

盈佳期货交易终端用户手册

目 录

第 1 章 概述.....	3
1.1 主要特点.....	3
1.2 人工交易功能.....	5
1.2.1 结算单查询及确认.....	5
1.2.2 行情订阅.....	5
1.2.3 交易.....	6
1.2.4 银期转账.....	6
1.3 程序化交易功能.....	6
1.3.1 交易策略参数配置.....	7
1.3.2 交易策略执行.....	8
1.3.3 交易策略执行日志.....	9
1.3.4 交易策略开发.....	10
第 2 章 交易策略开发环境准备.....	12
2.1 VC6 安装.....	12
2.2 盈佳期货交易终端安装.....	12
第 3 章 交易策略开发实例.....	12
3.1 创建交易策略工程.....	13
3.2 编写交易策略配置文件.....	18
3.3 编写交易策略代码.....	20
3.3.1 获取交易策略参数.....	20
3.3.2 交易信号检测.....	22
3.3.3 止赢止损判定.....	25
3.3.4 资金管理逻辑.....	26
3.4 交易策略部署.....	27
3.5 交易策略执行过程.....	27
3.6 注意事项.....	28
第 4 章 交易策略接口说明.....	28
4.1 IPolicy 接口说明.....	28
4.1.1 成员变量.....	28
4.1.2 成员方法.....	29
4.2 IOrderAPI 接口说明.....	31
4.2.1 交易类接口.....	31
4.2.2 报单管理类接口:	31
4.2.3 行情类接口.....	32
4.2.4 仓位类接口.....	32
4.2.5 资金类接口.....	33
4.2.6 日志类接口.....	33
第 5 章 交易策略回归测试工具.....	33
5.1 行情收集器.....	33
5.2 仿真测试终端.....	35

第 6 章 COM 接口	36
6.1 COM 包构成.....	37
6.2 COM 注册	37
6.3 COM 接口方法.....	37
6.4 COM 触发事件.....	42
6.5 COM 使用示例.....	43
6.5.1 VB6.0 示例	43
6.5.2 VC6.0 示例	45
第 7 章 盈佳服务.....	50

第1章 概述

1.1 主要特点

盈佳期货交易终端是基于综合交易平台接口（CTP-API）开发的标准期货交易终端。综合交易平台是由上海信息技术有限公司推出的新一代期货柜台交易系统，该平台对外公开、免费发布 CTP-API 供第三方软件商及程序化交易投资者接入期货柜台交易系统进行期货交易。

盈佳期货交易终端是综合交易平台众多标准终端之一，该终端不仅给用户提供简单快捷的人工交易界面，同时还提供了高效的程序化交易开发及运行框架，其主要特点如下：

1. 用户可以订阅综合交易平台发布的期货市场行情，并设定自定义合约组以方便用户查看特定合约的行情。
2. 资金、持仓、报单及成交信息实时刷新，用户关注的信息能及时展现。
3. 快速开仓、平仓及快捷反手，并且可以同时进行多个合约的开仓和平仓。
4. 对 CTP-API 进行了期货业务层面的封装，用户可以基于盈佳的接口编写程序化交易策略，而不用关注 CTP-API 的底层细节。
5. 提供程序化交易运行框架，用户可以自由启动、暂停及停止程序化交易策略。

6. 标准化的参数配置文件为程序化交易策略的参数化提供了极其便利的条件。
7. “盈佳行情收集器”为行情数据的收集和存储提供了便利的手段。
8. “盈佳交易策略模拟器”使用户可以通过回放历史行情数据，大大方便了用户的交易策略测试。
9. 盈佳期货交易终端还提供可供非 c++语言调用的 COM 接口，熟悉 VB、delphi 甚至 excel 编程的用户也可以基于该 COM 接口实现期货交易策略的开发。

图-1.1 是盈佳期货交易终端的主窗口界面。界面的上部为行情栏，可以进行任意的期货合约行情分组显示。紧接行情窗口之下是手工下单板，可以进行快速下单、预埋、快捷反手及条件单的操作。手工下单板下面是报单列表窗口，显示所有报单信息，同时还提供了“选中撤单”和“全部撤单”功能。下部是持仓、成交、持仓明细、资金账号、预埋单及交易策略信息窗口。

主窗口最下端的状态条，提供了最近一次操作的提示及报错信息显示、行情及交易前置的连接信息以及可用资金、可取资金、风险度、持仓盈亏等用户资金账户信息。

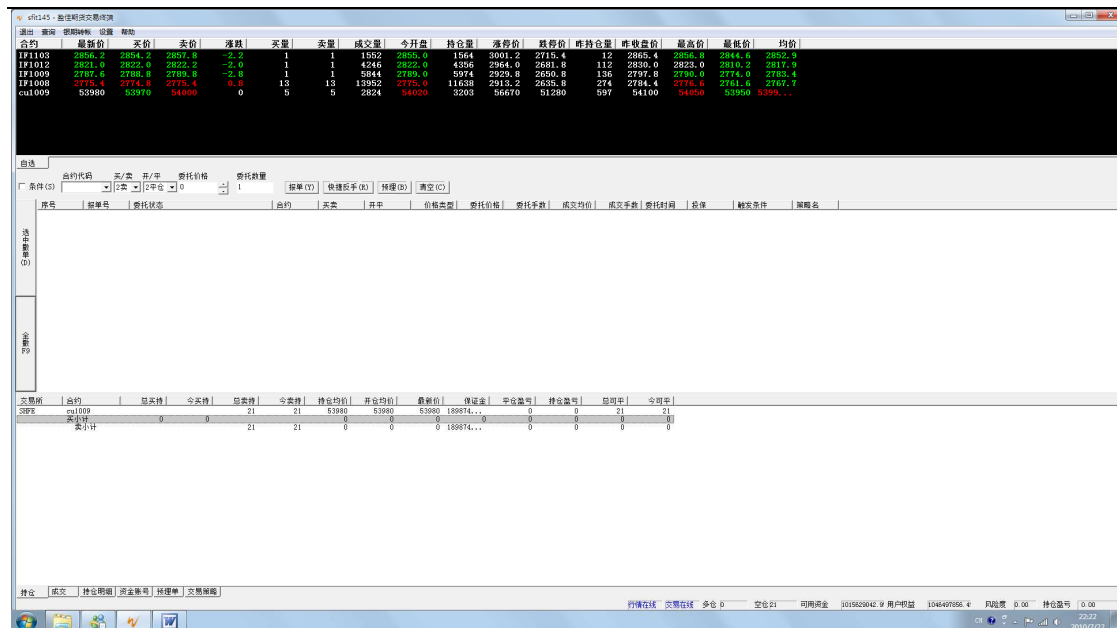


图-1.1

1.2 人工交易功能

盈佳期货交易终端提供了丰富、简便、快捷的人工交易界面，用户可以使用人工交易界面查看结算单、资金账户、持仓、报单、成交信息，进行报单、撤单及银期转账等交易操作。

1.2.1 结算单查询及确认

用户每个交易日使用盈佳期货交易终端进行交易交易之前，必须确认一次上一交易日的结算账单，否则将不可以进行交易操作。用户每个交易日第一次登录盈佳期货交易终端时将弹出结算单确认窗口，用户点击确认按钮进行确认，当天再次登录时将不再弹出该窗口。后续需要查询结算单可以使用“查询”菜单项下的“结算单...”命令。

1.2.2 行情订阅

行情窗口提供了合约组及合约组所属合约的管理功能，可以分组

显示用户订阅的期货合约行情。

1.2.3 交易

盈佳期货交易终端在多个界面提供交易指令触发功能。除了手工下单板对交易指令的支持以及“报单列表”窗口对撤单指令的支持。在行情窗口，用户可以鼠标左击“买价”、“买量”或“卖价”、“卖量”将在手工下单板填入报单信息，双击则直接发出报单。

在持仓窗口提供了双击现有持仓触发平仓的功能。另外，在“设置”菜单项下，还提供了丰富的“下单参数”、“合约手数”及“合约快捷键”的设置功能。

对于用户人工交易指令以及程序化交易策略的执行结果，盈佳期货交易终端提供了高度实时性的查询窗口，能及时反映用户持仓、成交及资金账号的最新状态。

1.2.4 银期转账

作为综合交易平台的标准终端，盈佳期货交易终端还提供了银期转账功能。用户可以在资金账户和各银行签约的银行账户间进行资金划转、各签约银行账户余额的查询以及当日转账明细的查询。

1.3 程序化交易功能

用户基于盈佳期货交易终端提供的程序化交易接口，使用 C++ 语言编写交易策略并形成动态链接库文件（dll 文件），随交易策略参数配置文件一起放置到盈佳期货交易终端的可执行文件所在目录，在下

次登录盈佳期货交易终端时，盈佳期货交易终端的程序化交易框架将能够方便的管理交易策略的启动、暂停、停止以及交易策略个性化参数的配置。

1.3.1 交易策略参数配置

交易策略部署到终端后，重新启动终端在界面中的“交易策略”中可以看到新部署的策略，如图-1.2 所示：



图-1.2

在每个策略后面都有两列，分别是“配置”和“启动”，选择“配置”，将弹出如图-1.3 中对话框，进行设置这个策略的全局参数，所适用的合约，以及每个合约级别的参数：

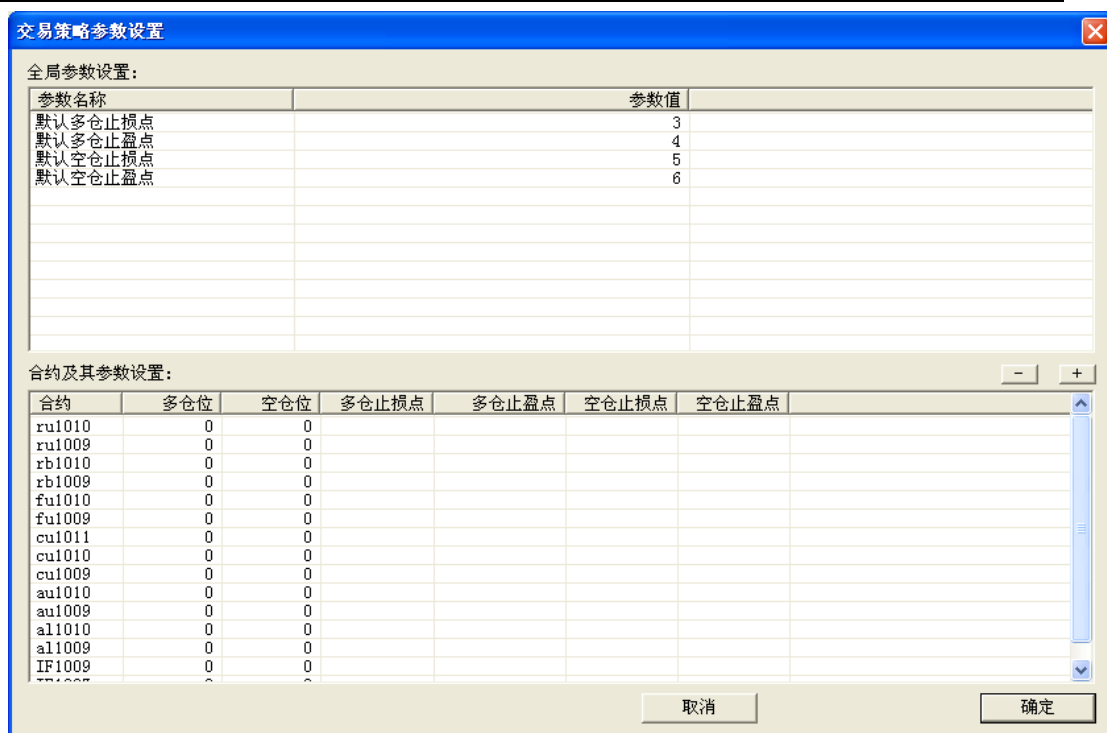


图-1.3

上半部分的列表位当前这个策略的全局交易参数，鼠标单击“参数值”列的单元格，可以编辑这些参数值；下半部分的列表为这个策略所适用的合约，以及跟每个合约相关的参数值，其中“多仓位”和“空仓位”是由框架维护的每个策略所开的该合约的多头仓位和空头仓位，这些仓位与其他策略所开的该合约的仓位相互独立。鼠标单击其中的单元格可以进行各参数值（包括多仓位和控仓位）的修改。修改了参数值之后，需要选择“确定”按钮，修改后的参数值会重新被设置到交易策略中。

1.3.2 交易策略执行

如果要启动某一个策略，在该策略后面的“启动”文本上鼠标单击，会弹出启动策略确认对话框，确定后将启动该策略；如果成功启

动，则该策略的状态被设置为“运行”，同时“启动”变为“停止”，并且在“停止”的右边会显示“暂停”；如果选择“停止”，则该交易策略会停止运行，该对象占用的内存会被释放，不再处理行情通知，成交回报，报单回报等信息；如果选择“暂停”，则停止接受新的行情，但是该策略所发出报单的状态变更，成交回报等信息会继续处理，用户单击“暂停”后，策略的状态变为“暂停中”，同时原来的“暂停”变为“继续”，可以单击“继续”让该策略继续运行，即继续处理行情通知。

1.3.3 交易策略执行日志

交易策略代码中利用框架提供的日志接口写日志到文件的同时，可以根据配置被同时显示到交易策略列表右边的日志列表窗口中：如果当前所发出的日志的级别大于或等于(FATAL 级别最高)名称为screen 的 Logger 的级别时，就会被同时显示在日志列表窗口中，否则只会写在文件中。

Screen 的日志级别在安装目录下的 log.properties 文件中定义，如下所示（其中黑体部分定义 screen Logger 的级别）：

```
# Define the root logger
log4cplus.rootLogger=TRACE,main

log4cplus.logger.screen=INFO,window
log4cplus.additivity.screen=FALSE

# Define a file appender named "main"
log4cplus.appender.main=log4cplus::RollingFileAppender
log4cplus.appender.main.MaxFileSize=200KB
log4cplus.appender.main.File=./logs/policy.log
log4cplus.appender.main.MaxBackupIndex=3
```

```
log4cplus.appender.main.layout=log4cplus::PatternLayout
log4cplus.appender.main.layout.ConversionPattern=%-5p-[%t][%D{%H:%M:%S %Q}]%m
%n

# Define a file appender named screen
log4cplus.appender.window=log4cplus::RollingFileAppender
log4cplus.appender.window.MaxFileSize=200KB
log4cplus.appender.window.File=./logs/screen.log
log4cplus.appender.window.MaxBackupIndex=3
log4cplus.appender.window.layout=log4cplus::PatternLayout
log4cplus.appender.window.layout.ConversionPattern=%-5p-[%t][%D{%H:%M:%S %Q}]%
m%n
```

1.3.4 交易策略开发

盈佳期货交易终端不仅提供了程序化交易策略的运行框架及众多可供用户直接使用的交易策略库，还提供了编制可在盈佳期货交易终端运行的交易策略的接口集。该接口集对综合交易平台 API 进行了必要的系统封装，使用户既可以摆脱对综合交易平台 API 系统底层逻辑的处理，又能充分享受到综合交易平台 API 接口无与伦比的速度。

盈佳期货交易终端提供的程序化交易策略接口集包括 IPolicy 和 IOrderAPI 接口。

每个盈佳期货交易策略类必须实现 IPolicy 接口，该接口提供了交易策略的启动、暂停及停止函数以供盈佳期货交易终端的框架对交易策略的运行进行管理。该接口还提供了行情通知、报单通知、成交通知等接口，以方便在交易策略收到以上信息时能进行相应的处理。对于每个交易策略，该接口还提供了全局参数及针对合约的参数设置函数，用户可以为每个交易策略定义不限数量的全局参数，和为不同的合约配置不同的参数值。在策略库(dll)及参数配置文件配置到盈佳期

货交易终端后，用户可以通过打开交易策略参数配置窗口，对个参数值进行可视化的配置管理。

IOrderAPI 是供交易策略类发送交易指令，查询合约、报单、成交、资金及持仓等信息的接口。

用户自定义的交易策略可以通过实现 IPolicy 接口对报单成功、失败及成交完成后进行相应的逻辑处理，对新收到行情数据进行分析。在产生满足交易策略既定条件的交易信号时，则调用 IOrderAPI 完成交易指令的发送。交易策略的启动、暂停、停止及参数的配置则完全可以交给盈佳期货交易终端的程序化交易框架完成。这样，用户不仅可以编制自己使用的交易策略，还可以编制可以公开发布和销售的交易策略库。

第2章 交易策略开发环境准备

2.1 VC6 安装

公开发布的盈佳期货交易终端支持使用 VC6 编制的交易策略动态链接库文件（dll 文件）。VC6 的安装请参考 microsoft 提供的帮助文件。

2.2 盈佳期货交易终端安装

运行盈佳期货交易终端安装包（winner_mock_setup.exe），根据安装提示完成盈佳期货交易终端的安装。安装过程中，用户需要确认盈佳期货交易终端的免责条款方可完成安装并进行交易。请用户仔细阅读免责条款，并自行决定是否使用盈佳期货交易终端以及承担使用盈佳期货交易终端产生的所有后果。

安装完成后，您将可以使用“盈佳期货交易终端”以及“交易策略仿真运行”和“期货行情收集”等工具软件。

第3章 交易策略开发实例

本章将通过编制一个简单的均值突破交易策略来介绍盈佳期货交易终端交易策略开发的完整过程。均值突破交易策略的详细逻辑描述如下：

变量定义：对选定的合约，定义最近收到的 n 次行情数据(最新价)

的均值为 $MV(n)$ ，标准差为 $Std(n)$ ，每次开仓交易手数为 v 手，开仓后的合约最新价最高值为 $max(n)$ ，最低值为 $min(n)$ 。

交易信号： 1) 最新价 $> MV(n) + Std(n)$ ，以卖一价买开仓 v 手；

2) 最新价 $< MV(n) - Std(n)$ ，以买一价卖开仓 v 手。

止盈止损： 使用移动止盈止损，最新价 $< 0.999 * max(n)$ 时，以买一价卖平仓了结全部多头头寸；最新价 $> 1.001 * min(n)$ ，以卖一价买平仓了结全部空头头寸。

资金管理： 可用资金大于动态权益的 50% 时才允许开立新的头寸。

3.1 创建交易策略工程

1. 打开 VC6，选择菜单：文件->新建，在弹出的对话框中选择“工程”->Win32 Dynamic-Link Library，然后输入工程名称和位置，最后确定。如图-1.4 所示：

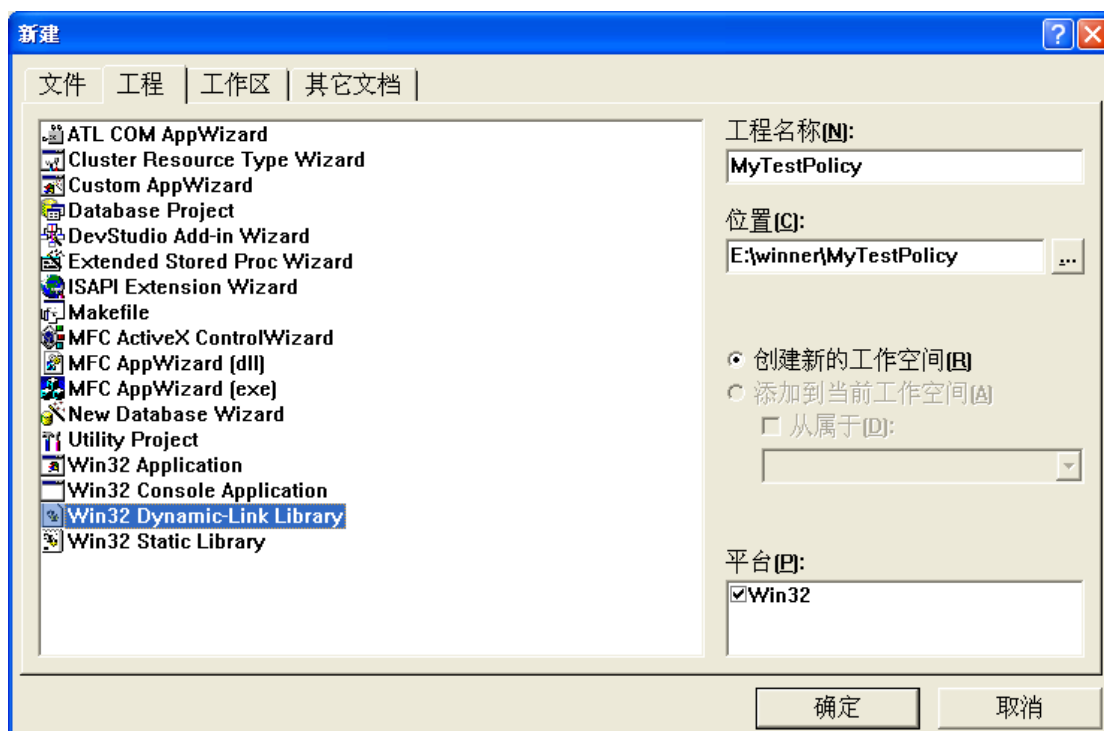


图-1.4

2. 确定后弹出如图-1.5 所示对话框,

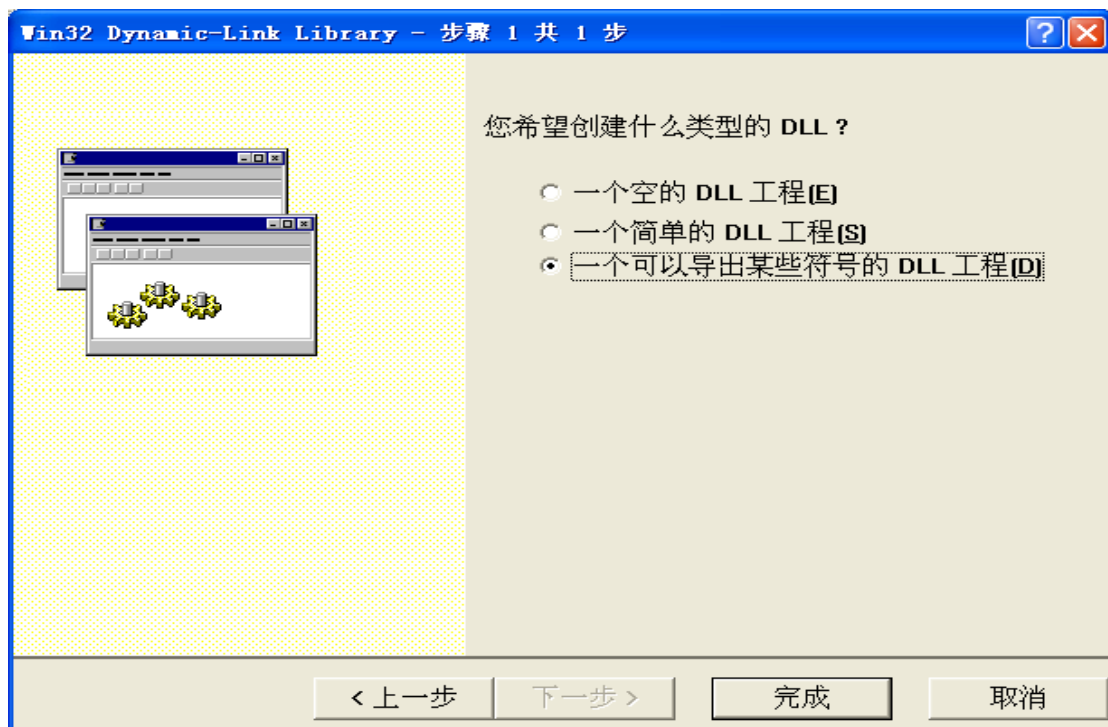


图-1.5

选择第三项：一个可以导出某些符号的 DLL 工程，然后选择“完成”，在弹出的对话框中再选择“确定”。

3. 打 开 StdAfx.h 文 件 ， 将 其 中 #define WIN32_LEAN_AND_MEAN 注释掉，如图-1.6 所示：

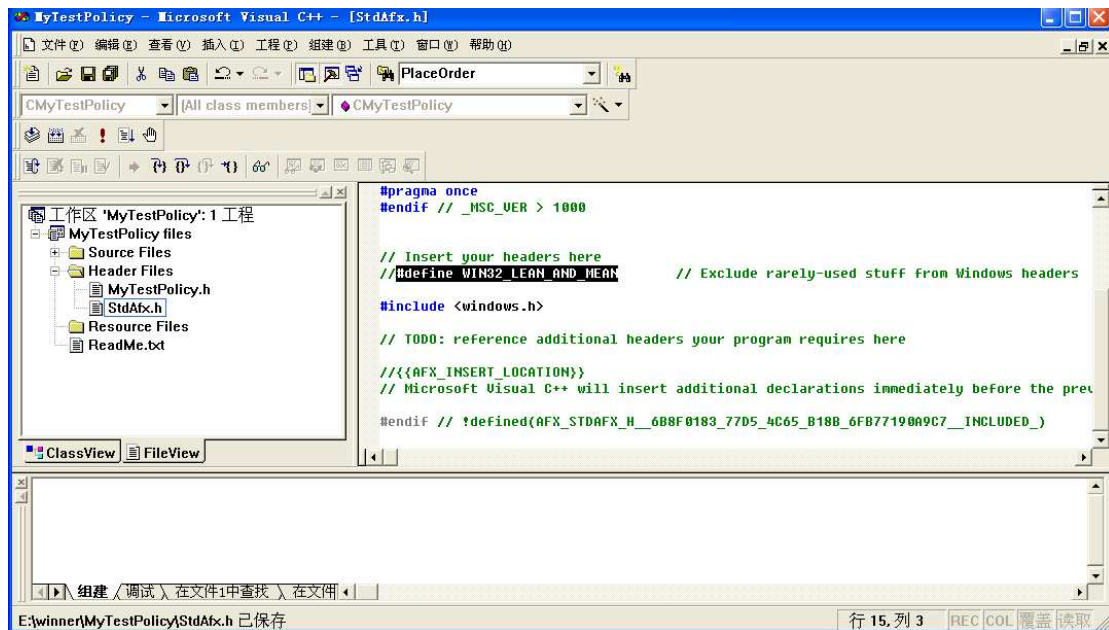


图-1.6

4. 选择菜单：“工具”->“选项”，在弹出的对话框中，选择“目录”，在”Include files”中增加 PolicyBase 项目的目录（盈佳交易终端的安装目录下会有这个子目录）；

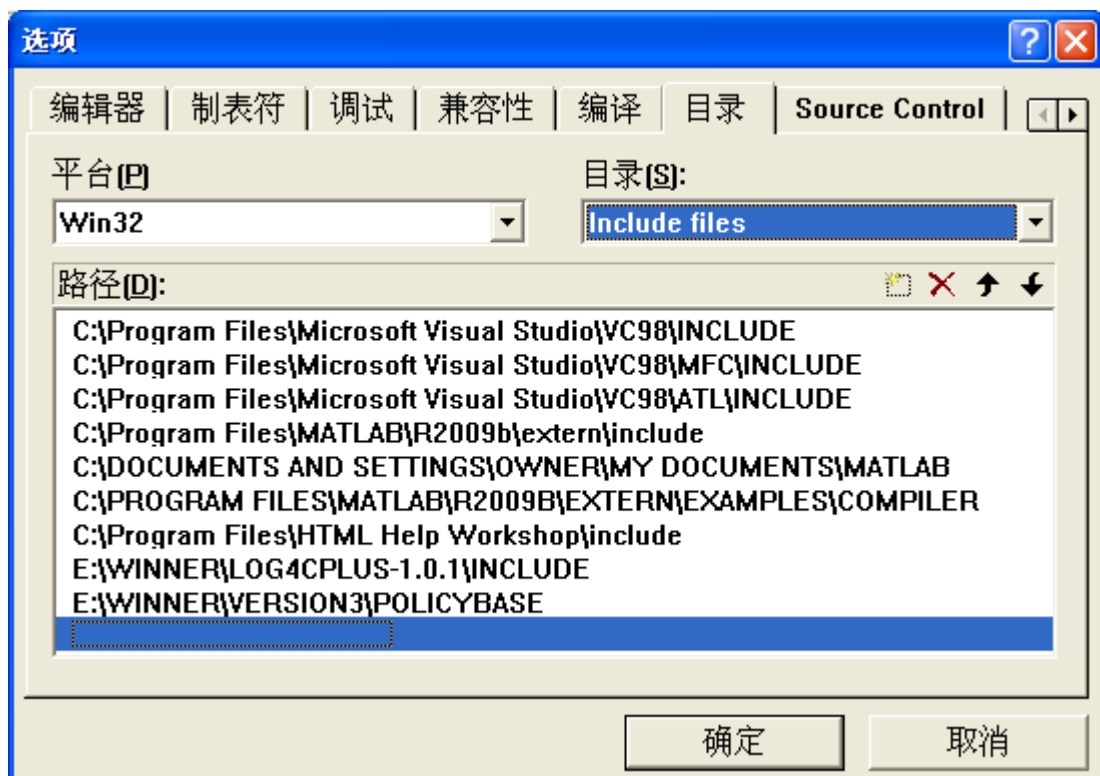


图-1.7

在 Library Files 中增加 PolicyBase.lib 所在的目录，缺省为盈佳交易终端框架的安装目录下。

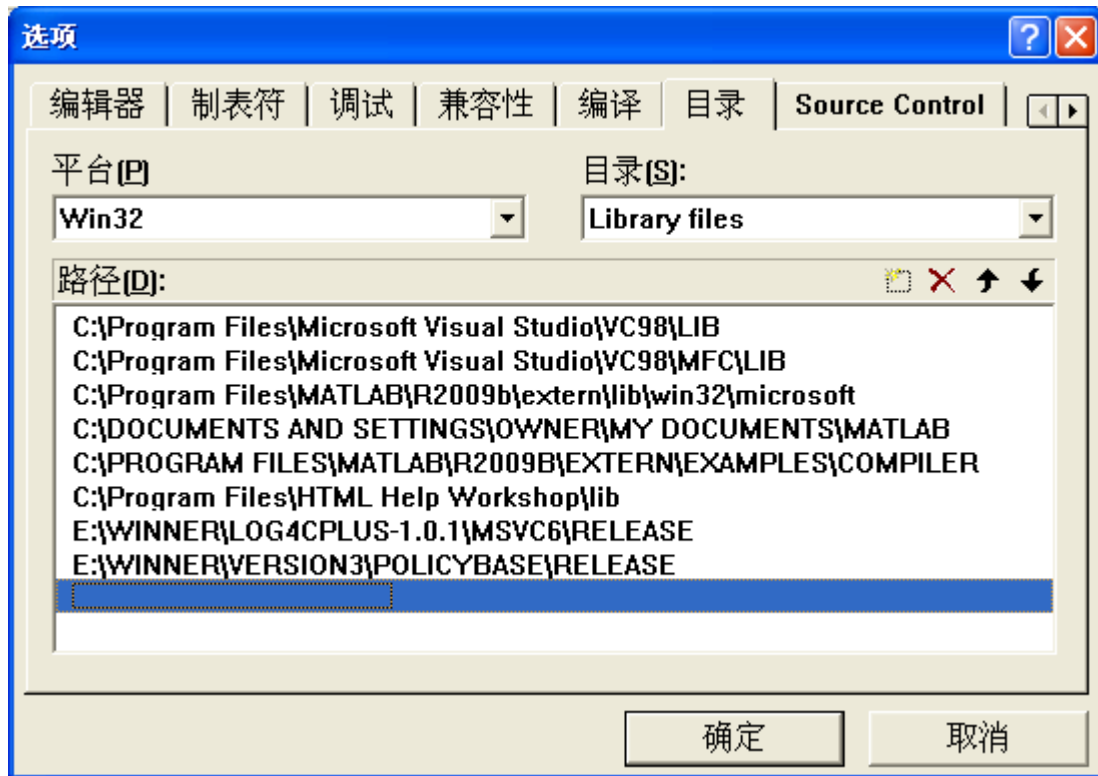


图-1.7

注意：这一步骤只需完成一次即可，其他策略的开发就可以忽略这一步骤了。

5. 选择 “工程” -> “设置”，然后在 “连接” (link) 中的对象/库模块中 增加 policybase.lib,如下图所示：

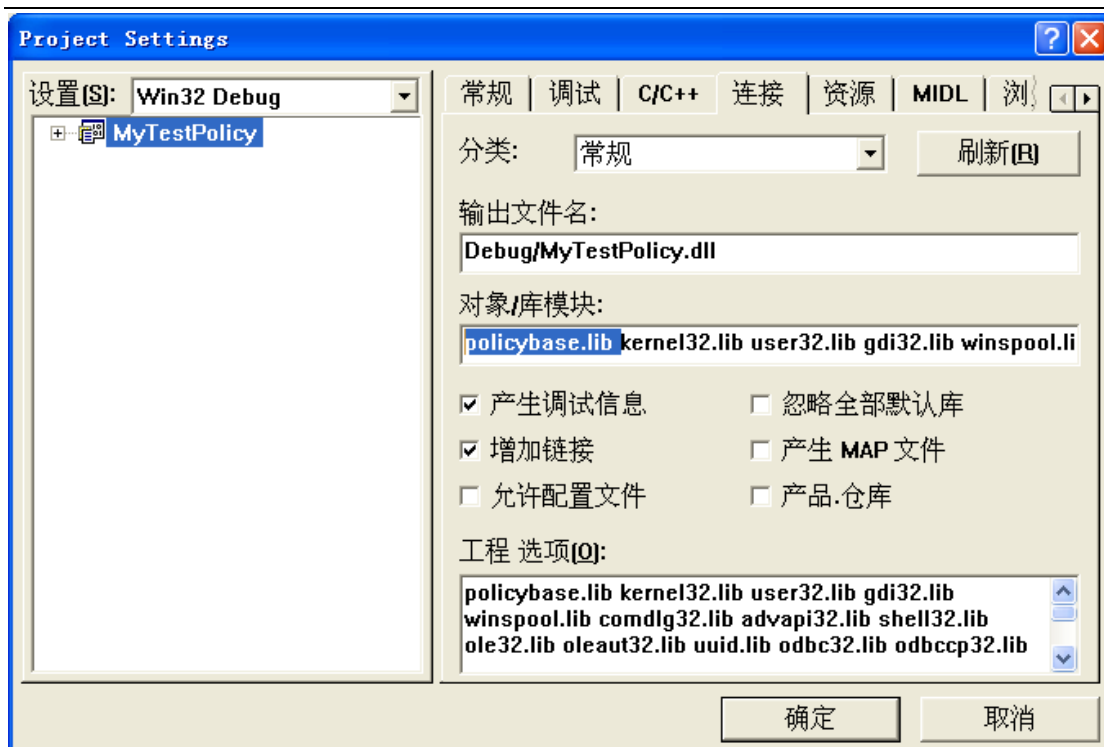


图-1.8

6. 打开 MyTestPolicy.h 文件，做如下修改：

- 1) 删除不必要的如下内容(包括 MyTestPolicy.cpp 中对应的内容)：

```
extern MYTESTPOLICY_API int nMyTestPolicy;
MYTESTPOLICY_API int fnMyTestPolicy(void);
```

- 2) 增加 #include “PolicyBase.h”, 并且让 CMyTestPolicy 从 IPolicy 继承下来（访问类型为 public）。

```
class MYTESTPOLICY_API CTestPolicy:public IPolicy
```

- 3) 增加如下内容

```
extern "C"{
    IPolicy *CreatePolicy(){
        return new CMyTestPolicy();
    }
}
```

至此，CMyTestPolicy.h 文件应该如下所示：

```
// The following ifdef block is the standard way of creating macros which make exporting
// from a DLL simpler. All files within this DLL are compiled with the
```

```
MYTESTPOLICY_EXPORTS
// symbol defined on the command line. this symbol should not be defined on any project
// that uses this DLL. This way any other project whose source files include this file see
// MYTESTPOLICY_API functions as being imported from a DLL, whereas this DLL sees
symbols
// defined with this macro as being exported.
#ifdef MYTESTPOLICY_EXPORTS
#define MYTESTPOLICY_API __declspec(dllexport)
#else
#define MYTESTPOLICY_API __declspec(dllimport)
#endif

#include "PolicyBase.h"

// This class is exported from the MyTestPolicy.dll
class MYTESTPOLICY_API CMyTestPolicy:public IPolicy {
public:
    CMyTestPolicy(void);

};

extern "C"{
    IPolicy *CreatePolicy(){
        return new CMyTestPolicy();
    }
}
```

3.2 编写交易策略配置文件

盈佳期货交易终端运行的每个交易策略，都需要一个后缀名为“.policy”，且符合 xml 格式的配置文件。均值突破交易策略配置文件的文件名为“MyTestPolicy.policy”。内容如下：

```
<policy>
  <name>均值突破交易策略</name>
  <description>最新价超过均值的正负一个标准差之外时时开仓。</description>
  <author>winner</author>
  <dll>MyTestPolicy.dll</dll>
  <globalParams>
    <param name="样本数据总数" value="30"/>
    <param name="每笔开仓手数" value="0"/>
    <param name="止损比例" value="0.001" />
  </globalParams>
</policy>
```

```

</globalParams>
<instruments>
  <code value="IF1105" longPos="0" shortPos="0" />
  <code value="cu1105" longPos="0" shortPos="0" />
  <code value="a1105" longPos="0" shortPos="0" />
</instruments>
<instrumentParams>
  <param name="样本数据总数">
    <instrument code=" IF1105" value="15" />
    <instrument code=" cu1105" value="10" />
    <instrument code=" a1105" value="20" />
  </param>
  <param name="每笔开仓手数">
    <instrument code=" IF1105" value="1" />
    <instrument code=" cu1105" value="1" />
    <instrument code=" a1105" value="10" />
  </param>
  <param name="止损比例">
    <instrument code=" IF1105" value="0.002" />
    <instrument code=" cu1105" value="0.003" />
    <instrument code=" a1105" value="0.005" />
  </param>
</instrumentParams>
</policy>

```

其中<name>中的内容为这个策略的名称，必须的字段，并且所有策略的名称必须唯一。<dll>节点下的内容是这个策略编译成的 DLL 全名称。<globalParams>下面为这个策略全局的参数，每个参数包括名称和当前值。

<instruments>下面为这个策略所适用的合约，可以包含多个，每个合约配置如下所示:<code value="IF1105" longPos="0" shortPos="0" />其中 longPos 和 shortPos 的值是该策略所开（或者说可利用的多仓位和空仓位。当该策略在开仓或者平仓时，这两个值会被更新。若策略运行之前已经存在对应合约的持仓，则需要将持仓数量赋予 longPos 和 shortPos。

在<instrumentParams>下面可以单独为每个合约设置与全局参数默认值不同的参数值。

3.3 编写交易策略代码

在交易策略参数配置文件“MyTestPolicy.policy”中，我们定义了交易策略需要的参数及参数值。盈佳期货交易终端提供了加载交易策略参数到变量的接口函数，将参数值读入变量后，我们就可以在交易策略运行过程中使用在策略配置文件中设置的各参数值。

3.3.1 获取交易策略参数

首先定义一个参数集结构体，包含所有我们即将用到的参数项。结构体定义在“MyTestPolicy.h”中“#include "PolicyBase.h"”的下一行。我们的交易策略参数集包含全局参数集以及与每个合约对应的参数集。因此，我们需要使用 std 提供的 map 数据结构来管理我们需要的参数集，定义的全局变量 m_params 即用来存储交易策略配置文件中设置的各参数值：

```
#include "PolicyBase.h"
#include <map>
#include <string>

struct CPARAMS{
    int nSampleSize;
    int nOpenVolume;
    double dStopRatio;
};
using namespace std;
map<string,CPARAMS> m_params;
```

在 CMyTestPolicy 类中实现 IPolicy 接口的 Start()函数，在该函数

中输出交易策略日志：

```
void CMyTestPolicy::Start()
{
    m_pOrderAPI->Log_Info(NULL,"均值突破策略已经启动！");
}
```

完成交易策略参数变量的定义之后，在 CMyTestPolicy 类中实现 IPolicy 接口的 SetGlobalStringParam()函数，该函数在交易策略启动后将会被盈佳期货交易终端的程序化交易框架调用，并传入交易策略配置文件中的全局参数值。我们在该函数的实现中，将传入的参数值存储在之前定义的变量中，以备交易策略运行时使用：

```
void CMyTestPolicy::SetGlobalStringParam(const char *pszCode, const char *pszValue)
{
    map<string, CPARAMS >::iterator it = m_params.find("global");
    if(it == NULL || it == m_params.end()){//未加载全局参数
        CPARAMS *pCPARAMS=new CPARAMS();
        it=m_params.insert(pair<string, CPARAMS >("global", pCPARAMS));
    }
    if(strcmp(pszCode, "样本数据总数")==0){
        it->second->nSampleSize = atof(pszValue);
    }else if(strcmp(pszCode, " 每笔开仓手数")==0){
        it->second->nOpenVolume= atof(pszValue);
    }else if(strcmp(pszCode,"止损比例")==0){
        it->second->dStopRatio= atof(pszValue);
    }
}
```

接下来，我们需要加载交易策略选定合约的个性化参数值。盈佳期货交易终端在 IPolicy 接口中提供了 SetInstrumentStringParam()来读取交易策略配置文件中各合约的参数值，我们在该函数的实现中，将传入的参数值存储在 m_params 中：

```
void CMyTestPolicy::SetInstrumentStringParam(const char *pszInstrumentName, const char *pszParamName, const char *pszValue)
{
    map<string, CPARAMS >::iterator it = m_params.find(pszInstrumentName);
    if(it == NULL || it == m_params.end()){//未加载该合约参数
```

```

CPARAMS *pCPARAMS=new CPARAMS();
it=m_params.insert(pair<string, CPARAMS >( pszInstrumentName, pCPARAMS));
}
if(strcmp(pszParamName,"样本数据总数")==0){
    it->second->nSampleSize = atof(pszValue);
}else if(strcmp(pszParamName,"每笔开仓手数")==0){
    it->second->nOpenVolume = atof(pszValue);
}else if(strcmp(pszParamName,"止损比例")==0){
    it->second->dStopRatio = atof(pszValue);
}
}
}

```

3.3.2 交易信号检测

当交易策略参数中设置的合约有新的行情时，IPolicy 接口的 NotifyNewMD()函数将会被框架调用，以传送新的行情数据供交易策略使用。我们示例中的交易信号是建立在最新价序列的基础上：

- 1) 最新价 > $MV(n) + Std(n)$ ，以卖一价买开仓 v 手；
- 2) 最新价 < $MV(n) - Std(n)$ ，以买一价卖开仓 v 手。

我们首先创建一个统计数据处理类 CStatSeries，该类的功能主要是存储新的行情数据序列(最新的 n 个数据)，并计算数据序列的均值和标准差。

```

//StatSeries.h
#pragma once
#include <vector>
#include <math.h>

using namespace std;
class CStatSeries
{
public:
    CStatSeries(int nMaxNum );
    ~CStatSeries(void);
private:
    bool m_isSync;
    int m_nMaxNum;

```

```
double m_dAV;
double m_dSTD;
double m_dMax;
double m_dMin;
vector<double> m_vecSeries;
public:
    double getAV(void);
    double getSTD(void);
    void addData(double dNew);
    double getLastVal(void);
    double getMax(void);
    double getMin(void);
    int getSize(void);
};

//StatSeries.cpp
#include "StdAfx.h"
#include "StatSeries.h"

CStatSeries::CStatSeries(int nMaxNum)
:m_dAV(0),m_dSTD(0)
{
    m_nMaxNum=nMaxNum;
}

CStatSeries::~CStatSeries(void)
{
}

double CStatSeries::getAV(void)
{
    return m_dAV;
}

double CStatSeries::getSTD(void)
{
    if (!m_isSync){//标准差未同步，计算标准差
        m_dSTD=0;
        int nSize=m_vecSeries.size();
        if (nSize>=MIN_NORMAL_NUM){//大于 MIN_NORMAL_NUM 是为了尽量满足正态性
            for(int i=0;i<nSize;i++){
                double dDiff = abs(m_vecSeries.at(i)-m_dAV);
                m_dSTD+=pow((double)dDiff,2);          //离差平方和
            }
        }
    }
}
```

```

    }
    m_dSTD/=(m_vecSeries.size()-1);           //离差平方和平均，无偏估
计。
    m_dSTD=sqrt(m_dSTD);
    m_isSync=true;
    }
    }
    return m_dSTD;
}

void CStatSeries::addData(double dNew)
{
    m_isSync=false;//加入新数据后标准差需要重新计算,但现在只计算均值，标准差用
    时再算。

    m_vecSeries.push_back(dNew);
    int nSize=(int)m_vecSeries.size();
    if(nSize>m_nMaxNum){ //数据超过容量，需要删除最老的数据
        double dFront=m_vecSeries.front();
        m_vecSeries.erase(m_vecSeries.begin());
        m_dAV=m_dAV*(m_nMaxNum)+dNew-dFront;
        nSize=m_nMaxNum;
    }
    else{
        m_dAV=m_dAV*(nSize-1)+dNew;
    }
    m_dAV/=nSize;
}

double CStatSeries::getLastVal(void)
{ //取最后更新的值
    return m_vecSeries.back();
}

double CStatSeries::getMax(void)
{ //取最大值
    int nSize=m_vecSeries.size();
    m_dMax=0;
    for(int i=0;i<nSize;i++){
        if(m_dMax<m_vecSeries.at(i)) m_dMax=m_vecSeries.at(i)
    }
    return m_dMax;
}

```

```
double CStatSeries::getMin(void)
{//取最小值
    int nSize=m_vecSeries.size();
    m_dMax=0;
    for(int i=0;i<nSize;i++){
        if(m_dMin>m_vecSeries.at(i)) m_dMin=m_vecSeries.at(i)
    }
    return m_dMin;
}

int CStatSeries::getSize(void)
{
    return (int)m_vecSeries.size();
}
```

完成了基本的数据处理类，现在我们就可以实现 IPolicy 接口的

NotifyNewMD()函数：

```
virtual void NotifyNewMD(CThostFtdcDepthMarketDataField *pField){
    static map<string,CStatSeries> m_mdData;
    map<string, CStatSeries >::iterator it = m_mdData.find(pField->InstrumentID);
    if(it = NULL || it = m_mdData.end()){//未加载对应合约的行情数据
        CStatSeries *pCStatSeries =new CStatSeries (
            m_params.find(pField->InstrumentID)-> nSampleSize);
        it= m_mdData.insert(pair<string, CStatSeries >( pField->InstrumentID,
            pCStatSeries));
    }
    it->second->addData(pField->LastPrice)

    ///交易信号检测
    if(pField->LastPrice > it->second->getAV()+it->second->getSTD()){//买开仓
        m_pOrderAPI->Buy(pField->InstrumentID,
            m_params.find(pField->InstrumentID)->nOpenVolume,pField->AskPrice1);
    }
    else if(pField->LastPrice < it->second->getAV()-it->second->getSTD()){//卖开仓
        m_pOrderAPI-> Short (pField->InstrumentID,
            m_params.find(pField->InstrumentID)->nOpenVolume,pField-> BidPrice1);
    }
};
```

3.3.3 止赢止损判定

现在我们在 NotifyNewMD()函数中加入止盈止损相关的处理逻辑

辑：使用移动止盈止损，最新价 $<0.999 * \max(n)$ 时，以买一价卖平仓了结全部多头头寸；最新价 $>1.001 * \min(n)$ ，以卖一价买平仓了结全部空头头寸。

```
virtual void NotifyNewMD(CThostFtdcDepthMarketDataField *pField){
    ...
    ///交易信号检测
    ...
    ///止盈止损判定
    long longPos = m_pOrderAPI->GetLong(pField->InstrumentID);
    long shortPos = m_pOrderAPI->GetShort(pField->InstrumentID);

    if(longPos>0){//多头持仓，行情下跌止损
        if(pField->LastPrice <it->second->getMax()*(1-
            m_params.find(pField->InstrumentID)->dStopRatio)){
            m_pOrderAPI->Sell(pField->InstrumentID, longPos,pField->BidPrice1);
        }
    }

    if(shortPos >0){//空头持仓，行情上涨止损
        if(pField->LastPrice <it->second->getMin()*(1+
            m_params.find(pField->InstrumentID)->dStopRatio)){
            m_pOrderAPI-> Cover(pField->InstrumentID, shortPos,pField-> AskPrice1);
        }
    }
};
```

3.3.4 资金管理逻辑

最后，我们在开仓之前加入资金管理的逻辑：可用资金大于动态权益的 50%时才允许开立新的头寸。

```
virtual void NotifyNewMD(CThostFtdcDepthMarketDataField *pField){
    ...
    ///交易信号检测与资金管理逻辑
    CThostFtdcTradingAccountField tradeAccount;
    m_pOrderAPI->GetTradingAccount(&tradeAccount);
    if(tradeAccount->Available> tradeAccount->Balance/2){//资金管理逻辑
        if(pField->LastPrice > it->second->getAV()+it->second->getSTD()){//买开仓
            m_pOrderAPI->Buy(pField->InstrumentID,
                m_params.find(pField->InstrumentID)->nOpenVolume,pField->AskPrice1);
        }
    }
};
```

```

    }
    else if(pField->LastPrice < it->second->getAV()-it->second->getSTD()){//卖开仓
        m_pOrderAPI-> Short (pField->InstrumentID,
            m_params.find(pField->InstrumentID)->nOpenVolume,pField->
            BidPrice1);
    }
} //资金管理逻辑
...
///止盈止损判定
};

```

3.4 交易策略部署

编译 MyTestPolicy.dll, 连同交易策略配置文件 MyTestPolicy.policy 一起复制到盈佳程交易终端的安装目录下, 然后重新启动盈佳交易终端, 名称为“均值突破交易策略”的交易策略将显示在交易策略窗口。用户可以设置交易策略的交易参数, 并控制交易策略的运行。

3.5 交易策略执行过程

1. 首先 CreatePolicy() 方法被调用创建该策略的一个对象, 并且 SetOrderAPI 方法被调用 (这个方法在 IPolicy 中有缺省实现), 设置框架封装的行情/交易接口的一个对象。
2. 根据交易策略配置文件, 调用 SetGlobalStringParam(const char *pszName,const char *pszValue)设置该交易策略的全局参数; 调用 SetInstrumentStringParam(const char * pszInstrument,const char * pszParamName,const char * Value)方法设置该策略所适用的合约级别参数。策略需要将这些参数保存下来。
3. Start()方法被调用;

4. 框架订阅该策略所适用合约的行情;
5. 当所订阅的行情到达后, NotifyNewMD()方法被调用;
6. 当该策略所发出的报单有成交后, NotifyTrade()方法被调用;
7. 当报单失败或者被撤后, NotifyOrderFailed()方法被调用;
8. 当策略注册的定时器(几秒后触发的定时器)到期时 NotifyTimer()
方法被调用;
9. 停止策略时 Stop 方法被调用。

3.6 注意事项

1. 在策略代码中, 不要使用 UI;
2. 在策略代码中, 不要使用 Sleep 方法;可以注册 RegisterTimer 方法完成。

第4章 交易策略接口说明

4.1 IPolicy 接口说明

IPolicy 是每个策略都必须继承的接口, 该接口提供了一些成员变量和虚函数供子类继承或覆盖, 下面逐一介绍。

4.1.1 成员变量

1) IOrderAPI * m_pOrderAPI;

这个变量缺省情况下, 由框架调用方法 IPolicy 的 SetOrderAPI 方法赋值 (参见 IPolicy 类的申明), SetOrderAPI 方法在 Start 方法之前

被调用，并且只被调用一次。一般情况下，子类不需要覆盖 SetOrderAPI 方法。

2) int m_nID;

标识策略的编号，由框架通过 SetID 方法来赋值。不同的策略由不同的编号，该编号设置在 CThostFtdcOrderField2 结构体中的 PolicyID 字段。在策略列表中的策略没有增减的情况下，同一个策略不同次运行时的 m_nID 相同。

4.1.2 成员方法

1) SetGlobalStringParam

设置一个交易策略的全局参数。该方法在一个策略启动时被调用，设置某个参数的初始值（此时在 Start 方法之前被调用）；在运行时，如果用户通过界面改了某个全局参数的值，则该方法也会被调用来设置更改后的值。

2) SetGlobalParam

同 SetGlobalStringParam 基本相同，如果某个参数的类型被设置为 number 则通过此方法来设置值，否则通过 SetGlobalStringParam 来设置值。

3) SetInstrumentStringParam

设置一个交易策略的合约级别参数。该方法在一个策略启动时被调用，设置某个合约参数的初始值（此时在 Start 方法之前被调用）；在运行时，如果用户通过界面改了某个合约参数的值，则该方法也会

被调用来设置更改后的值。

4) SetInstrumentParam

同 SetInstrumentStringParam 基本相同，如果某个合约参数的类型被设置为 number 则通过此方法来设置值，否则通过 SetInstrumentStringParam 来设置值。

5) SetInstruments

设置当前策略所适用的合约，在 Start 方法之前被调用。

6) Start

当用户通过界面“启动”策略时，该方法被调用。

7) Stop

当用户通过界面“停止”策略时，该方法被调用。

8) Pause

当用户通过界面“暂停”策略时，该方法被调用。

9) Resume

当用户通过界面“继续”策略时，该方法被调用。

10) NotifyNewMD

当该策略所适用的合约有新行情到达时被调用。注意，如果策略处于暂停状态，则方法会被调用。参数为一个行情结构体，具体说明参见上期技术接口的头文件。

11) NotifyTrade

该策略所发出的报单有成交时被调用。

12) NotifyOrderFailed

该策略所发出的报单未成功或者被撤时，本方法被调用。

13) NotifyOrderFinished

该策略所发出的某个报单完全成交后，本方法被调用。

14) NotifyTimer

该策略所注册的定时器到期后，本方法被调用。

4.2 IOrderAPI 接口说明

4.2.1 交易类接口

- ✧ Buy- 买入开仓
- ✧ Sell –卖出平仓
- ✧ Short – 卖出开仓
- ✧ Cover – 买入平仓
- ✧ CancelOrder – 撤单

4.2.2 报单管理类接口：

- ✧ GetOrder – 查询一个报单信息
- ✧ GetOrderStatus – 查询一个报单状态
- ✧ GetOpenOrderIDs – 查询所有可撤的报单编号
- ✧ IsOrderFinished – 判断一个报单是否已经成交完成；
- ✧ IsOrderFailed – 判断一个报单是否已经撤单或者是错误报单；

4.2.3 行情类接口

- ✧ GetOpenPrice -查询一个合约的开盘价;
- ✧ GetAskPrice-查询一个合约的当前卖一价;
- ✧ GetBidPrice-查询一个合约的当前买一价;
- ✧ GetLastPrice--查询一个合约的最新成交价;
- ✧ GetHighestPrice -查询一个合约从开盘到现在为止的最高价;
- ✧ GetLowestPrice-查询一个合约从开盘到现在为止的最低价;
- ✧ GetLowerLimitPrice 查询一个合约的跌停价
- ✧ GetHighLimitPrice -查询一个合约的涨停价;
- ✧ GetMultiply 查询一个合约的乘数;
- ✧ GetMarketData 查询一个合约的最新行情信息 (CTP 返回的完整行情信息)

4.2.4 仓位类接口

- ✧ GetLong 查询一个合约当前的多头 (买持) 仓位
- ✧ GetLongClosable 查询一个合约当前的多头 (买持) 仓位的可平量
- ✧ GetShort 查询一个合约当前的空头 (卖持) 仓位
- ✧ GetShortClosable 查询一个合约当前的空头 (卖持) 仓位的可平量
- ✧ GetLongByMe 查询当前策略所开的一个合约多头 (买持) 仓位
- ✧ GetShortByMe 查询当前策略所开的一个合约空头 (卖持) 仓位

- ✧ `GetPositionPrice` 查询一个合约的多头(买持)仓位和空头(卖持)仓位的持仓均价;

4.2.5 资金类接口

- ✧ `GetTradingAccount` 查询一个账户当前的资金账户信息

4.2.6 日志类接口

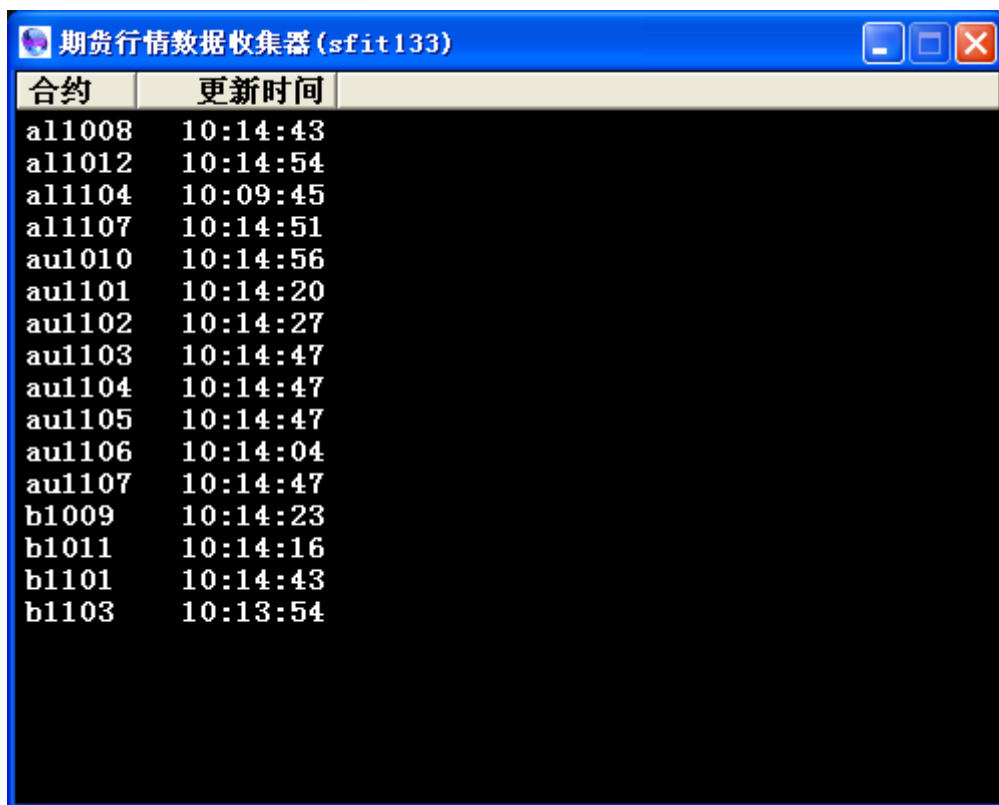
- ✧ `Log_Trace` 输出跟踪日志
- ✧ `Log_Debug` 输出调试日志
- ✧ `Log_Info` 输出信息日志
- ✧ `Log_Warn` 输出警告日志
- ✧ `Log_Error` 输出错误信息
- ✧ `Log_Fatal` 输出致命错误日志
- ✧ `IsEnabledFor` 判断是否允许某个级别的日志
- ✧ `GetLoggerLevel` 查询某个 `Logger` 的日志级别

上述的方法的消息描述请参见安装目录\policybase\policybase.h 文件。

第5章 交易策略回归测试工具

5.1 行情收集器

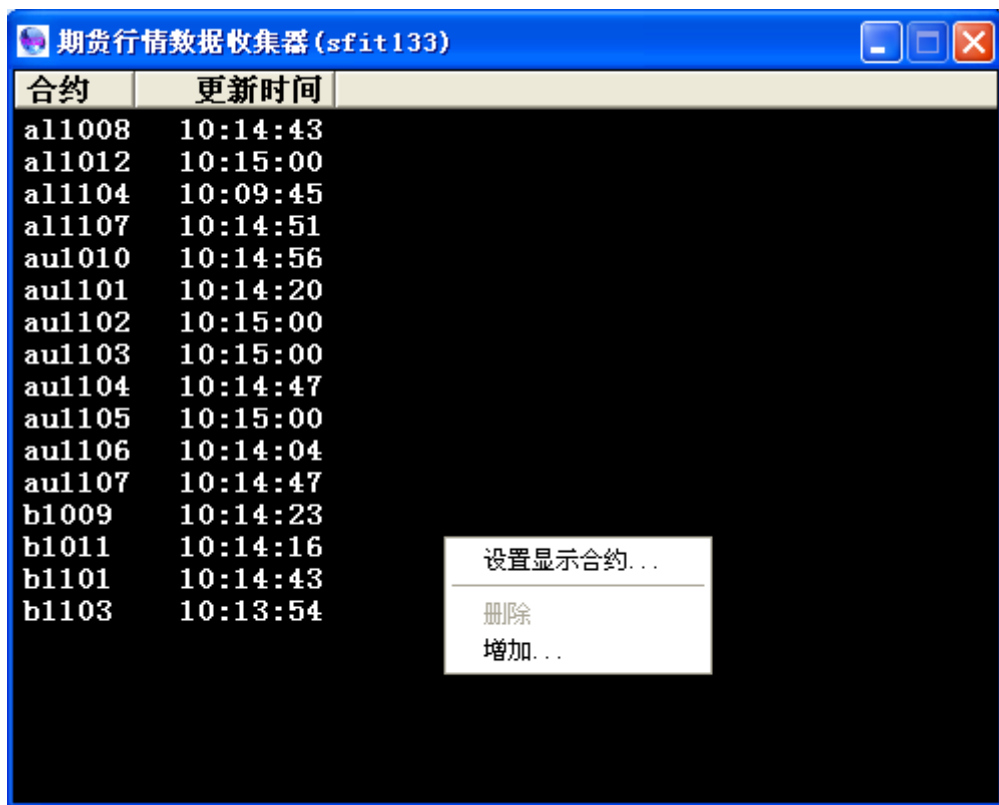
行情收集器的界面如下所示:



合约	更新时间
al1008	10:14:43
al1012	10:14:54
al1104	10:09:45
al1107	10:14:51
au1010	10:14:56
au1101	10:14:20
au1102	10:14:27
au1103	10:14:47
au1104	10:14:47
au1105	10:14:47
au1106	10:14:04
au1107	10:14:47
b1009	10:14:23
b1011	10:14:16
b1101	10:14:43
b1103	10:13:54

图-1.9

以上列表为当前正在收集的合约，可以通过右键进行合约的增减：



合约	更新时间
al1008	10:14:43
al1012	10:15:00
al1104	10:09:45
al1107	10:14:51
au1010	10:14:56
au1101	10:14:20
au1102	10:15:00
au1103	10:15:00
au1104	10:14:47
au1105	10:15:00
au1106	10:14:04
au1107	10:14:47
b1009	10:14:23
b1011	10:14:16
b1101	10:14:43
b1103	10:13:54

设置显示合约...

删除

增加...

图-1.10

收集到的行情存储于终端安装目录下 **data** 子目录中的以交易日命名的文件里。这些行情数据文件，可以被用在仿真测试终端中进行历史数据回归测试。



图-1.11

5.2 仿真测试终端

仿真测试终端是用来对策略进行测试的工具。可以利用行情收集器收集到的历史数据进行测试，也可以直接连接到实盘中，利用真实行情进行测试。

两种模式的切换可以通过界面上的“行情数据”进行切换：



图-1.12

在选择“回放历史行情”的时候，需要选择一个文件，这个文件需要是由行情收紧器产生的数据文件。

仿真交易终端按照如下原则进行模拟成交（注意:仿真交易终端中都是模拟成交，而不是真实的成交）：

1. 发出买单时，如果此时行情的卖价小于或等于报单价，则成交，否则挂单；
2. 对于挂单中的买单，如果此时行情的卖价或者最新价小于或等于报单价，则成交，否则继续挂单，直到撤单为止；
3. 卖单也是对应的处理方式；

模拟成交后，同样的计算持仓和盈亏。

第6章 COM 接口

盈佳也发布了一个基于 COM 标准的 CTP-API 接口封装，通过这个 COM 接口，用户可以利用任何支持 COM 的语言来接入 CTP；该

COM 组件在内部进行仓位和资金的自动计算，并对用户屏蔽了上期所的平今和平昨的差别（优先平今），对用户非常友好。

6.1 COM 包构成

该 COM 组件在运行时，需要下面几个文件：

- ✧ ctpcom.dll
- ✧ thosttraderapi.dll
- ✧ thostmduserapi.dll
- ✧ config.xml（可以有多个，不同的期货公司需要不同的配置文件，在登录时指定这个配置文件的名称）

6.2 COM 注册

在使用 COM 组件之间，需要先注册注本 COM 组件，步骤如下：

- ✧ 运行 cmd，
- ✧ cd 到 ctpcom.dll 所在的目录，
- ✧ 运行 Regsvr32 ctpcom.dll

6.3 COM 接口方法

该 COM 提供的接口(c++语言描述)如下：

- ✧ Login([in] BSTR ConfigFile,[in] BSTR UserID,[in] BSTR Password,[out] int *ErrorID);

说明：

登录到 CTP 系统。必须首先调用该方法成功登录后，

才能调用其他方法。

参数:

ConfigFile, 配置文件的名称, 该配置文件必须和 ctpcom.dll 放在同一个目录下。通过指定不同的配置文件, 可以创建多个 ctpcom 对象, 连接到不同的期货经纪公司。

UserID, 投资者账户名

Password, 投资者密码;

ErrorID, 返回登录错误代码, 为 0 时表示登录成功, 其他值表示登录失败。

✧ Buy([in] BSTR pszInstrumentID,[in] int volume,[in] double price,[out] long* OrderID);

说明:

对指定的合约买入开仓;

参数:

pszInstrumentID, 合约代码, 区分大小写;

volume, 开仓手数,

price, 开仓价格, 0 表示以市价开仓(对于上期所, 以停板价+立即撤单来模拟)

OrderID, 返回该报单的唯一编号,

可以通过这个编号来跟踪此报单;

✧ Sell([in] BSTR pszInstrumentID,[in] int volume,[in] double price,[out] long* OrderID);

说明:

对指定的合约卖出平仓;

参数:

pszInstrumentID, 合约代码, 区分大小写;

volume, 平仓手数,

price, 平仓价格, 0 表示以市价平仓(对于上期所,
以停板价+立即撤单来模拟)

OrderID, 返回该报单的唯一编号,

可以通过这个编号来跟踪此报单;

✧ Short([in] BSTR pszInstrumentID,[in] int volume,[in] double
price,[out] long* OrderID);

说明:

对指定的合约卖出开仓;

参数:

pszInstrumentID, 合约代码, 区分大小写;

volume, 开仓手数,

price, 开仓价格, 0 表示以市价开仓(对于上期所,
以停板价+立即撤单来模拟)

OrderID, 返回该报单的唯一编号, 可以通过这个
编号来跟踪此报单;

✧ Cover([in] BSTR pszInstrumentID,[in] int volume,[in] double
price,[out] long* OrderID);

说明:

对指定的合约买入平仓;

参数:

pszInstrumentID, 合约代码, 区分大小写;

volume, 平仓手数,

price, 平仓价格, 0 表示以市价平仓(对于上期所,
以停板价+立即撤单来模拟)

OrderID, 返回该报单的唯一编号, 可以通过这个
编号来跟踪此报单;

✧ CancelOrder([in] long OrderID,[out]int* status);

说明:

对指定的报单进行撤单操作;

参数:

OrderID, 指定的报单号;

Status, 返回代码, >0 表示已经对该报单发出撤单指令,
<=0 表示该报单不存在, 或者已经不可撤;

✧ SubscribeMD([in] BSTR InstrumentID);

说明:

订阅某个合约的行情数据。在订阅了后, 会通过
OnMarketData 事件将该合约的行情发出。

参数:

InstrumentID – 合约代码, 区分大小写。

✧ GetLongPos([in] BSTR InstrumentID,[out] long * LongPos);

说明:

返回某个合约的多头（买持）仓位。

参数:

InstrumentID – 合约代码，区分大小写。

LongPos – 返回的多头（买持）仓位大小；

✧ GetLongClosable([in] BSTR InstrumentID,[out] long * LongClosable);

说明:

返回某个合约的多头（买持）可平仓位。

参数:

InstrumentID – 合约代码，区分大小写。

LongClosable – 返回的多头（买持）仓位可平量；

✧ GetShortPos([in] BSTR InstrumentID,[out] long * ShortPos)

说明:

返回某个合约的空头（卖持）仓位。

参数:

InstrumentID – 合约代码，区分大小写。

LongPos – 返回的空头（卖持）仓位大小；

✧ GetShortClosable([in] BSTR InstrumentID,[out] long * ShortClosable);

说明:

返回某个合约的空头（卖持）可平仓位。

参数：

InstrumentID – 合约代码，区分大小写。

ShortClosable– 返回的空头（卖持）仓位的可平量；

✧ IsOrderOpen([in] long OrderID,[out] BOOL *IsOpen);

说明：

判断某个报单是否处于可撤状态。

参数：

OrderID – 报单编号。

IsOpen– 不为 0 表示处于可撤状态，0 表示不可撤；

6.4 COM 触发事件

客户端通过事件来接受行情、资金和仓位等信息。该 COM 组件发出的事件如下：

✧ OnAccount – 当投资者账户信息发生变化时，此事件被触发；

✧ OnMarketData - 当某个合约有新行情到达时，此事件被触发；

✧ OnOrder – 当某个报单信息发生变化时，此事件被触发；

✧ OnOrderCanceled - 当出现报单错误或者报单被撤时，此事件被触发；

✧ OnOrderFinished – 当某个报单全部成交后，此事件被触

发；此事件总是在一个报单的所有 OnTrade 事件之后发生。

- ✧ OnPosition – 当某个合约的仓位信息发生变化时，此事件被触发；
- ✧ OnTrade – 当所发出的报单有成交时，此事件被触发；
- ✧ OnTradeConnected – 当交易连线后，此事件被触发；
- ✧ OnTradeDisconnected – 当交易断线后，此事件被触发；
- ✧ OnMDConnected – 当行情连线后，此事件被触发；
- ✧ OnMDDisconnected – 当行情断线后，此事件被触发；
- ✧ OnOrderActionFailed([in] long OrderID,[in] int ErrorID);当撤单失败时，此事件被触发；

6.5 COM 使用示例

6.5.1 VB6.0 示例

```
Dim WithEvents ctp As ICTPClientAPI

Private Sub Login_Click()

    Dim errorID As Long
    Dim succ As Long

    Set ctp = New CTPCOMLib.ICTPClientAPI

    Call ctp.Login("config.xml", "00100", "888888", errorID)
    If errorID = 0 Then
        MsgBox "登录 ok"
    Else
        MsgBox "登录错误: " & errorID
    End If

End Sub
```

```
Private Sub Buy_Click()  
    Dim OrderID As Long  
    Dim index As Integer  
    Dim tm1, tm2 As Date  
  
    Call ctp.Buy("IF1008", 1, 0, OrderID)
```

```
End Sub
```

```
Private Sub MD_Click()  
    Call ctp.SubscribeMD("IF1008")  
End Sub
```

```
Private Sub Close_Click()  
    Set ctp = Nothing
```

```
End Sub
```

```
Private Sub Sell_Click()  
    Dim OrderID As Long  
  
    Call ctp.Sell("IF1008", 1, 0, OrderID)  
End Sub
```

```
Private Sub ctp_OnAccount(ByVal PreBalance As Double, ByVal Balance As Double, ByVal  
Available As Long, ByVal Commission As Double, ByVal Margin As Double, ByVal CloseProfit  
As Double, ByVal PositionProfit As Double)  
    Label1.Caption = Available  
End Sub
```

```
Private Sub ctp_OnMarketData(ByVal InstrumentID As String, ByVal BidPrice1 As Double,  
ByVal BidVolume1 As Long, ByVal AskPrice1 As Double, ByVal AskVolume1 As Long, ByVal  
OpenPrice As Double, ByVal HighestPrice As Double, ByVal LowestPrice As Double, ByVal  
LastPrice As Double, ByVal OpenInterest As Long, ByVal Volume As Long, ByVal  
UpperLimitPrice As Double, ByVal LowerLimitPrice As Double, ByVal UpdateTime As String,  
ByVal UpdateMilliSecond As Long)
```

```
End Sub
```

```
Private Sub ctp_OnPosition(ByVal InstrumentID As String, ByVal IsLong As Long, ByVal  
Volume As Long, ByVal CloseProfit As Double, ByVal PositionProfit As Double, ByVal  
TotalClosable As Long, ByVal TodayClosable As Long)  
    Label2.Caption = Volume
```

End Sub

6.5.2 VC6.0 示例

在 VC 中创建一个 CEventHandler 类，如下：

```
class CCTPComEventHandler
:public IDispatchImpl<1,CCTPComEventHandler>
{
private:

    CComPtr<IICTPClientAPI> m_pCtpClient;

    void ShowHResultMessage(HRESULT hr){
        CString cs;
        CString msg = _com_error(hr).ErrorMessage();
        cs.Format(_T("Error 0x%08x: %s"), hr, msg);

        AfxMessageBox(cs);
    }

public:
    IICTPClientAPI* CreateCTPCom(){
        HRESULT hr;

        hr = m_pCtpClient.CoCreateInstance(CLSID_ICTPClientAPI);
        if(hr != S_OK){
            ShowHResultMessage(hr);
            return NULL;
        }

        hr = AtlGetObjectSourceInterface(m_pCtpClient, &this->m_libid,
            &this->m_iid, &this->m_wMajorVerNum, &this->m_wMinorVerNum);

        if(FAILED(hr)){
            ShowHResultMessage(hr);
            return NULL;
        }

        hr = DispEventAdvise(m_pCtpClient,&DIID__IICTPClientAPIEvents);

        if(FAILED(hr)){
            ShowHResultMessage(hr);
```

```
}

IICTPClientAPI *pCtpCom=NULL;
hr =m_pCtpClient->QueryInterface(IID_IICTPClientAPI,(void **)&pCtpCom);
if(hr != S_OK){
    ShowHResultMessage(hr);
    return NULL;
}

return pCtpCom;

}

BEGIN_SINK_MAP(CCTPComEventHandler)
SINK_ENTRY(1, 1, OnMarketData)
SINK_ENTRY(1, 2, OnAccount)
SINK_ENTRY(1, 3, OnOrder)
SINK_ENTRY(1, 4, OnTrade)
SINK_ENTRY(1, 5, OnOrderCanceled)
SINK_ENTRY(1, 6, OnOrderFinished)
SINK_ENTRY(1, 7, OnTradeConnected)
SINK_ENTRY(1, 8, OnTradeDisconnected)
SINK_ENTRY(1, 9, OnMDConnected)
SINK_ENTRY(1, 10, OnMDDisconnected)
SINK_ENTRY(1, 11, OnPosition)
END_SINK_MAP()

STDMETHODIMP OnMarketData(BSTR InstrumentID, DOUBLE BidPrice1, INT
BidVolume1, DOUBLE AskPrice1, INT AskVolume1, DOUBLE OpenPrice, DOUBLE
HighestPrice, DOUBLE LowestPrice, DOUBLE LastPrice, LONG OpenInterest, LONG
Volume, DOUBLE UpperLimitPrice, DOUBLE LowerLimitPrice, BSTR UpdateTime,
    INT UpdateMilliSecond){
    //TODO 在这里处理行情;
    return S_OK;
}

STDMETHODIMP OnAccount(DOUBLE PreBalance, DOUBLE Balance, INT
Available, DOUBLE Commission, DOUBLE Margin, DOUBLE CloseProfit, DOUBLE
PositionProfit)
{
    //TODO 在这里处理账户通知;
    return S_OK;
}
```

```
STDMETHODIMP OnOrder(LONG OrderID, BSTR InstrumentID, INT IsBuy, INT IsOpen, INT Volume, DOUBLE Price, INT TradedVolume, DOUBLE AvgTradePrice, INT OrderStatus)
```

```
{
    //TODO 在这里处理报单回报
    return S_OK;
}
```

```
STDMETHODIMP OnTrade(LONG OrderID, BSTR InstrumentID, INT IsBuy, INT IsOpen, INT ThisTradeVolume, DOUBLE ThisTradePrice)
```

```
{
    //TODO 在这里处理成交回报
    return S_OK;
}
```

```
STDMETHODIMP OnOrderCanceled(LONG OrderID)
```

```
{
    //TODO 在这里处理下单失败或者撤单通知;
    return S_OK;
}
```

```
STDMETHODIMP OnOrderFinished(LONG OrderID)
```

```
{
    //TODO 在这里处理报单全部成交通知;
    return S_OK;
}
```

```
STDMETHODIMP OnTradeConnected()
```

```
{
    return S_OK;
}
```

```
STDMETHODIMP OnTradeDisconnected()
```

```
{
    return S_OK;
}
```

```
STDMETHODIMP OnMDConnected()
```

```
{
    return S_OK;
}
```

```
STDMETHODIMP OnMDDisconnected()
```

```

    {
        return S_OK;
    }

    STDMETHODCALLTYPE OnPosition(BSTR InstrumentID, INT IsLong, INT Volume,
    DOUBLE CloseProfit, DOUBLE PositionProfit, INT TotalClosable, INT TodayClosable)
    {
        return S_OK;
    }

    //Helper for finding the function index for a DISPID
    //!!!!!!!!!!MS altcom Bug Fix!!!!!!!!!!!!!!
    HRESULT GetFuncInfoFromId(const IID& /*iid*/, DISPID dispidMember, LCID lcid,
    _ATL_FUNC_INFO& info)
    {
        CComPtr<ITypeInfo> spTypeInfo;
        if (InlineIsEqualGUID(*_tih.m_plibid, GUID_NULL))
        {
            _tih.m_plibid = &m_libid;
            _tih.m_pguid = &m_iid;
            _tih.m_wMajor = m_wMajorVerNum;
            _tih.m_wMinor = m_wMinorVerNum;
        }
        HRESULT hr = _tih.GetTI(lcid, &spTypeInfo);
        if (FAILED(hr))
            return hr;
        CComQIPtr<ITypeInfo2, &IID_ITypeInfo2> spTypeInfo2 = spTypeInfo;
        FUNCDESC* pFuncDesc = NULL;
        if (spTypeInfo2 != NULL)
        {
            UINT nIndex;
            hr = spTypeInfo2->GetFuncIndexOfMemId(dispidMember, INVOKE_FUNC,
            &nIndex);

            if (FAILED(hr))
                return hr;
            hr = spTypeInfo->GetFuncDesc(nIndex, &pFuncDesc);
            if (FAILED(hr))
                return hr;
        }
        else // search for funcdesc
        {
            TYPEATTR* pAttr;
            hr = spTypeInfo->GetTypeAttr(&pAttr);

```

```

        if (FAILED(hr))
            return hr;
        for (int i=0;i<pAttr->cFuncs;i++)
        {
            hr = spTypeInfo->GetFuncDesc(i, &pFuncDesc);
            if (FAILED(hr))
                return hr;
            if (pFuncDesc->memid == dispidMember)
                break;
            spTypeInfo->ReleaseFuncDesc(pFuncDesc);
            pFuncDesc = NULL;
        }
        spTypeInfo->ReleaseTypeAttr(pAttr);
        if (pFuncDesc == NULL)
            return E_FAIL;
    }

    // If this assert occurs, then add a #define _ATL_MAX_VARTYPES nnnn
    // before including atlcom.h
    ATLASSERT(pFuncDesc->cParams <= _ATL_MAX_VARTYPES);
    if (pFuncDesc->cParams > _ATL_MAX_VARTYPES)
        return E_FAIL;

    for (int i=0; i<pFuncDesc->cParams; i++)
    {
        info.pVarTypes[i] = pFuncDesc->lprgelemdescParam[i].tdesc.vt;
        if (info.pVarTypes[i] == VT_PTR)
            info.pVarTypes[i] = pFuncDesc->lprgelemdescParam[i].tdesc.lptdesc->vt |
VT_BYREF;
        if (info.pVarTypes[i] == VT_USERDEFINED)
            info.pVarTypes[i] =
GetUserDefinedType(spTypeInfo,pFuncDesc->lprgelemdescParam[i].tdesc.hreftype);
    }

    VARTYPE vtReturn = pFuncDesc->elemdescFunc.tdesc.vt;
    switch(vtReturn)
    {
    case VT_INT:
        vtReturn = VT_I4;
        break;
    case VT_UINT:
        vtReturn = VT_UI4;
        break;
    case VT_VOID:

```

```
        vtReturn = VT_EMPTY; // this is how DispCallFunc() represents void
        break;
    case VT_HRESULT:
        vtReturn = VT_ERROR;
        break;
    }
    info.vtReturn = vtReturn;
    info.cc = pFuncDesc->callconv;
    info.nParams = pFuncDesc->cParams;
    spTypeInfo->ReleaseFuncDesc(pFuncDesc);
    return S_OK;
}

public:
    CCTPComEventHandler();
    virtual ~CCTPComEventHandler();
};
```

通过这个类可以创建 COM 对象的实例，并监听 COM 所发出的事件（在事件方法中 return S_OK 之前添加自己处理逻辑即可）。。

第7章 盈佳服务

最大限度的满足投资者的需求就是盈佳期货交易终端软件的目标。盈佳期货交易终端软件除提供按照合同规定的标准优质服务之外，还提供基于本终端软件的程序化交易策略的快速定制服务，满足投资者在手工下单、程序化交易以及个性化等各方面的需求。

联系人：褚光华； MSN（邮箱）：guanghuachu@hotmail.com

QQ：1206528070； 电话：021-58919288； 手机：13524168419