版本说明
2022/09/05 初稿
2022/09/09 调整onOrder接口；新增onCancelOrder接口；
2022/09/19 增加配置文件部分线程运行模式说明；增加部分常见问题说明；

用法概览

以下是使用API的基本流程：

1. 配置API文件；
2. 启动应用；
3. 创建API对象；
4. 启动API对象；
5. 查询合约；
6. 发送报单；
7. 处理报单回报；
8. 处理成交回报；
9. 撤销报单；
10. 停止API对象；
11. 退出应用；

用法示例伪代码：

```
int main(int argc, const char *argv[]) {
    std::cout << "xtf api version: " << getXTFVersion() << std::endl;

    // 传入配置文件路径，创建API对象
    XTFApi *api = makeXTFApi("./xtf_trader_api.config");
    if (api == nullptr) {
        std::cout << "xtf api make failed." << std::endl;
        return -1;
    }

    // 创建SPI回调处理对象
    XTFTrader spi(api);

    // 启动API对象，启动成功后会自动调用spi->onStart()接口
    int ret = api->start(&spi);
    if (ret != 0) {
        printf("api start failed, error code: %d\n", ret);
        exit(0);
    }
    sleep(1);

    // 登录柜台，登录结果在spi->onLogin()接口中返回
    ret = api->login();
    if (ret != 0) {
        printf("api login failed, error code: %d\n", ret);
        exit(0);
    }
    sleep(1);

    // 创建报单对象
    XTFInputOrder inputOrder;
    inputOrder.localOrderID = 1; // 建议使用本地唯一的编号
    inputOrder.direction = XTF_D_Buy;
    inputOrder.offsetFlag = XTF_OF_Open;
    inputOrder.orderType = XTF_ODT_FAK;
    inputOrder.price = 326.20;
    inputOrder.volume = 1;
    inputOrder.channelSelectionType = XTF_CS_Auto;
    inputOrder.channelID = 0;
    inputOrder.orderFlag = XTF_ODF_Normal;
```

```
    // 查询可用的合约对象
    inputOrder.instrument = api->getInstrumentByID("au2212");
    if (inputOrder.instrument == nullptr) {
        printf("instrument not found\n");
        exit(0);
    }

    // 发送报单
    ret = api->insertOrder(inputOrder);
    if (ret != 0) {
        printf("api insert order failed, error code: %d\n", ret);
        exit(0);
    }
    sleep(1);

    // 登出柜台
    ret = api->logout();
    if (ret != 0) {
        printf("api logout failed, error code: %d\n", ret);
        exit(0);
    }
    sleep(1);

    // 停止API对象，API对象停止后，api即失效，不能再次使用。
    ret = api->stop();
    if (ret != 0) {
        printf("api stop failed, error code: %d\n", ret);
        exit(0);
    }

    api = nullptr;
    sleep(1);

    std::cout << "xtf api exit." << std::endl;
    return 0;
}
```

配置API

API提供配置文件的方式创建实例和设置运行参数。配置文件内容如下：

```
############################################################
# 创建API实例时，请使用独立的配置文件。
# 多个账号请创建多个API实例，每个API实例使用各自不同的配置文件。

# 资金账号
ACCOUNT_ID=41008313_1

# 账号密码
ACCOUNT_PWD=111111

# 看穿式监管需要的APP_ID
APP_ID=1234

# 看穿式监管需要的授权码
AUTH_CODE=1234

# 查询使用的地址和端口（TCP）
QUERY_SERVER_IP=192.168.4.63
QUERY_SERVER_PORT=33333

# 交易使用的地址和端口（UDP）
TRADE_SERVER_IP=192.168.4.63
TRADE_SERVER_PORT=62000

# 回报数据处理线程绑核，默认不绑核，该线程以busy loop模式运行
# 回报数据处理线程的绑核，可以加快数据的接收和处理，建议为每个API实例绑定一个隔离的CPU核。
```

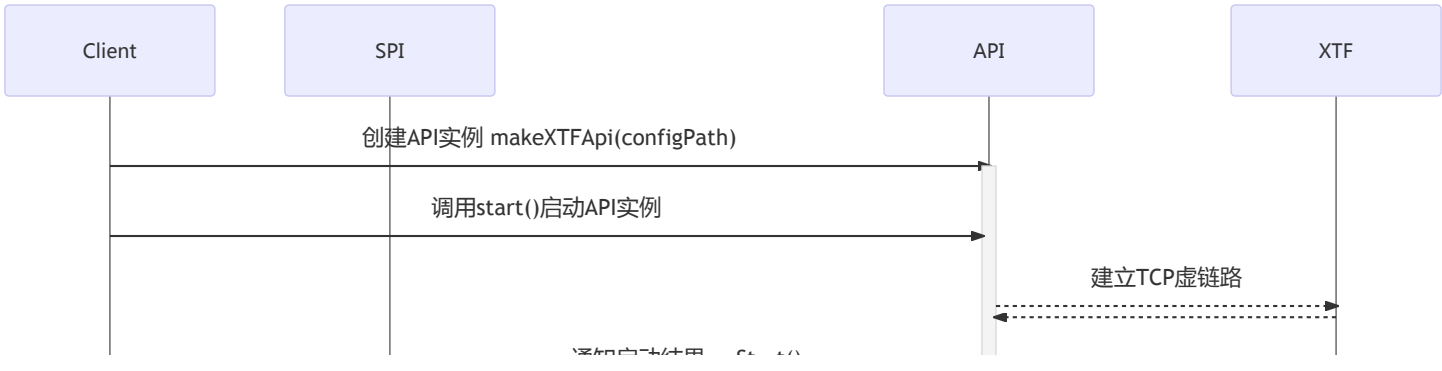
```
#TCP_WORKER_CORE_ID=3

# UDP预热处理时间间隔，单位：毫秒，取值范围：[10,50]
WARM_INTERVAL=20

# UDP预热处理线程绑核，默认不绑核，该线程不是以busy loop模式运行
# 建议将预热线程与用户策略线程绑在同一个物理CPU上
#TRADE_WORKER_CORE_ID=4

# 公共处理线程绑核，默认不绑核，该线程不是以busy loop模式运行
# 公共处理线程主要用于处理通用任务，例如：
# - 慢速数据处理；
# - 心跳超时维护；
# - 超时重连处理；
# - 其他任务，等；
# 说明：
# - 负数表示不绑核；
# - 公共处理线程是多个API实例共用的，只需要在不同的API实例配置文件中设置一次即可，以第一个API实例创建的配置生效；
#TASK_WORKER_CORE_ID=5
```

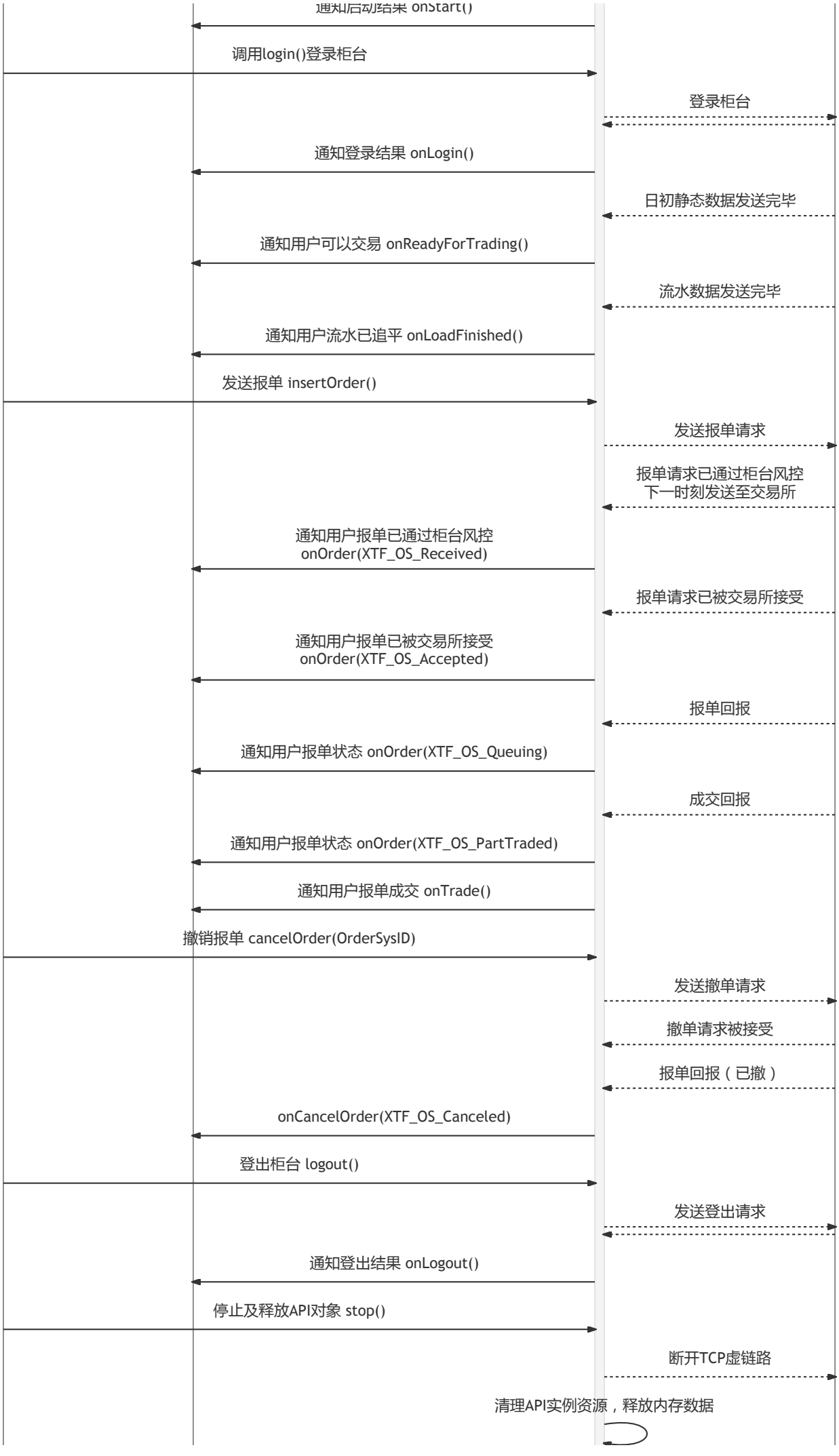
以下参数是必选配置，需要用户根据实际的内容进行编辑配置：

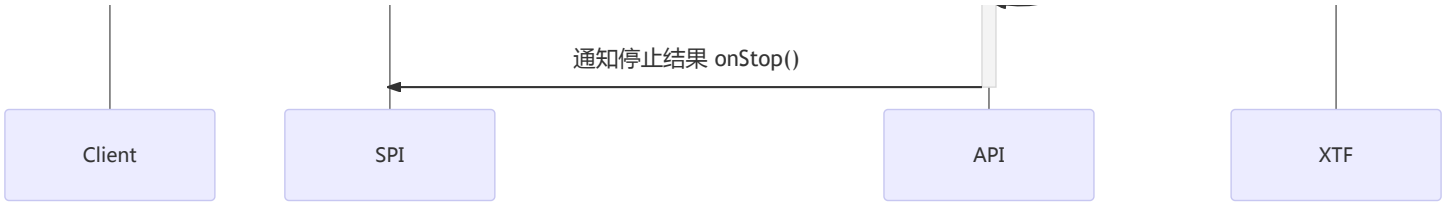
- ACCOUNT_ID
- ACCOUNT_PWD
- APP_ID
- AUTH_CODE
- QUERY_SERVER_IP
- QUERY_SERVER_PORT
- TRADE_SERVER_IP
- TRADE_SERVER_PORT

事件时序图

调用不同的API接口，会触发不同的事件回调。下图是一个API生命周期内的核心事件回调时序图：




通知启动结果 onStart()
调用login()登录柜台
登录柜台
通知登录结果 onLogin()
日初静态数据发送完毕
通知用户可以交易 onReadyForTrading()
流水数据发送完毕
通知用户流水已追平 onLoadFinished()
发送报单 insertOrder()
发送报单请求
报单请求已通过柜台风控
下一时刻发送至交易所
通知用户报单已通过柜台风控
onOrder(XTF_OS_Received)
报单请求已被交易所接受
通知用户报单已被交易所接受
onOrder(XTF_OS_Accepted)
报单回报
通知用户报单状态 onOrder(XTF_OS_Queuing)
成交回报
通知用户报单状态 onOrder(XTF_OS_PartTraded)
通知用户报单成交 onTrade()
撤销报单 cancelOrder(OrderSysID)
发送撤单请求
撤单请求被接受
报单回报（已撤）
onCancelOrder(XTF_OS_Canceled)
登出柜台 logout()
发送登出请求
通知登出结果 onLogout()
停止及释放API对象 stop()
断开TCP虚链路
清理API实例资源，释放内存数据



创建API

接口定义：XTFApi* makeXTFApi(const char *configPath)；

接口功能：创建一个API实例。API实例的参数配置由configPath指定的配置文件确定。

有两种方式创建API实例：

1. 配置文件方式，需要传入配置文件路径。如果传入的路径打开失败，那么创建API实例失败，返回空指针；
2. 参数设置方式，不需要传入配置文件路径。调用函数时配置文件路径参数传入空指针（nullptr）即可。
   创建成功后，需要通过API->setConfig()接口设置以下必选参数：
   "ACCOUNT_ID"：资金账户ID
   "ACCOUNT_PWD"：资金账户密码
   "APP_ID"：应用程序ID
   "AUTH_CODE"：认证授权码
   "TRADE_SERVER_IP"：交易服务地址
   "TRADE_SERVER_PORT"：交易服务端口
   "QUERY_SERVER_IP"：查询服务地址
   "QUERY_SERVER_PORT"：查询服务端口
   详细内容请参考API->setConfig()接口注释说明。

启动API

使用API对象之前，需要启动API。

接口定义：

```
int start(XTFSpi *spi);
```

接口功能：启动API，传入XTFSpi回调对象。

API启动后，会分配所需的各类资源，并自动向配置文件中设置的柜台地址发起连接。在一切准备就绪后，调用XTFSpi->onStart()接口，通知用户启动成功与否。

停止API

使用API结束，需要停止API，释放相关资源。

接口定义：

```
int stop();
```

接口功能：停止API。

API停止后，会自动断开与柜台的TCP连接，并释放所有相关的资源。

API停止后，API实例对象不再有效，不可使用。

登录柜台

API启动后，用户并没有登录柜台。需要调用登录接口，向柜台发起登录。

接口定义：

```
int login();
int login(const char *accountID, const char *password);
int login(const char *accountID, const char *password, const char *appID, const char *authCode);
```

接口功能：登录柜台。

登录接口默认是没有参数的，登录的用户和密码等参数，默认从配置文件读取。为方便使用，增加了两个重载接口，可以设置登录用户和密码等参数。

需要说明的是：每个API实例，只能对应一个用户登录。如果API登出后，使用另外的用户再次登录，将会报用户错误。例如：

```
api->login("abc", "111111"); // OK
api->logout(); // OK
api->login("cde", "222222"); // Error，每个API实例，只能对应一个用户登录。
```

登出柜台

接口定义：

```
int logout();
```

接口功能：登出柜台。

登出柜台后，用户不能报单和查询等操作。必须重新登录后，才能进行下一步操作。

登录柜台后，API与柜台之间的TCP连接依然保留，API内部保留了用户的所有数据，并没有释放。调用stop()接口，才真正的释放资源。

报单流程

登录柜台后，柜台返回用户相关的所有数据。当数据准备好之后，API调用onReadyForTrading()接口通知用户，可以开始报单。

> 注意：此时用户虽可以报单，但当日的流水数据并没有接收完毕。如果需要计算仓位和资金，还需要等待流水数据接收完毕，才能处理。

当流水数据接收完毕后，API调用onLoadFinished()接口通知用户，表明流水数据已追平，可以计算仓位和资金。

> 建议：如果用户不关心流水，则收到onReadyForTrading()接口之后，即可进行报单操作。
>
> 反之，应该等待 onLoadFinished() 接口之后，再进行报单操作。

插入报单

接口定义：

```
int insertOrder(const XTFInputOrder &inputOrder);
```

接口说明：向柜台发送一个报单。

返回值：如果报单发送成功，则返回0；否则，返回非0值。

发送报单，需要构造报单对象。报单对象的定义如下：

```
class XTFInputOrder {
public:
    XTFLocalOrderID         localOrderID;           ///< 本地报单编号（用户）
                                                    ///< 本地报单编号需要由用户保证唯一性，用于本地存储索引。
                                                    ///< 注意不能与柜台保留的几个特殊ID冲突：
                                                    ///< 1. 非本柜台报单固定为0x88888888；
                                                    ///< 2. 柜台清流启动后的历史报单固定为0xd8888888；
                                                    ///< 3. 柜台平仓报单固定为0xe8888888；
                                                    ///< 为保证报单性能，API不做本地报单编号重复的校验。
                                                    ///< 如果API发生了断线重连，在历史流水追平之后，请继续保持后续本地报单编号与
历史报单编号的唯一性。
                                                    ///< 如果不能保证本地报单编号的唯一性，请不要使用API的订单管理功能。
                                                    ///< API允许客户端使用同一用户名/口令多次登录，但客户端需要使用某种机制确保本
地报单编号不发生重复。
                                                    ///< 比如：（奇偶交替、分割号段）+单向递增。

    XTFDirection            direction;              ///< 买卖方向
    XTFOffsetFlag           offsetFlag;             ///< 开平仓标志
    XTFOrderType            orderType;              ///< 报单类型：限价(GFD)/市价/FAK/FOK
    double                  price;                  ///< 报单价格
    uint32_t                volume;                 ///< 报单数量
    uint32_t                minVolume;              ///< 最小成交数量。当报单类型为FAK时，
                                                    ///< 如果 minVolume > 1，那么API默认使用最小成交数量进行报单；
                                                    ///< 如果 minVolume ≤ 0，那么API默认使用任意成交数量进行报单；
    XTFChannelSelectionType channelSelectionType;   ///< 席位编号选择类型
    uint8_t                 channelID;              ///< 席位编号
    XTFOrderFlag            orderFlag;              ///< 报单标志（不使用，默认都是普通报单）
    XTFUserRef              userRef;                ///< 用户自定义数据，发送到柜台后，柜台不作处理，保留原值返回给用户

    XTF_CONST XTFInstrument *instrument;            ///< 报单合约对象
};
```

参数各字段详细说明，可以参考头文件注释，按需填写即可。这里，主要说明一下合约对象instrument。

合约对象是一个本地对象的指针，保存了报单需要的各类信息。建议在报单之前，调用API查询接口预先做查询，并保存起来，避免每次报单都需要做查询操作。

批量报单

API支持一次发送最大16个报单。

接口定义：

```
int insertOrders(const XTFInputOrder inputOrders[], size_t orderCount);
```

接口说明：向柜台发送批量报单，最大16个。

接口定义：

```
void onOrder(int errorCode, const XTFOrder *order);
```

接口说明：当报单成功或报单状态变化时，收到该接口的回调。

详细流程，参考报单回调流程一节。

撤单流程

API提供了多个撤单接口，可以按照报单对象XTFOrder直接撤单，也可以按照报单编号进行撤单。两种方式的撤单，都提供了批量撤单接口。

按报单对象撤单

接口定义：

```
int cancelOrder(const XTFOrder *order);
int cancelOrders(const XTFOrder *orders[], size_t orderCount);
```

接口说明：按照指定的报单对象进行撤单。

按照报单对象进行撤单，效率比按照编号撤单更好一些。避免了报单的本地查找开销。不过，需要用户自己保存需要撤单的对象指针。

按报单编号撤单

接口定义：

```
int cancelOrder(XTFOrderIDType orderIDType, long orderID);
int cancelOrders(XTFOrderIDType orderIDType, long orderIDs[], size_t orderCount);
int cancelOrder(XTFLocalOrderID localOrderID, const XTFInstrument *instrument);
```

接口说明：按照报单编号进行撤单。

接口提供了按多种报单编号类型进行撤单，报单编号类型包括：

1. 本地报单编号；
2. 柜台报单编号；
3. 交易所报单编号；

撤单回报

撤单成功或失败后，会调用撤单回报接口处理。

接口定义：

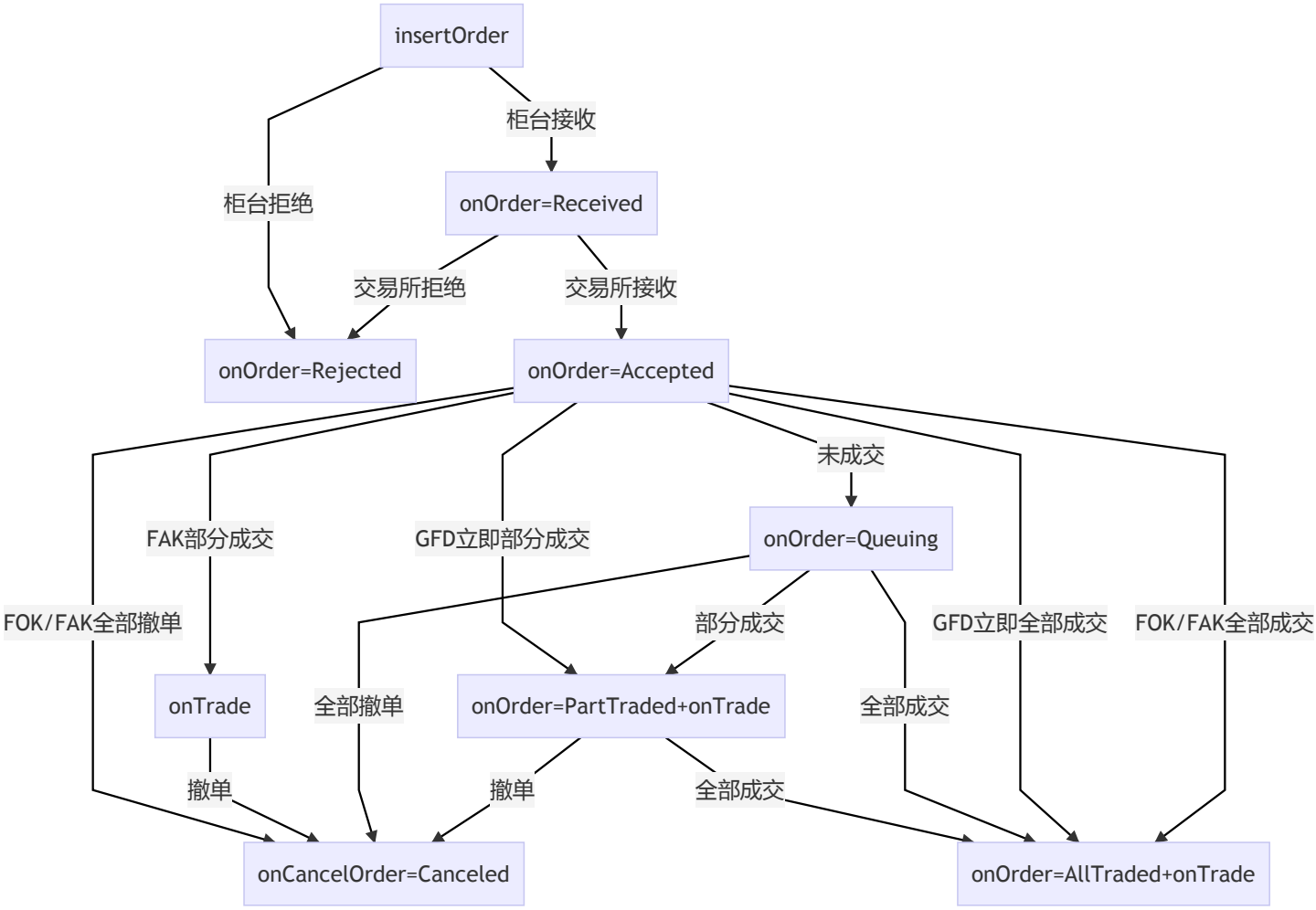
```
void onCancelOrder(int errorCode, const XTFOrder *cancelOrder);
```

接口说明：当撤单成功或失败时，收到该接口的回调。

详细流程，参考报单回调流程一节。

报单回调流程

接口回调顺序图如下：



报单回报状态的版本说明：

- `onOrder=Received` 在692版本及之前不支持 `Received` 状态。insertOrder发起报单请求后，如果柜台拒绝或交易所拒绝，会直接转为 `Rejected` 状态，交易所接收后，进入 `Accepted` 状态。

限价单类型的说明：

- GFD限价单：`OrderFlag == XTF_ODF_Normal && OrderType == XTF_ODT_Limit`
- FAK限价单：`OrderFlag == XTF_ODF_Normal && OrderType == XTF_ODT_FAK`
- FOK限价单：`OrderFlag == XTF_ODF_Normal && OrderType == XTF_ODT_FOK`

GFD限价单的报单回报流程（正常）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
	第3次回调	XTF_OA_Return	0	XTF_OS_Queuing	收到交易所的排队回报
全部成交					
	第4次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Queuing`状态；
- 所有事件通过`onOrder`回调接口通知用户；

GFD限价单的报单回报流程（排队后部分成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert0		XTF_OS_Accepted	收到插入报单的交易所应答
	第3次回调	XTF_OA_Return	0	XTF_OS_Queuing	收到交易所的排队回报
部分成交					
	第4次回调	XTF_OA_Return	0	XTF_OS_PartTraded	收到交易所的部成回报
	成交回报				
全部成交					
	第5次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 所有事件通过`onOrder`回调接口通知用户；
- 对于报单数量（volume字段）较大的场景，部分成交的回调可能有多次；

GFD限价单的报单回报流程（立即部分成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert0		XTF_OS_Accepted	收到插入报单的交易所应答
立即部分成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_PartTraded	收到交易所的部成回报
	成交回报				
全部成交					
	第4次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 如果GFD限价单报单后立即部分成交，则不会通知`XTF_OS_Queuing`状态：
- 所有事件通过`onOrder`回调接口通知用户；
- 对于报单数量（volume字段）较大的场景，部分成交的回调可能有多次；

GFD限价单的报单回报流程（柜台拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	柜台错误码	XTF_OS_Rejected	收到插入报单的柜台拒单

说明：

- 拒单事件通过`onOrder`回调接口通知用户；

GFD限价单的报单回报流程（交易所拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	交易所错误码	XTF_OS_Rejected	收到插入报单的交易所拒单

说明：

- 所有事件通过`onOrder`回调接口通知用户；

撤销GFD限价单的报单回报流程（正常）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
	第3次回调	XTF_OA_Return	0	XTF_OS_Queuing	收到交易所的排队回报
撤销报单					
	第4次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

撤销GFD限价单的报单回报流程（排队后部分成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
	第3次回调	XTF_OA_Return	0	XTF_OS_Queuing	收到交易所的排队回报
部分成交					
	第4次回调	XTF_OA_Return	0	XTF_OS_PartTraded	收到交易所的部成回报
	成交回报				
撤销报单					
	第5次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

撤销GFD限价单的报单回报流程（立即部分成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
部分成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_PartTraded	收到交易所的部成回报
	成交回报				
撤销报单					
	第4次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

撤销GFD限价单的报单回报流程（撤单失败）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
全部成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				
撤销报单					
	第4次回调	XTF_OA_Cancel	撤单失败错误码	—	收到交易所的撤单错误回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单失败事件通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

FAK限价单的报单回报流程（全部成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
全部成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 所有事件通过`onOrder`回调接口通知用户；

FAK限价单的报单回报流程（部分成交剩余撤单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
部分成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_PartTraded	收到交易所的部成回报
	成交回报				
自动撤单					
	第4次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报事件通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

FAK限价单的报单回报流程（全部撤单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
自动撤单					
	第3次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报事件通过`onCancelOrder`回调接口通知用户；

- 其他事件通过`onOrder`回调接口通知用户；

FAK限价单的报单回报流程（柜台拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	柜台错误码	XTF_OS_Rejected	收到插入报单的柜台拒单

说明：

- 拒单事件通过`onOrder`回调接口通知用户；

FAK限价单的报单回报流程（交易所拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	交易所错误码	XTF_OS_Rejected	收到插入报单的交易所拒单

说明：

- 所有事件通过`onOrder`回调接口通知用户；

FOK限价单的报单回报流程（全部成交）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
全部成交					
	第3次回调	XTF_OA_Return	0	XTF_OS_AllTraded	收到交易所的全成回报
	成交回报				

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 所有事件通过`onOrder`回调接口通知用户；

FOK限价单的报单回报流程（全部撤单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	0	XTF_OS_Accepted	收到插入报单的交易所应答
自动撤单					
	第3次回调	XTF_OA_Return	0	XTF_OS_Canceled	收到交易所的撤单回报

说明：

- 如果应答和回报发生乱序，则不会通知`XTF_OS_Accepted`状态；
- 撤单回报通过`onCancelOrder`回调接口通知用户；
- 其他事件通过`onOrder`回调接口通知用户；

FOK限价单的报单回报流程（柜台拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	柜台错误码	XTF_OS_Rejected	收到插入报单的柜台拒单

说明：

- 所有事件通过`onOrder`回调接口通知用户；

FOK限价单的报单回报流程（交易所拒单）

		actionCode	errorCode	orderStatus	
插入报单					
	第1次回调	XTF_OA_Insert	0	XTF_OS_Received	收到插入报单的柜台应答
	第2次回调	XTF_OA_Insert	交易所错误码	XTF_OS_Rejected	收到插入报单的交易所拒单

说明：

- 所有事件通过`onOrder`回调接口通知用户；

行权和对冲

行权报单和对冲报单都使用`XTFExecOrder`，数据结构定义如下：

```
class XTFExecOrder {
public:
    XTFLocalOrderID         localOrderID;           ///< 本地报单编号，需要由用户保证唯一性，用于本地存储索引。
                                                    ///< 不能和XTFInputOrder本地报单编号发送冲突，应与XTFInputOrder的本地报单
编号统一处理。
    XTFOrderFlag            orderFlag;              ///< 报单标志，用于区分是行权还是自对冲
    XTFOrderType            orderType;              ///< 报单类型：
                                                    ///< - 行权：XTF_ODT_SelfClose | XTF_ODT_NotSelfClose
                                                    ///< - 对冲：XTF_ODT_SelfCloseOptions | XTF_ODT_SelfCloseFutures
    XTFOffsetFlag           offsetFlag;             ///< 开平仓标志：XTF_OF_Close | XTF_OF_CloseToday |
XTF_OF_CloseYesterday
    XTFDirection            direction;              ///< 行权和对冲方向：
                                                    ///< - XTF_D_Buy：请求行权、请求对冲；
```

```
                                                ///< - XTF_D_Sell：放弃行权、请求不对冲；
    XTFHedgeFlag            hedgeFlag;          ///< 投机套保标志
    double                  minProfit;          ///< 行权最小利润
    uint16_t                volume;             ///< 行权数量
    XTFChannelSelectionType channelSelectionType;   ///< 席位编号选择类型
    uint8_t                 channelID;          ///< 席位编号
    XTFUserRef              userRef;            ///< 用户自定义数据，发送到柜台后，柜台不作处理，保留原值返回给用户
    XTF_CONST XTFInstrument *instrument;        ///< 报单合约对象
};
```

通过XTFOrderFlag、XTFOrderType、XTFDirection三个字段区分是请求行权、放弃行权、请求对冲、请求不对冲。下面的表格给出了不同报单对应的字段值：

	XTFOrderFlag	XTFDirection	XTFOrderType
请求行权（且行权后自对冲期权）	XTF_ODF_OptionsExecute	XTF_D_Buy	XTF_ODT_SelfClose
请求行权（且行权后不对冲期权）	XTF_ODF_OptionsExecute	XTF_D_Buy	XTF_ODT_NotSelfClose
放弃行权	XTF_ODF_OptionsExecute	XTF_D_Sell	—
请求期权对冲	XTF_ODF_OptionsSelfClose	XTF_D_Buy	XTF_ODT_SelfCloseOptions
请求履约对冲	XTF_ODF_OptionsSelfClose	XTF_D_Buy	XTF_ODT_SelfCloseFutures
请求期权不对冲	XTF_ODF_OptionsSelfClose	XTF_D_Sell	XTF_ODT_SelfCloseOptions
请求履约不对冲	XTF_ODF_OptionsSelfClose	XTF_D_Sell	XTF_ODT_SelfCloseFutures

按照上述表格构造行权（对冲）报单后，通过以下接口即可实现报撤单请求：

1. 请求报单：insertExecOrder(const XTFExecOrder &execOrder);
2. 撤销请求：cancelExecOrder(const XTFOrder *order);

柜台重启

在API使用过程中，如果盘中柜台发生重启，那么需要API调用者实现`onServerReboot()`接口，完成失效数据的清理。

清理的数据类型包括所有的XTF开头的数据结构。

查询接口

查询账户

接口定义：

```
const XTFAccount* getAccount();
```

接口说明：查询当前账户信息。

查询合约

接口定义：

```
int getInstrumentCount();
const XTFInstrument* getInstrument(int pos);
const XTFInstrument* getInstrumentByID(const char *instrumentID);
```

接口说明：查询合约数量和指定合约。

建议：报单前可以先把目标合约对象查找并存储起来，以加快报单的速度。

查询仓位

接口定义：

```
int XTFAccount::getPositionCount() const;
const XTFPosition* XTFAccount::getPosition(int pos) const;
```

接口说明：查询当前用户下的所有仓位信息。

说明：如果仓位已平，通过该接口是无法查询的。

查询报单

接口定义：

```
int XTFAccount::getOrderCount() const;
const XTFOrder* XTFAccount::getOrder(int pos) const;
```

接口说明：查询当前用户下的所有报单信息，包括所有状态的报单。