MetaTrader 4 平台
MetaQuotes Language 4 (MQL4)
编程参考

（简体中文 第 1 版）

严泽平　修订

MetaQuotes SOFTWARE

2008. 04

前　言

MetaQuotes (迈达克)软件公司是全球外汇、CFD 和期货在线交易软件的领导厂商之一。该公司一直致力于在线金融交易软件的研制和专业开发，MetaTrader 4 交易平台（MT4）就是该公司在此领域多年研究成果，全球金融机构使用其开发的 MT4 交易平台可以为客户提供专业的、高质量的在线交易服务。目前，全球有 100 余家经纪公司和银行采用了 MT4 交易平台向客户提供在线金融交易服务，国内民生银行也选用了 MT4 作为外汇交易平台。

MT4 交易平台是 MetaQuotes 公司专业开发的产品，它比世界上那些大型经纪公司或银行自研的面向普通客户的交易平台更专业、运行更稳定、功能更强悍，深受朋友们喜爱、好评，特别是其内置的面向高端客户的 MQL4 编程语言，强大的编程能力无人能与之比肩，是 MT4 平台最大特色之一。

MetaQuotes Language 4 (MQL4) 是 MT4 平台客户端程序内置的交易策略编程语言。这种语言允许用户创建自己的智能交易程序(EA)，使自己的交易操作能够自动地执行，此外，用户还可以使用 MQL4 创建自定义指标、脚本和库。

MQL4 编程语言包含了大量可以分析当前数据及历史数据所必备的函数，像大多数计算机程序设计语言一样，它也包含了多种数据类型、各种运算符、预定义常量、数组、控制语句和丰富的数据处理函数，内置了一些基本的技术指标和用户定单管理命令等。

集成在客户端的 MetaEditor 4(ME4 文本编辑器)是编写 MQL4 程序源码的集成开发环境。ME4 支持语法高亮显示，带有在线帮助系统，用户可在客户端内利用历史数据模拟运行程序，自行评估程序运行效果。

鉴于 MT4 平台在国内已有应用，在国际上已有百余家公司采用，国内炒汇朋友不可避免要用到该平台，学习和应用 MQL4 语言就是高手们抢占先机、走向辉煌的必经之路。MQL4 语法类似于 C 语言，而 C 语言又是国内院校基本教学语言，已开设多年，国内应有大批朋友熟悉其语法结构，可谓驾轻就熟，易学易用，稍有编程修养的朋友经过短期学习和钻研，应该能够很快掌握，熟练应用，因此，MQL4 特别适合会编程的炒汇高手实现自己的交易策略，或向他人提供自己开发

的智能交易程序。今后，它还可能为你成就一番事业，创造一份新的高收入职业，前景可期。或许，下一个汇市传奇就是你创造的，你的故事会让后人久久津津乐道，无限崇敬。

由于迈达克公司进入中国市场时间不长，在上海设有分公司，但本地化工作似乎做得不够理想。作为 EA 编程必读资料，迈达克公司网站在线编程参考资料分别提供了俄语、英语和中文版本，其中中文版资料大量采用了机器翻译，再辅以人工翻译，可读性较差，用词、用语不太符合国人阅读习惯，技术指标和术语翻译有违内地惯例，甚至有些字、词、句翻译错误，这给国内朋友学习和应用带来了不便之处。本人在学习中也深受其扰，参照其英文版和中文版，本人利用工作之余一边学习，一边重新对其进行了翻译、整理和校对，对个别明显与上下文语境和语义不符的地方，借助俄语到英语翻译软件进行了复核，经过近一个月断断续续地努力，总算整理成文，希望对大家有用，也算是本人为推动 MT4 平台在国内的应用尽点绵薄之力。

本人觉得翻译 MQL4 编程参考资料真是不太容易。译者需要有良好的英语基础、程序设计语言知识(尤其是 C/C++语言)、股票和外汇基本理论才能得心应手，缺一不可。翻译这种跨专业的外文资料，特别是涉及到各领域的概念、术语、技术指标名称和惯用语等要符合国人阅读习惯、使用习惯和专业习惯，其实并非易事。本人正巧具备了这几个方面知识，就信手为大家做点工作吧，对我来说，既是一次挑战，也是一次锻炼，更是一段学习的历程。

在这里，还是要感谢 MetaQuotes 公司给我们提供了业界领先的软件，期待着支持面向对象编程的 MQL5 早日面世，感谢原中文版作者辛勤劳动。

限于水平和时间，文中可能还有不当之处，望各位朋友见谅并提宝贵意见。欢迎来信斧正，本人将在第二版中收录、致谢，继续维护本文档。

有梦才会有追求，愿大家都发财！

严泽平

2008-4-29 于磬园校区，一座美丽的校园

目 录

MQL 简介

MetaQuotes Language 4 (MQL4) 是一种新型的内置型交易策略编程语言。这种语言允许用户创建自己的智能交易程序，使自己的交易操作能够自动地执行，尤其适合用户实现自己的交易策略。除此之外，用户还可以使用 MQL4 创建自定义指标、脚本和库。

MQL4 包含了大量可以分析当前数据及历史数据所必备的函数，也包含了基本的算术运算符和逻辑运算符，内置了一些基本的技术指标和定单管理命令。

MetaEditor 4(文本编辑器)用于编写 MQL4 程序源码，支持语法高亮显示，帮助用户得心应手地编写智能交易程序。我们把 MQL 语言字典当作 MQL4 语言的帮助系统，这是一部简化的指南，却包含了我们可能用到的各种功能，类目、运算符、保留字和其它的语言成份都已分门别类，我们能够从中找到每一语言成份的说明。

MQL4 可以编写出不同功能、不同目的的程序：

- 智能交易

  智能交易是一种附加到特定图表的自动交易系统(MTS)。对于指定的货币对，它随着收到的每一跳(tick)数据自动运行，如果此时智能交易还在忙着处理前一跳数据的话(也就是说智能交易还未完成它的操作)，就不会再次被启动了。这种交易系统能够在提醒用户可以交易的同时，自动将交易定单直接送到交易服务器。与大多数交易系统一样，客户端支持用历史数据测试=交易策略，并在图表上显示买入和卖出点。

  智能交易存储在 terminal_directory\experts 目录。

- 自定义指标

  自定义指标是指那些已集成在客户端的指标之外用户独自编写出的新的技术指标。和内置的技术指标一样，不能用来自动交易，只能作为分析数据的工具。

  自定义指标储存在 terminal_directory\experts\indicators 目录。

- 脚本

  脚本是执行特定功能的一段程序。和智能交易不同，脚本不能单独运行，只能被调用。

  脚本存储在 terminal_dictionary\experts\scripts 目录。

- 库

  库是经常使用的自定义函数集合。库中脚本不能单独运行。

  库存储在 terminal_directory\experts\libraries 目录。

- 包含文件

包含文件常用于在源码中引用程序块。在编译阶段，这些文件能够被包含在智能交易、脚本、自定义指标和库的源码中。使用包含文件比调用库更优越，因为调用库会导致额外开销。

建议包含文件储存在 terminal_directory\experts\include 目录。

基础

MetaQuotes Language 4 (MQL4)是一种新型的内置型交易策略编程语言。这种语言允许用户自行编写智能交易程序，自动化完成交易管理，特别适合用户实现自己的智能交易策略。此外，用户还能创建自己的技术指标(自定义指标)、脚本和库。

语法

MQL4 语法类似于 C 语言，除了下列这些特点:'

- 没有地址运算符；'
- 没有 do ...=while 语句；'
- 没有 goto ...=语句；'
- 没有 [条件][表达式 1]:[表达式=2]=语句；'
- 没有复合数据类型 (结构)；'
- 不允许复合赋值，例如：val1=val2=0； arr[i++]=val；'
  cond (cnt=OrdersTotal)>0；=等等；'
- 不可以中途终止逻辑表达式的计算。

- 注释=

多行注释以 /*=符号开始，以 */ 符号结束，不能嵌套使用。单行注释以 // 符号开始，到行末结束，它可以被嵌套到多行注释之中。注释允许空行和空白。

示例:=

```
// 单行注释=
/*  multi-'
    line          // 嵌入单行注释'
    comment
*/
```

- 标识符

标识符用于命名变量、函数和数据类型，其长度不能超过 31 个字符。标识符可以使用下列符号:数字 0-9、大小写英文字母 a-z 和 A-Z=(区分大小写字母)，还有下划线(_)。标识符的首字符不可以是数字，也不能使用保留字。

示例:=

```
NAME1 name1 Total_5 Paper
```

- 保留字

下面列出的标识符是系统自用的保留字，每个保留字都有其特定含义，用户不能随意使用。

数据类型	储存类型	操作符	其他
bool=	Extern	Break	False
color	Static	Case	True
datetime		Continue	
double		Default	
int		Else	
string		For	
void		If	
		Return	
		Switch=	
		While	

数据类型

任何程序都要依靠数据来运行，数据因目的不同可以有不同的类型。比如，数组可以存取整型数据，价格可以用浮点型双精度数表示。在 MQL 4 中没有专门用来表示货币值的数据类型。

不同的数据类型有不同的处理速度，整型数据是最快的。处理双精度的数据需要使用特殊的协处理器，所以，处理浮点型数据比较复杂，它比处理整型数据要慢一些。字符串处理速度是最慢的，因为它要分配/重分配计算机动态内存。

主要的数据类型如下:=

- 整型数据 (int)
- 布尔型数据 (bool)

- 字符型数据 (char)
- 字符串型数据 (string)
- 浮点型数据 (double)
- 颜色数据 (color)
- 日期时间数据 (datetime)

color 型和 datetime 型仅仅是为了让我们更清楚地区分图表内容和输入这些参数时才有意义，这些参数可在智能交易的属性标签和自定义指标“输入”=标签中设置。color 和 datatime 型数据用整数来表示。int 和 double 型统称为数值型。

在表达式运算中使用了隐式类型转换(自动转换)。

- 类型转换=

在 MQL4 表达式中使用了隐式类型转换。由低到高转换类型的优先级:=

```
int (bool,color,datetime); =
double; =
string; =
```

在运算完成之前(除数据已被定义的)，数据自动转换为高优先级类型。当赋=值操作完成时，数据会转换成目标数据类型。

示例:=

```
int    i   1 / 2; =   // 没有类型转换，结果为 0
int    i   1 / 2.0; =  // 表达式转换成双精度型，再转成整型，结果为 0
double d   1.0 / 2.0; // 没有类型转换，结果为 0.5
double d   1 / 2.0; = // 表达式转换成双精度型，和目标类型一样，结果为 0.5
double d   1 / 2; =   // 整型表达式转成双精度型，结果为 0.0
string s   1.0 / 8;=  //表达式转成双精度型,再转成字符串型，结果为 "0.12500000" (含 10 个字符)
string s   NULL;=  // 整型常量转成字符串型,结果为 "0" (含 1 个字符)
string s   "Ticket #"+12345;=// 表达式转成字符串型，与目标类型相同,=结果为"Ticket #12345"=
```

类型转换不但运用在常量中，还被运用在相应的变量中。

- 整型常量

十进制:=数字 0 -=9=，包括正负号。

示例:=

```
12, 111,=-956 1007=
```

十六进制:=数字 0 - 9 ,=字符 a - f=或者 A - F 代表 10 - 15；=以 0x=或者 0X=开头。

示例:=

```
0x0A, 0x12, 0X12, 0x2f, 0xA3, 0Xa3, 0X7C7=
```

整型数据占用 4 字节内存空间，其数值范围介于 -2147483648 ～= 2147483647=之间。如果超出这个范围，则视为无效。

- 字符常量

任何用单引号括起来的单个字符或者类似 '\x10'形式的十六进制 ASCII=码都是整型的字符常量。字符常量中包含的一些特殊字符，如单引号(')、双引=号(")、问号(?)、反斜杠(\)和控制符，必须以反斜杠开头(\)转义表示，如下表所示：

换行	NL (LF)	\n
制表符	HT	\t
回车	CR=	\r
反斜线	\=	\\=
单引号=	'=	\'=
双引号=	"=	\"=
十六进制 ASCII=	hh=	\xhh=

如果上述字符不使用反斜杠引导，结果将不被定义:=

```
int a   'A';
int b=  '$'; =
int c   '©'; =      // 代码 0xA9=
int d   '\xAE'; =  // 符号®代码
```

字符数据占用 4 字节内存空间。其数值范围介于 0 ～ 255 之间。如果超=出这个范围，则视为无效。

- 布尔型常量

布尔型常量用来表示真值=true 和假值 false。它还可以用数字 1 和 0 表示。True 和 TRUE，False 和 FALSE 都可使用(字母不区分大小写)。

示例:=

```
bool a   true; =
bool=b=  false; =
```

```
bool=c    1; =
```

布尔型常量占用长度为 4 字节的内存空间，它用 0 或 1 的值表示。

- 浮点常量(双精度常量)

浮点型常量由整数部分、小数点(.)和小数部分组成，其中整数部分和小数部分是一组十进制数字。

示例:=

```
double a    12.111; =
double b=   -956.1007; =
double c    0.0001; =
double d    16; =
```

浮点型数据 (双精度)占用 8=字节的内存空间。其数值范围介于 $-1.7e^{-308}$ ～ $1.7e^{308}$ 之间。如果超出这个范围，结果则视为无效。

- 字符串常量

字符串常量是用双引号括起来的一连串 ASCII=字符，如: "Character constant"。

字符串常量是双引号里一组字符，它是字符串型数据。如果需要在字符串中插入一个双引号("), 在它前面必须使用反斜杠(\)。如果有前置反斜杠(\)，任=何特殊字符都能用在字符串中。字符串长度介于 0 到 255 个字符之间，如果超=过这个长度，尾部多余的字符将被忽略，编译器也会有相应的提示。

示例:=

```
"This is a character string"=
"Copyright symbol=\t\xA9"=
"this line contains a line feed symbol=\n"=
"C:\\Program Files\\MetaTrader 4"=
"A"="1234567890"="0"="$"=
```

字符串常量占用 8=个字节的内存空间。其中第一部分为长整型，存储字符串缓冲区的长度，第二部分是存储字符串缓冲区的 32=位地址。

- 颜色常量

颜色常量可以用三种方法表示： 用字符表示、用整数表示或用颜色名(只能是已命名的 Web 颜色)。

用字符表示的方法是用三个数字来表示三种主要颜色分量的比例：红、绿、蓝。 颜色常量以 C 开头，并用单引号括起来。每种颜色分量所占的比例在 0 ～= 255 之间。

用整数表示的方法是以十六进制或十进制数字的形式写出颜色值。十六进制数字看起来像 0x00RRGGBB=这样，其中 RR=是红色分量的比例，GG=是绿色分量的比例，BB=是蓝色分量的比例。十进制数不能直接体现红、绿、蓝的比例，而是十六进制整数的十进制表示形式。

特殊的颜色名可以参阅所谓的 Web 颜色表。

示例:

```
// 用字符表示数据
C'128,128,128'=    // 灰色
C'0x00,0x00,0xFF'=// 蓝色
// 用颜色名表示
Red
Yellow=
Black
// 用整数表示
0xFFFFFF           // 白色
16777215           // 白色
0x008000           // 绿色
32768=             // 绿色
```

颜色常量占用 4 字节存储空间。第一个字节一般被忽略，后三个字节包含了红、绿、蓝的组成信息。

- 日期时间常量

日期时间常量可以当作一个文本行来表示，它由 6=个部分组成：年、月、日、时、分、秒，以 D 开头，并用单引号括起来，其中日期(年、月、日)、时间(时、分、秒)、甚至两者都可以同时省略。日期时间型常量变化范围= 1970.1.1 - 2037.12.31。

示例:=

```
D'2004.01.01 00:00'=     // 新年
D'1980.07.19 12:30:27'=
D'19.07.1980 12:30:27'=
D'19.07.1980 12'=        //等于  D'1980.07.19 12:00:00'=
D'01.01.2004'=           //等于  D'01.01.2004 00:00:00'=
D'12:30:27'=             //等于 D'[编译日期] 12:30:27'=
D''=                     //等于 D'[编译日期] 00:00:00'=
```

日期时间常量在内存中表示成 4 字节长整型数，其值是从 1970 年 1 月 1 日 00:00 开始以来的总秒数。

运算符和表达式=

某些字符和字符的组合是特别重要的，它们被称为运算符，例如:=

```
+ - *=/ %=     算术运算符
&&=||=         逻辑运算符
  +   *=       赋值运算符
```

运算符通常用在表达式中，当合适的操作数提供给他们时才有意义。

需要特别注意标点符号，如圆括号、方括号、逗号、冒号、分号。

运算符、标点符号、空格用来分割语句的不同部分。

- 表达式=

一个表达式由一个或多个操作数和运算符组成，它们可以分几行书写。

示例:=

```
a++;  b=  10; =
x=  (y=*=z) /
    (w=+ 2) + 127; =
```

以分号(;)结束的表达式是一个操作符。

- 算术运算符

算术运算符包括加法和乘法运算符:=

```
求和                    i   j=+ 2; =
求差                    i   j=-=3; =
改变符号=               x=  -=x; =
求积=                   z=  3=*=x; =
求商                    i   j=/ 5; =
求模                    minutes   time %=60; =
自增=                   i++; =
自减=                   k--; =
```

自增/自减运算符不能用在表达式中。

示例:=

```
int a=3; =
a++; =              // 合法表达式=
int b=(a++)*3; =    // 非法表达式=
```

- 赋值运算符

  表达式的值赋给左边的变量。

```
把变量 x=的值赋给变量 y=                    y    x; =
```

下列表达式中赋值运算符结合了算术运算符或位运算符:=

```
y 值加上 x                          y=+  x; =
y=值减去=x=                         y=-= x; =
y=值乘以 x=                         y=*= x; =
y=值除以 x=                         y=/  x; =
求=y/x=模                           y=%= x; =
y=值逻辑右移=x=位                   y=>>= x; =
y=值逻辑左移=x=位                   y=<<= x; =
按位与运算                          y=&= x; =
按位或运算                          y=|= x; =
x=和 y=逻辑异或                     y=^= x; =
```

表达式只能有一个赋值运算符。位运算符只能用于整型数据。逻辑移位运算符中 x=值只能是小于 5 位的二进制数，过大的数值将会被拒绝，所以移动范围只能是 0 到 31 位。用 %= 运算符求模 (求 y/x=的模)，其结果符号与被除数相同。

- 关系运算符

  整数 0 代表逻辑假值=FALSE，任何非零的值代表逻辑真值=TRUE。

  用关系运算符或逻辑运算符组成的表达式值是 0(False)或 1(True)。

```
a 等于 b,结果为 true                a    b; =
a 不等于 b,结果为 true              a != b; =
a 小于 b,结果为 true                a<=b; =
a 大于 b,结果为 true                a >b; =
a 小于等于 b,结果为 true            a <=b; =
a 大于等于 b,结果为 true            a >= b; =
```

2=个浮点型数值不能直接用 或 != 运算符进行比较，这就是为何要把两数相减，并将所得结果与 null=进行比较的原因。

- 布尔运算符

逻辑非运算符(!)的操作数必须是算术类型。如果运算值是 FALSE (0)，其非运算的结果为 TRUE (1)；如果运算值不同于 FALSE (0)，其非运算的结果等于 FALSE (0)。

```
if(!a) Print("不是 'a'"); =
```

x=和 y 值的逻辑或(||)运算。如果 x=或 y 值有一个为真 true(1)，表达式=值为 TRUE (1)，否则，值为 FALSE (0)。逻辑表达式采用整体计算，即不采用所谓的“简估”方法。

```
if(x<0 ||=x>=max_bars) Print("超出范围"); =
```

x=和 y 值的逻辑与“&&”运算。如果 x=和 y 值都是真 true(1)，表达式值=为 TRUE (1)。否则,=结果为 FALSE (0)。

```
if(p!=x=&&=p>y) Print("TRUE"); =
```

- 位运算符

取反运算符(~)对变量值二进制代码按位取反。表达式值中所有 1 变为 0，0 变为 1。

```
b   ~n; =
```

将二进制代码表示的 x=逻辑右移 y=位。右移是逻辑运算，即左侧用零填=满。

```
x=  x >> y; =
```

将二进制代码表示的 x=逻辑左移 y 位，右侧(低位)用零填满。

```
x=  x << y; =
```

二进制表示的 x=和 y=位运算与(&)。如果 X=和 y 两个相应的二进制位都为 1，则该位的结果为 1，否则为 0。

```
b=  ((x & y) != 0); =
```

二进制表示的 x=和 y=位运算或(|)。如果 x=和 y 两个相应的二进制位都为 0，则该位的结果为 0，否则为 1。

```
b=  x | y;
```

二进制表示的 x=和 y=位运算异或(^)。如果 x=和 y 两个相应的二进制位相=异，则该位的结果为 1，否则为 0。

```
b=  x ^ y;
```

位运算符只适合整型。

- 其它运算符

下标

数组的第 i 个元素的序号，i 可以为整数或整型表达式。

示例:

```
array[i] = 3;  //将 3 赋给数组的第 i 个元素。
```

只有整数才能成为数组下标。只允许使用四维或四维以下的数组。每维索引是从 0 到每维大小-1。 举一个例子，一个有 50 个元素组成的一维数组，引用第一个数组元素用 array[0]，这样，最后一个数组元素将是 array[49]。

如果访问数组超出了数组下标范围，系统将会出错 ERR_ARRAY_INDEX_OUT_OF_RANGE(下标超过范围),通过调用 GetLastError() 函数，获取这些错误信息。

调用 x1,x2,...,xn 形式参数的函数

每个形参可以是一个常量、一个变量和相应类型的表达式。传递的参数用逗号隔开，并放在圆括号()内，函数名放在圆括号()前。

用此函数返回表达式值。如果函数返回值定义为 void 型，这些函数调用就不能放在赋值号右边。请确认表达式 x1,x2,...,xn 是按序执行的。

示例:

```
double SL=Bid-25*Point;
int
ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,SL,Ask+25*Point,"My
comment",123,0,Red);
```

逗号运算符

用逗号分隔的表达式是按从左到右的顺序执行。所有表达式都是先算左边的，再算右边的，其结果的类型和值由右边表达式决定。函数的参数表也可视为一个范例(参见上面)。

示例:

```
for(i=0,j=99;  i<100;  i++,j--) Print(数组[i][j]);
```

- 优先级规则

下表中同组运算符优先级相同。排在前面的运算符优先级高，排在后面的运算符优先级低。运算符按优先级顺序从高到低排列如下：

()	函数调用	从左到右
[]	引用数组元素	
!	非运算符	从右到左
-	改变符号	
++	自增	
--	自减	
~	“按位取反”位运算符	
&	“按位与”位运算符	从左到右
|	“按位或”位运算符	
^	“按位异或”位运算符	
<<	逻辑左移	
>>	逻辑右移	
*	乘法	从左到右
/	除法	
%	求模	
+	加法	从左到右
-	减法	
<	小于	从左到右
<=	小于等于	
>	大于	
>=	大于等于	
==	等于	
!=	不等于	
||	逻辑或	从左到右
&&	逻辑与	从左到右
=	赋值	从右到左
+=	加法复合赋值运算符	
-=	减法复合赋值运算符	
*=	乘法复合赋值运算符	
/=	除法复合赋值运算符	
%=	求模复合赋值运算符	
>>=	右移复合赋值运算符	
<<=	左移复合赋值运算符	
&=	按位与复合赋值运算符	
|=	按位或复合赋值运算符	
^=	按位异或复合赋值运算符	
,	逗号	从左到右

高优先级的圆括号可以改变运算符的执行顺序。

注意:在 MQL4 程序中，运算符的优先级不同于 C 语言。

控制语句

语句描述了完成一项任务的算法操作规则。程序本身就是这样的语句序列。一条条语句之间以分号；分隔。

一条语句能占一行或几行。二个或更多语句也能书写于同一行上。控制执行顺序的语句(if、if-else、 switch、while 和 for) 可以相互嵌套。

示例:

```
if(Month() == 12)
  if(Day() == 31) Print("新年快乐!");
```

- 语句块

一个复合语句(一个语句块)可以由一条语句组成，也可以由大括弧{}括起来的一条或多条任意类型的语句组成。大括弧{}后面不必跟着分号(；)。

示例:

```
if(x==0)
  {
   Print("无效位置 x=",x);
   return;
}
```

- 表达式操作符

任何以分号(；)结束的表达式都被视为是一个操作符。这里罗列了一些表达式操作符的范例:

赋值运算符:

```
Identifier=expression;
x=3;
y=x=3;  // 错误
```

赋值运算符在表达式操作符中只能用一次。

函数调用操作符:

```
Function_name(argument1,..., argumentN);

FileClose(file);
```

空语句操作符

仅仅只有一个分号(；)组成，用来表示控制语句中无任何语句。

- break 语句

break 语句可以终止最邻近的 switch、while 或 for 语句的执行。Break 语句由最近的终止条件语句控制。本语句目的之一就是当满足一定条件时完成循环的执行。

示例:

```
// 搜索数组中第一个值为零的元素
for(i=0; i<array_size; i++)
  if((array[i]==0)
    break;
```

- continue 语句

continue 语句控制最邻近的循环语句 while 或 for 返回循环开始,接着执行下一次循环。continue 语句与 break 语句作用相反。

示例:

```
//  统计数组中非零元素个数
int func(int array[])
  {
   int array_size=ArraySize(array);
   int sum=0;
   for(int i=0; i<array_size;  i++)
     {
      if(a[i]==0) continue;
      sum+=a[i];
     }
   return(sum);
  }
```

- return 语句

Return 语句结束当前函数的执行,返回到调用程序。语句 return(expression); 结束当前函数的执行并带回函数的结果，其中的表达式必须放在圆括号内，不能使用赋值号。

示例:

```
int CalcSum(int x, int y)
  {
     return(x+y);
  }
```

函数值的类型定义为 void 型时，return 语句后面不能使用表达式:

```
void SomeFunction()
  {
   Print("Hello!");
   return;     // 这个语句能删除
  }
```

本函数右大括弧}表明这个不带表达式的 return 语句将明确执行。

- if-else 语句

如果条件表达式为 true，语句序列 operator 1 执行, 语句序列 operator 2 将不执行。如果表达式为 false，语句序列 operator 2 执行，而语句序列 operator 1 不执行。

```
if (expression)
     operator1
else
     operator2
```

if 语句的 else 部分可能被省略，因此，如果本语句省略了 else 部分，if 语句在嵌套时可能会出现歧义。在这种情况下， else 会匹配前面最近的不带 else 部分的 if 语句。

示例:

```
//   else 部分匹配第二个 if 语句:
  if(x>1)
  if(y==2) z=5;
  else     z=6;

//   else 部分匹配第一个 if 语句:
if(x>1)
  {
   if(y==2) z=5;
  }
else        z=6;
```

```
//  if 语句嵌套
if (x=='a')
  {
   y=1;
  }
else if(x=='b')
  {
   y=2;
   z=3;
  }
else if(x=='c')
  {
   y = 4;
  }
else Print("ERROR");
```

- switch 多分支语句

switch 语句比较 expression 表达式值和 case 中 constant 常量值，如果相等，就执行相应的语句块。每个 case 中 constant 值必须是一个整数、字符串常量或常量表达式。常量表达式中不能包含变量和函数调用。Switch 后面的表达式(expression)必须是整型。

```
switch(expression)
  {
   case constant: operators
   case constant: operators
     ...
   default: operators
  }
```

如果没有一个 case 表达式值等于 expression 表达式值，将执行 default 后面的语句块。此处 default 语句并不是必需的。如果没有一个 case 常量和 expression 值一致，而 default 语句也不可用，那么不会执行任何动作。关键字 case 及其常量就像标签，即使 switch 语句带有数个 case 语句，程序都会按序执行后面所有的语句块，直至遇到 break 语句为止。

在编译期间，编译程序将计算常量表达式的值，不允许在一个 switch 语句内同时存在两个相同的常量值。

示例:

```
switch(x)
```

```
  {
   case 'A':
      Print("CASE A");
      break;
   case 'B':
   case 'C':
      Print("CASE B or C");
      break;
   default:
      Print("NOT A, B or C");
      break;
  }
```

- while 循环语句

如果条件表达式 expression 值为 true， while 语句一直执行直至条件表达式变成 false。如果条件表达式为 false，将跳到下一条语句。

```
while(expression)
  operator;
```

本语句执行前，表达式 expression 必须事先定义过。因此，如果表达式值一开始就为 false，while 语句根本不会执行。

示例:

```
while(k<n)
  {
   y=y*x;
   k++;
  }
```

- for 循环语句

表达式 1（Expression1）定义循环的初始变量，表达式 2(Expression2)是循环终止条件。当表达式 2(Expression2) 为真(true)的时候，循环体重复执行直至 Expression2 变为假(false)。如果 Expression2 变为 false，循环将会被中断，并继续执行循环语句的下一条语句。每次循环结束后都会计算表达式 3 (Expression3)，用于改变循环条件。

```
for (Expression1;  Expression2;  Expression3)
   operator;
```

for 语句与下列 while 语句等价：

```
Expression1;
while(Expression2)
  {
   operator;
   Expression3;
  };
```

For 语句中我们可以省略三个表达式中任一部分或全部，但是，其中分号；却不能省略。如果省略了表达式 2(expression2)，则被视作真值。for(;;)语句是一个死循环，它相当于 while(1) 语句。表达式 1 和表达式 3 都可以内嵌多个逗号(,)分隔的表达式。

示例:

```
for(x=1; x<=7; x++) Print(MathPower(x,2));

for(; ; )
  {
   Print(MathPower(x,2));
   x++;
   if(x>10) break;
  }

for(i=0,j=n-1; i<n; i++,j--) a[i]=a[j];
```

函数

函数是一段已命名的程序，它可以从程序任一部分多次调用。它是由函数返回值的类型说明、函数名称、形式参数和语句块组成。传递参数的个数被限定在 64 个之内。

示例:

```
double                          // 返回值的类型
linfunc (double x, double a, double b) // 函数名称和参数列表
{
                                // 语句块
   return (a + b);               // 返回表达式值
 }
```

Return 语句可以返回函数内表达式的值。如有必要，表达式值的类型可以转换为函数结果类型。没有返回值的函数必须定义成“void”型。

示例:

```
void errmesg(string s)
  {
   Print("错误: "+s);
  }
```

函数的参数可能存在默认值，这些默认值是用相应类型常量定义的。

示例:

```
int somefunc(double a, double d=0.0001, int n=5, bool b=true, string
s="passed string")
  {
   Print("要求输入参数 a=",a);
   Print("下列参数被传递: d=",d," n=",n," b=",b," s=",s);
   return (0);
  }
```

如果为某个参数指定了默认值，那么所有后续的参数也必须指定默认值。

错误范例:

```
int somefunc(double a, double d=0.0001, int n, bool b, string s="passed
string")
  {
  }
```

- 函数调用

如果在表达式前出现一个曾未用过的名字，又后跟一个左括号，在上下文环境中它将被视作函数的名字。

```
函数名称 (x1, x2,..., xn)
```

函数自变量(形式参数) 按值的方式传递，也就是说，先计算每一个表达式 x1, . . . , xn 的值，再将其值传递给函数，表达式计算顺序及其值的传递要确保无误。在执行期间，系统将检查那些提供给函数的值和参数的类型。这种形式的函数调用被称为“值传递”。调用函数是为了获得函数返回的表达式值。函数的定义类型必须符合函数返回值的类型。在全局层次，我们可以在程序的任何位置定义和说明函数，即要在其他函数之外定义，在函数之内，不能定义或说明另一个函数。

例如:

```
int start()
  {
```

```
   double some_array[4]={0.3, 1.4, 2.5, 3.6};
   double a=linfunc(some_array, 10.5, 8);
   //...
  }
double linfunc(double x[], double a, double b)
  {
   return (a*x[0] + b);
  }
```

当我们使用默认参数调用函数时，参数列表使用规则是有要求的。参数列表中无默认值的参数必须指定值，且不可以省略。有默认值的参数要么全省略掉，要么指定值，不可只省略其中部分参数。

例如:

```
void somefunc(double init,double sec=0.0001,int level=10);   // 函数原型

somefunc();                     // 错误调用, 第一个参数必须赋值。
somefunc(3.14);                 // 正确调用
somefunc(3.14, 0.0002);         // 正确调用
somefunc(3.14, 0.0002, 10);     // 正确调用
```

当我们调用一个函数时，不可以省略参数，即使那些存在默认值的参数也不行。

```
somefunc(3.14, , 10);           // 错误调用。第二个参数被省略。
```

- 特殊函数

在 MQL4 中存在三种预定义名称的函数:

init() 是一个在模块初始化时调用的函数，可以用此函数在开始自定义指标或者自动交易之前做初始化操作。如果这个函数不可用，初始化时就不会调用任何函数。

start() 是主函数。对于智能交易，它在收到下一跳数据后调用该函数。对于自定义指标，在指标添加到图表之后，或在客户端开始运行之时，也可在收到下一跳数据之后，该函数被调用。对于脚本，在脚本被添加到图表之后立即执行并初始化。如果模块中根本不存在 start()函数，模块（智能交易、脚本或自定义指标）就不能执行。

deinit() 是一个模块的析构函数（注：借用 C++概念），执行与 init() 函数相反的操作。Deinit()函数往往用来做“清理善后” 的工作，例如，创建

对象时开辟了一片内存空间，退出模块前需要释放。如果用户没有编写该函数，它也不会执行任何操作。

预定义函数可以带参数。不过，当客户端调用这些函数时，并不能从外部向它传递参数，只能采用默认值。 start(), init()和 deinit()函数可从模块任何一点按照常规函数规则调用，等同于其他函数。

不推荐从 init()函数调用 start()函数或是执行交易操作。对于图表数据、市价等，在模块初始化期间，数据可能残缺不全，这时，init() 和 deinit() 函数必须尽快地完成任务。在调用 start()函数之前，千万不要开始尝试交易操作。

变量

变量必须在使用之前定义。变量必须拥有唯一的标识名。变量的说明包括要定义的变量名及其类型。变量的说明不是语句。

基本类型如下：

- bool - 布尔值 true 和 false;
- string - 字符串;
- double - 带浮点的双精度数。

示例:

```
string MessageBox;
int    Orders;
double SymbolPrice;
bool   bLog;
```

其它类型:

- color - 代表 RGB 颜色的整数;
- datetime - 日期和时间，无符号整数，代表从 1970 年 1 月 1 日 上午 00:00 开始以来总秒数。

其它的数据类型仅仅在输入参数说明时才有意义，可以更方便地在属性窗口查看。

示例:

```
datetime tBegin_Data   = D'2004.01.01 00:00';
color    cModify_Color = C'0x44,0xB9,0xE6';
```

数组

数组是有下标的同类型的一组数据。

```
int    a[50];          // 50 个元素的一维整数数组
double m[7][50];       // 7 个一维数组组成二维数组
                       //每一维数组由 50 个整数组成。
```

只有整数才能作为数组的下标。不允许使用四维以上的数组。数组元素的下标从 0 开始。一维数组的最后一个元素下标是数组大小-1。这就意味着，在由 50 个元素组成的数组中，引用数组的最后一个元素是 a[49]。同样的规则也适用于多维数组：任一维数组下标都是从 0 开始，到数组大小-1 结束。从上例可以看出，这个二维数组的最后一个元素将会出现像 m[6][49]这样的下标。

如果引用超出数组下标范围，系统将产生数组下标越界错误 ERR_ARRAY_INDEX_OUT_OF_RANGE，这些信息可通过 GetLastError()函数得到。

- 局部变量

在一个函数内部定义的变量是局部变量。局部变量的作用范围被限定在所定义的函数内。局部变量可以用任意一个表达式值进行初始化。每次调用函数都会初始化一次局部变量。局部变量存储在函数申请的临时空间中。

示例:

```
int somefunc()
  {
   int ret_code=0;
   ....
   return(ret_code);
  }
```

- 形式参数

传递给函数的参数都是局部变量。作用范围限制在函数内。形式参数名称应与外部定义的变量名和函数内部定义的局部变量名不同。调用函数时形参变量必须赋值，当然，在函数内，这些形参变量也可以被赋值。

示例:

```
void func(int x[], double y, bool z)
  {
   if(y>0.0 && !z)
      Print(x[0]);
   ...
  }
```

形参变量可用常数进行初始化。在这种情况下，初始化值被当作默认值。

另外，紧随其后的形参变量也必须初始化。

示例:

```
void func(int x, double y = 0.0, bool z = true)
  {
   ...
  }
```

当调用函数时，初始化过的参数可能被省略不写，默认值会代替它们。

示例:

```
func(123, 0.5);     //参数 z 被省略,函数内使用 z 的默认值
```

从外部模块导入的 MQL4 库函数无法初始化参数的默认值。

参数采用值传递方式。也就是说，在任何情况下，在被调用函数内部，如果修改了参数值，这种修改结果将不会带回主调用函数内。数组可以作为函数的参数，但是，如果以数组作为参数，修改数组元素值是不允许的。

它还可能通过引用进行参数传递(称之为引用传递或地址传递)。在这种情况下，通过引用传递，这些参数的修改将被传递给调用函数中对应的变量。数组元素无法通过引用传递给参数。通过引用传递参数只能在一个模块内进行，库并不提供引用传递。为了表明参数是通过引用传递，在定义时必须在数据类型后放置&符号。

示例:

```
void func(int& x, double& y, double& z[])
  {
   double calculated_tp;
   ...
   for(int i=0;  i<OrdersTotal();  i++)
     {
      if(i==ArraySize(z))       break;
      if(OrderSelect(i)==false) break;
      z[i]=OrderOpenPrice();
     }
   x=i;
   y=calculated_tp;
  }
```

数组也可以使用引用传递，同理，所有修改将反映在源数组中。不同于简单的参数，数组也可以通过引用传递给库函数。

采用引用传递方式的参数无法初始化默认值。

传递给函数的最大参数个数不得超过 64 个。

- 静态变量

"static(静态)"存储类型用于定义一个静态变量。在数据类型前指定"static"说明符说明定义的是一个静态变量。

示例:

```
int somefunc()
  {
   static int flag=10;
   ....
   return(flag);
  }
```

静态变量被存放在内存静态存储区里，在函数运行结束后静态变量的值不会丢失。同一模块内所有变量，除函数的形参变量外，都能定义成静态变量。静态变量只能由相应类型的常量初始化，这点与一般的局部变量有所不同,局部变量可由任意类型的表达式进行初始化。如果静态变量没有明确地初始化,它将被初始化为零。静态变量只可在"init()" 函数之前初始化一次。当从定义了静态变量的函数内部退出时，静态变量值不会丢失。

- 全局变量

全局变量只能定义在函数之外，与函数同级，也就是说，在任何模块中全局变量都不是局部的。

示例:

```
int GlobalFlag=10;    // 全局变量
int start()
  {
   ...
  }
```

全局变量的作用域是整个程序。全局变量可从任意函数内访问。 如果它的值没有被明确定义，初始化值就为零。一个全局变量只能由相应类型的常量进行初始化。全局变量只可以在程序装入到客户机内存时初始化一次。

注: 全局定义的变量不能与那些能被 GlobalVariable…()函数访问的客户端全局变量混淆。

- 定义外部变量

外部存储类型 extern 可以定义一个外部变量。 在数据类型之前冠以

“extern”说明符指明定义外部变量。

示例:

```
extern double InputParameter1 = 1.0;
extern color  InputParameter2 = red;
int init()
  {
   ...
  }
```

外部变量决定程序的数据输入，他们会直接显示在程序属性窗口。数组本身不能作为外部变量。

- 变量初始化

定义变量时可以初始化值。如果变量的初始值未被明确指定，它就被初始化为零(0)。 全局变量和静态变量仅能被相应类型的常量初始化，而局部变量可以被任意类型的表达式初始化，并不局限于常量。

全局变量和静态变量只能初始化一次。局部变量在被相应的函数调用时每次都会初始化。

示例:

```
int    n       = 1;
double p       = MarketInfo(Symbol(),MODE_POINT);
string s       = "hello";
double f[]     = { 0.0, 0.236, 0.382, 0.5, 0.618, 1.0 };
int    a[4][4] = { 1, 1, 1, 1,  2, 2, 2, 2,  3, 3, 3, 3,  4, 4, 4, 4 };
```

数组元素值列表必须被包含在大括号{}内，省略初始化的值被认为零。如果初始化定义时没有指定数组大小,编译器将根据初始化值列表的大小来定义。多维数组按照一维数组的顺序进行初始化, 即初始化序列中不需要另加大括号。所有数组, 包括那些在局部范围内定义的数组，只能用常数进行初始化。

- 外部函数定义

定义在程序其它部分内的外部函数类型必须明确地说明。缺乏这样定义也许导致在程序的编译、连接、运行时出错。说明一个外部对象时,必须使用关键字#import。

示例:

```
#import "user32.dll"
```

```
  int     MessageBoxA(int hWnd ,string szText,string szCaption,int
nType);
  int     SendMessageA(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex4"
  double  round(double value);
#import
```

Import 语句清楚地说明这些函数是从外部 DLL 库或编译过的 EX4 库中引用的。

变量的指针能传给导入的 DLL 函数。字符串型数据能作为指针传给相应的内存块(我们应该记住，字符串数据的内部表示分为两个部分：内存块长度和内存块的指针)。如果有必要传递整型或双精度型数据,那么这些类型的一维数组也能作为参数引用传递。

示例:

```
#import "some_lib.dll"
  void    PassIntegerByref(int& OneInt[]);
#import
int start()
  {
   int array[1];
//...
   PassIntegerByref(array);
   Print(array[0]);
//...
  }
```

编译预处理

预处理程序是 MQL4 编译程序的一个特殊的组成部分，用于在程序被编译之前预先准备好程序源码。

预处理程序能增强程序源码的可读性。在 MQL4 程序中，通过包含指定的文件可以获得结构化源码，给常量取一个易于记忆的名字有助于增强源码的可读性。

预处理程序也允许 MQL4 程序定义特定的参数。

如果 # 号置于程序的第一行，那么该行就是预处理程序的控制指令。预处理程序指令以回车换行结束。

- 预定义常量

使用 #define 命令，我们可以在程序中定义符号名或符号常量代表特定的字符串。 随后，编译程序会把所有符号名替换成相应的字符串。事实上，这些名称可以被任意的文本替换，并不局限于数字:

```
#define identifier value
```

常量符号名同样遵守变量名的命名规则,值可以是以下任意类型:

```
#define ABC          100
#define PI           0.314
#define COMPANY_NAME "MetaQuotes Software Corp."

...

void ShowCopyright()
  {
   Print("版权所有 © 2001-2007, ",COMPANY_NAME);
   Print("http://www.metaquotes.net");
```

- 编译控制

每个 MQL4 程序允许指定额外的特定参数，这些参数用 #property 命名，它不需要用户明确地启动程序，就可帮助客户端提供合适的服务。要注意的是，这个功能还与技术指标的外部设定有关。

```
#property identifier value
```

常数	类型	描述
Link	string	连接公司网站网址
copyright	string	公司或厂商名称
stacksize	int	堆栈大小
library		库;
indicator_chart_window	void	在图表窗口显示指标
indicator_separate_window	void	在独立显示窗口中显示指标
indicator_buffers	int	计算指标的缓冲区个数，最大为 8
indicator_minimum	double	独立显示的指标窗口下端缩放比例
indicator_maximum	double	独立显示的指标窗口上端缩放比例
indicator_colorN	color	线 1 到线 8 的显示颜色

indicator_widthN	int	线 1 到线 8 的显示宽度
indicator_styleN	int	线 1 到线 8 的显示样式
indicator_levelN	double	自定义指标的第 n 条水平线(n 为 1-8)
indicator_levelcolor	color	自定义指标的第 n 条水平线颜色(n 为 1-8)
indicator_levelwidth	int	自定义指标的第 n 条水平线宽度(n 为 1-8)
indicator_levelstyle	int	自定义指标的第 n 条水平线样式(n 为 1-8)
show_confirm	void	在脚本运行之前显示确认框
show_inputs	void	在脚本运行之前显示它的属性表；禁用 show_confirm 属性

示例:

```
#property link          "http://www.metaquotes.net"
#property copyright     "MetaQuotes Software Corp."
#property library
#property stacksize     1024
```

在所执行模块的设置中，编译程序将会保存这些已说明的值。

- 文件包含

#include 命令行可以放置在程序的任意部分，但是所有的“文件包含”通常都被统一放置在源代码的开头。调用格式：

```
#include <file_name>
#include "file_name";
```

示例:

```
#include <WinUser32.mqh>
#include "mylib.mqh"
```

预处理程序将用 WinUser32.mqh 文件内容替换这一行。尖括号表示 WinUser32.mqh 文件将会从默认目录调用(通常默认目录为 terminal_directory\experts\include)。不会搜索当前目录。

如果文件名用引号括起来，将在当前目录中搜索该文件(源码主文件所在位置)。不会搜索标准目录。

- 函数导入

函数可从 MQL4 编译过的模块 (*.EX4 文件) 和操作系统文件模块(*.DLL 文件)导入过来。模块名需要在 #import 指令中指定。因为编译程序能够以适当的方式生成被导入的函数调用和传递参数, 函数的完整说明是必需的。函数说明要紧跟在 #import "module name"命令后面,以新的 #import 命令(不带参数)结束导入函数说明块。

```
#import "file_name"
    func1 define;
    func2 define;
    ...
    funcN define;
#import
```

导入函数必须有唯一的名称。相同名称的函数无法从不同的模块同时导入。导入的函数名称不能与那些内置函数冲突。

由于导入函数是在模块之外编译的, 编译程序无法检查参数传递的正确性。这就是为什么, 为了避免运行时错误,有必要精确地说明参数类型定义和参数顺序的原因。传递到导入函数(从 EX4 和从 DLL 模块) 的这些参数不能通过默认值获得值。

示例:

```
#import "user32.dll"
   int    MessageBoxA(int hWnd, string lpText, string lpCaption, int
uType);
#import "stdlib.ex4"
   string ErrorDescription(int error_code);
   int    RGB(int red_value, int green_value, int blue_value);
   bool   CompareDoubles(double number1, double number2);
   string DoubleToStrMorePrecision(double number, int precision);
   string IntegerToHexString(int integer_number);
#import "Expert 示例.dll"
   int    GetIntValue(int);
   double GetDoubleValue(double);
   string GetStringValue(string);
   double GetArrayItemValue(double arr[], int, int);
   bool   SetArrayItemValue(double& arr[], int,int, double);
   double GetRatesItemValue(double rates[][6], int, int, int);
   int    SortStringArray(string& arr[], int);
   int    ProcessStringArray(string& arr[], int);
#import
```

对于在 MQL4 程序执行期间导入的函数，采用了所谓的“后期联编”。这就意味着只要导入的函数未被调用，相应的模块(EX4 或 DLL)就不会被加载。

不推荐使用全路径文件名 Drive:\Directory\FileName.Ext 加载模块。MQL4 库会从 terminal_dir\experts\libraries 文件夹中载入进来。如果没有找到库，就会尝试从 terminal_dir\experts 文件夹中加载。

预定义标准常量

为了简化程序编写,使程序文本编辑感觉更加简便,在 MQL4 中预定义了标准常量。

标准变量类似于宏代换，而且有 int 型。

这些常量按用途进行分组。

- 序列化数组

序列化数组标识符用在 ArrayCopySeries(), iHighest() 和 iLowest()函数。

可以是以下任意值:

常数	值	描述
MODE_OPEN	0	开盘价
MODE_LOW	1	最低价
MODE_HIGH	2	最高价
MODE_CLOSE	3	收盘价
MODE_VOLUME	4	成交量，用在 iLowest()和 iHighest()函数中。
MODE_TIME	5	开盘时间，用在 ArrayCopySeries()函数中。

- 图表时段

图表时段(图表的时间周期)。可以是以下任意值:

常数	值	描述
PERIOD_M1	1	1 分钟
PERIOD_M5	5	5 分钟
PERIOD_M15	15	15 分钟
PERIOD_M30	30	30 分钟

PERIOD_H1	60	1 小时
PERIOD_H4	240	4 小时
PERIOD_D1	1440	日
PERIOD_W1	10080	周
PERIOD_MN1	43200	月
0 (zero)	0	采用图表中当前时段。

- 交易类型

用于 OrderSend()函数的交易类型。可以是以下任意值:

常数	值	描述
OP_BUY	0	即时成交买单
OP_SELL	1	即时成交卖单
OP_BUYLIMIT	2	限价挂单买单
OP_SELLLIMIT	3	限价挂单卖单
OP_BUYSTOP	4	止损挂单买单
OP_SELLSTOP	5	止损挂单卖单

- 价格常量

价格常量，它可以是以下的任意值：

常数	值	描述
PRICE_CLOSE	0	平仓价
PRICE_OPEN	1	开仓价
PRICE_HIGH	2	最高价
PRICE_LOW	3	最低价
PRICE_MEDIAN	4	均价(high+low)/2
PRICE_TYPICAL	5	标准价 (high+low+close)/3
PRICE_WEIGHTED	6	加权收盘价 (high+low+close+close)/4

- 市场信息

市场信息常量，用在 MarketInfo()函数。可以是以下任意值:

常数	值	描述
MODE_LOW	1	今日最低价。
MODE_HIGH	2	今日最高价。
MODE_TIME	5	最新收到的一跳时间（最新知道的服务器时间）。
MODE_BID	9	最新收到的买价。对于当前货币对，它被保存在预定义变量 bid 中。
MODE_ASK	10	最新收到的卖价。对于当前货币对，它被保存在预定义变量 ask 中。
MODE_POINT	11	当前报价的点值。对于当前货币对，它被保存在预定义变量 point 中。
MODE_DIGITS	12	在货币对价格中小数点后小数位数。对于当前货币对，它被保存在预定义变量 digits 中。
MODE_SPREAD	13	变动点数(点差)。
MODE_STOPLEVEL	14	止损位点数。
MODE_LOTSIZE	15	标准手大小。
MODE_TICK 值	16	当前报价一跳值。
MODE_TICKSIZE	17	当前报价一跳大小。
MODE_SWAPLONG	18	掉期多头头寸。
MODE_SWAPSHORT	19	掉期空头头寸。
MODE_STARTING	20	交易开始日期（通常用作期货）。
MODE_EXPIRATION	21	交易到期日（通常用作期货）。
MODE_TRADEALLOWED	22	货币对是否允许交易。
MODE_MINLOT	23	允许最小的一手数量。
MODE_LOTSTEP	24	改变标准手的增幅。
MODE_MAXLOT	25	允许最大的一手数量。
MODE_SWAPTYPE	26	掉期计算方法。0 - 按点； 1 - 按基准货币对； 2 - 按利率； 3 - 按保证金货币。
MODE_PROFITCALCMODE	27	赢利计算模式 0 - 外汇； 1 - CFD(差价合约)； 2 - 期货。
MODE_MARGINCALCMODE	28	保证金计算模式 0 - 外汇； 1 - CFD； 2 - 期货； 3 - CFD 指数。
MODE_MARGININIT	29	1 手的初始保证金要求。

MODE_MARGINMAINTENANCE	30	维持开仓 1 手的保证金。
MODE_MARGINHEDGED	31	1 手的对冲保证金。
MODE_MARGINREQUIRED	32	要求买 1 手保证金的余额。
MODE_FREEZELEVEL	33	冻结定单的点位。如果执行价出现在冻结定单的点位范围内，定单将不能被修改、取消或关闭。

- 画线样式

SetIndexStyle()函数中画线样式的列表。可以是以下任意值：

常数	值	描述
DRAW_LINE	0	画线
DRAW_SECTION	1	画线条
DRAW_HISTOGRAM	2	画柱状图
DRAW_ARROW	3	画箭头(货币对)
DRAW_ZIGZAG	4	画折线
DRAW_NONE	12	不画线

画线样式，宽度为 1 时有效。可以是以下任意值：

常数	值	描述
STYLE_SOLID	0	实线
STYLE_DASH	1	虚线
STYLE_DOT	2	点线
STYLE_DASHDOT	3	点划线
STYLE_DASHDOTDOT	4	双点划线

- 箭头代码

箭头代码常量，预定义箭头代码列表。可以是以下的任意值：

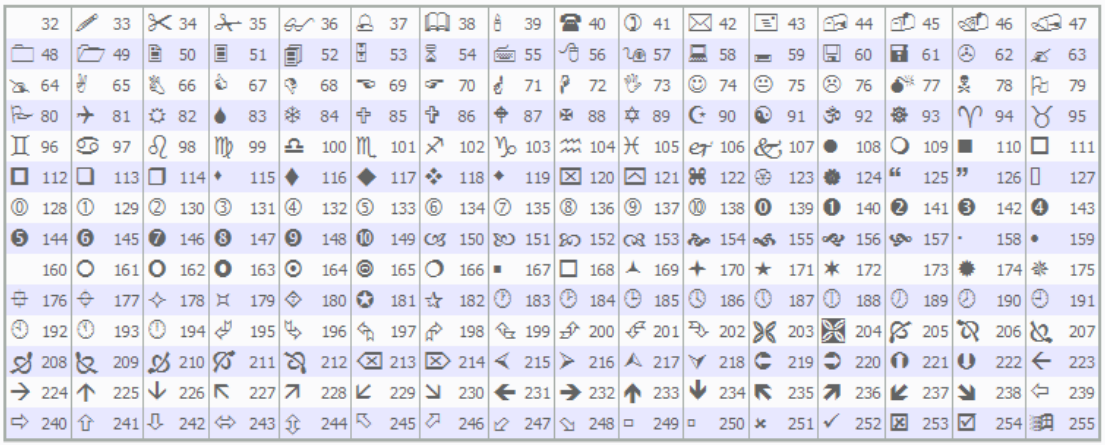
常数	值	描述
SYMBOL_THUMBSUP	67	大拇指向上(👍)
SYMBOL_THUMBSDOWN	68	大拇指向下符号(👎)
SYMBOL_ARROWUP	241	向上箭头符号(⇧)

SYMBOL_ARROWDOWN	242	向下箭头符号(⇩)
SYMBOL_STOPSIGN	251	打叉符号(×)
SYMBOL_CHECKSIGN	252	打勾符号(✓)

对于价格和时间的特殊箭头代码。可以是以下任意值:

常数	值	描述
	1	右上转箭头 (↱)
	2	右下转箭头(↳)
	3	左指向三角 (◄)
	4	破折号(-)
SYMBOL_LEFTPRICE	5	左侧价格标签
SYMBOL_RIGHTPRICE	6	右侧价格标签

- Wingdings 符号



- Web 颜色名称

Black	DarkGreen	DarkSlateGray	Olive	Green	Teal	Navy	Purple
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOliveGreen	SaddleBrown	ForestGreen	OliveDrab
SeaGreen	DarkGoldenrod	DarkSlateBlue	Sienna	MediumBlue	Brown	DarkTurquoise	DimGray
LightSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	Crimson	SteelBlue
Goldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	LimeGreen	OrangeRed
DarkOrange	Orange	Gold	Yellow	Chartreuse	Lime	SpringGreen	Aqua
DeepSkyBlue	Blue	Magenta	Red	Gray	SlateGray	Peru	BlueViolet
LightSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	SlateBlue	DarkKhaki
IndianRed	MediumOrchid	GreenYellow	MediumAquamarine	DarkSeaGreen	Tomato	RosyBrown	Orchid
MediumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	MediumSlateBlue	Tan
DarkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	SkyBlue	LightSalmon
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSkyBlue	LightSteelBlue	LightBlue
PaleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGray	Wheat	NavajoWhite
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	LightGoldenrod	BlanchedAlmond
LemonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	LightCyan	Linen
Lavender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	Honeydew	AliceBlue
LavenderBlush	MintCream	Snow	White				

- 指标线

指标线常量用在 iMACD(), iRVI() 和 iStochastic() 函数指标中。

可以是以下的任意值:

常数	值	描述
MODE_MAIN	0	基本指标线。
MODE_SIGNAL	1	信号线。

指标线常量用在 iADX()函数指标中。

常数	值	描述
MODE_MAIN	0	基本指标线。
MODE_PLUSDI	1	+DI 指标线。
MODE_MINUSDI	2	-DI 指标线。

指标线常量用在 iBands(), iEnvelopes(), iEnvelopesOnArray(), iFractals() 和 iGator() 函数指标中。

常数	值	描述
MODE_UPPER	1	上指标线。
MODE_LOWER	2	下指标线。

- 一目均衡表代码

Ichimoku Kinko Hyo(一目均衡表)常量作为请求数据源使用在 iIchimoku()指标调用中。

可以为以下任意值:

常数	值	描述
MODE_TENKANSEN	1	Tenkan-sen －转换线(红线)
MODE_KIJUNSEN	2	Kijun-sen － 基准线(蓝线)
MODE_SENKOUSPANA	3	Senkou Span A － 前移指标 A
MODE_SENKOUSPANB	4	Senkou Span B － 前移指标 B
MODE_CHINKOUSPAN	5	Chinkou Span － 云带(抵抗带)(绿线)

- 移动平均线计算方法

移动平均线计算方法用在 iAlligator(), iEnvelopes(), iEnvelopesOnArray, iForce(), iGator(), iMA(), iMAOnArray(), iStdDev(), iStdDevOnArray(), iStochastic()指标中。

可以使以下任意值:

常数	值	描述
MODE_SMA	0	简单移动平均
MODE_EMA	1	指数移动平均
MODE_SMMA	2	平滑移动平均
MODE_LWMA	3	线性加权移动平均

- 信息框

信息框函数返回代码。

如果一个信息框中有“Cancel（取消）”按钮，当你选择 Cancel 按钮或是按了 ESC 键时，函数将返回 IDCANCEL 值。如果信息框中不存在 Cancel 按钮，按 ESC 键无效。

注：信息框返回码定义在 WinUser32.mqh 文件中。

常数	值	描述
IDOK	1	选择 OK(确定) 按钮
IDCANCEL	2	选择 Cancel(取消) 按钮
IDABORT	3	选择 Abort(中止) 按钮
IDRETRY	4	选择 Retry(重试) 按钮
IDIGNORE	5	选择 Ignore(忽略) 按钮
IDYES	6	选择 Yes(是) 按钮

IDNO	7	选择 No(否) 按钮
IDTRYAGAIN	10	选择 Try Again(再试一次) 按钮
IDCONTINUE	11	选择 Continue(继续) 按钮

MessageBox 函数标志描述了一个对话框的内容和行为。这个值还可以和下列值组合应用。

为了在信息框中显示相应按钮，可以指定下列值之一。

常数	值	描述
MB_OK	0x00000000	消息框包含一个按钮: OK，这是默认值
MB_OKCANCEL	0x00000001	消息框包含两个按钮: OK 和 Cancel
MB_ABORTRETRYIGNORE	0x00000002	消息框包含三个按钮: Abort, Retry 和 Ignore
MB_YESNOCANCEL	0x00000003	消息框包含三个按钮: Yes, No 和 Cancel
MB_YESNO	0x00000004	消息框包含两个按钮: Yes 和 No
MB_RETRYCANCEL	0x00000005	消息框包含两个按钮: Retry 和 Cancel
MB_CANCELTRYCONTINUE	0x00000006	Windows 2000: 消息框包含三个按钮: Cancel, Try Again, Continue。 用这个消息框类型代替 MB_ABORTRETRYIGNORE。

为了在信息框中显示图标，可以指定下列值之一。

常数	值	描述
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	显示禁止标志图标
MB_ICONQUESTION	0x00000020	显示问号图标
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	显示感叹号图标
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	显示小写字母 i 组成的图标

为了在信息框内显示默认的图标，可以指定下列值之一。

常数	值	描述

MB_DEFBUTTON1	0x00000000	默认第一个按钮。 除非指定 MB_DEFBUTTON2, MB_DEFBUTTON3, MB_DEFBUTTON4，否则，MB_DEFBUTTON1 就是默认。
MB_DEFBUTTON2	0x00000100	第二个按钮设为默认。
MB_DEFBUTTON3	0x00000200	第三个按钮设为默认。
MB_DEFBUTTON4	0x00000300	第四个按钮设为默认。

MessageBox() 函数行为标志定义在 WinUser32.mqh 文件内， 这就是为什么这个头文件必须通过#include <WinUser32.mqh >包含进来的原因。此处，没有列出所有的标志，详见 Win32 API 说明。

- 对象类型

对象类型常量用在 ObjectCreate(), ObjectsDeleteAll() 和 ObjectType() 函数。可以使用以下任意值:

对象可能有 1-3 个相关类型的坐标。

常数	值	描述
OBJ_VLINE	0	垂直线。使用第一个坐标的时间部分。
OBJ_HLINE	1	水平线。使用第一个坐标的价格部分。
OBJ_TREND	2	趋势线。使用 2 个坐标。
OBJ_TRENDBYANGLE	3	趋势角度。使用 1 个坐标。用 ObjectSet() 函数设置线的角度。
OBJ_REGRESSION	4	回归。使用头两个坐标的时间部分。
OBJ_CHANNEL	5	通道。使用 3 个坐标。
OBJ_STDDEVCHANNEL	6	标准差通道。使用头两个坐标的时间部分。
OBJ_GANNLINE	7	甘氏线。使用 2 个坐标，但忽略第二个坐标的价格部分。
OBJ_GANNFAN	8	甘氏扇形线。使用 2 个坐标，但忽略第二个坐标的价格部分。
OBJ_GANNGRID	9	甘氏网格线。使用 2 个坐标，但忽略第二个坐标的价格部分
OBJ_FIBO	10	斐波纳契回撤。使用 2 个坐标。
OBJ_FIBOTIMES	11	斐波纳契时间周期线。应用 2 个坐标。
OBJ_FIBOFAN	12	斐波纳契扇形线。使用 2 个坐标。
OBJ_FIBOARC	13	斐波纳契弧线。使用 2 个坐标。

OBJ_EXPANSION	14 斐波纳契扩展。使用 3 个坐标。
OBJ_FIBOCHANNEL	15 斐波纳契通道。使用 3 个坐标。
OBJ_RECTANGLE	16 矩形。使用 2 个坐标。
OBJ_TRIANGLE	17 三角形。使用 3 个坐标。
OBJ_ELLIPSE	18 椭圆形。使用 2 个坐标。
OBJ_PITCHFORK	19 安德鲁分叉线。使用 3 个坐标。
OBJ_CYCLES	20 周期线。使用 2 个坐标。
OBJ_TEXT	21 文本。使用 1 坐标。
OBJ_ARROW	22 箭头。使用 1 个坐标。
OBJ_LABEL	23 文本标签。按像素使用 1 个坐标。

- 对象属性

对象值索引用在 ObjectGet() 和 ObjectSet() 函数。它可能是以下的任意值:

常数	值	类型	描述
OBJPROP_TIME1	0	datetime	设置/获取第一个坐标时间部分的日期时间值。
OBJPROP_PRICE1	1	double	设置/获取第一个坐标价格部分的双精度值。
OBJPROP_TIME2	2	datetime	设置/获取第二个坐标时间部分的日期时间值。
OBJPROP_PRICE2	3	double	设置/获取第二个坐标价格部分的双精度值。
OBJPROP_TIME3	4	datetime	设置/获取第三个坐标时间部分的日期时间值。
OBJPROP_PRICE3	5	double	设置/获取第三个坐标价格部分的双精度值。
OBJPROP_COLOR	6	color	设置/获取对象颜色的颜色值。
OBJPROP_STYLE	7	int	设置/获取对象的线型样式。值为下列 STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT 常量之一。
OBJPROP_WIDTH	8	int	设置/获取对象线宽的整数值。可以从 1 到 5。

OBJPROP_BACK	9	bool	设置/获取对象背景线标志的布尔值。
OBJPROP_RAY	10	bool	设置/获取对象射线标志的布尔值。
OBJPROP_ELLIPSE	11	bool	设置/获取斐波纳契弧线椭圆标志的布尔值。
OBJPROP_SCALE	12	double	设置/获取缩放对象属性的双精度值。
OBJPROP_ANGLE	13	double	设置/获取角对象属性的双精度值（度）。
OBJPROP_ARROWCODE	14	int	设置/获取箭头代码对象属性的整数值。
OBJPROP_TIMEFRAMES	15	int	设置/获取图表时段对象属性，其值为一个或多个时段对象显示常数的组合值。
OBJPROP_DEVIATION	16	double	设置/获取标准差对象离差属性的双精度值。
OBJPROP_FONTSIZE	100	int	设置/获取文本对象字体大小的整数值。
OBJPROP_CORNER	101	int	设置/获取标签对象固定角的整数值。必须是从 0-3。
OBJPROP_XDISTANCE	102	int	设置/获取固定 X 间隔对象属性的整数值。
OBJPROP_YDISTANCE	103	int	设置/获取固定 Y 间隔对象属性的整数值。
OBJPROP_FIBOLEVELS	200	int	设置/获取斐波纳契对象水平线个数的整数值。可以从 0 到 32 。
OBJPROP_LEVELCOLOR	201	color	设置/获取对象水平线颜色的颜色值。
OBJPROP_LEVELSTYLE	202	int	设置/获取对象水平线样式的值。其值为 STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT 之一。
OBJPROP_LEVELWIDTH	203	int	设置/获取对象水平线宽度的整数值。可以从 1 到 5。

OBJPROP_FIRSTLEVEL+n 210+n	int	斐波纳契对象水平线编号，此处 n 为设置/获取第 n 条水平线编号。可以是从 0 到 31 。

- 对象可视性

对象显示的图表时段。在 ObjectSet()函数中设置 OBJPROP_TIMEFRAMES 属性。

常数	值	描述
OBJ_PERIOD_M1	0x0001	对象只在 1 分钟图表中显示。
OBJ_PERIOD_M5	0x0002	对象只在 5 分钟图表中显示。
OBJ_PERIOD_M15	0x0004	对象只在 15 分钟图表中显示。
OBJ_PERIOD_M30	0x0008	对象只在 30 分钟图表中显示。
OBJ_PERIOD_H1	0x0010	对象只在 1 小时图表中显示。
OBJ_PERIOD_H4	0x0020	对象只在 4 小时图表中显示。
OBJ_PERIOD_D1	0x0040	对象只在日图表中显示。
OBJ_PERIOD_W1	0x0080	对象只在周图表中显示。
OBJ_PERIOD_MN1	0x0100	对象只在月图表中显示。
OBJ_ALL_PERIODS	0x01FF	对象在所有时段图表中显示。
NULL	0	对象在所有时段图表中显示。
EMPTY	-1	在所有时段图表中不显示。

- 未初始化原因代码

使用 UninitializeReason() 函数返回未初始化原因代码。可以是以下任意值:

常数	值	描述
	0	脚本独立完成执行。
REASON_REMOVE	1	从图表中删除智能交易。
REASON_RECOMPILE	2	重新编译智能交易。
REASON_CHARTCHANGE	3	改变了图表货币对和时段。
REASON_CHARTCLOSE	4	关闭图表。
REASON_PARAMETERS	5	用户改变了输入参数。

REASON_ACCOUNT	6	其他账户已激活。

- 特殊常量

特殊常量用于描述参数和变量状态。可以是以下任意值:

常数	值	描述
NULL	0	表明字符串为空值。
EMPTY	-1	表明参数为空值。
EMPTY_VALUE	0x7FFFFFFF	默认自定义指标为空值。
CLR_NONE	0xFFFFFFFF	表明颜色为空值。
WHOLE_ARRAY	0	配合数组函数应用，表示数组所有元素将被处理。

- 错误代码

GetLastError()函数返回出错代码，错误代码常量定义在 stderror.mqh 文件里。想要获取错误的文本信息，请调用在 stdlib.mqh 文件中定义的 ErrorDescription()函数。

```
#include <stderror.mqh>
#include <stdlib.mqh>
void SendMyMessage(string text)
  {
   int check;
   SendMail("some subject", text);
   check=GetLastError();
   if(check!=ERR_NO_ERROR) Print("Cannot send message, error: ",Error描述(check));
  }
```

从交易服务器返回的错误码:

常数	值	描述
ERR_NO_ERROR	0	没有错误返回。
ERR_NO_RESULT	1	没有错误返回，但结果不明。
ERR_COMMON_ERROR	2	一般错误。
ERR_INVALID_TRADE_PARAMETERS	3	无效交易参数。
ERR_SERVER_BUSY	4	交易服务器忙。
ERR_OLD_VERSION	5	客户端版本太旧。

ERR_NO_CONNECTION	6	没有连上交易服务器。
ERR_NOT_ENOUGH_RIGHTS	7	权限不足。
ERR_TOO_FREQUENT_REQUESTS	8	请求过于频繁。
ERR_MALFUNCTIONAL_TRADE	9	无效交易。
ERR_ACCOUNT_DISABLED	64	账户禁用。
ERR_INVALID_ACCOUNT	65	无效账户。
ERR_TRADE_TIMEOUT	128	交易超时。
ERR_INVALID_PRICE	129	无效价格。
ERR_INVALID_STOPS	130	无效平仓。
ERR_INVALID_TRADE_VOLUME	131	无效交易量。
ERR_MARKET_CLOSED	132	市场关闭。
ERR_TRADE_DISABLED	133	交易被禁止。
ERR_NOT_ENOUGH_MONEY	134	资金不足。
ERR_PRICE_CHANGED	135	价格已变动。
ERR_OFF_QUOTES	136	无报价。
ERR_BROKER_BUSY	137	经纪商忙。
ERR_REQUOTE	138	重新报价。
ERR_ORDER_LOCKED	139	定单被锁定。
ERR_LONG_POSITIONS_ONLY_ALLOWED	140	只允许多头头寸。
ERR_TOO_MANY_REQUESTS	141	请求过多。
ERR_TRADE_MODIFY_DENIED	145	因为定单过于接近市价，修改被拒绝。
ERR_TRADE_CONTEXT_BUSY	146	交易系统忙。
ERR_TRADE_EXPIRATION_DENIED	147	交易过期被经纪商拒绝。
ERR_TRADE_TOO_MANY_ORDERS	148	开仓和挂单总数已达到经纪商的限定。

MQL4 运行时错误代码:

常数	值	描述
ERR_NO_MQLERROR	4000	没有错误。
ERR_WRONG_FUNCTION_POINTER	4001	错误的函数指针。
ERR_ARRAY_INDEX_OUT_OF_RANGE	4002	数组下标越界。

ERR_NO_MEMORY_FOR_CALL_STACK	4003	没有足够内存开设函数调用堆栈。
ERR_RECURSIVE_STACK_OVERFLOW	4004	递归堆栈溢出。
ERR_NOT_ENOUGH_STACK_FOR_PARAM	4005	没有足够堆栈储存参数。
ERR_NO_MEMORY_FOR_PARAM_STRING	4006	没有足够内存储存字符串参数。
ERR_NO_MEMORY_FOR_TEMP_STRING	4007	没有足够内存保存临时字符串。
ERR_NOT_INITIALIZED_STRING	4008	没有初始化字符串。
ERR_NOT_INITIALIZED_ARRAYSTRING	4009	在数组中没有初始化字符串。
ERR_NO_MEMORY_FOR_ARRAYSTRING	4010	没有足够内存保存字符串数组。
ERR_TOO_LONG_STRING	4011	字符串过长。
ERR_REMAINDER_FROM_ZERO_DIVIDE	4012	余额除零。
ERR_ZERO_DIVIDE	4013	除零。
ERR_UNKNOWN_COMMAND	4014	未知命令。
ERR_WRONG_JUMP	4015	错误跳转(不产生错误)。
ERR_NOT_INITIALIZED_ARRAY	4016	没有初始化数组。
ERR_DLL_CALLS_NOT_ALLOWED	4017	不许调用 DLL 。
ERR_CANNOT_LOAD_LIBRARY	4018	不能加载库。
ERR_CANNOT_CALL_FUNCTION	4019	不能调用函数。
ERR_EXTERNAL_CALLS_NOT_ALLOWED	4020	不许调用智能交易函数。
ERR_NO_MEMORY_FOR_RETURNED_STR	4021	没有足够内存保存函数返回的字符串。
ERR_SYSTEM_BUSY	4022	系统忙 (不产生错误)。
ERR_INVALID_FUNCTION_PARAMSCNT	4050	非法函数参数个数。
ERR_INVALID_FUNCTION_PARAM 值	4051	非法函数参数值。
ERR_STRING_FUNCTION_INTERNAL	4052	字符串函数内部出错。
ERR_SOME_ARRAY_ERROR	4053	数组错误。
ERR_INCORRECT_SERIESARRAY_USING	4054	使用了错误的序列数组。
ERR_CUSTOM_INDICATOR_ERROR	4055	自定义指标错误。
ERR_INCOMPATIBLE_ARRAYS	4056	数组不兼容。
ERR_GLOBAL_VARIABLES_PROCESSING	4057	全局变量处理错误。
ERR_GLOBAL_VARIABLE_NOT_FOUND	4058	全局变量未找到。

ERR_FUNC_NOT_ALLOWED_IN_TESTING	4059	不许在测试模式使用函数。
ERR_FUNCTION_NOT_CONFIRMED	4060	函数不被允许。
ERR_SEND_MAIL_ERROR	4061	邮件发送错误。
ERR_STRING_PARAMETER_EXPECTED	4062	要求字符串参数。
ERR_INTEGER_PARAMETER_EXPECTED	4063	要求整数参数。
ERR_DOUBLE_PARAMETER_EXPECTED	4064	要求双精度参数。
ERR_ARRAY_AS_PARAMETER_EXPECTED	4065	要求用数组作参数。
ERR_HISTORY_WILL_UPDATED	4066	正在更新历史数据。
ERR_TRADE_ERROR	4067	交易执行出错。
ERR_END_OF_FILE	4099	到了文件尾。
ERR_SOME_FILE_ERROR	4100	文件错误。
ERR_WRONG_FILE_NAME	4101	错误文件名称。
ERR_TOO_MANY_OPENED_FILES	4102	打开文件过多。
ERR_CANNOT_OPEN_FILE	4103	不能打开文件。
ERR_INCOMPATIBLE_FILEACCESS	4104	不兼容文件访问。
ERR_NO_ORDER_SELECTED	4105	没有选择定单。
ERR_UNKNOWN_SYMBOL	4106	未知货币对。
ERR_INVALID_PRICE_PARAM	4107	交易函数的价格参数无效。
ERR_INVALID_TICKET	4108	无效定单。
ERR_TRADE_NOT_ALLOWED	4109	不允许交易。在智能交易中勾选“Allow live trading(允许实时交易)”。
ERR_LONGS_NOT_ALLOWED	4110	不允许多头，检查智能交易属性。
ERR_SHORTS_NOT_ALLOWED	4111	不允许空头，检查智能交易属性。
ERR_OBJECT_ALREADY_EXISTS	4200	对象已经存在。
ERR_UNKNOWN_OBJECT_PROPERTY	4201	未知对象属性。
ERR_OBJECT_DOES_NOT_EXIST	4202	对象不存在。
ERR_UNKNOWN_OBJECT_TYPE	4203	未知对象类型。
ERR_NO_OBJECT_NAME	4204	缺少对象名称。
ERR_OBJECT_COORDINATES_ERROR	4205	对象坐标错误。
ERR_NO_SPECIFIED_SUBWINDOW	4206	没有找到子窗口。

ERR_SOME_OBJECT_ERROR	4207 对象函数错误。

预定义变量

对于每个可执行的 MQL4 程序,在程序启动时，诸多的预定义变量可以轻松应对图表中的价格状态：智能交易、脚本或是自定义指标。

库可以使用模块中变量。

为了能安全、快速地访问这些数据，客户端程序会为每个单独启动的程序提供预定义变量的本地副本。这些数据会在每次运行智能交易或自定义指标后自动更新，或调用 RefreshRates()函数更新。

- 获取最新卖价 Ask

double Ask

获取当前货币对的最新卖价。请调用 RefreshRates()函数更新数据。

参见 MarketInfo()。

示例:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
 {

OrderSend(Symbol(),OP_BUY,Lots,Ask,3,Bid-StopLoss*Point,Ask+TakePr
ofit*Point,
                "My order #2",3,D'2005.10.10 12:30',Red);
     return;
}
```

- 获取图表柱数 Bars

int Bars

获取当前图表中柱数。

参见 iBars()。

示例:

```
  int counter=1;
  for(int i=1;  i<=Bars;  i++)
    {
```

```
    Print(关闭[i-1]);  }
```

- 获取最新买价 Bid

double Bid

获取当前货币对的最新买价。请调用 RefreshRates()函数更新数据。

参见 MarketInfo()。

示例:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
    {

OrderSend("EURUSD",OP_SELL,Lots,Bid,3,Ask+StopLoss*Point,Bid-TakeP
rofit*Point,
                "My Order #2",3,D'2005.10.10 12:30',Red);
     return(0);
    }
```

- 获取每个柱子收盘价 Close[]

double Close[]

收盘价序列数组，包含当前图表中每个柱子的收盘价。

收盘价按逆序方式保存在数组元素中，即从最新柱子的收盘价到第一个柱子的收盘价。最新柱子的收盘价在数组中下标为 0，图表中第一个柱子收盘价，即最老柱子的收盘价，它的下标为 Bars-1。

参见 iClose()。

示例:

```
  int handle = FileOpen("file.csv", FILE_CSV|FILE_WRITE, "; ");
  if(handle>0)
    {
     // 写入表格的栏目标题
     FileWrite(handle, "Time; Open; High; Low; Close; Volume");
     // 写入数据
     for(int i=0;  i<Bars;  i++)
       FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i],
Volume[i]);
     FileClose(handle);
    }
```

- 获取汇率小数位数 Digits

  int Digits

  获取当前货币对汇率的小数位数。

  参见 MarketInfo()。

示例:

```
Print(DoubleToStr(Close[0], 小数位));
```

- 获取每个柱子最高价 High[]

double High[]

最高价序列数组，包含当前图表每个柱子的最高价。

最高价按逆序方式保存在数组元素中，即从最新柱子的最高价到第一个柱子的最高价。最新柱子的最高价在数组中下标为 0，图表中第一个柱子最高价，即最老柱子的最高价，它的下标为 Bars-1。

参见 iHigh()。

示例:

```
//---- 最大值
i=Bars-KPeriod;
if(counted_bars>KPeriod) i=Bars-counted_bars-1;
while(i>=0)
{
  double max=-1000000;
  k = i + KPeriod-1;
  while(k>=i)
  {
   price=High[k];
   if(max<price) max=price;
     k--;
   }
   HighesBuffer[i]=max;
   i--;
}
//----
```

- 获取每个柱子最低价 Low[]

double Low[]

最低价序列数组，包含当前图表中每个柱子的最低价。

最低价按逆序方式储存在数组元素中，即从最新柱子的最低价到第一个柱子的最低价。最新柱子的最低价在数组中下标为 0，图表中第一个柱子最低价，即最老柱子的最低价，它的下标为 Bars-1。

参见 iLow()。

示例:

```
//---- 最小值
   i=Bars-KPeriod;
   if(counted_bars>KPeriod) i=Bars-counted_bars-1;
   while(i>=0)
     {
      double min=1000000;
      k = i + KPeriod-1;
      while(k>=i)
        {
         price=Low[k];
         if(min>price) min=price;
         k--;
        }
      LowesBuffer[i]=min;
      i--;
     }
//----
```

- 获取每个柱子开盘价 Open[]

double Open[]

开盘价序列数组，包含当前图表每个柱子的开盘价。

开盘价按逆序方式储存在数组元素中，即从最新柱子的开盘价到第一个柱子的开盘价。最新柱子的开盘价在数组中下标为 0，图表中第一个柱子开盘价，即最老柱子的开盘价，它的下标为 Bars-1。

参见 iOpen()。

示例:

```
   i = Bars - counted_bars - 1;
   while(i>=0)
```

```
    {
     double high  = High[i];
     double low   = Low[i];
     double open  = Open[i];
     double close = Close[i];
     AccumulationBuffer[i] = (close-low) - (high-close);
     if(AccumulationBuffer[i] != 0)
       {
        double diff = high - low;
        if(0==diff)
           AccumulationBuffer[i] = 0;
        else
          {
           AccumulationBuffer[i] /= diff;
           AccumulationBuffer[i] *= Volume[i];
          }
       }
     if(i<Bars-1) AccumulationBuffer[i] += AccumulationBuffer[i+1];
     i--;
    }
```

- 获取当前货币对点大小 Point

double Point

返回图表中当前货币对一点的大小。

参见 MarketInfo()。

示例:

```
OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point);
```

- 获取每个柱子开盘时间 Time[]

datetime Time[]

开盘时间序列数组，包含当前图表中每个柱子的开盘时间。日期时间型数据按秒计算，是从 1970 年 1 月 1 日 00:00 开始的秒数。

开盘时间按逆序方式储存在数组元素中，即从最新柱子的开盘时间到第一个柱子的开盘时间。最新柱子的开盘时间在数组中下标为 0，图表中第一个柱子的开盘时间，即最老柱子的开盘时间，它的下标为 Bars-1。

参见 iTime()。

示例:

```
   for(i=Bars-2;  i>=0;  i--)
     {
      if(High[i+1] > LastHigh) LastHigh = High[i+1];
      if(Low[i+1] < LastLow)   LastLow  = Low[i+1];
      //----
      if(TimeDay(Time[i]) != TimeDay(Time[i+1]))
        {
         P = (LastHigh + LastLow + Close[i+1])/3;
         R1 = P*2 - LastLow;
         S1 = P*2 - LastHigh;
         R2 = P + LastHigh - LastLow;
         S2 = P - (LastHigh - LastLow);
         R3 = P*2 + LastHigh - LastLow*2;
         S3 = P*2 - (LastHigh*2 - LastLow);
         LastLow  = Open[i];
         LastHigh = Open[i];
        }
      //----
      PBuffer[i]  = P;
      S1Buffer[i] = S1;
      R1Buffer[i] = R1;
      S2Buffer[i] = S2;
      R2Buffer[i] = R2;
      S3Buffer[i] = S3;
      R3Buffer[i] = R3;
     }
```

- 获取每个柱子一跳成交量 Volume[]

double Volume[]

成交量序列数组，包含当前图表中每个柱子的一跳成交量。

一跳成交量按逆序方式储存在数组元素中，即从最新柱子的一跳成交量到第一个柱子的一跳成交量。最新柱子的一跳成交量在数组中下标为 0，图表中第一个柱子，即最老柱子的一跳成交量，它的下标为 Bars-1。

参见 iVolume()。

示例:

```
      if(i==0 && time0<i_time+periodseconds)
        {
         d_volume += Volume[0];
         if(Low[0]<d_low)   d_low = Low[0];
         if(High[0]>d_high) d_high = High[0];
         d_close = Close[0];
        }
      last_fpos = FileTell(ExtHandle);
      last_volume = Volume[i];
      FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
      FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
      FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
      FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
      FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
      FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);
```

程序运行

想让MQL4程序运行起来,就必须对它进行编译(按"编译" 按钮或F5键) 。在程序编译过程中不允许出现任何错误(允许有警告信息，但一定要对它进行具体分析)。 这样的话，在相应的目录terminal_dir\experts、terminal_dir\experts\indicators, 或 terminal_dir\experts\scripts中，需要创建一个扩展名为.EX4的可执行文件，也就是这个文件能够被用户运行起来。

用户通过鼠标可以将智能交易、自定义指标和脚本从客户端“导航”窗口拖动到相应的图表上(拖曳技术)，它们会附加到已打开的任一图表上。MQL4程序只有在客户端已启动的基础上才能运行。

要停止智能交易运行,必须从图表的上下文菜单中删除它(智能交易系统-取消)。“启用智能交易”按钮状态将会直接影响智能交易的运行。

要使自定义指标停止运行，应该将它从图表中删除。

自定义指标和智能交易一直运行着，直到用户从图表中明确地删除它们为止。有关附加的智能交易和自定义指标的信息在客户端启动之时保存于客户端内。在脚本已经完成操作之后、当前图表被关闭或其状态已经发生改变、或当客户端中断运行时，脚本都只执行一次任务，并被自动删除掉。由于不保存运行状态信息，当客户端再次重新启动时脚本并不自动运行。

在同一个图表内，智能交易、脚本和更多数量的指标可以同时运行。

- 程序运行

程序附加到图表之后，它立即开始与 init()函数协同运行。在客户端开始运行和历史数据（仅与智能交易有关，而与指标无关）被另外加载之后、当货币对和图表时段改变了、在 MetaEditor 里重新编译之后、或者从智能交易或自定义指标的设置窗口改变了输入参数，附加到图表上的智能交易和自定义指标中 init()函数将会运行。当计算结果发生变化时，智能交易同样也会被初始化。

每一个附加到图表上的程序可以借助 deinit()函数完成善后工作。当客户端关机、图表被关闭、在货币对或图表时段改变之前、程序重编译成功、改变了输入参数或计算结果改变了，deinit()函数都会运行。在 deinit()函数执行期间,用户可以使用 UninitializeReason()函数查看未初始化成功的原因。
deinit()函数必须在 2.5 秒内执行完毕， 如果函数在这段时间内没有完成它的工作,它将会被强制完成。脚本是这一条规则的例外,由于它们能正常地自主完成它们的工作,它的运行不取决于任何外界的命令。如果一个脚本工作时间过长(例如，永无止境的死循环)，用户可以应用外部命令结束它的运行 (从图表的上下文菜单中删除脚本、在原有的图表上添加新的脚本、改变图表的货币对或时段)。在这种情况下,deinit()函数同样被限制在 2.5 秒内。

在收到新报价时,智能交易和自定义指标中 start()函数将会被执行。当新报价进入时，如果 start()函数还在忙着处理前面的报价，智能交易会忽略新报价而不予处理。在 start()函数运行期间，所有新收的报价都会被一跳而过，直到当前 start()函数运行完成为止。从那之后,仅在收到源源不断的新报价时，start()函数才会运行。对于自定义的指标,由于收到了新报价，当前图表货币对或时段发生了变化，start()函数将会重新运行。Start()函数是否运行还取决于“启用/禁用智能交易”按钮状态，当“启用/禁用智能交易”按钮处于禁用状态时，start()函数不会运行，但是，当通过该按钮切换到禁用状态时，不会中断当前 start()函数运行。在智能交易属性窗口打开时，start()函数将停止运行，在其执行期间也不会被打开。

从图表中卸载程序、改变货币对或图表时段、计算结果发生了变化、关闭图表、客户端关机，都将会中断程序的执行。如果 start()函数还在已发出命令让它停止工作的时刻运行，那留给它的工作时间依然被限制在 2.5 秒之内。程序知道用内置 IsStopped()函数试着把它关闭并正确地结束它的工作。

脚本的执行不依赖收到的报价。在货币对或图表时段发生改变时，脚本将完成它的工作并从客户端上卸载下来。

脚本和智能交易运行在自己的线程里，自定义指标则工作在主接口线程上。但是，如果一个自定义指标被 iCustom() 函数调用，这个指标就运行在调用它的程序线程中，库(导入)函数也同样工作在调用程序的线程里。

- 导入函数调用

为了在 mql4 程序执行期间导入函数，采用了所谓的“滞后联编”。这就意味着,导入函数被调用前,不会被加载相应的模块(EX4 或 DLL)。MQL4 和 DLL 库在调用模块线程下运行。

这里不推荐使用全路径模块名 Drive:\Directory\FileName.Ext 加载模块。MQL4 库是 从 terminal_dir\experts\libraries 文件夹中载入进来。如果库中未找到，将会尝试从 terminal_dir\experts 文件夹下加载。

系统动态链接库(DLL)按照操作系统规则加载。如果一个 DLL 已经加载过(例如，被其他智能交易系统，或从同时启动的另一个客户端程序加载)，则将直接引用已加载过的库，否则，搜索会按照以下顺序进行:

1. terminal_dir\experts\libraries 目录。
2. terminal_dir 目录，即客户端程序启动目录。
3. 当前目录。
4. windows_dir\SYSTEM32 系统目录(或是 win98 下 windows_dir\SYSTEM)。
5. windows_dir 目录，即操作系统安装目录。
6. 系统环境变量 PATH 中列出的目录。

如果一个 DLL 运行时调用了另一个 DLL，而后者又不可用，那么前者就不会被加载。

与系统库不同，自定义库(MQL4)是每个调用模块各自单独加载，而不管是否被其它模块独立地加载过。例如，调用者.ex4 模块可以从 lib1.ex4 和 lib2.ex4 库中调用函数，反过来，lib1.ex4 库也可以从 lib2.ex4 库中调用函数。在这种情况下，一个以上 lib1.ex4 副本和两份 lib2.ex4 副本被加载，根本不考虑所有的调用都来自同一个调用者.ex4 模块。

从 DLL 导入到 MQL 4 程序的函数必须符合 Windows API 函数的链接规则。为了确保在程序源码里符合这样的规定，我们用 C 或 C++ 语言编程时使用关键字 __stdcall 予以说明，表示是用微软公司编译器编译。上述链接规定有以下特点:

-为了将函数参数以适当的方式置入堆栈，调用函数(这种情况下，是一个 MQL4 程序) 必须"看得见"被调用函数的原型(来自 DLL 导入函数)；

- 调用函数(这种情况下，是一个 MQL4 程序) 以逆序方式将参数置入堆栈，也就是说，从右到左；它就是以这种顺序让导入函数读取传递给它的参数；

- 参数通过值传递，除那些明确说明通过引用传递之外；

- 在读取传递给它的参数后，导入函数本身将会自行清除堆栈。

在描述导入函数原型时，由于所有的参数必须明确地传递给导入函数，所以参数的默认值是无用的。

如果调用导入函数失败(智能交易设置不允许 DLL 导入，或是由于一些原因导致无法加载相应的库)，智能交易会停止运行并把相关信息输出到“智能交易停止”日志。另外，智能交易只有在重新初始化之后才能启动。重新编译或打开智能交易属性表并按 OK 键，其结果就是智能交易系统重新初始化。

- 运行时错误代码

在客户端程序运行子系统时，执行一个 MQL4 程序发生了异外情况，其错误代码将被保存下来。对于每一个 MQL4 程序执行，存在一个特殊的 last_error 变量。在 init()函数运行之前，last_error 变量必须归零。如果在计算阶段或调用内置函数时发生错误，last_error 变量会收到相应的错误代码。存储在这个变量中的值可以用 GetLastError() 函数访问，之后，last_error 变量将归零。

这里罗列了直接导致程序立即停止运行的关键错误：

常数	值	描述
ERR_WRONG_FUNCTION_POINTER	4001	在调用内部函数时，发现了错误的函数指针
ERR_NO_MEMORY_FOR_CALL_STACK	4003	在调用内部函数时，不能为函数调用堆栈再分配内存
ERR_RECURSIVE_STACK_OVERFLOW	4004	递回调用函数时，数据堆栈溢出
ERR_NO_MEMORY_FOR_PARAM_STRING	4006	在调用内部函数时，不能为作为函数参数的字符串分配内存
ERR_NO_MEMORY_FOR_TEMP_STRING	4007	不能为字符串操作分配临时缓冲区
ERR_NO_MEMORY_FOR_ARRAYSTRING	4010	赋值时，不能为数组中字符串重新分配内存
ERR_TOO_LONG_STRING	4011	赋值时，太长的字符串被送到服务缓冲区（不能再为服务缓冲区分配内存）
ERR_REMAINDER_FROM_ZERO_DIVIDE	4012	余数除 0 错误
ERR_ZERO_DIVIDE	4013	除 0 错误
ERR_UNKNOWN_COMMAND	4014	未知命令

如果在产生致命错误时程序停止了工作，这些错误代码可能被下次启动的程序用 GetLastError()函数读取，也可被非初始化函数读取。在程序或非初始化函数开始运行之前，last_error 变量不会归零。

这里罗列了调用导入函数时发生的致命错误，这些错误会引起智能交易或自定义指标立即停止启动函数的执行，直到用户重新初始化为止。

常数	值	描述
ERR_CANNOT_LOAD_LIBRARY	4018	调用导入函数时,载入 DLL 或 EX4 库发生错误
ERR_CANNOT_CALL_FUNCTION	4019	调用导入函数时， 发现 DLL 或 EX4 库不包含被调用函数
ERR_DLL_CALLS_NOT_ALLOWED	4017	调用导入 DLL 函数时，发现禁止 DLL 导入
ERR_EXTERNAL_CALLS_NOT_ALLOWED	4020	调用导入 EX4 函数时，发现禁止导入外部 EX4

其他错误不中断程序执行。

常数	值	描述
ERR_ARRAY_INDEX_OUT_OF_RANGE	4002	超界访问数组
ERR_NOT_INITIALIZED_STRING	4008	未初始化字符串；没有值赋给在表达式中充当操作数的字符串
ERR_NOT_INITIALIZED_ARRAYSTRING	4009	未初始化字符串数组；没有值赋给在表达式中充当操作数的字符串
ERR_NO_MEMORY_FOR_RETURNED_STR	4021	不能为函数返回的字符串重新分配内存

从不产生 ERR_NO_MQLERROR (4000) 代码。

这里罗列了可能只是由于软件或硬件故障而产生的许多错误。如果下列描述的错误反复出现，应与开发商联络。

常数	值	描述
ERR_WRONG_FUNCTION_POINTER	4001	调用内部函数时，发现函数指针错误
ERR_UNKNOWN_COMMAND	4014	未知命令
ERR_NOT_INITIALIZED_ARRAY	4016	未初始化数组
ERR_INVALID_FUNCTION_PARAMSCNT	4050	函数参数个数不正确
ERR_STRING_FUNCTION_INTERNAL	4052	字符串函数内部出错
ERR_TRADE_ERROR	4067	交易函数执行出错
ERR_SOME_OBJECT_ERROR	4207	对象函数出错

这里罗列的是经常修改 last_error 变量值的函数。

函数	错误代码
AccountFreeMarginCheck	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_UNKNOWN_SYMBOL (4106), ERR_NOT_ENOUGH_MONEY (134)
OrderSend	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), ERR_LONGS_NOT_ALLOWED (4110), ERR_SHORTS_NOT_ALLOWED (4111), 交易服务器返回的错误代码
OrderClose	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 交易服务器返回的错误代码
OrderCloseBy	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 交易服务器返回的错误代码
OrderDelete	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 交易服务器返回的错误代码
OrderModify	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051),

	ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), 交易服务器返回的错误代码
GetLastError	ERR_NO_ERROR (0)

如果有错误发生，下面这些函数就会修改 last_error 变量值。

函数	错误代码
ArrayBsearch	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayCopy	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayCopyRates	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062),
ArrayCopySeries	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_HISTORY_WILL_UPDATED (4066), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayDimension	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
ArrayGetAsSeries	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
ArrayInitialize	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)

ArrayIsSeries	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
ArrayMaximum	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayMinimum	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayRange	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArrayResize	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
ArraySetAsSeries	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
ArraySize	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
ArraySort	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileClose	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileDelete	ERR_WRONG_FILE_NAME (4101), ERR_SOME_FILE_ERROR (4100)
FileFlush	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileIsEnding	ERR_INVALID_FUNCTION_PARAMVALUE (4051)

FileIsLineEnding	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileOpen	ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103)
FileOpenHistory	ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103)
FileReadArray	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_ARRAY_ERROR (4053), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099)
FileReadDouble	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099)
FileReadInteger	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099)
FileReadNumber	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099)
FileReadString	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_TOO_LONG_STRING (4011), ERR_END_OF_FILE (4099)
FileSeek	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileSize	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
FileTell	ERR_INVALID_FUNCTION_PARAMVALUE (4051)

FileWrite	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100)
FileWriteDouble	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100)
FileWriteInteger	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100)
FileWriteString	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_STRING_PARAMETER_EXPECTED (4062)
FileWriteArray	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100),
GlobalVariableCheck	ERR_STRING_PARAMETER_EXPECTED (4062)
GlobalVariableDel	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057)
GlobalVariableGet	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058)
GlobalVariablesDeleteAll	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057)
GlobalVariableSet	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058)
GlobalVariableSetOnCondition	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058)
iCustom	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
technical indicators, series access functions	ERR_HISTORY_WILL_UPDATED (4066)
technical indicators OnArray	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)

IndicatorBuffers	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
IndicatorDigits	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
IndicatorShortName	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
MarketInfo	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_UNKNOWN_SYMBOL (4106), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
MathArccos	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
MathArcsin	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
MathMod	ERR_ZERO_DIVIDE (4013)
MathSqrt	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
MessageBox	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062)
ObjectCreate	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_ALREADY_EXISTS (4200), ERR_NO_SPECIFIED_SUBWINDOW (4206)
ObjectDelete	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
ObjectDescription	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
ObjectFind	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204)

ObjectGet	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
ObjectGetFiboDescription	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
ObjectGetShiftByValue	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205)
ObjectGetValueByShift	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205)
ObjectMove	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_DOES_NOT_EXIST (4202)
ObjectName	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
ObjectSet	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
ObjectSetText	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
ObjectSetFiboDescription	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE

	(4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
ObjectType	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
OrderClosePrice	ERR_NO_ORDER_SELECTED (4105)
OrderCloseTime	ERR_NO_ORDER_SELECTED (4105)
OrderComment	ERR_NO_ORDER_SELECTED (4105)
OrderCommission	ERR_NO_ORDER_SELECTED (4105)
OrderExpiration	ERR_NO_ORDER_SELECTED (4105)
OrderLots	ERR_NO_ORDER_SELECTED (4105)
OrderMagicNumber	ERR_NO_ORDER_SELECTED (4105)
OrderOpenPrice	ERR_NO_ORDER_SELECTED (4105)
OrderOpenTime	ERR_NO_ORDER_SELECTED (4105)
OrderPrint	ERR_NO_ORDER_SELECTED (4105)
OrderProfit	ERR_NO_ORDER_SELECTED (4105)
OrderStopLoss	ERR_NO_ORDER_SELECTED (4105)
OrderSwap	ERR_NO_ORDER_SELECTED (4105)
OrderSymbol	ERR_NO_ORDER_SELECTED (4105)
OrderTakeProfit	ERR_NO_ORDER_SELECTED (4105)
OrderTicket	ERR_NO_ORDER_SELECTED (4105)
OrderType	ERR_NO_ORDER_SELECTED (4105)
PlaySound	ERR_WRONG_FILE_NAME (4101)
SendFTP	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062)
SendMail	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNCTION_NOT_CONFIRMED (4060), ERR_SEND_MAIL_ERROR (4061)
SetIndexArrow	ERR_INVALID_FUNCTION_PARAMVALUE (4051)

SetIndexBuffer	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056)
SetIndexDrawBegin	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
SetIndexEmptyValue	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
SetIndexLabel	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062)
SetIndexShift	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
SetIndexStyle	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
SetLevelValue	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
Sleep	ERR_CUSTOM_INDICATOR_ERROR (4055)
StringFind	ERR_STRING_PARAMETER_EXPECTED (4062)
StringGetChar	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
StringLen	ERR_STRING_PARAMETER_EXPECTED (4062)
StringSetChar	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_NOT_INITIALIZED_STRING (4008), ERR_TOO_LONG_STRING (4011), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
StringSubstr	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_TOO_LONG_STRING (4011)
StringTrimLeft	ERR_STRING_PARAMETER_EXPECTED (4062)
StringTrimRight	ERR_STRING_PARAMETER_EXPECTED (4062)
WindowIsVisible	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059)
WindowFind	ERR_FUNC_NOT_ALLOWED_IN_TESTING

	(4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008)
WindowHandle	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008)
WindowScreenShot	ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051)

下列函数从不改变 last_error 变量值。

AccountBalance, AccountCompany, AccountCredit, AccountCurrency, AccountEquity, AccountFreeMargin, AccountLeverage, AccountMargin, AccountName, AccountNumber, AccountProfit, AccountServer, Alert, CharToStr, Comment, Day, DayOfWeek, DayOfYear, DoubleToStr, GetTickCount, HideTestIndicators, Hour, IndicatorCounted, IsConnected, IsDemo, IsDllsAllowed, IsExpertEnabled, IsLibrariesAllowed, IsOptimization, IsStopped, IsTesting, IsTradeAllowed, IsTradeContextBusy, IsVisualMode, MathAbs, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathMax, MathMin, MathPow, MathRand, MathRound, MathSin, MathSrand, MathTan, Minute, Month, NormalizeDouble, ObjectsDeleteAll, ObjectsTotal, OrderSelect, OrdersHistoryTotal, Period, Print, RefreshRates, Seconds, SetLevelStyle, StringConcatenate, StrToTime, StrToDouble, Symbol, TerminalCompany, TerminalName, TerminalPath, TimeCurrent, TimeDay, TimeDayOfWeek, TimeDayOfYear, TimeHour, TimeLocal, TimeMinute, TimeMonth, TimeSeconds, TimeToStr, TimeYear, UninitializeReason, WindowBarsPerChart, WindowFirstVisibleBar, WindowPriceOnDropped, WindowRedraw, WindowTimeOnDropped, WindowsTotal, WindowOnDropped, WindowXOnDropped, WindowYOnDropped, Year

账户信息

访问已激活账户信息的一组函数。

- 获取账户余额 AccountBalance()

```
double AccountBalance()
```

返回当前账户余额(账户中钱数)。

示例:

```
Print("账户余额= ",AccountBalance());
```

- 获取账户信用额度 AccountCredit()

double AccountCredit()

返回当前账户信用额度。

示例:

```
Print(" 账户信用 ", AccountCredit());
```

- 获取账户注册的外汇公司名 AccountCompany()

string AccountCompany()

返回当前账户注册的外汇公司名。

示例:

```
Print("账户注册公司名", AccountCompany());
```

- 获取账户货币名称 AccountCurrency()

string AccountCurrency()

返回当前账户所用的货币名称。

示例:

```
Print(" 账户货币名 ", AccountCurrency());
```

- 获取账户净值 AccountEquity()

double AccountEquity()

返回当前账户的资金净值。资金净值计算取决于交易服务器的设置。

示例:

```
Print("账户净值 = ",AccountEquity());
```

- 获取账户可用保证金 AccountFreeMargin()

double AccountFreeMargin()

返回当前账户的可用保证金金额。

示例:

```
Print("账户可用保证金 = ",AccountFreeMargin());
```

- 检查可用保证金 AccountFreeMarginCheck()

double AccountFreeMarginCheck(string symbol, int cmd, double volume)

当前账户以现价在指定的位置开仓后还剩余的保证金。如果保证金余额不足，将会生成错误 134(ERR_NOT_ENOUGH_MONEY)。

参数:

symbol	-	交易货币对。
Cmd	-	交易类型，可能是 OP_BUY 或者 OP_SELL。
volume	-	手数。

示例:

```
if(AccountFreeMarginCheck(Symbol(),OP_BUY,Lots)<=0 ||
GetLastError()==134) return;
```

- 可用保证金计算模式 AccountFreeMarginMode()

double AccountFreeMarginMode()

允许开仓的可用保证金计算模式。计算方式可能采取以下值:

0 - 浮动止赢/止损不参与计算
1 - 浮动止赢和止损在开仓部位上参与保证金余额计算;
2 - 只有赢利参与计算，不考虑当前开仓部位的亏损;
3 - 只有亏损额参与计算, 不考虑当前开仓部位的亏损。

示例:

```
if(AccountFreeMarginMode()==0)
   Print("浮动止赢/止损不用。");
```

- 获取当前账户杠杆比率 AccountLeverage()

int AccountLeverage()

返回当前账户杠杆比率。

示例:

```
Print("账户#",AccountNumber(), " 杠杆比率 ", AccountLeverage());
```

- 获取账户已用保证金 AccountMargin()

double AccountMargin()

返回当前账户已用保证金金额。

示例:

```
Print("账户已用保证金 ", AccountMargin());
```

- 获取账户名称 AccountName()

string AccountName()

返回当前账户名称。

示例:

```
Print("账户名称", AccountName());
```

- 获取当前账户账号 AccountNumber()

int AccountNumber()

返回当前账户的账号。

示例:

```
Print("账户号码 ", AccountNumber());
```

- 获取账户赢利金额 AccountProfit()

double AccountProfit()

返回账户赢利金额。

示例:

```
Print(" 账户赢利 ", AccountProfit());
```

- 获取连接服务器名称 AccountServer()

string AccountServer()

返回连接服务器的名称。

示例:

```
Print("服务器名称 ", AccountServer());
```

- 获取停止交易标准 AccountStopoutLevel()

int AccountStopoutLevel()

返回停止交易标准。

示例:

```
Print("停止标准 = ", AccountStopoutLevel());
```

- 获取停止交易标准的计算方式 AccountStopoutMode()

Int AccountStopoutMode()

返回停止交易标准的计算方式。计算方式可以采用下列值:

0 - 计算保证金和净值之间的百分比;

1 - 比较剩余保证金标准和绝对值。

示例:

```
int level=AccountStopoutLevel();
if(AccountStopoutMode()==0)
   Print("停止标准= ",level, "%");
else
   Print("停止标准= ", level, " ", AccountCurrency());
```

数组处理函数

用于数组处理的一组函数。

数组的最大维数为四维。每一维索引编号为 0 至数组大小-1。例如，在 50 个元素的一维数组中，调用第一个元素用 array[0]，最后一个元素用 array[49]。

使用下列函数(除了那些能够改变数组定性和定量特性的函数外)，我们能够处理预定义的时间序列函数 Time[], Open[], High[], Low[], Close[], Volume[] 。

- 搜索数组 ArrayBsearch()

```
int ArrayBsearch(double array[], double value, void count, void start, void direction)
```

如果在数组中找到，返回第一次找到的元素下标。如果未找到，则返回最接近搜索值的元素下标。

此函数不能用于字符串型数组或序列数组(不包括柱子开盘时间的序列数

组)。

注: 二进制查找只能在已排序的数组中进行。使用 ArraySort() 函数可以排序数值数组。

参数:

array[]	-	被搜索的数值数组
value	-	要搜索的值
count	-	要搜索的元素数量，默认搜索所有的数组
start	-	搜索的起始位置，默认从第一个元素开始
direction	-	搜索的方向，它可能使用下列值： MODE_ASCEND 顺序搜索, MODE_DESCEND 逆序搜索。

示例:

```
   datetime daytimes[];
   int      shift=10,dayshift;
   // Time[]中所有数值降序排列
   ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
   if(Time[shift]>=daytimes[0]) dayshift=0;
   else
     {

dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESC
END);
      if(Period()<PERIOD_D1) dayshift++;
     }
   Print(TimeToStr(Time[shift]),″ corresponds to ″,dayshift,″ day bar
opened at ″,
         TimeToStr(daytimes[dayshift]));
```

- 数组复制 ArrayCopy()

```
Int          void dest[], object source[], void start_dest,
ArrayCopy(   void start_source, void count)
```

复制一个数组数据到另外一个数组。 数组必须类型相同，只有同类型的 double、int、datetime、 color 和 bool 型数组之间可以被复制。

返回被复制元素的个数。

参数:

Dest[]	-	目标数组。
source[]	-	源数组。
start_dest	-	接收数据的目标数组起始下标，默认为 0。
start_source	-	读取数据的源数组起始下标，默认为 0 。
Count	-	要复制的元素个数。默认值为 WHOLE_ARRAY 常量。

示例:

```
  double array1[][6];
  double array2[10][6];
  // 用数据填充 array2 数组
  ArrayCopyRates(array1);
  ArrayCopy(array2,array1,0,0,60);
  // 现在 array2 有了来自于历史的前 10 柱数据 (第一个柱子索引为
[Bars-1])
  ArrayCopy(array2,array1,0,Bars*6-60,60);
  // 现在 array2 有了来自于历史的后 10 柱数据(最后一柱索引为[0])
```

- 复制柱子数据到二维数组 ArrayCopyRates()

```
int ArrayCopyRates(Void dest_array[], void symbol, void timeframe)
```

复制图表上柱子数据到一个二维数组，并返回已复制的柱数。如果返回-1，表示复制失败。数组的第二维有 6 个项目，分别是:

0 - 开盘时间
1 - 开盘价
2 - 最低价
3 - 最高价
4 - 收盘价
5 - 成交量

如果想从另一个图表获取数据(货币对名称/图表时段与当前不同)，而相应的图表又没有打开，需要从服务器下载必要的数据，那么出现这种状况是有可能的。这种情况下， 错误信息 ERR_HISTORY_WILL_UPDATED (4066 - 历史数据正在更新) 将被放到 last_error 变量中，并且不得不再次下载数据(查看范例 ArrayCopySeries())。

注:此数组通常用于向 DLL 函数传递数据。

没有真正为数据数组分配内存，也没有真正地执行复制。当存取这样的数组时，访问将被重定向。

参数:

dest_array[]	-	二维的双精度目标数组
Symbol	-	货币对名称(当前货币对名称)
Timeframe	-	时段，可以是任意时段值

示例:

```
double array1[][6];
ArrayCopyRates(array1,"EURUSD", PERIOD_H1);
Print("当前柱 ",TimeToStr(array1[0][0]),"开盘价", array1[0][1]);
```

- 复制柱子数据到一维数组 ArrayCopySeries()

```
int                Void array[], int series_index, void symbol,
ArrayCopySeries(   void timeframe)
```

复制一组柱子数据到一个数组，并返回复制的元素个数。

没有真正为数据数组分配内存，也没有真正地执行复制。当存取这样数组时，访问将被重定向。自定义指标中指标数组是例外情况，这种情况下，数组被真正复制。

如果数据从不同货币对/图表时段复制,可能会缺少数据。这种情况下，错误信息 ERR_HISTORY_WILL_UPDATED (4066 -历史数据正在更新)将被放到 last_error 变量中，在一段时间之后重新尝试复制。

注: 如果 series_index 是 MODE_TIME, 那么传递给函数的数组必须是日期时间型数组。

参数:

array[]	-	一维数值型目标数组。
series_index	-	序列数组识别符,必须是序列数组常量列表中识别符值。
symbol	-	当前货币对名称。
timeframe	-	图表时段，可以是图表时段常量列表中任意值。

示例:

```
datetime daytimes[];
int      shift=10,dayshift,error;
//---- 此 Time[] 数组降序排列
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
error=GetLastError();
if(error==4066)
  {
   //---- 试读两次以上
```

```
   for(int i=0; i<2;  i++)
     {
      Sleep(5000);
      ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
      //---- 检查当前每日柱子时间
      datetime last_day=daytimes[0];
      if(Year()==TimeYear(last_day) && Month()==TimeMonth(last_day)
&& Day()==TimeDay(last_day)) break;
     }
  }
if(Time[shift]>=daytimes[0]) dayshift=0;
else
  {

dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESC
END);
   if(Period()<PERIOD_D1) dayshift++;
  }
Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar
opened at ", TimeToStr(daytimes[dayshift]));
```

- 返回数组维数 ArrayDimension()

int ArrayDimension(object array[])

返回数组的维数。

参数:

```
array[]  -  数组。
```

示例:

```
int num_array[10][5];
int dim_size;
dim_size=ArrayDimension(num_array);
// dim_size=2
```

- 判断序列数组 ArrayGetAsSeries()

Bool ArrayGetAsSeries(object array[])

如果数组是按序列数组形式排列(数组元素从最后到最开始排序过)，返回

TRUE，否则，返回 FALSE。

参数:

array[]	-	需要检查的数组。

示例:

```
if(ArrayGetAsSeries(array1)==true)
  Print("array 1 作为序列数组排列");
else
  Print("array 1 正常排列(从左到右)");
```

- 数组初始化 ArrayInitialize()

int ArrayInitialize(void array[], double value)

数值型数组元素值设为同一个数值，返回已初始化的元素个数。

注:在自定义指标 init() 函数中不建议初始化指标缓冲区，在缓冲区分配和重分配内存时这种函数自动初始化为"空值"。

参数:

Array[]	-	需要初始化的数值型数组。
Value	-	设定的新值。

示例:

```
//----所有数组元素初始化为 2.1
double myarray[10];
ArrayInitialize(myarray,2.1);
```

- 判断数组是否序列化 ArrayIsSeries()

Bool ArrayIsSeries(Object array[])

如果数组经检查是序列的(Time[]、Open[]、Close[]、High[]、Low[]或Volume[])，返回 TRUE，否则，返回 FALSE。

参数:

array[]	-	需要检查的数组。

示例:

```
if(ArrayIsSeries(array1)==false)
  ArrayInitialize(array1,0);
```

```
   else
     {
      Print("序列数组不能被初始化!");
      return(-1);
     }
```

- 返回数组中最大值位置 ArrayMaximum()

int ArrayMaximum(Double array[], void count, void start)

找出数组元素中最大值。函数返回数组元素最大值的下标。

参数:

```
array[]   -   要搜索的数值数组。
count     -   要搜索的数组元素个数。
start     -   开始搜索的初始下标。
```

示例:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
int    maxValueIdx=ArrayMaximum(num_array);
Print("最大值 = ", num_array[maxValueIdx]);
```

- 返回数组中最小值位置 ArrayMinimum()

int ArrayMinimum(double array[], void count, void start)

找出数组中最小值的位置。函数返回数组中最小值的下标。

参数:

```
array[]   -   搜索的数值型数组。
count     -   要搜索的数组元素个数。
start     -   开始搜索的初始下标。
```

示例:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
int    minValueidx=ArrayMinimum(num_array);
Print("最小值 = ", num_array[minValueIdx]);
```

- 获取数组元素个数 ArrayRange()

int ArrayRange(Object array[], int range_index)

返回给定数组指定维数中元素的个数。由于下标从零开始，数组的大小要比最大下标数多 1 。

参数:

```
array[]       -   要检查的数组。
range_index   -   指定的维数。
```

示例:

```
int    dim_size;
double num_array[10,10,10];
dim_size=ArrayRange(num_array, 1);
```

- 重设数组大小 ArrayResize()

```
int ArrayResize(Void array[], int new_size)
```

设定数组第一维的新大小。如果执行成功的话，函数将返回新数组的元素个数，否则，返回 -1，数组大小并不改变。

注：函数执行完成后，在函数内局部定义和重设大小的数组将维持原样不变。 在函数被重新调用后，这种数组的大小将会与定义时大小有差异。

参数:

```
array[]    -   要重设大小的数组。
new_size   -   第一维数组大小的新值。
```

示例:

```
double array1[][4];
int    element_count=ArrayResize(array1, 20);
// 新的大小 - 80 个元素
```

- 序列化数组 ArraySetAsSeries()

```
Bool ArraySetAsSeries(void array[], bool set)
```

设定数组的排列方向。如果设置参数值为 TRUE，数组将按逆序排列，也就是说，数组元素下标为 0 的值是最后的值。如果其值为 FALSE，表明数组是一个正常的排列顺序，此函数原样返回以前的数组。

参数:

```
array[]   -   数值型数组。
set       -   数组索引顺序。
```

示例:

```
double macd_buffer[300];
double signal_buffer[300];
int    i,limit=ArraySize(macd_buffer);
ArraySetAsSeries(macd_buffer,true);

for(i=0;  i<limit;  i++)

macd_buffer[i]=iMA(NULL,0,12,0,MODE_EMA,PRICE_CLOSE,i)-iMA(NULL,0,
26,0,MODE_EMA,PRICE_CLOSE,i);

for(i=0;  i<limit;  i++)
   signal_buffer[i]=iMAOnArray(macd_buffer,limit,9,0,MODE_SMA,i);
```

- 返回数组大小 ArraySize()

int ArraySize(object array[])

返回数组元素的个数。对于一个一维数组，用 ArraySize()函数返回的大小和 ArrayRange(array,0)的结果相等。

参数:

```
array[]   -   任何类型的数组。
```

示例:

```
int count=ArraySize(array1);
for(int i=0;  i<count;  i++)
  {
   // 一些计算
  }
```

- 数组排序 ArraySort()

int ArraySort(void array[], void count, void start, void sort_dir)

根据数值型数组的第一维进行排序。ArraySort()函数不能排序序列数组。

参数:

```
array[]    -   被排序的数值型数组。
count      -   参加排序的元素个数。
```

start	-	排序的起始下标。
sort_dir	-	排序方式，可以选用下列值： MODE_ASCEND 顺序排列， MODE_DESCEND 逆序排列。

示例:

```
double num_array[5]={4,1,6,3,9};
// 数组包含 4,1,6,3,9 这些值
ArraySort(num_array);
// 排序后数组 1,3,4,6,9
ArraySort(num_array,WHOLE_ARRAY,0,MODE_DESCEND);
//排序后数组 9,6,4,3,1
```

检测当前客户端状态

这组函数可以检测客户端的当前状态，包括运行 MQL4 程序的环境状态。

- 获取最新产生的错误信息 GetLastError()

int GetLastError()

本函数先返回最新产生的错误信息，然后保存出错代码的 last_error 变量值归零，所以，再次调用 GetLastError()函数将返回 0。

示例:

```
int err;
int handle=FileOpen("somefile.dat", FILE_READ|FILE_BIN);
if(handle<1)
  {
   err=GetLastError();
   Print("错误(",err,"): ",ErrorDescription(err));
   return(0);
  }
```

- 判断连接状态 IsConnected()

Bool IsConnected()

本函数返回在客户端和执行数据中转任务的服务器之间主连接状态。如果成功建立到服务器的连接，返回 TRUE，否则，返回 FALSE。

示例:

```
  if(!IsConnected())
    {
     Print("没有连接!");
     return(0);
    }
  // 需要打开连接的智能交易主程序
  // ...
```

- 判断是否是模拟账户 IsDemo()

bool IsDemo()

如果智能交易在模拟账户里运行，返回 TRUE ，否则，返回 FALSE。

示例:

```
  if(IsDemo()) Print("在模拟账户运行");
  else Print("在真实账户运行");
```

- 判断是否允许调用 DLL 函数 IsDllsAllowed()

bool IsDllsAllowed()

如果智能交易允许调用 DLL 函数，返回 TRUE，否则，返回 FALSE。

参见 IsLibrariesAllowed(), IsTradeAllowed()。

示例:

```
  #import "user32.dll"
     int     MessageBoxA(int hWnd, string szText, string
szCaption,int nType);
  ...
  ...
  if(IsDllsAllowed()==false)
    {
     Print("DLL 调用不允许。智能交易不能运行。");
     return(0);
    }
    // 智能交易主程序调用外部 DLL 函数
    MessageBoxA(0,"an message","Message",MB_OK);
```

- 判断智能交易是否开启 IsExpertEnabled()

bool IsExpertEnabled()

如果启用智能交易运行，返回 TRUE，否则，返回 FALSE。

示例:

```
while(!IsStopped())
  {
   ...
   if(!IsExpertEnabled()) break;
  }
```

- 判断是否允许调用库函数 IsLibrariesAllowed()

Bool IsLibrariesAllowed()

如果智能交易允许调用库中函数，返回 TRUE，否则，返回 FALSE。

参见 IsDllsAllowed(), IsTradeAllowed()。

示例:

```
#import "somelibrary.ex4"
   int somefunc();
...
...
if(IsLibrariesAllowed()==false)
  {
   Print("不允许调用脚本库");
   return(0);
  }
// 智能交易主程序调用外部 DLL 函数
somefunc();
```

- 判断智能交易是否为优化模式 IsOptimization()

Bool IsOptimization()

如果智能交易运行在策略测试器的优化模式，返回 TRUE，否则，返回 FALSE。

示例:

```
if(IsOptimization()) return(0);
```

- 判断智能交易是否中止 IsStopped()

Bool IsStopped()

如果程序（一个智能交易程序或一个脚本程序）得到了停止运行的命令，返回 TRUE，否则，返回 FALSE。在客户端强制中止执行之前，程序还能继续运行 2.5 秒。

示例:

```
while(expr!=false)
  {
   if(IsStopped()==true) return(0);
   // 长时间运行的循环
   // ...
  }
```

- 判断智能交易是否在测试模式中运行 IsTesting()

bool IsTesting()

如果智能交易运行在测试模式中，返回 TRUE，否则，返回 FALSE。

示例:

```
if(IsTesting()) Print("测试中");
```

- 判断智能交易是否允许交易 IsTradeAllowed()

bool IsTradeAllowed()

如果智能交易程序允许交易，而且执行交易的线程没有被占用，返回 TRUE ，否则，返回 FALSE。

参见 IsDllsAllowed(), IsLibrariesAllowed(), IsTradeContextBusy()。

示例:

```
if(IsTradeAllowed()) Print("允许交易");
```

- 判断智能交易线程是否忙 IsTradeContextBusy()

Bool IsTradeContextBusy()

如果执行交易的线程被另一个智能交易占用，返回 TRUE，否则，返回 FALSE。

参见 IsTradeAllowed()。

示例:

```
if(IsTradeContextBusy()) Print("交易忙，请稍等");
```

- 判断智能交易是否用“可视模式”测试 IsVisualMode()

Bool IsVisualMode()

如果智能交易用“可视模式”测试，返回 TRUE，否则，返回 FALSE。

示例:

```
if(IsVisualMode()) Comment("可视模式开启");
```

- 获取未初始化原因 UninitializeReason()

int UninitializeReason()

返回智能交易、自定义指标和脚本的未初始化原因代码。返回值为未初始化原因代码之一。 本函数同样可以在函数 init() 中调用，用于分析上次运行出错原因。

示例:

```
// 这是范例
int deinit()
  {
   switch(UninitializeReason())
     {
      case REASON_CHARTCLOSE:
      case REASON_REMOVE:      CleanUp();  break;  // 清理和所有资源重分配
      case REASON_RECOMPILE:
      case REASON_CHARTCHANGE:
      case REASON_参数:
      case REASON_ACCOUNT:     StoreData();  break;   // 准备重新开始
     }
   //...
  }
```

客户端信息

本组函数返回客户端信息。

- 获取客户端所属公司名称 TerminalCompany()

string TerminalCompany()

返回客户端所属公司名称。

示例:

```
Print("公司名称 ",TerminalCompany());
```

- 获取客户端名称 TerminalName()

string TerminalName()

返回客户端名称。

示例:

```
Print("终端名称",TerminalName());
```

- 获取客户端文件目录 TerminalPath()

string TerminalPath()

返回路径，客户端程序所在的目录。

示例:

```
Print("工作目录",TerminalPath());
```

常规函数

常规用途函数，不涉及任何专用函数。

- 弹出警告窗口 Alert()

Void Alert(...)

弹出一个包含用户提示信息的警告窗口。参数可以是任意类型，总数不得超过 64 个。

数组不能作为参数传递给 alert()函数。

双精度型数据可以输出到小数点后 4 位。要想输出更高精度的数据，请使用 DoubleToStr()函数。

布尔型、日期时间型和颜色型数据作为数值型数据输出。

使用 TimeToStr()函数将日期时间型数据转换成字符串，再以字符串形式输出。

参见 Comment() 和 Print() 函数。

参数:

```
...   -   任意值，参数用逗号分隔，最多为 64 个。
```

示例:

```
  if(Close[0]>SignalLevel)
    Alert("收到收盘价", Close[0],"!!!");
```

- 在图表左上角标注信息 Comment()

Void Comment(...)

本函数可以在图表左上角标注用户说明。参数可以是任意类型，最多 64 个。

数组不能作为参数传递给 Comment()函数。

双精度型的数据可以输出到小数点后 4 位。要想输出更高精度的数据，请使用 DoubleToStr()函数。

布尔型、日期时间型和颜色型数据作为数值型数据输出。

使用 TimeToStr()函数将日期时间型数据转换成字符串，再以字符串形式输出。

参见 Comment() 和 Print() 函数。

参数:

```
...   -   任意值，参数用逗号分隔，最多为 64 个。
```

示例:

```
  double free=AccountFreeMargin();
  Comment("账户可用保证金: ",DoubleToStr(free,2),"\n","当前时间:
",TimeToStr(TimeCurrent()));
```

- 取回运行时间 GetTickCount()

int GetTickCount()

GetTickCount() 函数取回自从系统被启动以来已经过去的毫秒数。它决定于系统时间的设定。

示例:

```
int start=GetTickCount();
// 计算...
Print("计算时间：", GetTickCount()-start, " milliseconds.");
```

- 获取市场观察窗口中数据 MarketInfo()

double MarketInfo(string symbol, int type)

返回在市场观察窗口中列出的不同货币对的数据。当前货币对的说明存储在预定义变量中。

参数:

symbol	-	货币对。
type	-	请求返回定义的信息类型标识符，可以是请求标别符的任意值。

示例:

```
double bid   =MarketInfo("EURUSD",MODE_BID);
double ask   =MarketInfo("EURUSD",MODE_ASK);
double point =MarketInfo("EURUSD",MODE_POINT);
int    digits=MarketInfo("EURUSD",MODE_DIGITS);
int    spread=MarketInfo("EURUSD",MODE_SPREAD);
```

- 显示信息框 MessageBox()

int MessageBox(Void text, void caption, void flags)

MessageBox()函数可以创建、显示和控制信息框。信息框内包含应用程序定义的信息内容和标题，也可以是预定义的图标和按钮的任意组合。如果函数成功运行，返回值就是 MessageBox 函数返回码的其中之一。

由于本函数在接口线程内执行，而且还不能放慢速度，所以本函数不能从自定义指标中调用。

参数:

Text	-	包含要显示的任意文字。
caption	-	要显示的窗口标题。如果参数为 NULL，智能交易名称显示在标题上。

flags	-	决定窗口类型和操作的可选项。它们代表 messagebox 函数标志常量的一种组合。

示例:

```
  #include <WinUser32.mqh>
    if(ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30',
1.0045)==false)
    {
     int ret=MessageBox(" ObjectCreate() function returned the
"+GetLastError()+" error\nContinue?", "Question",
MB_YESNO|MB_ICONQUESTION);
     if(ret==IDNO) return(false);
    }
  // 继续
```

- 播放声音文件 PlaySound()

void PlaySound(string filename)

本函数播放一个声音文件。文件必须放在 terminal_dir\sounds 目录或子目录内。

参数:

Filename	-	音频文件名。

示例:

```
  if(IsDemo()) PlaySound("alert.wav");
```

- 输出结果 Print()

Void Print(...)

向智能交易日志输出文本信息。参数可以是任意类型，总数不得超过 64。

数组不能作为参数传递给 Print()函数。

双精度型数据可以输出到小数点后 4 位。要想输出更高精度的数据，请使用 DoubleToStr()函数。

布尔型、日期时间型和颜色型数据作为数值型数据输出。

为了以字符串形式输出日期时间型数据，可用 TimeToStr()函数转换后输出。

参见 Comment() 和 Alert() 函数。

参数:

...	-	任意值，参数用逗号分隔，最多为 64 个。

示例:

```
Print("当前可用保证金 ", AccountFreeMargin());
Print("当前时间 ", TimeToStr(TimeCurrent()));
double pi=3.141592653589793;
Print("PI 的值 ", DoubleToStr(pi,8));
// 输出:PI 值为 3.14159265
// 输出数组
for(int i=0; i<10; i++)
  Print(close[i]);
```

- 发送文件到 FTP 服务器 SendFTP()

bool SendFTP(String filename, void ftp_path)

发送文件到 FTP 服务器，其通信参数在“工具>选项>公开” 标签内设置。如果发送失败，返回 FALSE。

本函数在测试模式下不起作用，也不能从自定义指标中调用。

发送的文件必须在 terminal_directory\experts\files 目录或子目录内。

如果未设置 FTP 地址，或未提供访问密码，文件不会发送成功。

参数:

Filename	-	发送的文件。
ftp_path	-	FTP 路径。如果没有指定路径，会使用设置中说明的路径。

示例:

```
int lasterror=0;
if(!SendFTP("report.txt"))
  lasterror=GetLastError();
```

- 发送电子邮件 SendMail()

Void SendMail(string subject, string some_text)

发送电子邮件。邮件地址设置在“工具>选项>EMail” 标签内。

此项功能可以在客户端设置中禁用，或者省略指定的邮件地址。调用

GetLastError() 函数可以获得详细出错信息。

参数:

```
Subject     -   标题文本。
some_text   -   邮件主体内容。
```

示例:

```
  double lastclose=Close[0];
  if(lastclose<my_signal)
    SendMail("发自智能交易", "价格下降到
"+DoubleToStr(lastclose,Digits));
```

- 暂停程序运行 Sleep()

Void Sleep(int milliseconds)

Sleep()函数可以临时暂停运行当前智能交易程序一段时间。

由于 Sleep()函数运行在接口线程中，而且又不能减速，所以本函数不能在自定义指标内调用。

当 Sleep()函数运行时，智能交易每隔 0.1 秒会自动检测一次停止标志的状态。

参数:

```
Milliseconds   -   暂停毫秒数。
```

示例:

```
//---- 暂停 10 秒
   Sleep(10000);
```

数据类型转换函数

本组函数提供从一种格式到另一种格式的数据转换功能。

特别要注意 NormalizeDouble()函数，它确保了价格表示所需的准确性。在商业交易中，我们不可能使用非标准价格，其精度要超出交易服务器的要求，至少多出一位。

- ASCII 码转换成字符串 CharToStr()

String CharToStr(int char_code)

将 ASCII 码转换成字符。

参数:

Char_code	-	字符的 ACSII 码值。

示例:

```
string str="WORL" + CharToStr(44);  // 44 是'D'的 ASCII 码。
// 结果字符串就是“WORLD”。
```

- 浮点型数据转换成字符串 DoubleToStr()

```
string DoubleToStr(double value, int digits)
```

将双精度浮点型数值转换成指定精度的字符串。

参数:

value	-	浮点型数值。
digits	-	精度要求，小数点后位数(0-8)。

示例:

```
string value=DoubleToStr(1.28473418, 5);
// 值为"1.28473"
```

- 标准化双精度型数值 NormalizeDouble()

```
double NormalizeDouble(double value, int digits)
```

浮点型数值四舍五入到指定的精度，返回标准化双精度型数值。

计算止损值和赢利值，挂单开盘价也要求用指定精度标准化。这个精度要求需要在预定义的 digits 变量中设定。

参数:

value	-	浮点数值。
digits	-	精确要求，小数点后位数 (0-8)。

示例:

```
double var1=0.123456789;
Print(DoubleToStr(NormalizeDouble(var1,5),8));
//  输入信息: 0.12346000
```

- 字符串型数据转换成浮点型 StrToDouble()

Double StrToDouble(string value)

将数值形式的字符串转换成双精度型数值(带浮点的双精度格式)。

参数:

value - 数值形式的字符串。

示例:

```
double var=StrToDouble("103.2812");
```

- 字符串数据转换成整型 StrToInteger()

int StrToInteger(string value)

将含有数值形式的字符串转换成整型数值。

参数:

value - 数值形式的字符串。

示例:

```
int var1=StrToInteger("1024");
```

- 字符串转换成日期时间 StrToTime

datetime StrToTime(string value)

将 “yyyy.mm.dd hh:mm” 形式的字符串转换成日期时间(秒数从 1970 年 1 月 1 日 00:00 算起)。

参数:

value - "yyyy.mm.dd hh:mm"形式的字符串。

示例:

```
datetime var1;
var1=StrToTime("2003.8.12 17:35");
var1=StrToTime("17:35");        // 返回当前日期和指定时间
var1=StrToTime("2003.8.12");    // 返回"2003.8.12 00:00"
```

- 日期时间型数据转换成字符串 TimeToStr()

string TimeToStr(datetime value, void mode)

日期时间型数据(从 1970 年 1 月 1 日起经过的秒数)转换为 "yyyy.mm.dd

hh:mm"格式的字符串。

参数:

value	-	从 1970 年 1 月 1 日 00:00 所经过的秒数。
mode	-	数据输出形式可以是下列一个或者多个组合: TIME_DATE 结果格式为 "yyyy.mm.dd", TIME_MINUTES 结果格式为 "hh:mm", TIME_SECONDS 结果格式为 "hh:mm:ss"。

示例:

```
string var1=TimeToStr(TimeCurrent(),TIME_DATE|TIME_SECONDS);
```

自定义指标

本组函数用于设计自定义指标。

这些函数不能用在智能交易和脚本中。

- 指标缓冲区 IndicatorBuffers()

Void IndicatorBuffers(int count)

为指标缓冲区分配内存，用于自定义指标计算。缓冲区的个数不能超过 8 个或者是小于在自定义缓冲区属性中所给出的值。如果自定义指标要求额外的缓冲区用于统计，那么这个函数必须使用指定的总缓冲区数。

参数:

count	-	分配缓冲区的总数。缓冲区数应该在 indicator_buffers 常量值和 8 之间。

示例:

```
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- 指标参数
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- 指标缓冲区
double     ind_buffer1[];
```

```
double     ind_buffer2[];
double     ind_buffer3[];
//+------------------------------------------------------------------
---+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------
---+
int init()
  {
//---- 使用 2 个额外的缓冲区用于计数。
   IndicatorBuffers(3);
//---- 画线设置
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个指标缓冲区位置
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//----  数据窗口名称和指标子窗口标签名称

IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化结束
   return(0);
  }
```

▪指标计数 IndicatorCounted

int IndicatorCounted()

在自定义指标上次启动之后，函数返回未改变的柱数。曾计算过的柱数不必重新计算。大多数情况下，相同的指标值不需要重算。本函数用于优化计算。

注：最新的柱子不必考虑重算，在多数情况下，这个柱子有必要重算，然而，有时会遇到边界情况，也就是在新柱子的第一跳时从智能交易调用自定义指标。可能先前柱子的最后一跳还没有处理完（因为在这一跳进入时上一跳还没有处理完），因为这个原因，自定义指标将不会被调用和计算。为了避免指标计算出错，IndicatorCounted()函数将返回前一个柱数。

示例：

```
  int start()
```

```
    {
     int limit;
     int counted_bars=IndicatorCounted();
  //---- 检验可能出现错误
     if(counted_bars<0) return(-1);
  //---- 最后统计的柱数将被重算
     if(counted_bars>0) counted_bars--;
     limit=Bars-counted_bars;
  //---- 主循环
     for(int i=0;  i<limit;  i++)
       {
        //---- ma_shift 设为 0，因为 SetIndexShif 调用

ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i)
;

ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i)
;

ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i)
;
       }
  //---- 完成
     return(0);
    }
```

- 设置指标精度 IndicatorDigits

```
void IndicatorDigits(int digits)
```

设置指标精度(小数点后位数)使其值显示直观化。货币对价格精度采用默认值，指标会添加到图表上。

参数:

```
digits   -   精确要求，小数点后位数。
```

示例:

```
int init()
  {
//----2 个额外缓冲区用于计算。
```

```
   IndicatorBuffers(3);
//---- 画线参数设置
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(Digits+2);
//----一指标的 3 个已分配缓冲区
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- 数据窗口的简称和指标子窗口的"简称"

IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
   return(0);
  }
```

- 设置指标简称 IndicatorShortName()

Void IndicatorShortName(string name)

设置显示在数据窗口和子窗口中自定义指标的"简称"。

参数:

```
name   -   新简称。
```

示例:

```
int init()
  {
//----用于计算 2 个附加缓冲区
   IndicatorBuffers(3);
//---- 画线设定
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个指标缓冲区
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- 数据窗口和自定义子窗口标签的名称
```

```
IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
   return(0);
  }
```

- 设置一个箭头符号 SetIndexArrow()

void SetIndexArrow(int index, int code)

为 DRAW_ARROW 类型的指标线设置一个箭头符号。

箭头代码范围限于 33 到 255 之间，超过无效。

参数:

index	-	指标线。必须在 0 至 7 之间。
code	-	来自 Wingdings 字体或数组常量的符号代码。

示例:

```
int init()
  {
//---- 2 个指标缓冲区
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- 画线参数设置
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
    SetIndexStyle(1,DRAW_ARROW);
    SetIndexArrow(1,218);
//---- 显示在数据窗口
    SetIndexLabel(0,"Fractal Up");
    SetIndexLabel(1,"Fractal Down");
//---- 初始化完成
   return(0);
  }
```

- 绑定数组到缓冲区 SetIndexBuffer()

bool SetIndexBuffer(Int index, double array[])

绑定全局数组到自定义指标预定义的缓冲区。需要计算指标缓冲区的个数

由 IndicatorBuffers() 函数设定并且不能超过 8 个。如果成功，返回 TRUE，否则，将返回 FALSE。如果想获得更详细的信息，请调用 GetLastError()函数。

参数:

index	-	指标线，必须在 0 至 7 之间。
array[]	-	存储计算指标值的数组。

示例:

```
double ExtBufferSilver[];
int init()
  {
    SetIndexBuffer(0, ExtBufferSilver);  // 第 1 线缓冲
    // ...
  }
```

- 设置指标线起始位置 SetIndexDrawBegin

void SetIndexDrawBegin(int index, int begin)

设置从给出的指标线开始绘制的柱数(从数据开始)。指标线从左边绘制到右边，已给出的柱子左边的数组值不会显示在图表或数据窗口中。设置 0 作为默认值,所有数据将被绘出。

参数:

index	-	指标线，必须在 0 至 7 之间。
begin	-	第一个画出的柱子仓位数。

示例:

```
int init()
  {
//----用于计算的 2 个额外缓冲区
   IndicatorBuffers(3);
//----画线设定
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 绘制 3 个附加缓冲区
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
```

```
   SetIndexBuffer(2,ind_buffer3);
//---- 数据窗口和自定义子窗口标签的名称

IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- 初始化完成
   return(0);
  }
```

- 设置图表画线空值 SetIndexEmptyValue

Void SetIndexEmptyValue(int index, double value)

设置图表画线空值。默认值不绘出或不显示在数据窗口。默认值为 EMPTY_VALUE。

参数:

```
index   -   指标线，必须在 0 至 7 之间。
value   -   新“空值”。
```

示例:

```
int init()
  {
//----2 个附加缓冲区
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- 画线参数设置
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
    SetIndexStyle(1,DRAW_ARROW);
    SetIndexArrow(1,218);
//---- 0 值不显示
    SetIndexEmptyValue(0,0.0);
    SetIndexEmptyValue(1,0.0);
//---- 在 DataWindow 窗口不显示
    SetIndexLabel(0,"Fractal Up");
    SetIndexLabel(1,"Fractal Down");
//---- 初始化完成
   return(0);
  }
```

- 设置画线说明 SetIndexLabel()

void SetIndexLabel(int index, string text)

设置在数据窗口和快速提示中显示的画线说明。

参数:

index	-	指标线，必须在 0 至 7 之间。
Text	-	标签文本。 NULL 表示指标值在数据窗口中不显示。

示例:

```
//+------------------------------------------------------------------
---+
//| Ichimoku Kinko Hyo 初始化函数
|
//+------------------------------------------------------------------
---+
int init()
  {
//----
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,Tenkan_Buffer);
   SetIndexDrawBegin(0,Tenkan-1);
   SetIndexLabel(0,"Tenkan Sen");
//----
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1,Kijun_Buffer);
   SetIndexDrawBegin(1,Kijun-1);
   SetIndexLabel(1,"Kijun Sen");
//----
   a_begin=Kijun;  if(a_begin<Tenkan) a_begin=Tenkan;
   SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(2,SpanA_Buffer);
   SetIndexDrawBegin(2,Kijun+a_begin-1);
   SetIndexShift(2,Kijun);
//---- 在数据窗口向上 Kumo 线不显示
   SetIndexLabel(2,NULL);
   SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(5,SpanA2_Buffer);
   SetIndexDrawBegin(5,Kijun+a_begin-1);
```

```
   SetIndexShift(5,Kijun);
   SetIndexLabel(5,"Senkou Span A");
//----
   SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(3,SpanB_Buffer);
   SetIndexDrawBegin(3,Kijun+Senkou-1);
   SetIndexShift(3,Kijun);
//---- 在数据窗口向下 Kumo 线不显示
   SetIndexLabel(3,NULL);
//----
   SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(6,SpanB2_Buffer);
   SetIndexDrawBegin(6,Kijun+Senkou-1);
   SetIndexShift(6,Kijun);
   SetIndexLabel(6,"Senkou Span B");
//----
   SetIndexStyle(4,DRAW_LINE);
   SetIndexBuffer(4,Chinkou_Buffer);
   SetIndexShift(4,-Kijun);
   SetIndexLabel(4,"Chinkou Span");
//----
   return(0);
  }
```

- 设置画线偏离值 SetIndexShift()

Void SetIndexShift(int index, int shift)

设置画线偏离值。对于正值，画线将会移向右侧，否则，移向左侧，即当前柱子的计算值将相对地偏离当前柱子而绘线。

参数:

```
index   -   指标线，必须在 0 至 7 之间。
shift   -   柱子偏离值。
```

示例:

```
//+------------------------------------------------------------------
---+
//| Alligator 初始化函数                                             |
```

```
//+------------------------------------------------------------------
---+
int init()
  {
//----画线时线的偏离
   SetIndexShift(0,JawsShift);
   SetIndexShift(1,TeethShift);
   SetIndexShift(2,LipsShift);
//---- 画线时跳过的第一个仓位
   SetIndexDrawBegin(0,JawsShift+JawsPeriod);
   SetIndexDrawBegin(1,TeethShift+TeethPeriod);
   SetIndexDrawBegin(2,LipsShift+LipsPeriod);
//---- 绘制 3 个附加缓冲区
   SetIndexBuffer(0,ExtBlueBuffer);
   SetIndexBuffer(1,ExtRedBuffer);
   SetIndexBuffer(2,ExtLimeBuffer);
//---- 画线设定
   SetIndexStyle(0,DRAW_LINE);
   SetIndexStyle(1,DRAW_LINE);
   SetIndexStyle(2,DRAW_LINE);
//---- 指标线标签
   SetIndexLabel(0,"Gator Jaws");
   SetIndexLabel(1,"Gator Teeth");
   SetIndexLabel(2,"Gator Lips");
//---- 初始化完成
   return(0);
  }
```

- 设置指标线样式 SetIndexStyle()

```
Void SetIndexStyle(int index, int type, void style, void width,
                   void clr)
```

为指定的指标线设置新类型、样式、宽度和颜色。

参数:

index	-	线编号，必须在 0 至 7 之间。
type	-	形状样式，可以是划线形状样式列表中任意一个。
style	-	线型。可以应用一个像素的粗线，可以是划线形状样式列表其中一个。 EMPTY 值表示线型不变。

width	-	线宽。有效值是 1,2,3,4,5。EMPTY 值表示线宽不变。
clr	-	线的颜色。省略本参数表示颜色将保持不变。

示例:

```
SetIndexStyle(3, DRAW_LINE, EMPTY, 2, Red);
```

- 设置水平线样式 SetLevelStyle()

Void SetLevelStyle(int draw_style, int line_width, void clr)

本函数设置输出到独立窗口的指标水平线的新样式、宽度和颜色。

参数:

draw_style	-	画线样式。可以是划线的形状样式列表其中一个。 EMPTY 值表示样式不变。
line_width	-	线宽。有效值可以是 1,2,3,4,5。EMPTY 值表示样式不变。
clr	-	线的颜色。空值 CLR_NONE 表示颜色不变。

示例:

```
//---- 显示红色水平实线
   SetLevelStyle(STYLE_SOLID,2,Red)
```

- 设置指标水平线值 SetLevelValue()

Void SetLevelValue(int level, double value)

本函数设置在独立窗口输出的水平指标线的值。

参数:

level	-	水平线编号(0-31)。
value	-	给出的指标水平线值。

示例:

```
SetLevelValue(1,3.14);
```

日期时间处理函数

本组函数用于处理日期时间型数据 (从 1970 年 1 月 1 日 00:00 开始已经过的秒数)。

- 获取今日是本月第几天 Day()

Int Day()

返回本月的当天，即最新的服务器时间的本月当天。

注：在测试中，仿真最新的已知的服务器时间。

示例：

```
if(Day()<5) return(0);
```

- 获取今日是星期几 DayOfWeek()

int DayOfWeek()

返回这周的星期几，(0-星期天,1,2,3,4,5,6 以此类推)，时间是来自最后已知的服务器上的时间。

注：在测试中，仿真最后已知的服务器时间。

示例：

```
//  假期不工作
if(DayOfWeek()==0 || DayOfWeek()==6) return(0);
```

- 获取今日是本年第几天 DayOfYear()

int DayOfYear()

返回今日是本年第几天(1 代表 1 月 1 日.., 365(6)就是 12 月 31 日)，即最后已知的服务器时间中本年第几天。

注：在测试中，仿真最后已知的服务器时间。

示例：

```
if(DayOfYear()==245)
  return(true);
```

- 获取当前小时数 Hour()

int Hour()

在程序开始前，返回服务器时间中小时数(0,1,2,..23) (在程序执行期间，这个值不会改变)。

注：在测试中，仿真最后已知的服务器时间。

示例：

```
bool is_siesta=false;
if(Hour()>=12 || Hour()<17)
   is_siesta=true;
```

- 获取当前分钟数 Minute()

int Minute()

返回服务器时间中分钟数(0,1,2,..59)。

示例:

```
if(Minute()<=15)
  return("first quarter");
```

- 获取当前的月份 Month()

int Month()

返回服务器时间中月数(1,2,..12)。

注: 在测试中, 仿真最后已知的服务器时间。

示例:

```
if(Month()<=5)
  return("the first half year");
```

- 获取当前的秒数 Seconds()

int Seconds()

返回服务器时间中秒数。

示例:

```
if(Seconds()<=15)
  return(0);
```

- 获取服务器时间 TimeCurrent()

datetime TimeCurrent()

返回最后访问的服务器时间(最新的行情输入时间)，从 1970 年 1 月 1 日 00:00 算起。

注: 在测试中, 仿真最后已知的服务器时间。

示例:

```
if(TimeCurrent()-OrderOpenTime()<360) return(0);
```

- 获取指定日期中天数 TimeDay()

int TimeDay(Datetime date)

返回指定日期中天数（1-31）。

参数:

```
date   -   日期时间，从 1970 年 1 月 1 日 00:00 算起。
```

示例:

```
int day=TimeDay(D'2003.12.31');
// 天数为 31
```

- 获取指定日期是星期几 TimeDayOfWeek()

int TimeDayOfWeek(datetime date)

返回指定日期是星期几(0 代表星期天, 1, 2, 3, 4, 5, 6 类推)。

参数:

```
date   -   日期时间，从 1970 年 1 月 1 日 00:00 算起。
```

示例:

```
int weekday=TimeDayOfWeek(D'2004.11.2');
// 天数是 2 - 星期二
```

- 获取指定日期是一年中第几天 TimeDayOfYear()

int TimeDayOfYear(datetime date)

返回指定日期是一年中第几天(1 意味 1 月 1 日.., 365(6) 表示 12 月 31 日)。

参数:

```
date   -   日期时间，从 1970 年 1 月 1 日 00:00 算起。
```

示例:

```
int day=TimeDayOfYear(TimeCurrent());
```

- 获取指定时间中小时数 TimeHour()

int TimeHour(datetime time)

返回指定时间中小时数。

参数:

time - 日期时间，从 1970 年 1 月 1 日 00:00 算起。

示例:

```
int h=TimeHour(TimeCurrent());
```

- 获取当前电脑时间 TimeLocal()

datetime TimeLocal()

返回本地计算机的当前时间，从 1970 年 1 月 1 日 00:00 算起。

注: 在测试中, 仿真本地时间和最后已知的服务器时间。

示例:

```
if(TimeLocal()-OrderOpenTime()<360) return(0);
```

- 获取指定时间中分钟数 TimeMinute()

int TimeMinute(datetime time)

返回指定的时间中分钟。

参数:

Time - 日期时间，从 1970 年 1 月 1 日 00:00 算起。

示例:

```
int m=TimeMinute(TimeCurrent());
```

- 获取指定时间中月份 TimeMonth()

int TimeMonth(datetime time)

返回指定时间中月份。

参数:

time - 日期时间，从 1970 年 1 月 1 日 00:00 算起。

示例:

```
int m=TimeMonth(TimeCurrent());
```

- 获取指定时间中秒数 TimeSeconds()

int TimeSeconds(datetime time)

返回指定时间中秒数。

参数:

time - 日期时间，从 1970 年 1 月 1 日 00:00 算起。

示例:

```
int m=TimeSeconds(TimeCurrent());
```

- 获取指定时间中年份 TimeYear

int TimeYear(datetime time)

返回指定时间中年份，返回值的范围可以在 1970 到 2037 之间。

参数:

time - 日期时间，从 1970 年 1 月 1 日 00:00 算起。

示例:

```
int y=TimeYear(TimeCurrent());
```

- 获取当前年份 Year()

int Year()

返回当前的年份，即服务器时间中年份。

注: 在测试中, 仿真最后已知的服务器时间。

示例:

```
// 如果时间范围在 2006 年 1 月 1 日到 4 月 30 日之间，返回。
  if(Year()==2006 && Month()<5)
  return(0);
```

文件操作函数

本组函数用于处理文件。

存放工作文件的三个目录(含子目录):

- /HISTORY/<当前经纪商> - 特别提供的 FileOpenHistory 函数;
- /EXPERTS/FILES - 公用目录;
- /TESTER/FILES - 测试专用。

禁止访问来自于其他目录的工作文件。

- 关闭文件 FileClose()

```
void FileClose(int handle)
```

关闭以前用 FileOpen() 函数打开的文件。

参数:

handle	-	FileOpen()函数返回的文件句柄。

示例:

```
int handle=FileOpen("filename", FILE_CSV|FILE_READ);
if(handle>0)
  {
   // 运行文件 ...
   FileClose(handle);
  }
```

- 删除文件 FileDelete()

```
void FileDelete(string filename)
```

删除指定的文件。

如果想获取详细的错误信息, 请调用 GetLastError()函数。

如果文件是在 terminal_dir\experts\files 目录 (在测试情况下, terminal_directory\tester\files)或它的子目录, 仅仅删除这个文件。

参数:

filename	-	目录和文件名。

示例:

```
//文件 my_table.csv 将从目录 terminal_dir\experts\files 目录删除
int lastError;
FileDelete("my_table.csv");
```

```
lastError=GetLastError();
if(laseError!=ERR_NOERROR)
  {
   Print("错误 (",lastError,") 删除文件 my_table.csv");
   return(0);
  }
```

- 清除文件缓冲区并存盘 FileFlush()

void FileFlush(int handle)

清除缓存中数据并存到磁盘上。

注：FileFlush() 函数只能在文件读写操作中调用。

文件关闭时，数据将自动保存到磁盘，因此，在调用 FileClose() 函数之前没有必要调用 FileFlush() 函数。

参数:

handle - 文件句柄，FileOpen()函数返回的句柄。

示例:

```
int bars_count=Bars;
int handle=FileOpen("mydat.csv",FILE_CSV|FILE_WRITE);
if(handle>0)
  {
   FileWrite(handle, "#","OPEN","CLOSE","HIGH","LOW");
   for(int i=0; i<bars_count; i++)
     FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileFlush(handle);
   ...
   for(int i=0; i<bars_count; i++)
     FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileClose(handle);
  }
```

- 判断文件指针是否到文件尾 FileIsEnding()

bool FileIsEnding(int handle)

如果文件指针是在文件尾,返回 true，否则，返回 false。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

如果文件指针在只读期间到达文件尾， GetLastError() 函数将返回错误 ERR_END_OF_FILE (4099)。

参数:

handle - 文件句柄, FileOpen()函数返回值。

示例:

```
if(FileIsEnding(h1))
  {
   FileClose(h1);
   return(false);
  }
```

- 判断文件指针是否指向行尾 FileIsLineEnding()

bool FileIsLineEnding(int handle)

如果 CSV 文件指针指向行末,返回 true，否则，返回 false。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参数:

handle - 文件句柄，FileOpen()函数返回的句柄。

示例:

```
if(FileIsLineEnding(h1))
  {
   FileClose(h1);
   return(false);
  }
```

- 打开文件 FileOpen()

int FileOpen(string filename, int mode, void delimiter)

为输入或输出信息打开文件。如果打开文件成功，返回句柄，否则，返回 -1。如果想获取详细的错误信息, 请调用 GetLastError() 函数。

注: 文件只能在 terminal_directory\experts\files 文件夹(智能交易测试在 terminal_directory\tester\files 目录) 或子目录内被打开。

FILE_BIN 和 FILE_CSV 模式不能同时使用。

如果 FILE_WRITE 与 FILE_READ 不组合应用, 长度为零的文件可以打开。

如果文件包含数据，它们能被删除。如果有必要向现存文件中添加数据，必须使用 FILE_READ 和 FILE_WRITE 组合打开文件。

如果 FILE_READ 与 FILE_WRITE 不组合应用，只能打开现存文件。如果文件不存在，可以使用 FILE_WRITE 模式创建。

在一个模块内最多能够同时打开 32 个文件。在同一个模块内文件打开的句柄不能传递给其它模块(库)。

参数:

filename	-	文件名称
mode	-	打开模式。可以是以下的一种或是多种组合: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE。
delimiter	-	csv 文件内容限定符。默认值为分号';' 。

示例:

```
int handle;
handle=FileOpen("my_data.csv",FILE_CSV|FILE_READ,'; ');
if(handle<1)
  {
   Print("未找到 my_data.dat 文件，错误", GetLastError());
   return(false);
  }
```

- 打开历史目录下文件 FileOpenHistory()

```
int FileOpenHistory(string filename, int mode, void delimiter)
```

在当前历史目录(terminal_directory\history\server_name)或子目录内打开文件。如果打开成功，返回文件句柄，如果函数失败，返回-1。如果想获取详细的错误信息，请调用 GetLastError()函数。

注: 客户端有可能连接到不同经纪公司的服务器。每个经纪公司的历史数据(HST 文件)会存储在 terminal_directory\history 相对应的子目录内。

文件在离线时同样可以打开，不必输入新数据就能把它制成图表。

参数:

filename	-	文件名称
mode	-	打开模式。可以是以下的一种或是组合: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE。
delimiter	-	csv 文件内容限定符。默认值为';'符号。

示例:

```
int handle=FileOpenHistory("USDX240.HST",FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("不能创建 USDX240.HST 文件");
   return(false);
  }
// 运行文件
// ...
FileClose(handle);
```

- 文件读取 FileReadArray()

int FileReadArray(int handle, void array[], int start, int count)

从二进制文件读取指定数量的内容到数组中。读取之前，确认数组要足够大。函数返回实际读取内容的数量。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
array[]	-	保存数据的数组。
start	-	数组中存储的开始位置。
count	-	读取内容个数。

示例:

```
int handle;
double varray[10];
handle=FileOpen("filename.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   FileReadArray(handle, varray, 0, 10);
   FileClose(handle);
  }
```

- 读取文件双精度数据 FileReadDouble()

double FileReadDouble(Int handle, void size)

从二进制文件当前位置读取浮点型双精度数据。数值格式长度可以是 8 byte 的 double 型或者是 4 byte 的 float 型。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
size	-	数值格式大小，DOUBLE_VALUE(8 bytes) 或者 FLOAT_VALUE(4 bytes)。

示例:

```
int handle;
double value;
handle=FileOpen("mydata.dat",FILE_BIN);
if(handle>0)
  {
   value=FileReadDouble(handle,DOUBLE_VALUE);
   FileClose(handle);
  }
```

- 读取文件中整数 FileReadInteger()

int FileReadInteger(int handle, void size)

本函数从二进制文件中当前位置读取整型数据，数值可以是 1,2,4 字节的长度。如果格式大小不指定，系统设法读 4 字节的值。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
size	-	数值格式大小，CHAR_VALUE(1 字节)，SHORT_VALUE(2 字节)或者 LONG_VALUE(4 字节)。

示例:

```
int handle;
int value;
handle=FileOpen("mydata.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   value=FileReadInteger(h1,2);
   FileClose(handle);
  }
```

- 读取文本文件中数值 FileReadNumber()

double FileReadNumber(int handle)

在文件当前位置读取分隔符之前数值。只能为 CSV 文件。

如果想获取详细的错误信息，请调用 GetLastError()函数。

参数:

handle - 用 FileOpen()函数返回的句柄。

示例:

```
int handle;
int value;
handle=FileOpen("filename.csv", FILE_CSV, '; ');
if(handle>0)
  {
   value=FileReadNumber(handle);
   FileClose(handle);
  }
```

- 读取文件中字符串 FileReadString()

string FileReadString(int handle, void length)

本函数从文件当前位置读取字符串，适用于 CSV 和二进制文件。对于文本文件，读取在分隔符之前字符串。对于二进制文件，将读取指定个数的字符串。

如果想获取详细的错误信息，请调用 GetLastError()函数。

参数:

handle - 用 FileOpen()返回的句柄。
length - 要读取的字符串长度 。

示例:

```
int handle;
string str;
handle=FileOpen("filename.csv", FILE_CSV|FILE_READ);
if(handle>0)
  {
   str=FileReadString(handle);
```

```
    FileClose(handle);
   }
```

- 移动文件指针 FileSeek()

bool FileSeek(int handle, int offset, int origin)

本函数可以从文件头、文件尾或当前位置按字节移动文件指针到一个偏移的新位置。接下来读写操作就在一个新位置开始。

如果文件指针移动成功了，本函数返回 TRUE， 否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
offset	-	按字节数相对原点的偏移量。
origin	-	初始位置。 其值可以是以下任意常数: SEEK_CUR - 当前位置， SEEK_SET - 文件头， SEEK_END - 文件尾。

示例:

```
  int handle=FileOpen("filename.csv",
FILE_CSV|FILE_READ|FILE_WRITE, '; ');
  if(handle>0)
    {
     FileSeek(handle, 0, SEEK_END);
     //----在文件尾添加数据
     FileWrite(handle, data1, data2);
     FileClose(handle);
     handle=0;
    }
```

- 获取文件大小 FileSize()

int FileSize(int handle)

本函数返回文件大小(字节数)。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

```
handle   -   用 FileOpen()函数返回的句柄。
```

示例:

```
int handle;
int size;
handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   size=FileSize(handle);
   Print("my_table.dat 大小为 ", 大小 " bytes");
   FileClose(handle);
  }
```

- 获取文件指针位置 FileTell()

```
int FileTell(int handle)
```

返回文件指针的当前位置。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

```
handle   -   用 FileOpen()函数返回的句柄。
```

示例:

```
int handle;
int pos;
handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
// 读取数据
pos=FileTell(handle);
Print("current position is ", pos);
```

- 写入 CSV 文件 FileWrite()

```
int FileWrite(int handle, ...)
```

本函数用于向 CSV 文件写入数据，自动插入数据分隔符。在写入文件后，每行的尾端将会添加"\r\n" 回车换行符号。数值将会被转变成文本(参看 Print() 函数)。

本函数返回写入的字符个数，如果写入出错，返回一个负数。

如果想获取详细的错误信息，查看 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
...	-	用户写入的数据用逗号分隔，可以多达 63 个参数。int 型和 double 型数据自动地被转换成字符串，但不自动转换颜色型、日期时间型和布尔型数据，他们被当作数值写入文件。数组不能作为参数传递。

示例:

```
  int handle;
  datetime orderOpen=OrderOpenTime();
  handle=FileOpen("filename", FILE_CSV|FILE_WRITE, '; ');
  if(handle>0)
    {
     FileWrite(handle, Close[0], Open[0], High[0], Low[0],
TimeToStr(orderOpen));
     FileClose(handle);
    }
```

- 数组内容写入文件 FileWriteArray()

```
int FileWriteArray(int handle, object array[], int start, int count)
```

本函数将数组内容写入一个二进制文件。整型、布尔型、日期时间型和颜色型数组元素作为 4 字节整数写入文件；双精度型数组元素作为 8 字节浮点数写入文件；字符串型数组直接写入文件，并在每串末尾自动添加"\r\n"回车换行符。

本函数返回写入内容的个数，如果写入出错，返回负值。

如果想获取详细的错误信息，请调用 GetLastError()函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
array[]	-	要写入文件的数组。
start	-	数组起始位置(写入文件的第一个数组元素下标)。
count	-	要写入文件的数组元素个数。

示例:

```
  int handle;
  double BarOpenValues[10];
  //  复制前十个柱到数组
```

```
  for(int i=0; i<10;  i++)
    BarOpenValues[i]=Open[i];
  //数组写入到文件
  handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
  if(handle>0)
    {
     FileWriteArray(handle, BarOpenValues, 3, 7);  // 写入最后 7 个
元素
     FileClose(handle);
    }
```

- 双精度数值写入文件 FileWriteDouble()

```
int FileWriteDouble(int handle, double value, void size)
```

本函数将一个带浮点的双精度数写入二进制文件。如果格式被指定为 FLOAT_VALUE，值将作为 4 字节浮点数写入(浮点型)，否则，它以 8 字节浮点数格式写入(双精度型)。

返回实际写入的字节数，如果写入出错，返回负值。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
value	-	双精度值。
size	-	可选格式。可以是以下的任意值: DOUBLE_VALUE (8 字节，默认值) FLOAT_VALUE (4 字节)。

示例:

```
  int handle;
  double var1=0.345;
  handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
  if(handle<1)
    {
     Print("不能打开错误文件-",GetLastError());
     return(0);
    }
  FileWriteDouble(h1, var1, DOUBLE_VALUE);
  //...
```

```
FileClose(handle);
```

- 整数写入文件 FileWriteInteger()

```
int FileWriteInteger(int handle, int value, void size)
```

本函数将整数写入一个二进制文件。如果大小是 SHORT_VALUE，其值将作为 2 字节整数(短整型)写入；如果大小是 CHAR_VALUE，其值将作为 1 字节整数(字符型)写入；如果大小是 LONG_VALUE，其值将作为 4 字节整数(长整型)写入。

本函数返回实际写入的字节数，如果写入出错，返回负值。

如果想获取详细的错误信息，查看 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
value	-	写入值
size	-	可选格式。可以是以下任意值: CHAR_VALUE (1 字节), SHORT_VALUE (2 字节), LONG_VALUE (4 字节，默认值)。

示例:

```
int handle;
int value=10;
handle=FileOpen("filename.dat", FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("不能打开错误文件-",GetLastError());
   return(0);
  }
FileWriteInteger(handle, value, SHORT_VALUE);
//...
FileClose(handle);
```

- 字符串写入文件 FileWriteString()

```
int FileWriteString(int handle, string value, int length)
```

本函数将字符串写入二进制文件的当前位置。

本函数返回实际写入的字节数，如果写入出错，返回负值。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

handle	-	用 FileOpen()函数返回的句柄。
value	-	写入的字符串。
length	-	写入的字符串长度。如果字符串长度超出给定的值，多余部分将被截掉。 如果它较短，不足部分将用二进制 0 填充到给定的长度。

示例:

```
int handle;
string str="some string";
handle=FileOpen("filename.bin", FILE_BIN|FILE_WRITE);
  if(handle<1)
  {
   Print("不能打开错误文件-",GetLastError());
   return(0);
  }
FileWriteString(h1, str, 8);
FileClose(handle);
```

全局变量

和全局变量一起使用的一组函数。

客户端全局变量不要与 MQL4 程序中定义的全局变量混淆。

最后访问的全局变量可以在客户端内保存 4 个星期，然后将自动删除。对于全局变量的访问不仅仅是新值的设定，也可以对全局变量进行读取。

在客户端启动的所有 MQL4 程序可以同时访问客户端的全局变量。

- 检查全局变量是否存在 GlobalVariableCheck()

bool GlobalVariableCheck(string name)

如果全局变量存在，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，查看 GetLastError() 函数。

参数:

name	-	全局变量名称。

示例:

```
// 使用前检查变量
if(!GlobalVariableCheck("g1"))
  GlobalVariableSet("g1",1);
```

- 删除全局变量 GlobalVariableDel()

bool GlobalVariableDel(string name)

删除全局变量。如果函数成功，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

```
name   -   全局变量名称。
```

示例:

```
// 删除名称为  "gvar_1"的全局变量("gvar_1");
```

- 获取全局变量值 GlobalVariableGet()

Double GlobalVariableGet(string name)

返回现有的全局变量值，如果发生错误，返回 0。

如果想获取详细的错误信息, 请调用 GetLastError()函数。

参数:

```
name   -   全局变量名称。
```

示例:

```
double v1=GlobalVariableGet("g1");
//---- 检查函数调用结果
if(GetLastError()!=0) return(false);
//---- 继续处理
```

- 获取指定索引的全局变量名 GlobalVariableName()

String GlobalVariableName(int index)

本函数通过全局变量列表中索引号返回全局变量的名称。

如果想获取详细的错误信息, 请调用 GetLastError()函数。

参数:

index	-	全局变量列表中索引。 索引必须等于 0 且小于 GlobalVariablesTotal()值。

示例:

```
int    var_total=GlobalVariablesTotal();
string name;
for(int i=0; i<var_total; i++)
  {
   name=GlobalVariableName(i);
   Print(i,": 全局变量名称 - ",name);
  }
```

- 设置全局变量值 GlobalVariableSet()

```
datetime GlobalVariableSet(string name, double value)
```

设置全局变量新的值。如果它不存在，系统将创建一个新的全局变量。如果函数成功，其返回值将是最后存取时间，否则，返回值将是 0。

如果想获取详细的错误信息,请调用 GetLastError() 函数。

参数:

name	-	全局变量名称。
value	-	新的数值。

示例:

```
//---- 尝试设定新值
if(GlobalVariableSet("BarsTotal",Bars)==0)
  return(false);
//---- 继续处理
```

- 根据条件设置全局变量值 GlobalVariableSetOnCondition()

```
Bool                              string name, double value,
GlobalVariableSetOnCondition(     double check_value)
```

如果全局变量的当前值等于第三个参数 check_value，就设置现有的全局变量的新值。如果全局变量不存在，函数将产生错误 ERR_GLOBAL_VARIABLE_NOT_FOUND (4058) 并且返回 FALSE。当函数执行成功时，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

如果全局变量的当前值与 check_value 不同，函数将返回 FALSE。

函数为全局变量提供了强大的访问能力，这就是为什么在一个客户端内同时运行的几个智能交易程序用它作为通信信号相互通信的原因。

参数:

name	-	全局变量名称。
value	-	新值。
check_value	-	与当前全局变量值相比较的值。

示例:

```
  int init()
    {
     //---- 创建全局变量
     GlobalVariableSet("DATAFILE_SEM",0);
     //...
    }

  int start()
    {
     //---- 尝试锁住公共资源
     while(!IsStopped())
       {
        //---- 锁住
        if(GlobalVariableSetOnCondition("DATAFILE_SEM",1,0)==true)
break;
        //---- 可以删除变量吗?
        if(GetLastError()==ERR_GLOBAL_VARIABLE_NOT_FOUND)
return(0);
        //---- 等待
        Sleep(500);
       }
     //---- 资源被锁
     // ...做些工作
     //----解锁资源
     GlobalVariableSet("DATAFILE_SEM",0);
    }
```

- 删除全局变量 GlobalVariablesDeleteAll()

int GlobalVariablesDeleteAll(void prefix_name)

删除全局变量。如果全局变量名称的前缀没有指定，所有全局变量将被删除，否则，只有那些符合指定前缀开头的变量被删除。函数返回被删除的变量个数。

参数:

prefix_name - 要被删除的全局变量名称前缀。

示例:

```
Print(GlobalVariablesDeleteAll("test_")," test 开头的全局变量被删除");
```

- 获取全局变量总数 GlobalVariablesTotal()

int GlobalVariablesTotal()

函数返回全局变量总数。

示例:

```
Print(GlobalVariablesTotal(),"全局变量被查到");
```

数学和三角函数

一组数学和三角函数。

- 求绝对值 MathAbs()

double MathAbs(double value)

返回指定数值的绝对值。

参数:

value - 数值。

示例:

```
double dx=-3.141593, dy;
// 计算 MathAbs
dy=MathAbs(dx);
Print("The absolute value of ",dx," is ",dy);
// 输出: -3.141593 的绝对值为 3.141593
```

- 求反余弦 MathArccos()

double MathArccos(double x)

MathArccos 函数返回 x 在 0-π (用弧度) 范围内的反余弦。 如果 x 小于 -1 或超出 1， MathArccos 返回 NaN(值不确定)。

参数:

X - 计算值在-1 到 1 之间反余弦。

示例:

```
double x=0.32696, y;
y=matharcsin(x);
Print("反正弦",x," = ",y);
y=matharccos(x);
Print("反余弦 ",x," = ",y);
//输出: 0.326960 反正弦=0.333085
//输出: 0.326960 反余弦 =1.237711
```

- 求反正弦 MathArcsin()

double MathArcsin(double x)

返回 x 在 -π/2 到 π/2 范围内反正弦。如果 x 小于-1 或超出 1，返回 NaN(值不确定)。

参数:

X - 计算反正弦的值。

示例:

```
double x=0.32696, y;
y=MathArcsin(x);
Print("反正弦",x," = ",y);
y=matharccos(x);
Print("反余弦 ",x," = ",y);
//输出: 0.326960 反正弦=0.333085
//输出: 0.326960 反余弦=1.237711
```

- 求反正切 MathArctan()

double MathArctan(double x)

本函数返回 x 的反正切值。如果 x 为 0，返回 0。返回值必须在-π/2 到 π/2。

参数:

X	-	求反正切的数值。

示例:

```
double x=-862.42, y;
y=MathArctan(x);
Print("反正切 ",x," 是 ",y);
//输出: -862.42 反正切是 -1.5696
```

- 取最小整数 MathCeil()

double MathCeil(double x)

MathCeil 函数返回一个大于或等于 x 的最小整数。

参数:

X	-	数值。

示例:

```
double y;
y=MathCeil(2.8);
Print("上限 2.8 is ",y);
y=MathCeil(-2.8);
Print("上限 -2.8 is ",y);
/*输出:
  2.8 的最小整数为 3
   -2.8 的最小整数为-2  */
```

- 求余弦 MathCos()

double MathCos(double value)

返回指定角度的余弦。

参数:

value	-	用弧度表示的角度值。

示例:

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("正弦(",x,") = ",y);
y=MathCos(x);
Print("余弦(",x,") = ",y);
//输出: 正弦(1.5708)=1
//        余弦(1.5708)=0
```

- 求 e 的幂 MathExp()

double MathExp(double d)

返回 e 的 d 次幂。在溢出情况下，函数返回 INF (无穷大)，下溢时返回 0。

参数:

```
D   -   指定乘方的数字。
```

示例:

```
double x=2.302585093,y;
y=MathExp(x);
Print("MathExp(",x,") = ",y);
//输出: MathExp(2.3026)=10
```

- 取整数 MathFloor()

double MathFloor(double x)

MathFloor 函数返回一个小于或等于 x 的最大整数。

参数:

```
X   -   数值。
```

示例:

```
double y;
y=MathFloor(2.8);
Print("2.8 最大整数是 ",y);
y=MathFloor(-2.8);
Print(" -2.8 最大整数是 ",y);
/*输出:
```

```
    2.8 最大整数为 2
    -2.8 最大整数为-3 */
```

- 求自然对数 MathLog()

double MathLog(double x)

如果成功，MathLog 函数返回 x 的自然对数。如果 x 是负值，这些函数返回 NaN (值不确定)。如果 x 是 0，他们返回 INF (无穷大)。

参数:

```
X   -   求自然对数的值。
```

示例:

```
double x=9000.0,y;
y=MathLog(x);
Print("MathLog(",x,") = ", y);
//输出: MathLog(9000)=9.10498
```

- 求最大值 MathMax()

double MathMax(double value1, double value2)

返回两个数值中最大值。

参数:

```
value1   -   第一个数值。
value2   -   第二个数值。
```

示例:

```
double result=MathMax(1.08,Bid);
```

- 求最小值 MathMin()

double MathMin(double value1, double value2)

返回两个数值中最小值。

参数:

```
value1   -   第一个数值。
value2   -   第二个数值。
```

示例:

```
double result=MathMin(1.08,Ask);
```

- 求模 MathMod()

double MathMod(double value, double value2)

此函数返回两个数相除的浮点型余数。

MathMod 函数计算 x / y 的浮点余数 f ,因此 x = i * y + f ,其中 i 是整数， f 与 x 有相同的正负号，且 f 的绝对值要小于 y 的绝对值。

参数:

```
value    -   被除数。
value2   -   除数。
```

示例:

```
double x=-10.0,y=3.0,z;
z=MathMod(x,y);
Print("余数 ",x," / ",y," 为 ",z);
//输出: -10 / 3 的余数为 -1
```

- 求幂 MathPow()

double MathPow(double base, double exponent)

返回基数指定次方的值。

参数:

```
Base       -   基数。
exponent   -   指数值。
```

示例:

```
double x=2.0,y=3.0,z;
z=MathPow(x,y);
Printf(x," 的",y,"次乘方为", z);
//输出: 2 的 3 次乘方为 8
```

- 获取随机整数 MathRand()

int MathRand()

MathRand 函数返回一个在 0 到 32767 范围内的伪随机整数。在调用 MathRand 之前，需要使用生成伪随机数的种子。

示例:

```
MathSrand(TimeLocal());
// 显示 10 个数字。
for(int i=0; i<10; i++ )
  Print("随机值  ", MathRand());
```

- 求四舍五入值 MathRound()

double MathRound(double value)

返回四舍五入到最接近指定数值的整数。

参数:

```
value   -   要四舍五入的值。
```

示例:

```
double y=MathRound(2.8);
Print("2.8 的四舍五入值为 ",y);
y=MathRound(2.4);
Print(" -2.4 的四舍五入值为 ",y);
//输出:  2.8 的四舍五入值为 3
//          -2.4 的四舍五入值为 -2
```

- 求正弦 MathSin()

double MathSin(double value)

返回指定角的正弦。

参数:

```
value   -   弧度表示的角度值。
```

示例:

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("MathSin(",x,") = ",y);
```

```
y=MathCos(x);
Print("MathCos(",x,") = ",y);
//输出: MathSin(1.5708)=1
//        MathCos(1.5708)=0
```

- 求平方根 MathSqrt()

double MathSqrt(double x)

MathSqrt 函数返回 x 的平方根。如果 x 为负值,返回无穷大(与 NaN 相同)。

参数:

```
X   -   非负数值。
```

示例:

```
double question=45.35, answer;
answer=MathSqrt(question);
if(question<0)
  Print("错误: MathSqrt 返回",answer," 答案");
else
  Print("",question,"的平方根为 ", answer);
//输出:45.35 的平方根为 6.73
```

- 获取随机数 MathSrand()

Void MathSrand(Int seed)

MathSrand() 函数为生成一组伪随机整数设置一个起点。为了重新初始化随机数生成器，使用 1 作为种子。用其它数值作为种子可以将生成器设定到一个随机起点。 MathRand 取回已生成的伪随机数。在调用 MathSrand()之前，调用 MathRand 函数和用 1 作为种子调用 MathSrand 函数都会生成相同的序列数。

参数:

```
seed   -   生成随机数的种子。
```

示例:

```
MathSrand(TimeLocal());
// 显示 10 个随机数。
for(int i=0; i<10; i++ )
  Print("随机数 ", MathRand());
```

- 求正切 MathTan()

double MathTan(Double x)

MathTan 返回 x 的正切值。

参数:

X	-	弧度角

示例:

```
double pi=3.1415926535;
double x,y
x=MathTan(pi/4);
Print("MathTan(",pi/4," = ",x);
//输出:MathTan(0.7856)=1
```

对象操作函数

- 创建对象 ObjectCreate()

```
Bool          string name, int type, int window, datetime time1,
ObjectCreate( double price1, void time2, void price2, void time3,
              void price3)
```

在指定的窗口中用指定的名称、类型和最初的坐标创建对象。根据对象类型与对象有关的坐标个数可以是可以是 1 到 3 个。 如果函数成功，返回 TRUE，否则，返回 FALSE。要想获得详细的错误信息, 请调用 GetLastError()函数。OBJ_LABEL 类型的对象忽略坐标。使用 ObjectSet() 设定 OBJPROP_XDISTANCE 和 OBJPROP_YDISTANCE 属性。

注: 图表子窗口 (如果子窗口图表中带有指标)编号从 1 开始。图表主窗口总是存在的，且索引为零。

坐标必须成对传递：时间和价格。例如，OBJ_VLINE 对象只需要时间，但价格(任意值)也必须同时传递。

参数:

name	-	对象唯一的名称。
type	-	对象类型，它可以是对象类型枚举中任意值。
window	-	要添加对象的窗口索引。窗口索引必须大于等于 0 并且小于 WindowsTotal()值。

```
time1    -   第一点时间。
price1   -   第一点价格值。
time2    -   第二点时间。
price2   -   第二点价格值。
time3    -   第三点时间。
price3   -   第三点价格值。
```

示例:

```
  // 新文本对象
  if(!ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30',
1.0045))
    {
     Print("错误:不能创建文本! 代码 #",GetLastError());
     return(0);
    }
  // 新标签对象
  if(!ObjectCreate("label_object", OBJ_LABEL, 0, 0, 0))
    {
     Print("错误:不能创建 label_object! 代码 #",GetLastError());
     return(0);
    }
  ObjectSet("label_object", OBJPROP_XDISTANCE, 200);
  ObjectSet("label_object", OBJPROP_YDISTANCE, 100);
```

- 删除对象 ObjectDelete()

Bool ObjectDelete(string name)

删除指定名称的对象。如果函数成功，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

```
name   -   要删除的对象名称。
```

示例:

```
ObjectDelete("text_object");
```

- 获取对象说明 ObjectDescription()

string ObjectDescription(string name)

返回对象说明。对于 OBJ_TEXT 和 OBJ_LABEL 类型对象，这些对象设置的文本将返回。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参见 ObjectSetText() 函数。

参数:

Name	-	对象名称。

示例:

```
//  将图表对象列表写入文件
int    handle, total;
string obj_name,fname;
// 文件名称
fname="objlist_"+Symbol();
handle=FileOpen(fname,FILE_CSV|FILE_WRITE);
if(handle!=false)
  {
   total=ObjectsTotal();
   for(int i=-; i<total; i++)
     {
      obj_name=ObjectName(i);
      FileWrite(handle,"Object "+obj_name+" description=
"+ObjectDescription(obj_name));
     }
   FileClose(handle);
  }
```

- 查找指定对象 ObjectFind()

int ObjectFind(string name)

查找指定名称的对象。函数返回包含找到的对象窗口索引。如果查找失败，返回 -1 。如果想获取详细的错误信息，请调用 GetLastError() 函数。图表子窗口 (如果图表子窗口带有指标)编号从 1 开始。图表主窗口总是存在且索引为零。

参数:

name	-	查找的对象名称。

示例:

```
if(ObjectFind("line_object2")!=win_idx) return(0);
```

- 获取指定对象的属性值 ObjectGet()

double ObjectGet(string name, int index)

函数返回指定对象的属性值。为了检查错误，请调用 GetLastError() 函数。

参见 ObjectSet()函数。

参数:

```
name    -   对象名称。
index   -   对象属性的索引。它可以是对象属性枚举值的任意值。
```

示例:

```
color oldColor=ObjectGet("hline12", OBJPROP_COLOR);
```

- 获取斐波纳契对象说明 ObjectGetFiboDescription()

string ObjectGetFiboDescription(string name, int index)

返回斐波纳契对象的水平线说明。斐波纳契水平线的数量取决于对象类型。斐波纳契水平线个数最大为 32。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参见 ObjectSetFiboDescription() 函数。

参数:

```
name    -   斐波纳契对象名称。
index   -   斐波纳契水平线索引(0-31)。
```

示例:

```
#include <stdlib.mqh>
  ...
  string text;
  for(int i=0; i<32; i++)
    {
     text=ObjectGetFiboDescription(MyObjectName,i);
     //---- 检查对象是否少于 32 条水平线
```

```
    if(GetLastError()!=ERR_NO_ERROR) break;
    Print(MyObjectName,"水平线: ",i," description: ",text);
   }
```

- 计算并返回柱子索引 ObjectGetShiftByValue()

int ObjectGetShiftByValue(string name, double value)

本函数计算并返回已给出价格的柱子索引（相对于当前柱子的偏移量）。柱子索引由第一坐标和第二坐标应用线性方程计算出来，适用于趋势线和相似的对象。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参见 ObjectGetValueByShift() 函数。

参数:

name	-	对象名称。
value	-	价格值。

示例:

```
int shift=ObjectGetShiftByValue("MyTrendLine#123", 1.34);
```

- 计算并返回指定柱子价格值 ObjectGetValueByShift()

Double ObjectGetValueByShift(string name, int shift)

本函数计算并返回指定柱子的价格值(相对于当前柱子的偏移量)。柱子索引由第一坐标和第二坐标应用线性方程计算出来，适用于趋势线和相似的对象。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参见 ObjectGetShiftByValue() 函数。

参数:

name	-	对象名称。
shift	-	柱子的索引。

示例:

```
double price=ObjectGetValueByShift("MyTrendLine#123", 11);
```

- 移动对象 ObjectMove()

Bool ObjectMove(string name, int point, datetime time1, double price1)

本函数在图表中可以移动一个对象座标。对象可能根据他们的类型有一个到三个座标。如果函数执行成功，返回 TRUE，否则，FALSE。如果想获取详细的错误信息，请调用 GetLastError() 函数。对象坐标从 0 开始。

参数:

name	-	对象名称。
point	-	坐标索引(0-2)。
time1	-	新的时间值。
price1	-	新价格值。

示例:

```
ObjectMove("MyTrend", 1, D'2005.02.25 12:30', 1.2345);
```

- 获取对象名 ObjectName()

string ObjectName(int index)

本函数根据对象列表中对象索引返回它的对象名称。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

index	-	对象列表中对象索引。对象索引必须大于等于 0，小于 ObjectsTotal()值。

示例:

```
int    obj_total=ObjectsTotal();
string name;
for(int i=0; i<obj_total; i++)
  {
   name=ObjectName(i);
   Print(i,"对象名称为 " + name);
  }
```

- 删除指定对象 ObjectsDeleteAll()

int ObjectsDeleteAll(Void window, void type)

在指定的图表子窗口中删除指定类型的全部对象。本函数返回已删除的对象个数。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

注：图表子窗口（如果子窗口图表带有指标)编号从 1 开始。主窗口总是存在且索引为零。如果窗口索引错误或值为-1，将从全部图表中删除对象。

如果 type 值等于 -1 或者这个参数丢失，全部对象将从指定的子窗口中删除。

参数:

window	-	可选的参数。要删除的对象所在的窗口索引，必须大于等于 -1（EMPTY 为默认值）并且小于 WindowsTotal()值。
type	-	可选的参数。被删除的对象类型。它可以是对象类型的任意枚举值或用 EMPTY 常量表示删除所有任意类型的全部对象。

示例:

```
ObjectsDeleteAll(2, OBJ_HLINE);  // 从第二个子窗口删除全部水平线。
ObjectsDeleteAll(2);              // 从第二个子窗口删除全部对象。
ObjectsDeleteAll();               //从图表中删除除全部对象。
```

- 修改指定对象属性 ObjectSet()

```
Bool ObjectSet(string name, int index, double value)
```

修改指定对象的属性值。如果函数成功，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参见 ObjectGet() 函数。

参数:

Name	-	对象名称。
Index	-	对象值的索引。它可以是任意对象属性枚举值。
Value	-	给定的新属性值。

示例:

```
// 将第一个坐标移到最后一个柱子的时间
ObjectSet("MyTrend", OBJPROP_TIME1, Time[0]);
// 设定第二个斐波纳契水平线
ObjectSet("MyFibo", OBJPROP_FIRSTLEVEL+1, 1.234);
// 设置对象可视性，对象显示在 15 分钟和 1 小时图表上
ObjectSet("MyObject", OBJPROP_TIMEFRAMES, OBJ_PERIOD_M15 |
OBJ_PERIOD_H1);
```

- 设置斐波纳契对象说明 ObjectSetFiboDescription()

```
Bool ObjectSetFiboDescription(string name, int index, string text)
```

本函数为斐波纳契对象的水平线设置一个新的说明。斐波纳契水平线数量取决于对象类型。斐波纳契水平线最多有 32 条。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

参数:

name	-	对象名称。
index	-	斐波纳契水平线的索引(0-31)。
text	-	新的水平线说明

示例:

```
ObjectSetFiboDescription("MyFiboObject",2,"Second line");
```

- 修改对象说明 ObjectSetText()

```
Bool ObjectSetText(    string name, string text, int font_size, void font, void text_color)
```

修改对象说明。对于 OBJ_TEXT 和 OBJ_LABEL 对象，这个说明作为图表上一个文本行显示。如果函数成功，返回 TRUE，否则，返回 FALSE。

如果想获取详细的错误信息，请调用 GetLastError() 函数。

font_size(字体大小)，font_name(字体名称) 和 text_color(文本颜色) 参数只能用于 OBJ_TEXT 和 OBJ_LABEL 对象。对于其它类型对象，这些参数被忽略。

参见 ObjectDescription() 函数。

参数:

name	-	对象名称。
text	-	描述对象的文本。
font_size	-	字体大小(点数)。
font	-	字体名称。
text_color	-	文本颜色。

示例:

```
ObjectSetText("text_object", "Hello world!", 10, "Times New Roman", Green);
```

- 获取指定类型对象总数 ObjectsTotal()

int ObjectsTotal(Void type)

返回图表中指定对象的总数。

参数:

type	-	可选的参数。要统计的对象类型，它可以是任意对象类型枚举值或用 EMPTY 常量表示统计任意类型的全部对象个数。

示例:

```
int    obj_total=ObjectsTotal();
string name;
for(int i=0; i<obj_total; i++)
  {
   name=ObjectName(i);
   Print(i,"对象名称#",i," is " + name);
  }
```

- 获取对象类型 ObjectType()

int ObjectType(string name)

本函数返回对象的类型值。

如果想获取详细的错误信息, 请调用 GetLastError() 函数。

参数:

name	-	对象名称。

示例:

```
if(ObjectType("line_object2")!=OBJ_HLINE) return(0);
```

字符串处理函数

一组处理字符串类型的函数。

- 字符串连接 StringConcatenate()

String StringConcatenate(...)

本函数生成字符串形式的数据并且返回。参数可以为任意类型，总数不超

过 64 个。

按照 Print(), Alert() 和 Comment()函数的同样规则，把参数转换成字符串，其返回值就是把函数参数转换成字符串再连接起来的结果。

StringConcatenate()函数要比使用加号运算符(+)连接字符串运行更快、更节省内存。

参数:

...	-	用逗号分隔所有价格值，最多 64 个参数。

示例:

```
string text;
text=StringConcatenate("账号保证金余额 ", AccountFreeMargin(), "
当前时间 ", TimeToStr(TimeCurrent()));
// 文本="账号保证金余额 " + AccountFreeMargin() + "当前时间 " +
TimeToStr(TimeCurrent())
Print(text);
```

- 搜索子字符串 StringFind()

```
int StringFind(string text, string matched_text, void start)
```

搜索子字符串。函数返回子字符串在搜索字符串中开始位置，如果未找到，返回 -1。

参数:

Text	-	被搜索的字符串。
matched_text	-	需要搜索的子字符串。
Start	-	搜索开始位置。

示例:

```
string text="快速的棕色小狗跳过懒惰的狐狸";
int index=StringFind(text, "小狗跳过", 0);
if(index!=16)
  Print("oops!");
```

- 获取字符串中指定字符 ASCII 值 StringGetChar()

```
int StringGetChar(string text, int pos)
```

返回字符串中指定位置的字符 ASCII 值。

参数:

```
text   -   字符串。
pos    -   字符串的字符位置，可以从 0 至 StringLen(text)-1。
```

示例:

```
int char_code=StringGetChar("abcdefgh", 3);
// 'c'字符码是 99
```

- 求字符串长度 StringLen()

```
int StringLen(string text)
```

返回一个字符串中字符个数。

参数:

```
text   -   要统计长度的字符串。
```

示例:

```
string str="some text";
if(StringLen(str)<5) return(0);
```

- 替换字符串中字符 StringSetChar()

```
string StringSetChar(string text, int pos, int value)
```

返回在指定位置被替换过字符的字符串。

参数:

```
text    -   要修改的字符串。
pos     -   字符串中字符位置，可以从 0 至 StringLen(text)。
value   -   新字符的 ASCII 码值。
```

示例:

```
string str="abcdefgh";
string str1=StringSetChar(str, 3, 'D');
// str1 is "abcDefgh"
```

- 截取子字符串 StringSubstr()

```
string StringSubstr(string text, int start, void length)
```

从字符串给出的位置起截取子字符串。

如果可能，此函数返回提取的子字符串，否则，返回一个空字符串。

参数:

text	-	要从中截取子字符串的字符串。
start	-	子字符串开始位置，可以是 0 至 StringLen(text)-1。
length	-	字符串截取长度。如果参数值大于等于 0 或者参数没有指定，从给定的位置起截取到串尾。

示例:

```
string text="The quick brown dog jumps over the lazy fox";
string substr=StringSubstr(text, 4, 5);
// 提取的子字符串是"quick"单词
```

- 删除字符串前导字符 StringTrimLeft()

string StringTrimLeft(string text)

本函数删除字符串左侧的回车符、空格和制表符。如果可能，函数将返回删除过的字符串，否则，返回空字符串。

参数:

text	-	左侧要截掉的字符串。

示例:

```
string str1="  Hello world   ";
string str2=StringTrimLeft(str);
// 截掉 str2 变量后将是 "Hello World   "
```

- 删除字符串尾部字符 StringTrimRight()

string StringTrimRight(string text)

本函数删除字符串右侧的回车符、空格和制表符。如果可能，函数将返回删除过的字符串，否则，返回空字符串。

参数:

text	-	要删除右侧字符的字符串。

示例:

```
string str1="  Hello world   ";
```

```
string str2=StringTrimRight(str);
// 删除后，str2 变量将是 "Hello World"
```

技术分析指标

用于计算标准指标和自定义指标的一组函数。

对于智能交易(或其他 MQL4 程序)接受其他指标的值，这个值不可能存在于图表之内。这个请求的指标将在调用模块线程中被加载和计算。

我们不仅可以计算当前图表中任何指标，还可以计算任何有效的货币对/图表时段的数据。如果请求数据(货币对名称/图表时段不同于当前图表)源自其他图表，出现这种情况可能是相应的图表没有在客户端内打开，数据需要从服务器上下载。这种情况下，错误信息 ERR_HISTORY_WILL_UPDATED (4066 - 历史数据正在更新中)将被放置于 last_error 变量中，需要再次请求更新数据(参看 ArrayCopySeries()范例)。

- 加速/减速振荡指标 iAC()

```
double iAC(string symbol, int timeframe, int shift)
```

计算比尔·威廉斯加速/减速振荡器。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double result=iAC(NULL, 0, 1);
```

- 离散指标 iAD()

```
double iAD(string symbol, int timeframe, int shift)
```

计算离散指标并且返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表

		时段。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double result=iAD(NULL, 0, 1);
```

- 鳄鱼指标 iAlligator()

```
double      string symbol, int timeframe, int jaw_period,
iAlligator( int jaw_shift, int teeth_period, int teeth_shift,
            int lips_period, int lips_shift, int ma_method,
            int applied_price, int mode, int shift)
```

计算比尔·威廉斯的鳄鱼指标并且退回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
jaw_period	-	蓝线平均周期(鳄鱼的下颌)。
jaw_shift	-	蓝线偏移量。
teeth_period	-	红线平均周期(鳄鱼的牙)。
teeth_shift	-	红线偏移量。
lips_period	-	绿线平均周期(鳄鱼的嘴唇)。
lips_shift	-	绿线偏移量。
ma_method	-	MA 方法。可以是任意的移动平均法。
applied_price	-	应用的价格。它可以是应用价格枚举的任意值。
mode	-	数据来源，指标线的标识符。可以是以下任意值: MODE_GATORJAW - 鳄鱼下颌(蓝色)平衡线, MODE_GATORTEETH - 鳄鱼牙(红色)平衡线, MODE_GATORLIPS - 鳄鱼嘴唇(绿色)平衡线。
shift	-	相对当前柱子向前移动一定数量周期的偏移量。

示例:

```
  double jaw_val=iAlligator(NULL, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA,
PRICE_MEDIAN, MODE_GATORJAW, 1);
```

- 平均动向指标 iADX()

```
double   string symbol, int timeframe, int period, int applied_price,
iADX(    int mode, int shift)
```

计算动向指标并且返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是图表任意时段枚举值，0 表示当前图表时段。
period	-	计算平均周期。
applied_price	-	请求价格。它可以是任意请求价格枚举值。
mode	-	指标线索引。它可以是任意指标线索引枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iADX(NULL,0,14,PRICE_HIGH,MODE_MAIN,0)>iADX(NULL,0,14,PRICE_HIG
H,MODE_PLUSDI,0)) return(0);
```

- 平均波幅通道指标 iATR()

```
double iATR(string symbol, int timeframe, int period, int shift)
```

计算平均波幅通道指标并且返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算平均周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  if(iATR(NULL,0,12,0)>iATR(NULL,0,20,0)) return(0);
```

- 动能指标 iAO()

double iAO(string symbol, int timeframe, int shift)

计算比尔·威廉斯的动能指标并且退回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iAO(NULL, 0, 2);
```

- 熊动力指标 iBearsPower()

double iBearsPower(string symbol, int timeframe, int period,
int applied_price, int shift)

计算熊功力指标并且返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
Timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
Period	-	计算平均周期。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iBearsPower(NULL, 0, 13,PRICE_CLOSE,0);
```

- 保力加通道指标 iBands()

double iBands(string symbol, int timeframe, int period, int deviation,
int bands_shift, int applied_price, int mode, int shift)

计算保力加通道技术指标并返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
Timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
Period	-	计算平均周期的主线。
Deviation	-	与主线的偏差。
bands_shift	-	指标相对图转移。
applied_price	-	应用的价格。它可以是任意的请求价格枚举值。
Mode	-	显示索引行。它可以是任意的指标线识别符枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iBands(NULL,0,20,2,0,PRICE_LOW,MODE_LOWER,0)>Low[0]) return(0);
```

- 基于数组保力加通道指标 iBandsOnArray()

```
double iBandsOnArray(double array[], int total, int period,
int deviation, int bands_shift, int mode, int shift)
```

基于数值型数组中数据，计算保力加通道指标。不同于 iBands (...)，iBandsOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	有数据的数组。
Total	-	要处理的元素个数。 0 意味整个数组。
Period	-	主线计算的平均周期。
Deviation	-	偏离主线的偏差。
bands_shift	-	相对图表的指标偏离量。
Mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iBands(ExtBuffer,total,2,0,MODE_LOWER,0)>Low[0]) return(0);
```

- 牛动力指标 iBullsPower()

```
double iBullsPower(string symbol, int timeframe, int period,
int applied_price, int shift)
```

计算牛动力指标并返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
Timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
Period	-	计算平均周期。
applied_price	-	请求的价格。它可以是任意的请求价格枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double val=iBullsPower(NULL, 0, 13,PRICE_CLOSE,0);
```

- 商品通道指标 iCCI()

```
double     string symbol, int timeframe, int period, int applied_price,
iCCI(      int shift)
```

计算商品通道指标并且返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算平均周期。
applied_price	-	请求的价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iCCI(NULL,0,12,PRICE_TYPICAL,0)>iCCI(NULL,0,20,PRICE_TYPICAL,0)
) return(0);
```

- 基于数组商品通道指标 iCCIOnArray()

```
double iCCIOnArray(double array[], int total, int period, int shift)
```

基于数值型数组中数据，计算商品通道指标。不同于 iCCI (...)，iCCIOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	存有数据的数组。
total	-	要处理的元素个数。0 意味整个数组。
period	-	计算平均周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iCCIOnArray(ExtBuffer,total,12,0)>iCCI(NULL,0,20,PRICE_TYPICAL,
0)) return(0);
```

- 自定义指标 iCustom()

```
double iCustom( string symbol, int timeframe, string name, ..., int mode,
int shift)
```

计算指定的自定义指标并返回它的值。自定义指标程序(*.EX4 文件)需要编译，并且程序必须放在 terminal_directory\experts\indicators 目录内。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
name	-	自定义指标编译过的程序名。
...	-	参数设置(如果需要)。传递的参数和他们的顺序必须与自定义指标外部参数声明的顺序和类型一致。
mode	-	行索引。可以从 0 到 7，并且必须与 SetIndexBuffer 函数使用的索引一致。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iCustom(NULL, 0, "示例 Ind",13,1,0);
```

- DeMarker 指标 iDeMarker()

```
double iDeMarker(string symbol, int timeframe, int period, int shift)
```

计算 DeMarker 指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算的平均周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iDeMarker(NULL, 0, 13, 1);
```

- 包络线指标 iEnvelopes()

```
Double      string symbol, int timeframe, int ma_period,
iEnvelopes( int ma_method, int ma_shift, int applied_price,
            double deviation, int mode, int shift)
```

计算包络线指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
ma_period	-	主线计算的平均周期。
ma_method	-	MA 方法。它可以是任意的移动平均方法枚举值。
ma_shift	-	MA 偏移量。根据图表时段，相对于图表的指标线偏移量。
applied_price	-	请求的价格。它可以是任意的请求价格枚举值。
deviation	-	与主线的百分比偏差。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一

		定数量周期的偏移量)。

示例:

```
  double val=iEnvelopes(NULL, 0,
13,MODE_SMA,10,PRICE_CLOSE,0.2,MODE_UPPER,0);
```

- 基于数组包络线指标 iEnvelopesOnArray()

```
double              double array[], int total, int ma_period,
iEnvelopesOnArray(  int ma_method, int ma_shift, double deviation,
                    int mode, int shift)
```

基于数值型数组中数据，计算包络线指标。不同于 iEnvelopes (...)，iEnvelopesOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	存有数据的数组。
total	-	要处理的元素个数。
ma_period	-	主线计算的平均周期。
ma_method	-	MA 方法。它可以是其中任意移动平均线枚举值。
ma_shift	-	MA 偏移量。根据图表时段，相对于图表的指标线偏移量。
deviation	-	与主线的偏差。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double val=iEnvelopesOnArray(ExtBuffer, 0, 13, MODE_SMA, 0.2,
MODE_UPPER,0 );
```

- 强力指标 iForce()

```
double      string symbol, int timeframe, int period, int ma_method,
iForce(     int applied_price, int shift)
```

计算强力指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算平均周期。
ma_method	-	MA 方法。它可以是其中任意移动平均线枚举值。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iForce(NULL, 0, 13,MODE_SMA,PRICE_CLOSE,0);
```

- 分形指标 iFractals()

```
double iFractals(string symbol, int timeframe, int mode, int shift)
```

计算分形指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iFractals(NULL, 0, MODE_UPPER, 3);
```

- 加多摆动指标 iGator()

```
Double   string symbol, int timeframe, int jaw_period, int jaw_shift,
iGator(  int teeth_period, int teeth_shift, int lips_period,
         int lips_shift, int ma_method, int applied_price, int mode,
         int shift)
```

计算加多摆动指标。 摆动指标在鳄鱼红线和蓝线(上部柱状图)之间以及红线和绿线(下部柱状图)之间显示是不同的。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
jaw_period	-	蓝线平均周期(鳄鱼的下颌)。
jaw_shift	-	蓝线相对图表的偏移量。
teeth_period	-	红线平均周期(鳄鱼的牙)。
teeth_shift	-	红线相对图表的偏移量。
lips_period	-	绿线平均周期(鳄鱼的嘴唇)。
lips_shift	-	绿线相对图表的偏移量。
ma_method	-	MA 方法。它可以是其中任意移动平均线枚举值。
applied_price	-	请求的价格。它可以是任意的请求价格枚举值。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double jaw_val=iGator(NULL, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA,
PRICE_MEDIAN, MODE_UPPER, 1);
```

- 一目平衡表指标 iIchimoku()

```
double        string symbol, int timeframe, int tenkan_sen,
iIchimoku(    int kijun_sen, int senkou_span_b, int mode, int shift)
```

计算一目平衡表指标(Ichimoku Kinko Hyo)并且返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称，NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
tenkan_sen	-	转换线(红线)周期。
kijun_sen	-	基准线(蓝线) 周期。
senkou_span_b	-	前移指标 B 周期。
mode	-	数据源。它可以是任意的一目平衡表模式枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double tenkan_sen=iIchimoku(NULL, 0, 9, 26, 52, MODE_TENKANSEN, 1);
```

- 比尔·威廉斯市场促进指数指标 iBWMFI()

```
double iBWMFI(string symbol, int timeframe, int shift)
```

计算比尔·威廉斯市场促进指数指标并且返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iBWMFI(NULL, 0, 0);
```

- 动量指标 iMomentum()

```
Double          string symbol, int timeframe, int period,
iMomentum(      int applied_price, int shift)
```

计算动量指标并返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
Timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
Period	-	价格变化计算的周期(总柱数)。
applied_price	-	请求的价格。它可以是任意的请求价格枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iMomentum(NULL,0,12,PRICE_CLOSE,0)>iMomentum(NULL,0,20,PRICE_CLOSE,0)) return(0);
```

- 基于数组动量指标 iMomentumOnArray()

```
double                  double array[], int total, int period,
iMomentumOnArray(       int shift)
```

基于数值型数组中数据，计算动量指标。不同于 iMomentum(...),iMomentumOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	数据数组。
total	-	数组元素的个数。
period	-	计算价格变化的周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iMomentumOnArray(mybuffer,100,12,0)>iMomentumOnArray(mubuffer,100,20,0)) return(0);
```

- 资金流量指标 iMFI()

```
double iMFI(string symbol, int timeframe, int period, int shift)
```

计算资金流量指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值。0 表示当前图表时段。
period	-	指标计算的周期(总柱数)。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iMFI(NULL,0,14,0)>iMFI(NULL,0,14,1)) return(0);
```

- 移动平均线指标 iMA()

```
double   string symbol, int timeframe, int period, int ma_shift,
iMA(     int ma_method, int applied_price, int shift)
```

计算移动平均线指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算的平均周期。
ma_shift	-	MA 偏移量。根据图表时段，相对于图表的指标线偏移量。
ma_method	-	MA 方法。它可以是其中任意移动平均方法枚举值。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
AlligatorJawsBuffer[i]=iMA(NULL,0,13,8,MODE_SMMA,PRICE_MEDIAN,i);
```

- 基于数组移动平均指标 iMAOnArray()

```
double          Double array[], int total, int period, int ma_shift,
iMAOnArray(     int ma_method, int shift)
```

基于数值型数组中数据,计算移动平均指标。不同于 iMA(...),iMAOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	存有数据的数组。
total	-	元素个数。
period	-	计算的平均周期。
ma_shift	-	MA 移动。根据图表时段，相对于图表的指标线偏移量。
ma_method	-	MA 方法。它可以是其中任意移动平均方法枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double macurrent=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,0);
double macurrentslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,0);
```

```
   double maprev=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,1);
   double maprevslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,1);
   //----
   if(maprev<maprevslow && macurrent>=macurrentslow)
     Alert("穿过");
```

- 移动平均振荡器指标 iOsMA()

```
double   string symbol, int timeframe, int fast_ema_period,
iOsMA(   int slow_ema_period, int signal_period, int applied_price,
         int shift)
```

计算移动平均振荡器指标并返回它的值。在某些系统中有时调用 MACD 直方柱。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
fast_ema_period	-	快速移动平均值计算的周期数。
slow_ema_period	-	慢速移动平均值计算的周期数。
signal_period	-	信号移动平均值计算的周期数。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iOsMA(NULL,0,12,26,9,PRICE_OPEN,1)>iOsMA(NULL,0,12,26,9,PRICE_O
PEN,0)) return(0);
```

- 平滑异同移动平均线指标 iMACD()

```
double   string symbol, int timeframe, int fast_ema_period,
iMACD(   int slow_ema_period, int signal_period, int applied_price,
         int mode, int shift)
```

计算平滑异同移动平均线指标并返回它的值。在系统中， OsMA 称作 MACD 直方柱， 这个指标显示二条线。 在客户端，平滑异同移动平均线指标被画为直方柱。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
fast_ema_period	-	快速移动平均值计算的周期数。
slow_ema_period	-	缓慢移动平均值计算的周期数。
signal_period	-	信号移动平均值计算的周期数。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定周期数的偏移量)。

示例:

```
if(iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0)>iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0)) return(0);
```

- 能量潮指标 iOBV()

```
double iOBV(string symbol, int timeframe, int applied_price,
            int shift)
```

计算能量潮(平衡成交量法、累积能量线)指标并返回它的值。

参数:

Symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
Timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
Shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double val=iOBV(NULL, 0, PRICE_CLOSE, 1);
```

- 抛物线转向指标 iSAR()

```
double      string symbol, int timeframe, double step, double maximum,
iSAR(       int shift)
```

计算抛物线转向指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
step	-	步长(增量)，通常为 0.02。
maximum	-	最大值，通常为 0.2。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iSAR(NULL,0,0.02,0.2,0)>Close[0]) return(0);
```

- 相对强弱指标 iRSI()

```
double iRSI(string symbol, int timeframe, int period, int applied_price, int shift)
```

计算相对强弱指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意图表时段枚举值，0 表示当前图表时段。
period	-	计算周期数。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>iRSI(NULL,0,14,PRICE_CLOSE,1))
return(0);
```

- 基于数组相对强弱指标 iRSIOnArray()

```
double iRSIOnArray(double array[], int total, int period, int shift)
```

基于数值型数组中数据，计算相对强弱指标。不同于 iRSI(...),iRSIOnArray 函数并不通过货币对名称、图表时段、请求价格的方

式获取数据，而是要求价格数据必须事先准备好，再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	存有数据的数组。
total	-	要处理的元素个数。
period	-	计算价格变化的周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iRSIOnArray(ExtBuffer,1000,14,0)>iRSI(NULL,0,14,PRICE_CLOSE,1))
return(0);
```

- 相对活力指标 iRVI()

```
double      String symbol, int timeframe, int period, int mode,
iRVI(       int shift)
```

计算相对活力指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
period	-	计算的周期数。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
  double val=iRVI(NULL, 0, 10,MODE_MAIN,0);
```

- 标准差指标 iStdDev()

```
double      string symbol, int timeframe, int ma_period, int ma_shift,
iStdDev(    int ma_method, int applied_price, int shift)
```

计算标准差指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
ma_period	-	MA 周期。
ma_shift	-	MA 偏移量。
ma_method	-	MA 方法。它可以是其中任意的移动平均法枚举值。
applied_price	-	请求价格。它可以是任意的请求价格枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iStdDev(NULL,0,10,0,MODE_EMA,PRICE_CLOSE,0);
```

- 基于数组标准差指标 iStdDevOnArray()

```
double              double array[], int total, int ma_period,
iStdDevOnArray(     int ma_shift, int ma_method, int shift)
```

计算标准离差指标在不同数组上的数据存储。与不同 iStdDev(...),iStdDevOnArray 函数不由标志名字， 时间周期，应用的价格采取数据。 必须提前准备价格数据。指标从左到右被计算。要对数组元素至于系列列阵(即从右到左)访问，你必须使用 ArraySetAsSeries 函数。

基于数值型数组中数据，计算标准差指标。不同于 iStdDev(...),iStdDevOnArray 函数并不通过货币对名称、图表时段、请求价格的方式获取数据,而是要求价格数据必须事先准备好,再从左边到右边计算指标。为了能按照序列数组那样存取数组元素(即从右到左)，你必须使用 ArraySetAsSeries 函数。

参数:

array[]	-	数据数组。
total	-	将计数的项目的数量。
ma_period	-	MA 周期。
ma_shift	-	MA 移动。
ma_method	-	MA 方法。它可以是其中任意平滑移动平均线枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
double val=iStdDevOnArray(ExtBuffer,100,10,0,MODE_EMA,0);
```

- 随机震荡指标 iStochastic()

```
double          string symbol, int timeframe, int %Kperiod,
iStochastic(    int %Dperiod, int slowing, int method, int price_field,
                int mode, int shift)
```

计算随机震荡指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表时段。
%Kperiod	-	%K 线周期。
%Dperiod	-	%D 线周期。
slowing	-	减速值。
method	-	MA 方法。它可以是任意的滑移动平均线枚举值。
price_field	-	价格参数。可以是下列值: 0 - Low/High 或者 1 - Close/Close。
mode	-	指标线索引。它可以是任意的指标线识别符枚举值。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_MAIN,0)>iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_SIGNAL,0))
    return(0);
```

- 威廉指标 iWPR()

```
double iWPR(string symbol, int timeframe, int period, int shift)
```

计算威廉指标并返回它的值。

参数:

symbol	-	要计算指标数据的证券货币对名称。NULL 表示当前货币对。
timeframe	-	图表时段。可以是任意的图表时段枚举值，0 表示当前图表

		时段。
period	-	计算周期。
shift	-	从指标缓冲区获取值的索引(相对当前柱子向前移动一定数量周期的偏移量)。

示例:

```
if(iWPR(NULL,0,14,0)>iWPR(NULL,0,14,1)) return(0);
```

获取时段内价格数据

用于获取货币对/时段内价格数据的一组函数。

如果请求数据(货币对名称/时段不同于当前图表)来自其他图表，这种情况可能是相应的图表没有在客户端内打开，并且需要从服务器上下载数据。这种情况下， 错误 ERR_HISTORY_WILL_UPDATED (4066 - 请求的历史数据正在更新中)将被放置于 last_error 变量中，并且不得不重新请求更新(参看 ArrayCopySeries()范例)。

在测试中，同一货币对价格数据或不同时段的价格数据被精确地模拟（除成交量外）。其他时段的成交量不模拟，其它货币对的价格数据同样也不模拟。无论何种情况，一个时段内柱子总数会被精确地模拟。

- 获取柱数 iBars()

```
int iBars(string symbol, int timeframe)
```

返回指定的图表上柱子的数量。

对于当前图表，柱子数量的信息保存在预定义的变量 Bars 中。

参数:

Symbol	-	用于计算指标的货币对,NULL 意味着使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着采用当前图表时段。

示例:

```
  Print("PERIOD_H1 时段下货币对'EURUSD'柱数
",iBars("EUROUSD",PERIOD_H1));
```

- 搜索柱子 iBarShift()

```
int iBarShift(string symbol, int timeframe, datetime time, void exact)
```

根据开盘时间搜索柱子。本函数返回带指定开盘时间的柱子。如果带指定开盘时间的柱子丢失，函数将返回-1 或最近的柱子，这取决于 exact 参数设置。

参数:

Symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
Timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
Time	-	查找值 (柱子的开盘时间)。
Exact	-	未发现柱子的返回模式。false-返回最近的柱子，true-返回 -1。

示例:

```
datetime some_time=D'2004.03.21 12:00';
int      shift=iBarShift("EUROUSD",PERIOD_M1,some_time);
Print("带有打开时间的柱子 ",TimeToStr(some_time)," 是 ",shift);
```

- 柱子收盘价 iClose()

double iClose(string symbol, int timeframe, int shift)

返回带有时段和偏移量的指定货币对柱子的收盘价。如果本地历史表为空(历史数据未加载)，函数返回 0。

对于当前图表，收盘价的信息保存在预定义数组 Close[]中。

参数:

Symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
Shift	-	从指标缓冲区上获取值的索引(相对于当前柱子的偏移量)。

示例:

```
Print("对于 USDCHF H1 当前柱: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

- 柱子最高值 iHigh()

```
double iHigh(string symbol, int timeframe, int shift)
```

返回带有图表时段和偏移量的指定货币对柱子的最高值。如果本地历史表为空(历史数据未加载)，函数返回 0。

对于当前图表，最高价的信息保存在预定义数组 High[]中。

参数:

Symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
Shift	-	从指标缓冲区上获取值的索引(相对于当前柱子的偏移量)。

示例:

```
  Print("对于 USDCHF H1 当前柱: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

- 柱子最高值偏离量 iHighest()

```
int          string symbol, int timeframe, int type, void count,
iHighest(    void start)
```

根据类型返回在一个指定时间周期内最大值的偏离量。

参数:

Symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
Timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
Type	-	序列数组的识别符。它可以是序列数据识别符枚举的任意值。
Count	-	要计算出的周期数(沿着开头柱子到后面柱子的方向)。
Start	-	开始柱子，相对于当前柱子，开始采集数据的柱子。

示例:

```
  double val;
  // 在 20 个连续柱子范围内计算最大值
  // 在当前图表上从第 4 个至第 23 个的索引
  val=High[iHighest(NULL,0,MODE_HIGH,20,4)];
```

- 柱子低值 iLow()

```
double iLow(string symbol, int timeframe, int shift)
```

返回带有图表时段和偏移量的指定货币对柱子的低值。如果本地历史表为空(历史数据未加载)，函数返回 0。

对于当前图表，低价格的信息保存在预定义数组 Low[]中。

参数:

symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
shift	-	从指标缓冲区上获取值的索引。

示例:

```
  Print("对于 USDCHF H1 当前柱: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

- 最小值的偏离量 iLowest()

```
int iLowest(string symbol, int timeframe, int type, void count,
            void start)
```

根据类型返回在一个指定时间周期内最小值的偏离量。

参数:

symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
type	-	序列数组的识别符。它可以是任意的序列数组识别符枚举

		值。
count	-	时间周期数。
start	-	开始柱子，相对于当前柱子，开始采集数据的柱子。

示例:

```
// 在范围内计算连续 10 个柱的最低值
// 在当前图表从第 10 个到第 19 个的索引
double val=Low[iLowest(NULL,0,MODE_LOW,10,10)];
```

- 柱子开盘价 iOpen()

double iOpen(string symbol, int timeframe, int shift)

返回带有图表时段和偏移量的指定货币对柱子的开盘价。如果本地历史表为空(历史数据未加载)，函数返回 0。

对于当前图表，开盘价的信息保存在预定义数组 Open[]中。

参数:

symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
shift	-	从指标缓冲区上获取值的索引(相对于当前柱子的偏移量)。

示例:

```
  Print("对于 USDCHF H1 当前柱: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

- 柱子时间值 iTime()

datetime iTime(string symbol, int timeframe, int shift)

返回带有图表时段和偏移量的指定货币对柱子的时间值。如果本地历史表为空(历史数据未加载)，函数返回 0。

对于当前图表，时间信息保存在预定义数组 Time[]中。

参数:

Symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。。
Shift	-	从指标缓冲区上获取值的索引(相对于当前柱子的偏移量)。

示例:

```
  Print("对于 USDCHF H1 当前货币对: ",iTime("USDCHF",PERIOD_H1,i),",
",  iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

- 一跳成交量 iVolume()

```
double iVolume(string symbol, int timeframe, int shift)
```

返回带有图表时段和偏移量的指定货币对柱子的一跳成交量。如果本地历史表为空(历史数据未加载)，函数返回 0。

如果加载历史为空，函数返回 0。

对于当前图表，关于成交量的信息在预定义数组中命名 Volume[]。

参数:

symbol	-	用于计算指标的货币对名称，NULL 意味使用当前货币对名称。
timeframe	-	图表时段。可以是时段枚举的任意值，0 意味着使用当前图表时段。
shift	-	从指标缓冲区上获取值的索引(相对于当前柱子的偏移量)。

示例:

```
  Print("对于 USDCHF H1 的当前柱: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",

iHigh("USDCHF",PERIOD_H1,i),", ",  iLow("USDCHF",PERIOD_H1,i),", ",

iClose("USDCHF",PERIOD_H1,i),", ", iVolume("USDCHF",PERIOD_H1,i));
```

交易处理函数

交易管理的一组函数。

从自定义指标中不能调用 OrderSend()、 OrderClose、 OrderCloseBy、 OrderDelete 和 OrderModify 交易函数。

本组交易函数应用于智能交易和脚本中。只有特定智能交易的“允许实时交易”属性被选中，才能调用交易函数。

为了在智能交易和脚本中进行交易，在程序交易环境(智能交易和脚本中自动交易的环境)中，只能为它提供一个线程。这就是为什么，如果这个交易环境被一个智能交易操作占用，由于会出现 146 错误 (ERR_TRADE_CONTEXT_BUSY)，其他智能交易或脚本在此时就不能调用成功。为了能确定是否有交易在进行，可使用 IsTradeAllowed()函数检测。为了清除交易环境下访问共享，我们能够使用一个基于全局变量的交易信号，其值可以使用 GlobalVariableSetOnCondition()函数改变。

- 交易函数错误代码

任何交易业务(OrderSend(), OrderClose, OrderCloseBy, OrderDelete 和 OrderModify 函数)都会因为一些原因导致失败，返回负的定单号或 FALSE，我们通过调用 GetLastError()函数能找出失败的原因。每一个错误必须以不同的方式加以处理，最常见的建议列举如下：

从交易服务器返回的错误代码：

常数	值	描述
ERR_NO_ERROR	0	交易业务成功。
ERR_NO_RESULT	1	OrderModify 试图用一个同样的值替换原先已设定好的值。应该修改其中一个或多个值，再反复尝试着修改。
ERR_COMMON_ERROR	2	常规错误。所有交易必须停止运行，直到查清错误为止。客户端和操作系统可能要重启。
ERR_INVALID_TRADE_参数	3	传递给交易函数的参数无效，例如, 货币对错误,未知交易操作, 负的下滑值，不存在的定单号等等。程序必须修改。
ERR_SERVER_BUSY	4	交易服务器忙。过一段时间(几分钟)后再重试。

ERR_OLD_VERSION	5	客户端的旧版本，要求安装最新版。
ERR_NO_CONNECTION	6	交易服务器没有接不上。需要确认连接没有断开(例如，应用 IsConnected 函数)，过一段时间(5 秒后)再重试。
ERR_TOO_FREQUENT_REQUESTS	8	交易请求过于频繁，必须减少请求次数，程序需要修改。
ERR_ACCOUNT_DISABLED	64	账户被禁用，所有交易必须停止。
ERR_INVALID_ACCOUNT	65	账号无效，所有交易必须停止。
ERR_TRADE_TIMEOUT	128	交易超时。在重试前(至少 1 分钟时间)，必须确认交易业务确实没有成功（一个新头寸没有建立，已存在定单未被修改或删除，已存在的头寸未平单）
ERR_INVALID_PRICE	129	无效开价或报价，也许是价格不正常。延时 5 秒后，有必要用 RefreshRates 函数更新数据，再重试。 如果错误没有消失，停止所有交易，修改程序。
ERR_INVALID_STOPS	130	止损太近、价格计算错误和非正常价。 由于价格过时，发生错误时，需要反复重试。延时 5 秒后，有必要使用 RefreshRates()函数刷新数据再次重试。如果错误依然没有消失，尝试停止所有运行交易，修改程序。
ERR_INVALID_TRADE_VOLUME	131	无效交易量，尝试停止所有运行交易，修改程序。
ERR_MARKET_CLOSED	132	市场关闭。过一段时间后，重新尝试。
ERR_TRADE_DISABLED	133	交易禁止。所有运行交易必须停止。
ERR_NOT_ENOUGH_MONEY	134	资金不足无法交易。参数相同的交易不能重复进行。延时 5 秒或更多，再用小额资金重试，但要确定有足够的资金完成交易。
ERR_PRICE_CHANGED	135	价格已改变。用 RefreshRates 函

		数无延时更新数据，再重试。
ERR_OFF_QUOTES	136	没有报价。由于有多种原因，交易商不提供价格或拒绝提供价格(比方说，本场交易价格未启动，价格不确定，市场变化太快)。延时 5 秒后，有必要用 RefreshRates 函数更新数据，再重试。
ERR_REQUOTE	138	请求的报价已过时，或者买价和卖价混淆。延时 5 秒后，有必要使用 RefreshRates()函数刷新数据再次重试。如果错误依然没有消失，尝试停止所有运行交易，修改程序。
ERR_ORDER_LOCKED	139	交易定单被锁住，正在处理中。尝试停止所有运行交易，修改程序。
ERR_LONG_POSITIONS_ONLY_ALLOWED	140	只允许买进，禁止卖出。
ERR_TOO_MANY_REQUESTS	141	请求过多，必须减少请求次数，修改程序。
	142	定单按次序排列。它不是一个错误，而是客户端和交易服务器之间一个代码。在交易执行期间，碰巧连接断开或重新连接时，这种代码的出现次数非常少。此代码和错误代码 128 一样处理。
	143	定单已经被执行交易商接受。它不是一个错误,而是客户端和交易服务器之间一个代码。它和代码 142 出现的原因一样，处理方法参照错误 128。
	144	在手动确认期间，定单已经被客户放弃。它不是一个错误,而是客户端和交易服务器之间一个代码。
ERR_TRADE_MODIFY_DENIED	145	由于定单太接近市价或被锁定，修改被否定。延时 15 秒后，用 RefreshRates 函数更新数据再次重试。
ERR_TRADE_CONTEXT_BUSY	146	交易线程忙。只有在 IsTradeContextBusy 函数返回

		FALSE 后重试。
ERR_TRADE_EXPIRATION_DENIED	147	交易商否定挂单过期使用。如果期限为零，交易可以重试。
ERR_TRADE_TOO_MANY_ORDERS	148	开仓和挂单交易总数已经达到经纪商设定的限度。只有在现有仓位平单或挂单删除之后才可以新开仓或挂单。

- 平单 OrderClose()

```
bool OrderClose( int ticket, double lots, double price, int slippage, void Color)
```

定单平单。如果函数执行成功，返回 TRUE。如果函数执行失败，返回 FALSE。想要获得详细错误信息，请调用 GetLastError()函数。

参数:

ticket	-	定单号。
Lots	-	手数。
price	-	收盘价。
slippage	-	最高滑点数。
Color	-	图表中平单箭头颜色。如果参数丢失或用 CLR_NONE 值，将不会在图表中画出。

示例:

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
  {
   OrderClose(order_id,1,Ask,3,Red);
   return(0);
  }
```

- 逆序平单 OrderCloseBy()

```
bool OrderCloseBy(int ticket, int opposite, void Color)
```

按仓位逆序进行平单。如果函数执行成功，返回 TRUE。如果失败，返回 FALSE。想要获得详细错误信息，请调用 GetLastError()函数。

参数:

Ticket	-	定单号。

opposite	-	定单逆序顺序号
Color	-	图表中平单箭头颜色。如果参数丢失或用 CLR_NONE 值，将不会在图表中画出。

示例:

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
  {
   OrderCloseBy(order_id,opposite_id);
   return(0);
  }
```

- 收盘价 OrderClosePrice()

double OrderClosePrice()

返回当前定单的收盘价。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(ticket,SELECT_BY_POS)==true)
  Print("定单收盘价 ", "定单编号= ",OrderClosePrice()的收盘价格);
else
  Print("OrderSelect 失败错误代码是",GetLastError());
```

- 平单时间 OrderCloseTime()

datetime OrderCloseTime()

返回当前定单的平单时间。如果定单的平单时间不是 0，所选定单会从账户历史重新尝试。开仓和挂单交易的平仓时间必须等于 0。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10,SELECT_BY_POS,MODE_HISTORY)==true)
  {
   datetime ctm=OrderOpenTime();
   if(ctm>0) Print("定单 10 开仓时间", ctm);
   ctm=OrderCloseTime();
   if(ctm>0) Print("定单 10 平单时间", ctm);
  }
```

```
else
  Print("OrderSelect 失败错误代码是",GetLastError());
```

- 定单注释 OrderComment()

string OrderComment()

返回当前定单的注释。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
string comment;
if(OrderSelect(10,SELECT_BY_TICKET)==false)
  { Print("OrderSelect 失败错误代码是",GetLastError());
   return(0);
  }
comment = OrderComment();
// ...
```

- 计算定单佣金数 OrderCommission()

double OrderCommission()

计算当前定单的佣金数。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10,SELECT_BY_POS)==true)
  Print("定单 10 佣金",OrderCommission());
else
 Print("OrderSelect 失败错误代码是",GetLastError());
```

- 删除挂单 OrderDelete()

bool OrderDelete(int ticket, void Color)

删除以前挂单。如果函数执行成功，返回 TRUE,如果失败，返回 FALSE。想要获得详细错误信息，请调用 GetLastError()函数。

参数:

```
ticket   -   定单编号。
```

Color	-	图表中平单箭头颜色。如果参数丢失或用 CLR_NONE 值，将不会在图表中画出。

示例:

```
if(Ask>var1)
  {
   OrderDelete(order_ticket);
   return(0);
  }
```

- 挂单过期时间 OrderExpiration()

datetime OrderExpiration()

返回挂单的过期时间。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
  Print("定单 #10 过期时间为",OrderExpiration());
else
  Print("OrderSelect 返回的",GetLastError()错误);
```

- 当前定单手数 OrderLots()

double OrderLots()

返回当前定单的手数。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10,SELECT_BY_POS)==true)
  Print("定单 10 手数",OrderLots());
else
  Print("OrderSelect 返回的 ",GetLastError()错误);
```

- 所选订单魔数编号 OrderMagicNumber()

int OrderMagicNumber()

返回当前选定的定单魔数编号(魔数不会重复)。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10,SELECT_BY_POS)==true)
  Print("定单 10 魔数编号", OrderMagicNumber());
else
  Print("OrderSelect 返回的 ",GetLastError()错误);
```

- 修改挂单 OrderModify()

```
bool          int ticket, double price, double stoploss,
OrderModify(  double takeprofit, datetime expiration,
              void arrow_color)
```

修改以前的开仓或挂单参数。如果函数成功，返回 TRUE。如果函数失败，返回 FALSE。 如果想获取详细的错误信息，请调用 GetLastError()函数。

注: 只有挂单才能修改开盘价和过期时间。

如果用原值作为参数传递给函数，将会产生 1 错误 (ERR_NO_RESULT)。

在某些交易服务器上，挂单的过期时间被禁用。这种情况下，当在过期参数中指定一个非零值时，将生成 147 错误(ERR_TRADE_EXPIRATION_DENIED)。

参数:

Ticket	-	定单编号。
price	-	收盘价
stoploss	-	新的止损水平。
takeprofit	-	新的赢利水平。
expiration	-	挂单有效时间。
arrow_color	-	在图表中允许对止损/赢利的箭头颜色进行修改。如果参数丢失或用 CLR_NONE 值，在图表中将不再显示。

示例:

```
if(TrailingStop>0)
  {
   OrderSelect(12345,SELECT_BY_TICKET);
   if(Bid-OrderOpenPrice()>Point*TrailingStop)
     {
      if(OrderStopLoss()<Bid-Point*TrailingStop)
        {
```

```
OrderModify(OrderTicket(),OrderOpenPrice(),Bid-Point*TrailingStop,
OrderTakeProfit(),0,Blue);
           return(0);
          }
       }
    }
```

- 定单开盘价 OrderOpenPrice()

double OrderOpenPrice()

返回当前定单开盘价。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10, SELECT_BY_POS)==true)
  Print("对于定单 10 开盘价",OrderOpenPrice());
else
  Print("OrderSelect 返回错误",GetLastError());
```

- 开单时间 OrderOpenTime()

datetime OrderOpenTime()

返回当前定单的开单时间。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10, SELECT_BY_POS)==true)
  Print("定单 10 开单时间",OrderOpenTime());
else
  Print("OrderSelect 返回的错误 ",GetLastError());
```

- 打印选择定单信息 OrderPrint()

Void OrderPrint()

按以下格式打印日志中当前定单信息:

定单编号； 开单时间； 交易业务； 手数总数； 开盘价； 止损价； 止赢价； 平单时间； 收盘价； 佣金； 掉期； 盈利； 注释； 指定编码； 挂单过期日期

定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
  OrderPrint();
else
 Print("OrderSelect 失败错误代码是",GetLastError());
```

- 定单净盈利金额 OrderProfit()

double OrderProfit()

返回当前定单的净盈利金额 (除掉期和佣金外)。对于开仓部位，当前盈利并不真实。对于平单，它为固定盈利。

返回当前定单盈利。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(10, SELECT_BY_POS)==true)
  Print("定单 10 盈利",OrderProfit());
else
  Print("OrderSelect 返回的错误",GetLastError());
```

- 选择定单 OrderSelect()

Bool OrderSelect(int index, int select, void pool)

本函数选择一个定单，等待做进一步地处理。如果函数成功，返回 TRUE。如果函数失败，返回 FALSE。想要获得详细错误信息，请调用 GetLastError()函数。

如果通过定单编号选定定单，pool 参数应忽略。此定单编号是其唯一识别符。找出所选定单的列表，对平单时间进行分析。如果平单时间为零，此定单就是开单或挂单，或从终端开仓部位列表中选出。我们可以从定单类型区别开单和挂单。如果定单的平单时间不等于 0，此定单就是一个平单，或是一个已删除的挂单，也可能是从终端历史中被选出来的，他们同样可以根据定单类型相互区别。

参数:

index	-	定单索引或定单，这取决于第二个参数。
select	-	选定标记。可以为以下的任意值: SELECT_BY_POS - 定单表中索引 SELECT_BY_TICKET - 索引就定单。
pool	-	可选择定单索引。当选择 SELECT_BY_POS 参数时使用，可以为以下的任意值: MODE_TRADES (default)- 来自于交易列表中定单(开单和挂单)， MODE_HISTORY - 来自于历史表中的定单(平单和取消定单)。

示例:

```
  if(OrderSelect(12470, SELECT_BY_TICKET)==true)
    {
     Print("定单 #12470 开盘价", OrderOpenPrice());
     Print("定单 #12470 收盘价 ", OrderClosePrice());
    }
  else
    Print("OrderSelect 返回的错误 ",GetLastError());
```

- 发出定单 OrderSend()

```
int         string symbol, int cmd, double volume, double price,
OrderSend( int slippage, double stoploss, double takeprofit,
           void comment, void magic, void expiration,
           void arrow_color)
```

主要功能用于开仓部位和挂单交易。

由交易服务器返回定单的定单编号，如果失败，返回-1。想要获得额外的错误信息, 请调用 GetLastError()函数。

注：市价定单开始交易时(OP_SELL 或 OP_BUY)，只有最新卖价或买价可以当作开盘价来用。如果执行与当前证券不同的业务,必须使用带有 MODE_BID 或 MODE_ASK 参数的 MarketInfo()函数获得要买的证券最新的报价。 预测或是不标准的价格不可用。如果在市场的报价中没有所要的开盘价或者没有按照小数点后位数的要求标准化，将会导致 129 错误(ERR_INVALID_PRICE)。 如果要求的开盘价日期完全过期, 将会导致 138 错误(ERR_REQUOTE)。 如果请求的价格过期了，但是还处于市价里，就以现价或仍在 price+-slippage(现价+-滑点)范围内现价开仓。

止损价和赢利价不能太靠近市价。 最小的止损点数可以使用带 MODE_STOPLEVEL 参数的 MarketInfo ()函数得到。 在出错或止损位设置不正确的情况下，将会导致 130 错误(ERR_INVALID_STOPS)。

在发出挂单时,开盘价也不能太靠近市价。挂单价和当前市价之间最小的点数可以使用带 MODE_STOPLEVEL 参数的 MarketInfo ()函数得到。在挂单开盘价错误的情况下，将会导致 130 错误(ERR_INVALID_STOPS)。

挂单交易的过期时间在一些交易服务器上被禁用。在这种情况下，如果 expiration（过期）参数指定了非零值，反而会导致 147 错误 (ERR_TRADE_EXPIRATION_DENIED)产生。

在某些交易服务器上，开仓单和挂单的总数有所限制。如果超出限额，就不能再开仓，或者不能再挂单了。如果还开仓或挂单，交易服务器会返回 148 错误 (ERR_TRADE_TOO_MANY_ORDERS)。

参数:

symbol	-	交易货币对。
cmd	-	购买方式。可以是任意的购买方式枚举值。
volume	-	购买手数。
price	-	交易首选价。
slippage	-	买卖定单的最大允许滑点数。
stoploss	-	止损水平。
takeprofit	-	赢利水平。
comment	-	注释文本。注释的最后部分可以由服务器修改。
magic	-	定单的指定号码。可以当作用户定义的识别码使用。
expiration	-	定单过期时间(只限挂单)。
arrow_color	-	图表上开仓箭头标记的颜色。如果参数丢失或用 CLR_NONE 值，开仓箭头就不在图表中画出。

示例:

```
  int ticket;
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
    {

ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Ask-25*Point,Ask+25*Point
,"My order #2",16384,0,Green);
     if(ticket<0)
       {
        Print("OrderSend 失败错误 #",GetLastError());
        return(0);
       }
    }
```

- 历史表中平单数 OrdersHistoryTotal()

int OrdersHistoryTotal()

返回载入到终端账户历史表中平单数。历史表的大小取决于终端"Account history（账户历史）" 标签中当前设置。

示例:

```
// 从交易历史表中获得的信息
int i,hstTotal=OrdersHistoryTotal();
for(i=0; i<hstTotal; i++)
  {
   //---- 检查选择结果
   if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
     {
      Print("访问历史表失败，错误信息 (",GetLastError(),")");
      break;
     }
   // 处理定单
```

- 止损值 OrderStopLoss()

double OrderStopLoss()

返回当前定单的止损值。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(ticket,SELECT_BY_POS)==true)
  Print("定单 10 止损值", OrderStopLoss());
else
 Print("OrderSelect 失败错误码是",GetLastError());
```

- 定单总数 OrdersTotal()

int OrdersTotal()

返回市场单和挂单的总数。

示例:

```
int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");
if(handle<0) return(0);
```

```
  // 写标题
  FileWrite(handle,"#","开盘价","买入时间","货币对","手数");
  int total=OrdersTotal();
  // 编写定单命令
  for(int pos=0; pos<total; pos++)
    {
     if(OrderSelect(pos,SELECT_BY_POS)==false) continue;

FileWrite(handle,OrderTicket(),OrderOpenPrice(),OrderOpenTime(),Or
derSymbol(),OrderLots());
    }
  FileClose(handle);
```

- 定单掉期值 OrderSwap（）

double OrderSwap()

返回当前定单掉期值。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
  if(OrderSelect(order_id, SELECT_BY_TICKET)==true)
    Print("定单 #", order_id, " 掉期 ",OrderSwap());
  else
   Print("OrderSelect 失败错误代码是",GetLastError());
```

- 定单的货币对 OrderSymbol()

string OrderSymbol()

返回当前定单的货币对。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
  if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("定单 #", OrderTicket(), " 货币对 ", OrderSymbol());
  else
   Print("OrderSelect 失败错误代码是",GetLastError());
```

- 赢利金额 OrderTakeProfit()

double OrderTakeProfit()

返回当前定单的赢利金额。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(12, SELECT_BY_POS)==true)
  Print("定单 #",OrderTicket()," 盈利: ", OrderTakeProfit());
else
  Print("OrderSelect() 返回错误 - ",GetLastError());
```

- 定单编号 OrderTicket()

int OrderTicket()

返回当前定单的定单编号。

注:定单必须用 OrderSelect()函数提前选定。

示例:

```
if(OrderSelect(12, SELECT_BY_POS)==true)
  order=OrderTicket();
else
 Print("OrderSelect 失败错误代码",GetLastError());
```

- 定单类型 OrderType()

int OrderType()

返回当前定单的定单类型。可以是以下的任意值:

OP_BUY -买进,
OP_SELL - 卖出,
OP_BUYLIMIT - 挂单买入限定,
OP_BUYSTOP - 挂单止损限定,
OP_SELLLIMIT - 挂单卖出限定,
OP_SELLSTOP - 挂单止损限定。

注: 定单必须由 OrderSelect()函数选择。

示例:

```
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
  {
```

```
    order_type=OrderType();
    // ...
   }
 else
   Print("OrderSelect() 返回错误 - ",GetLastError());
```

窗口数据处理函数

用于当前图表窗口的一组函数。

- 隐藏测试指标标志 HideTestIndicators()

void HideTestIndicators(bool hide)

本函数设置一个供智能交易调用的隐藏指标标志。在智能交易测试完成，打开了相应的图表，标出过的指标将不会出现在测试图表中。每个调用过的指标将首先用当前隐藏的标志标记。

特别要注意到，只有那些直接从智能交易调用的指标才可以在测试图表中画出。

参数:

hide - 如果需要隐藏指标，其值为 TRUE，否则，为 FALSE。

示例:

```
HideTestIndicators(true);
MaCurrent=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,0);
MaPrevious=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,1);
HideTestIndicators(false);
```

- 获取图表时段 Period()

int Period()

返回在用时段(图表周期)的分钟总数。

示例:

```
Print("时段 ", Period());
```

- 刷新数据 RefreshRates()

bool RefreshRates()

刷新预定义变量和序列数组中数据。当智能交易计算时间过长时，本函数可以自动更新数据。如果数据被更新，返回 TRUE，否则，返回 FALSE。数据不被更新的唯一原因就是他们是客户端的当前数据。

智能交易和脚本只处理它们自己的历史数据的副本。在智能交易和脚本第一次启动的时候，当前货币对数据就已经被复制过。每次智能交易或脚本启动时，会更新最初得到的副本。在智能交易和脚本运行时，可能会收到一个或多个新的步进数据，现有数据可能会过期。

示例:

```
   int ticket;
   while(true)
     {
      ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"expert
comment",255,0,CLR_NONE);
      if(ticket<=0)
        {
         int error=GetLastError();
         //---- 资金不足
         if(error==134) break;
         //---- 等待 10 秒钟
         Sleep(10000);
         //---- 刷新价格数据
         RefreshRates();
         break;
        }
      else
        {
         OrderSelect(ticket,SELECT_BY_TICKET);
         OrderPrint();
         break;
        }
     }
```

- 获取货币对名称 Symbol()

string Symbol()

返回当前货币对名称。

示例:

```
int total=OrdersTotal();
for(int pos=0; pos<total; pos++)
  {
   // 因为此时可能平单或删除定单，检测选择结果!
   if(OrderSelect(pos, SELECT_BY_POS)==false) continue;
   if(OrderType()>OP_SELL || OrderSymbol()!=Symbol()) continue;
   // 执行过程...
  }
```

- 获取图表中柱子总数 WindowBarsPerChart()

int WindowBarsPerChart()

本函数返回在图表上可见的柱子总数。

示例:

```
// 对于可见柱工作。
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0;  i<bars_count;  i++,bar--)
  {
   // ...
  }
```

- 获取程序名称 WindowExpertName()

string WindowExpertName()

返回智能交易、脚本、自定义指标和库的名称，这取决于 MQL4 程序的调用。

示例:

```
string name=WindowExpertName();
GlobalVariablesDeleteAll(name);
```

- 搜索窗口中指标 WindowFind()

int WindowFind(String name)

如果发现有名称的指标，本函数返回包含指定指标的窗口索引，否则，返回 -1。

注：当 init()函数运行时，如果自定义指标自己找自己，则 WindowFind()函数返回 -1 。

参数:

name	-	指标简称。

示例:

```
int win_idx=WindowFind("MACD(12,26,9)");
```

- 获取第一个显示的柱子 WindowFirstVisibleBar()

Int WindowFirstVisibleBar()

本函数在当前图表窗口中返回第一个显示的柱子。必须考虑到价格柱子的逆序编号，即从最后一个价格到第一个价格排列。在价格数组中当前柱子是最新的，索引为 0，最老的柱子索引为 Bars-1。如果第一个显示的柱子编号为 2 或更高，但少于图表中能显示的柱子总数，这就意味着图表窗口没有完全填满，离窗口侧边还有空白。

示例:

```
// 处理显示的柱子
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0;  i<bars_count;  i++,bar--)
  {
   // ...
  }
```

- 获取系统窗口句柄 WindowHandle()

int WindowHandle(String symbol, int timeframe)

返回包含特定图表的系统窗口句柄。如果图表在函数调用时还没有打开，返回为 0。

参数:

symbol	-	货币对名称。
timeframe	-	时段。可以是任意的时段枚举值，0 意味着使用当前图表中时段。

示例:

```
int win_handle=WindowHandle("USDX",PERIOD_H1);
```

```
if(win_handle!=0)
  Print("发现带有 USDX 和 H1 的窗口。价格数组将被立即复制。");
```

- 图表子窗口是否可见 WindowIsVisible()

bool WindowIsVisible(int index)

如果图表子窗口可见，返回 TRUE，否则，返回 FALSE。由于指标的可视属性可以放置在图表子窗口里，图表子窗口能被隐藏。

参数:

index - 图表子窗口索引。

示例:

```
int maywin=WindowFind("MyMACD");
if(maywin>-1 && WindowIsVisible(maywin)==true)
  Print("MyMACD 窗口可见");
else
  Print(" MyMACD 窗口未发现或不可见");
```

- 获取窗口索引 WindowOnDropped()

int WindowOnDropped()

返回应用了智能交易、自定义指标和脚本的窗口索引。只有智能交易、自定义指标或脚本被鼠标拖到窗口上，这个值才是有效的。

注：对于那些被初始化函数调用的自定义指标 (通过 init()函数调用)，此索引没有被定义。

返回的索引就是运行自定义指标的窗口编号(0-图表主菜单，指标子窗口从 1 开始编号)。在自定义指标运行期间，它可以自己创建新的子窗口，并且这个子窗口的编号和那些真正用鼠标把指标拖过去应用的窗口是不同的。

参见 WindowXOnDropped(), WindowYOnDropped()。

示例:

```
if(WindowOnDropped()!=0)
  {
   Print("指标'MyIndicator'必须被应用到主图表窗口!");
   return(false);
  }
```

- 获取窗口纵坐标刻度最大值 WindowPriceMax()

double WindowPriceMax(void index)

返回当前图表指定的子窗口纵坐标刻度的最大值(0-图表主窗口，指标子窗口编号从 1 开始)。如果子窗口索引没有指定，将返回主图表窗口价格刻度的最大值。

参见 WindowPriceMin(), WindowFirstVisibleBar(), WindowBarsPerChart()。

参数:

index - 图表子窗口索引 (0 -主图表窗口)。

示例:

```
double   top=WindowPriceMax();
double   bottom=WindowPriceMin();
datetime left=Time[WindowFirstVisibleBar()];
int      right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom)
;
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();
```

- 获取窗口纵坐标刻度最小值 WindowPriceMin()

double WindowPriceMin(void index)

返回当前图表指定的子窗口纵坐标刻度的最小值(0-图表主窗口，指标子窗口编号从 1 开始)。如果子窗口索引没有指定，将返回主图表窗口价格刻度的最小值。

参见 WindowPriceMax(), WindowFirstVisibleBar(), WindowBarsPerChart()。

参数:

index - 图表子窗口索引 (0 - 主图表窗口)。

示例:

```
double   top=WindowPriceMax();
double   bottom=WindowPriceMin();
```

```
datetime left=Time[WindowFirstVisibleBar()];
int      right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom)
;
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();
```

- 获取下滑价格 WindowPriceOnDropped()

double WindowPriceOnDropped()

返回图表点的智能交易或脚本价格下滑价格。只有智能交易、自定义指标或脚本被鼠标拖到窗口上，这个值才是有效的。

注: 对于自定义指标，这个值是不确定的。

示例:

```
double   drop_price=WindowPriceOnDropped();
datetime drop_time=WindowTimeOnDropped();
//---- 可能未指定 (zero)
if(drop_time>0)
  {
   ObjectCreate("价格下滑水平", OBJ_HLINE, 0, drop_price);
   ObjectCreate("下滑时间", OBJ_VLINE, 0, drop_time);
  }
```

- 重绘当前图表 WindowRedraw()

Void WindowRedraw()

强制重画当前图表。在货币对属性改变之后，通常会应用本函数。

示例:

```
//----设置货币对新属性
ObjectMove(object_name1, 0, Time[index], price);
ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
ObjectSet(line_name, OBJPROP_TIME2, time2);
```

```
  ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
  //---- 现在重画
  WindowRedraw();
```

- 窗口图表快照 WindowScreenShot()

```
bool                string filename, int size_x, int size_y,
WindowScreenShot(   void start_bar, void chart_scale, void chart_mode)
```

以 GIF 文件形式保存当前图表图像。如果失败，返回 FALSE。想要得到详细的错误信息，请调用 GetLastError() 函数。

屏幕快照保存在 terminal_dir\experts\files (在测试情况下，文件放在 terminal_dir\tester\files 目录) 目录或其子目录中。

参数:

filename	-	保存屏幕快照的文件名。
size_x	-	屏幕宽度(像素)。
size_y	-	屏幕高度(像素)。
start_bar	-	屏幕快照中第一个可见柱子。如果价格值设定为 0 ，当前第一个可见柱子将被除去。如果价格值未设置或为负值，图表尾部图像将会生成。
chart_scale	-	屏幕快照图表的水平刻度。范围可以从 0 到 5，如果没有值或设为负值，将直接使用当前图表刻度。
chart_mode	-	图表显示模式。可以采用下列值: CHART_BAR (0 是柱子的顺序), CHART_CANDLE (1 是蜡烛柱的顺序),CHART_LINE (2 是收盘价线)。如果没有值或设为负值，图表会以当前模式显示。

示例:

```
  int lasterror=0;
  //----测试模式下平单
  if(IsTesting() && ExtTradesCounter<TradesTotal())
    {
     //---- 使 WindowScreenShot 进行深入检测

if(!WindowScreenShot("shots\\tester"+ExtShotsCounter+".gif",640,48
0))
        lasterror=GetLastError();
     else ExtShotsCounter++;
     ExtTradesCounter=TradesTotal();
```

```
    }
```

- 价格下滑时间 WindowTimeOnDropped()

datetime WindowTimeOnDropped()

返回图表指出的智能交易或脚本价格下滑时间部分。只有在智能交易、自定义指标或脚本应用鼠标的帮助下，绑定的值是准确的。

注: 对于自定义指标的价格值是不确定的。

返回应用了智能交易或脚本的图表点的下滑时间。只有智能交易、自定义指标或脚本被鼠标拖到窗口上，这个值才是有效的。

注: 对于自定义指标，这个值是不确定的。

示例:

```
double   drop_price=WindowPriceOnDropped();
datetime drop_time=WindowTimeOnDropped();
//---- 可能未指定 (zero)
if(drop_time>0)
  {
   ObjectCreate("Dropped price line", OBJ_HLINE, 0, drop_price);
   ObjectCreate("Dropped time line", OBJ_VLINE, 0, drop_time);
  }
```

- 获取指标窗口数 WindowsTotal()

int WindowsTotal()

返回在图表中指标窗口数(包括主图表)。

示例:

```
Print("指标窗口数 = ", WindowsTotal());
```

- X 轴下滑像素值 WindowXOnDropped()

int WindowXOnDropped()

在图表窗口的客户区域点，应用了智能交易或脚本，当鼠标拖动时，返回 x 轴上的下滑点像素值。当以映像点 X 轴智能交易或脚本下滑时，返回价格值。只有在智能交易或脚本应用鼠标技术("拖曳") 的情况下，这个值是准确的。

参见 WindowYOnDropped(), WindowOnDropped()

示例:

```
Print("智能交易下滑点 x=",WindowXOnDropped()," 
y=",WindowYOnDropped());
```

▪Y 轴下滑像素值 WindowYOnDropped()

int WindowYOnDropped()

在图表窗口的客户区域点，应用了智能交易或脚本，当鼠标拖动时，返回 y 轴上的下滑点像素值。当以映像点 y 轴智能交易或脚本下滑时，返回价格值。只有在智能交易或脚本应用鼠标技术("拖曳") 的情况下，这个值是准确的。

参见 WindowXOnDropped(), WindowPriceOnDropped(), WindowOnDropped()。

示例:

```
 Print("智能交易附加到窗口的点 x=",WindowXOnDropped()," 
y=",WindowYOnDropped());
```

过时函数

出于系统化的需要，在 MQL4 语言后继发展过程中，有些函数名称可能被重新命名或归入另一类，因此，函数旧名称可能无法继续使用，ME4 编辑器也不支持。由于编译器还能正确处理函数旧名称，旧函数暂时还能使用，不过，我们还是强烈推荐你使用新名称。

旧名称	新名称
BarsPerWindow	WindowBarsPerChart
ClientTerminalName	TerminalName
CurTime	TimeCurrent
CompanyName	TerminalCompany
FirstVisibleBar	WindowFirstVisibleBar
Highest	iHighest
HistoryTotal	OrdersHistoryTotal
LocalTime	TimeLocal
Lowest	iLowest
ObjectsRedraw	WindowRedraw,
PriceOnDropped	WindowPriceOnDropped

ScreenShot	WindowScreenShot
ServerAddress	AccountServer
TimeOnDropped	WindowTimeOnDropped

致谢

欢迎斧正，此处为您的辛勤劳动保留，期待着出现您的名字。