Tac FinTech

# TacMars 交易系统
# API 使用手册

**V2.3.2**

**上海广策信息技术有限公司**

**2022 年 10 月 17 日**

# 声明



# 商标声明



# 注意

本文档内容仅作指导使用，本文档内容会根据产品版本升级或其他原因进行不定期更新，除另有约定外，本文档所载内容不构成任何担保。

用户购买的产品受与广策签订的商业合同条款约束，本文档内容所陈述的产品性能，服务也许全部或部分与用户所购产品不符，除合同另有约定，广策对本文档所载内容不做任何明示或暗示的保证。

上海广策信息技术有限公司
地址：上海市浦东新区达尔文路 88 号半岛科技园 10 号楼
2 层电话：021-68547307 传真：021-68547307
邮箱：support@tacfintech.cn

# 文档修订历史

| 版本 | 日期 | 描述 | 修改人 |
|---|---|---|---|
| 2.3.2 | 2022.10.17 | 处理掉过高的 cpu 占用 | 余国良 |
| 2.3.1 | 2022.09.01 | 回单性能优化等 | 余国良 |
| 2.3.0 | 2022.08.22 | 1. 更新了 ReqOrderInsert、ReqOrderAction 返回值<br>2. ReqOrderInsert 在开启 Udp 时只支持 uint32_t OrderRef 上限<br>3. 增加了 SetUdpSendOpen 和 SetOpenFastSend、OnUdpMsgRej 方法<br>4.升级至此版本或更高时，需要重新编译。 | 潘鹰、余国良 |
| 2.1.4 | 2022.03.06 | 更新 ReqQryInstrument 方法，增加回单优化 | 狄云丰 |
| 2.1.2 | 2021.11.16 | 增加清理流文件、绑核。 | 杨丰 |

# 目录

# 第一章 TacMars 交易系统接口介绍

本部分主要介绍 TacMars 交易系统的接口，包括：

第一节引入 TacMars 交易系统的接口，TraderAPI 用于柜台系统下达交易、控制指令，接收私有流（含报单插入、报单操作响应和成交回报）、公共信息（如合约状态）、响应流。

第二节介绍了 API 使用的通信结构，重点说明了数据流。

第三节介绍了 API 对应于不同类型应用的编程接口。

第四节介绍了 API 的工作模式，包括线程之间的通信、私有流信息传输的可靠性实现。

作为 TacMars 交易系统的接口基础，TraderAPI 的用户应该仔细阅读第一部分内容。

# 1.1.介绍

## 1.1.1. TraderAPI 简介

TraderAPI 是一个基于 C++的类库，通过使用和扩展类库提供的接口来实现全部的交易功能，包括报单与录入、报单撤销、查询合约、查询资金、查询持仓等。该类库包含以下 4 个文件：

| 文件名 | 文件描述 |
|---|---|
| TacFtdcTraderApi.h | 交易接口头文件 |
| TacFtdcUserApiStruct.h | 定义了一系列业务相关的数据结构的头文件 |
| TacFtdcUserApiDataType.h | 定义了一系列业务相关的数据类型的头文件 |
| Libtactraderapi.so | 动态链接库二进制文件 |

目前只提供 Linux 64 位版本。

## 1.1.2. TraderAPI 运行平台

目前发布了以下操作系统平台的版本：

- Linux x64：CentOS 7

（.h 文件和.so 文件在 CentOS 7.2/gcc 4.8.5 下完成编译。）如果需要其他操作系统版本请联系：

上海广策信息科技有限公司电话：**86-21-51821796**

邮箱：**support@tacfintech.cn**

# 1.2.网路体系结构

TraderAPI 使用自定义通讯协议与 TacMars 交易系统进行通信。

## 1.2.1. 通讯模式

自定义的通讯模式共有两种：

- 对话通讯模式
- 私有通讯模式

对话通讯模式是指由客户系统主动发起的通讯请求。该请求被交易系统接收和处理，并给予响应，例如报单。这种通讯模式与普通的客户/服务器模式相同。

私有通讯模式是指交易系统主动向客户发出的信息，例如成交回报等。

## 1.2.2. 数据流

TacMars 支持对话通讯模式、私有通讯模式。

**1**、对话通讯模式

对话通讯模式下支持对话数据流。

对话数据流是一个双向数据流，客户系统发送交易请求，交易系统反馈应答。交易系统不维护对话流的状态。系统故障时，对话数据流会重置，在途的数据可能会丢失。

**2**、私有通讯模式

在私有通讯模式下，数据流是可靠的。在一个交易日内，客户系统断线后恢复连接时，可以请求交易系统发送指定序号之后的私有数据流数据。私有数据流向客户系统提供报单状态报告、成交回报等信息。

交易系统维护每个客户的私有流，有关该客户的回报信息，如报单回报、成交回报等都通过会员私有流下发。

# 1.3.接口模式

## 1.3.1. TraderAPI 接口

TraderAPI 提供两个接口，分别为 CTacFtdcTraderApi 和 CTacFtdcTraderSpi。

客户系统可以通过 CTacFtdcTraderApi 发出操作请求，通过继承 CTacFtdcTraderSpi 并重载回调函数来处理交易系统的回复或响应。

### **1.3.1.1.** 对话流和查询流编程接口

通过对话流进行通讯的编程接口通常如下：

```
////请求： int

CTacFtdcTraderApi::ReqXXX(

                    CTacFtdcXXXField *pReqXXX,

                    int nRequestID)

////响应： void

CTacFtdcTraderSpi::OnRspXXX(

                    CTacFtdcXXXField *pRspXXX,

                    CTacFtdcRspInfoField
                    *pRspInfo, int nRequestID, bool
                    bIsLast)
```

其中请求接口第一个参数为请求的内容，不能为空。该参数根据请求命令的不同使用不同的类，该类的成员变量的类型和合法的数值请参阅 API 接口。

请求接口的第二个参数为请求号，暂时未用。

当收到交易系统应答时，CTacFtdcTraderSpi 的回调函数会被调用。如果响应数据不止一个，则回调函数会被多次调用。

回调函数一共包含四个参数。其中：

- 第一个参数为响应的具体数据，如果出错或没有结果有可能为 NULL。
- 第二个参数为处理结果，表明本次请求的处理结果是成功还是失败。在发生多次回调时，除了第一次回调，其它的回调该参数都可能为 NULL。

- 第三个参数为请求号，暂时不用。
- 第四个参数为响应结束标志，表明是否是本次响应的最后一次回调。

**1.3.1.2.** 私有流编程接口

私有流中的数据为用户的私有信息，包括报单回报、成交回报等。

通过私有流接收回报的编程接口通常如下：

```
void CTacFtdcTraderSpi::OnRtnXXX(CTacFtdcXXXField *pXXX);
```

当收到交易系统通过私有流发布的回报数据时，CTacFtdcTraderSpi 的回调函数会被调用。回调函数的参数为回报的具体内容。

# 1.4.运行模式

## 1.4.1. 工作流程

客户系统/行情接收系统和交易系统的交互过程分为 2 个阶段：初始化阶段和功能调用阶段。

**1.4.1.1.** 初始化阶段

在初始化阶段，客户系统/行情接收系统的程序必须完成如下步骤（具体代码请参考开发实例）：

| 顺序 | 客户系统 |
|---|---|
| 1 | 产生一个 CTacFtdcTraderApi 实例； |
| 2 | 产生一个事件处理的实例； |
| 3 | 注册一个事件处理的实例； |
| 4 | 订阅私有流； |
| 5 | 设置交易前置的网络地址； |
| 6 | 初始化。 |

注意事项：在每一个 API 进程中只能生成一个 CTacFtdcTraderApi 实例。每一个
CTacFtdcTraderApi 实例只能初始化一次。

#### 1.4.1.2. 功能调用阶段

在功能调用阶段，客户系统可以任意调用交易接口中的请求方法，如 ReqUserLogin、ReqOrderInsert 等，同时提供回调函数以响应回报信息。注意事项：

API 请求的输入参数不能为 NULL。

API 请求的返回参数，0 表示正确，其他表示错误，详细错误编码请查表。

### 1.4.2. 工作线程

客户应用程序至少由两个线程组成，一个是应用程序主线程，一个是 API 工作线程（TraderAPI）。应用程序与 TacMars 的通信是由 API 工作线程驱动的。

CTacFtdcTraderApi 提供的接口不是线程安全的，一个 API 对象只能由一个线程调用，或者客户自己加锁保护。

CTacFtdcTraderSpi 提供的接口回调是由 TraderAPI 工作线程驱动，通过实现 SPI 中的接口方法，从 TacMars 收取所需数据。

如果客户系统应用程序重载的某个回调函数阻塞，则等于阻塞了 TraderAPI 工作线程，API 与 TacMars 的通信会停止，因此通常应该迅速返回。在 CTacFtdcTraderSpi 派生类的回调函数中，可以利用将数据放入缓冲区来实现迅速返回。

### 1.4.3. 与柜台系统的连接

TraderAPI 使用自定义协议与 TacMars 系统进行通信。TraderAPI 使用 CTacFtdcTraderApi::RegisterFront 方法注册柜台系统的网络地址。

### 1.4.4. 本地文件

TraderAPI 在运行过程中，会将一些数据写入本地文件中。调用 CreateFtdcTraderApi 函数，可以传递一个参数，指明存储本地文件的路径。该路径必须在运行前创建完成。本地文件的扩展名都是“.con”。

### 1.4.5. 日志文件

TraderAPI 提供了两个日志接口，用于记录通信日志。SetLogLevel 用于打开请求日志，SetLogFilePath 用于设置日志目录路径。日志打开后，所有的业务数据将记入日志。

### 1.4.6. 大页内存

TraderAPI v2.1.2 支持大页内存，大页内存可以带来更稳定的内存表现，抖动更小。

运行命令：

echo 1024 > /proc/sys/vm/nr_hugepages 来使能大页内存。

若服务器不支持大页内存，TraderAPI 会使用普通内存，对 API 的功能没有影响。

# 第二章 TraderAPI 参考手册

本章主要供交易系统开发商阅读，包括：

第一节从系统管理和业务的角度说明 TraderAPI 提供的接口和方法。

第二节为 TraderAPI 的参考手册。

第三节为 TraderAPI 的一个程序例子。

## 2.1.TraderAPI 接口分类

### 2.1.1. 管理接口

TraderAPI 的管理接口是对 API 的生命周期和运行参数进行控制。

| 接口类型 | 接口名称 | 说明 |
|---|---|---|
| 生命周期管理接口 | CTacFtdcTraderApi:: CreateFtdcTraderApi | 创建 TraderApi 实例 |
| | CTacFtdcTraderApi:: GetApiVersion | 获取 API 版本 |
| | CTacFtdcTraderApi:: Release | 删除接口实例 |
| | CTacFtdcTraderApi:: Init | 初始化 |
| | CTacFtdcTraderApi:: Join | 等待接口线程结束运行 |
| 参数管理接口 | CTacFtdcTraderApi:: RegisterSpi | 注册回调接口 |
| | CTacFtdcTraderApi:: RegisterFront | 注册前置机网络地址 |
| 订阅接口 | CTacFtdcTraderApi:: SubscribePrivateTopic | 订阅私有流 |
| 日志接口 | CTacFtdcTraderApi:: SetLogFilePath | 设置文件路径 |
| | CTacFtdcTraderApi:: SetLogLevel | 打开/关闭日志 |
| 通信状态接口 | CTacFtdcTraderSpi:: OnFrontConnected | 与交易系统建立起通信连接时（还未登录前），该方法被调用 |
| | CTacFtdcTraderSpi:: OnFrontDisconnected | 与交易系统通信连接断开时，该方法被调用 |

### 2.1.2. 业务接口

| 业务类型 | 业务 | 请求接口 / 响应接口 | 数据流 |
|---|---|---|---|
| App 认证 | App 认证 | CTacFtdcTraderApi:: ReqAuthenticate<br>CTacFtdcTraderSpi::OnRspAuthenticate | N/A |
| 登录 | 登录 | CTacFtdcTraderApi:: ReqUserLogin<br>CTacFtdcTraderSpi::OnRspUserLogin | N/A |
| | 登出 | CTacFtdcTraderApi::ReqUserLogout<br>CTacFtdcTraderSpi::OnRspUserLogout | 对话流 |
| 订阅 | 订阅主题 | CTacFtdcTraderApi::SubscribePrivateTopic | 私有流 |

| | | | |
|---|---|---|---|
| 交易 | 报单录入 | CTacFtdcTraderApi::ReqOrderInsert<br>CTacFtdcTraderSpi::OnRspOrderInsert | 对话流 |
| | 报单操作 | CTacFtdcTraderApi::ReqOrderAction<br>CTacFtdcTraderSpi::OnRspOrderAction | 对话流 |
| 私有回报 | 成交回报 | CTacFtdcTraderSpi::OnRtnTrade | 私有流 |
| | 报单回报 | CTacFtdcTraderSpi::OnRtnOrder | 私有流 |
| 查询 | 查询合约 | CTacFtdcTraderApi:: ReqQryInstrument<br>CTacFtdcTraderSpi::OnRspQryInstrument | 对话流 |
| | 查询资金账户 | CTacFtdcTraderApi:: ReqQryAccount<br>CTacFtdcTraderSpi::OnRspQryAccount | 对话流 |
| | 查询持仓 | CTacFtdcTraderApi:: ReqQryPosition | 对话流 |
| 业务类型 | 业务 | 请求接口 / 响应接口 | 数据流 |
| | | CTacFtdcTraderSpi::OnRspQryPosition | |

# 2.2.TraderAPI 参考手册

## 2.2.1. CTacFtdcTraderSpi 接口

CTacFtdcTraderSpi 实现了事件通知接口。用户必须派生 CTacFtdcTraderSpi 接口，编写事件处理方法来处理感兴趣的事件。

### 2.2.1.1. OnFrontConnected 方法

当客户系统与 TacMars 交易系统的交易前置机建立起 TCP 虚链路（连接）后，该方法被调用。该连接是由 API 自动建立的。

函数原型：

```
void OnFrontConnected();
```

注意：OnFrontConnected 被调用仅说明 TCP 连接成功，客户系统必须自行登录，才能进行后续的业务操作。登录失败不会回调该方法。

### 2.2.1.2. OnFrontDisconnected 方法

当客户系统与 TacMars 交易系统的交易前置机的 TCP 虚链路断开时，该方法被调用。当发生这个情况后，API 会自动重新连接，客户系统可不做处理。

自动重连地址，可能是原来注册的地址，也可能是系统支持的其它可用的通信地址，它由程序自动选择。

函数原型：

```
void OnFrontDisconnected (int nReason);
```

参数：

nReason：连接断开原因：

- 0x1001 网络读失败
- 0x1002 网络写失败
- 0x2003 收到错误报文

2.2.1.3. **OnRspAuthenticate** 方法

当客户系统发出 App 认证请求之后，交易系统返回响应时，该方法会被调用，通知客户系统认证是否成功。

函数原型：

```
void OnRspAuthenticate(
        CTacFtdcRspAuthenticateField
        *pRspAuthenticate, CTacFtdcRspInfoField
        *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pRspAuthenticate**：返回用户 App 认证信息的地址。

用户 App 认证信息结构：

```
struct CTacFtdcRspAuthenticateField {
    ///AppID
    TTacFtdcAppIDType   AppID;
    ///用户代码
    TTacFtdcUserIDType          UserID;
};
```

**pRspInfo**：返回用户响应信息的地址。以下同。错误代码为 0 时，表示操作成功，以下同。响应信息结构：

```
struct CTacFtdcRspInfoField {
    ///错误代码
    TTacFtdcErrorIDType ErrorID;
    ///错误信息
    TTacFtdcErrorMsgType    ErrorMsg;
};
```

可能出现的错误：

| 错误代码 | 错误提示 | 可能的原因 |
|---|---|---|
| 3 | 认证失败 | AppID 和授权码不匹配 |

**nRequestID**：返回用户登录请求的 ID，该 ID 由用户在登录时指定。

**bIsLast**：指示该次返回是否为针对 nRequestID 的最后一次返回。

#### 2.2.1.4. **OnRspUserLogin** 方法

当客户系统发出登录请求之后，交易系统返回响应时，该方法会被调用，通知客户系统登录是否成功。

函数原型：

```
void OnRspUserLogin(
        CTacFtdcRspUserLoginField
        *pRspUserLogin, CTacFtdcRspInfoField
        *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pRspUserLogin**：返回用户登录信息的地址。

用户登录信息结构：

```
struct CTacFtdcRspUserLoginField {
    ///用户代码
    TTacFtdcUserIDType UserID;
    ///最大报单引用
    TTacFtdcOrderRefType    MaxOrderRef;
    ///交易系统名称
    TTacFtdcSystemNameType SystemName;
    ///交易日
    TTacFtdcDateType    TradingDay;
    ///登录成功时间
    TTacFtdcTimeType    LoginTime;};
```

**pRspInfo**：返回用户响应信息的地址。以下同。错误代码为 0 时，表示操作成功，以下同。响应信息结构：

```
struct CTacFtdcRspInfoField {
    ///错误代码
    TTacFtdcErrorIDType ErrorID;
    ///错误信息
    TTacFtdcErrorMsgType    ErrorMsg;
};
可能出现的错误：
错误代码错误提示                        可能的原因
3       客户找不到或密码错              客户没有配置好或密码错
```

**nRequestID**：返回用户登录请求的 ID，该 ID 由用户在登录时指定。

**bIsLast**：指示该次返回是否为针对 nRequestID 的最后一次返回。

#### 2.2.1.5. **OnRspUserLogout** 方法

当客户系统发出退出请求之后，交易系统返回响应时，该方法会被调用，通知客户系统退出是否成功。

函数原型：

```
void OnRspUserLogout(
    CTacFtdcRspUserLogoutField
    *pRspUserLogout, CTacFtdcRspInfoField
    *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pRspUserLogout**：返回用户退出信息的地址。用户登出信息结构：

```
struct CTacFtdcRspUserLogoutField {
    ///用户代码
    TTacFtdcUserIDType UserID;
};
```

**pRspInfo**：返回用户响应信息的地址。响应信息结构：

```
struct CTacFtdcRspInfoField {
    ///错误代码
    TTacFtdcErrorIDType ErrorID;
    ///错误信息
    TTacFtdcErrorMsgType    ErrorMsg;
};
可能出现的错误：无
```

**nRequestID**：返回用户登出请求的 ID，该 ID 由用户在登出时指定。

**bIsLast**：指示该次返回是否为针对 nRequestID 的最后一次返回。

#### 2.2.1.6. **OnRspOrderInsert** 方法

报单录入应答。当客户系统发出过报单录入指令后，交易系统返回响应时，该方法会被调用。

函数原型：

```
void OnRspOrderInsert(
    CTacFtdcRspOrderInsertField *pRspInputOrder,
    CTacFtdcRspInfoField *pRspInfo, int
    nRequestID, bool bIsLast);
```

参数：

**pRspInputOrder**：指向报单录入结构的地址，包含了提交报单录入时的输入数据，和所系统返回的报单编号。注意：结构中的某些字段与报单录入时不同，返回为空值。输入报单结构：

```
struct CTacFtdcRspOrderInsertField {
    ///报单引用,一个客户一天内唯一
TTacFtdcOrderRefType        OrderRef;
    ///柜台订单号,一个柜台一天内唯一
TTacFtdcOrderRefType        OrderLocalID;
    ///交易所订单号，一个交易所一天内唯一
TTacFtdcOrderRefType        OrderSysID;
    ///客户代码
    TTacFtdcClientIDTypeClientID;
};
```

**pRspInfo**：指向响应信息结构的地址。响应信息结构：

```
struct CTacFtdcRspInfoField {
    ///错误代码
    TTacFtdcErrorIDType ErrorID;
    ///错误信息
        TTacFtdcErrorMsgType    ErrorMsg;
};
```

可能出现的错误：

| 错误代码 | 错误提示 | 可能的原因 |
|---|---|---|
| 2 | 合约找不到 | 找不到报单中的合约 |
| 3 | 客户找不到 | 找不到报单中的客户 |
| 4 | 客户找不到 | 找不到报单中的客户 |
| 7 | 资金不足 | 资金不足 |
| 16 | 仓位不足 | 平仓时，仓位不足 |

**nRequestID**：暂时未用。

**bIsLast**：指示该次返回是否为针对 nRequestID 的最后一次返回。

注意：

CTacFtdcRspInfoField.ErrorID 为 0，意味着本次报单录入成功。CTacFtdcInputOrderField *pInputOrder 报单失败时，只能返回 OrderRef 用于匹配客户本地的报单；如果成功，能返回 OrderSysID、OrderLocalID 和 OrderRef。该报单的具体内容需要从私有流获得。

### 2.2.1.7. **OnRspOrderAction** 方法

报单操作应答。报单操作包括报单的撤销。当客户系统发出报单操作指令后，交易系统返回响应时，该方法会被调用。

函数原型：

```
void OnRspOrderAction(
        CTacFtdcRspOrderActionField
        *pRspOrderAction, CTacFtdcRspInfoField
        *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pOrderAction**：指向报单操作结构的地址，包含了提交报单操作的输入数据，和交易系统返回的报单编号。报单操作结构：

```
struct CTacFtdcOrderActionField {
    ///报单引用。订单中的 OrderRef，在没有收到报单回报或者响应时，使用 OrderRef 撤单。
    TTacFtdcOrderRefType    OrderRef;
    ///柜台订单号。柜台流水号。建议客户使用柜台流水号撤单，效率最高。
    TTacFtdcOrderRefType    OrderLocalID;
    ///客户代码
    TTacFtdcClientIDTypeClientID;
};
```

**pRspInfo**：指向响应信息结构的地址。响应信息结构：

```
struct CTacFtdcRspInfoField {
    ///错误代码
    TTacFtdcErrorIDType ErrorID;
    ///错误信息
    TTacFtdcErrorMsgType    ErrorMsg;
};
```

可能出现的错误：

| 错误代码 | 错误提示 | 可能的原因 |
|---|---|---|
| 12 | 订单找不到 | 订单找不到 |
| 13 | 全部成交或者已经撤单 | 全部成交或者已经撤单 |
| 25 | 没有撤单权限 | 有撤单请求处理中 |

**nRequestID**：暂时未用。

**bIsLast**：指示该次返回是否为针对 nRequestID 的最后一次返回。

### 2.2.1.8. **OnRtnTrade** 方法

成交回报。当发生成交时交易系统会通知客户系统，该方法会被调用。

函数原型：

```
void OnRtnTrade(CTacFtdcRtnTradeField *pTrade);
```

参数：

**pTrade**：指向成交回报结构的地址。成交回报结构：

```
struct CTacFtdcRtnTradeField {
    ///成交编号
    TTacFtdcOrderRefType        TradeID;
    ///交易所订单号
    TTacFtdcOrderRefType        OrderSysID;
    ///柜台报单号
```

```
        TTacFtdcOrderRefType      OrderLocalID;
        ///报单引用
        TTacFtdcOrderRefType      OrderRef;
        ///客户代码
        TTacFtdcClientIDType      ClientID;
        ///合约代码
        TTacFtdcInstrumentIDType InstrumentID;
        ///买卖方向
        TTacFtdcDirectionType     Direction;
          ///开平标志
        TTacFtdcOffsetFlagType    OffsetFlag;
        ///投机套保标志
        TTacFtdcHedgeFlagType     HedgeFlag;
        ///成交价格
        TTacFtdcPriceType     Price;
        ///数量
        TTacFtdcVolumeType Volume;
        ///交易所代码
        TTacFtdcExchangeIDType   ExchangeID;
};
```

2.2.1.9. **OnRtnOrder** 方法

报单回报。当客户系统进行报单录入、报单操作及其它原因（如部分成交）导致报单状态发生变化时，交易系统会主动通知客户系统，该方法会被调用。

函数原型：

```
void OnRtnOrder(CTacFtdcRtnOrderField *pOrder);
```

参数：

**pOrder**：指向报单回报结构的地址。注意：报单回报中的某些字段未使用，交易系统返回为空值。报单回报结构：

```
struct CTacFtdcRtnOrderField {
    ///交易所订单号
    TTacFtdcOrderRefType    OrderSysID;
    ///柜台报单号
    TTacFtdcOrderRefType    OrderLocalID;
    ///报单引用
    TTacFtdcOrderRefType    OrderRef;
    ///报单状态
    TTacFtdcOrderStatusType  OrderStatus;
    ///成交数量
    TTacFtdcVolumeType VolumeTraded;
    ///剩余数量
    TTacFtdcVolumeType VolumeTotal;
    ///客户代码
    TTacFtdcClientIDTypeClientID;
    ///合约代码
    TTacFtdcInstrumentIDType InstrumentID;
    ///报单价格条件，现在只能填 TAC_FTDC_OPT_LimitPrice
    TTacFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TTacFtdcDirectionType    Direction;
    ///开平标志
    TTacFtdcOffsetFlagType    OffsetFlag;
    ///投机套保标志
    TTacFtdcHedgeFlagType    HedgeFlag;
    ///价格
    TTacFtdcPriceType    LimitPrice;
    ///有效期类型：GFD，FOK，FAK
    TTacFtdcTimeConditionType    TimeCondition;
    ///报单数量
    TTacFtdcVolumeType VolumeTotalOriginal;
    ///交易所代码
    TTacFtdcExchangeIDType  ExchangeID;
};
```

2.2.1.10.**OnRspQryInstrument** 方法

查询合约响应。当客户系统查询合约，有合约返回或者查询合约结束时，该方法会被调用。

函数原型：

```
void OnRspQryInstrument(CTacFtdcInstrumentField *pInstrument, CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pInstrument**：指向合约结构的地址。使用前，客户端程序要检查是否为空。

合约结构：

```
struct CTacFtdcInstrumentField
{
    ///合约代码
    TTacFtdcInstrumentIDType InstrumentID;
    ///交易所代码
    TTacFtdcExchangeIDType ExchangeID;
    ///限价单最大下单量
    TTacFtdcVolumeType MaxLimitOrderVolume;
    ///限价单最小下单量
    TTacFtdcVolumeType MinLimitOrderVolume;
    ///合约数量乘数
    TTacFtdcVolumeMultipleType VolumeMultiple;
    ///最小变动价位
    TTacFtdcPriceType PriceTick;
    ///前结算价
    TTacFtdcPriceType PreSettlementPrice;
    ///涨停价
    TTacFtdcPriceType UpperLimitPrice;
    ///跌停价
    TTacFtdcPriceType LowerLimitPrice;
    ///多头保证金率
    TTacFtdcRatioType LongMarginRatio;
    ///空头保证金率
    TTacFtdcRatioType ShortMarginRatio;
    ///是否使用大额单边保证金算法
    TTacFtdcBoolType MaxMarginSideAlgorithm;
    ///开仓手续费率
    TTacFtdcRatioType OpenFeeRatio;
    ///平仓手续费率
    TTacFtdcRatioType CloseFeeRatio;
    ///平今仓手续费率
    TTacFtdcRatioType CloseTodayFeeRatio;
};
```

**pRspInfo**：相应信息。使用前，客户端程序要检查是否为空。 **nRequestID**：请求 ID。暂时未用。

**bIsLast**：最后一个报文标识。如果是 true，则说明是最后一个报文。

### 2.2.1.11.**OnRspQryAccount** 方法

查询资金账户响应。客户查询资金账户时，该方法会被调用。

函数原型：

```
void OnRspQryAccount(CTacFtdcAccountField *pAccount, CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pAccount**：指向资金账户结构的地址。客户端使用前，要检查地址是否为空。资金账户结构：

```
struct CTacFtdcAccountField
{
///账户
TTacFtdcClientIDType	ClientID;
///初始资金
TTacFtdcMoneyType	InitFund;
///冻结的保证金
TTacFtdcMoneyType	FrozenMargin;
///冻结的手续费
TTacFtdcMoneyType	FrozenFee;
///当前的保证金
TTacFtdcMoneyType	CurrMargin;
///手续费
TTacFtdcMoneyType	Fee;
///平仓盈亏
TTacFtdcMoneyType	CloseProfit;
///持仓盈亏
TTacFtdcMoneyType	PositionProfit;
///可用资金
TTacFtdcMoneyType	Available;
```

```
    ///权益
    TTacFtdcMoneyType       Balance;
    ///冻结的权利金
    TTacFtdcMoneyType       FrozenPremium;
    ///权利金
    TTacFtdcMoneyType       Premium;
    ///保证金优惠额
    TTacFtdcMoneyType       Discount;
};
```

**pRspInfo**：相应信息。使用前，客户端程序要检查是否为空。 **nRequestID**：请求 ID。暂时未用。

**bIsLast**：最后一个报文标识。如果是 true，则说明是最后一个报文。

### 2.2.1.12.**OnRspQryPosition** 方法

查询持仓响应。当客户端查询持仓时，该方法会被调用。

函数原型：

```
void OnRspQryPosition(CTacFtdcPositionField *pPosition, CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

**pPosition**：指持仓结构的地址。客户端使用前，需要检查地址是否为空。持仓结构：

```
struct CTacFtdcPositionField
{
    ///合约代码
    TTacFtdcInstrumentIDType    InstrumentID;
    ///客户代码
    TTacFtdcClientIDType        ClientID;
    ///投机套保标志
    TTacFtdcHedgeFlagType       HedgeFlag;
    ///持仓多空方向
    TTacFtdcPosiDirectionType   PosiDirection;
    ///上日持仓
    TTacFtdcVolumeType          YdPosition;
```

```
    ///今日持仓
    TTacFtdcVolumeType          Position;
    ///多头冻结
    TTacFtdcVolumeType          LongFrozen;
    ///空头冻结
    TTacFtdcVolumeType          ShortFrozen;
    ///持仓总保证金
    TTacFtdcMoneyType           PositionMargin;
    ///持仓保证金优惠额
    TTacFtdcMoneyType           Discount;
};
```

**pRspInfo**：相应信息。使用前，客户端程序要检查是否为空。

**nRequestID**：请求 ID。暂时未用。

**bIsLast**：最后一个报文标识。如果是 true，则说明是最后一个报文。

### 2.2.1.13.**OnRtnClientDiscount** 方法

柜台自动组合合约对锁，并报送交易所成功后，发生保证金优惠变化时，该方法会被调用。

函数原型：

```
void OnRtnClientDiscount(CTacFtdcClientFundDiscount *pInfo);
```

参数：

**pInfo**：指向账户资金优惠信息结构体：

```
struct CTacFtdcPositionField
{
    ///客户代码
    TTacFtdcClientIDType        ClientID;
    ///账户总保证金
    TTacFtdcMoneyType           TotalUsedMargin;
    ///账户总保证金优惠额
    TTacFtdcMoneyType           TotalDiscount;
};
```

本方法与调用 ReqQryAccount 获得的资金数额相同，私有流恢复时不会再次推送。

### 2.2.1.14.**OnRtnInstrumentDiscount** 方法

柜台自动组合合约对锁，并报送交易所成功后，发生保证金优惠变化时，该方法会被调用。函数原型：

```
void OnRtnInstrumentDiscount(CTacFtdcInstrumentDiscount *pInfo);
```

参数：

**pInfo**：指向合约信息明细信息结构体：

```
struct CTacFtdcPositionField
{
    ///客户代码
    TTacFtdcClientIDType        ClientID;
    ///合约代码
    TTacFtdcInstrumentIDType    InstrumentID;
    ///多头单腿持仓
    TTacFtdcMoneyType           SingleLegLongPosition
    ///空头单腿持仓
    TTacFtdcVolumeType          SingleLegShortPosition;
    ///多头组合持仓
    TTacFtdcVolumeType          CombineLongPosition;
    ///空头组合持仓
    TTacFtdcVolumeType          CombineShortPosition;
    ///多头总保证金
    TTacFtdcMoneyType           TotalLongUsedMargin;
    ///多头保证金优惠额
    TTacFtdcMoneyType           TotalLongDiscount;
    ///空头总保证金
    TTacFtdcMoneyType           TotalShortUsedMargin;
    ///空头保证金优惠额
    TTacFtdcMoneyType           TotalShortDiscount;
};
```

本方法与调用 ReqQryPosition 接口返回的资金数额相同，私有流恢复时不会再次推送。

#### 2.2.1.15.**OnUdpMsgRej** 方法

检测到 Udp 丢包。

函数原型：

```
void OnUdpMsgRej(int udpID);
```

参数：

**udpID**：丢失的 Udp 包的 ID，用于和 ReqOrderInsert 的返回值做比对，范围 0~65535，溢出回环。

### 2.2.2. **CTacFtdcTraderApi 接口**

CTacFtdcTraderApi 接口提供给用户的功能包括，报单与报价的录入、报单的撤销等功能。

#### 2.2.2.1. CreateFtdcTraderApi 方法

产生一个 CTacFtdcTraderApi 的一个实例，不能通过 new 来产生。函数原型：

```
static CTacFtdcTraderApi *CreateFtdcTraderApi(const char *pszFlowPath = "");
```

参数：

**pszFlowPath**：常量字符指针，用于指定一个文件目录来存储交易系统发布消息的状态。 默认值代表当前目录。返回值：

返回一个指向 CTacFtdcTraderApi 实例的指针。

关于流文件的更多说明详见 2.2.2.13

#### 2.2.2.2. GetApiVersion 方法

获取 API 版本号。

函数原型：

```
const char *GetApiVersion() ;
```

参数：**None**

返回值：返回一个指向版本标识字符串的常量指针。

2.2.2.3. Release 方法

释放一个 CTacFtdcTraderApi 实例。不能使用 delete 方法。函数原型：

```
void Release();
```

2.2.2.4. Init 方法

使客户系统与交易系统建立连接，连接成功后可以进行登录。

函数原型：

```
void Init();
```

2.2.2.5. Join 方法

客户系统等待一个接口实例线程的结束。

函数原型：

```
void Join();
```

2.2.2.6. GetTradingDay 方法

获得当前交易日。只有成功登录到交易系统之后才会取到正确的值。

函数原型：

```
const char *GetTradingDay();
```

返回值：返回一个指向日期信息字符串的常量指针。

2.2.2.7. RegisterSpi 方法

注册一个派生自 CTacFtdcTraderSpi 接口类的实例，该实例将完成事件处理。

函数原型：

```
void RegisterSpi(CTacFtdcTraderSpi *pSpi) ;
```

参数：

pSpi：实现了 CTacFtdcTraderSpi 接口的实例指针。

2.2.2.8. RegisterFront 方法

设置交易所前置系统的网络通讯地址。该方法要在

Init 方法之前调用。

函数原型：

```
void RegisterFront(const char *pszFrontAddress);
```

参数：

**pszFrontAddress**：指向柜台系统网络通讯地址的指针。服务器地址的格式为：“protocol://ipaddress:port”，如：”tcp://127.0.0.1:17001”。 “tcp”代表传输协议，“127.0.0.1”代表服务器地址。”17001”代表服务器端口号。

2.2.2.9. SetLogFilePath 方法

设置日志文件路径。用户必须有权限在该路径下创建文件。该参数必须是包含文件名称的全路径。函数原型：

```
virtual int SetLogFilePath (const char *pszLogFilePrefix);
```

参数：

**pszLogFilePrefix**：日志文件名。

2.2.2.10.SetLogLevel 方法

设置记录日志的开关。函数原型：

```
virtual int SetLogLevel (bool isLogMsg);
```

参数：

**isLogMsg**：True 开始记录日志；False 停止记录日志。

2.2.2.11.SetTraderApiCpuAffinity 方法

设置 cpu 亲和性。

函数原型：

```
virtual int SetTraderApiCpuAffinity(int recv_cpu, int send_cpu);
```

参数：

**cpu1**：spi 线程绑定的 core id，设为-1 则不绑定。不绑定时不会占用 100% CPU

**cpu2**：下单辅助线程绑定的 core id，设为-1 则不绑定。不绑定时不会占用 100%CPU

建议：recv_cpu、send_cpu 都绑定 CPU 核心，如对接收信息的速度无要求，想节省 cpu，recv_cpu 可设定为-1。

2.2.2.12.SetUseAsyncRequest 方法

设置是否使用异步发送报单请求。函数原型：

```
virtual int SetTraderApiCpuAffinity(bool isAsyncSend);
```

参数：

**isAsyncSend**：为 True 则使用异步发送报单请求，此模式吞吐量较大。为 False 则使用同步方式发送请求，不再占用 SetTraderApiCpuAffinity 中 send_cpu 所绑定的核。

2.2.2.13.SubscribePrivateTopic 方法

订阅会员私有流。该方法要在 Init 方法前调用。若不调用则不会收到私有流的数据。函数原型：

```
void SubscribePrivateTopic(TE_RESUME_TYPE nResumeType);
```

参数：

**nResumeType** 私有流重传方式：

- TAC_TERT_RESTART：从本交易日开始重传；
- TAC_TERT_RESUME：从上次收到的续传。
- TAC_TERT_QUICK：只传送登录后客户私有流的内容。

注意：

如果使用 TAC_TERT_RESUME 模式，每个交易日开盘前，都需要删除流文件。流文件默认存放在报单工具下的 flow 文件夹内。

2.2.2.14.ReqAuthenticate 方法

用户发出 App 认证请求。

函数原型：

```
int ReqAuthenticate(
        CTacFtdcReqAuthenticateField *pAuthenticate,
        int nRequestID);
```

参数：

**pAuthenticate**：指向用户 App 认证请求结构的地址。用户 App 认证请求结构：

```
struct CTacFtdcAuthenticateField {
    ///AppID
    TTacFtdcAppIDType          AppID;
    ///授权码
    TTacFtdcAuthCodeType       AuthCode;
    ///用户代码
TTacFtdcUserIDType                  UserID;
};
```

**nRequestID**：用户登录请求的 ID，该 ID 由用户指定，管理。

返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -4, 表示非法参数；

2.2.2.15.ReqUserLogin 方法

用户发出登录请求。调用该方法之前需先调用 ReqAuthenticate 方法。需求多用户登录，可以通过多次调用 ReqUserLogin 方法实现。

函数原型：

```
int ReqUserLogin(
    CTacFtdcReqUserLoginField *pUserLoginField,
    int nRequestID);
```

参数：

**pUserLoginField**：指向用户登录请求结构的地址。用户登录请求结构：

```
struct CTacFtdcUserLoginField {
    ///用户代码
    TTacFtdcUserIDType UserID;
    ///密码
    TTacFtdcPasswordType       Password;
```

```
        ///IP 地址
        TTacFtdcIPAddressType      IPAddress;
        ///Mac 地址
        TTacFtdcMacAddressType   MacAddress;
        ///是否启用 UDP 回单，填
        TAC_FTDC_Enable_Udp_Response/TAC_FTDC_Unable_Udp_Response
         EnableUdpResponse   TAC_FTDC_Enable_Udp_Response
         //如果启用回单，填监听的端口
         UdpPort   UdpPort
         启用回单，填监听的 UDP IP 地址
         UdpIP     UdpIP
         启用回单，填监听的 UDPMAC 地址
         UdpMac 可不填
};
```

**nRequestID**：用户登录请求的 ID，该 ID 由用户指定，管理。

返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -4, 表示非法参数；
- -5, 表示等待登录响应；
- -6，表示流文件建立失败

2.2.2.16.ReqUserLogout 方法

用户发出登出请求。

函数原型：

```
int ReqUserLogout(
        CTacFtdcReqUserLogoutField *pReqUserLogout,
        int nRequestID);
```

参数：

**pReqUserLogout**：指向用户登出请求结构的地址。用户登出请求结构：

```
struct CTacFtdcUserLogoutField {
    ///用户代码
    TTacFtdcUserIDType UserID;
```

```
};
```

**nRequestID**：用户登出请求的 ID，该 ID 由用户指定，管理。

返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -3，表示用户未登录
- -4, 表示非法参数；

2.2.2.17.ReqOrderInsert 方法

客户系统发出报单录入请求。

函数原型：

```
int ReqOrderInsert(
        CTacFtdcInputOrderField *pInputOrder,
        int nRequestID);
```

参数：

**pInputOrder**：指向输入报单结构的地址，输入报单结构：

```
struct CTacFtdcInputOrderField {
        ///合约代码
        TTacFtdcInstrumentIDType InstrumentID;
        ///报单价格条件：现在只能是 TAC_FTDC_OPT_LimitPrice
        TTacFtdcOrderPriceTypeType      OrderPriceType;
        ///买卖方向
        TTacFtdcDirectionType       Direction;
        ///开平标志
        TTacFtdcOffsetFlagType      OffsetFlag;
        ///价格
        TTacFtdcPriceType     LimitPrice;
        ///数量
        TTacFtdcVolumeType VolumeTotalOriginal;
        ///有效期类型：现在只能是 GFD，FOK，FAK
        TTacFtdcTimeConditionType       TimeCondition;
        ///报单引用,在开启 udp 时只支持 uint32_t 的数值上限
```

```
    TTacFtdcOrderRefType    OrderRef;
    ///客户代码
    TTacFtdcClientIDTypeClientID;
    ///投机套保标志
    TTacFtdcHedgeFlagType   HedgeFlag;
    ///交易所代码
    TTacFtdcExchangeIDType  ExchangeID;
};
```

**nRequestID**：暂时未用。

返回值：

- 大于等于 0，成功，开启 udp 报单时返回 udp 索引，范围(0~65535)，用于和 OnUdpMsgRej 中的 ID 比对；
- -1，表示网络发送失败；
- -2, 表示流控超限被拒绝；
- -3, 表示用户未登录；
- -4, 表示非法参数；
- -5, 报单合约未找到；

2.2.2.18.ReqOrderAction 方法

客户系统发出撤销报单请求。

函数原型：

```
int ReqOrderAction(
    CTacFtdcOrderActionField *pOrderAction,
    int nRequestID);
```

参数：

**pOrderAction**：指向报单操作结构的地址。报单操作结构：

```
struct CTacFtdcOrderActionField {
    ///原报单引用:填写原报单的 OrderRef；填‘0’表示不引用 OrderRef
    TTacFtdcOrderRefType    OrderRef;
    ///原柜台订单号：填写原报单的 OrderLocalID。填‘0’表示不引用 OrderLocalID
    TTacFtdcOrderRefType    OrderLocalID;
    ///客户代码:必填。
    TTacFtdcClientIDTypeClientID;
};
```

**nRequestID**：暂时未用。

返回值：

- 大于等于 0，成功，udp 报单时返回 udp 索引，范围(0~65535)，用于和 OnUdpMsgRej 中的 ID 比对；
- -1，表示网络发送失败；
- -2, 表示流控超限被拒绝；
- -3, 表示用户未登录；
- -4, 表示非法参数；注意：

撤销报单时优先取 OrderLocalID，OrderLocalID 为零则取 OrderRef。

2.2.2.19.ReqQryInstrument 方法

客户系统发出查询合约请求。

函数原型：

```
int ReqQryInstrument (
    CTacFtdcQryInstrumentField *pQryInstrument,
    int nRequestID);
```

参数：

**pQryInstrument**：指向查询合约结构的地址。查询合约结构：

```
struct CTacFtdcQryInstrumentField
{
    ///交易所代码：必填
TTacFtdcExchangeIDType          ExchangeID;
    ///合约代码:：必填填写具体合约代码，查询指定合约；填空值（'\0'），查询所有合约
TTacFtdcInstrumentIDType        InstrumentID;
    ///客户代码：必填
TTacFtdcClientIDType            ClientID;
};
```

**nRequestID**：暂时未用。

返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -2, 表示流控超限被拒绝；
- -3, 表示用户未登录；
- -4, 表示非法参数；

2.2.2.20.ReqQryAccount 方法

客户系统发出查询账户请求。

函数原型：

```
int ReqQryAccount (
    CTacFtdcQryAccountField *pQryAccount,
    int nRequestID);
```

参数：

**pQryAccount**：指向查询账户结构的地址。查询账户结构：

```
struct CTacFtdcQryAccountField
{
    ///客户代码：必填
    TTacFtdcClientIDType
};                              ClientID;
```

**nRequestID**：暂时未用。

返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -2, 表示流控超限被拒绝；
- -3, 表示用户未登录；
- -4, 表示非法参数；

2.2.2.21.ReqQryPosition 方法

客户系统发出查询持仓请求。

函数原型：

```
int ReqQryPosition (
        CTacFtdcQryPositionField *pQryPosition,
        int nRequestID)；
```

参数：

**pQryAccount**：指向查询持仓结构的地址。查询持仓结构：

```
struct CTacFtdcQryPositionField
{
    ///客户代码：必填
    TTacFtdcClientIDType            ClientID;
    ///合约代码:填写单一合约则返回指定合约持仓，填空（'\0'）则返回所有合约持仓。
    TTacFtdcInstrumentIDType        InstrumentID;
};
```

**nRequestID**：暂时未用。返回值：

- 0，代表成功。
- -1，表示网络发送失败；
- -2, 表示流控超限被拒绝；
- -3, 表示用户未登录；
- -4, 表示非法参数；

2.2.2.22.SetUdpSendOpen 方法

使用 Udp 协议发送，可以增加发送速度，但是有可能丢包，默认关闭。

函数原型：

**bool SetUdpSendOpen (bool isOpen);**

参数：

**isOpen:** True 使用 udp 发送；False 使用 tcp 发送。

2.2.2.23.SetOpenFastSend 方法

调整底层行为，在某些特定硬件上可以增加发送速度，但是收包速度会稍微下降，默认开启。

函数原型：

**bool SetOpenFastSend (bool isOpen);**

参数：

**isOpen:** True 打开发包加速；False 关闭发包加速。

## 2.3.TraderAPI 开发示例

```
// tradeapitest.cpp :
// 一个简单的例子，介绍 CTacFtdcTraderApi 和 CTacFtdcTraderSpi 接口的使用。
// 本例将演示一个报单录入操作的过程
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/eventfd.h>
#include <unistd.h>

#include "TacFtdcTraderApi.h"
#include "trade_demo.h"

#define TacmarsFrontAddress "tcp://127.0.0.1:9998"
#define ClientAppID          "abc_123_!@#"
#define ClientAuthCode       "123456"
#define ClientUserID         "1001"
#define ClientPassword       "123123"
#define ClientLogFilePath    "MyClientLog_" ClientUserID
```

```
int requestSeq = 0;
int event = eventfd(0, EFD_SEMAPHORE);

SimpleHandler::SimpleHandler(CTacFtdcTraderApi* pApi, const char* app, const char*
auth, const char* user, const char* pass) : m_pApi(pApi)
{ strncpy(m_app, app, sizeof(m_app));
    strncpy(m_auth, auth, sizeof(m_auth));
    strncpy(m_user, user, sizeof(m_user));
    strncpy(m_pass, pass, sizeof(m_pass));
}

void SimpleHandler::OnFrontConnected()
{ printf("Front %s connected\n", TacmarsFrontAddress);

    CTacFtdcAuthenticateField req = {};

    // Set APP ID
    strncpy(req.AppID, m_app, sizeof(req.AppID));

    // Set Authenticate Code strncpy(req.AuthCode,
    m_auth, sizeof(req.AuthCode));

    // Set User ID
    strncpy(req.UserID, m_user, sizeof(req.UserID));

    // Submit Login request m_pApi->ReqAuthenticate(&req,
    requestSeq++);
}

void SimpleHandler::OnFrontDisconnected(int nReason)
{ printf("Front %s disconnected, Reason %d\n", TacmarsFrontAddress, nReason);
}

///错误应答
void SimpleHandler::OnRspError(CTacFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast)
{ printf("OnRspError, ErrorID=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
}

void SimpleHandler::OnRspAuthenticate(CTacFtdcRspAuthenticateField
*pRspAuthenticate, CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{ if (pRspInfo->ErrorID != 0)
```

```
    { printf("Authentication Failed\n");
        exit(-1);
    }

    CTacFtdcUserLoginField req = {};

    // Set Client ID strncpy(req.UserID, m_user,
    sizeof(req.UserID));

    // Set Client password strncpy(req.Password, m_pass,
    sizeof(req.Password));

    // Submit Login request m_pApi->ReqUserLogin(&req,
    requestSeq++);
}

///用户登录应答
void SimpleHandler::OnRspUserLogin(CTacFtdcRspUserLoginField *pRspUserLogin,
CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{ printf("OnRspUserLogin:\n"); printf("ErrorID=[%d], ErrorMessage=[%s],
    User=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg, pRspUserLogin-
    >UserID);

    if (pRspInfo->ErrorID != 0)
    { printf("Login Failed\n");
        exit(-1);
    }

    CTacFtdcInputOrderField req = {};

    // Set instrument ID strcpy(req.InstrumentID,
    "ag1806");

    // Set price type req.OrderPriceType =
    TAC_FTDC_OPT_LimitPrice;

    // Set order direction
    req.Direction = TAC_FTDC_D_Buy;

    // Set order offset req.OffsetFlag
    = TAC_FTDC_OF_Open;
```

```
    // Set price
    req.LimitPrice = 10000.0f;

    // Set order volume
    req.VolumeTotalOriginal = 1;

    // Set time condition
    req.TimeCondition = TAC_FTDC_TC_GFD;

    // Set the client local order reference ID
    req.OrderRef = 1;

    // Set client ID which insert the order. // It
    should be the same with the login client ID
    strcpy(req.ClientID, m_user);

    // Set hedge flag req.HedgeFlag =
    TAC_FTDC_HF_Speculation;

    // Set exchange ID req.ExchangeID =
    TAC_FTDC_EXID_SHFE;

    // Submit the order to Tacmars int ret = m_pApi-
    >ReqOrderInsert(&req, requestSeq++);

    if (ret < 0)
    { printf("ReqOrderInsert Error, return value[%d]\n", ret); }
}

///用户登出应答
void SimpleHandler::OnRspUserLogout(CTacFtdcUserLogoutField *pUserLogout,
CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{ printf("OnRspUserLogout:\n"); printf("ErrorID=[%d],
    ErrorMessage=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);

    eventfd_write(event, 1);
}

///报单录入应答
void SimpleHandler::OnRspOrderInsert(CTacFtdcRspOrderInsertField *pRspOrderInsert,
CTacFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
```

```
{ printf("OnRspOrderInsert:\n"); printf("ErrorID=[%d],
    ErrorMessage=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);

    if (pRspInfo->ErrorID != 0)
    { printf("OrderInsert Failed\n");
        exit(-1);
    }
}

///报单回报
void SimpleHandler::OnRtnOrder(CTacFtdcRtnOrderField *pOrder)
{ printf("OnRtnOrder:\n");
    printf("OrderRefID=[%ld], OrderLocalID=[%ld], OrderSysID=[%ld],
OrderStatus=[%c]\n", pOrder->OrderRef, pOrder->OrderLocalID, pOrder-
          >OrderSysID,
pOrder->OrderStatus); }

///成交回报
void SimpleHandler::OnRtnTrade(CTacFtdcRtnTradeField *pTrade)
{ printf("OnRtnTrade:\n");
    printf("OrderRefID=[%ld], OrderLocalID=[%ld], OrderSysID=[%ld], TradeID=[%ld],
Price=[%f], Volume=[%d]\n", pTrade->OrderRef, pTrade->OrderLocalID,
pTrade->OrderSysID, pTrade->TradeID, pTrade->Price, pTrade->Volume);
}

int main()
{
    CTacFtdcTraderApi* pUserApi = CTacFtdcTraderApi::CreateFtdcTraderApi("flow");

    SimpleHandler spiHandler(pUserApi, ClientAppID, ClientAuthCode, ClientUserID,
ClientPassword);

    if (pUserApi == nullptr)
    { printf("Create CTacFtdcTraderApi failed\n");
        exit(-1);
    }

    printf("------%s------\n", pUserApi->GetApiVersion());

    pUserApi->RegisterSpi(&spiHandler);

    pUserApi->RegisterFront(TacmarsFrontAddress);
```

```
    pUserApi->SubscribePrivateTopic(TAC_TERT_RESUME);

    pUserApi->SetLogFilePath(ClientLogFilePath);

    pUserApi->SetLogLevel(true);

    pUserApi->SetTraderApiCpuAffinity(1, 2);

    // Start worker thread and init api after setup configuration pUserApi->Init();

    // Logger thread starts in 2 seconds, so sleep sometime to wait log output
    sleep(3);

    CTacFtdcUserLogoutField req = {};

    // Set Client ID strcpy(req.UserID,
    ClientUserID);

    // Submit Logout request pUserApi->ReqUserLogout(&req,
    requestSeq++);

    eventfd_read(event, nullptr);

    pUserApi->Release();
}
```

# 第三章 附录

## 3.1.错误编码列表

| 错误编号 | 错误提示 | 错误原因 |
|---|---|---|
| 0 | Success | 成功 |
| 1 | OrderServer Offline | 席位没有登录交易所 |
| 2 | Instrument Not Found | 合约找不到 |
| 3 | Client Not Found | 客户找不到 |
| 4 | Client Not Available | 客户无效，暂时不能用 |
| 5 | Check Open Limit Fails | 不能开仓 |
| 6 | Check Cancel Limit Fails | 不能撤单。找不到订单或者已经有撤单请求在处理。 |
| 7 | Check Fund Fails | 开仓资金不足 |
| 8 | Check Order Price Tick Fails | 价格最小变动价位不正确 |
| 9 | Check Order Price Limits Fails | 超过涨跌停板 |
| 10 | Check Order Size Fails | 报单量错误 |
| 11 | Internal Update Fails | 柜台内部更新失败 |
| 12 | Order Not Found | 订单找不到 |
| 13 | Order Status Unable to Cancel | 订单状态不对，不能撤单。 |
| 14 | Order Place Args Check Fails | 下单参数检查失败 |
| 15 | Remove Mode Exceeds Frozen Close Count | 移除订单超过冻结平仓数 |
| 16 | Check Close Position Fails | 平仓持仓不足 |
| 17 | Not Market Time for This Instrument | 非市场交易时间 |
| 18 | Funding Account Not Found | 资金账户找不到 |
| 19 | Funding Account Not Allowed to Trade | 资金账户不能交易 |
| 20 | Check Place Permission Fails | 没有下单权限 |
| 21 | Exchange Not Found | 找不到交易所 |
| 22 | Trading Account Not Found | 交易编码找不到 |
| 23 | Trading Account Not Allowed to Trade | 交易编码没有交易权限 |
| 24 | Product Not Found | 产品找不到 |
| 25 | Check Cancel Permission Fails | 没有撤单权限 |
| 26 | Order Attached Args Not Found | 下单参数错误 |
| 27 | OrderServer DisableTrading | 柜台不能交易 |
| 28 | Invalid OrderRef | 订单的 OrderRef 非法 |
| 29 | Input Field Error | 报单域填写错误 |
| 30 | IOC Order Only Apply to Continous Trading | IOC 需在连续交易阶段 |

| | | |
|---|---|---|
| 31 | GFA Order Only Apply to Auction Trading | GFA 需在竞价阶段 |
| 32 | Market Order Can Not Queue | 市价单不能排队 |
| 33 | GTD Order Expired | GTD 报单过期了 |
| 34 | Client Position Limit Exceed | 超出客户限仓 |
| 35 | Close Only | 只能平仓 |
| 36 | Order Had Supended | 报单已被挂起 |
| 37 | Order Had Actived | 报单已被激活 |
| 38 | GTD Order Date Missed | GTD 报单没有设定日期 |
| 39 | Unsupported Order Type | 不被支持的报单类型 |
| 40 | Stop Order Only Apply to Continuous Trading | 止损单仅用于连续交易 |
| 41 | Stop Order Must be IOC or GFD | 止损单需是 IOC 或 GFD |
| 42 | Stop Order Must Have Stop Price | 止损报单需说明止损价 |
| 43 | Not Enough Hedge Volume | 保值额度不足 |
| 44 | Can Not Close Today For Hedge | 当日套保仓位不能平仓 |
| 45 | Market Maker Cannot Do Hedge | 做市商不能进行套保 |
| 46 | Too Many Stop Order | 太多止损单 |
| 47 | Unmatched ClientID | ClientID 不匹配 |
| 48 | Out Of Valid Price Range | 报单价格不在价格波动带范围内 |
| 49 | InvalidInput | 非法字段 |
| 50 | InvalidAppID | AppID 不存在 |
| 51 | AuthenticationFail | AppID 和授权码不匹配 |

## 3.2 登录错误编码列表

| 错误编号 | 错误提示 | 错误原因 |
|---|---|---|
| 49 | InvalidUser | 用户不存在 |
| 50 | WrongPassword | 密码错误 |
| 51 | AlreadyLogin | 用户已登录 |
| 52 | NotSupport | API 版本不支持 |
| 54 | OrderServerOffline | 席位未登录/断开 |
| 55 | AppNotAuthenticate | 登录前未进行认证 |