蓝目训练版策略编写指南

外接语言部分

本工具将支持外接 C++开发的 WINDOWS 动态链接库 DLL，以及 PYTHON 语言的调用。本次先介绍 DLL 插件的外挂。其中黑色字体部分为 dll 原理说明，红色部分为用户需要编写的内容的说明。

1.函数命名约定

指标宏语言(script)编写，为了区分内置静态函数和外部动态加载函数，对函数命名做了一些约定：

- 内置函数直接以函数名称开头，比如 SIN, MAX,等。
- 外挂 DLL 函数，以关键词“DLL_”和文件名（文件名称限定字幕数字，其它字符不支持）开头，后跟”@”符号与函数，不区分大小写， 比如”dll_file@fourier”，表示 dll 插件 file 文件中的函数 fourier。
- 外挂 PY 语言（需安装完整的 PYTHON 环境），以文件名（文件名称限定字幕数字，其它字符不支持）和关键词“PY_”开头，区分大小写， 比如”py_func@pyfile”，表示启动运行 python 的 pyfile.py 这个文件，并运行 func 这个函数。

2，如何使用 VISUAL C++编写 DLL 的规范

有关 DLL 的编写常规规范，本文不做说明。这里只介绍一下如何定义外挂函数的接口规范，以及初始化、参数传递等方法，并且给除了基于 visual c++的工程模板和一些函数编写例子。

DLL 程序设计的时候，必须提供如下 5 个接口函数：

■ 初始化函数，此函数负责算法函数的名称和地址映射。

```
typedef bool (*i_init)();
```

■ 单变量函数，输入和输出是一个变量而非数组或者矩阵。该函数暂时保留未用。

```
typedef void (*i_dynamic_func_a)(float _arg_input, float &_arg_output);
```

■ 单参数函数，输入和输出都是一个一维数组，该函数可能是使用最多的一种。

```
typedef void (*i_dynamic_func_b)(std::deque<float> _arg_input, std::deque<float> &_arg_output);
```

■ 多参数函数，输入是多个参数，输出是一个一维数组。

```
typedef void (*i_dynamic_func_c)(std::deque< std::deque<float> > _arg_input, std::deque<float> &_arg_output);
```

■ 矩阵函数，输入是多个参数，输出亦是多个参数。也即输入和输出均为矩阵。该函数暂时保留未用。

```
typedef void (*i_dynamic_func_d)(std::deque< std::deque<float> > _arg_input, std::deque< std::deque<float> > &_arg_output);
```

其中初始化函数已经声明好，剩下四个函数需要在 Algorithms.h 中声明，示例如下：

```
class CAlgorithms
{
public:
    CAlgorithms(void);
    ~CAlgorithms(void);

    static void Example01(float _arg_input, float &_arg_output);
    static void Example02(std::deque<float> _arg_input, std::deque<float> &_arg_output);
    static void Example03(std::deque< std::deque<float> > _arg_input,  std::deque<float> &_arg_output);
    static void Example04(std::deque< std::deque<float> > _arg_input, std::deque< std::deque<float> > &_arg_output);
```

函数的命名规范为 函数名称+(1,2,3,4)，分别对应四个函数。

函数具体功能的实现需要写在 Algorithms.cpp 中

脚本语言对外部函数的调用，最终底层实现了一个 C++的 DLL 接口函数的调用，这里做一个对应关系说明：

■ SCRIP 和 DLL 之间在 单参数模式调用下的对应关系如下：

SCRIPT 语句：A:=DLL_FILE1@FUNC01(CLOSE);

DLL 接口函数如下：

```
dynamic_func_b("DLL_FILE1@FUNC01",
                INPUT,
                OUTPUT);
```

参数分别为 函数名称， 以 INPUT 为输入参数，返回结果在 OUTPUT，最后通过脚本解释器底层把 OUTPUT 的值赋给变量 A 。

dynamic_func_b 的 c++函数原型如下：

```
typedef void (*i_dynamic_func_b)(std::deque<float> _arg_input,
std::deque<float> &_arg_output);
```

■ SCRIP 和 DLL 之间在 多参数模式调用下的对应关系如下：

SCRIPT 语句：A:=DLL_FILE1@FUNC01(ARG1,ARG2,...);

DLL 接口函数如下：

```
dynamic_func_c("DLL_FILE1@FUNC01",
                std::deque<INPUT>,
```

```
OUTPUT);
```

参数分别为 函数名称， std::deque<input>是一个二维数组参数，返回结果在 OUTPUT，最后通过脚本解释器底层把 OUTPUT 的值赋给变量 A .

dynamic_func_c 的 c++函数原型如下：

```
typedef void (*i_dynamic_func_c)(std::deque< std::deque<float> > _arg_input, std::deque<float> &_arg_output);
```

另外，函数增加两个初始化函数 `bool dynamic_init();` 其主要作用是创建一个函数映射表，即每个函数的名称和其多个重载函数的地址映射。C++函数原型和指针极其映射表的定义结构如下：

```
typedef bool (*i_init)();
typedef void (*i_dynamic_func_a)(float _arg_input, float &_arg_output);
typedef void (*i_dynamic_func_b)(std::deque<float> _arg_input, std::deque<float> &_arg_output);
typedef void (*i_dynamic_func_c)(std::deque< std::deque<float> > _arg_input, std::deque<float> &_arg_output);
typedef void (*i_dynamic_func_d)(std::deque< std::deque<float> > _arg_input, std::deque< std::deque<float> > &_arg_output);

typedef struct
{
    i_dynamic_func_a dynamic_func_a_;
    i_dynamic_func_b dynamic_func_b_;
    i_dynamic_func_c dynamic_func_c_;
}tag_FUNC_LIST;
```

为了完成该映射，需要在 algo_plugin.cpp 中对函数进行初始化

函数初始化的编写实例如下：

```
bool dynamic_init()
{
    tag_FUNC_LIST _func_list;

    _func_list.dynamic_func_a_ = (i_dynamic_func_a)CAlgorithms::Example01;
    _func_list.dynamic_func_b_ = (i_dynamic_func_b)CAlgorithms::Example02;
    _func_list.dynamic_func_c_ = (i_dynamic_func_c)CAlgorithms::Example03;
```

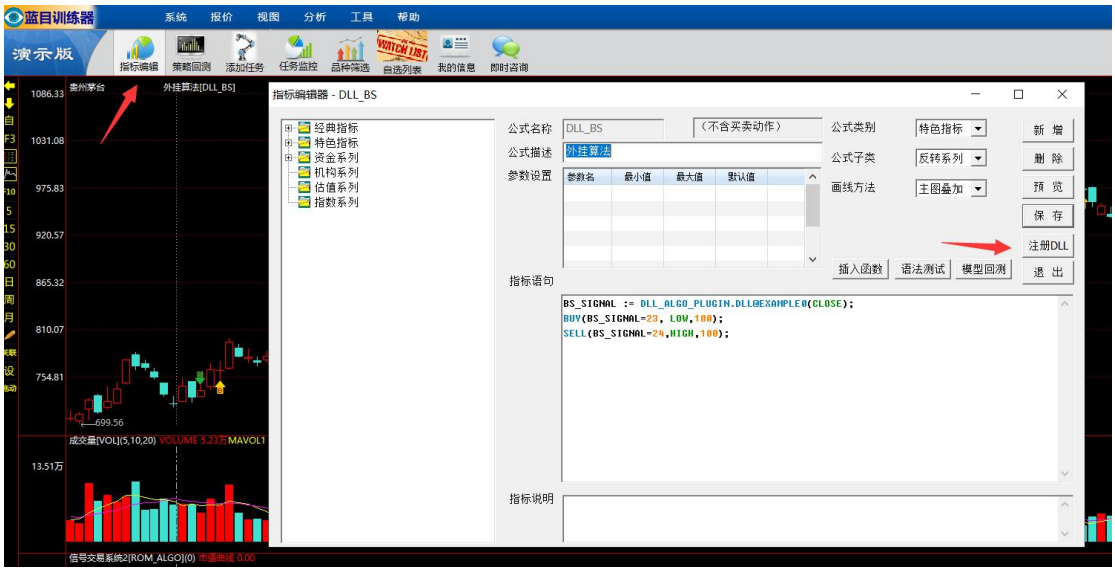
```
        g_map_functions.insert(std::make_pair("EXAMPLE0",_func_list));

        _func_list.dynamic_func_a_ = (i_dynamic_func_a)CAlgorithms::Example11;
        _func_list.dynamic_func_b_ = (i_dynamic_func_b)CAlgorithms::Example12;
        _func_list.dynamic_func_c_ = (i_dynamic_func_c)CAlgorithms::Example13;
        g_map_functions.insert(std::make_pair("EXAMPLE1",_func_list));

        // To do, add more functions in the following ......
        //...

        return true;
    }
```

注意 g_map_functions.insert 语句中的函数名称必须全部大写，否则无法识别

最后，用 visual c++编写的 DLL 工程示例规范，详见 algo_plugin。

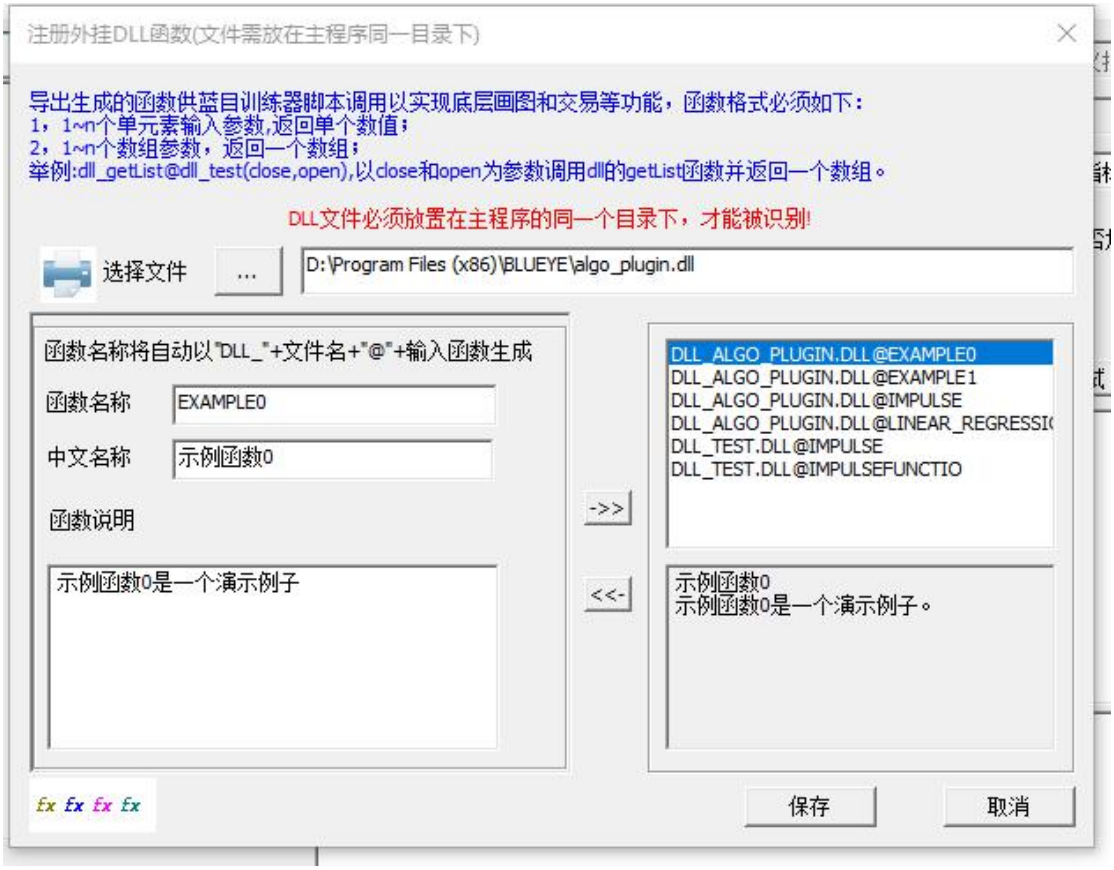
3. DLL 插件的注册

在策略编辑的窗口，使用“注册 DLL”按钮，如下图所示：



点击“注册 DLL”后，进入如下界面。

如上所示，先选择导入一个已经编译测试好好了的 DLL 文件，导入后如果该文件是一个新的则需要注册全部函数，如果已经注册过，则会自动把原来注册的函数全部列出来。然后对该 DLL 的函数进行增删维护（如果要修改，可以先删除再增加）。如下图所示：



用户添加的函数名称，系统会自动增加前缀，格式为“DLL_” + 文件名 + “@” 函数名，比如刚刚添加了一个名称为 EXAMPLE0 的函数，则系统生成的函数全名为：DLL_ALGO_PLUGIN.DLL@EXAMPLE0，以后在脚本语言中如果需要使用该函数，则需要书写该全名，这样编译器在编译的时候会自动搜索导向到指定库文件名称的指定函数。

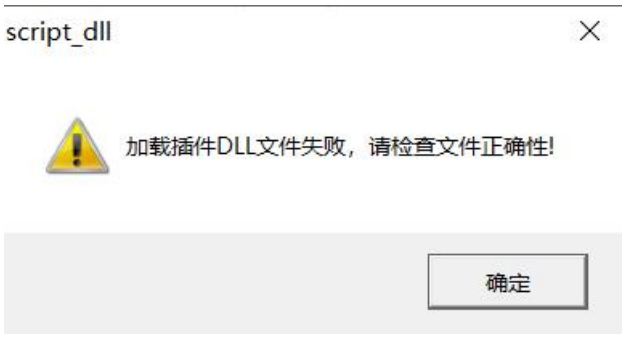
注意事项：

1、函数名称必须和 dll 中 g_map_functions.insert 语句中插入的函数名称相同，否则无法识别

2、函数名称不需要加(1,2,3,4)，编译器会根据输入输出来自动选择函数

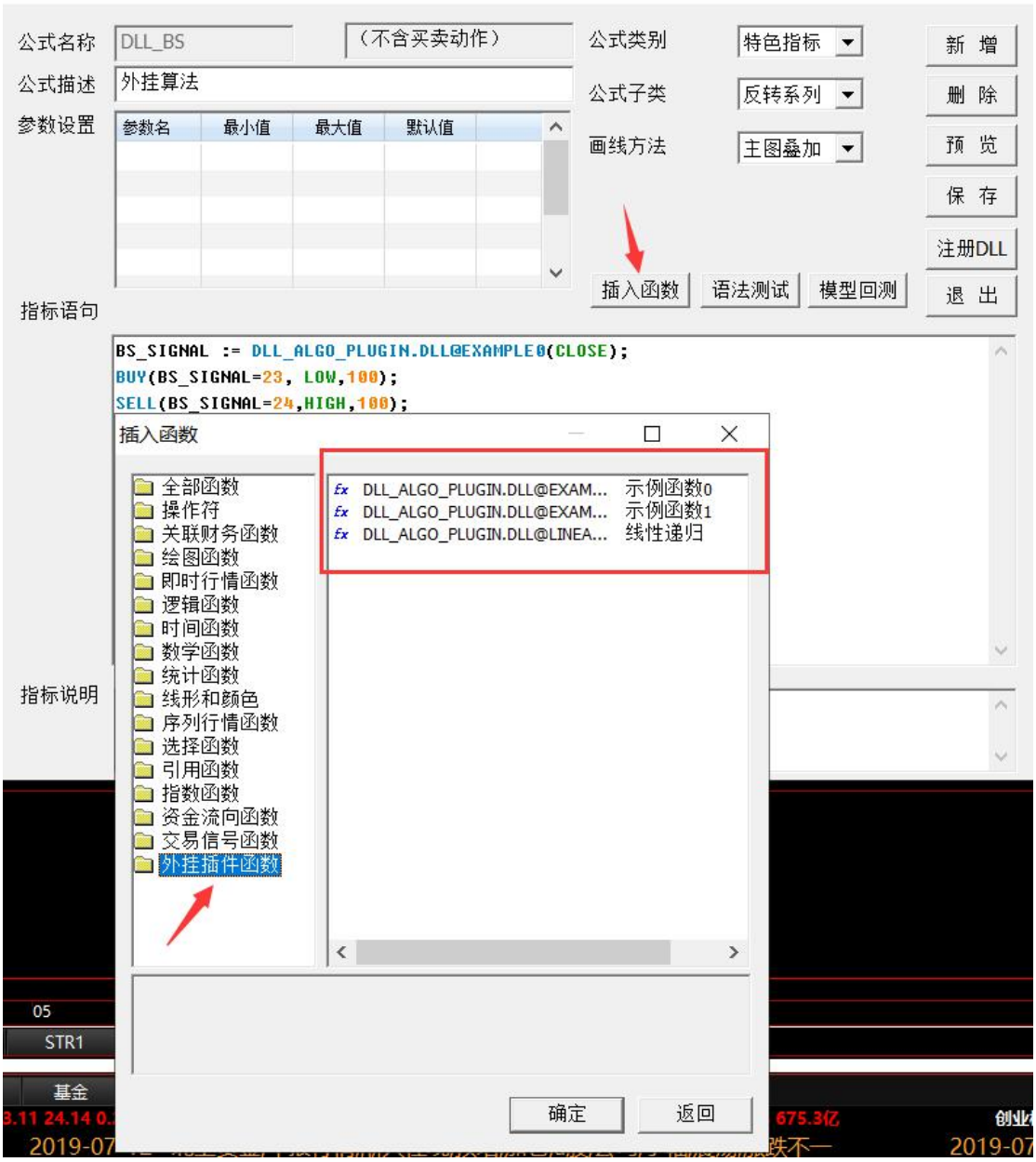
3、Dll 必须位于主程序 exe 同一目录下

注意，如果导入的文件是一个非法的 DLL（接口函数不存在），则会出现如下提示报错，不能通过。

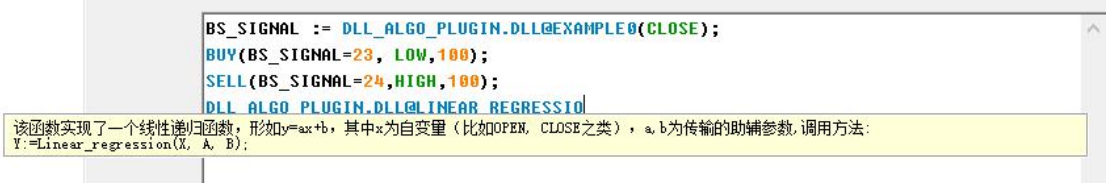


添加函数完成以后，点击“保存”按钮，即可实现了注册。

这时候再点击插入函数页面，则会在“外挂插件函数”中看见刚刚添加的这些函数。



插件注册完成后，需要重新启动软件，才能生效，这时候打开指标编辑器，点击插入刚刚增加的函数，则会出现颜色变蓝，以及使用说明的浮动信息提示，如下图所示：



4.Python 语言的对接和使用

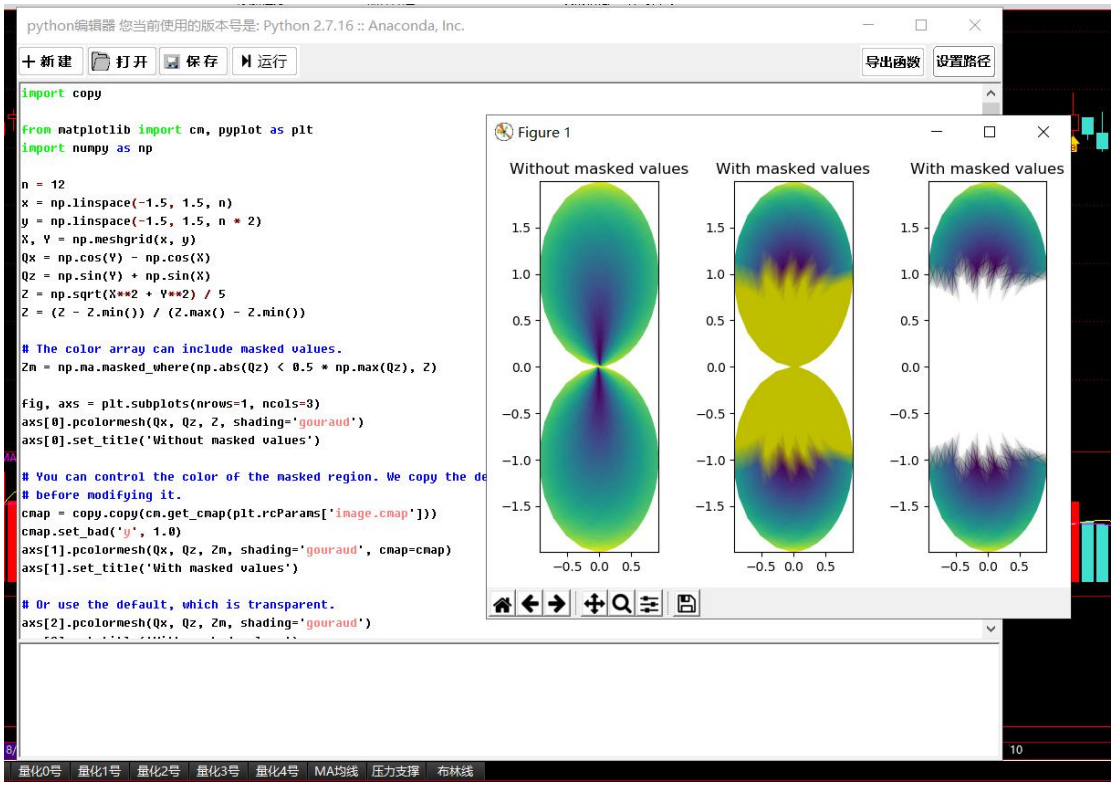
Python 作为近年来流行热门和使用量增长最快的语言，特别是在人工智能和量化交易这两个领域内被广泛使用。

“蓝目训练器”支持 python 语言的编辑与运行环境，并且支持调用 python 的标准化函数的调用。

python 编辑器的启动在如下图的第二个按钮。

点击后出现 python 窗口。

可以通过“新建”，创建新的空白窗口输入或者从其他地方粘贴 PYTHON 代码进来。也可以打开一个 python 文件（默认后缀名.py），打开后自动运行，也可以点击“运行”按钮进行运行。



这里主要介绍如何编写 python 函数并且导出供行情分析指标显示以及产生交易信号（买卖）。

首先，作为分析软件的 script 语言调用的函数，都定义了标准的输入和输出接口，这里的 python 函数的输入输出，和 dll 插件使用统一的接口规范，调用的时候的唯一区别是通过函数命名规范来区分是调用了一个 dll 插件还是 python 函数，前面已经提及，这里在重申如下：

■外挂 DLL 函数，以关键词“DLL_”和文件名（文件名称限定字幕数字，其它字符不支持）开头，后跟”@”符号与函数，不区分大小写， 比如”dll_file@fourier”，表示 dll 插件 file 文件中的函数 fourier。

■外挂 PY 语言（需安装完整的 PYTHON 环境），以文件名（文件名称限定字幕数字，其它字符不支持）和关键词“PY_”开头，区分大小写，比如”py_func@pyfile”，表示启动运行 python 的 pyfile.py 这个文件，并运行 func 这个函数。

其次，python 函数的定义规范，必须按照单参数函数、多参数函数和矩阵函数的输入输出定义方法来编写 python 函数，由于目前系统仅需要一个数组的输入和输出函数，以及多数组输入单数组数组这两种标准（其它标准后续陆续介绍），因此把两种形式的 python 函数编写示例介绍如下：

a，一个数组输入、一个数组输出的 python 示例：

```
def testList(input):
    t = [float(x) for x in input]
    df = ta.MA(np.array(t),timeperiod=5)
    return df.tolist()
```

b,多个数组输入、一个数组输出的 python 示例：

```
def testList2(input1,input2):
    a = np.array(input1)
    b = np.array(input2)
    c = a+b
    output = []
    for x in c:
        output.append(x)
    return output
```

再次，script 调用 python 可使用如下方式：

```
A:PY_ROM@TESTLIST2(high,low);
```

最后，本系统提供了一个 python 函数的指标显示，可以使用 pytest 调出来显示，如下图所示：

另外，系统提供了一个 python 的导出函数编写例子文件，详见 rom.py(在应用程序文件夹 python 目录下)，可以使用 python 编辑器打开显示，如下图所示：