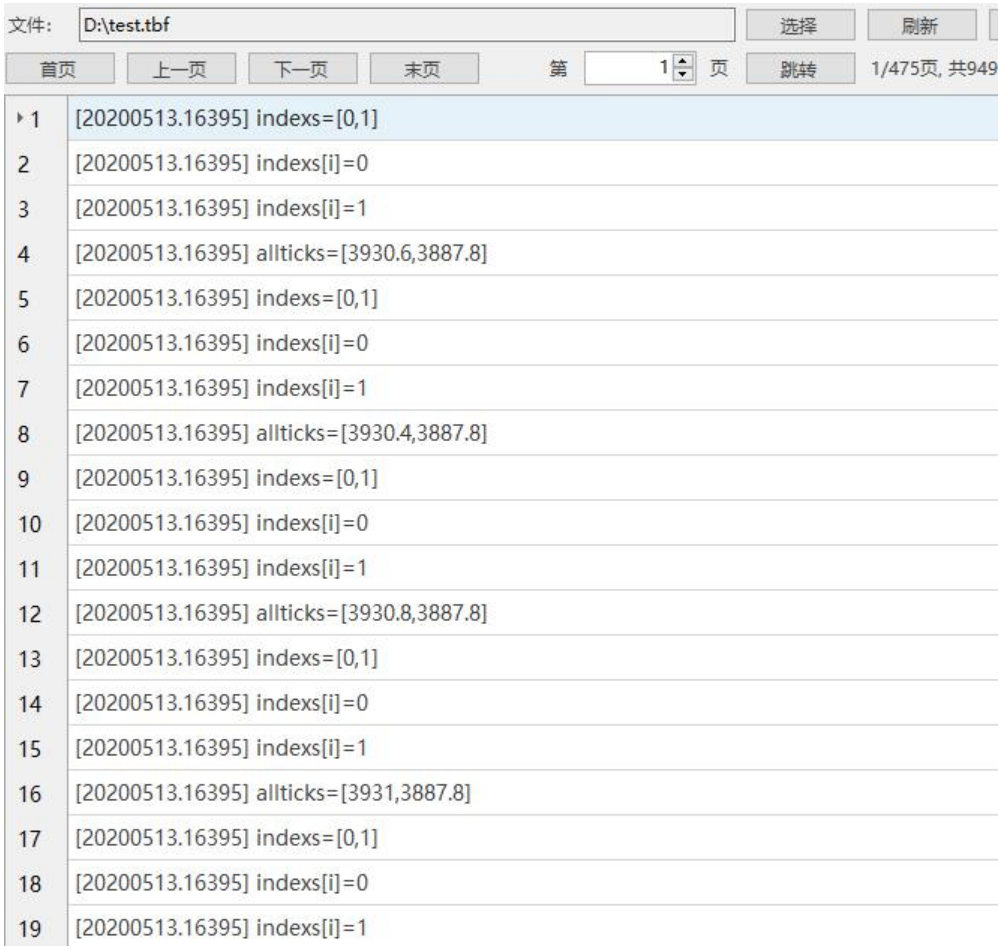
```
    //现价
                Data[indexs[i]].Commentary("open=="+text(tickdata.open));
    //开盘价
                Data[indexs[i]].Commentary("high="+text(tickdata.high));
    //最高价
                Data[indexs[i]].Commentary("low="+text(tickdata.low));
    //最低价
                Data[indexs[i]].Commentary("limitUp="+text(tickdata.limitUp));
    //涨停价
                Data[indexs[i]].Commentary("limitDown="+text(tickdata.limitDown));
    //跌停价
                Data[indexs[i]].Commentary("preClose="+text(tickdata.preClose));
    //昨收盘
                Data[indexs[i]].Commentary("preSettlement="+text(tickdata.preSettlement));
    //昨结算
                Data[indexs[i]].Commentary("symbol="+tickdata.symbol);
    //合约名
                Data[indexs[i]].Commentary("dateTime="+text(tickdata.dateTime));
            //时间, Numeric 类型 20190611.1345
                Data[indexs[i]].Commentary("volume="+text(tickdata.volume));
    //现手
                Data[indexs[i]].Commentary("totalVolume="+text(tickdata.totalVolume));
    //总成交
                Data[indexs[i]].Commentary("openInt    =    "    +    text(tickdata.openInt));
// 持仓量
                Data[indexs[i]].Commentary("avgPrice    =    "    +    text(tickdata.avgPrice));
// 实时均价(今日结算价)
                Data[indexs[i]].Commentary("totalTurnOver = " + text(tickdata.totalTurnOver));
// 总成交金额(单位: 元)
                Data[indexs[i]].Commentary("insideVol  =  "  +  text(tickdata.insideVolume));
// 内盘
                Data[indexs[i]].Commentary("outsideVol  =  " + text(tickdata.outsideVolume));
// 外盘
                Data[indexs[i]].Commentary("odayClose   =   "   +   text(tickdata.todayClose));
// 今日收盘价
                Data[indexs[i]].Commentary("settlePrice  =  "  +  text(tickdata.settlePrice));
// 当日结算价
                Data[indexs[i]].Commentary("status    =    "    +    text(tickdata.status));
// 状态(暂未启用)
                Data[indexs[i]].Commentary("preOpenInt  =  "  +  text(tickdata.preOpenInt));
// 昨持仓
                Data[indexs[i]].Commentary("turnOver = " + text(tickdata.turnOver));
    //成交金额
            }
```

```
    }
}

//定时器更新事件函数，参数 id 表示定时器的编号，millsecs 表示定时间的间隔毫秒值
OnTimer(Integer id,Integer millsecs)
{
    maxCount = maxCount+1;
    //收到 10 次 timer 时间之后退订
    if(maxCount == 10)
    {
        //退订深度行情 传入图层 ID
        UnsubscribeBar(layer1);
        layer1 = -1;
    }
}
```

案例四：多个 TICK 数据的订阅处理

学习焦点：

通过这个案例主要了解一下多个 TICK 数据如何获取和使用。
1）注意 onbar 机制中 indexs 参数的意义。
Indexs 的具体意义是每一个时刻所接受的 TICKS 的图层号。



2）tick 数据如何获取

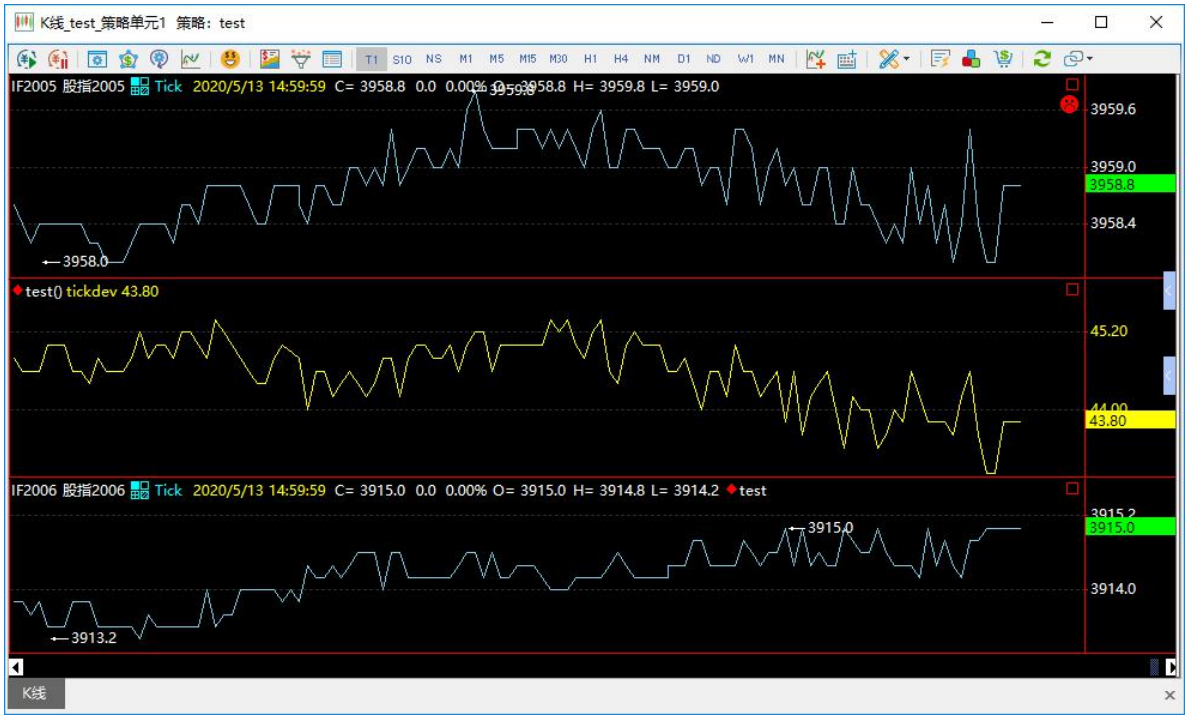
Data[indexs[i]].gettick(tickdata)会把 Data[indexs[i]]的 tick 存储到 Data[indexs[i]].tickdata 的变量里面。

特别注意：tickdata 是个局部变量，在每个图层都有自己的 tickdata。函数参数当中 tickdata 前面虽然没有加 Data[indexs[i]]，但是根据 TBQuant 的机制，函数里面的局部变量不加 data[i]前缀的默认跟随函数前面的前缀。

代码如下：

```
Params
Vars
  Tick tickdata;
  Global Array<Numeric> allticks;  //存储多个合约的 TICK 的最新价
  Series<Numeric> tickdev;         //计算合约的价差
  Global Integer i;
Defs
  //log 输出
  Integer LogFile(String str)
  {
    FileAppend("d://"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
```

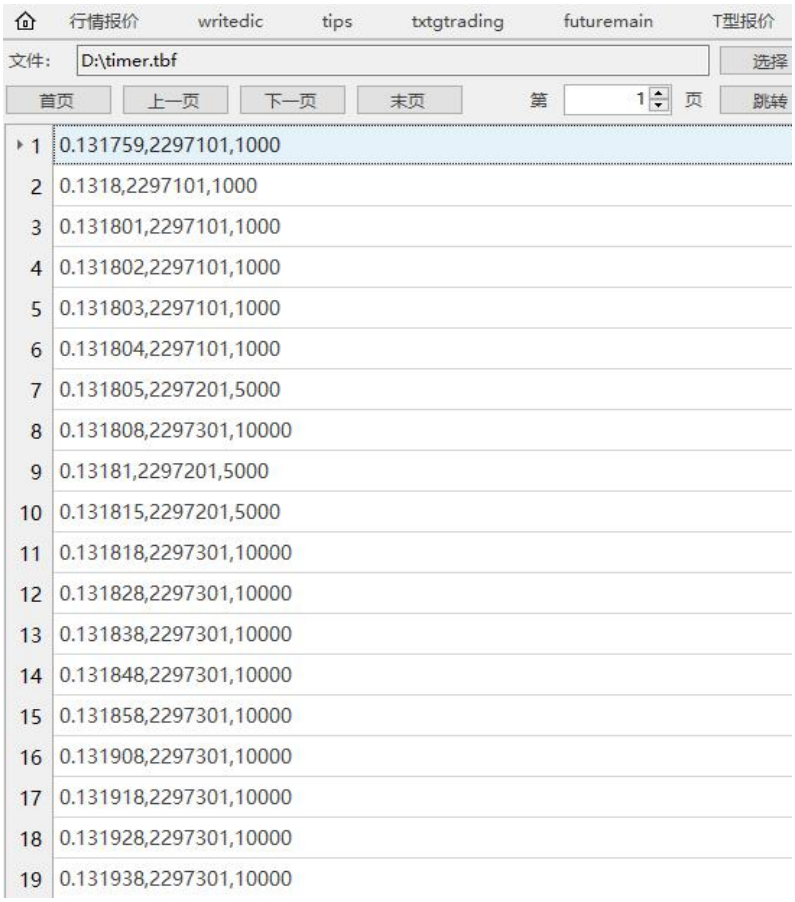
```
  }
Events
OnInit()
{
  SubscribeBar("IF1908.CFFEX","Tick" ,20200513,0);      //订阅 if1908 的 TICK
  SubscribeBar("IF1909.CFFEX","Tick" ,20200513,0);      //订阅 if1909 的 TICK
  FileDelete("d://"+FormulaName()+".tbf");
}
//读取每一个时刻可以接收到的 TICK，把 tick 的最新价存储到 allticks 的数组中。
OnBar(ArrayRef<Integer> indexs)
{
  LogFile("indexs="+TextArray(indexs));
  For i=0 To GetArraySize(indexs)-1
  {
    LogFile("indexs[i]="+Text(indexs[i]));
    Data[indexs[i]].GetTick(tickdata);
    allticks[indexs[i]]=Data[indexs[i]].tickdata.last;
  }
  LogFile("allticks="+TextArray(allticks));
  if(allticks[0]*allticks[1]>0)      //计算价差并画图
    tickdev=allticks[0]-allticks[1];
  PlotNumeric("tickdev",tickdev);
}
```



案例五：多个定时器的处理

学习焦点：

通过这个案例主要了解下多个定时器怎么创建、销毁和识别。
创建定时器和订阅 BAR 数据一样会返回一个编号，而多个定时器的事件都是通过 ontimer 接收，所以可以通过这个编号来区分不同的定时器。
定时器的公式只能在策略交易或 K 线运行，在策略研究无法运行。
这是在 20200519.131800 启动策略的输出文档。



代码如下：

```
Params
  //此处添加参数
Vars
  //此处添加变量
  Global Integer timeID1;
  Global Integer timeID2;
  Global Integer timeID3;
  Global String filepath;
Events
OnInit()
{
```

```
  //创建定时器、间隔时间 ,开始时间, 启动次数
    timeID1 = CreateTimer(1000);  //创建定时器,1000 毫秒触发一次, 立即开始, 一直触发
    timeID2 = CreateTimer(5000, 20200519.131800,3);  //创建定时器,5000 毫秒触发一次,
20200519.131800 开始, 触发 3 次
    timeID3 = CreateTimer(10000,0,10);  //创建定时器,10000 毫秒触发一次, 立即开始, 触发 10
次
    filepath="d:/timer.tbf";
    FileDelete(filepath);
}
OnTimer(Integer id,Integer millsecs)
{
  //可以通过 id 判断是哪个定时器触发的定时时间
  if(id == timeID1)
  {
    FileAppend(filepath,Text(CurrentTime)+","+Text(id)+","+Text(millsecs));
  }
  if(id == timeID2)
  {
    FileAppend(filepath,Text(CurrentTime)+","+Text(id)+","+Text(millsecs));
    //通过定时器 Id 可以停止定时器
    StopTimer(timeID1);
  }
  if(id == timeID3)
  {
    FileAppend(filepath,Text(CurrentTime)+","+Text(id)+","+Text(millsecs));
  }
}
```

案例六：onbar 事件的应用

学习焦点：

通过这个案例主要了解 onbar 事件的三个细节：

1、引用参数 indexs 是什么？

Indexs 在历史行情存储的是所有数据源的图层号，在实时行情存储的是实时时刻接收到 TICKS 对应的数据源图层号。这里需要对 onbar 的运行机制稍微了解一下，对于历史 BAR 则是每根 BAR 只运行一次，对于实盘行情正在形成的 BAR 则每接收到一个新的 TICK 都会运行一次。

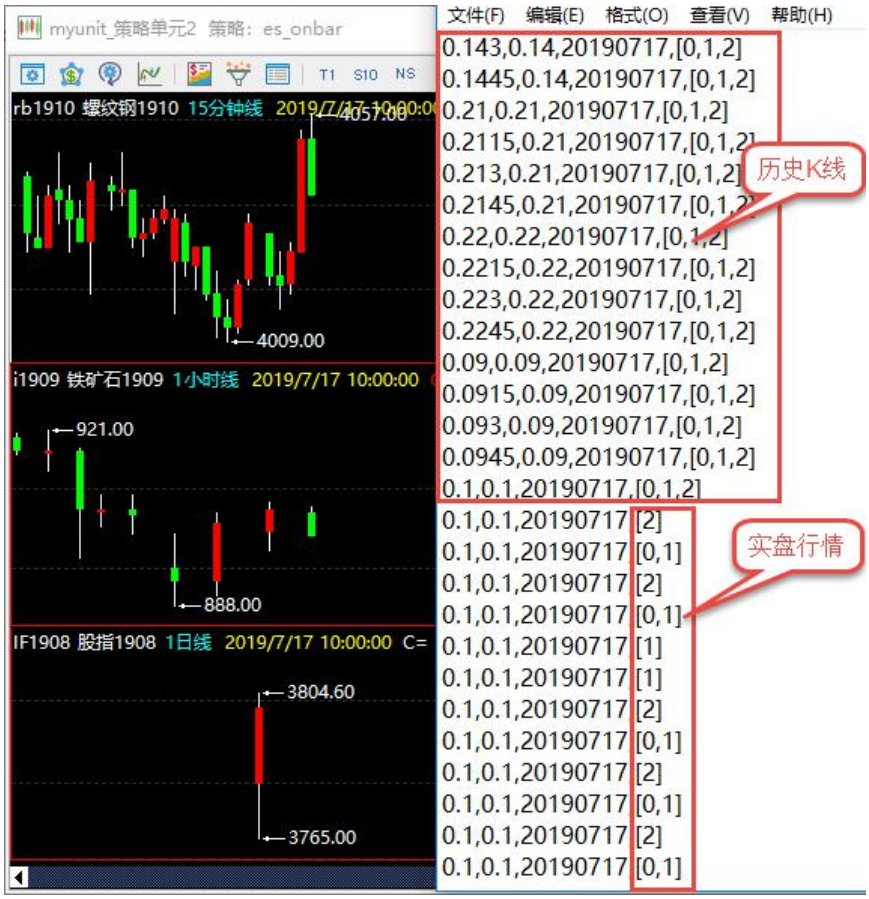
结合本节案例，我们叠加的三个品种，历史 K 线的起始日期是 20190716.0930，我在 20190717.1000 的时刻启动了这个策略交易开始接收实时 TICK 行情。

在代码中我们有下面这个输出语句：

```
FileAppend("d:/indexs.tbf",Text(data0.Time)+","+Text(data1.Time)+","+Text(data2.Date)
+","+TextIntegerArray(indexs));
```

可以看到下图的输出中从 20190717.1000 的时点，输出记录分为两个部分，上面是历史 K 线的记录，

Indexs 的内容是【0，1，2】；下面是实时 TICK 的记录，indexs 的内容则是实时的时刻接收到 TICKS 对应的数据源图层号。比如实盘第一个时刻只接收到了 1 个 tick，这个 tick 来自于图层 2，所以 indexs 的内容就是【2】，长度为 1 的数组。



2、getbar 函数如何获取 bar 数据，BAR 属性有哪些？

获取 bar 数据的函数是 getbar 通过在 getbar 前面加上 data[i].的前缀来确定调取的 BAR 的数据源的图层号。

Bar 的属性在代码后段的 commentary 语句我们可以看到有 8 种属性。

3、onbar 与 beginend 的兼容

可以说 onbar 是完全兼容 beginend 的。所以之前版本的 beginend 的代码可以完全 copy 到 onbar 中直接运行。

不同之处在于 onbar 增加了 indexs 的引用参数，可以识别下实时 TICK 的接收。

代码如下：

```
Params

Vars
  Numeric index;
  Bar curBar;
  Numeric i;
  Global Array<Numeric> layers;
  Global Numeric sub;
```

```
Events
 OnInit()
{
   layers[0]=SubscribeBar("rb1910.SHFE", "15m", 20190716.0930);
   layers[1]=SubscribeBar("i1909.DCE", "1h", 20190716.0930);
}
//OnBar 事件，同之前版本的 Begin-End，在操作界面添加商品，或者在公式中订阅 Bar 数据。
//收到 K 线行情时触发 OnBar 事件，参数为有更新的图层的下标数组。
Onbar(ArrayRef<Integer> indexs)
{
    if(Sub == 0)
   {
      //在 OnInit 以外的事件中也可以订阅行情，公式自动从第一根 Bar 重新运行，全局变量不重置，
不会再次执行 OnInit
      layers[3]=SubscribeBar("IF1908.CFFEX", "1d", 20190716.0930);
      //将全局变量置为 1，防止重新运行时多次订阅，形成死循环
      Sub = 1;
      FileDelete("d:/indexs.txt");
   }
    FileAppend("d:/indexs.tbf",Text(data0.Time)+","+Text(data1.Time)+","+Text(data2.Date)+
","+TextArray(indexs));
   for index = 0 to GetArraySize(indexs) - 1
   {
      //GetBar 函数用于获取图层的 Bar 结构
      //参数 1 为 Bar 结构，传出参数，用于接收 Bar 数据
      //参数 2 为回溯，取前面第几个 Bar 的数据
      //回溯可以不填，默认为获取当前 Bar
      data[indexs[index]].GetBar(curBar);

      //回溯 6，获取前面第 6 根 bar
      data[indexs[index]].GetBar(curBar,6);

      //可以根据 Bar 获取 K 线的开高低收
      i = indexs[index];        //indexs[index]为图层号

      data[i].commentary(Text(curBar.datetime));      //bar 时间
      data[i].commentary(Text(curBar.open));          //bar 开盘价
      data[i].commentary(Text(curBar.high));          //bar 最高价
      data[i].commentary(Text(curBar.low));           //bar 最低价
      data[i].commentary(Text(curBar.close));         //bar 收盘价
      data[i].commentary(Text(curBar.volume));        //bar 成交量
      data[i].commentary(Text(curBar.openInt));       //bar 持仓量
      data[i].commentary(Text(curBar.turnOver));      //bar 成交金额
   }
```

```
    }
```

3.3 事件驱动进阶使用案例

案例七：onbaropen 函数的具体应用

学习焦点：

通过这个案例主要了解下 onbaropen 的使用，以及 onbar 和 onbaropen 的区别。
Onbaropen 与 onbar 运行机制的唯一区别在于在实时行情的 BAR 上，onbar 是每个 TICK 都会驱动代码运行一次，而 onbaropen 只在这根 BAR 的第一个 TICK 才会驱动代码运行一遍。

代码如下：

```
Params
  //此处添加参数
  Numeric FastLength(5);
  Numeric SlowLength(20);
Vars
  Numeric AvgValue1;
  Numeric AvgValue2;
  Global Integer i(0);
  Global Integer currentbarlast(0);
Defs
  Integer LogFile(String str)
  {
    FileAppend("d:/"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
  }

Events
  //此处实现事件函数

  //初始化事件函数，策略运行期间，首先运行且只有一次
  OnInit()
  {
    SubscribeBar("rb1910.SHFE","1m",DateTimeAdd(SystemDateTime(),-300*60));
  }
  //Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组.
  //区别于 onbar 的地方在于只在实时行情 bar 的第一个 TICK 运行
  OnBarOpen(ArrayRef<Integer> indexs)
  {
```

```
If(CurrentBar>CurrentBarlast)
   {
   LogFile("newbar begin!");
   i=0;
   }
Else
   i=i+1;
CurrentBarlast=CurrentBar;
LogFile("newbar ticksnum ="+Text(i));
AvgValue1 = Average (Close,FastLength);
AvgValue2 = Average(Close,SlowLength);
PlotNumeric("MA1",AvgValue1);
PlotNumeric("MA2",AvgValue2);

If(MarketPosition <>1 && AvgValue1 > AvgValue2)
{
   Buy(1,Open);
}

If(MarketPosition <>-1 && AvgValue1 < AvgValue2)
{
   SellShort(1,Open);
}
}
```

案例八：交易助手的实现

学习焦点：

通过这个案例主要了解如何通过事件函数来对委托单进行控制。交易助手实现的功能就是对委托单进行管理，主要操作就是：超时撤单和撤单后重发委托。

本案例主要关注几点：

1、首先理解委托单的来源和处理，以及不同事件函数处理哪一部分。

委托单的来源有两个：

1）启动策略之后，ontimer()定时把交易账号的历史委托单先同步进来；

2）onorder()事件实时的接收状态发生变化的委托单。

委托单的处理主要是两个：

1）如果未完成的委托单超时，需要撤单。这个操作在 ontimer 中进行。

2）如果撤单完成，是否重发委托。这个操作在 onorder 中进行。

在 oninit 中完成两件事：

1） 订阅账户的所有委托单

2） 创建一个定时器

在 ontimer 中完成两件事：

1）把所有账户的处于未完成状态的历史委托单中给同步进来，把委托单 ID 存放于 activeOrderMap；

2）执行了 checkActiveOrder，这个函数主要是对未完成的委托单中超时的，进行撤单操作。

在 onorder 中完成两件事：

1）处理委托单的变化。对于状态有变化的委托单，如果新的状态不是完成状态的，存入 activeOrderMap；否则从 activeOrderMap 中删除。

2）如果是已撤销状态的，并且委托源是本策略的，会重发委托。

2、委托单从产生到终结有多少种状态，应该怎么应对

委托单总共有 7 种状态，通常分为完成和未完成两大类。

其中 enum_deleted,enum_filled,enum_canceled 是完成的，其它是未完成的。

枚举值	基本含义	对应整数值
Enum_delcare	申报中	1
Enum_declared	申报完成	2
Enum_deleted	交易所废单	3
Enum_fillpart	部分成交	4
Enum_filled	全部成交	5
Enum_canceled	已经撤销	6
Enum_canceling	撤销中	7

3、不同账户不同类型的委托单怎么获取

首先在 oninit()中用 A_SubscribeTradeByCreateId(Enum_Trade_Source_ALL)函数要订阅所有账户的委托单。然后在 ontimer()中使用 A_GetUnFillOrderIDs(ids,"",i)函数可以获取第 i 个账户的未完成状态的委托单编号，并将委托单存储于 ids 中。

还有另外一个途径就是 onorder()会接收状态改变的委托单。

4、委托单的发出和撤销的函数怎么使用

A_SendOrderEx 函数的语法不再介绍，可以参照 TBQuant 网站【函数】里面 order 的属性和 A_SendOrderEx 的参数说明进行理解。这里主要讲参数里面的两点设置。

1）委托数量用 ord.volume-ord.fillVolume

这是考虑了委托单部分成交的情况，所以做这个处理

2）委托价格用 ord.price+/-priceOffset

这是考虑了委托偏移，要根据买卖方向进行加减

A_DeleteOrderEx ()这个函数用于撤单，可以直接参照 TBQuant 网站【函数】里面的说明。

5、Map 的使用

Map 是 TBQuant 新增的一种扩展数据类型。在这个案例当中使用 Map 来存储查找删除委托单，比使用数组要方便很多。大家可以学习下 Map 的基本函数。

代码如下：

```
//-------------------------------------------------------------------------
```

```
// 简称: demo_EventTradeHelper
// 名称: 事件驱动_交易助手实例
// 类别: 公式应用
// 类型: 内嵌应用
//--------------------------------------------------------------------
Params
    //此处添加参数
    Numeric intervalSecs(15);   //监控间隔
    Numeric priceOffset(0);     //重新委托价格偏移
Vars
    //此处添加变量
    Numeric tmp;    //系统时间
    Integer cnt(0); //账户个数
    Integer i(0);   //遍历数组
    Integer j(0);   //遍历数组
    Order tmpOrder; //委托
    Array<Integer> orderIds;            //委托 ID
    Array<Integer> activeMapKeys;   //未成交委托 map 的 key, 用于遍历 map
    Global Integer timerId;             //定时器 ID
    Global Map<Integer, Order>  activeOrderMap;     //未成交的委托
    Global Map<String, Integer> accountIdMap;       //已经同步过委托的账户
Defs
    //此处添加公式函数
    //log 输出
    Integer LogFile(StringRef str)
    {
        FileAppend(FormulaName()+".txt","["+Text(SystemDateTime())+"] "+ str);
        Return 0;
    }
    //检查未成交委托, 根据委托时间撤单
    Integer CheckActiveOrder()
    {
        LogFile("检查未成交委托数量: "+Text(GetMapSize(activeOrderMap)));
        //获取 Map 变量中所有的 Key 集合
        GetMapKeys(activeOrderMap,activeMapKeys);
        //遍历 Key 集合访问 Map 变量的 key 对应的 Value
        For i=0 To GetArraySize(activeMapKeys)-1
        {
            tmpOrder=activeOrderMap[activeMapKeys[i]];
            tmp=SystemDateTime();
            //非交易时段忽略
            If(!IsTradingTime(tmpOrder.symbol,tmp))
            {
                Continue;
```

```
            }

    if(DateDiff(IntPart(tmpOrder.createDateTime),IntPart(tmp))*24*3600+TimeDiff(tmpOrder.cre
ateDateTime,tmp)> intervalSecs)
            {
                LogFile("撤单: "+tmpOrder.accountId+",index="+Text(tmpOrder.orderId)    );
                A_DeleteOrderEx(activeMapKeys[i]);
            }
        }
        Return 0;
    }
Events
    //初始化事件函数，策略运行期间，首先运行且只有一次
    OnInit()
    {
        timerId=createTimer(2000);
        A_SubscribeTradeByCreateId(Enum_Trade_Source_ALL);//订阅账户的所有委托，默认只触发本
策略发出的委托
    }

    //委托更新事件函数，参数 ord 表示更新的委托结构体
    OnOrder(OrderRef ord)
    {
        //如果没有同步过的账号委托，暂时不加入监控，要先在 Timer 同步
        //检查当前 key 是否已经添加到 Map 变量中
        If(!MapContain(accountIdMap, ord.accountId))
        {
            Return;
        }
        //如果委托已经撤单，且撤单操作员是当前策略，则重新委托
        LogFile(" 委托更新:
"+ord.accountId+",index="+Text(ord.orderId)+",exchOrdId="+ord.exchOrderId+",status="+Text(or
d.status) + "," + ord.createSource +","+ord.cancelSource+","+FormulaName()+","+ord.note);
        iF(ord.status == Enum_Canceled)
        {
            //检查当前 key 是否已经添加到 Map 变量中
            If(MapContain(activeOrderMap, ord.orderId))
            {
                //根据 Key 删除当前记录
                MapErase(activeOrderMap, ord.orderId);
                LogFile(" 删除监控委托:
"+ord.accountId+",oldindex="+Text(ord.orderId)+",exchOrdId="+ord.exchOrderId);

                If(ord.cancelSource == A_GetOrderCreateSource())
```

```
                {
                    If(ord.side==Enum_Buy)
                    {

    A_SendOrderEx(ord.symbol,ord.side,ord.combOffset,ord.volume-ord.fillVolume,ord.price+pri
ceOffset,orderIds,"",A_AccountIndex(ord.accountId,ord.brokerId));
                    }
                    Else
                    {

    A_SendOrderEx(ord.symbol,ord.side,ord.combOffset,ord.volume-ord.fillVolume,ord.price-pri
ceOffset,orderIds,"",A_AccountIndex(ord.accountId,ord.brokerId));
                    }
                    LogFile(" 撤单成功重新委托:
"+ord.accountId+",oldindex="+Text(ord.orderId)+",exchOrdId="+ord.exchOrderId);
                }
            }
        }
        Else If(ord.status == Enum_Filled || ord.status == Enum_Deleted)
        {
            //检查当前 key 是否已经添加到 Map 变量中
            If(MapContain(activeOrderMap, ord.orderId))
            {
                //根据 Key 删除当前记录
                MapErase(activeOrderMap, ord.orderId);
                LogFile(" 删除监控委托:
"+ord.accountId+",oldindex="+Text(ord.orderId)+",exchOrdId="+ord.exchOrderId);
            }
        }
        Else
        {
            activeOrderMap[ord.orderId] = ord;
        }
    }

    //定时器更新事件函数, 参数 id 表示定时器的编号, millsecs 表示定时间的间隔毫秒值
    OnTimer(Integer id,Integer millsecs)
    {
        //循环监控所有账号
        cnt = A_AccountCount();
        For i=0 To cnt-1
        {
            //检查是否已经同步过委托
            if(MapContain(accountIdMap,A_AccountID(i)))
```

```
            {
                Continue;
            }
            accountIdMap[A_AccountID(i)] = A_BrokerID(i);
            LogFile(A_AccountID(i)+" 开始同步委托");
            //同步未成交委托
            If(A_GetUnFillOrderIDs(orderIds,"",i))
            {
                For j=0 To GetArraySize(orderIds)-1
                {
                    If(A_GetOrder(orderIds[j], tmpOrder))
                    {
                        //放入 Map 直接下标访问进行赋值 下标为 Key 必须唯一
                        activeOrderMap[orderIds[j]] = tmpOrder;
                        LogFile(A_AccountID(i)+" 同步委托: "+Text(orderIds[j]));
                    }
                }
            }
            LogFile(A_AccountID(i)+" 同步委托完成");
        }
        CheckActiveOrder();
    }


//--------------------------------------------------------------------------
// 编译版本 2019/07/26 125622
// 版权所有 TradeBlazer Software 2003－2025
// 更改声明 TradeBlazer Software 保留对 TradeBlazer 平台
//          每一版本的 TradeBlazer 公式修改和重写的权利
//--------------------------------------------------------------------------
```

案例九：一个简单高频策略的完整处理

学习焦点：

通过这个案例主要了解一个高频策略会用到的策略和委托成交持仓等各方面的协调处理。主要关注点：
1）订阅行情、委托管理、持仓更新完整的处理流程。
2）委托单的存储和管理。注意委托单的七种状态。
3）仓位读取和属性的使用。仓位读取有两种方式：A_GetPosition(sym,myPos);在之后使用的是 onposition ()获取。
4）学习 onbaropen 的使用。在本案例当中每隔两根 bar 触发一次交易，并且只在 bar 的第一个 TICK 触发交易，这种情形使用 onbaropen 会比较方便。
仓位的属性则有助于策略进行最大仓位控制。

属性	类型	说明
brokerId	Integer	经济公司ID
accountId	String	资金账户ID
symbol	String	合约代码
longCurrentVolume	Integer	多头当前持仓
longYesterdayVolume	Integer	多头剩余昨仓
longActiveVolume	Integer	多头未成交的报单净委托量
longActiveCloseVolume	Integer	多头未成交的报单平仓委托量
longCanSellVolume	Integer	多头可平量
longMarketValue	Numeric	多头持仓市值
longAvgPrice	Numeric	多头均价（买均价）
longFloatPorfit	Numeric	多头浮动盈亏（买盈亏）
longUseMarginAmount	Numeric	多头占用的保证金
shortCurrentVolume	Integer	空头当前持仓
shortYesterdayVolume	Integer	空头剩余昨仓
shortActiveVolume	Integer	空头未成交的报单净委托量
shortActiveCloseVolume	Integer	空头未成交的报单平仓委托量
shortCanCoverVolume	Integer	空头可平量
shortMarketValue	Numeric	空头持仓市值
shortAvgPrice	Numeric	空头均价（买均价）
shortFloatPorfit	Numeric	空头浮动盈亏（买盈亏）
shortUseMarginAmount	Numeric	空头占用的保证金

代码如下：

```
/*
策略思路：
如果多头净仓位超过最大仓位，则增加空头仓位；
如果空头净仓位超过最大仓位，则增加多头仓位。
*/
Params
```

```
  //此处添加参数
  Integer netMaxPos(5);//净持仓上线
  String mySym("ag1912.SHFE");
Vars
  //此处添加变量
  Integer side(0);
  Integer lMaxPos(0);
  Integer sMaxPos(0);
  Array<Integer> ids;
  Global Position myPos;
  Global Integer i(0);
Defs
  //log 输出
  Integer LogFile(String str)
  {
    FileAppend(FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
  }


  //控制仓位循环发单
  Integer trySendOrder(StringRef sym,Numeric price,Integer volume)
  {
    If(len(myPos.symbol)==0)
    {
      A_GetPosition(sym,myPos);
    }
    lMaxPos=myPos.longCurrentVolume+myPos.longActiveVolume+myPos.longActiveCloseVolume;
    sMaxPos=myPos.shortCurrentVolume+myPos.shortActiveVolume+myPos.shortActiveCloseVolum
e;

    //多头超过最大持仓,提高空头持仓
    If(lMaxPos>=netMaxPos)
    {
      side=-1;
    }
    //空头超过最大持仓, 提高多头持仓
    If(sMaxPos>=netMaxPos)
    {
      side=1;
    }
    //如果有未成交单先平仓
    If(myPos.longActiveVolume !=0 || myPos.shortActiveVolume!=0
||myPos.longActiveCloseVolume !=0 || myPos.shortActiveCloseVolume!=0)
```

```
    {
      A_DeleteAccountOrder(sym);
      Return 1;
    }
    LogFile("try:"+Text(side)+",l="+Text(lMaxPos)+",s="+Text(sMaxPos));
    If(side>0)
    {
      if(sMaxPos==0)
      {
        A_SendOrderEx(sym,Enum_Buy,Enum_Entry,volume,price,ids);
      }Else if(myPos.shortCurrentVolume >0)
      {
        A_SendOrderEx(sym,Enum_Buy,Enum_Exit,volume,price,ids);
      }
    }Else
    {
      if(lMaxPos==0)
      {
        A_SendOrderEx(sym,Enum_Sell,Enum_Entry,volume,price,ids);
      }Else if(myPos.longCurrentVolume >0)
      {
        A_SendOrderEx(sym,Enum_Sell,Enum_Exit,volume,price,ids);
      }
    }
    Return 1;
  }
Events

  //初始化事件函数，策略运行期间，首先运行且只有一次
  OnInit()
  {
    SubscribeBar(mySym,"10s");
  }

  //OnBarOpen 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
  OnBarOpen(ArrayRef<Integer> indexs)
  {
    Range[i=0:DataCount-1]
    {
      if(BarStatus==2)
      {
        If(BarCount%2==0)
        {
          trySendOrder(Symbol(),Open,1);
```

```
            }
        }
    }
}

//持仓更新事件函数，参数 pos 表示更新的持仓结构体
OnPosition(PositionRef pos)
{
    myPos=pos;
        LogFile("持仓更新:
"+pos.symbol+",lsz="+Text(pos.longCurrentVolume)+",ssz="+Text(pos.shortCurrentVolume));
}

//委托更新事件函数，参数 ord 表示更新的委托结构体
OnOrder(OrderRef ord)
{
    //如果委托已经撤单，且撤单操作员是当前策略，则重新委托
        LogFile("委托更新:
"+ord.accountId+",index="+Text(ord.orderId)+",exchOrdId="+ord.exchOrderId+","+ord.note+",
"+ord.createSource);
}
```

案例十：OnReady 函数的具体应用

学习焦点：

通过这个案例主要了解如何通过 onReady 设置保证金率、手续费、滑点、委托偏移和委托映射和忽略，以及数据源不参与策略报告统计。

首先需要注意的三点：

1、菜单和代码都可以设置这些内容，如果两者设置内容不一样，以哪一个设置为准？

答：以后设置为准，先运行公式再设置菜单，就是菜单的设置生效；先设置菜单再运行公式，就是公式的设置生效。

2、这些设置可以对策略单元的不同数据源分开进行设置吗?还是只能所有数据源统一设置？

答：保证金、手续费、滑点和委托偏移委托映射，都可以在函数前面加 data[i]来实现只对 data[i]进行个性设置。忽略自动交易的相关函数目前还不能区分数据源。

3、可以在哪些事件中设置这些内容？

答：保证金手续费滑点可以放在 OnReady 和 OnBar 中；委托偏移委托映射 /忽略自动，数据源不参与策略报告统计可以放在 Oninit、OnReady 和 OnBar 中。

然后我们分四类来说明分别如何设置：

1）保证金、手续费、滑点设置

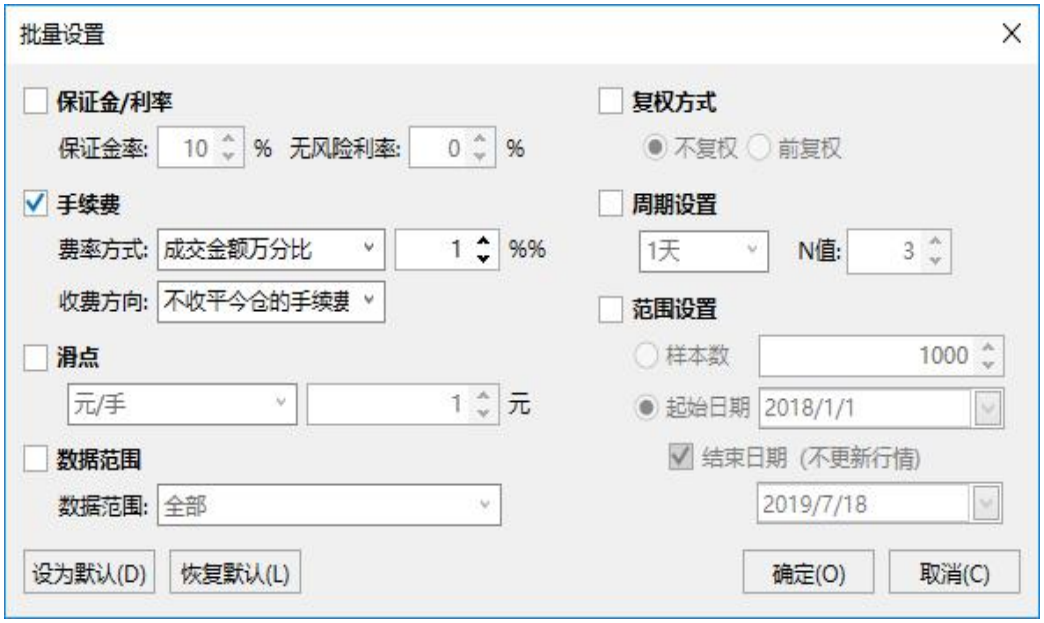
保证金比率的设置通过下面函数实现：

SetMarginRate(0.1)：将保证金比率设置为 10%。

手续费的设置的各参数的含义如下：

SetCommissionRate(Enum_Rate_AmountPerHand, 10.0)；手续费设置为一手 10 元
SetCommissionRate(Enum_Rate_PointPerHand, 10.0)； 手续费设置为一手 10 跳
SetCommissionRate(Enum_Rate_ByFillAmount, 10.0)； 手续费设置为成交金额的万分之 10
SetCommissionRate(Enum_Rate_OnlyExit, 10.0)； 手续费设置为只对平仓收取
SetCommissionRate(Enum_Rate_FreeOfExitToday, 10.0)； 手续费设置为平今仓免收
手续费实际应用设置：
比如下图所示的手续费设置，怎么通过代码实现。
手续费设置一般是要设置两项内容，这样可以用下面的 bitor 把两个参数括起来。这样就可以实现下图所示的功能。

```
SetCommissionRate(BitOr(Enum_Rate_FreeOfExitToday, Enum_Rate_ByFillAmount),1);
```



滑点通过下面两个函数实现：
SetSlippage(Enum_Rate_AmountPerHand, 10.0)；滑点设置为一手 10 元
SetSlippage(Enum_Rate_PointPerHand, 10.0)；滑点设置为一手 10 跳

2）委托设置

委托设置分两类，一种是委托价格的偏移，一种是合约的映射设置。
委托价格的偏移通过下面函数实现：
SetOrderPriceOffset(1)；按照按叫买叫卖价委托偏移 1 跳。
委托映射通过下面两个函数实现：
SetOrderMap2MainSymbol()；映射到主力合约
Bool SetOrderMap2AppointedSymbol(StringArrayRef symbols, NumericArrayRef multiples)映射到指定合约，可以指定合约和下单的倍数。
委托映射可以方便的应用于批量下单指数交易。
比如我们自己根据 rb1910,i1909,j1909 按照手数 20：4：1 的比例做了一个黑色指数，这个在 TBQuant 里面可以方便实现，我们这里只说明下单的委托映射环节，如何实现指数成分商品的批量下单。
我们可以用这样一段代码来实现：

```
Events
OnReady()
{
```

```
    symbols[0] = "rb1910.SHFE";
    Symbols[1]="i1909.DCE";
    Symbols[2]="j1909.DCE";

    multiples[0] = 20.0;
    multiples[1] = 4.0;
    multiples[2] = 1.0;

    SetOrderMap2AppointedSymbol(symbols, multiples);
    SetOrderPriceOffset(1);
}
```

特别注意：合约的代码后面要加上交易所的代码。

3）忽略自动交易处理

忽略自动和清除忽略自动有两类函数：

忽略自动用下面的四个函数，需要指定委托单的类别。

```
AddTradeFlag(Enum_Trade_Ignore_Buy);  //忽略多开
AddTradeFlag(Enum_Trade_Ignore_Sell);  //忽略多平
AddTradeFlag(Enum_Trade_Ignore_SellShort);  //忽略空开
AddTradeFlag(Enum_Trade_Ignore_BuyToCover);  //忽略空平
```

清除忽略自动用下面四个函数，也需要指定委托单的类别。

```
ClearTradeFlag(Enum_Trade_Ignore_Buy);  //清除忽略多开
ClearTradeFlag(Enum_Trade_Ignore_Sell);  //清除忽略多平
ClearTradeFlag(Enum_Trade_Ignore_SellShort);  //清除忽略空开
ClearTradeFlag(Enum_Trade_Ignore_BuyToCover);  //清除忽略空平
```

4）数据源不参与策略报告统计

`Data[i].AddDataFlag (Enum_Data_NotGenReport)`；数据源 i 不参与策略报告统计。

案例十一：Rerun 应用：参数自动优化及应用

学习焦点：

通过这个案例主要理解下 rerun 函数的使用。
1）Rerun 重新运行公式，但并不运行 oninit
2）全局变量也不进行初始化
3）rerun 只能在策略交易应用，在策略研究无效

策略思路：

普通的高低点通道策略，如何在代码中根据历史行情选择最优参数，应用到接下来的交易当中。

代码如下：

```
Params
  Numeric nums(5);     //确定参数数组的最大下标
Vars
  Series<Numeric> highline;   //高点通道
  Series<Numeric> lowline;  //低点通道
  Global Array<Numeric> hllen;  //通道参数的数组
  Global Array<Numeric> profit;  //每个参数对应的历史测试的净利润
  Global Array<Integer> id1;     //排序数组下标
  Global Array<Integer> id2;     //排序数组下标
  Global Integer i;          //循环变量
  Global Numeric optdone(0);     //是否完成优化
Defs
  //写文件函数的定义
  Integer LogFile(String str)
  {
    FileAppend("d://"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
  }
Events
  //此处实现事件函数
  //初始化事件函数，策略运行期间，首先运行且只有一次
  OnInit()
  {
    //订阅行情
    SubscribeBar("rb888.SHFE","1d",DateTimeAdd(SystemDateTime(),-300*24*3600));
    //设置参数优化的范围，以及排序下标数组的初始化
    for i=0 to nums
    {
      hllen[i]=5*(i+1);
      id1[i]=i;
```

```
    }
    id2=id1;
    i=0;
    FileDelete("d://"+FormulaName()+".txt");
}
OnReady()
{
    SetMarginRate(0.1);                    //将保证金比率设置为 10%。
    SetCommissionRate(Enum_Rate_ByFillAmount, 5.0); //手续费设置为成交金额的万分之 5
    SetSlippage(Enum_Rate_PointPerHand, 1.0);       //滑点设置为一手 1 跳
}
//Bar 更新事件函数，参数 indexs 表示变化的数据源图层 ID 数组
OnBar(ArrayRef<Integer> indexs)
{
    //计算指标
    highline=Highest(High[1],hllen[i]);
    lowline=Lowest(Low[1],hllen[i]);
    PlotNumeric("highline",highline);
    PlotNumeric("lowline",lowline);
    //进行交易
    If(MarketPosition <>1 && high>=highline)
    {
        Buy(1,max(highline,Open));
    }
    If(MarketPosition <>-1 && low<=lowline)
    {
        SellShort(1,min(lowline,Open));
    }
    //是否改变参数进行优化
    If(Date==20190730 and optdone==0)
    {
        profit[i]=NetProfit;
        //继续优化
        if(i<nums)
        {
            LogFile("ReRun...i="+Text(i));
            i=i+1;
            ReRun();
        }
        Else//优化完成，排序选择最优参数进行交易。
        {
            SortIds(profit,id1,id2,0,nums,False);
            i=id1[0];
            optdone=1;
```

```
            LogFile("ReRun...thebest="+Text(i));
            LogFile(TextArray(profit));
            ReRun();
        }
    }
}
```

运行结果：

案例十二：ReStart 应用：选股加波段交易的系统实现

学习焦点：

通过这个案例主要了解下 restart 的应用，可以很方便地把之前必须通过多次手工设置才能实现的繁琐工作全部自动化。

1）restart 是重启策略交易，所以所有变量包括全局变量都会初始化。所以运行过程中要把需要保留的信息写到本地。

2）restart 是重启策略交易，所以数据源也可以全部初始化。这个需要自己加条件进行区分。

策略思路：

1)给定初始股票池，根据一定规则进行选股，确定子股票池；
2)对自股票池加载波段交易策略，进行波段交易。

代码实现：

```
Params
  //此处添加参数
  Numeric choosedate(20200102);      //选股日期
  Numeric beginTime(20190101.090000);  //开始时间
  String dicname1("F1010010006"); //板块节点（科创板）
  string base1("stocktrade");         //股票池存储标志的块名
  String name1("choosecode");         //股票池存储标志的键名
  String freq1("1d");            //选股周期
  String freq2("5m");              //交易周期
  Numeric hllen(20);              //波段参数
Vars
  //此处添加变量
  Global String myfreq;           //数据周期
  Global Integer i(0);           //循环变量
  Global Integer n;             //股票池个数
  Global Array<String> myArray;      //股票池代码
  Global Array<String> subarray;      //子股票池代码
  Global Array<Numeric> zdf;        //涨跌幅
  Global Array<Integer> id1;        //数组下标
  Global Array<Integer> id2;        //数组下标
  Series<Numeric> highline;        //高点通道
  Series<Numeric> lowline;        //低点通道
  Global string choosecode;        //选股时间
  Global string dicsymbol;          //基础数据关联标的
  Global string dicname;       //基础数据的键名
Defs
  //写文件函数的定义
  Integer LogFile(String str)
  {
    FileAppend("d://"+FormulaName()+".tbf","["+Text(SystemDateTime())+"] "+ str);
    Return 0;
  }
Events
  OnInit()
  {
```

```
  //从数据库读取子股票池是否有选好的标志 无效值则没有选好，choosedone 则表示选好了
  choosecode=GetTBProfileString(base1,name1);
  LogFile("begin...choosecode="+choosecode);
  //根据股票池是否确立，确定行情订阅
  if(choosecode==InvalidString)
  {
    FileDelete("d://"+FormulaName()+".txt");
    LogFile("init1...choosecode="+choosecode);
    myfreq=freq1;
    dicsymbol=dicname1;
    dicname="TB_INDUSTRY";
  }
  Else
  {
    myfreq=freq2;
    dicsymbol="myindustry";
    dicname="001";
  }
  GetDicValue(dicname,dicsymbol,SystemDateTime(),myArray);
  n=GetArraySize(myArray);
  LogFile("myArray="+TextArray(myArray));
  //订阅行情
  For i=0 To n-1
  {  //加载所有成分合约到图层
    SubscribeBar(myArray[i],myfreq,beginTime);
  }
  SetBackBarMaxCount(1);
  //SetInitCapital(20000000);
}
OnBar(ArrayRef<Integer> indexs)
{
  //选股
  if(choosecode==InvalidString)
  {
    //涨跌幅排序筛选股票
    if(date==choosedate)
    {
      //涨跌幅排序选取跌幅前三名作为股票池
      Range[i=0:n-1]
      {
        zdf[i]=Close/Open-1;
        id1[i]=i;
        id2[i]=i;
      }
```

```
        SortIds(zdf,id1,id2,0,n-1,True);
        for i=0 to 2
        {
          subarray[i]=myArray[id1[i]];
        }
        LogFile("subarray="+TextArray(subarray));
        SetDicValue("001","myindustry",SystemDateTime(),subarray,True);
        LogFile("subarray write done!");
        SetTBProfileString(base1,name1,"choosedone!");
        ReStart;
      }
    }
    Else //波段交易
    {
      Range[0:n-1]
      {
        highline=highest(High[1],hllen);
        lowline=Lowest(Low[1],hllen);
        PlotNumeric("highline",highline);
        PlotNumeric("lowline",lowline);
        if(MarketPosition<>1 and High>=highline)
          Buy(200,Max(Open,highline));
        If(MarketPosition==1 and DateDiff(entrydate,Date)>0 and Low<=lowline)
          Sell(0,Min(Open,lowline));
      }
    }
}
```

运行结果：

公式	商品代码	商品简称	周期	分类	算法	起始日期	结束日期	更新时间	bar数	仓位
test	688258	卓易信息	5分钟		×	2019/12/09	2020/01/15	15:29:59	1371	200
	688006	杭可科技	5分钟		×	2019/07/22	2020/01/15	15:29:59	6008	200
	688199	久日新材	5分钟		×	2019/11/05	2020/01/15	15:29:59	2569	200

test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
[20200115.1529] init1...choosecode=N/A
[20200115.1529] myArray=
[688168.SSE,688369.SSE,688028.SSE,688033.SSE,688068.SSE,688009.SSE,688015.SSE,688181.SSE,688198
.SSE,688058.SSE,688111.SSE,688066.SSE,688078.SSE,688011.SSE,688081.SSE,688089.SSE,688100.SSE,68
8116.SSE,688003.SSE,688029.SSE,688022.SSE,688030.SSE,688001.SSE,688218.SSE,688258.SSE,688300.SS
E,688358.SSE,688399.SSE,688266.SSE,688166.SSE,688178.SSE,688037.SSE,688278.SSE,688196.SSE,68801
0.SSE,688139.SSE,688002.SSE,688021.SSE,688363.SSE,688101.SSE,688122.SSE,688333.SSE,688008.SSE,6
88099.SSE,688098.SSE,688366.SSE,688012.SSE,688016.SSE,688188.SSE,688368.SSE,688019.SSE,688018.S
SE,688202.SSE,688123.SSE,688118.SSE,688158.SSE,688199.SSE,688108.SSE,688006.SSE,688310.SSE,6882
88.SSE,688023.SSE,688298.SSE,688039.SSE,688088.SSE,688299.SSE,688005.SSE,688036.SSE,688321.SSE,
688020.SSE,688388.SSE,688007.SSE,688138.SSE,688389.SSE,688026.SSE,688025.SSE,688268.SSE,688159.
SSE,688128.SSE,688357.SSE]
[20200115.1529] subarray=[688258.SSE,688006.SSE,688199.SSE]
[20200115.1529] subarray write done!
[20200115.1529] begin...choosecode=choosedone!
[20200115.1529] myArray=[688258.SSE,688006.SSE,688199.SSE]
[20200115.152917] begin...choosecode=choosedone!
[20200115.152917] myArray=[688258.SSE,688006.SSE,688199.SSE]
[20200115.153311] begin...choosecode=choosedone!
[20200115.153311] myArray=[688258.SSE,688006.SSE,688199.SSE]
[20200115.15334] begin...choosecode=choosedone!
[20200115.15334] myArray=[688258.SSE,688006.SSE,688199.SSE]
[20200115.153341] begin...choosecode=choosedone!
```

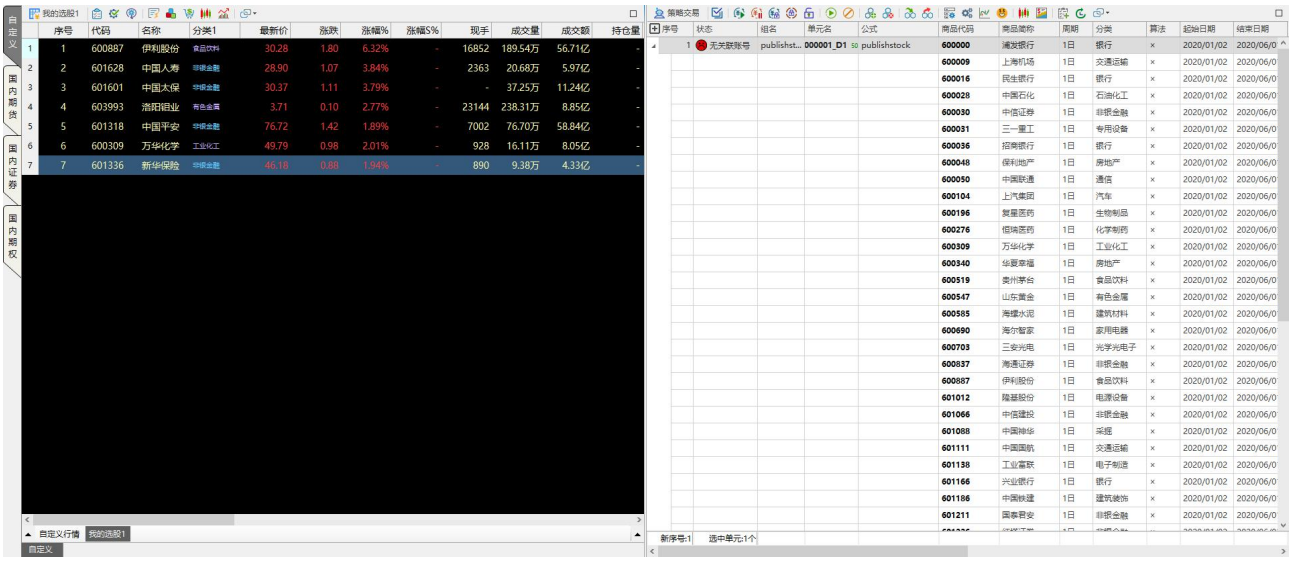
案例十三：事件发送

学习焦点：

这个案例主要是演示用户如何使用事件发送函数。在这个案例中我们使用了一个简单的四周法则来进行选股，然后利用事件发送函数将选股结果发送到自定义的行情报价。
注意：事件发送的内容存储在一个 MAP 里面。这个 MAP 需要给两个键值进行赋值，一个是 instruments 这个是用户选出的股票代码，格式以逗号进行分割。另一个是 industry，这个是行情报价的板块名称，需要指定二级和三级板块名，格式以下划线分割。

工作区设置：

工作区分为左右两个部分，左边是行情报价，右边是策略交易单元。
策略交易单元，我们数据源选择上证 50 的成分股。



公式代码：

```
Params
    Numeric length(20); //四周周期
Vars
    Map<String,String> context;   //推送自定义合约的 MAP
    Global String tmpSyms;        //自定义合约的股票
    Global Numeric senddatetime; //推送的时间
    Series<Numeric> highline;     //四周高点
Events
OnBar(ArrayRef<Integer> indexs)
{
    Range[0:DataSourceSize -1]
    {
        highline=highest(high[1],length);
        PlotNumeric("highline",highline);
```

```
        if(barstatus==2 and High>highline)   //根据四周进行选股
            tmpSyms = tmpSyms+Symbol+",";
    }
    //最后一根 BAR 推送一次
    if(BarStatus==2 and senddatetime<>Date+Time)
    {
        Commentary(tmpSyms);
        context["合约集合"]=tmpSyms;  //选股合约
        context["板块名称"]="选股板块_自定义 1";  //自定义行情设置，格式是：一级板块_二级板块
        context["添加方式"]="override";  //更新方式：override,append
        PublishEvent("系统-选股事件",context,"行情报价");  //发送选股事件到行情报价
        senddatetime=Date+time;
    }
}
```

第三部分 函数摘要

一、系统函数

1.1 数学计算

Abs：返回参数的绝对值
Acos：返回参数的反余弦值
Acosh：返回参数的反双曲余弦值
Asin：返回参数的反正弦值
Asinh：返回参数的反双曲正弦值
Atan：返回参数的反正切值
Atan2：返回给定的 X 及 Y 坐标值的反正切值
Atanh：返回参数的反双曲正切值
Ceiling：将参数 Number 沿绝对值增大的方向，舍入为最接近的整数或基数 Significance 的最小倍数
Combin：计算从给定数目的对象集合中提取若干对象的组合数
Cos：返回给定角度的余弦值
Cosh：返回参数的双曲余弦值
Ctan：返回给定角度的余切值
Cum：求累计值
Compare：字符串比较
Even：返回沿绝对值增大方向取整后最接近的偶数
Exp：返回 e 的 Number 次幂
Fact：返回数的阶乘
Floor：将参数 Number 沿绝对值减小的方向去尾舍入，使其等于最接近的 Significance 的倍数
FracPart：返回实数舍入后的小数值
IntPart：返回实数舍入后的整数值
Ln：返回一个数的自然对数

Log: 按所指定的底数，返回一个数的对数
Max: 求最大值
Min: 求最小值
Mod: 返回两数相除的余数
Neg: 返回参数的负绝对值
Odd: 返回对指定数值进行舍入后的奇数
Power: 返回给定数字的乘幂
Permutation: 求排列
Pi: 返回数字 3.1415926535898
Round: 返回某个数字按指定位数舍入后的数字
RoundDown: 靠近零值，向下（绝对值减小的方向）舍入数字
RoundUp: 远离零值，向上（绝对值增大的方向）舍入数字
Rand: 返回位于两个指定数之间的一个随机数
Sign: 返回数字的符号
Sin: 返回给定角度的正弦值
Sinh: 返回某一数字的双曲正弦值
Sqr: 返回参数的平方
Sqrt: 返回参数的正平方根
Summation: 求和
SummationFC: 快速求和
Tan: 返回给定角度的正切值
Tanh: 返回某一数字的双曲正切值

1.2 数学统计

Compare: 数值指定精度比较
FftFliter: 傅里叶滤波的预测
Fisher: 求 Fisher 变换
FisherInv: 求反 Fisher 变换
FactorStandard: 因子标准化
FactorPure: 因子提纯
LinearRegSlope: 求线性回归的斜率
LinearRegValue: 求线性回归值
MLRS: 多元线性回归的求解
Norms: 正态分布概率密度函数
Normsdist: 正态分布概率分布函数
Normsinv: 正态分布概率分布函数的反函数
QuadProg: 二次规划问题最优解

1.3 字符串函数

Exact: 该函数测试两个字符串是否完全相同
FindFirstOf: 查找字符串第一次出现的位置
FindLastOf: 查找字符串最后一次出现的位置

Left: 返回文本串的前 lCount 位
Len: 返回文本串中的字符数
Lower: 将一个文字串中的所有大写字母转换为小写字母
Mid: 返回文本串的从 lFirst 开始的 lCount 位
Right: 返回文本串的后 lCount 位
StringSplit: 分割字符串
StringReplace: 替换字符串
Section: 返回字符串包含从位置开始到位置结束的字段
Trim: 除了文本两边所有的空格
TextArray: 将一维或二维数组转化为字符串
TextMap: 将参数中的映射表转化为字符串
Text: 将数值或对象转化为字符串
Upper: 将一个文字串中的所有小写字母转换为大写字母
Value: 将代表数字的文字串转换成数字

1.4 数组函数

ArrayCopy: 复制二维数组内容
ArrayClear: 数组的全部删除
ArrayClear: 删除二维数组的全部内容
ArrayCompare: 对两个数组进行比较
ArrayCompare: 对两个二维数组进行比较
ArrayCopy: 复制数组的内容
ArraySwap: 交换二维数组的内容
ArrayEqual: 检验两个数组是否相等
ArrayEqual: 检验两个二维数组是否相等
ArrayErase: 数组的元素删除
ArrayErase: 二维数组删除行
ArrayErase: 二维数组的指定元素删除
ArrayInsert: 数组的单个数据插入
ArrayInsert: 二维数组插入单个元素
ArrayInsertRange: 数组的多个数据插入
ArrayInsertRange: 二维数组插入多个元素
ArraySort: 对数组进行排序
ArraySort: 二维数组排序
ArraySwap: 交换数组的内容
AverageArray: 求数组的平均值
AvgDeviationArray: 求数组的平均偏差
BoolArrayClear: 布尔型数组的全部删除
CoefficientRArray: 求数组的皮尔森相关系数
CorrelationArray: 求数组的相关系数
CovarArray: 求数组的协方差
DevSqrdArray: 求数组的偏差均方和
Diag: 提取矩阵主对角线元素

ExtremesArray: 求数组的极值
GetArraySize: 获取数组的大小
HarmonicMeanArray: 求数组的调和平均数
HighestArray: 求数组的最大值
KurtosisArray: 求数组的峰度系数
LinearRegAngleArray: 求数组的线性回归的角度
LinearRegArray: 求数组的线性回归
LinearRegForecastArray: 求数组的线性回归的预测值
LinearRegInterceptArray: 求数组的线性回归的截距
LinearRegSlopeArray: 求数组的线性回归的斜率
LowestArray: 求数组的最小值
MedianArray: 求数组的中位数
ModeArray: 求数组的众数
Na1Max: 一维数组的最大值
Na1Min: 一维数组的最小值
Na1Sort: 一维数组排序带下标
Na1Sort2: 一维数组排序带下标，原数组不变
Na2Abs: 二维数组的绝对值
Na2Add: 二维数组的加法
Na2Dev: 二维数组的减法
Na2Inv: 二维数组的求逆
Na2Mul: 二维数组的乘法
Na2Sum: 二维数组的元素求和
Na2Trans: 二维数组的转置
NormalCumDensityArray: 求数组的正态分布累计概率密度
NormalDensityArray: 求数组的正态分布概率密度
NthExtremesArray: 求数组的第 N 个极值
NthHigherArray: 求数组的第 N 大的值
NthLowerArray: 求数组的第 N 小的值
Na1Sqrt: 一维数组的平方根
Na1Mul: 一维数组乘法
Na1Muln: 数组与实数的乘法
Na1Add: 一维数组加法
Na1Div: 一维数组除法
Na1Dev: 一维数组减法
Na1Norm2: 求一维数组的欧几里得范数
SkewnessArray: 求数组的偏度系数
SortIds: 一维数组排序带双重下标
StandardDevArray: 求数组的标准差
SummationArray: 求数组的和
SetArraySize: 设置数组的大小和初始值
SetArraySize: 设置二维数组的大小和初始值
VariancePSArray: 求数组的估计方差
ArrayColumnErase: 二维数组删除指定列

1.5 容器函数

ArrayPushBack: 数组末尾添加元素
GetMapKeys: 获取容器映射表的所有 key 值
GetMapSize: 获取容器映射表的大小\
GetMapKey: 获取映射表中指定索引位置的关键字的值
GetMapValue: 获取映射表中指定索引位置的值
MapClear: 删除映射表的全部内容
MapContain: 判断映射表中是否存在 key 值
MapErase: 映射表的指定元素删除
MapFind: 映射表的指定元素查找
MapLowerBound: 获取映射表中第一个大于或等于 key 的元素的值或索引
MapUpperBound: 获取映射表中第一个大于 key 的元素的值或索引

1.6 期权函数

BjerkStensCall: 计算美式期权的理论价
BjerkStensPhi: 该函数被 BjerkStensCall 调用，用于计算美式期权的理论价格
BlackModel: 获取期货期权的理论价格
BlackScholes: 获取股票期权的理论价格
Delta: 获取期权的 Delta 值
Gamma: 获取期权的 Gamma 值
ImpliedVolatility: 获取股票期权的隐含波动率
Intrinsic: 获取期权的内在价值
Option_Binomialtree: 期权的二叉树定价
Option_Bsprice: 期权的 BS 定价
Option_Delta: 期权的希腊值之 DELTA
Option_Gamma: 期权的希腊值之 GAMMA
Option_Rho: 期权的希腊值之 RHO
Option_Theta: 期权的希腊值之 THETA
Option_Vega: 期权的希腊值之 VEGA
OptionPrice: 获取期权的理论价格
OptionsComplex: 计算期权的理论价格和希腊字母值
OptionStyle: 返回期权的类型，欧式还是美式
OptionType: 返回期权的类型，是看涨还是看跌期权
Option_ImVolatility: 推算期权的隐含波动率
Rho: 获取期权的 Rho 值
StrikePrice: 返回期权的行权
Theta: 获取期权的 Theta 值
Vega: 获取期权的 Vega 值
Volatility: 历史波动率

1.7 事件函数

BjerkStensCall: 计算美式期权的理论价
CreateTimer: 创建 Timer
CreateProperty: 添加合约属性
GetBar: 获取 bar 数据
GetTick: 获取 tick 数据
GetTick: 获取指定合约的 tick 数据
GetProperty: 获取 bar 上的合约属性
Getproperty (2) : 获取历史时间上的对应合约的合约属性
OnBar: Bar 数据更新驱动
OnBarOpen: 在新 bar 的第一次执行之前驱动
OnFill: 成交更新驱动
OnInit: 初始化事件
OnOrder: 委托更新驱动
OnPosition: 持仓更新驱动
OnReady: 在所有的数据源准备完成后调用，应用在数据源的设置等操作
OnTimer: 定时器更新驱动
OnSignal: 代理信号事件
OnStrategyPosition: 策略账户仓更新事件函数
OnEvent: 通用事件
OnExit: 退出事件
OnAccount: 账户资金更新驱动
OnAccountStatus: 账户状态更新驱动
PushBar: 抛数据到公式系统
PublishEvent: 发布事件
ReRun: 重新运行公式
ReStart: 重新运行交易单元
Stop: 停止交易单元
StopTimer: 停止定时器
SubscribeBar: 订阅 bar 行情
SubscribeEvent: 根据事件名称订阅事件
UnsubscribeBar: 取消订阅 bar 行情
UnsubscribeEvent: 根据事件名称退订事件
OnBarClose: 在下一根 bar 开始前重新执行当前 bar 最后一次

1.8 内嵌结构体

Account: 账户资金
AccountStatus: 账户状态
Bar: Bar 数据
CodeProperty: 合约属性
Fill: 成交
Order: 委托
OrderRequest: 报单请求
Position: 持仓

Plot: Plot - Plot 对象
Signal: 信号
Tick: 行情快照

1.9 基础数据函数

DeleteDicValue: 根据键名、关联名删除基础数据
GetDicNames: 获取基础数据键名列表
GetDicSymbols: 获取基础数据关联标的列表
GetDicTime: 获取基础数据的时间（与 Bar 关联）
GetDicTimeRange: 获取当前公式中基础数据所有时间集合
GetDicValue: 获取小于等于某个时间点的最近一笔基础数据的值
GetDicSymbolDescription: 获取基础数据标的的描述信息
SetDicValue: 动态指定键名、关联名、时间写基础数据
GetDicDescription: 获取基础数据说明
GetDicSecondKeys: 获取基础数据二级键名
GetDicTypeAsEnum: 获取基础数据类型
GetDicTypeAsString: 获取基础数据类型名称

1.10 BAR 数据函数

BarCount: 当前公式应用商品数据的 Bar 总数
BarExistStatus: 当前公式应用商品当前 Bar 的运行状态值
BarStatus: 当前公式应用商品当前 Bar 的状态值
C: 当前公式应用商品在当前 Bar 的收盘价
Close: 当前公式应用商品在当前 Bar 的收盘价
CurrentBar: 当前公式应用商品在当前 Bar 的索引值
D: 当前公式应用商品在当前 Bar 的日期
DataConvert: 跨周期数据转换函数
DataCount: 返回当前公式应用图表数据源的总数
DataSourceSize: 获取数据源的个数
Date: 当前公式应用商品在当前 Bar 的日期
EndDate: 当前公式应用商品在当前 Bar 的结束日期
EndTime: 当前公式应用商品在当前 Bar 的结束时间
H: 当前公式应用商品在当前 Bar 的最高价
High: 当前公式应用商品在当前 Bar 的最高价
HistoryDataExist: 当前公式应用商品的历史数据是否有效
L: 当前公式应用商品在当前 Bar 的最低价
Low: 当前公式应用商品在当前 Bar 的最低价
NextBarDateTime: 获取当前 bar 的下一根 bar 时间
O: 当前公式应用商品在当前 Bar 的开盘价
Open: 当前公式应用商品在当前 Bar 的开盘价
OpenInt: 当前公式应用商品在当前 Bar 的持仓量

QuoteDataExist: 当前公式应用商品的行情数据是否有效
Rollover: 当前公式应用商品在当前 Bar 的除权系数
TurnOver: 当前公式应用商品在当前 Bar 的成交金额
T: 当前公式应用商品在当前 Bar 的时间
Time: 当前公式应用商品在当前 Bar 的时间
TradingBeginDatetime: 当前公式应用商品在当前 Bar 的真实开始交易时间
TradingEndDateTime: 当前公式应用商品在当前 Bar 的真实结束交易时间
V: 当前公式应用商品在当前 Bar 的成交量
Vol: 当前公式应用商品在当前 Bar 的成交量

1.11 BAR 数据交易函数

Buy: 产生一个多头建仓操作
BuyToCover: 产生一个空头平仓操作
Sell: 产生一个多头平仓操作
SellShort: 产生一个空头建仓操作

1.12 BAR 数据策略函数

AvgBarsEvenTrade: 获得保本交易的平均持仓 Bar 数
AvgBarsLosTrade: 获得亏损交易的平均持仓 Bar 数
AvgBarsWinTrade: 获得盈利交易的平均持仓 Bar 数
AvgEntryPrice: 获得当前持仓的平均建仓价格
BarsSinceEntry: 获得当前持仓的第一个建仓位置到当前位置的 Bar 计数
BarsSinceExit: 获得最近平仓位置到当前位置的 Bar 计数
BarsSinceLastEntry: 获得当前持仓的最后一个建仓位置到当前位置的 Bar 计数
ContractProfit: 获得当前持仓位置的每手浮动盈亏
CurrentContracts: 获得当前持仓的持仓合约数
CurrentEntries: 获得当前持仓的建仓次数
EntryDate: 获得当前持仓的第一个建仓位置的日期
EntryPrice: 获得当前持仓的第一个建仓价格
EntryTime: 获得当前持仓的第一个建仓位置的时间
ExitDate: 获得最近平仓位置 Bar 日期
ExitPrice: 获得最近平仓位置的平仓价格
ExitTime: 获得最近平仓位置 Bar 时间
ExitSize: 获得平仓次数
GrossLoss: 获得累计的总亏损
GrossProfit: 获得累计的总利润
GetRunningTradeStrategySize: 获取运行中的交易公式数量
LargestLosTrade: 获得最大单次交易亏损数
LargestWinTrade: 获得最大单次交易盈利数
LastEntryDate: 获得当前持仓的最后一个建仓位置的日期
LastEntryPrice: 获得当前持仓的最后一个建仓价格
LastEntryTime: 获得当前持仓的最后一个建仓位置的时间

MarketPosition: 获得当前持仓状态
MaxConsecLosers: 获得最大连续亏损交易手数
MaxConsecWinners: 获得最大连续盈利交易手数
MaxContracts: 获得当前持仓的最大持仓合约数
MaxContractsHeld: 获得最大的持仓合约数
MaxEntries: 获得最大的建仓次数
MaxPositionLoss: 获得当前持仓的最大浮动亏损数
MaxPositionProfit: 获得当前持仓的最大浮动盈利数
NetProfit: 获得已经平仓交易的累计的净利润
NumEvenTrades: 获得保本交易的总手数
NumLosTrades: 获得亏损交易的总手数
NumWinTrades: 获得盈利交易的总手数
Portfolio_NumLossTrades: 获得投资组合的交易亏损手数
Portfolio_NumWinTrades: 获得投资组合的交易盈利手数
Portfolio_PercentProfit: 获得投资组合的交易成功率
Portfolio_PositionProfit: 获得投资组合的浮动盈亏
Portfolio_TotalProfit: 获得投资组合的累计交易盈亏
Portfolio_TotalTrades: 获得投资组合的总交易手数
Portfolio_UsedMargin: 获得当前的持仓保证金
PositionProfit: 获得当前持仓位置的浮动盈亏
PercentProfit: 获得盈利的成功率
Portfolio_CurrentEntries: 获得投资组合的当前建仓次数
Portfolio_GrossLoss: 获得投资组合的毛损
Portfolio_GrossProfit: 获得投资组合的毛利
Portfolio_MaxDrawDown: 获得投资组合的最大资产回撤
Portfolio_NetProfit: 获得投资组合的已经平仓交易的净利润
Portfolio_CurrentCapital: 获得按当前 Bar 开盘价计算的可用资金
Portfolio_CurrentEquity: 获得投资组合的动态权益
Portfolio_InitCapital: 获得投资组合的初始资金
Portfolio_MaxDrawDownRatio: 获得投资组合的最大资产回撤比率
SignalSize: 获得交易信号个数
TotalBarsEvenTrades: 获得保本交易的总持仓 Bar 数
TotalBarsLosTrades: 获得亏损交易的总持仓 Bar 数
TotalBarsWinTrades: 获得盈利交易的总持仓 Bar 数
TotalTrades: 获得交易的总手数

1.13 BAR 数据统计函数

AdaptiveMovAvg: 求卡夫曼自适应移动平均
Average: 求平均
AverageD: 求 N 天以来的价格平均值
AverageFC: 快速计算平均值
AvgDeviation: 求平均背离
AvgPrice: 求平均价格
AvgTrueRange: 求平均真实范围

BarsSinceToday: 求当天的第一个数据到当前的 Bar 数
CloseD: 求 N 天前的收盘价
CoefficientR: 求皮尔森相关系数
Correlation: 求相关系数
CountIf: 获取最近 N 周期条件满足的计数
Covar: 求协方差
CrossOver: 求是否上穿
CrossUnder: 求是否下穿
DEMA: 求双指数平均
Detrend: 求趋势平滑
DevSqrd: 求偏差均方和
DayBarsNumI: 一个交易日包含多少根 BAR 的函数
Extremes: 求极值
HarmonicMean: 求调和平均数
HighD: 求 N 天前的最高价
Highest: 求最高值
HighestBar: 求最高值出现的 Bar
HighestBarFC: 求最高值出现的 Bar(快速计算版本)
HighestFC: 求最高值(快速计算版本)
Kurtosis: 求峰度系数
LinearReg: 求线性回归
LinearRegAngle: 求线性回归的角度
LowD: 求 N 天前的最低价
Lowest: 求最低
LowestBar: 求最低值出现的 Bar
LowestBarFC: 求最低值出现的 Bar(快速计算版本)
LowestFC: 求最低(快速计算版本)
Median: 求中位数
MidPoint: 求中点
Mode: 求众数
Momentum: 求动量
NormalCumDensity: 求指定数据的正态分布累计概率密度
NormalDensity: 求指定数据的正态分布概率密度
NormalSCDensity: 求指定数据的正态分布累计概率密度
NthCon: 第 N 个满足条件的 Bar 距当前的 Bar 数目
NthExtremes: 求 N 极值
NthHigher: 求第 N 高
NthHigherBar: 求第 N 高出现的 Bar
NthLower: 求第 N 低值
NthLowerBar: 求第 N 低出现的 Bar
OpenD: 求 N 天前的开盘价
OpenIntD: 求 N 天前的持仓量
PriceOscillator: 求振荡
ParabolicSAR: 求抛物线转向

PercentChange: 求涨跌幅
PercentR: 求威廉指标
Pivot: 求转折
RateOfChange: 求变动率
SwingLowBar: 求波谷点出现的 Bar
Skewness: 求偏度系数
SMA: 求移动平均
StandardDev: 求标准差
SwingHigh: 求波峰点
SwingHighBar: 求波峰点出现的 Bar
SwingLow: 求波谷点
SAverage: 求平滑平均
TrueDate: 求指定 bar 的真正交易日期
TrueHigh: 求真实高点
TrueLow: 求真实低点
TrueRange: 求真实范围
VariancePS: 求估计方差
VolD: 求 N 天前的成交量
WAverage: 求权重平均
XAverage: 求指数平均

1.14 行情快照函数

GetReadyBar: OnReady 中获取 Bar 数据
GetReadyTick: OnReady 中获取 Tick 数据
GetWorkspaceName: 获取工作区名称
Q_AskPrice: 当前公式应用商品的最新卖盘价格
Q_AskPriceFlag: 当前公式应用商品的卖盘价格变化标志
Q_AskVol: 当前公式应用商品的最新卖盘量
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
Q_BidPriceFlag: 当前公式应用商品的买盘价格变化标志
Q_BidVol: 当前公式应用商品的最新买盘量
Q_Close: 当前公式应用商品的当日收盘价（当日未收盘则取得昨日收盘价）
Q_High: 当前公式应用商品的当日最高价
Q_HisHigh: 当前公式应用商品的历史最高价（暂时不可用）
Q_HisLow: 当前公式应用商品的历史最低价（暂时不可用）
Q_InsideVol: 当前公式应用商品的内盘
Q_Last: 当前公式应用商品的最新价
Q_LastDate: 当前公式应用商品的最新成交日期
Q_LastFlag: 当前公式应用商品的最新价变化标志
Q_LastTime: 当前公式应用商品的最新成交时间
Q_LastVol: 当前公式应用商品的现手
Q_Low: 当前公式应用商品的当日最低价

Q_LowerLimit: 当前公式应用商品的当日跌停板价
Q_Open: 当前公式应用商品的当日开盘价
Q_OpenInt: 当前公式应用商品的持仓量
Q_OpenIntFlag: 当前公式应用商品的持仓量变化标志
Q_Oscillation: 当前公式应用商品的振幅
Q_OutsideVol: 当前公式应用商品的外盘
Q_PreOpenInt: 当前公式应用商品的昨日持仓量
Q_PreSettlePrice: 当前公式应用商品的昨日结算价
Q_PriceChg: 当前公式应用商品的当日涨跌
Q_PriceChgRatio: 当前公式应用商品的当日涨跌幅
Q_TickChg: 当前公式应用商品的最新笔升跌
Q_TodayEntryVol: 当前公式应用商品的当日开仓量
Q_TodayExitVol: 当前公式应用商品的当日平仓量
Q_TotalVol: 当前公式应用商品的当日成交量
Q_TurnOver: 当前公式应用商品的成交金额
Q_UpperLimit: 当前公式应用商品的当日涨停板价
SetDataRange: 设置行情范围

1.15 属性函数

BarInterval: 当前公式应用商品数据的周期数值
BarType: 当前公式应用商品数据的周期类型值
BidAskSize: 当前公式应用商品数据的买卖盘个数
BigPointValue: 当前公式应用商品数据的一个整数点的价值
BarExist: 判断当前图层是否对齐
BaseShares: 获取合约最小交易量
CanMarketOrder: 当前公式应用商品是否支持市价委托
CanShortTrade: 当前公式应用商品是否支持空头交易
CanStopOrder: 当前公式应用商品是否支持 STOP 委托
CanTrade: 当前公式应用商品是否支持交易
Category: 当前公式应用商品的大类信息
ContractSize: 当前商品的合约大小
ContractUnit: 当前公式应用商品的每张合约包含的基本单位数量
CurrencyName: 当前公式应用商品交易的货币名称
CurrencySymbol: 当前公式应用商品交易的货币符号
DataFlag: 获取图层标志
DateTimeToBarIndex: 获取指定时间所对应的当前公式应用商品 bar 索引值
ExchangeName: 当前公式应用商品的交易所名称
ExpiredDate: 当前公式应用商品的最后交易日
ExchangeCode: 当前公式应用商品的交易所编码
FormulaName: 获得当前执行的公式名称
Frequency: 获取周期范围
FirstSignalIndex: 获取当前公式应用商品第一个信号的 Bar 索引
GetGroupName: 获取分组名称

GetIndustryBysymbol: 获取合约所处的板块
GetSessionCount: 获取交易时间段的个数
GetSessionEndTime: 获取指定交易时间段的结束时间
GetSessionStartTime: 获取指定交易时间段的起始时间
GetUnitName: 获取单元名称
GetUserID: 当前登录的用户 ID
GetLayerOrderMapSymbols: 获取图层委托映射合约
GetMarkSymbols: 获取选股合约
HasRollover: 当前 bar 是否发生除权
Hide: 隐藏当前图层 k 线和指标画线
HideKline: 隐藏当前图层 K 线，保留指标画线
IndustryName: 获取板块名称
IsAddData: 判断行情数据是否是追补的数据
IsStartBar: 是否为启动时间点
IsMarkSymbol: 判断合约是否选中
IsTradeEnabled: 是否开启了自动交易
MaxSingleTradeSize: 当前公式应用商品的单笔交易限量
MinMove: 当前公式应用商品的最小变动量
MarkSymbol: 根据图层合约选股
MarkSymbol2: 根据指定合约选股
OpenDate: 当前公式应用商品的上市日期
PriceScale: 当前公式应用商品的计数单位
RelativeSymbol: 当前公式应用商品的关联代码
Symbol: 当前公式应用商品的代码
SymbolId: 获取当前公式应用商品的内部唯一编码
SymbolName: 当前公式应用商品的名称
SymbolType: 当前公式应用商品的小类信息,例如 Cu,Ru,SR 等
SetConsecEntries: 设置最大连续建仓次数
SetMaxTrades: 设置最大交易次数
Show: 显示当前图层 k 线和指标画线
ShowKline: 显示当前图层 k 线
TradingDayLeft: 商品截至到期剩余的日期
TradingDate: 根据当前 bar 时间或指定时间，判断是哪个交易日
TradeFlag: 获取忽略自动方向
GetOrderPriceOffset: 获取委托偏移值
SetTriggerBarClose: 设置触发 OnBarClose 的时间点
StrategyMode: 当前运行模式
StrategyModeName ： 当前运行模式名称

1.16 枚举函数

Enum_1Pix: 返回 1 个像素点画线线宽
Enum_2Pix: 返回 2 个像素点画线线宽
Enum_3Pix: 返回 3 个像素点画线线宽

Enum_4Pix: 返回 4 个像素点画线线宽
Enum_5Pix: 返回 5 个像素点画线线宽
Enum_6Pix: 返回 6 个像素点画线线宽
Enum_7Pix: 返回 7 个像素点画线线宽
Enum_Trade_Source_Algo: 算法单
Enum_Trade_Source_ALL: 所有操作单
Enum_AmericanOption: 返回美式期权的枚举值
Enum_Rate_AmountPerHand: 比率类型
Enum_Trade_Source_Program: 程序单
Enum_Bar: 返回类型为柱形的画线类型
Enum_Broken: 返回破折线画线类型
Enum_Buy: 返回买卖状态的买入枚举值
Enum_CallOption: 返回看涨期权的枚举值
Enum_Canceled: 返回委托状态的已撤单枚举值
Enum_Canceling: 返回委托状态的正在撤单枚举值
Enum_Cross: 返回类型为十字形的画线类型
Enum_Dash: 返回虚线画线类型
Enum_Dash_Dot: 返回点划线画线类型
Enum_Declare: 返回委托状态的正在申报枚举值
Enum_Declared: 返回委托状态的已申报枚举值
Enum_Deleted: 返回委托状态的已废除枚举值
Enum_Dot: 返回类型为点的画线样式
Enum_Entry: 返回开平仓状态的开仓枚举值
Enum_EuropeanOption: 返回欧式期权的枚举值
Enum_Exit: 返回开平仓状态的平仓自动枚举值
Enum_ExitToday: 返回开平仓状态的平今仓枚举值
Enum_Trade_Source_Extra: 非本机单
Enum_Filled: 返回委托状态的全部成交枚举值
Enum_FillPart: 返回委托状态的部分成交枚举值
Enum_Rate_FreeOfExitToday: 比率类型
Enum_Trade_Ignore_Buy: 忽略型号类型
Enum_Trade_Ignore_Buy2Cover: 忽略型号类型
Enum_Trade_Ignore_Sell: 忽略型号类型
Enum_Trade_Ignore_SellShort: 忽略型号类型
Enum_Line: 返回类型为直线的画线样式
Enum_Trade_Source_Manual: 手工单
Enum_Data_NotGenReport: 返回图层不参与生成报告的枚举值
Enum_Rate_OnlyExit: 比率类型
Enum_Rate_PointPerHand: 比率类型
Enum_Trade_Source_Monitor: 头寸监控单
Enum_PutOption: 返回看跌期权的枚举值
Enum_RatioByFillAmount: 比率类型
Enum_Sell: 返回买卖状态的卖出枚举值
Enum_Solid: 返回实线画线类型

Enum_SPFlat: 返回选股无持仓的枚举值
Enum_SPLong: 返回选股多头持仓的枚举值
Enum_SPNone: 返回选股无信号的枚举值
Enum_SPShort: 返回选股空头持仓的枚举值
Enum_SectionCaseInsensitiveSeps: 大小写不敏感设置枚举值
Enum_SectionDefault: 默认字符分割选项设置枚举值
Enum_SectionIncludeLeadingSep: 保留开头分割符设置枚举值
Enum_SectionIncludeTrailingSep: 保留结尾分割符设置枚举值
Enum_SectionSkipEmpty: 跳过空字符设置枚举值
Enum_Trade_Source_TBPY: TPBPY 单
Enum_Trade_Source_Helper: 交易助手单
enum_categoryfunds: 返回基金的枚举值
Enum_COKOR: 返回韩元的枚举值
Enum_COCAD: 返回加拿大元的枚举值
Enum_Data_OnlyDay: 返回仅日盘的枚举值
enum_categoryfutures: 返回期货的枚举值
Enum_COJPY: 返回日元的枚举值
Enum_CategorySpotTrans: 返回现货的枚举值
Enum_Data_RolloverBackWard: 返回后复权的枚举值
Enum_COGBP: 返回英镑的枚举值
Enum_COUSA: 返回美元的枚举值
Enum_CategoryStocks: 返回股票的枚举值
Enum_CORMB: 返回人民币的枚举值
Enum_CategoryForexs: 返回外汇的枚举值
Enum_CategoryBonds: 返回债券的枚举值
Enum_COCHF: 返回瑞士法郎元的枚举值
Enum_COAUD: 返回澳大利亚元的枚举值
Enum_CategoryOptions: 返回期权的枚举值
Enum_COHK: 返回港币的枚举值
Enum_CategoryForeignFutures: 返回外盘期货的枚举值
Enum_Data_OnlyNight: 返回仅夜盘的枚举值
Enum_CategoryForeignStocks: 返回外盘股票的枚举值
Enum_COEUR: 返回殴元的枚举值
Enum_Data_AutoSwapPosition: 返回根据除权自动换仓的枚举值
Enum_Data_RolloverRealPrice: 返回除权还原真实价格的枚举值
Enum_HedgeType_Arbitrage: 返回套利的枚举值
Enum_HedgeType_Hedge: 返回套保的枚举值
Enum_HedgeType_MarketMaker: 返回市商的枚举值
Enum_HedgeType_Speculatio: 返回投机的枚举值
Enum_PriceType_BestToCancel: 返回最优五档即时成交剩余撤单(沪深)的枚举值
Enum_PriceType_BestToLimit: 返回最优五档即时成交剩余转限价(沪深)的枚举值
Enum_PriceType_FixPrice: 返回盘后固定价交易(沪)的枚举值
Enum_PriceType_Limit: 返回限价的枚举值
Enum_PriceType_Market: 返回市价的枚举值

Enum_PriceType_OpponentBest: 返回对手最优价(深)的枚举值
Enum_PriceType_OwnBest: 返回本方最优价（深）的枚举值
Enum_PriceType_TotalFilledOrCancel: 返回全成交或撤单(深)的枚举值
Enum_Data_IgnoreSwapSignalCalc: 忽略换仓信号计算标志
Enum_Signal_NotSend: 不发单标志
Enum_Signal_UnCorrectPrice: 不矫正价格标志
Enum_Compare_EqualTo: 返回等于号的枚举值
Enum_Compare_Greater: 返回大于号的枚举值
Enum_Compare_GreaterEqual: 返回大于等于号的枚举值
Enum_Compare_Less: 返回小于号的枚举值
Enum_Compare_LessEqual: 返回小于等于号的枚举值
Enum_Compare_NotEqualTo: 返回不等于号的枚举值
Enum_Offset_ByJump: 返回按跳偏移的偏移方式枚举值
Enum_Offset_ByPercent: 返回按百分比偏移的偏移方式枚举值
Enum_Data_CCData: 返回订阅 CC 数据的枚举值
Enum_Strategy_Finished: 返回公式运行结束的枚举值
Enum_Signal_History: 返回历史信号的枚举值
Enum_Data_EndTime: 返回 K 线使用结束时间的枚举值
Enum_Strategy_Error: 返回公式运行出错的枚举值
Enum_Signal_Additional: 返回补充信号枚举值
Enum_MarginBuying: 返回买卖状态的融资买入枚举值
Enum_SecurityReimbursement: 返回买卖状态的买券还券枚举值
Enum_SellingReimbursement: 返回买卖状态的卖券还款枚举值
Enum_ShortSelling: 返回买卖状态的融券卖出枚举值
Enum_Trade_Sub_All: 返回订阅所有枚举值
Enum_Trade_Sub_Realtime: 返回仅订阅实时枚举值
Enum_Trade_Sub_Today: 返回订阅当日枚举值
Enum_Data_NoQuoteData: 返回不需要任何行情数据的枚举值
Enum_Signal_SwapPosition: 返回系统换仓信号枚举值
Enum_Mode_Strategy_Chart: 返回图表交易类型枚举值
Enum_Mode_Strategy_Optimize: 返回策略研究或者参数优化类型枚举值
Enum_Mode_Strategy_Pattern: 返回模式交易类型枚举值
Enum_Mode_Strategy_Radar: 返回策略雷达类型枚举值
Enum_Mode_Strategy_Trade: 返回策略交易类型枚举值
Enum_TimeZone_Berlin: 返回柏林时区枚举值
Enum_TimeZone_Chicago: 返回芝加哥时区枚举值
Enum_TimeZone_Hong_Kong: 返回香港时区枚举值
Enum_TimeZone_Kuala_Lumpur: 返回吉隆坡时区枚举值
Enum_TimeZone_London: 返回伦敦时区枚举值
Enum_TimeZone_New_York: 返回纽约时区枚举值
Enum_TimeZone_Reykjavik: 返回世界标准时区枚举值
Enum_TimeZone_Seoul: 返回首尔时区枚举值
Enum_TimeZone_ShangHai: 返回上海时区枚举值
Enum_TimeZone_Singapore: 返回新加坡时区枚举值

Enum_TimeZone_Taipei: 返回台北时区枚举
Enum_TimeZone_Tokyo: 返回东京时区枚举值
Enum_Error_Null: 返回没有错误信息枚举值
Enum_Error_SignalIgnore: 返回忽略信号枚举值
Enum_Error_Unknown: 返回未知错误类型枚举值
Enum_DicType_Bool: 返回布尔基础数据类型枚举值
Enum_DicType_Bool2DArray: 返回布尔二维数组基础数据类型枚举值
Enum_DicType_BoolArray: 返回布尔数组基础数据类型枚举值
Enum_DicType_Integer: 返回整型基础数据类型枚举值
Enum_DicType_Integer2DArray: 返回整型二维数组基础数据类型枚举值
Enum_DicType_IntegerArray: 返回整型数组基础数据类型枚举值
Enum_DicType_Numeric: 返回数值型基础数据类型枚举值
Enum_DicType_Numeric2DArray: 返回数值型二维数组基础数据类型枚举值
Enum_DicType_NumericArray: 返回数值型数组基础数据类型枚举值
Enum_DicType_String: 返回字符串基础数据类型枚举值
Enum_DicType_String2DArray: 返回字符串二维数组基础数据类型枚举值
Enum_DicType_StringArray: 返回字符串数组基础数据类型枚举值
Enum_DicType_Unknown: 返回未知基础数据类型枚举值
Enum_Error_AccountInitialzing: 返回账户正在初始化枚举值
Enum_Error_AutoTradeDisabled: 返回自动交易未开启枚举值
Enum_Error_InvalidAccount: 返回账户错误枚举值
Enum_Error_InvalidParam: 返回无效参数枚举值
Enum_Error_NetError: 返回网络错误枚举值
Enum_Error_NotExist: 返回不存在枚举值

1.17 颜色函数

Black: 返回黑色的 RGB 值
Blue: 返回蓝色的 RGB 值
Cyan: 返回青色的 RGB 值。
DarkBrown: 返回茶色的 RGB 值
DarkCyan: 返回深青色的 RGB 值
DarkGray: 返回深灰色的 RGB 值
DarkGreen: 返回深绿色的 RGB 值
DarkMagenta: 返回深褐色的 RGB 值
DarkRed: 返回深红色的 RGB 值
DefaultColor: 返回默认颜色值
Green: 返回绿色的 RGB 值
LightGray: 返回浅灰色的 RGB 值
Magenta: 返回紫红色的 RGB 值
Red: 返回红色的 RGB 值
Rgb: 返回自定义颜色值
White: 返回白色的 RGB 值

Yellow: 返回黄色的 RGB 值

1.18 时间函数

Local2UTC: 本地时区时间转成 UTC 时间
BeginDateTime: 获取图层设置的开始时间
CurrentDate: 获取交易开拓者平台的当前日期
CurrentTime: 获取交易开拓者平台(操作系统)的当前时间
DateAdd: 返回已添加指定天数的日期
DateDiff: 返回两个日期之间的天数间隔
DateTimeAdd: 在指定日期上累加指定秒数
DateTimeDiff: 返回两个时间之间的间隔秒数
DateTimeToString: 将日期时间值转化为字符串类型
DateToString: 将日期值转化为字符串类型
Day: 获得当前 Bar 的日信息
DayFromDateTime: 获取输入日期时间的日信息
EndDateTime: 获取图层设置的结束时间
Friday: 获得星期五的值
Hour: 获得当前 Bar 的小时信息
HourFromDateTime: 获取输入日期时间的小时信息
IsTradingTime: 判断指定时间是否交易时间
IsTradingTime: 判断合约在指定时间是否可以交易
MakeDate: 将参数生成日期值
MakeDateTime: 将参数生成日期时间值
MakeTime: 将参数生成时间值
MilliSecond: 获得当前 Bar 的毫秒信息
MilliSecondFromDateTime: 获取输入日期时间的毫秒信息
Minute: 获得当前 Bar 的分钟信息
MinuteFromDateTime: 获取输入日期时间的分钟信息
Monday: 获得星期一的值
Month: 获得当前 Bar 的月信息
MonthFromDateTime: 获取输入日期时间的月信息
NextTradingDateTime: 获取某个时间最近的交易时间
SystemDateTime: 获取交易开拓者平台的当前日期时间
StringToDate: 将字符串转化为日期
StringToDateTime: 将字符串转化为日期时间
StringToTime: 将字符串转化为时间
Sunday: 获得星期日的值
Saturday: 获得星期六的值
Second: 获得当前 Bar 的秒信息
SecondFromDateTime: 获取输入日期时间的秒信息
TimeToString: 将时间值转化为字符串类型
TradingTotalSecs: 获取两时间点之间的可交易秒数
Tuesday: 获得星期二的值

Thursday: 获得星期四的值
TimeDiff: 返回两个时间之间的间隔秒数，忽略日期差异
TradingOpenDateTime: 获取当前 bar 或指定合约输入时间的交易开盘时间点
UTC2Local: UTC 时间转成本地时区时间
Wednesday: 获得星期三的值
WeekdayFromDateTime: 获取输入日期时间的周信息
Weekday: 获得当前 Bar 的周信息
Year: 获得当前 Bar 的年信息
YearFromDateTime: 获取输入日期时间的年信息
DateTime2Utc: 时区日期时间转 UTC 时间
IsTradingDay: 判断是否是交易日
NextTradingDay: 获取下一个交易日
NextTradingDay: 获取下一个交易日
PreTradingDay: 获取上一个交易日
Utc2DateTime: UTC 时间转时区日期时间
TimeZone2String: 时区枚举转字符串

1.19 输出函数

Commentary: 在超级图表当前 Bar 添加一行注释信息
GetPlotNumericValue: 根据公式名、指标名获取当前公式应用商品当前 bar 上的数值型指标值
Print: 在超级图表控制台中输出当前 Bar 的一行注释信息
PlotAuto: PlotNumeric 函数的扩展函数
PlotBool: 在当前 Bar 输出一个布尔值
PlotDic: 在当前 Bar 的输出信息中添加基础数据信息
PlotKline: 在当前 Bar 输出一个 K 线
PlotNumeric: 在当前 Bar 输出一个数值
PlotString: 在当前 Bar 输出一个字符串
PlotAutoPlotBool: PlotBool 函数的扩展函数
PlotAutoPlotString: PlotAuto 函数的扩展函数
Print2Quote: 输出到量化看盘
Print2Optimize: 添加优化目标的名称和值
Unplot: 删除曾经输出的值

1.20 文件数据库

FileAppend: 在指定文件中追加一行字符串
FileDelete: 删除指定文件
GetGlobalVar: 获取某个索引的全局变量值。暂时只支持存储数字，可自己通过 0 和 1 来表示 bool 型的 true 和 false
GetGlobalVar2: 获取某个字符串索引的全局变量值。暂时只支持存储数字，可自己通过 0 和 1 来表示 bool 型的 true 和 false
GetTBProfileString: 读取公式信息文件指定块中的键名对应的字符串
GetTBProfileString2File: 读取公式信息文件指定块中的键名对应的字符串

gValue: 某个索引的全局变量，暂时只支持存储数值，SetGlobalVar 和 GetGlobalVar 简化版本
PlayWavSound: 播放指定路径的 Wav 声音文件
ReadCSVFile: 读取指定的 csv 文件
SetTBProfileString: 把给定的键名及其值写入到公式信息文件的相应块中
SetTBProfileString2File: 把给定的键名及其值写入到公式信息文件的相应块中
SetGlobalVar: 设置某个索引的全局变量值。暂时只支持存储数字，可自己通过 0 和 1 来表示 bool 型的 true 和 false
SetGlobalVar2: 设置某个字符串索引的全局变量值。暂时只支持存储数字，可自己通过 0 和 1 来表示 bool 型的 true 和 false

1.21 策略单元设置

AddDataFlag: 设置图层标志
AddTradeFlag: 忽略自动
AddStrategyFlag: 设置公式标识
ClearDataFlag: 清除图层标志
ClearTradeFlag: 清除忽略自动
InitialMargin: 当前公式应用商品的初始保证金
MaintenanceMargin: 当前公式应用商品的维持保证金
MarginRatio: 当前公式应用商品的默认保证金比率
MaxBarsBack: 获得当前公式应用所需的最大回溯 Bar 数
SetOrderPriceOffset: 按叫买叫卖价委托偏移
SetSlippage: 设置滑点
SetBackBarMaxCount: 设置最大回溯 bar 数
SetBeginBarMaxCount: 设置最大起始 bar 数
SetCommissionRate: 设置手续费率
SetMarginRate: 设置保证金率
SetOrderMap2AppointedSymbol: 委托映射到指定合约
SetOrderMap2MainSymbol: 委托映射到主力
SetInitCapital: 设置初始资金

1.22 账户函数

A_AccountCount: 返回当前账号总数
A_AccountID: 返回当前公式应用的交易帐户 ID
A_AccountIDs: 返回当前公式应用的所有交易账户 ID
A_AccountIndex: 账户下标
A_BrokerID: 返回当前公式应用的交易帐户对应的交易商 ID
A_BrokerIDs: 返回当前公式应用的所有交易账户的交易商 ID
A_BrokerName: 返回当前交易账户的交易商名称
A_BuyAvgPrice: 返回当前公式应用的帐户下当前商品按结算价计算的买入持仓均价
A_BuyPosition: 返回当前公式应用的帐户下当前商品的买入持仓
A_BuyProfitLoss: 返回当前公式应用的帐户下当前商品按结算价计算的买入持仓盈亏
A_CurrentEquity: 返回当前公式应用的交易帐户的动态权益

A_DeleteAccountOrder: 撤账户的所有未成交单
A_DeleteOrder: 针对当前公式应用的帐户、商品发送撤单指令
A_DeleteOrderEx: 撤指定报单索引的委托单
A_FreeMargin: 返回当前公式应用的交易帐户的可用资金
A_DeleteAccountOrder: 撤账户指定合约的所有未成交单
A_GetUnFillOrderIDs: 获取指定合约未完成的报单数组
A_GetFill: 获取指定报单的成交信息
A_GetFillCount: 返回指定报单的成交笔数
A_GetLastOpenOrderIndex: 返回当前公式应用的帐户下当前商品的最后一个未成交委托单索引，按输入参数为条件
A_GetLastOrderIndex: 返回当前公式应用的帐户下当前商品的最后一个当日委托单索引，按输入参数为条件
A_GetOpenOrderCount: 返回当前公式应用的帐户下当前商品的未成交委托单数量
A_GetOrder: 获取报单的详细信息
A_GetOrderCount: 返回当前公式应用的帐户下当前商品的当日委托单数量
A_GetPosition: 获取当前仓位
A_GetPosition: 获取指定合约的仓位
A_GetUnFillOrderIDs: 获取未完成的报单数组
A_OpenOrderBuyOrSell: 返回当前公式应用的帐户下当前商品的某个未成交委托单的买卖类型
A_OpenOrderComment: 根据索引取委托注释，只针对未成交单的索引
A_OpenOrderContractNo: 返回当前公式应用的帐户下当前商品的某个未成交的委托单的合同号。(本函数范围是所有未成交的委托单的合同号,有区别于 A_OrderContractNo)
A_OpenOrderEntryOrExit: 返回当前公式应用的帐户下当前商品的某个未成交委托单的开平仓状态
A_OpenOrderFilledLot: 返回当前公式应用的帐户下当前商品的某个未成交委托单的成交数
A_OpenOrderFilledPrice: 返回当前公式应用的帐户下当前商品的某个未成交委托单的成交价格
A_OpenOrderLot: 返回当前公式应用的帐户下当前商品的某个未成交委托单的委托数量
A_OpenOrderPrice: 返回当前公式应用的帐户下当前商品的某个未成交委托单的委托价格
A_OpenOrderStatus: 返回当前公式应用的帐户下当前商品的某个未成交委托单的状态
A_OpenOrderTime: 返回当前公式应用的帐户下当前商品的某个未成交委托单的委托时间
A_OrderBuyOrSell: 返回当前公式应用的帐户下当前商品的某个委托单的买卖类型
A_OrderCanceledLot: 返回当前公式应用的帐户下当前商品的某个委托单的撤单数量
A_OrderComment: 根据索引取委托注释，只针对当日委托单的索引
A_OrderContractNo: 返回当前公式应用的帐户下当前商品的某个委托单的合同号(本函数范围是所有的委托单的合同号,有区别于 A_OpenOrderContractNo)
A_OrderEntryOrExit: 返回当前公式应用的帐户下当前商品的某个委托单的开平仓状态
A_OrderFilledLot: 返回当前公式应用的帐户下当前商品的某个委托单的成交数量
A_OrderFilledPrice: 返回当前公式应用的帐户下当前商品的某个委托单的成交价格
A_OrderLot: 返回当前公式应用的帐户下当前商品的某个委托单的委托数量
A_OrderPrice: 返回当前公式应用的帐户下当前商品的某个委托单的委托价格
A_OrderStatus: 返回当前公式应用的帐户下当前商品的某个委托单的状态
A_OrderTime: 返回当前公式应用的帐户下当前商品的某个委托单的委托时间
A_PositionProfitLoss: 返回当前公式应用的帐户下当前商品按结算价计算的持仓盈亏
A_PreviousEquity: 返回当前公式应用的交易帐户的昨日结存
A_ProfitLoss: 返回当前公式应用的交易帐户按结算价计算的浮动盈亏

A_SellAvgPrice: 返回当前公式应用的帐户下当前商品按结算价计算的卖出持仓均价
A_SellPosition: 返回当前公式应用的帐户下当前商品的卖出持仓
A_SellProfitLoss: 返回当前公式应用的帐户下当前商品按结算价计算的卖出持仓盈亏
A_SendOrder: 针对当前公式应用的帐户、商品发送委托单
A_SubscribeTradeByCreateId: 根据操作源 ID 订阅委托
A_TodayBuyPosition: 返回当前公式应用的帐户下当前商品的当日买入持仓
A_TodayDeposit: 返回当前公式应用的交易帐户的当日入金
A_TodayDrawing: 返回当前公式应用的交易帐户的当日出金
A_TodaySellPosition: 返回当前公式应用的帐户下当前商品的当日卖出持仓
A_TotalAvgPrice: 返回当前公式应用的帐户下当前商品按结算价计算的持仓均价
A_TotalFreeze: 返回当前公式应用的交易帐户的冻结资金
A_TotalMargin: 返回当前公式应用的交易帐户的保证金
A_TotalPosition: 返回当前公式应用的帐户下当前商品的总持仓
AccountDataExist: 当前公式应用商品的帐户数据是否有效
A_AccountDataExist: 当前公式应用商品的帐户数据是否有效
A_GetOrderCreateSource: 获取报单源名称
A_ProfitLoss0: 返回当前公式应用的交易帐户按成本价计算的浮动盈亏
A_SellAvgPrice0: 返回当前公式应用的帐户下当前商品按成本价计算的卖出持仓均价
A_SellProfitLoss0: 返回当前公式应用的帐户下当前商品按成本价计算的卖出持仓盈亏
A_TotalAvgPrice0: 返回当前公式应用的帐户下当前商品按成本价计算的持仓均价
A_BuyAvgPrice0: 返回当前公式应用的帐户下当前商品按成本价计算的买入持仓均价
A_BuyProfitLoss0: 返回当前公式应用的帐户下当前商品按成本价计算的买入持仓盈亏
A_PositionProfitLoss0: 返回当前公式应用的帐户下当前商品按成本价计算的持仓盈亏

1.23 TBQuant 账户函数

A_Buy: 指定账户指定合约开多头单
A_BuyToCover: 指定账户指定合约平空头单
A_GetOrderIDs: 获取报单数组。
A_GetOrderIDs2: 获取指定合约报单数组
A_Sell: 指定账户指定合约平多头单
A_SellShort: 指定账户指定合约开空头单
A_SendOrderEx: 账户下单操作
A_GetPositionSymbols: 获取账户持仓的合约代码
A_GetPreOrders: 获取指定合约指定操作源上的历史委托
A_GetPreFills: 获取指定合约指定操作源上的历史成交
A_GetSignalNetPosition: 查询账户信号净仓
A_SubscribeTradeByCreateSource: 根据操作源订阅委托
A_BindTradeAccount: 绑定交易账号
A_CanMarginBuySize: 最大可融资买入数量
A_CanShortSellSize: 最大可融券卖出数量
A_GetAccountRate: 获取账户倍数
A_SetAccountRate: 设置账户倍数
A_SetSubscribeTradeRange: 设置交易订阅范围

Fill: 成交
GetOrderMapRelatedSymbols: 获取委托映射合约
Order: 委托
Position: 持仓
A_DeleteOrderV2: 撤指定报单索引的委托单
A_SendOrderV2: 账户下单操作
A_GetAccount: 获取指定账户资金信息

1.24 其他函数

Alert: 产生一个报警动作
AlertEnabled: 返回当前公式应用的报警设置
BitAnd: 位与操
BitOr: 位或操作
BitHas: 位操作，判断 Flag 在 BitSrc 中是否存在
GetFormulaNames: 获取已加载公式的名称
IIF: 执行真假值判断，根据逻辑测试的真假值返回不同的数值
IIFString: 执行真假值判断，根据逻辑测试的真假值返回不同的字符串
InvalidInteger: 返回整型的无效值
InvalidNumeric: 返回数值型的无效值
InvalidString: 字符串的无效值
SetPremiseFormulas: 设置当前公式依赖其他公式信息

二、Python 函数

2.1 对象类型

Bar: Bar 数据
exchange: 交易所代码
frequency: 周期
Fill: 成交
Instrument: 合约
order: 委托
Position: 持仓
symbol: 合约代码
Tick: 行情快照
underlying_symbol: 品种代码

2.2 行情数据

get_history_n: 查询历史行情最新 N 条
get_history: 查询历史行情
get_current_tick: 查询实时行情快照
get_instrument: 查询合约

get_main_instrument: 查询期货主力合约

2.3 基础数据

delete_fundamental: 删除基础数据
get_fundamental: 查询基础数据
set_symbol_description: 设置关联信息描述
set_name_description: 设置关键字描述
write_fundamental: 写入基础数据

2.4 事件数据

on_init: 初始化驱动
on_tick: 数据驱动
on_bar: 数据驱动
on_position: 持仓数据驱动
on_order: 委托数据驱动
on_fill: 成交数据驱动
on_timer: 定时器数据驱动

2.5 交易数据

cancel_order: 撤单
get_account: 查询账户
get_position: 查询持仓
get_order: 查询委托
get_notrade_order: 查询未成交委托
get_fill: 查询成交
get_status: 查询账户状态
send_order: 下单

2.6 其它

exe: 运行策略
exit: 终止策略运行
get_last_err: 查询最近一次错误信息
init: 初始化

2.7 Context

创建与关闭定时器: 创建与关闭定时器
订阅与退订 Tick 数据: 订阅与退订 Tick 数据
订阅与退订 Bar 数据: 订阅与退订 Bar 数据
订阅 Account 数据: 订阅 Account 数据

说明：说明

第四部分 基础数据摘要

一、系统-板块

INDUSTRY	INDUSTRY
F102001	中金所
F1020010001	全部
F1020010002	主力
F1020010003	指数
F1020010004	连续
F1020010005	中证 500 股指(IC)
F1020010006	股指(IF)
F1020010007	上证 50 股指(IH)
F1020010008	10 年期国债(T)
F1020010009	5 年期国债(TF)
F1020010010	2 年期国债(TS)
F102002	上期所
F1020020001	全部
F1020020002	主力
F1020020003	指数
F1020020004	连续
F1020020005	橡胶(ru)
F1020020006	纸浆(sp)
F1020020007	热轧卷板(hc)
F1020020008	螺纹钢(rb)
F1020020009	线材(wr)
F1020020010	不锈钢(ss)
F1020020011	沥青(bu)
F1020020012	燃油(fu)
F1020020013	沪铝(al)
F1020020014	沪铜(cu)
F1020020015	沪镍(ni)
F1020020016	沪铅(pb)
F1020020017	沪锡(sn)
F1020020018	沪锌(zn)
F1020020019	白银(ag)
F1020020020	黄金(au)
F102003	大商所

F1020030001	全部
F1020030002	主力
F1020030003	指数
F1020030004	连续
F1020030005	胶合板(bb)
F1020030006	纤维板(fb)
F1020030007	鸡蛋(jd)
F1020030008	铁矿石(i)
F1020030009	焦炭(j)
F1020030010	焦煤(jm)
F1020030011	塑料(l)
F1020030012	聚丙烯(pp)
F1020030013	乙二醇(eg)
F1020030014	PVC(v)
F1020030015	苯乙烯(eb)
F1020030016	豆一(a)
F1020030017	豆二(b)
F1020030018	玉米(c)
F1020030019	玉米淀粉(cs)
F1020030020	豆粕(m)
F1020030021	棕榈油(p)
F1020030022	豆油(y)
F1020030023	粳米(rr)
F102004	郑商所
F1020040001	全部
F1020040002	主力
F1020040003	指数
F1020040004	连续
F1020040005	棉花(CF)
F1020040006	玻璃(FG)
F1020040007	白糖(SR)
F1020040008	棉纱(CY)
F1020040009	硅铁(SF)
F1020040010	锰硅(SM)
F1020040011	动力煤(ZC)
F1020040012	甲醇(MA)
F1020040013	PTA(TA)
F1020040014	尿素(UR)
F1020040015	粳稻(JR)
F1020040016	晚籼稻(LR)
F1020040017	菜籽油(OI)

F1020040018	普麦(PM)
F1020040019	稻谷(RI)
F1020040020	菜籽粕(RM)
F1020040021	油菜籽(RS)
F1020040022	强麦(WH)
F1020040023	苹果(AP)
F1020040024	红枣(CJ)
F1020040025	纯碱(SA)
F102005	上海能源
F1020050001	全部
F1020050002	主力
F1020050003	指数
F1020050004	连续
F1020050005	20 号胶(nr)
F1020050006	原油(sc)
F102007	主力
F1020070001	全部
F1020070002	股指
F1020070003	有色金属
F1020070004	软商品
F1020070005	黑色相关
F1020070006	贵金属
F1020070007	能源化工
F1020070008	农产品
F1020070009	债券
F102008	指数
F1020080001	全部
F1020080002	股指
F1020080003	有色金属
F1020080004	软商品
F1020080005	黑色相关
F1020080006	贵金属
F1020080007	能源化工
F1020080008	农产品
F1020080009	债券
F102009	连续
F1020090001	全部
F1020090002	股指
F1020090003	有色金属
F1020090004	软商品
F1020090005	黑色相关

F1020090006	贵金属
F1020090007	能源化工
F1020090008	农产品
F1020090009	债券
F102010	期货分类
F1020100001	股指
F1020100002	有色金属
F1020100003	软商品
F1020100004	黑色相关
F1020100005	贵金属
F1020100006	能源化工
F1020100007	农产品
F1020100008	债券
F101	国内证券
F101001	上交所
F1010010001	A 股
F1010010002	B 股
F1010010003	指数
F1010010004	债券
F1010010005	基金
F1010010006	科创板
F101002	深交所
F1010020001	A 股
F1010020002	B 股
F1010020003	指数
F1010020004	债券
F1010020005	基金
F1010020006	创业板
F101003	沪深 A 股
F1010030000	全部
F1010030100	采掘
F1010030200	传媒
F1010030310	电机
F1010030320	电气自动化设备
F1010030330	电源设备
F1010030340	高低压设备
F1010030410	元件
F1010030420	光学光电子
F1010030430	其他电子
F1010030440	半导体
F1010030450	电子制造

F1010030500	房地产
F1010030600	纺织服装
F1010030700	非银金融
F1010030800	钢铁
F1010030900	公用事业
F1010031000	国防军工
F1010031111	化肥农药
F1010031112	工业化工
F1010031113	日用化工
F1010031114	其他化学制品
F1010031120	化学原料
F1010031130	化学纤维
F1010031140	塑料
F1010031150	橡胶
F1010031160	石油化工
F1010031210	专用设备
F1010031220	仪器仪表
F1010031230	运输设备
F1010031240	通用机械
F1010031250	金属制品
F1010031311	IT 服务
F1010031312	软件开发
F1010031320	计算机设备
F1010031400	家用电器
F1010031500	建筑材料
F1010031600	建筑装饰
F1010031700	交通运输
F1010031800	农林牧渔
F1010031900	汽车
F1010032000	轻工制造
F1010032100	商业贸易
F1010032200	食品饮料
F1010032300	通信
F1010032400	休闲服务
F1010032510	中药
F1010032520	化学制药
F1010032530	医疗器械
F1010032540	医疗服务
F1010032550	医药商业
F1010032560	生物制品
F1010032600	银行

F1010032700	有色金属
F1010032800	综合
F101004	沪深 B 股
F1010040000	全部
F1010040001	上海 B 股
F1010040002	深圳 B 股
F101010	沪深指数
F1010100000	全部
F1010100001	上海指数
F1010100002	深圳指数
F1010100003	行业指数
F101005	沪深基金
F1010050001	ETF
F1010050002	LOF
F1010050003	分级基金
F1010050004	封闭式基金
F1010050005	交易型货币基金
F1010050006	T+0 货币基金
F101006	沪深债券
F1010060001	国债
F1010060002	地方债
F1010060003	公司债
F1010060004	企业债
F1010060005	可转债
F1010060006	金融债
F1010060007	可交换债
F1010060008	资产支持证券
F1010060009	政府支持机构债
F101007	指数板块
F1010070001	上证 50
F1010070002	沪深 300
F1010070003	中证 500
F1010070004	沪深 300 历史
F101008	行业板块
F1010080100	采掘
F1010080200	传媒
F1010080310	电机
F1010080320	电气自动化设备
F1010080330	电源设备
F1010080340	高低压设备
F1010080410	元件

F1010080420	光学光电子
F1010080430	其他电子
F1010080440	半导体
F1010080450	电子制造
F1010080500	房地产
F1010080600	纺织服装
F1010080700	非银金融
F1010080800	钢铁
F1010080900	公用事业
F1010081000	国防军工
F1010081111	化肥农药
F1010081112	工业化工
F1010081113	日用化工
F1010081114	其他化学制品
F1010081120	化学原料
F1010081130	化学纤维
F1010081140	塑料
F1010081150	橡胶
F1010081160	石油化工
F1010081210	专用设备
F1010081220	仪器仪表
F1010081230	运输设备
F1010081240	通用机械
F1010081250	金属制品
F1010081311	IT 服务
F1010081312	软件开发
F1010081320	计算机设备
F1010081400	家用电器
F1010081500	建筑材料
F1010081600	建筑装饰
F1010081700	交通运输
F1010081800	农林牧渔
F1010081900	汽车
F1010082000	轻工制造
F1010082100	商业贸易
F1010082200	食品饮料
F1010082300	通信
F1010082400	休闲服务
F1010082510	中药
F1010082520	化学制药
F1010082530	医疗器械

F1010082540	医疗服务
F1010082550	医药商业
F1010082560	生物制品
F1010082600	银行
F1010082700	有色金属
F1010082800	综合
F101009	地域板块
F1010090100	上海
F1010090200	江苏
F1010090300	浙江
F1010090400	安徽
F1010090500	福建
F1010090600	江西
F1010090700	山东
F1010090800	广东
F1010090900	广西
F1010091000	海南
F1010091100	重庆
F1010091200	四川
F1010091300	贵州
F1010091400	云南
F1010091500	西藏
F1010091600	陕西
F1010091700	甘肃
F1010091800	青海
F1010091900	宁夏
F1010092000	新疆
F1010092100	北京
F1010092200	天津
F1010092300	河北
F1010092400	山西
F1010092500	内蒙古
F1010092600	辽宁
F1010092700	吉林
F1010092800	黑龙江
F1010092900	河南
F1010093000	湖北
F1010093100	湖南
F103	国内期权
F103005	上交所
F1030050000	期权

F103006	深交所
F1030060000	期权
F103007	中金所
F1030070000	期权
F103001	上期所
F1030010000	期权
F103002	大商所
F1030020000	期权
F103003	郑商所
F1030030000	期权
F103004	期权分类
F1030040001	股指期权
F1030040002	有色金属
F1030040003	软商品
F1030040004	黑色相关
F1030040005	贵金属
F1030040006	能源化工
F1030040007	农产品
F1030040009	ETF 期权

二、系统-行情报价

name	TB_QUOTE_capital	TB_QUOTE_finance
description	行情报价股本	行情报价财务
type	double_1d	double_1d
data0	发布时间	发布时间
data1	上市日期	每股收益
data2	总股本	每股净资产
data3	流通股本	加权净资产收益率
data4		营业总收入
data5		营业总收入同比
data6		营业利润
data7		非经营性净收益
data8		利润总额
data9		净利润
data10		归母净利润
data11		归母净利润同比
data12		未分配利润
data13		每股未分配利润
data14		销售毛利润

data15		总资产
data16		流动资产
data17		固定资产
data18		无形资产
data19		总负债
data20		流动负债
data21		非流动负债
data22		资产负债比率
data23		股东权益
data24		股东权益比
data25		每股公积金
data26		归母净利润报告期

三、系统-合约属性

name	TB_CodeProperty_all_ForeignFutures	TB_CodeProperty_all_ForeignStocks	TB_CodeProperty_all_Futures	TB_CodeProperty_all_Options	TB_CodeProperty_all_Stocks
description	外盘期货合约属性	外盘股票合约属性	期货合约属性	期权合约属性	股票合约属性
type	string_1d	string_1d	string_1d	string_1d	string_1d
data0	合约编码	合约编码	合约编码	合约编码	合约编码
data1	商品编码	商品编码	商品编码	商品编码	商品编码
data2	交易所编码	交易所编码	交易所编码	交易所编码	交易所编码
data3	商品代码	商品代码	商品代码	商品代码	商品代码
data4	商品名称_GBK	商品名称_GBK	商品名称_GBK	商品名称_GBK	商品名称_GBK
data5	商品详称_GBK	商品详称_GBK	商品详称_GBK	商品详称_GBK	商品详称_GBK
data6	交易时段	交易时段	交易时段	交易时段	交易时段
data7	交易日偏移	交易日偏移	交易日偏移	交易日偏移	交易日偏移
data8	币值编码	币值编码	币值编码	币值编码	币值编码
data9	价格单位	价格单位	价格单位	价格单位	价格单位
data10	小数位数	小数位数	小数位数	小数位数	小数位数
data11	最小变动单位	最小变动单位	最小变动单位	最小变动单位	最小变动单位
data12	合约乘数	合约乘数	合约乘数	合约乘数	合约乘数
data13	商品单位	商品单位	商品单位	商品单位	商品单位
data14	所在地区	所在地区	所在地区	所在地区	所在地区
data15	交易类型	交易类型	交易类型	交易类型	交易类型
data16	交易特性	交易特性	交易特性	交易特性	交易特性
data17	保证金比例	保证金比例	保证金比例	保证金比例	保证金比例
data18	合约年份	合约年份	合约年份	合约年份	合约年份
data19	合约月份	合约月份	合约月份	合约月份	合约月份

data20	到期日	到期日	到期日	到期日	到期日
data21	保证金计算方式	保证金计算方式	保证金计算方式	保证金计算方式	保证金计算方式
data22	期权类型	期权类型	期权类型	期权类型	期权类型
data23	欧美式期权	欧美式期权	欧美式期权	欧美式期权	欧美式期权
data24	期权执行价	期权执行价	期权执行价	期权执行价	期权执行价
data25	创建日期	创建日期	创建日期	创建日期	创建日期
data26	上市日期	上市日期	上市日期	上市日期	上市日期
data27	交割开始时间	交割开始时间	交割开始时间	交割开始时间	交割开始时间
data28	交割结束时间	交割结束时间	交割结束时间	交割结束时间	交割结束时间
data29	交易最小限量	交易最小限量	交易最小限量	交易最小限量	交易最小限量
data30	交易最小单位	交易最小单位	交易最小单位	交易最小单位	交易最小单位
data31	商品品种	商品品种	商品品种	商品品种	商品品种
data32	最小交易增量	最小交易增量	最小交易增量	最小交易增量	最小交易增量
data33	一手量	一手量	一手量	一手量	一手量
data34	预留字段 1	预留字段 1	预留字段 1	预留字段 1	预留字段 1

四、除权换月

name	TB_ROLLOVER	TB_XDXR
description	换月	除权除息
type	string_1d	double_1d
data0	期货换月合约	发布时间
data1	期货换月前价格	每股送股
data2	期货换月后价格	配股比例
data3		配股价格
data4		每股派现金
data5		股票除权前价格

五、财务分析

name	TB_FINANALYSIS_CAPITALSTRUCTURE	TB_FINANALYSIS_CASH	TB_FINANALYSIS_DIVIDENDS	TB_FINANALYSIS_DUPONT	TB_FINANALYSIS_EARNINGSQUALITY	TB_FINANALYSIS_GROWTH	TB_FINANALYSIS_OPERATION	TB_FINANALYSIS_PERSHARE	TB_FINANALYSIS_REVENUEQUALITY	TB_FINANALYSIS_SOLVENCY
description	资本结构	现金流量	现金分红	杜邦分析	盈利质量	成长能力	营运能力	每股指标	收益质量	偿付能力
type	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d

data0	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间
data1	资产负债率(%)	销售商品提供劳务收到的现金/营业收入(%)	每股现金及现金等价物余额(元/股)	权益乘数_杜邦分析(%)	净资产收益率_平均_计算值(%)	基本每股收益同比增长(%)	流动比率	基本每股收益(元/股)	经营活动净收益/利润总额(%)	产权比率(%)
data2	流动资产/总资产(%)	销售商品提供劳务收到的现金/营业收入_TTM(%)	每股股利(元/股)	归属母公司股东的净利润/净利润(%)	净资产收益率_加权_公布值(%)	稀释每股收益同比增长(%)	速动比率	稀释每股收益(元/股)	经营活动净收益/利润总额_TTM(%)	归属母公司股东的权益/负债合计(%)
data3	非流动资产/总资产(%)	经营活动产生的现金流量净额/营业收入(%)	股利保障倍数(倍)	净利润/营业总收入(%)	净资产收益率_摊薄_公布值(%)	营业收入同比增长(%)	超速动比率	每股收益_期末股本摊薄(元/股)	对联营合营公司投资收益/利润总额(%)	归属母公司股东的权益/带息债务(%)
data4	固定资产比率(%)	经营活动产生的现金流量净额/营业收入_TTM(%)	现金股利保障倍数(倍)	净利润/利润总额(%)	净资产收益率_扣除_摊薄(%)	营业利润同比增长(%)	营业周期(天/次)	每股收益_TTM(元/股)	对联营合营公司投资收益/利润总额_TTM(%)	有形净值债务率(%)
data5	无形资产比率(%)	经营活动产生的现金流量净额/经营活动净收益(%)	股利支付率(%)	利润总额/息税前利润(%)	净资产收益率_扣除_加权(%)	利润总额同比增长(%)	存货周转率(次)	每股净资产(元/股)	价值变动净收益/利润总额(%)	有形净值/带息债务(%)
data6	长期借款/总资产(%)	经营活动产生的现金流量净额/经营活动净收益_TTM(%)	留存盈余比率(%)	息税前利润/营业总收入(%)	净资产收益率_TTM(%)	净利润同比增长(%)	存货周转天数(天/次)	每股营业总收入(元/股)	价值变动净收益/利润总额_TTM(%)	有形净值/净债务(%)
data7	应付债券	资本支			总资产	归属母公	应收帐	每股营业收	营业外收	息税折

	/总资产(%)	出/折旧和摊销			报酬率(%)	司股东的净利润同比增长(%)	款周转率(次)	入(元/股)	支净额／利润总额(%)	旧摊销前利润／负债合计
data8	归属母公司股东的权益／全部投入资本(%)	现金及现金等价物净增加额(元)			总资产报酬率_TTM(%)	归属母公司股东的净利润(扣除)同比增长(%)	应收帐款周转天数(天/次)	每股营业收入_TTM(元/股)	营业外收支净额／利润总额_TTM(%)	经营活动产生现金流量净额/负债合计
data9	带息债务／全部投入资本(%)	经营活动产生的现金流量净额(元)			总资产净利率(%)	过去五年同期归属母公司净利润平均增幅(%)	应付帐款周转率(次)	每股营业利润(元/股)	所得税／利润总额(%)	经营活动产生现金流量净额/带息债务
data10	流动负债／负债合计(%)	销售商品提供劳务收到的现金(元)			总资产净利率_TTM(%)	经营活动产生的现金流量净额同比增长(%)	应付帐款周转天数(天/次)	每股息税前利润(元/股)	扣除非经常损益后的净利润／净利润(%)	经营活动产生现金流量净额/流动负债
data11	非流动负债／负债合计(%)	自由现金流量(元)			投入资本回报率(%)	每股经营活动产生的现金流量净额同比增长(%)	流动资产周转率(次)	每股资本公积金(元/股)		经营活动产生现金流量净额/净债务
data12	股东权益比率(%)	净利润现金含量(%)			销售净利率(%)	净资产收益率(摊薄)同比增长(%)	固定资产周转率(次)	每股盈余公积(元/股)		利息保障倍数(倍)
data13	权益乘数(%)	营业收入现金含量(%)			销售净利率_TTM(%)	净资产同比增长(%)	股东权益周转率(次)	每股公积金(元/股)		长期负债与营运资金比率
data14	营运资金(元)	总资产现金回收率(%)			销售毛利率(%)	总资产同比增长(%)	总资产周转率(次)	每股未分配利润(元/股)		现金流动负债比
data15	长期负债/股东权益合计				销售毛利率_TTM(%)	每股净资产相对年初增长率(%)		每股留存收益(元/股)		
data16	长期资产				销售成	归属母公		每股经营活		

	适合率				本率(%)	司股东的权益相对年初增长率(%)		动产生的现金流量净额(元/股)		
data17					销售期间费用率(%)	资产总计相对年初增长率(%)		每股经营活动产生的现金流量净额_TTM(元/股)		
data18					销售期间费用率_TTM(%)	可持续增长率(%)		每股现金流量净额(元/股)		
data19					净利润／营业总收入(%)			每股现金流量净额_TTM(元/股)		
data20					净利润／营业总收入_TTM(%)			每股企业自由现金流量(元/股)		
data21					营业利润／营业总收入(%)			每股股东自由现金流量(元/股)		
data22					营业利润／营业总收入_TTM(%)					
data23					息税前利润／营业总收入(%)					
data24					息税前利润／营业总收入_TTM(%)					
data25					营业总成本／营业总收入(%)					

data26					营业总成本／营业总收入_TTM(%)					
data27					销售费用／营业总收入(%)					
data28					销售费用／营业总收入_TTM(%)					
data29					管理费用／营业总收入(%)					
data30					管理费用／营业总收入_TTM(%)					
data31					财务费用／营业总收入(%)					
data32					财务费用／营业总收入_TTM(%)					
data33					资产减值损失／营业总收入(%)					
data34					资产减值损失／营业总收入_TTM(%)					
data35					归属母					

					公司净利润(元)					
data36					扣除非经常性损益后的净利润(元)					
data37					息税前利润(元)					
data38					息税折旧摊销前利润(元)					
data39					营业利润率(%)					
data40					成本费用利润率(%)					

六、利润分配表

name	TB_INCOME_EXPENSES	TB_INCOME_PROFIT	TB_INCOME_REVENUE	TB_INCOME_SUMMARY
description	营业支出	利润汇总	营业收入	收益汇总
type	double_1d	double_1d	double_1d	double_1d
data0	报告时间	报告时间	报告时间	报告时间
data1	退保金	加:营业外收入	营业收入	其他综合收益
data2	赔付支出	减:营业外支出	利息净收入	加:影响综合收益总额的调整项目
data3	减:摊回赔付支出	非流动资产处置净损失	利息收入	综合收益总额
data4	提取保险责任准备金	加:影响利润总额的其他科目	利息支出	归属于母公司所有者的综合收益总额
data5	减:摊回保险责任准备金	加:影响利润总额的调整项目	手续费及佣金净收入	归属于少数股东的综合收益总额
data6	保单红利支出	利润总额	手续费及佣金收入	加:影响母公司综合收益总额的调整项目
data7	分保费用	减:所得税费用	手续费及佣金支出	基本每股收益
data8	保险手续费及佣金支出	加:未确认的投资损	代理买卖证券业务净收	稀释每股收益

		失	入	
data9	营业税金及附加	加:影响净利润的其他科目	证券承销业务净收入	资产处置收益
data10	业务及管理费	加:影响净利润的调整项目	受托客户资产管理业务净收入	按经营持续性分类
data11	减:摊回分保费用	净利润	已赚保费	持续经营净利润
data12	资产减值损失	归属于母公司所有者的净利润	保险业务收入	终止经营净利润
data13	营业总成本	少数股东损益	分保费收入	按所有权归属分类
data14	营业成本	非经营性净收益	减:分出保费	研发费用
data15	其他营业成本	非经营性净收益特殊项目	提取未到期责任准备金	信用减值损失
data16	销售费用	非经营性净收益调整项目	投资净收益	净敞口套期收益
data17	管理费用	营业利润	对联营合营企业的投资收益	利息费用(财务费用)
data18	财务费用		公允价值变动净收益	利息收入(财务费用)
data19	营业支出 OR 营业总成本特殊项目		汇兑收益	其他权益工具投资公允价值变动
data20	营业总成本调整项目		营业收入 OR 营业总收入特殊项目	企业自身信用风险公允价值变动
data21			营业收入调整项目	其他债权投资公允价值变动
data22			营业总收入	金融资产重分类计入其他综合收益的金额
data23			其他营业收入	其他债权投资信用减值准备

七、现金流量表

name	TB_CASHFLOW_CASH	TB_CASHFLOW_CFF	TB_CASHFLOW_CFFCFINONCASH	TB_CASHFLOW_CFI	TB_CASHFLOW_CFO	TB_CASHFLOW_CFOINDIRECT
description	现金及现金等价物	筹资活动现金流	非现金筹资投资	投资活动现金流	经营活动现金流	净利润到经营活动现金流的转换
type	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d
data0	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间
data1	汇率变动对现金及现金等价物的影响	吸收投资收到的现金	债务转为资本	收回投资收到的现金	客户存款和同业存放款项净增加额	净利润
data2	影响现金及现金等价物的其他科目	子公司吸收少数股东投资收到的现金	一年内到期的可转换公司债券	取得投资收益收到的现金	向中央银行借款净增加额	加:资产减值准备

data3	影响现金及现金等价物的调整项目	发行债券收到的现金	融资租入固定资产	处置固定资产、无形资产和其他长期资产收回的现金净额	向其他金融机构拆入资金净增加额	固定资产折旧
data4	现金及现金等价物净增加额	取得借款收到的现金		处置子公司及其他营业单位收到的现金净额	收回已核销贷款	无形资产摊销
data5	加:期初现金及现金等价物余额	收到其他与筹资活动有关的现金		收到其他与投资活动有关的现金	收取利息、手续费及佣金的现金	长期待摊费用摊销
data6	期末现金及现金等价物余额	筹资活动现金流入特殊项目		投资活动现金流入特殊项目	处置交易性金融资产净增加额	待摊费用减少(减:增加)
data7	现金的期末余额	筹资活动现金流入调整项目		投资活动现金流入调整项目	回购业务资金净增加额	预提费用增加(减:减少)
data8	减:现金的期初余额	筹资活动现金流入小计		投资活动现金流入小计	收到原保险合同保费取得的现金	处置固定资产、无形资产和其他长期资产的损失
data9	加:现金等价物的期末余额	偿还债务支付的现金		购建固定资产、无形资产和其他长期资产支付的现金	收到再保业务现金净额	固定资产报废损失
data10	减:现金等价物的期初余额	分配股利、利润或偿付利息支付的现金		投资支付的现金	保户储金及投资款净增加额	公允价值变动损失
data11	(附注)现金特殊项目	子公司支付给少数股东的股利、利润或偿付的利息		取得子公司及其他营业单位支付的现金净额	收到其他与经营活动有关的现金	财务费用
data12	(附注)现金调整项目	支付其他与筹资活动有关的现金		质押贷款净增加额	经营活动现金流入特殊项目	投资损失
data13	(附注)现金及现金等价物净增加额	筹资活动现金流出特殊项目		支付其他与投资活动有关的现金	经营活动现金流入调整项目	递延所得税资产减少
data14	加:现金净额前后对比调整项目	筹资活动现金流出调整项目		投资活动现金流出特殊项目	经营活动现金流入小计	递延所得税负债增加
data15		筹资活动现金流出小计		投资活动现金流出调整项目	支付给职工以及为职工支付的现金	存货的减少
data16		筹资活动流量现金净额调整项目		投资活动现金流出小计	支付的各项税费	经营性应收项目的减少
data17		筹资活动产生的现金流量净额		投资活动现金流量净额调整项目	客户贷款及垫款净增加额	经营性应付项目的增加
data18				投资活动产生的现金	存放中央银行	其他

				流量净额	和同业款项净增加额	
data19					拆出资金净增加额	(附注)经营活动现金流量净额特殊项目
data20					支付手续费及佣金的现金	(附注)经营活动现金流量净额调整项目
data21					支付原保险合同赔付款项的现金	(附注)经营活动产生的现金流量净额
data22					支付再保业务现金净额	加:经营流量净额前后对比调整项目
data23					支付保单红利的现金	
data24					支付其他与经营活动有关的现金	
data25					经营活动现金流出特殊项目	
data26					经营活动现金流出调整项目	
data27					经营活动现金流出小计	
data28					经营活动现金流量净额调整项目	
data29					经营活动产生的现金流量净额	
data30					销售商品、提供劳务收到的现金	
data31					收到的税费返还	
data32					购买商品、接受劳务支付的现金	

八、资产负债表

name	TB_CASHFLOW_CASH	TB_CASHFLOW_CFF	TB_CASHFLOW_CFFCFINONCASH	TB_CASHFLOW_CFI	TB_CASHFLOW_CFO	TB_CASHFLOW_CFOINDIRECT
description	现金及现金等价物	筹资活动现金流	非现金筹资投资	投资活动现金流	经营活动现金流	净利润到经营活动现金流的转换
type	double_1d	double_1d	double_1d	double_1d	double_1d	double_1d
data0	报告时间	报告时间	报告时间	报告时间	报告时间	报告时间
data1	汇率变动对现金及现金等价物的影响	吸收投资收到的现金	债务转为资本	收回投资收到的现金	客户存款和同业存放款项净增加额	净利润
data2	影响现金及现金等价物的其他科目	子公司吸收少数股东投资收到的现金	一年内到期的可转换公司债券	取得投资收益收到的现金	向中央银行借款净增加额	加:资产减值准备
data3	影响现金及现金等价物的调整项目	发行债券收到的现金	融资租入固定资产	处置固定资产、无形资产和其他长期资产收回的现金净额	向其他金融机构拆入资金净增加额	固定资产折旧
data4	现金及现金等价物净增加额	取得借款收到的现金		处置子公司及其他营业单位收到的现金净额	收回已核销贷款	无形资产摊销
data5	加:期初现金及现金等价物余额	收到其他与筹资活动有关的现金		收到其他与投资活动有关的现金	收取利息、手续费及佣金的现金	长期待摊费用摊销
data6	期末现金及现金等价物余额	筹资活动现金流入特殊项目		投资活动现金流入特殊项目	处置交易性金融资产净增加额	待摊费用减少(减:增加)
data7	现金的期末余额	筹资活动现金流入调整项目		投资活动现金流入调整项目	回购业务资金净增加额	预提费用增加(减:减少)
data8	减:现金的期初余额	筹资活动现金流入小计		投资活动现金流入小计	收到原保险合同保费取得的现金	处置固定资产、无形资产和其他长期资产的损失
data9	加:现金等价物的期末余额	偿还债务支付的现金		购建固定资产、无形资产和其他长期资产支付的现金	收到再保业务现金净额	固定资产报废损失
data10	减:现金等价物的期初余额	分配股利、利润或偿付利息支付的现金		投资支付的现金	保户储金及投资款净增加额	公允价值变动损失
data11	(附注)现金特殊项目	子公司支付给少数股东的股利、利润或偿付的利息		取得子公司及其他营业单位支付的现金净额	收到其他与经营活动有关的现金	财务费用
data12	(附注)现金调整项目	支付其他与筹资活动有关的		质押贷款净增加额	经营活动现金流入特殊	投资损失

		现金			项目	
data13	(附注)现金及现金等价物净增加额	筹资活动现金流出特殊项目		支付其他与投资活动有关的现金	经营活动现金流入调整项目	递延所得税资产减少
data14	加:现金净额前后对比调整项目	筹资活动现金流出调整项目		投资活动现金流出特殊项目	经营活动现金流入小计	递延所得税负债增加
data15		筹资活动现金流出小计		投资活动现金流出调整项目	支付给职工以及为职工支付的现金	存货的减少
data16		筹资活动流量现金净额调整项目		投资活动现金流出小计	支付的各项税费	经营性应收项目的减少
data17		筹资活动产生的现金流量净额		投资活动现金流量净额调整项目	客户贷款及垫款净增加额	经营性应付项目的增加
data18				投资活动产生的现金流量净额	存放中央银行和同业款项净增加额	其他
data19					拆出资金净增加额	(附注)经营活动现金流量净额特殊项目
data20					支付手续费及佣金的现金	(附注)经营活动现金流量净额调整项目
data21					支付原保险合同赔付款项的现金	(附注)经营活动产生的现金流量净额
data22					支付再保业务现金净额	加:经营流量净额前后对比调整项目
data23					支付保单红利的现金	
data24					支付其他与经营活动有关的现金	
data25					经营活动现金流出特殊项目	
data26					经营活动现金流出调整项目	
data27					经营活动现金流出小计	

data28					经营活动现金流量净额调整项目	
data29					经营活动产生的现金流量净额	
data30					销售商品、提供劳务收到的现金	
data31					收到的税费返还	
data32					购买商品、接受劳务支付的现金	

九、股本结构

name	TB_CAPITAL_CHANGE	TB_CAPITAL_CIRCULATIONSHARES	TB_CAPITAL_LEVEL	TB_CAPITAL_NONCIRCULATIONSHARES	TB_CAPITAL_RESTRICTEDSTOCK
description	股本变动	流通股份	股本层次	未流通股份	限售股份
type	string_1d	double_1d	double_1d	double_1d	double_1d
data0	报告时间	报告时间	报告时间	报告时间	报告时间
data1	股本变动原因类别_GBK	流通股份(股)	总股本(股)	发起人股(股)	有限售条件的流通股(股)
data2	股本变动原因说明_GBK	已上市流通 A 股(包含高管股)(股)	A 股(股)	国家股(股)	国家持股(股)
data3		战略投资者配售持股(股)	流通 A 股(股)	国有法人股(股)	国有法人持股(股)
data4		一般法人配售持股(股)	有限售条件的流通 A 股(股)(计算)	境内法人股(股)	其他内资持股(股)
data5		基金配售持股(股)	无限售条件流通 A 股(股)(计算)	外资法人股(股)	境内法人持股(股)
data6		增发未上市(股)	有限售条件的流通 A 股(股)(披露)	其它发起人股(股)	境内自然人持股(股)
data7		配股未上市(股)	无限售条件流通 A 股(股)(披露)	募集法人股(股)	高管股(股)
data8		其他流通股份(股)	未流通 A 股(股)	募集国有法人股(股)	外资持股(股)
data9		有限售流通 A 股(股)	B 股(股)	自然人法人股(股)	境外法人持股(股)
data10		有限售流通股中职	流通 B 股(股)	职工股(股)	境外自然人持股

		工股(股)			(股)
data11		B 股_旧(股)	未流通 B 股(股)	转配股(股)	其他有限售(股)
data12		未流通 B 股_旧	H 股(股)	优先股及其他(股)	
data13		有限售 B 股(股)	海外上市股(股)	优先股(股)	
data14		外资持 A 股(股)	S 股(股)	其他外资股(股)	
data15			N 股(股)		
data16			增发、配股未上市股份(股) (披露)		

十、公司资料

name	TB_COMPANYINFO_AGENCYOFFICE	TB_COMPANYINFO_COMPANYPROFILE	TB_COMPANYINFO_COMPANYSTATUSCHANGE	TB_COMPANYINFO_CONTACTINFORMATION	TB_COMPANYINFO_CONTACTPERSONNEL	TB_COMPANYINFO_INDUSTRYDIVISION	TB_COMPANYINFO_INDUSTRYPRODUCTS	TB_COMPANYINFO_LISTEDSECURITIES	TB_COMPANYINFO_LISTINGSTATUS	TB_COMPANYINFO_NAME	TB_COMPANYINFO_OTHER	TB_COMPANYINFO_REGISTRATION	TB_COMPANYINFO_SHAREHOLDERSTOPTEN	TB_COMPANYINFO_NUMBEROFSHAREHOLDERS	TB_COMPANYINFO_SHAREHOLDERSCIRCULATETOPTEN	TB_COMPANYINFO_INSTITUTIONALPOSITION	TB_COMPANYINFO_STOCKLIFTING	TB_COMPANYINFO_INCOMESUMMARY	TB_COMPANYINFO_BALANCESUMMARY	TB_COMPANYINFO_CASHSUMMARY	TB_COMPANYINFO_FREEZEANDPLEDGE	TB_COMPANYINFO_FREEZEANDPLEDGEDETAILS
description	中介机构	公司概况	公司状态更改	联系信息	联系人员	行业划分	行业产品	上市证券	上市状态	公司名称	其他	注册信息	十大股东	股东人数	十大流通股东	机构持仓	股票解禁	利润表摘要	资产负债表摘要	现金流量表摘要	股东股权冻结和质押	股东股权冻结和质押明细
type	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_1d	string_2d	double_1d	string_2d	string_2d	string_1d	double_1d	double_1d	double_1d	string_2d	double_1d
data0	会计师事务所_GBK	公司中文名称_GBK	证券名称_GBK	国家地区_GBK	董事会秘书_GBK	发布时间	经营范围_GBK	A股代码_GBK	上市时间	公司中文名称_GBK	公司介绍_GBK	成立日期	名次	统计日期	名次	机构类型_GBK	解禁日期	报告时间	报告时间	报告时间	信息来源_GBK	截止日期
da	律	曾	证	省	董	一	主	A	退	公		法	股	总	股东	OFI	解禁	营业	资产	经营	类别	待回

ta1	师事务所_GBK	用名_GBK	券代码_GBK	份_GBK	秘电话_GBK	级行业名称_GBK	营范围_GBK	股名称_GBK	市时间	司英文名称_GBK		人代表_GBK	东名称_GBK	户数	名称_GBK	I	数量(万股)	收入	总计	活动现金流入小计	选择_GBK	购余量(万股)
data2	法律顾问_GBK	省份_GBK	变更类型_GBK	城市_GBK	董秘邮箱_GBK	二级行业名称_GBK	兼营范围_GBK	B股代码_GBK	市场板块	中文简称_GBK		营业执照号_GBK	股份类型_GBK	A股户数	股份类型_GBK	保险	新增占已流通比例	营业支出	负债合计	经营活动现金流出小计	股权被冻结质押股东名称_GBK	待回购无限售条件余量(万股)
data3		所属申万行业_GBK	变动原因说明_GBK	注册地址_GBK	证券事务代表_GBK	三级行业名称_GBK		B股名称_GBK		拼音简码_GBK		总经理_GBK	持股数_GBK	户均持股数	持股数	信托	已流通A股(万股)	营业利润	所有者权益(或股东权益)合计	经营活动产生的现金流量净额	股权被冻结质押股东序号	待回购有限售条件余量(万股)
data4		所属概念板块_GBK	变更时间	注册地邮编_GBK	证代电话_GBK	四级行业名称_GBK		H股代码_GBK		英文简称_GBK		注册资本	占总股本比例_GBK	A股户均持股数	占流通股比例	券商	待流通A股(万股)	利润总额	负债和所有者权益(或股东权益)总计	投资活动现金流入小计	接受股权质押方_GBK	质押笔数
data5		董事长_GBK		办公地址_GBK	证代邮箱_GBK			H股名称_GBK		曾用名_GBK		货币类型_GBK	增减_GBK	前十大股东持股比例		基金	流通比例	净利润	流动资产合计	投资活动现金流出小计	涉及股数(股)	质押比例
data6		法人代		办公地	联系人			美股代		公司类			变动比			社保	股本变动说明	其他综合收益	非流动资产合	投资活动产生	占冻结质押方	

		表_GBK		邮编_GBK	_GBK			码_GBK		型_GBK			例_GBK				_GBK		计	的现金流量净额	持股数比例	
data7		主营业务_GBK		联系电话_GBK	联系人电话_GBK			美股名称_GBK								机构汇总		综合收益总额	流动负债合计	筹资活动现金流入小计	占总股本比例	
data8				联系传真_GBK	联系人邮箱_GBK			保留股票代码_GBK											非流动负债合计	筹资活动现金流出小计	股权冻结质押原因_GBK	
data9				联系地址_GBK				保留股票名称_GBK											归属母公司股东权益合计	筹资活动产生的现金流量净额	冻结质押期限起始日	
data10				联系地址邮编_GBK																	冻结质押期限截止日	
data11				联系邮箱_GBK																	事项描述与进展说明_GBK	
data12				公司网																	初始质押股数	

				址_GBK																(股)	
data13				信息披露网址_GBK																预计解押日期	
data14				信息披露报纸_GBK																股权被冻结质押股东所属性质_GBK	
data15																				首次信息发布日期	
data16																				信息发布日期	

十一、会员持仓

name	TB_MEMBER_positions_long	TB_MEMBER_positions_short
description	多头持仓	空头持仓
type	string_2d	string_2d
data0	会员简称_GBK	会员简称_GBK
data1	持买单量	持卖单量
data2	增减	增减

十二、TB 指数类型字段

name	TB_INDEXFORM_STATISTICS	TB_INDEXFORM_POSITIONCHANGE	TB_INDEXFORM_QUOTA	TB_INDEXFORM_FIVEDVOLUME	TB_INDEXFORM_FIVEDTURNOVER
description	涨跌统计	仓位变化	行情指标	五日平均成交量	五日分钟级累计平均成交额

type	int_1d	double_1d	double_1d	double_1d	double_2d
data0	涨跌数	换手	流通市值	五日平均成交量	时间
data1	涨停数	净流入	总市值		五日分钟级累计平均成交额
data2	跌停数	流入	市盈率		
data3		流出	市净率		

十三、其它

name	TB_MEMBER	TB_STOCKINDEX_MEMBER	TB_FUTURESINDEX_MEMBER	TB_INDUSTRY	TB_CST_holiday	TB_holiday	TB_TradeTime
description	历史成分股	股票行业成分股	期货分类成分	板块	中国假日	假日	交易时间
type	string_1d	string_1d	string_1d	string_1d	int_1d	string_1d	string_1d

附录:

一、除权换月的后复权处理

1.1 真实合约数据进行后复权处理

历史数据回测是量化投资中重要的一环，历史数据回测对数据的连续性要求较高。但是，我们知道，股票有送配股、分红派息，期货合约更是有到期时间，股票除权除息后，价格往往发生了较大的变化，而期货单个合约的数据是较短的，这些都导致了历史数据的不连续。历史数据的不连续，使得策略回测的难度大幅增加。目前，大部分的股票软件都有对股票进行向前或向后除权，这种除权，作为目测观察没有问题，但却因为不是真实价格而无法进行历史测试，很多期货软件也提供了连续的合约指数，但是这种指数也因为不同月份合约之间存在着很大的差异性而无法用做历史测试。

TBQuant（开拓者量化平台）提供了处理股票除权、期货主力合约换月的方法，让量化投资者能极其方便的进行除权换月的数据处理。我们提供了两种方法，一种是基于真实合约数据进行后复权处理，第二种是系统直接提供后复权之后的数据。两种处理方式的底层逻辑是一致的，第二种使用更加方便。

下面我们来讲解 TBQuant 真实合约数据进行后复权处理的逻辑和代码实现。
其实换月除权的处理需要关注五点：

1、 如何识别除权换月的时刻
2、 如何把因为除权换月引起的跳空处理成连续
3、 除权换月时刻的持仓如何切换
4、 实际交易如何以真实价格发出委托单
5、 如果需要在处理完的连续数据上，继续写交易策略，可以写在哪里？

1.1.1 如何识别除权换月的时刻

首先我们定义两个基础数据类型的变量读取基础数据的信息。

```
Dic<Array<String>> fRollover("TB_ROLLOVER"); //[期货换月合约,期货换月前价格,期货换月后价格]
```

```
Dic<Array<Numeric>> sXDXR("TB_XDXR");  //[每股送股,配股比例,配股价格,每股派现金,收盘价]
```

对于期货的换月时刻,TBQuant 提供了 GetDicTime(fRollover,0)可以读取历史上最近的一次换月的时刻点，如果两根 BAR 的这个时间不一致，就可以知道换月发生了。
If(GetDicTime(fRollover, 0) <> GetDicTime(fRollover, 1) And fRollover[0][1] <> InvalidString And fRollover[0][2] <> InvalidString)。

注意：基础数据的数据类型，目前基础数据是一维数组的形式。对于期货换月的基础数据是一个字符串为元素的一维数组。数组内容可以从数据中心查询如下：
0　期货换月合约
1　期货换月前价格
2　期货换月后价格

对于股票，除权除息也是存储在一个数值型的一维数组里面。所以股票除权除息的时点判断也使用与期货类似的方法。

```
If(GetDicTime(sXDXR, 0) <> GetDicTime(sXDXR, 1) And (sXDXR[0][0] <> InvalidNumeric Or (sXDXR[0][1] <> InvalidNumeric And sXDXR[0][2] <> InvalidNumeric)
        Or sXDXR[0][3] <> InvalidNumeric Or sXDXR[0][4] <> InvalidNumeric) And sXDXR[0][5] <> InvalidNumeric)
```

股票除权除息基础数据的数组内容从数据中心查询如下：
0　发布时间
1　每股送股
2　配股比例
3　配股价格
4　每股派现金
5　股票除权前价格

注意：这种处理方法，仅适合 1 秒到 1 天之间的周期（包括 1 秒和 1 天）。其它周期判断会出现错误。

1.1.2 如何把因为除权换月引起的跳空处理成连续？

1.　新老合约或者股票的折算系数 myRollover

我们只需要把在换月的那一刻连续合约和真实合约的折算比率计算出来。这个折算比率也是历史上每次换月新旧合约价格比值的累乘。

期货的折算比率：
相关代码：

```
Commentary("原合约收盘价：" + fRollover[0][1]);
Commentary("新合约收盘价：" + fRollover[0][2]);
myRollover = rClose / Value(fRollover[0][2]);
```

对于股票除权除息，我们也是本着求出一个除权换月前后的新旧价格的折算比率的思路来处理的。只不过股票要综合以下三种情形：
a)分红送股：10 送 2 股转 1 股派 1.3 元的除权价=(股权登记日的收盘价-每股派现金)÷(1+每股送红股数+每股转增股数)=(95.42-0.13)/(1+0.2+0.1)=73.3 元
b)配股：配股除权价＝(股权登记日收盘价+配股比例×配股价)/(1+配股比例)。
d)除权价：（除权前一日收盘价+配股价 X 配股比率－每股派息）/（1+配股比率+送股比率）

相关代码：

```
        //派息
         if(sXDXR[0][4] <> InvalidNumeric)
         {
           xdxr_profit = sXDXR[0][4];
         }
         //送股
         If(sXDXR[0][1] <> InvalidNumeric)
         {
           xdxr_scrip_issue = sXDXR[0][1];
         }
         //配股
         If(sXDXR[0][2] <> InvalidNumeric And sXDXR[0][3] <> InvalidNumeric)
         {
           xdxr_ration = sXDXR[0][2];
           xdxr_price = sXDXR[0][3];
         }
         //计算除权价
            sRolloverPrice = (sXDXR[0][5] + xdxr_price * xdxr_ration - xdxr_profit) / (1 +
xdxr_ration + xdxr_scrip_issue);
         //除权系数
         myRollover = rClose / sRolloverPrice;
```

2. 计算连续合约的价格

有了连续合约和真实价格的折算系数，连续合约的计算就非常简单：

代码如下：

```
    rOpen = Open * myRollover;
     rHigh = High * myRollover;
     rLow = Low * myRollover;
     rClose = Close * myRollover;
```

1.1.3 除权换月时刻的持仓如何切换

首先：发生除权换月后的第一根 BAR 开盘进行换仓交易处理。这个时间点最符合实际交易的情形。
其次：换仓本着换仓前后总市值相等的原则进行。以期货多头持仓为例，我们使用了如

下的交易指令。

```
If(MarketPosition == 1)
    {
      Sell(0, Value(fRollover[0][1]), Enum_Signal_UnCorrectPrice);
      Buy(Lots, Value(fRollover[0][2]), Enum_Signal_UnCorrectPrice);
    }
```

注意：
1）对于交易信号产生的这根 bar，Value(fRollover[0][1])和 Value(fRollover[0][2])可能超出 BAR 的价格范围，所以需要增加 Enum_Signal_UnCorrectPrice 提醒系统处理这样意外。
2）因为采用了 Enum_Signal_UnCorrectPrice，所以图表上的信号并不会产生真实的委托下单。所以使用这个代码处理需要配合监控器的同步功能一并使用。

1.1.4 实际交易如何以真实价格发出委托单？

这个需要注意我们的委托映射机制，比如期货我们公式是加载在 888 的连续合约运行，委托映射可以映射在最新的主力合约下单，委托映射会自动读取新合约的相应价格作为实际委托单的价格。

注意：在换月时刻账户持仓的切换，是通过监控器的同步功能实现的。

1.1.5 后续交易策略代码的编写

本代码只是完成了对除权换月引起的跳空的处理，处理之后的数据是 ropen,rhigh,rlow,rclose。如果用户想要针对新的数据进行交易，可以在参数，变量，和正文代码部分添加相应的代码即可。
完整代码如下

```
Params
  /*
  可增加后续交易策略需要的参数
  */
Vars
  //期货
  Dic<Array<String>> fRollover("TB_ROLLOVER"); //[期货换月合约,期货换月前价格,期货换月后价格]
  //股票
  Dic<Array<Numeric>> sXDXR("TB_XDXR");  //[每股送股,配股比例,配股价格,每股派现金,收盘价]
  Numeric xdxr_price(0);        //配股价格
  Numeric xdxr_profit(0);       //分配现金
  Numeric xdxr_ration(0);       //配股比率
  Numeric xdxr_scrip_issue(0);  //每股分配股票（送股+转增）
  Series<Numeric> sRolloverPrice(0,2);    //股票除权价

  //共有变量
  Series<Numeric> rOpen(0,60);          //除权换月后新的开盘价
```

```
  Series<Numeric> rHigh(0,60);          //除权换月后新的最高价
  Series<Numeric> rLow(0,60);           //除权换月后新的最低价
  Series<Numeric> rClose;              //除权换月后新的收盘价
  Series<Numeric> myRollover(1);         //除权换月系数
  Numeric Lots(0);                  //交易手数
  Global Numeric i(0);
  /*
  可增加后续交易策略需要的变量
  */
Events
OnBar(ArrayRef<Integer> indexs)
{
  Range[i = 0 : DataCount() - 1]
  {
    Commentary(DateTimeToString(GetDicTime(fRollover, 1)));
    If(CurrentBar == 0)
    {
      rOpen = Open;
      rHigh = High;
      rLow = Low;
      rClose = Close;
      myRollover = 1;
    }
    Else If(ExchangeName <> "上海证券交易所" And ExchangeName <> "深圳证券交易所") //期货换月
    {
      //仅适合 1 秒到 1 天之间的周期（包括 1 秒和 1 天）。其它周期判断会出现错误
      Commentary(TextArray(fRollover[0]));
           If(GetDicTime(fRollover, 0) <> GetDicTime(fRollover, 1) And fRollover[0][1] <>
InvalidString And fRollover[0][2] <> InvalidString)
       {
         PlotBool("换月", True);
         Commentary("原合约收盘价: " + fRollover[0][1]);
         Commentary("新合约收盘价: " + fRollover[0][2]);
         myRollover = rClose / Value(fRollover[0][2]);
                      Lots = Max(1, Round(CurrentContracts * Value(fRollover[0][1]) /
Value(fRollover[0][2]),0));

         If(MarketPosition == 1)
         {
           Sell(0, Value(fRollover[0][1]), Enum_Signal_UnCorrectPrice);
           Buy(Lots, Value(fRollover[0][2]), Enum_Signal_UnCorrectPrice);
         }
         Else If(MarketPosition == -1)
         {
```

```
            BuyToCover(0, Value(fRollover[0][1]), Enum_Signal_UnCorrectPrice);
            SellShort(Lots, Value(fRollover[0][2]), Enum_Signal_UnCorrectPrice);
        }
      }
    }
    Else //股票除权
    {
       Commentary(TextArray(sXDXR[0]));
       If(GetDicTime(sXDXR, 0) <> GetDicTime(sXDXR, 1) And (sXDXR[0][0] <> InvalidNumeric Or
(sXDXR[0][1] <> InvalidNumeric And sXDXR[0][2] <> InvalidNumeric)
         Or sXDXR[0][3] <> InvalidNumeric Or sXDXR[0][4] <> InvalidNumeric) And sXDXR[0][5] <>
InvalidNumeric)
       {
         //派息
         if(sXDXR[0][4] <> InvalidNumeric)
         {
           xdxr_profit = sXDXR[0][4];
         }
         //送股
         If(sXDXR[0][1] <> InvalidNumeric)
         {
           xdxr_scrip_issue = sXDXR[0][1];
         }
         //配股
         If(sXDXR[0][2] <> InvalidNumeric And sXDXR[0][3] <> InvalidNumeric)
         {
           xdxr_ration = sXDXR[0][2];
           xdxr_price = sXDXR[0][3];
         }
         // 分红送股--10 送 2 股转 1 股派 1.3 元的除权价=(股权登记日的收盘价-每股派现金)÷(1+每股
送红股数+每股转增股数)=(95.42-0.13)/(1+0.2+0.1)=73.3 元
         //配股--配股除权价＝(股权登记日收盘价+配股比例×配股价)/(1+配股比例)。
         //除权价= (除权前一日收盘价+配股价 X 配股比率－每股派息) / (1+配股比率+送股比率)
                                                                    /*http:/
/investor.szse.cn/institute/video/classroom/operation/t20190131_564542.html*/
         //计算除权价
            sRolloverPrice = (sXDXR[0][5] + xdxr_price * xdxr_ration - xdxr_profit) / (1 +
xdxr_ration + xdxr_scrip_issue);
         //除权系数
         myRollover = rClose / sRolloverPrice;
         PlotBool("除权",True);
         Commentary("除权价: " + Text(sRolloverPrice));
         Commentary("除权系数: " + Text(myRollover));
```

```
            Lots = Round(CurrentContracts * (sXDXR[0][5] / sRolloverPrice + xdxr_ration), -2);
            If(MarketPosition == 1)
            {
              Sell(0, sXDXR[0][5], Enum_Signal_UnCorrectPrice);
              Buy(Lots, sRolloverPrice, Enum_Signal_UnCorrectPrice);
            }
          }
        }
        rOpen = Open * myRollover;
        rHigh = High * myRollover;
        rLow = Low * myRollover;
        rClose = Close * myRollover;
        PlotKline(rOpen, rHigh, rLow, rClose);
        /*
        可增加后续交易策略需要的代码
        */
        If(MarketPosition<>1)
          Buy(100,Open);
      }
}
```

1.2 系统直接提供后复权之后的数据

对于系统直接提供后复权的数据，用户只需要在换月的时刻进行换仓处理操作即可。另外用户如果想得到复权前的真实合约的价格可以直接处理 rollover 系统函数进行转换。

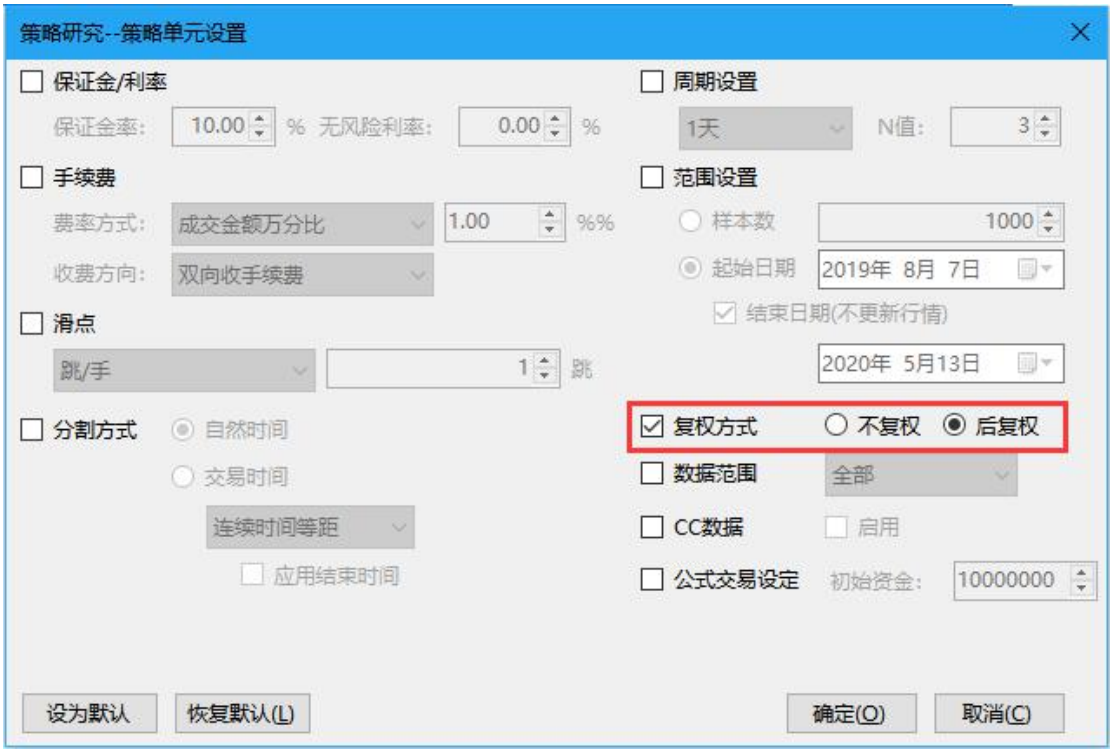
对于数据源使用后复权的用户需要注意以下三点：
1、系统后复权的逻辑和处理结果的读取
2、除权换月时刻的换仓处理
3、真实合约价格的获取

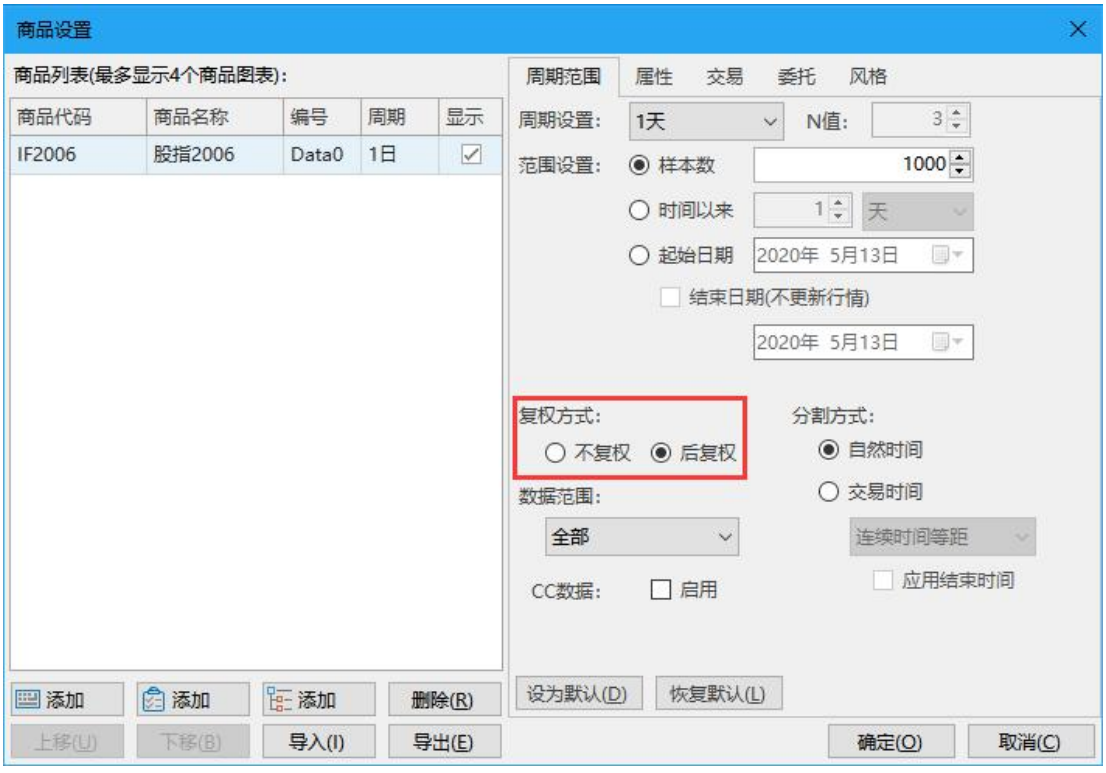
1.2.1 系统后复权的逻辑和处理结果的读取

系统后复权的逻辑在上一篇中有详细阐述，系统处理也是按照同样的逻辑来处理。这里介绍下处理后的结果怎么获取。系统处理之后的结果，首先数据源可以直接选取了后复权，其次提供了系统函数和数据源的属性 rollover，可以直接读取后复权前后真实合约和后复权之后合约的价格折算比率。

1. 后复权数据源的设置

用户在新建策略单元的时候，可以直接勾选复权方式，则调取的数据源就是后复权之后的数据源。也可以在 K 线图里面商品设置。如最后 K 线图所示：数据源是后复权的，指标副图里面是后复权倒推回来的真实连续合约。


策略研究--策略单元设置
保证金/利率
保证金率: 10.00 % 无风险利率: 0.00 %
手续费
费率方式: 成交金额万分比 1.00 %%
收费方向: 双向收手续费
滑点
跳/手 1 跳
分割方式
自然时间
交易时间
连续时间等距
应用结束时间
周期设置
1天 N值: 3
范围设置
样本数 1000
起始日期 2019年 8月 7日
结束日期(不更新行情)
2020年 5月13日
复权方式 不复权 后复权
数据范围 全部
CC数据 启用
公式交易设定 初始资金: 10000000
设为默认
恢复默认(L)
确定(O)
取消(C)


商品设置
商品列表(最多显示4个商品图表):
商品代码 商品名称 编号 周期 显示
IF2006 股指2006 Data0 1日
添加
添加
添加
删除(R)
上移(U)
下移(B)
导入(I)
导出(E)
周期范围
属性
交易
委托
风格
周期设置: 1天 N值: 3
范围设置: 样本数 1000
时间以来 1 天
起始日期 2020年 5月13日
结束日期(不更新行情)
2020年 5月13日
复权方式:
不复权 后复权
数据范围:
全部
CC数据: 启用
分割方式:
自然时间
交易时间
连续时间等距
应用结束时间
设为默认(D)
恢复默认(L)
确定(O)
取消(C)

2. rollover 的读取

Rollover 是真实合约的价格和后复权之后的价格之间的折算比率。这个系统根据数据源已经自动计算好，省区了用户自己计算的麻烦。
系统提供了两种方式来读取这个 Rollover。一种是系统函数，一种是结构体数据的属性。

数据函数 - Rollover

Rollover

说明	当前公式应用商品在当前Bar的除权系数。
语法	Numeric Rollover()
参数	无
备注	当前公式应用商品在当前Bar的除权系数，返回值为浮点数。
示例	无

内嵌结构体 - Bar

Bar - Bar数据

属性	类型	说明
dateTime	Numeric	开始时间，表示日期时间
open	Numeric	开盘价
high	Numeric	最高价
low	Numeric	最低价
close	Numeric	收盘价
turnOver	Numeric	成交金额
volume	Integer	成交量
openInt	Integer	持仓量
rollover	Numeric	除权系数
lastDateTime	Numeric	最后一个tick的时间戳，表示日期时间

1.2.2 除权换月时刻的换仓处理

首先识别除权换月时刻，可以很简单的使用 Rollover<>Rollover[1]来识别。
其次除权换月时刻的换仓还是基于等市值原则，对仓位进行调整。细节上需要注意的是平仓价格和开仓价格的确定，分别是 Close[1]/Rollover[1]，Close[1]/Rollover。

```
//除权换月时刻的换仓处理
    If(Rollover<>Rollover[1] And Rollover[1]<>InvalidNumeric)
    {
      PlotBool("换月",True);
      Commentary("原合约收盘价："+Text(Close[1]/Rollover[1]));
      Commentary("新合约收盘价："+Text(Close[1]/Rollover));
     lots=Max(1,Round(Abs(CurrentContracts*Rollover/Rollover[1]),IIF(Category==0,-2,0)));
          If(MarketPosition==1)
          {
            Sell(0,Close[1]/Rollover[1],Enum_Signal_UnCorrectPrice);
            Buy(lots,Close[1]/Rollover,Enum_Signal_UnCorrectPrice);
          }Else
          If(MarketPosition==-1)
          {
            BuyToCover(0,Close[1]/Rollover[1],Enum_Signal_UnCorrectPrice);
```

```
            SellShort(lots,Close[1]/Rollover,Enum_Signal_UnCorrectPrice);
        }
    }
```

1.2.3 真实合约价格的计算

真实合约价格的计算，只需要用后复权的价格处以 Rollover 就可以。

```
//真实价格
    If(Rollover<>InvalidNumeric)
    {
      RealOpen=Open/Rollover;
      RealHigh=High/Rollover;
      RealLow=Low/Rollover;
      RealClose=Close/Rollover;
    }
```

完整代码如下：

```
Params
  /*
  可增加后续交易策略需要的参数
  */
Vars
  Series<Numeric> RealOpen(0,60);       //真实合约的开盘价
  Series<Numeric> RealHigh(0,60);       //真实合约的最高价
  Series<Numeric> RealLow(0,60);        //真实合约的最低价
  Series<Numeric> RealClose;            //真实合约的收盘价
  Numeric Lots;                         //交易手数
  Global Numeric i;
  /*
  可增加后续交易策略需要的变量
  */
Events
OnBar(ArrayRef<Integer> indexs)
{
  Range[0:DataCount()-1]
  {
    Commentary(Text(Rollover()));
    //除权换月时刻的换仓处理
    If(Rollover<>Rollover[1] And Rollover[1]<>InvalidNumeric)
    {
      PlotBool("换月",True);
```

```
        Commentary("原合约收盘价: "+Text(Close[1]/Rollover[1]));
        Commentary("新合约收盘价: "+Text(Close[1]/Rollover));
        lots=Max(1,Round(Abs(CurrentContracts*Rollover/Rollover[1]),IIF(Category==0,-2,0)));
        If(MarketPosition==1)
        {
          Sell(0,Close[1]/Rollover[1],Enum_Signal_UnCorrectPrice);
          Buy(lots,Close[1]/Rollover,Enum_Signal_UnCorrectPrice);
        }Else
        If(MarketPosition==-1)
        {
          BuyToCover(0,Close[1]/Rollover[1],Enum_Signal_UnCorrectPrice);
          SellShort(lots,Close[1]/Rollover,Enum_Signal_UnCorrectPrice);
        }
      }
       //真实价格
      If(Rollover<>InvalidNumeric)
      {
        RealOpen=Open/Rollover;
        RealHigh=High/Rollover;
        RealLow=Low/Rollover;
        RealClose=Close/Rollover;
      }
      Else
      {
        RealOpen=Open;
        RealHigh=High;
        RealLow=Low;
        RealClose=Close;
      }
    /*
    可增加后续交易策略需要的代码
    */
    PlotKline(RealOpen,RealHigh,RealLow,RealClose);
    If(MarketPosition<>1)
      Buy(100,Open/Rollover,Enum_Signal_UnCorrectPrice);
    }
}
```

特别注意：这种写法因为交易信号都是采用真实价格，如果在最后一根 BAR 有持仓的情况下出测试报告，最后一根 bar 的结算价是复权后的收盘价。所以如果要得到真实的测试报告，需要自己对持仓强行平仓一下。

1.3 利用系统函数的简洁写法

1.3.1 系统函数：设置后复权和设置自动换仓

为了用户能更方便的处理除权换月的问题，TBQuant 推出了三个系统函数

```
AddDataFlag(Enum_Data_RolloverBackWard());//设置后复权
AddDataFlag(Enum_Data_RolloverRealPrice());//是否映射真实价格
AddDataFlag(Enum_Data_AutoSwapPosition());//设置自动换仓
```

设置后复权的指令和用户在数据源设置的菜单界面勾选后复权是一样的效果。
设置自动换仓的函数可以在期货换月和股票除权除息的时候，按照等市值原则自动换仓。
设置映射真实价格则对于交易指令的交易价格纠正为真实合约价格。

1.3.2 信号标注和测试报告的自动处理

设置映射真实价格之后系统实现的功能：
比如真实合约开盘价是 1000，复权后是 2000。在后复权的数据源上 Buy(1,open)的指令的处理有几点：

1. K 线上面信号标注位置，使用复权后的 open 价格 2000
2. K 线上面信号价格的注释，采用真实合约的 open 价格 1000
3. 测试报告中会使用真实合约的 open 价格 1000 进行测试。

注意：对于最后一根 BAR 有持仓，结算价也是收盘价对应的真实价格进行结算。

1.3.3 实际交易中的处理

实盘交易中还需要：

1. 映射主力
2. 除权换月时刻的监控器同步。

1.3.4 使用场景的灵活转变

一般自动换仓不管是什么策略测试都需要设置的。而有些策略需要在真实合约上进行，有些策略想利用复权后的连续合约，比如需要计算技术指标的策略。这两类策略可以通过设置是否后复权的参数来进行区分。

完整代码如下：

```
Params
  Bool  IsRollover(true);//是否后复权
  Bool  IsRolloverRealPrice(true);//是否映射真实价格
  Bool  IsAutoSwapPosition(true);//是否自动换仓
Events
OnInit()
{
  Range[0:DataCount-1]
  {
    If(IsRollover)
```

```
    {
      AddDataFlag(Enum_Data_RolloverBackWard());//设置后复权
    }
    If(IsRolloverRealPrice)
    {
      AddDataFlag(Enum_Data_RolloverRealPrice());//是否映射真实价格
    }
    If(IsAutoSwapPosition)
    {
      AddDataFlag(Enum_Data_AutoSwapPosition());//设置自动换仓
    }
  }
}
OnBar(ArrayRef<Integer> indexs)
{
  If(Rollover<>Rollover[1])
    PlotBool("true",True);
  If(MarketPosition<>1)
    Buy(100,Open);
}
```

二、除权换月影响到的函数

1、后复权，影响函数：

1、	GetBar	获取 Bar 数据
2、	Open	Bar 的开盘价
3、	O	Bar 的开盘价
4、	High	Bar 的最高价
5、	H	Bar 的最高价
6、	Low	Bar 的最低
7、	L	Bar 的最低价
8、	Close	Bar 的收盘价
9、	C	Bar 的收盘价
10、	HasRollover	当前 Bar 是否发生除权
11、	Rollover	当前 Bar 的除权系数
12、	Buy	多头建仓
13、	Sell	多头平仓
14、	SellShort	空头建仓
15、	BuyToCover	空头平仓
16、	FirstSignalIndex	第一次建仓位置
17、	NumWinTrades	交易盈利次数

18、NumLosTrades 交易亏损次数
19、NumEvenTrades 交易保本次数
20、TotalTrades 总交易次数
21、SignalSize 信号个数
22、PercentProfit 成功率
23、AvgBarsWinTrade 盈利交易平均持仓 Bar 数
24、AvgBarsLosTrade 亏损交易平均持仓 Bar 数
25、AvgBarsEvenTrade 保本交易平均持仓 Bar 数
26、TotalBarsWinTrades 盈利交易总持仓 Bar 数
27、TotalBarsLosTrades 亏损交易总持仓 Bar 数
28、TotalBarsEvenTrades 保本交易总持仓 Bar 数
29、LargestWinTrade 最大盈利数量
30、LargestLosTrade 最大亏损数量
31、GrossProfit 累计总盈利
32、GrossLoss 累计总亏损
33、NetProfit 净利润
34、MaxContractsHeld 最大持仓合约数
35、MaxConsecWinners 最大连续盈利次数
36、MaxConsecLosers 最大连续亏损次数
37、AvgEntryPrice 建仓平均价格
38、BarsSinceEntry 当前持仓的第一次建仓位置到当前位置的 Bar 计数
39、BarsSinceLastEntry 当前持仓的最后一次建仓位置到当前位置的 Bar 计数
40、BarsSinceExit 最近平仓位置到当前位置的 Bar 计数
41、MarketPosition 当前持仓状态
42、CurrentContracts 当前持仓合约数
43、CurrentEntries 当前建仓次数
44、ContractProfit 当前持仓位置的每手浮动盈亏
45、PositionProfit 当前持仓位置的浮动盈亏
46、EntryDate 当前持仓的第一次建仓日期
47、EntryTime 当前持仓的第一次建仓时间
48、EntryPrice 当前持仓的第一次建仓价格
49、ExitDate 最近一次平仓的 Bar 日期
50、ExitTime 最近一次平仓的 Bar 时间
51、ExitPrice 最近一次平仓的价格
52、LastEntryDate 当前持仓的最后一次建仓日期
53、LastEntryTime 当前持仓的最后一次建仓时间
54、LastEntryPrice 当前持仓的最后一次建仓价格
55、MaxContracts 当前持仓的最大持仓合约数
56、MaxEntries 最大建仓次数
57、MaxPositionProfit 当前持仓的最大浮动盈利数
58、MaxPositionLoss 当前持仓的最大浮动亏损数
59、Portfolio_GrossProfit 投资组合的毛利
60、Portfolio_GrossLoss 投资组合的毛损
61、Portfolio_NetProfit 投资组合的净利润

62、Portfolio_MaxDrawDown 投资组合的最大资产回撤
63、Portfolio_MaxDrawDownRatio 投资组合的最大资产回撤比率
64、Portfolio_NumLossTrades 投资组合的交易亏损次数
65、Portfolio_NumWinTrades 投资组合的交易盈利次数
66、Portfolio_TotalTrades 投资组合的总交易次数
67、Portfolio_PercentProfit 投资组合的成功率
68、Portfolio_CurrentEquity 投资组合的动态权益
69、Portfolio_PositionProfit 投资组合的浮动盈亏
70、Portfolio_CurrentEntries 投资组合的建仓合计次数
71、Portfolio_CurrentCapital 按当前 Bar 开盘价计算的可用资金
72、Portfolio_UsedMargin 当前持仓保证金
73、Portfolio_InitCapital 投资组合的初始资金
74、Portfolio_TotalProfit 投资组合的累计交易盈亏

2、自动换仓（后复权），影响函数同后复权。

3、自动换仓（不复权），影响函数:

1、HasRollover 当前 Bar 是否发生除权
2、Rollover 当前 Bar 的除权系数
3、NumWinTrades 交易盈利次数
4、NumLosTrades 交易亏损次数
5、NumEvenTrades 交易保本次数
6、TotalTrades 总交易次数
7、SignalSize 信号个数
8、PercentProfit 成功率
9、AvgBarsWinTrade 盈利交易平均持仓 Bar 数
10、AvgBarsLosTrade 亏损交易平均持仓 Bar 数
11、AvgBarsEvenTrade 保本交易平均持仓 Bar 数
12、TotalBarsWinTrades 盈利交易总持仓 Bar 数
13、TotalBarsLosTrades 亏损交易总持仓 Bar 数
14、TotalBarsEvenTrades 保本交易总持仓 Bar 数
15、LargestWinTrade 最大盈利数量
16、LargestLosTrade 最大亏损数量
17、GrossProfit 累计总盈利
18、GrossLoss 累计总亏损
19、NetProfit 净利润
20、MaxContractsHeld 最大持仓合约数
21、MaxConsecWinners 最大连续盈利次数
22、MaxConsecLosers 最大连续亏损次数
23、AvgEntryPrice 建仓平均价格
24、BarsSinceEntry 当前持仓的第一次建仓位置到当前位置的 Bar 计数
25、BarsSinceLastEntry 当前持仓的最后一次建仓位置到当前位置的 Bar 计数

26、BarsSinceExit　　最近平仓位置到当前位置的 Bar 计数
27、MarketPosition　　当前持仓状态
28、CurrentContracts　　当前持仓合约数
29、CurrentEntries　　当前建仓次数
30、ContractProfit　　当前持仓位置的每手浮动盈亏
31、PositionProfit　　当前持仓位置的浮动盈亏
32、EntryDate　　当前持仓的第一次建仓日期
33、EntryTime　　当前持仓的第一次建仓时间
34、EntryPrice　　当前持仓的第一次建仓价格
35、ExitDate　　最近一次平仓的 Bar 日期
36、ExitTime　　最近一次平仓的 Bar 时间
37、ExitPrice　　最近一次平仓的价格
38、LastEntryDate　　当前持仓的最后一次建仓日期
39、LastEntryTime　　当前持仓的最后一次建仓时间
40、LastEntryPrice　　当前持仓的最后一次建仓价格
41、MaxContracts　　当前持仓的最大持仓合约数
42、MaxEntries　　最大建仓次数
43、MaxPositionProfit　　当前持仓的最大浮动盈利数
44、MaxPositionLoss　　当前持仓的最大浮动亏损数
45、Portfolio_GrossProfit　　投资组合的毛利
46、Portfolio_GrossLoss　　投资组合的毛损
47、Portfolio_NetProfit　　投资组合的净利润
48、Portfolio_MaxDrawDown　　投资组合的最大资产回撤
49、Portfolio_MaxDrawDownRatio　　投资组合的最大资产回撤比率
50、Portfolio_NumLossTrades　　投资组合的交易亏损次数
51、Portfolio_NumWinTrades　　投资组合的交易盈利次数
52、Portfolio_TotalTrades　　投资组合的总交易次数
53、Portfolio_PercentProfit　　投资组合的成功率
54、Portfolio_CurrentEquity　　投资组合的动态权益
55、Portfolio_PositionProfit　　投资组合的浮动盈亏
56、Portfolio_CurrentEntries　　投资组合的建仓合计次数
57、Portfolio_CurrentCapital　　按当前 Bar 开盘价计算的可用资金
58、Portfolio_UsedMargin　　当前持仓保证金
59、Portfolio_InitCapital　　投资组合的初始资金
60、Portfolio_TotalProfit　　投资组合的累计交易盈亏

4、映射真实价格，后复权。影响同后复权。

5、映射真实价格，不复权，无意义。

6、自动换仓，忽略自动换仓影响，忽略影响的函数：

1、NumWinTrades	交易盈利次数
2、NumLosTrades	交易亏损次数
3、NumEvenTrades	交易保本次数
4、TotalTrades	总交易次数
5、PercentProfit	成功率
6、AvgBarsWinTrade	盈利交易平均持仓 Bar 数
7、AvgBarsLosTrade	亏损交易平均持仓 Bar 数
8、AvgBarsEvenTrade	保本交易平均持仓 Bar 数
9、TotalBarsWinTrades	盈利交易总持仓 Bar 数
10、TotalBarsLosTrades	亏损交易总持仓 Bar 数
11、TotalBarsEvenTrades	保本交易总持仓 Bar 数
12、LargestWinTrade	最大盈利数量
13、LargestLosTrade	最大亏损数量
14、MaxContractsHeld	最大持仓合约数
15、MaxConsecWinners	最大连续盈利次数
16、MaxConsecLosers	最大连续亏损次数
17、MaxEntries	最大建仓次数
18、Portfolio_NumLossTrades	投资组合的交易亏损次数
19、Portfolio_NumWinTrades	投资组合的交易盈利次数
20、Portfolio_TotalTrades	投资组合的总交易次数
21、Portfolio_PercentProfit	投资组合的成功率
22、Portfolio_C9urrentEntries	投资组合的建仓合计次数

忽略自动换仓影响时，换仓前后开仓市值不变，以开仓计算换仓后的开仓均价。
受影响函数：

1、GrossProfit	累计总盈利
2、GrossLoss	累计总亏损
3、NetProfit	净利润
4、MaxContractsHeld	最大持仓合约数
5、MaxConsecWinners	最大连续盈利次数
6、MaxConsecLosers	最大连续亏损次数
7、AvgEntryPrice	建仓平均价格
8、BarsSinceEntry	当前持仓的第一次建仓位置到当前位置的 Bar 计数
9、BarsSinceLastEntry	当前持仓的最后一次建仓位置到当前位置的 Bar 计数
10、BarsSinceExit	最近平仓位置到当前位置的 Bar 计数
11、MarketPosition	当前持仓状态
12、CurrentContracts	当前持仓合约数
13、CurrentEntries	当前建仓次数
14、ContractProfit	当前持仓位置的每手浮动盈亏
15、PositionProfit	当前持仓位置的浮动盈亏
16、EntryDate	当前持仓的第一次建仓日期
17、EntryTime	当前持仓的第一次建仓时间

18、EntryPrice 当前持仓的第一次建仓价格
19、ExitDate 最近一次平仓的 Bar 日期
20、ExitTime 最近一次平仓的 Bar 时间
21、ExitPrice 最近一次平仓的价格
22、LastEntryDate 当前持仓的最后一次建仓日期
23、LastEntryTime 当前持仓的最后一次建仓时间
24、LastEntryPrice 当前持仓的最后一次建仓价格
25、MaxContracts 当前持仓的最大持仓合约数
26、MaxPositionProfit 当前持仓的最大浮动盈利数
27、MaxPositionLoss 当前持仓的最大浮动亏损数
28、Portfolio_GrossProfit 投资组合的毛利
29、Portfolio_GrossLoss 投资组合的毛损
30、Portfolio_NetProfit 投资组合的净利润
31、Portfolio_MaxDrawDown 投资组合的最大资产回撤
32、Portfolio_MaxDrawDownRatio 投资组合的最大资产回撤比率
33、Portfolio0_CurrentEquity 投资组合的动态权益
34、Portfolio_PositionProfit 投资组合的浮动盈亏
35、Portfolio_CurrentCapital 按当前 Bar 开盘价计算的可用资金
36、Portfolio_UsedMargin 当前持仓保证金
37、Portfolio_InitCapital 投资组合的初始资金
38、Portfolio_TotalProfit 投资组合的累计交易盈亏