编制自动交易系统的基本知识

时间：2008-08-13 02:43:28　　来源：　　作者：

Introduction 介绍

鉴于许多中国的交易者对交易系统了解不多,本文解释使用 MQ4 语言编制自动交易系统的基本知识.

Title 编制自动交易系统的基本知识

一个交易系统大致包括以下几个方面:

1 开仓策略,即什么条件满足时开仓, 如某条线和某条线上交叉或下交叉,

2 平仓策略,即什么条件满足时平仓, 包括止赢设置,止损设置,和跟踪止赢设置三个方面.

3 资金管理, 其中一个方面就是下单的大小

4 时间管理, 如持仓时间,开平仓时间间隔等

5 账户状态分析,如交易历史,当前资金/仓位/各仓为盈亏状态等.

当然一个交易系统不必包括全部内容,本文做为入门知识也仅通过实例介绍交易系统程序的基本构成.

```
//+------------------------------------------------------------------
+
//|                              Designed by OKwh, China             |
//|                    Copyright 2007, OKwh Dxdcn                    |
//|                                       http://blog.sina.com.cn/FXTrade
|
//+------------------------------------------------------------------
+
#property  copyright  "Copyright 2007 , Dxd, China."
#property  link  "http://blog.sina.com.cn/FXTrade ,
http://www.mql4.com/users/DxdCn"
#define MAGICMA 200610011231
//+------------------------------------------------------------------
+
//| 注意没有指标文件那些 property                                    |
```

```
//+------------------------------------------------------------------
+
extern int whichmethod = 1;   //1~4 种下单方式  1 仅开仓, 2 有止损无止
赢, 3 有止赢无止损, 4 有止赢也有止损
extern double TakeProfit = 100;   //止赢点数
extern   double StopLoss = 20;    //止损点数
extern double MaximumRisk     = 0.3; //资金控制,控制下单量
extern double TrailingStop =25;     //跟踪止赢点数设置
extern   int maxOpen = 3;   //最多开仓次数限制
extern   int maxLots = 5;   //最多单仓持仓量限制
extern int bb = 0;       //非零就允许跟踪止赢
extern double MATrendPeriod=26;//使用 26 均线 开仓条件参数  本例子

int i, p2, xxx,p1, res;
double Lots;
datetime lasttime;       //时间控制, 仅当一个时间周期完成才检查条件
int init()   //初始化
{
Lots = 1;
lasttime = NULL;
return(0);
}
int deinit() { return(0); } //反初始化
//主程序
int start()
{
CheckForOpen();    //开仓 平仓 条件检查 和操作
if (bb>0)   CTP();   //跟踪止赢
return(0);
}
//+------下面是各子程序-------------------------------------------+
double LotsOptimized()   //确定下单量, 开仓调用 资金控制
{
double lot=Lots;
int   orders=HistoryTotal();   // history orders total
int   losses=0;              // number of losses orders without a break
//MarketInfo(Symbol(),MODE_MINLOT);     相关信息
//MarketInfo(Symbol(),MODE_MAXLOT);
//MarketInfo(Symbol(),MODE_LOTSTEP);
lot=NormalizeDouble(MaximumRisk *
AccountBalance()/AccountLeverage(),1);     //开仓量计算
if(lot<0.1) lot=0.1;
if(lot>maxLots) lot=maxLots;
return(lot);
```

```
}

//平仓持有的买单
void CloseBuy()
{
if (OrdersTotal( ) > 0 )
{
  for(i=OrdersTotal()-1;i>=0;i--)
  {
  if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)     break;
  if(OrderType()==OP_BUY)
  {
    OrderClose(OrderTicket(),OrderLots(),Bid,3,White);
    Sleep(5000);
  }
  }
}
}
//平仓持有的卖单
void CloseSell()
{
if (OrdersTotal( ) > 0 )
{
  for(i=OrdersTotal()-1;i>=0;i--)
  {
  if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)     break;
  if(OrderType()==OP_SELL)
    {
    OrderClose(OrderTicket(),OrderLots(),Ask,3,White);
    Sleep(5000);
    }
  }
}
}
//判断是否买或卖或平仓
int buyorsell()    //在这个函数计算设置你的交易信号  这里使用MACD 和MA
做例子
{
  double MacdCurrent, MacdPrevious, SignalCurrent;
  double SignalPrevious, MaCurrent, MaPrevious;
  MacdCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0);
  MacdPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,1);
  SignalCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0);
  SignalPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,1);
```

```
  MaCurrent=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,0);
  MaPrevious=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,1);
if(MacdCurrent<0 && MacdCurrent>SignalCurrent &&
MacdPrevious<SignalPrevious
    && MaCurrent>MaPrevious)
  return (1); // 买 Ma 在上升，Macd 在 0 线上，并且两线上交叉
if(MacdCurrent>0 && MacdCurrent<SignalCurrent &&
MacdPrevious>SignalPrevious
    && MaCurrent<MaPrevious)
  return (-1); // 卖
return (0); //不交易
}
int nowbuyorsell = 0;
void CheckForOpen()
{
if (Time[0] == lasttime ) return; //每时间周期检查一次  时间控制
lasttime = Time[0];
nowbuyorsell = buyorsell(); //获取买卖信号

if (nowbuyorsell == 1) //买  先结束已卖的
  CloseSell();
if (nowbuyorsell == -1) //卖  先结束已买的
    CloseBuy();  if (TimeDayOfWeek(CurTime()) == 1)
  {
  if (TimeHour(CurTime()) < 3 ) return; //周一早 8 点前不做 具体决定于
你的时区和服务器的时区  时间控制
  }
if (TimeDayOfWeek(CurTime()) == 5)
  {
  if (TimeHour(CurTime()) > 19 ) return; //周五晚 11 点后不做
  }

if (OrdersTotal( ) >= maxOpen) return ;
//如果已持有开仓次数达到最大，不做
if (nowbuyorsell==0) return;   //不交易
TradeOK();   //去下单交易
}
void TradeOK()   //去下单交易
{
int error ;
if (nowbuyorsell == 1) //买
  {
    switch (whichmethod)
    {
```

```
    case 1:
res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,0,0,"",MAGICMA,0,
Blue);break;
    case 2:
res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,Ask-StopLoss*Poin
t,0,"",MAGICMA,0,Blue); break;
    case 3:
res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,0,Ask+TakeProfit*
Point,"",MAGICMA,0,Blue);break;
    case 4:
res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,Ask-StopLoss*Poin
t,Ask+TakeProfit*Point,"",MAGICMA,0,Blue);break;
    default :
res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,0,0,"",MAGICMA,0,
Blue);break;
    }
    if (res <=0)
    {
    error=GetLastError();
    if(error==134)Print("Received 134 Error after OrderSend() !! ");
// not enough money
    if(error==135) RefreshRates();   // prices have changed
    }
    Sleep(5000);
    return ;
  }
if (nowbuyorsell == -1) //卖
  {
    switch (whichmethod)
    {
    case 1:
res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,0,0,"",MAGICMA,0
,Red); break;
    case 2:
res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,Bid+StopLoss*Poi
nt,0,"",MAGICMA,0,Red); break;
    case 3:
res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,0,Bid-TakeProfit
*Point,"",MAGICMA,0,Red); break;
    case 4:
res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,Bid+StopLoss*Poi
nt,Bid-TakeProfit*Point,"",MAGICMA,0,Red); break;
```

```
    default :
res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,0,0,"",MAGICMA,0
,Red); break;
    }
    if (res <=0)
    {
    error=GetLastError();
    if(error==134) Print("Received 134 Error after OrderSend() !! ");
// not enough money
    if(error==135) RefreshRates();   // prices have changed
    }
    Sleep(5000);
    return ;
  }
}
void CTP()   //跟踪止赢
{
bool bs = false;
for (int i = 0; i < OrdersTotal(); i++)
{
  if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)     break;
  if (OrderType() == OP_BUY)
  {
    if ((Bid - OrderOpenPrice()) > (TrailingStop *
MarketInfo(OrderSymbol(), MODE_POINT)))    //开仓价格 当前止损和当前
价格比较判断是否要修改跟踪止赢设置
    {
    if (OrderStopLoss() < Bid - TrailingStop * MarketInfo(OrderSymbol(),
MODE_POINT))
    {
      bs = OrderModify(OrderTicket(), OrderOpenPrice(), Bid -
TrailingStop * MarketInfo(OrderSymbol(), MODE_POINT),
OrderTakeProfit(),0, Green);
    }
    }
  }
  else if (OrderType() == OP_SELL)
  {
    if ((OrderOpenPrice() - Ask) > (TrailingStop *
MarketInfo(OrderSymbol(), MODE_POINT)))  //开仓价格 当前止损和当前价
格比较判断是否要修改跟踪止赢设置

    {
```

```
    if ((OrderStopLoss()) > (Ask + TrailingStop *
MarketInfo(OrderSymbol(), MODE_POINT)))
    {
      bs = OrderModify(OrderTicket(), OrderOpenPrice(),
        Ask + TrailingStop * MarketInfo(OrderSymbol(), MODE_POINT),
OrderTakeProfit(),0, Tan);
}
    }
  }
}
}
```

Conclusion 结论

本例 介绍了自动交易系统程序文件的基本构成, 略加修改就可以用于建立你自己系统.
比如根据你的下单策略修改 buyorSell()函数.

注: 本人 2006 比赛的系统就是这样的框架, 2007 比赛当然也将是这样的框架

简单的订单管理

时间：2008-08-13 02:38:19　　来源：　　作者：

1. 介绍

每个智能交易程序里都有一段代码是控制建仓的。它在所有的定单中不断搜索，通过信息选择仓位，然后进行修改和关闭。这段代码看上去都差不多，并且往往具有相同的功能。这就是为什么这段经常被重复的代码可以从程序中提取出来成为函数，从而使程序更易写更简洁。

首先，我们按功能把任务分成三个步骤 — 这三个步骤其实是三种智能交易程序:

- 智能交易程序在同一时间只能新建一个仓位
- 智能交易程序在同一时间可以新建每个类型的一个仓位(比如, 多头和空头的仓位)
- 智能交易程序可以同时新建多个仓位

2. 一个仓位

只新建一个仓位有许多中策略。这种控制代码块非常简单，但写出来也会耗费一定的时间和精力。

举一个简单的例子，一个来自于 MACD 线交叉点(信号线和基础线)的建仓信号，简化它的控制代码块，程序如下:

```
extern int  _MagicNumber = 1122;


int start()
{
    //---- 记住指标值做分析数据
    double MACD_1 = iMACD(Symbol(), 0, 12, 26, 9, PRICE_CLOSE,
                          MODE_MAIN, 1 );
    double MACD_2 = iMACD(Symbol(), 0, 12, 26, 9, PRICE_CLOSE,
                          MODE_MAIN, 2 );
```

```
    int _GetLastError = 0, _OrdersTotal = OrdersTotal();
    //---- 在开仓位置搜索

    for ( int z = _OrdersTotal - 1; z >= 0; z -- )

    {
// 如果在搜索中生成错误,转至下一个仓位
        if ( !OrderSelect( z, SELECT_BY_POS ) )

        {
          _GetLastError = GetLastError();
          Print("OrderSelect( ", z, ", SELECT_BY_POS ) - 错误 #",
                 _GetLastError );
            continue;
        }


// 如果当前的货币对没有开仓,
// 忽略过
        if ( OrderSymbol() != Symbol() ) continue;


// 如果 MagicNumber 不等于_MagicNumber,
// 忽略这个仓位
        if ( OrderMagicNumber() != _MagicNumber ) continue;

        //---- 如果 BUY 舱位开仓,

        if ( OrderType() == OP_BUY )
        {
            //---- 如果 MACD 遇到下降的零线,

            if(NormalizeDouble(MACD_1, Digits + 1) <  0.0 &&
               NormalizeDouble( MACD_2, Digits + 1) >= 0.0)

            {
                //---- 平仓
                if(!OrderClose( OrderTicket(), OrderLots(),
                   Bid, 5, Green))

                {
                   _GetLastError = GetLastError();
                   Alert("错误 OrderClose 鈅?", _GetLastError);
                   return(-1);
```

```
                }

            }
// 如果信号线没有改变,退出:
 // 开新仓位过早
            else
            { return(0); }

        }
        //---- 如果 SELL 仓位开仓,
        if ( OrderType() == OP_SELL )

        {
            //---- 如果 MACD 遇到上升的零线
            if(NormalizeDouble(MACD_1, Digits + 1) >  0.0 &&
               NormalizeDouble(MACD_2, Digits + 1 ) <= 0.0)

            {
              //---- 平仓
              if(!OrderClose( OrderTicket(), OrderLots(),

                 Ask, 5, Red))
              {
                _GetLastError = GetLastError();
                Alert( "错误 OrderClose 鈔?", _GetLastError );
                return(-1);
              }

            }
// 如果信号没有给便,退出:
// 开新仓位过早
            else return(0);
        }
    }


//+------------------------------------------------------------------
+
//|如果达到此点,说明没有开仓仓位
|
//| 检测可能开仓
|
//+------------------------------------------------------------------
+
```

```
    //---- 如果 MACD 遇到上升的零线,
    if ( NormalizeDouble( MACD_1, Digits + 1 ) >  0.0 &&
          NormalizeDouble( MACD_2, Digits + 1 ) <= 0.0    )

    {
        //---- 打开 BUY 仓位
        if(OrderSend( Symbol(), OP_BUY, 0.1, Ask, 5, 0.0, 0.0,
           "MACD_test", _MagicNumber, 0, Green ) < 0)

        {
            _GetLastError = GetLastError();
            Alert( "错误 OrderSend 鈔?", _GetLastError );
            return(-1);
        }

        return(0);
    }
    //---- 如果 MACD 遇到下降的零线,
    if(NormalizeDouble( MACD_1, Digits + 1 ) <  0.0 &&
          NormalizeDouble( MACD_2, Digits + 1 ) >= 0.0)

    {
        //---- open a SELL position
        if(OrderSend( Symbol(), OP_SELL, 0.1, Bid, 5, 0.0, 0.0,
            "MACD_test",
              _MagicNumber, 0, Red ) < 0 )

        {
            _GetLastError = GetLastError();
            Alert( "错误 OrderSend 鈔?", _GetLastError );
            return(-1);
        }

        return(0);
    }

    return(0);
}
```

现在我们把代码块写成函数。这个函数能够在所有的定单中搜索出需要的，并将其信息记录在全局变量中，程序如下:

```
int _Ticket = 0, _Type = 0; double _Lots = 0.0,

_OpenPrice = 0.0, _StopLoss = 0.0;
double _TakeProfit = 0.0; datetime _OpenTime = -1;

double _Profit = 0.0, _Swap = 0.0;
double _Commission = 0.0; string _Comment = "";

datetime _Expiration = -1;

void OneOrderInit( int magic )

{
    int _GetLastError, _OrdersTotal = OrdersTotal();

    _Ticket = 0; _Type = 0; _Lots = 0.0; _OpenPrice = 0.0;

_StopLoss = 0.0;
    _TakeProfit = 0.0; _OpenTime = -1; _Profit = 0.0;

_Swap = 0.0;
    _Commission = 0.0; _Comment = ""; _Expiration = -1;

    for ( int z = _OrdersTotal - 1; z >= 0; z -- )

    {
        if ( !OrderSelect( z, SELECT_BY_POS ) )

        {
          _GetLastError = GetLastError();
          Print("OrderSelect( ", z, ", SELECT_BY_POS ) -错误#",
                _GetLastError );
          continue;
        }

        if(OrderMagicNumber() == magic && OrderSymbol() ==
           Symbol())

        {
          _Ticket    = OrderTicket();
          _Type      = OrderType();
          _Lots      = NormalizeDouble( OrderLots(), 1 );
          _OpenPrice = NormalizeDouble( OrderOpenPrice(), Digits);
```

```
            _StopLoss = NormalizeDouble( OrderStopLoss(), Digits);
            _TakeProfit = NormalizeDouble( OrderTakeProfit(), Digits);
            _OpenTime   = OrderOpenTime();
            _Profit     = NormalizeDouble( OrderProfit(), 2 );
            _Swap       = NormalizeDouble( OrderSwap(), 2 );
            _Commission = NormalizeDouble( OrderCommission(), 2 );
            _Comment    = OrderComment();
            _Expiration = OrderExpiration();
            return;
          }

    }
}
```

如你所见，这非常简单：一共 11 个变量，每个都储存仓位的相关信息(ticket #, type, lot size, 等等). 当函数开始运行时，这些变量被归零。作为全局变量这是必需的。函数被调用时变量也可以不归零，但我们需要的不是先前的信息，我们需要的是最近的。然后所有的仓位会以标准的方式被搜索，一旦获得需要的信号和 MagicNumber 值，信息将被存储在相应的变量中。

现在我们将函数用到智能交易程序中:

```
extern int  _MagicNumber = 1122;

#include <OneOrderControl.mq4>

int start()

{
    int _GetLastError = 0;

// 记住开仓的参量(如果可用)
    OneOrderInit( _MagicNumber );

    //---- 记住指标值用作分析

    double MACD_1 = iMACD(Symbol(), 0, 12, 26, 9, PRICE_CLOSE,
                          MODE_MAIN, 1 );
    double MACD_2 = iMACD(Symbol(), 0, 12, 26, 9, PRICE_CLOSE,

                          MODE_MAIN, 2 );
```

```
// 现在,代替在仓位中的搜索
// 存在开仓:
if ( _Ticket > 0 )

{
    //----如果 BUY 仓位开仓,
    if ( _Type == OP_BUY )

    {
        //---- 如果 MACD 遇到下降的零线,
    if(NormalizeDouble( MACD_1, Digits + 1 ) <  0.0 &&
           NormalizeDouble( MACD_2, Digits + 1 ) >= 0.0)

        {
            //---- 平仓
            if(!OrderClose( _Ticket, _Lots, Bid, 5, Green))

            {
              _GetLastError = GetLastError();
              Alert( "错误 OrderClose 鈩?", _GetLastError);
              return(-1);
            }

        }
        // 如果信号没有改变，退出:
        // 开新仓位过早            else return(0);
    }
    //----如果 SELL 仓位开仓,

    if ( _Type == OP_SELL )
    {
        //---- 如果 MACD 遇到上升的零线

        if(NormalizeDouble( MACD_1, Digits + 1 ) >  0.0 &&
           NormalizeDouble( MACD_2, Digits + 1 ) <= 0.0)

        {
            //---- 平仓
            if(!OrderClose( _Ticket, _Lots, Ask, 5, Red))

            {
                _GetLastError = GetLastError();
                Alert( "错误 OrderClose 鈩?", _GetLastError);
```

```
                return(-1);
            }

        }
        // 如果信号没有改变，退出:
        // 开新仓位过早
        else return(0);
    }
}

// 如果智能交易没有开仓
// ( _Ticket == 0 )
// 如果 MACD 遇到上升的零线
if(NormalizeDouble( MACD_1, Digits + 1 ) >  0.0 &&
   NormalizeDouble( MACD_2, Digits + 1 ) <= 0.0)

{
    //---- 开 BUY 仓位
    if(OrderSend(Symbol(), OP_BUY, 0.1, Ask, 5, 0.0, 0.0,
        "CrossMACD", _MagicNumber, 0, Green ) < 0)

    {
        _GetLastError = GetLastError();
        Alert( "错误 OrderSend 鈔?", _GetLastError );
        return(-1);
    }

    return(0);
}
//---- 如果 MACD 遇到下降的零线
if ( NormalizeDouble( MACD_1, Digits + 1 ) <  0.0 &&
       NormalizeDouble( MACD_2, Digits + 1 ) >= 0.0    )

{
    //---- 开 SELL 仓位
    if(OrderSend(Symbol(), OP_SELL, 0.1, Bid, 5, 0.0, 0.0,
        "CrossMACD",
          _MagicNumber, 0, Red ) < 0 )

    {
        _GetLastError = GetLastError();
        Alert( "错误 OrderSend 鈔?", _GetLastError );
        return(-1);
    }
```

```
        return(0);
    }

    return(0);
}
```

如你所见，这段智能交易的程序显得更紧凑更易读。这是一个简单例子。

现在让我们解决下一个任务。

3. 每个类型的一个仓位

我们需要一个更复杂的智能交易程序来实现一些其它的功能。此程序能够新建许多不同类型的仓位，并进行操作。以下是这种程序的规则:

- 该程序运行时将设置两个待办定单：在卖出价+20 点设置买入止损，在买入价+20 点设置卖出止损;
- 当一个定单引发，另一个必须被取消;
- 建仓必须伴随追踪止损;
- 当仓位由于止损或盈利被关闭后，将被再次启动，也就是说两个待办定单将被设置。

程序如下:

```
extern int    _MagicNumber = 1123;


extern double Lot          = 0.1;
extern int    StopLoss     = 60;

// 止损点的间距 (0 - 无)
extern int    TakeProfit   = 100;
// 赢利点的间距 (0 - 无)
extern int    TrailingStop = 50;

// 追踪止损点 (0 - 无)

extern int    Luft         = 20;
```

```
// 挂单交易放置水平的间距


int start()

{
// 记住定单中每个票据
    int BuyStopOrder = 0, SellStopOrder = 0, BuyOrder = 0,

    SellOrder = 0;
    int _GetLastError = 0, _OrdersTotal = OrdersTotal();

// 在所有的开仓仓位搜索并记住
// 开仓仓位已存在的类型:
    for ( int z = _OrdersTotal - 1; z >= 0; z -- )

    {
        // 如果在搜索中生成错误,
        // 转至下一个仓位
        if ( !OrderSelect( z, SELECT_BY_POS ) )

        {
          _GetLastError = GetLastError();
          Print("OrderSelect(", z, ", SELECT_BY_POS) - Error #",
                _GetLastError );
          continue;
        }


// 如果当前货币对没有开仓仓位，忽略它
  if ( OrderSymbol() != Symbol() ) continue;


// 如果 MagicNumber 不等于 _MagicNumber,
// 忽略这个仓位
        if ( OrderMagicNumber() != _MagicNumber ) continue;

        // 取决于仓位类型,
        // 改变变量值:

        switch ( OrderType() )
        {
          case OP_BUY:      BuyOrder      = OrderTicket(); break;
          case OP_SELL:     SellOrder     = OrderTicket(); break;
```

```
      case OP_BUYSTOP:  BuyStopOrder  = OrderTicket(); break;
      case OP_SELLSTOP: SellStopOrder = OrderTicket(); break;
    }

}

//---- 如果我们有两个挂单交易，退出
//---- 等待他们开启
if ( BuyStopOrder > 0 && SellStopOrder > 0 ) return(0);

// 在全部定单中第二次搜索
// 现在运行它们:

_OrdersTotal = OrdersTotal();
for ( z = _OrdersTotal - 1; z >= 0; z -- )

{
    // 如果在仓位搜索中生成错误，
    // 转至下一个仓位
    if ( !OrderSelect( z, SELECT_BY_POS ) )

    {
        _GetLastError = GetLastError();
        Print("OrderSelect(", z, ", SELECT_BY_POS) - 错误 #",
              _GetLastError );
        continue;
    }



    // 如果对于当前的货币对没有开仓
    // 忽略它
    if ( OrderSymbol() != Symbol() ) continue;

    // 如果 MagicNumber 不等于 _MagicNumber,
    // 忽略这个仓位

    if ( OrderMagicNumber() != _MagicNumber ) continue;

    // 取决于仓位的类型，
    // 改变变量值:

    switch ( OrderType() )
    {
        //----如果 BUY 仓位开仓，
```

```
case OP_BUY:
{

    // 如果 SellStop 定单还没有删除,
    // 删除:
    if ( SellStopOrder > 0 )
    {

      if ( !OrderDelete( SellStopOrder ) )
      {

        Alert(OrderDelete Error #", GetLastError());
        return(-1);
      }

    }
    // 检测止损被移动:
    // 如果追踪止损的大小不是很小,
    if(TrailingStop > MarketInfo( Symbol(),

       MODE_STOPLEVEL ) )
    {
        // 如果赢利点超过追踪止损点,
        if(NormalizeDouble( Bid - OrderOpenPrice(),
           Digits ) >
           NormalizeDouble(TrailingStop*Point,

           Digits ) )
        {
            // 如果新的止损水平超过当前仓位的水平
            // (或者如果仓位没有止损),
            if(NormalizeDouble(Bid -
               TrailingStop*Point, Digits ) >
               OrderStopLoss() || OrderStopLoss() <=
               0.0 )

            {
                //---- 修改定单
                if(!OrderModify( OrderTicket(),
                   OrderOpenPrice(),
                   NormalizeDouble(Bid -

                   TrailingStop*Point, Digits ),
                   OrderTakeProfit(),
```

```
                    OrderExpiration()))

                {
                    Alert("OrderModify 错误 #",
                        GetLastError());
                    return(-1);
                }

            }
        }
    }
    // 如果没有开仓仓位，退出
    // 无事可做
    return(0);
}

// 下一个单元格与 BUY 仓位的单元个一样
// 这就是我们不能在单元格上标注的原因...
case OP_SELL:
{
    if ( BuyStopOrder > 0 )

    {
        if ( !OrderDelete( BuyStopOrder ) )

        {
            Alert("OrderDelete 错误 #",
                  GetLastError());
            return(-1);
        }

    }
    if(TrailingStop > MarketInfo( Symbol(),
       MODE_STOPLEVEL ) )

    {
      if(NormalizeDouble(OrderOpenPrice() - Ask,
      Digits) > NormalizeDouble(TrailingStop*Point,
      Digits ) )

        {
          if(NormalizeDouble(Ask + TrailingStop*Point,
             Digits ) < OrderStopLoss() ||
```

```
                        OrderStopLoss() <= 0.0 )
                       {
                         if(!OrderModify( OrderTicket(),
                            OrderOpenPrice(),
                      NormalizeDouble(Ask + TrailingStop*Point,

                            Digits), OrderTakeProfit(),
                            OrderExpiration() ) )
                            {

                              Alert("OrderModify Error #",
                                    GetLastError());
                              return(-1);
                            }

                       }
                    }
                }
                return(0);
            }
        }

    }


//+------------------------------------------------------------------
+
//| 如果执行达到此点，
|
//| 说明没有挂单和开仓。
|
//+------------------------------------------------------------------
+
//---- 放置 BuyStop 和 SellStop:
    double _OpenPriceLevel, _StopLossLevel, _TakeProfitLevel;
    _OpenPriceLevel = NormalizeDouble( Ask + Luft*Point, Digits );

    if ( StopLoss > 0 )

    { _StopLossLevel = NormalizeDouble( _OpenPriceLevel -
                       StopLoss*Point, Digits ); }

    else
    { _StopLossLevel = 0.0; }
```

```
if ( TakeProfit > 0 )

{ _TakeProfitLevel = NormalizeDouble( _OpenPriceLevel +
                     TakeProfit*Point, Digits ); }

else
{ _TakeProfitLevel = 0.0; }

if ( OrderSend ( Symbol(), OP_BUYSTOP, Lot, _OpenPriceLevel,
      5, _StopLossLevel, _TakeProfitLevel, "",
     _MagicNumber ) < 0 )

{
    Alert( "OrderSend Error #", GetLastError() );
    return(-1);
}



_OpenPriceLevel = NormalizeDouble(Bid - Luft*Point, Digits);

if ( StopLoss > 0 )

{ _StopLossLevel = NormalizeDouble( _OpenPriceLevel +
                   StopLoss*Point, Digits ); }

else
{ _StopLossLevel = 0.0; }

if ( TakeProfit > 0 )

{ _TakeProfitLevel = NormalizeDouble( _OpenPriceLevel -
                     TakeProfit*Point, Digits ); }

else
{ _TakeProfitLevel = 0.0; }

if ( OrderSend ( Symbol(), OP_SELLSTOP, Lot, _OpenPriceLevel,
     5, _StopLossLevel,
               _TakeProfitLevel, "", _MagicNumber ) < 0 )

{
    Alert( "OrderSend Error #", GetLastError() );
```

```
        return(-1);
    }


    return(0);
}
```

现在让我们写出可以简化控制建仓代码的函数，它必须用每个类型的定单进行搜索，然后将这些信息存储在全局变量里，程序如下:

```
// 在定单特性中的整体变量会被储存:
int _BuyTicket = 0, _SellTicket = 0, _BuyStopTicket = 0;

int _SellStopTicket = 0, _BuyLimitTicket = 0, _SellLimitTicket = 0;


double _BuyLots = 0.0, _SellLots = 0.0, _BuyStopLots = 0.0;

double _SellStopLots = 0.0, _BuyLimitLots = 0.0,
_SellLimitLots = 0.0;


double _BuyOpenPrice = 0.0, _SellOpenPrice = 0.0,
_BuyStopOpenPrice = 0.0;

double _SellStopOpenPrice = 0.0, _BuyLimitOpenPrice = 0.0,
_SellLimitOpenPrice = 0.0;


double _BuyStopLoss = 0.0, _SellStopLoss = 0.0, _BuyStopStopLoss = 0.0;

double _SellStopStopLoss = 0.0, _BuyLimitStopLoss = 0.0,
_SellLimitStopLoss = 0.0;


double _BuyTakeProfit = 0.0, _SellTakeProfit = 0.0,
_BuyStopTakeProfit = 0.0;

double _SellStopTakeProfit = 0.0, _BuyLimitTakeProfit = 0.0,
  _SellLimitTakeProfit = 0.0;
```

```
datetime _BuyOpenTime = -1, _SellOpenTime = -1,
_BuyStopOpenTime = -1;

datetime _SellStopOpenTime = -1, _BuyLimitOpenTime = -1,
_SellLimitOpenTime = -1;


double _BuyProfit = 0.0, _SellProfit = 0.0, _BuySwap = 0.0,

_SellSwap = 0.0;
double _BuyCommission = 0.0, _SellCommission = 0.0;


string _BuyComment = "", _SellComment = "", _BuyStopComment = "";

string _SellStopComment = "", _BuyLimitComment = "",
_SellLimitComment = "";


datetime _BuyStopExpiration = -1, _SellStopExpiration = -1;
datetime _BuyLimitExpiration = -1, _SellLimitExpiration = -1;


void OneTypeOrdersInit( int magic )
{
// 变量归零:
    _BuyTicket = 0; _SellTicket = 0; _BuyStopTicket = 0;
    _SellStopTicket = 0; _BuyLimitTicket = 0; _SellLimitTicket = 0;

    _BuyLots = 0.0; _SellLots = 0.0; _BuyStopLots = 0.0;
    _SellStopLots = 0.0; _BuyLimitLots = 0.0; _SellLimitLots = 0.0;

    _BuyOpenPrice = 0.0; _SellOpenPrice = 0.0; _BuyStopOpenPrice = 0.0;
    _SellStopOpenPrice = 0.0; _BuyLimitOpenPrice = 0.0;

_SellLimitOpenPrice = 0.0;

    _BuyStopLoss = 0.0; _SellStopLoss = 0.0; _BuyStopStopLoss = 0.0;
    _SellStopStopLoss = 0.0; _BuyLimitStopLoss = 0.0;

_SellLimitStopLoss = 0.0;

    _BuyTakeProfit = 0.0; _SellTakeProfit = 0.0;
```

```
_BuyStopTakeProfit = 0.0;
    _SellStopTakeProfit = 0.0; _BuyLimitTakeProfit = 0.0;

_SellLimitTakeProfit = 0.0;

    _BuyOpenTime = -1; _SellOpenTime = -1; _BuyStopOpenTime = -1;
    _SellStopOpenTime = -1; _BuyLimitOpenTime = -1;

_SellLimitOpenTime = -1;

    _BuyProfit = 0.0; _SellProfit = 0.0; _BuySwap = 0.0;

_SellSwap = 0.0;
    _BuyCommission = 0.0; _SellCommission = 0.0;

    _BuyComment = ""; _SellComment = ""; _BuyStopComment = "";
    _SellStopComment = ""; _BuyLimitComment = "";

_SellLimitComment = "";

    _BuyStopExpiration = -1; _SellStopExpiration = -1;
    _BuyLimitExpiration = -1; _SellLimitExpiration = -1;

    int _GetLastError = 0, _OrdersTotal = OrdersTotal();
    for ( int z = _OrdersTotal - 1; z >= 0; z -- )

    {
        if ( !OrderSelect( z, SELECT_BY_POS ) )

        {
          _GetLastError = GetLastError();
          Print("OrderSelect(", z, ",SELECT_BY_POS) - Error #",

                _GetLastError );
          continue;
        }
        if ( OrderMagicNumber() == magic && OrderSymbol() ==

            Symbol() )
        {
          switch ( OrderType() )

          {
            case OP_BUY:
```

```
  _BuyTicket     = OrderTicket();
  _BuyLots       = NormalizeDouble( OrderLots(), 1 );
  _BuyOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                    Digits );
  _BuyStopLoss   = NormalizeDouble( OrderStopLoss(),
                                    Digits );
  _BuyTakeProfit = NormalizeDouble( OrderTakeProfit(),
                                     Digits );
  _BuyOpenTime   = OrderOpenTime();
  _BuyProfit     = NormalizeDouble( OrderProfit(), 2 );
  _BuySwap       = NormalizeDouble( OrderSwap(), 2 );
  _BuyCommission = NormalizeDouble( OrderCommission(),
                                     2 );
  _BuyComment    = OrderComment();
  break;
case OP_SELL:
  _SellTicket     = OrderTicket();
  _SellLots       = NormalizeDouble( OrderLots(), 1 );
  _SellOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                      Digits );
  _SellStopLoss   = NormalizeDouble( OrderStopLoss(),
                                     Digits );
  _SellTakeProfit = NormalizeDouble( OrderTakeProfit(),
                                      Digits );
  _SellOpenTime   = OrderOpenTime();
  _SellProfit     = NormalizeDouble( OrderProfit(), 2 );
  _SellSwap       = NormalizeDouble( OrderSwap(), 2 );
  _SellCommission = NormalizeDouble( OrderCommission(),
                                     2 );
  _SellComment    = OrderComment();
  break;
case OP_BUYSTOP:
  _BuyStopTicket     = OrderTicket();
  _BuyStopLots       = NormalizeDouble( OrderLots(), 1 );
  _BuyStopOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                        Digits );
  _BuyStopStopLoss   = NormalizeDouble( OrderStopLoss(),
                                        Digits );
  _BuyStopTakeProfit = NormalizeDouble( OrderTakeProfit(),
                                        Digits );
  _BuyStopOpenTime   = OrderOpenTime();
  _BuyStopComment    = OrderComment();
  _BuyStopExpiration = OrderExpiration();
  break;
```

```
            case OP_SELLSTOP:
               _SellStopTicket     = OrderTicket();
               _SellStopLots       = NormalizeDouble( OrderLots(), 1 );
               _SellStopOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                                      Digits );
               _SellStopStopLoss   = NormalizeDouble( OrderStopLoss(),
                                                      Digits );
               _SellStopTakeProfit = NormalizeDouble( OrderTakeProfit(),
                                                      Digits );
               _SellStopOpenTime   = OrderOpenTime();
               _SellStopComment    = OrderComment();
               _SellStopExpiration = OrderExpiration();
               break;
            case OP_BUYLIMIT:
               _BuyLimitTicket     = OrderTicket();
               _BuyLimitLots       = NormalizeDouble( OrderLots(), 1 );
               _BuyLimitOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                                      Digits );
               _BuyLimitStopLoss   = NormalizeDouble( OrderStopLoss(),
                                                      Digits );
               _BuyLimitTakeProfit = NormalizeDouble( OrderTakeProfit(),
                                                      Digits );
               _BuyLimitOpenTime   = OrderOpenTime();
               _BuyLimitComment    = OrderComment();
               _BuyLimitExpiration = OrderExpiration();
               break;
             case OP_SELLLIMIT:
               _SellLimitTicket     = OrderTicket();
               _SellLimitLots       = NormalizeDouble( OrderLots(), 1 );
               _SellLimitOpenPrice  = NormalizeDouble( OrderOpenPrice(),
                                                       Digits );
               _SellLimitStopLoss   = NormalizeDouble( OrderStopLoss(),
                                                       Digits );
               _SellLimitTakeProfit =
NormalizeDouble( OrderTakeProfit(),
                                                      Digits );
               _SellLimitOpenTime   = OrderOpenTime();
               _SellLimitComment    = OrderComment();
               _SellLimitExpiration = OrderExpiration();
               break;
             }

         }
     }
```

```
}
```

现在我们将函数用到智能交易程序中:

```
extern int    _MagicNumber = 1123;
extern double Lot          = 0.1;

extern int    StopLoss     = 60;
// 止损点的间距(0 - d无)
extern int    TakeProfit   = 100;

// 赢利点的间距 (0 - 无)
extern int    TrailingStop = 50;
//追踪止损点 (0 - 无)

extern int    Luft         = 20;

// 挂单交易放置水平的间距

#include <OneTypeOrdersControl.mq4>

int start()
{
    int _GetLastError = 0;

    //---- 记住开仓的参量(如果可用)

    OneTypeOrdersInit( _MagicNumber );

    //---- 如果我们两个都是挂单交易,退出
    //---- 等待他们开启
    if ( _BuyStopTicket > 0 && _SellStopTicket > 0 ) return(0);

    //---- 如果 BUY 仓位开仓

    if ( _BuyTicket > 0 )
    {
        //---- 如果 SellStop 还没有删除，删除它:

        if ( _SellStopTicket > 0 )
        {
            if ( !OrderDelete( _SellStopTicket ) )
```

```
            {
                Alert( "OrderDelete 错误#", GetLastError() );
                return(-1);
            }

        }
        //---- 检测止损被移动:
        //---- 如果追踪止损不是很小,
        if ( TrailingStop > MarketInfo( Symbol(),

             MODE_STOPLEVEL ) )
        {
//---- 如果赢利仓位超过追踪止损点,
            if ( NormalizeDouble( Bid - _BuyOpenPrice, Digits ) >
                  NormalizeDouble( TrailingStop*Point, Digits ) )

            {
//---- 如果新止损水平超过当前仓位
//---- (或者当前仓位没有止损),
                if(NormalizeDouble( Bid - TrailingStop*Point,

                   Digits ) > _BuyStopLoss
                   || _BuyStopLoss <= 0.0 )

                {
                    //---- 修改定单
                    if ( !OrderModify( _BuyTicket, _BuyOpenPrice,
                          NormalizeDouble( Bid - TrailingStop*Point,
                           Digits ),
                          _BuyTakeProfit, 0 ) )

                    {
                        Alert( "OrderModify 错误#",
                               GetLastError() );
                        return(-1);
                    }

                }
            }
        }
//---- 如果没有开仓仓位，退出，无事可做
        return(0);
    }
```

```
//---- 这个单元格与 BUY 仓位的单元格相似

//---- 这就是我们不能做标注的原因...
    if ( _SellTicket > 0 )

    {
        if ( _BuyStopTicket > 0 )
        {

            if ( !OrderDelete( _BuyStopTicket ) )
            {

                Alert( "OrderDelete 错误 #", GetLastError() );
                return(-1);
            }

        }
        if(TrailingStop > MarketInfo( Symbol(), MODE_STOPLEVEL))

        {
            if(NormalizeDouble( _SellOpenPrice - Ask, Digits ) >
                 NormalizeDouble( TrailingStop*Point, Digits ) )

            {
                if(NormalizeDouble( Ask + TrailingStop*Point,

                   Digits ) < _SellStopLoss
                      || _SellStopLoss <= 0.0 )

                {
                    if(!OrderModify( _SellTicket, _SellOpenPrice,
                       NormalizeDouble( Ask + TrailingStop*Point,

                       Digits ),
                       _SellTakeProfit, 0 ) )
                    {
                        Alert( "OrderModify Error #",

                               GetLastError() );
                        return(-1);
                    }
                }
```

```
            }

        }
        return(0);
    }


//+------------------------------------------------------------------
+
//| 如果执行达到此点，
|
//| 说明没有挂单和开仓。
|

//+------------------------------------------------------------------
+
//---- 放置 BuyStop 和 SellStop:
    double _OpenPriceLevel, _StopLossLevel, _TakeProfitLevel;
    _OpenPriceLevel = NormalizeDouble( Ask + Luft*Point, Digits);

    if ( StopLoss > 0 )

      _StopLossLevel = NormalizeDouble( _OpenPriceLevel -
                       StopLoss*Point, Digits );
    else

      _StopLossLevel = 0.0;

    if ( TakeProfit > 0 )

      _TakeProfitLevel = NormalizeDouble( _OpenPriceLevel +
                         TakeProfit*Point, Digits );
    else

      _TakeProfitLevel = 0.0;

    if(OrderSend ( Symbol(), OP_BUYSTOP, Lot, _OpenPriceLevel,
       5, _StopLossLevel, _TakeProfitLevel, "", _MagicNumber ) < 0)

    {
        Alert( "OrderSend Error #", GetLastError() );
        return(-1);
    }
```

```
    _OpenPriceLevel = NormalizeDouble( Bid - Luft*Point, Digits);

    if ( StopLoss > 0 )

      _StopLossLevel = NormalizeDouble( _OpenPriceLevel +
                       StopLoss*Point, Digits );
    else

      _StopLossLevel = 0.0;

    if ( TakeProfit > 0 )

      _TakeProfitLevel = NormalizeDouble( _OpenPriceLevel -
                         TakeProfit*Point, Digits );
    else

      _TakeProfitLevel = 0.0;

    if(OrderSend ( Symbol(), OP_SELLSTOP, Lot, _OpenPriceLevel,
        5, _StopLossLevel, _TakeProfitLevel, "",
       _MagicNumber ) < 0 )

    {
        Alert( "OrderSend 错误 #", GetLastError() );
        return(-1);
    }


    return(0);
}
```

最初的和修改后的程序之间的不同是非常明显的 — 控制建仓的代码块变得更简单易懂。

现在轮到最复杂的智能交易程序了，它允许在同一时间无限制的新建多个仓位。

4. 控制所有仓位

现在需要有足够的变量来存储定单的信息，为此我们可以创建一些数组来实现这个目的。鉴于此，这个程序的功能将几乎和前面一样:

- 开始时所有的数组归零;
- 在所有的定单里搜索，找到符合需要的信号和 MagicNumber 值后，将这些信息存储在数组中;
- 为了使可用性更强，必须添加一个全局变量来记录智能交易程序的定单个数 — 这在访问数组时会很有用。

让我们立即开始编写该函数:

```
// 智能交易的全部定单总量变量将会存储:
int _ExpertOrdersTotal = 0;


// 定单特性的数组将会被存储:
int _OrderTicket[], _OrderType[];
double _OrderLots[], _OrderOpenPrice[], _OrderStopLoss[],

_OrderTakeProfit[];
double _OrderProfit[], _OrderSwap[], _OrderCommission[];
datetime _OrderOpenTime[], _OrderExpiration[];

string _OrderComment[];

void AllOrdersInit( int magic )
{

    int _GetLastError = 0, _OrdersTotal = OrdersTotal();

    // 按照当前仓位总数改变数组的大小
  // (if _OrdersTotal = 0, 改变数组总数为 1)

    int temp_value = MathMax( _OrdersTotal, 1 );
    ArrayResize( _OrderTicket,     temp_value );
    ArrayResize( _OrderType,       temp_value );
    ArrayResize( _OrderLots,       temp_value );
    ArrayResize( _OrderOpenPrice,  temp_value );
    ArrayResize( _OrderStopLoss,   temp_value );
    ArrayResize( _OrderTakeProfit, temp_value );
    ArrayResize( _OrderOpenTime,   temp_value );
    ArrayResize( _OrderProfit,     temp_value );
    ArrayResize( _OrderSwap,       temp_value );
    ArrayResize( _OrderCommission, temp_value );
    ArrayResize( _OrderComment,    temp_value );
    ArrayResize( _OrderExpiration, temp_value );
```

```
// zeroize the arrays

ArrayInitialize( _OrderTicket,     0 );
ArrayInitialize( _OrderType,       0 );
ArrayInitialize( _OrderLots,       0 );
ArrayInitialize( _OrderOpenPrice,  0 );
ArrayInitialize( _OrderStopLoss,   0 );
ArrayInitialize( _OrderTakeProfit, 0 );
ArrayInitialize( _OrderOpenTime,   0 );
ArrayInitialize( _OrderProfit,     0 );
ArrayInitialize( _OrderSwap,       0 );
ArrayInitialize( _OrderCommission, 0 );
ArrayInitialize( _OrderExpiration, 0 );

_ExpertOrdersTotal = 0;
for ( int z = _OrdersTotal - 1; z >= 0; z -- )

{
    if ( !OrderSelect( z, SELECT_BY_POS ) )

    {
      _GetLastError = GetLastError();
      Print("OrderSelect(", z, ",SELECT_BY_POS) -错误 #",
            _GetLastError );
      continue;
    }

    if ( OrderMagicNumber() == magic && OrderSymbol() ==
         Symbol() )

    {
        // 填数组
        _OrderTicket[_ExpertOrdersTotal] = OrderTicket();
        _OrderType[_ExpertOrdersTotal] = OrderType();
        _OrderLots[_ExpertOrdersTotal] =

        NormalizeDouble( OrderLots(), 1 );
        _OrderOpenPrice[_ExpertOrdersTotal] =

        NormalizeDouble( OrderOpenPrice(), Digits );
        _OrderStopLoss[_ExpertOrdersTotal] =

        NormalizeDouble( OrderStopLoss(), Digits );
```

```
            _OrderTakeProfit[_ExpertOrdersTotal] =

            NormalizeDouble( OrderTakeProfit(), Digits );
            _OrderOpenTime[_ExpertOrdersTotal] = OrderOpenTime();
            _OrderProfit[_ExpertOrdersTotal] =
            NormalizeDouble( OrderProfit(), 2 );
            _OrderSwap[_ExpertOrdersTotal] =

            NormalizeDouble( OrderSwap(), 2 );
            _OrderCommission[_ExpertOrdersTotal] =
            NormalizeDouble( OrderCommission(), 2 );
            _OrderComment[_ExpertOrdersTotal] = OrderComment();
            _OrderExpiration[_ExpertOrdersTotal] =
            OrderExpiration();
            _ExpertOrdersTotal++;
        }

    }

    // 按照智能交易所属仓位的总量改变数组的大小
    // (if _ExpertOrdersTotal = 0, 改变数组大小为 1)
    temp_value = MathMax( _ExpertOrdersTotal, 1 );
    ArrayResize( _OrderTicket,     temp_value );
    ArrayResize( _OrderType,       temp_value );
    ArrayResize( _OrderLots,       temp_value );
    ArrayResize( _OrderOpenPrice,  temp_value );
    ArrayResize( _OrderStopLoss,   temp_value );
    ArrayResize( _OrderTakeProfit, temp_value );
    ArrayResize( _OrderOpenTime,   temp_value );
    ArrayResize( _OrderProfit,     temp_value );
    ArrayResize( _OrderSwap,       temp_value );
    ArrayResize( _OrderCommission, temp_value );
    ArrayResize( _OrderComment,    temp_value );
    ArrayResize( _OrderExpiration, temp_value );

}
```

为了了解函数运行的详情，让我们来写一个简单的智能交易程序，它将显示该程序新建所有仓位的信息。

代码相当简单:

```
extern int _MagicNumber    = 0;
```

```
#include AllOrdersControl.mq4>

int start()
{
    AllOrdersInit( _MagicNumber );


    if ( _ExpertOrdersTotal > 0 )

    {
        string OrdersList = StringConcatenate(Symbol(),
                  ", MagicNumber ", _MagicNumber, ":\n");
        for ( int n = 0; n _ExpertOrdersTotal; n ++ )

        {
            OrdersList = StringConcatenate( OrdersList,
                           "Order # ", _OrderTicket[n],
                ", profit/loss: ",

                DoubleToStr( _OrderProfit[n], 2 ),
                "  ", AccountCurrency(), "\n" );
        }

        Comment( OrdersList );
    }

    return(0);
}
```

如果 _MagicNumber 设为 0, 智能交易程序将显示手动建仓的列表:

5. 总结

最后，我想来比较一下使用函数与否在搜索定单时的速度。为此，我们用相同的版本来连续测试 "Every tick" 模式 10 次(用_MagicNumber 最优化)，用 MetaTrader 软件来计算时间 — 时间时自己计算的。

结果如下:

智能交易	10 测试的时间 (mm:ss)
CrossMACD_beta (不包含函数)	07:42
CrossMACD	11:37
DoublePending_beta (不包含函数)	08:18
DoublePending	09:42

正如你的表格里看到的，使用函数的智能交易程序稍稍慢了一点，这作为使源代码简单易懂的代价，应该还算合理。

无论如何，每个人都有是否使用函数的自由

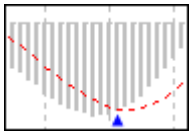
[转]智能交易示例

时间：2008-08-10 23:35:50 来源： 作者：

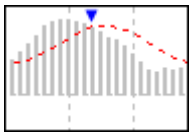
以 MQL4 语言程序为原理展示一个以标准 MACD 指标为基础简单的智能交易系统的创建。 在这个智能交易中，我们将看到一些特性的示例，像赢利水平的设定，追踪止损的设置等等。在我们的范例中，通过开仓和管理仓位来完成交易。

交易原理:

- Long (BUY) entry - MACD 指标在零以下, 从下至上并且穿过低端的信号线。



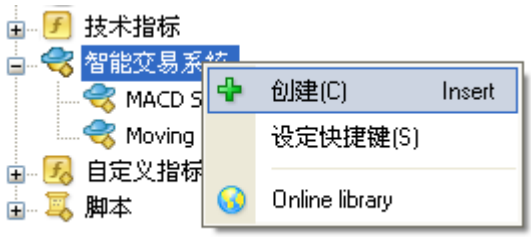
- Short (SELL) entry - MACD 指标在零以上,从上至下并且穿过顶端的信号线。



- Long exit - 赢利上限的执行，追踪止损的执行或当穿过 MACD 的信号线 (MACD 指标在零以上,从上至下并且穿过顶端的信号线)。
- Short exit - 赢利上限的执行，追踪止损的执行或当穿过 MACD 的信号线 (MACD 指标在零以下, 从下至上并且穿过低端的信号线)。

重要提示:从我们的分析上排除一些 MACD 指标微不足道的变化 (图表上的小 '山丘')，我们介绍一种补充检测 '山丘' 大小的办法如下:指标的大小应该在最低价的最后 5 个单元(5*Point,对于 USD/CHF = 0.0005 和 USD/JPY = 0.05).

步骤 1 – 编写智能交易的描述



在智能交易的导航窗口处，点击鼠标右键并且选择在菜单中的“创建新智能交易”。创建智能交易的初始提醒将会询问你进入数据中心。在显示窗口，填写智能交易的名称(名称) – MACD Sample, 作者(作者) –指出你的名字, 链接 (连接) –你网页的链接，注解 (注解) – MACD-基本智能交易的测试范例。

步骤 2 – 创建程序的基本结构

测试智能交易的源代码将只占据一点位置，但是还是有些量经常很难抓住，特别是我们不是专业的编成工作者 – 另外，我们不需要这些描述，不是吗？ :)

一个标准智能交易的结构构想，让我们看看以下部分的描述:

1. 初始变量
2. 初始数据检测
    - 检测图表,图表中的柱数
    - 检测外部变量值:标准手, S/L, T/P, T/S
3. 对于快速数据通道设置内部变量
4. 检测交易终端 – 是无效的吗？如果是:
    - 检测: 账户上的可用保证金...
    - 可能是看涨仓位 (BUY)?
        - 开设看涨仓位并退出

- 可能是卖空仓位 (SELL)?
  - 开设卖空仓位并退出

退出智能交易...

- 周期循环检验先前开仓
  - 如果是看涨仓位
    - 应该平仓?
    - 应该重新设定追踪止损?
  - 如果是卖空仓位
    - 应该平仓?
    - 应该重新设定追踪止损?

返回的结果很简单，只有 4 种。

现在让我们尝试一步一步地区完成列出的计划：

3. 初始变量

所有使用在智能交易程序中的变量必须按照 MetaQuotes Language 4 要求的指定。这就是为什么我们在程序的开始插入初始变量的原因

```
extern double TakeProfit = 50;
extern double Lots = 0.1;
extern double TrailingStop = 30;
extern double MACDOpenLevel=3;
extern double MACDCloseLevel=2;
extern double MATrendPeriod=26;
```

MetaQuotes 语言 4 是需要“外部变量”辅助的。外部变量可以从外部设定，在智能交易程序源代码设定之后不可以修改。提供一个额外的灵活性。在我们的程序中，MATrendPeriod 变量作为外部变量指定。 在程序开始我们插入这个变量。

```
extern double MATrendPeriod=26;
```

4. 检测初始数据

该代码部分通常使用在所有的智能交易中。因为是一个标准的检测：

```
// 初始数据检测
// 确认智能交易运行正常非常重要
//图表和用户设置不能出现任何错误
// 变量(Lots, StopLoss, TakeProfit,
```

```
// TrailingStop) 我们的情况需要检测 TakeProfit
// 图表中少于 100 柱
   if(Bars<100)
     {
      Print("少于 100 柱");
      return(0);
     }
   if(TakeProfit<10)
     {
      Print("赢利少于 10");
      return(0);  // 检测 TakeProfit
     }
```

5. **对于数据的快速通道设置内部变量**

在源代码中经常需要注意指标值或计算值。简化代码和数据放置在内部变量中。

```
6. int start()
7.   {
8.    double MacdCurrent, MacdPrevious, SignalCurrent;
9.    double SignalPrevious, MaCurrent, MaPrevious;
10.   int cnt, ticket, total;
11.
12.// 简化代码
13.//数据放置在内部变量中
14.
   MacdCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_M
   AIN,0);
15.
   MacdPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_
   MAIN,1);
16.
   SignalCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE
   _SIGNAL,0);
17.
   SignalPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MOD
   E_SIGNAL,1);
18.
   MaCurrent=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE
   _CLOSE,0);

   MaPrevious=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRIC
   E_CLOSE,1);
```

现在,用
iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0)代替，您可以在源代码中使用 MacdCurrent。

19. **检测交易终端 – 是空的吗?如果是:**
在我们的智能交易中，我们仅使用开单和操作挂单。不过，使更安全，我们来认识一种 对于先前定单交易终端检测:

```
total=OrdersTotal();
if(total<1)
  {
```

- **检测: 账户上的可用保证金...**
  在分析市场状况之前,检测你的账户上可用的自由保证金可以开仓。

```
      if(AccountFreeMargin()<(1000*Lots))
        {
         Print("没有资金.自由保证金 = ",
AccountFreeMargin());
         return(0);
        }
```

- **可能是看涨仓位 (BUY)?**
  进入看涨仓位的条件:MACD 低于零， 向上并且穿过信号线向下。这就是我们在 MQL4 中描述的 (注意我们在指标上的业务值保存在先前的变量中):

```
      // 尽可能检测看涨仓位 (BUY)
      if(MacdCurrent<0 &&
MacdCurrent>SignalCurrent &&
         MacdPrevious

MathAbs(MacdCurrent)>(MACDOpenLevel*Point) &&
         MaCurrent>MaPrevious)
        {

ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0
,Ask+TakeProfit*Point,
                              "macd
sample",16384,0,Green);
         if(ticket>0)
           {
```

```
if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_T
RADES))
                Print("BUY 开单 :
",OrderOpenPrice());
              }
          else Print("错误 opening BUY order :
",GetLastError());
          return(0);
         }
```

附加的检验‘山丘’的大小上面已经给出了描述。MACDOpenLevel 变量是一个用户指定变量，它不可能改变程序文本，但是却有很大的灵活性。在程序开始我们插入这个变量的描述。

- **可能是卖空仓位(SELL)？**
  进入卖空仓位的条件：MACD 高于零，向上并且穿过信号线向下。注解如下：

```
       // 尽可能的检测卖空仓位(SELL)
       if(MacdCurrent>0 &&
MacdCurrentSignalPrevious &&
          MacdCurrent>(MACDOpenLevel*Point) &&
MaCurrent
         {

ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,3,
0,Bid-TakeProfit*Point,
                              "macd
sample",16384,0,Red);
          if(ticket>0)
            {

if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_T
RADES))
                Print("SELL 开单 :
",OrderOpenPrice());
              }
          else Print("错误 SELL 定单开仓 :
",GetLastError());
          return(0);
         }
 return(0);
}
```

20. **周期循环检验先前开仓**

```
//进入市场的正确性非常重要
// 但是更重要的是安全退出...
for(cnt=0;cnt
  {
   OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
   if(OrderType()<=OP_SELL &&   // 检验开仓
      OrderSymbol()==Symbol())  // 检验货币对
     {
```

"cnt" - " 是一个循环的变量必须在程序开始指定如下:

```
 int cnt = 0;
```

- 如果是看涨仓位

```
   if(OrderType()==OP_BUY)    // 打开看张仓位
     {
```

- **应该平仓吗?**
  退出看涨仓位的条件: MACD 穿过信号线，MACD 高于零，向上并穿过信号线向下。

```
   if(MacdCurrent>0 &&
   MacdCurrentSignalPrevious &&
      MacdCurrent>(MACDCloseLevel*Point))
     {

   OrderClose(OrderTicket(),OrderLots(),Bi
   d,3,Violet); //平仓
      return(0); //退出
     }
```

- **应该重设追踪止损马?**
  我们设定追踪止损只有在仓位盈利已经超过追踪水平点，并且新的止损水平点好于先前的水平。

```
   // 检测追踪止损
   if(TrailingStop>0)
     {
```

```
if(Bid-OrderOpenPrice()>Point*TrailingS
top)
      {
       if(OrderStopLoss()
         {

OrderModify(OrderTicket(),OrderOpenPric
e(),Bid-Point*TrailingStop,

OrderTakeProfit(),0,Green);
          return(0);
         }
      }
  }
```

我们停止操作符。

```
   }
```

- 如果是卖空仓位

```
else //卖空仓位
  {
```

  - **应该平仓吗?**
    退出卖空仓位的条件:MACD 穿过信号线,MACD 低于零，向上并且穿过信号线向下。

```
if(MacdCurrent<0 &&
MacdCurrent>SignalCurrent &&
   MacdPrevious(MACDCloseLevel*Point))
  {

OrderClose(OrderTicket(),OrderLots(),As
k,3,Violet); //平仓
   return(0); // 退出
  }
```

  - **应该重设追踪止损吗?**
    我们设定追踪止损只有在仓位盈利已经超过追踪水平点,并且新的止损水平点好于先前的水平。

```
// 检测追踪止损
if(TrailingStop>0)
```

```
   {

if((OrderOpenPrice()-Ask)>(Point*Trailingstop))
     {

if((OrderStopLoss()>(Ask+Point*TrailingStop)) || (OrderStopLoss()==0))
        {

OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*TrailingStop,

OrderTakeProfit(),0,Red);
         return(0);
        }
     }
  }
```

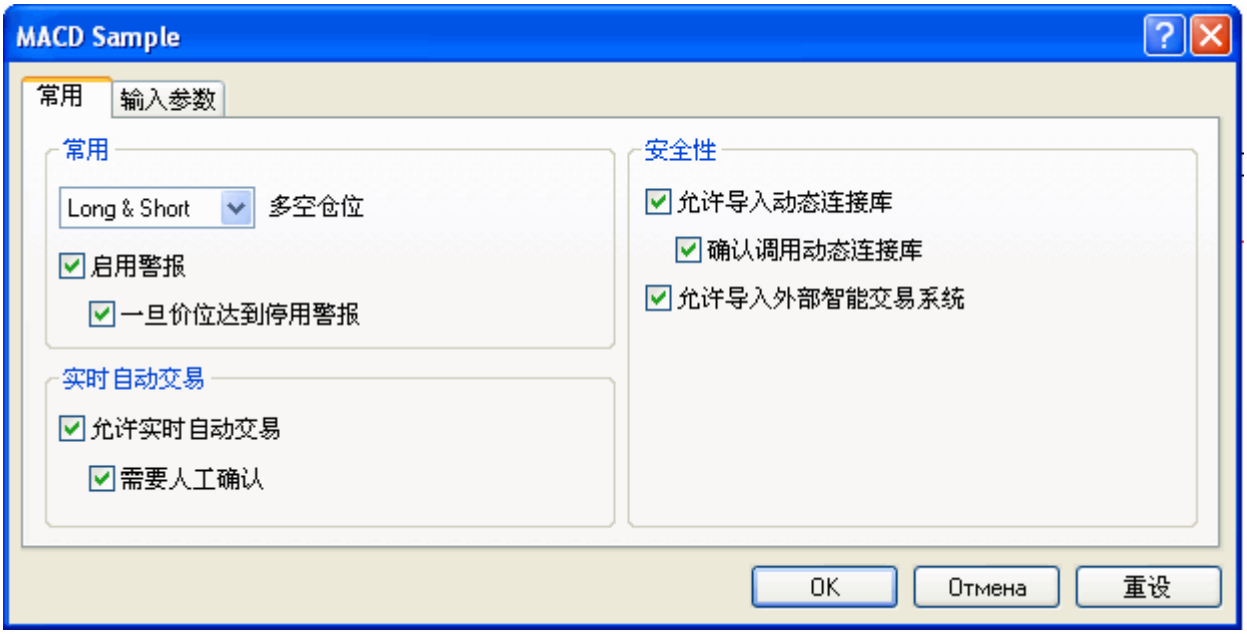
关闭所有残留开仓。

```
        }
      }
   }
 return(0);
}
```

这样一步一步地编写我们的智能交易...

步骤 3 - 集中程序的结果代码

让我们打开智能交易的设定：使用按钮打开"属性..."菜单。在窗口内指定运行参量的外部设定:



从先前部分集中全部代码：

```
//+------------------------------------------------------------------
+
//|                                                    MACD Sample.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/
|
//+------------------------------------------------------------------
+
extern  double  TakeProfit = 50;
extern  double  Lots = 0.1;
extern  double  TrailingStop = 30;
extern  double  MACDOpenLevel=3;
extern  double  MACDCloseLevel=2;
extern  double  MATrendPeriod=26;

//+------------------------------------------------------------------
+
//|
|
//+------------------------------------------------------------------
+
int  start()
  {
   double  MacdCurrent, MacdPrevious, SignalCurrent;
   double  SignalPrevious, MaCurrent, MaPrevious;
```

```
   int  cnt, ticket, total;
// 检测初始化数据
// 确定智能交易在图表中运行正常非常重要
// 用户在外部变量交易中不会产生任何错误
// 外部变量 (标准手数, 止损,赢利,
// 追踪止损) 在这种情况下，我们检测图表中赢利水平要小于 100 柱
   if(Bars<100)
     {
      Print("少于 100 柱");
      return(0);
     }
   if(TakeProfit<10)
     {
      Print("赢利少于 10");
      return(0);  // 检测赢利水平
     }
// 简化代码和加速通道
// 数据被放置在内部变量中
   MacdCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0);
   MacdPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,1);
   SignalCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0);
   SignalPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,1);
   MaCurrent=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,0);
   MaPrevious=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,1);
   total=OrdersTotal();
   if(total<1)
     {
      // 没有指定开单
      if(AccountFreeMargin()<(1000*Lots))
        {
         Print("没有资金. 自由保证金 = ", AccountFreeMargin());
         return(0);
        }
      // 尽可能检测看涨仓位 (BUY)
  if(MacdCurrent<0 && MacdCurrent>SignalCurrent &&
MacdPrevious<SignalPrevious &&
         MathAbs(MacdCurrent)>(MACDOpenLevel*Point) &&
MaCurrent>MaPrevious)
        {

ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,"m
acd sample",16384,0,Green);
         if(ticket>0)
           {
```

```
if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))  Print("BUY 定单开仓 : ",OrderOpenPrice());
           }
         else  Print("错误 BUY 定单开仓 : ",GetLastError());
         return(0);
        }
      // 尽可能检测卖空仓位(SELL)
      if(MacdCurrent>0 && MacdCurrent<SignalCurrent && MacdPrevious>SignalPrevious &&
         MacdCurrent>(MACDOpenLevel*Point) && MaCurrent<MaPrevious)
        {

ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,"macd sample",16384,0,Red);
         if(ticket>0)
           {

if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))  Print("SELL 定单开仓 : ",OrderOpenPrice());
           }
         else  Print("错误 SELL 定单开仓 : ",GetLastError());
         return(0);
        }
      return(0);
     }
   // 正确进入市场很重要,
   // 但正确退出市场更重要...
   for(cnt=0;cnt<total;cnt++)
     {
      OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
      if(OrderType()<=OP_SELL &&   // 检测开仓
         OrderSymbol()==Symbol())  // 检测货币对
        {
         if(OrderType()==OP_BUY)  // 看涨仓位开仓
           {
            // 需要平仓吗?
            if(MacdCurrent>0 && MacdCurrent<SignalCurrent && MacdPrevious>SignalPrevious &&
               MacdCurrent>(MACDCloseLevel*Point))
              {
               OrderClose(OrderTicket(),OrderLots(),Bid,3,Violet); //平仓
               return(0); // 退出
```

```
                }
            // 检测追踪止损
            if(TrailingStop>0)
              {
               if(Bid-OrderOpenPrice()>Point*TrailingStop)
                 {
                  if(OrderStopLoss()<Bid-Point*TrailingStop)
                    {

OrderModify(OrderTicket(),OrderOpenPrice(),Bid-Point*TrailingStop,Ord
erTakeProfit(),0,Green);
                     return(0);
                    }
                 }
              }
           }
         else  // 去卖空仓位
           {
            // 需要平仓吗?
            if(MacdCurrent<0 && MacdCurrent>SignalCurrent &&
               MacdPrevious<SignalPrevious &&
MathAbs(MacdCurrent)>(MACDCloseLevel*Point))
              {
               OrderClose(OrderTicket(),OrderLots(),Ask,3,Violet); //
平仓
               return(0); //退出
              }
            // 检测追踪止损
            if(TrailingStop>0)
              {
               if((OrderOpenPrice()-Ask)>(Point*TrailingStop))
                 {
                  if((OrderStopLoss()>(Ask+Point*TrailingStop)) ||
(OrderStopLoss()==0))
                    {

OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*TrailingStop,Ord
erTakeProfit(),0,Red);
                     return(0);
                    }
                 }
              }
           }
```

```
      }
    return(0);
  }
// 结束
```

对于最后智能交易的确认，只需要指定外部变量值 "Lots = 1","Stop Loss (S/L) = 0" (not used), "Take Profit (T/P) = 120" (appropriate for one-hour intervals), "Trailing Stop (T/S) = 30". 当然，你可以使用自己的值。按 "编写"按钮，如果没有任何错误信息出现 (你可以从 MetaEditor 的列表中复制), 按 "保存"键保存智能交易