华锐ATPTradeAPI
开发指南

版本 V3.2.3.3
二o二二年六月十三日

修订记录

日期	版本	修订者	修订说明
2020-11-12	v3.1.2.2	杜靖	初稿
2020-11-24	v3.1.3	何云飞	修正环境变量设置值的拼写no_low_lantecy 为no_low_latency
2020-11-30	v3.1.4	何云飞	修复文档内链接不能正常跳转的缺陷
2020-12-24	v3.1.5	何云飞	新增订单部撤流程（撤单响应在成交回报前），优化订单部撤流程图（撤单申报在多笔成交响应中），新增常见问题解析-如何区分下单委托与撤单委托
2021-03-03	v3.1.7	何云飞	新增算法交易功能介绍
2021-05-18	v3.1.9	何云飞	新增批量订单处理流程介绍
2021-07-15	V3.1.11	杜靖	修改订单部撤流程（撤单响应在成交回报前），新增订单部撤流程（撤单响应在多笔成交回报中）

接口约定

1. 异步调用

- ATP所有会话接口、业务接口都为异步接口
- 调用异步请求接口返回kSuccess，仅代表请求发送成功
- 继承ATPTradeHandler，并实现回调函数，接受异步响应

2. 用整型来表示浮点数类型

- API字段类型，Nx(y)表示通过整型来表示浮点数类型，其中x表示整数与消息总计位数，不包括小数点，y表示小数位数；
- 请使用者注意字段类型，如*现货集中竞价交易委托函数*的申报数量字段类型为N15(2)，表示申报数量为10股时，则调用接口需传入1000；委托价格字段类型为N13(4)，表示申报价格为5.5元，则调用接口需传入55000；
- ATPTradeAPI提供**DoubleExpandToInt**工具函数，可帮投资者完成浮点型到整型的转换。

目录

1. 快速上手

本文将以C++版本的tradeAPI为例编写一个现货竞价发单程序，旨在从零开始介绍tradeAPI使用方法

1.1. 取开发包

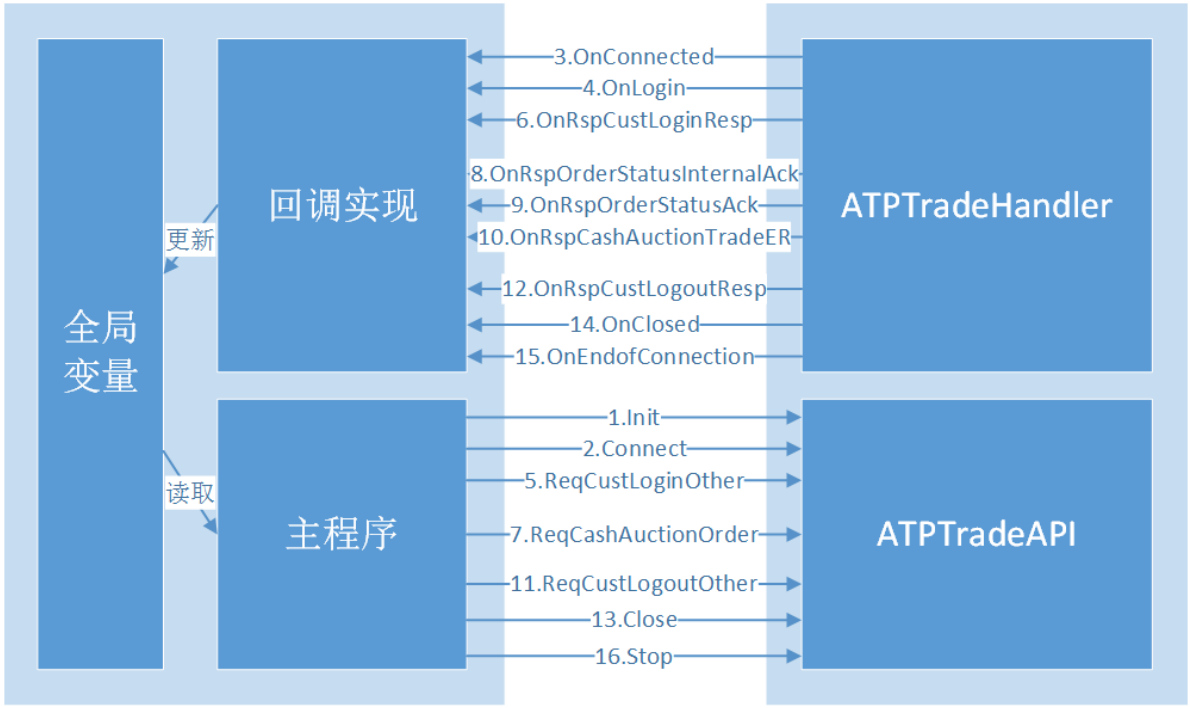
- 从官网上获取最新的tradeAPI开发包。tradeAPI开发包有windows、linux两个版本，本文将以linux版本作为示例
- 解压压缩包
- 解压后可看到atp_trade_api有include、lib、demo三个目录

目录	文件	说明
include	atp_trade_api.h	主头文件，开发者直接包含此头文件即可
include	err_code.xml	错误信息配置文件
include/trade/	atp_trade_client.h	异步调用接口
include/trade/	atp_trade_handler.h	回调处理句柄
include/trade/	atp_trade_constants.h	业务消息枚举类型定义
include/trade/	atp_trade_msg.h	业务消息的定义
include/trade/	atp_trade_types.h	业务消息的字段类型定义
include/trade/	atp_trade_export.h	符号导入宏定义
lib	libatptradeapi.so	ATPTradeAPI库文件
lib	libagwapi.so	ATPTradeAPI通信层库文件
lib	libagw_external_msg.so	协议编解码库文件
lib	libcommon_api.so	ATPTradeAPI公共库文件
lib	libagw_des.so	ATPTradeAPI公共库文件
lib	libmsg_proto.so	协议编解码基础库文件
demo	all_message_demo	全业务示例
demo	simple_demo	简单示例

1.2. 简单介绍

- ATPTradeAPI采用异步调用方式，异步调用函数定义在ATPTradeAPI类，回调函数定义在ATPTradeHandler类中
- 异步调用线程和回调线程不能在同一线程，建议定义原子变量传递数据和状态
- 正常调用流程

- 1 初始化ATPTradeAPI
- 2-6 建立连接并登录
- 7-10 发送委托接收回报
- 11-15 退出登录并关闭连接
- 16 清理ATPTradeAPI



1.3. 编辑代码

引用头文件

```
#if defined _WIN32
#include <windows.h>
#define sleep Sleep
#else
#include <unistd.h>
#endif
```

定义全局变量

```
std::atomic_bool g_connected_flag(false);           // 已连接标识
std::atomic_bool g_cust_logined_flag(false);        // 已登录标识

std::atomic_bool g_waiting_flag(false);             // 等待响应标识

ATPClientSeqIDType g_client_seq_id = 1;             //  客户系统消息号
ATPClientFeatureCodeType g_client_feature_code = "ip:127.0.0.1;mac:2222";  // 终端识别码，由券商指定规则

// 定义回报数据分区+序号的全局变量，在收到回报时更新此变量，在断线重连时传入该变量，指示客户端所收到的最大回报序号
std::map<int32_t, int32_t> report_sync;
```

定义回调函数

```cpp
// 回调类，继承自ATPTradeHandler，可以只实现部分回调函数
class CustHandler : public ATPTradeHandler
{
    // 登入回调
    virtual void OnLogin(const std::string& reason)
    {
        std::cout << "OnLogin Recv:" << reason << std::endl;
        g_connected_flag.store(true);
    }

    // 登出回调
    virtual void OnLogout(const std::string& reason)
    {
        std::cout << "OnLogout Recv:" << reason << std::endl;
        g_connected_flag.store(false);
        g_cust_logined_flag.store(false);
    }

    // 连接失败
    virtual void OnConnectFailure(const std::string &reason)
    {
        std::cout << "OnConnectFailure Recv:" << reason << std::endl;
        g_connected_flag.store(false);
        g_cust_logined_flag.store(false);
    }

    // 连接超时
    virtual void OnConnectTimeOut(const std::string &reason)
    {
        std::cout << "OnConnectTimeOut Recv:" << reason << std::endl;
        g_connected_flag.store(false);
        g_cust_logined_flag.store(false);
    }

    // 心跳超时
    virtual void OnHeartbeatTimeout(const std::string &reason)
    {
        std::cout << "OnHeartbeatTimeout Recv:" << reason << std::endl;
        g_connected_flag.store(false);
        g_cust_logined_flag.store(false);
    }

    // 连接关闭
    virtual void OnClosed(const std::string &reason)
    {
        std::cout << "OnClosed Recv:" << reason << std::endl;
        g_connected_flag.store(false);
        g_cust_logined_flag.store(false);
    }

    // 连接结束回调
    virtual void OnEndOfConnection(const std::string& reason)
    {
        std::cout << "OnEndOfConnection Recv:" << reason << std::endl;
        g_waiting_flag.store(false);
    }

    // 客户号登入回调
```

```
    virtual void OnRspCustLoginResp(const ATPRspCustLoginRespOtherMsg
&cust_login_resp)
    {
        std::cout << "OnRspCustLoginResp Recv:" << static_cast<uint32_t>
(cust_login_resp.permisson_error_code) << std::endl;
        if (cust_login_resp.permisson_error_code == 0)
        {
            g_cust_logined_flag.store(true);
        }
        else
        {
            std::cout << "Login rsp no:" << static_cast<uint32_t>
(cust_login_resp.permisson_error_code) << std::endl;
        }


        g_waiting_flag.store(false);
    }

    // 客户号登出回调
    virtual void OnRspCustLogoutResp(const ATPRspCustLogoutRespOtherMsg
&cust_logout_resp)
    {
        std::cout << "OnRspCustLogoutResp Recv:" << static_cast<uint32_t>
(cust_logout_resp.permisson_error_code) << std::endl;
        if (cust_logout_resp.permisson_error_code == 0)
        {
            g_cust_logined_flag.store(false);
        }
        else
        {
            std::cout << "Login rsp no:" << static_cast<uint32_t>
(cust_logout_resp.permisson_error_code) << std::endl;
        }
        g_waiting_flag.store(false);
    }

    // 订单下达内部响应
    virtual void OnRspOrderStatusInternalAck(const ATPRspOrderStatusAckMsg&
order_status_ack)
    {
        std::cout << "order_status_ack : " << std::endl;
        std::cout << "partition : " << (int32_t)order_status_ack.partition <<
            " index : " << order_status_ack.index <<
            " business_type : " << (int32_t)order_status_ack.business_type <<
            " cl_ord_no : " << order_status_ack.cl_ord_no <<
            " security_id : " << order_status_ack.security_id <<
            " market_id : " << order_status_ack.market_id <<
            " exec_type : " << order_status_ack.exec_type <<
            " ord_status : " << (int32_t)order_status_ack.ord_status <<
            " cust_id : " << order_status_ack.cust_id <<
            " fund_account_id : " << order_status_ack.fund_account_id <<
            " account_id : " << order_status_ack.account_id <<
            " price : " << order_status_ack.price <<
            " order_qty : " << order_status_ack.order_qty <<
            " leaves_qty : " << order_status_ack.leaves_qty <<
            " cum_qty : " << order_status_ack.cum_qty <<
            " side : " << order_status_ack.side <<
```

```
            " transact_time : " << order_status_ack.transact_time <<
            " user_info : " << order_status_ack.user_info <<
            " order_id : " << order_status_ack.order_id <<
            " cl_ord_id : " << order_status_ack.cl_ord_id <<
            " client_seq_id : " << order_status_ack.client_seq_id <<
            " orig_cl_ord_no : " << order_status_ack.orig_cl_ord_no <<
            " frozen_trade_value : " << order_status_ack.frozen_trade_value <<
            " frozen_fee : " << order_status_ack.frozen_fee <<
            " reject_reason_code : " << order_status_ack.reject_reason_code <<
            " ord_rej_reason : " << order_status_ack.ord_rej_reason <<
            " order_type : " << order_status_ack.order_type <<
            " time_in_force : " << order_status_ack.time_in_force <<
            " position_effect : " << order_status_ack.position_effect <<
            " covered_or_uncovered : " <<
(int32_t)order_status_ack.covered_or_uncovered <<
            " account_sub_code : " << order_status_ack.account_sub_code <<
std::endl;

        // 保存回报分区号、序号，用于断线重连时指定已收到最新回报序号
        report_sync[order_status_ack.partition] = order_status_ack.index;
    }

    // 订单下达交易所确认
    virtual void OnRspOrderStatusAck(const ATPRspOrderStatusAckMsg&
order_status_ack)
    {
        std::cout << "order_status_ack : " << std::endl;
        std::cout << "partition : " << (int32_t)order_status_ack.partition <<
            " index : " << order_status_ack.index <<
            " business_type : " << (int32_t)order_status_ack.business_type <<
            " cl_ord_no : " << order_status_ack.cl_ord_no <<
            " security_id : " << order_status_ack.security_id <<
            " market_id : " << order_status_ack.market_id <<
            " exec_type : " << order_status_ack.exec_type <<
            " ord_status : " << (int32_t)order_status_ack.ord_status <<
            " cust_id : " << order_status_ack.cust_id <<
            " fund_account_id : " << order_status_ack.fund_account_id <<
            " account_id : " << order_status_ack.account_id <<
            " price : " << order_status_ack.price <<
            " order_qty : " << order_status_ack.order_qty <<
            " leaves_qty : " << order_status_ack.leaves_qty <<
            " cum_qty : " << order_status_ack.cum_qty <<
            " side : " << order_status_ack.side <<
            " transact_time : " << order_status_ack.transact_time <<
            " user_info : " << order_status_ack.user_info <<
            " order_id : " << order_status_ack.order_id <<
            " cl_ord_id : " << order_status_ack.cl_ord_id <<
            " client_seq_id : " << order_status_ack.client_seq_id <<
            " orig_cl_ord_no : " << order_status_ack.orig_cl_ord_no <<
            " frozen_trade_value : " << order_status_ack.frozen_trade_value <<
            " frozen_fee : " << order_status_ack.frozen_fee <<
            " reject_reason_code : " << order_status_ack.reject_reason_code <<
            " ord_rej_reason : " << order_status_ack.ord_rej_reason <<
            " order_type : " << order_status_ack.order_type <<
            " time_in_force : " << order_status_ack.time_in_force <<
            " position_effect : " << order_status_ack.position_effect <<
            " covered_or_uncovered : " <<
(int32_t)order_status_ack.covered_or_uncovered <<
```

```
            " account_sub_code : " << order_status_ack.account_sub_code <<
            " quote_flag:"<<(int32_t)order_status_ack.quote_flag<< std::endl;

        // 保存回报分区号、序号，用于断线重连时指定已收到最新回报序号
        report_sync[order_status_ack.partition] = order_status_ack.index;
    }

    // 成交回报
    virtual void OnRspCashAuctionTradeER(const ATPRspCashAuctionTradeERMsg&
cash_auction_trade_er)
    {
        std::cout << "cash_auction_trade_er : " << std::endl;
        std::cout << "partition : " << (int32_t)cash_auction_trade_er.partition
<<
            " index : " << cash_auction_trade_er.index <<
            " business_type : " <<(int32_t)cash_auction_trade_er.business_type
<<
            " cl_ord_no : " << cash_auction_trade_er.cl_ord_no <<
            " security_id : " << cash_auction_trade_er.security_id <<
            " market_id : " << cash_auction_trade_er.market_id <<
            " exec_type : " << cash_auction_trade_er.exec_type <<
            " ord_status : " << (int32_t)cash_auction_trade_er.ord_status <<
            " cust_id : " << cash_auction_trade_er.cust_id <<
            " fund_account_id : " << cash_auction_trade_er.fund_account_id <<
            " account_id : " << cash_auction_trade_er.account_id <<
            " price : " << cash_auction_trade_er.price <<
            " order_qty : " << cash_auction_trade_er.order_qty <<
            " leaves_qty : " << cash_auction_trade_er.leaves_qty <<
            " cum_qty : " << cash_auction_trade_er.cum_qty <<
            " side : " << cash_auction_trade_er.side <<
            " transact_time : " << cash_auction_trade_er.transact_time <<
            " user_info : " << cash_auction_trade_er.user_info <<
            " order_id : " << cash_auction_trade_er.order_id <<
            " cl_ord_id : " << cash_auction_trade_er.cl_ord_id <<
            " exec_id : " << cash_auction_trade_er.exec_id <<
            " last_px : " << cash_auction_trade_er.last_px <<
            " last_qty : " << cash_auction_trade_er.last_qty <<
            " total_value_traded : " << cash_auction_trade_er.total_value_traded
<<
            " fee : " << cash_auction_trade_er.fee <<
            " cash_margin : " << cash_auction_trade_er.cash_margin << std::endl;

        // 保存回报分区号、序号，用于断线重连时指定已收到最新回报序号
        report_sync[cash_auction_trade_er.partition] =
cash_auction_trade_er.index;
    }

    // 订单下达内部拒绝
    virtual void OnRspBizRejection(const ATPRspBizRejectionOtherMsg&
biz_rejection)
    {
        std::cout << "biz_rejection : " << std::endl;
        std::cout << "transact_time : " << biz_rejection.transact_time <<
            " client_seq_id : " << biz_rejection.client_seq_id <<
            " msg_type : " << biz_rejection.api_msg_type <<
            " reject_reason_code : " << biz_rejection.reject_reason_code <<
            " business_reject_text : " << biz_rejection.business_reject_text <<
            " user_info : " << biz_rejection.user_info << std::endl;
```

```
    }
};
```

封装异步调用函数

```
// 建立连接
ATPRetCodeType connect(ATPTradeAPI* client, ATPTradeHandler* handler)
{
    // 设置连接信息
    ATPConnectProperty prop;
    prop.user = "user1";                                          // 网关用户名
    prop.password = "password1";                                  // 网关用户密
码
    prop.locations = { "127.0.0.1:32495", "127.0.0.1:32495" };    // 网关主备节
点的地址+端口
    prop.heartbeat_interval_milli = 5000;                         // 发送心跳的
时间间隔，单位：毫秒
    prop.connect_timeout_milli = 5000;                            // 连接超时时
间，单位：毫秒
    prop.reconnect_time = 10;                                     // 重试连接次
数
    prop.client_name = "api_demo";                                // 客户端程序
名字
    prop.client_version = "V1.0.0";                                  // 客户端程
序版本
    prop.report_sync = report_sync;                               // 回报同步数
据分区号+序号，首次是空，断线重连时填入的是接受到的最新分区号+序号
    prop.mode = 0;                                                // 模式0-同步
回报模式，模式1-快速登录模式，不同步回报

    // 建立连接
    while (!g_connected_flag.load())
    {
        // 在连接中
        if (g_waiting_flag.load())
        {
            sleep(0);
        }
        else
        {
            g_waiting_flag.store(true);
            // 建立连接
            ATPRetCodeType ec = client->Connect(prop, handler);
            if (ec != ErrorCode::kSuccess)
            {
                std::cout << "Invoke Connect error:" << ec << std::endl;
                return ec;
            }
            sleep(0);
        }
    }

    return kSuccess;
}

// 关闭连接
ATPRetCodeType close(ATPTradeAPI* client)
```

```
{
    g_waiting_flag.store(true);
    ATPRetCodeType ec = client->Close();
    if (ec != ErrorCode::kSuccess)
    {
        std::cout << "Invoke Close error:" << ec << std::endl;
        return ec;
    }

    while(g_waiting_flag.load())
    {
        sleep(0);
    }
    return ErrorCode::kSuccess;
}

// 登录
ATPRetCodeType login(ATPTradeAPI* client)
{
    // 设置登入消息
    ATPReqCustLoginOtherMsg login_msg;
    strncpy(login_msg.cust_id, "00000001", 17);                  // 客户号ID
    strncpy(login_msg.password, "123456", 129);                 // 客户号密码
    login_msg.login_mode = ATPCustLoginModeType::kCustIDMode;   // 登录模式，客户号
登录
    login_msg.client_seq_id = g_client_seq_id++;                // 客户系统消息号
    login_msg.order_way = '0';                                  // 委托方式，自助委
托
    login_msg.client_feature_code = g_client_feature_code;      // 终端识别码

    g_waiting_flag.store(true);
    ATPRetCodeType ec = client->ReqCustLoginOther(&login_msg);
    if (ec != ErrorCode::kSuccess)
    {
        std::cout << "Invoke CustLogin error:" << ec << std::endl;
        return ec;
    }

    while(g_waiting_flag.load())
    {
        sleep(0);
    }

    return ErrorCode::kSuccess;
}

// 登出
ATPRetCodeType logout(ATPTradeAPI* client)
{
    // 设置登出消息
    ATPReqCustLogoutOtherMsg logout_msg;
    strncpy(logout_msg.cust_id, "00000001", 17);                 // 客户号ID
    logout_msg.client_seq_id = g_client_seq_id++;               // 客户系统消息号
    logout_msg.client_feature_code = g_client_feature_code;      // 终端识别码

    g_waiting_flag.store(true);
    ATPRetCodeType ec = client->ReqCustLogoutOther(&logout_msg);
    if (ec != ErrorCode::kSuccess)
```

```
    {
        std::cout << "Invoke CustLogout error:" << ec << std::endl;
        return ec;
    }

    while(g_waiting_flag.load())
    {
        sleep(0);
    }

    return ErrorCode::kSuccess;
}

// 发送订单
ATPRetCodeType send(ATPTradeAPI* client)
{
    // 发送委托
    ATPReqCashAuctionOrderMsg* p = new ATPReqCashAuctionOrderMsg;

    strncpy(p->security_id, "000001", 9);                    // 证券代码
    p->market_id = ATPMarketIDConst::kShangHai;             // 市场ID，上海
    strncpy(p->cust_id, "00000001", 17);                    // 客户号ID
    strncpy(p->fund_account_id, "0000001", 17);             // 资金账户ID
    strncpy(p->account_id, "000000", 13);                   // 账户ID
    p->side = ATPSideConst::kBuy;                           // 买卖方向，买
    p->order_type = ATPOrdTypeConst::kFixedNew;             // 订单类型，限价
    p->price = 100000;                                      // 委托价格 N13(4)，
10.0000元
    p->order_qty = 10000;                                   // 申报数量（股票为股、
基金为份、上海债券为手，其他为张）N15(2)，100.00股
    p->client_seq_id = g_client_seq_id++;                   // 用户系统消息序号
    p->order_way = 'O';                                     // 委托方式，自助委托
    strncpy(p->password, "password1",129);                  // 客户密码
    p->client_feature_code = g_client_feature_code;         // 终端识别码

    ATPRetCodeType ec = client->ReqCashAuctionOrder(p);
    if (ec != ErrorCode::kSuccess)
    {
        std::cout << "Invoke Send error:" << ec << std::endl;
    }

    return ec;
}
```

定义建立连接并登入函数

```
// 初始化连接并完成登录
bool init(ATPTradeAPI* client, ATPTradeHandler* handler)
{
    // 初始化API
    const std::string station_name = ""; // 站点信息，该字段已经不使用
    const std::string cfg_path=".";      // 配置文件路径
    const std::string log_dir_path = ""; // 日志路径
    bool record_all_flag = true;         // 是否记录所有委托信息
    std::unordered_map<std::string,std::string> encrypt_cfg; // 加密库配置（具体使
用方法请参考引言中关于加密库的描述）
    bool connection_retention_flag=false;   // 是否启用会话保持
```

```
    // encrypt_cfg参数填写：
    encrypt_cfg["ENCRYPT_SCHEMA"]="0";                // 字符 0 表示 不对消息中的所有password 加密
    encrypt_cfg["ATP_ENCRYPT_PASSWORD"]="";           // 除登入及密码修改外其他消息的密码字段加密算法
    encrypt_cfg["ATP_LOGIN_ENCRYPT_PASSWORD"]="";     // 登入及密码修改消息中密码字段的加密算法so路径
    encrypt_cfg["GM_SM2_PUBLIC_KEY_PATH"]="";         // 采用国密算法时，通过该key配置GM算法配置加密使用的公钥路径
    encrypt_cfg["RSA_PUBLIC_KEY_PATH"]="";            // 如果使用rsa算法加密，通过该key配置 rsa算法配置加密使用的公钥路径

    ATPRetCodeType ec = ATPTradeAPI::Init(station_name,cfg_path,log_dir_path,record_all_flag,encrypt_cfg,connection_retention_flag);
    if (ec != ErrorCode::kSuccess)
    {
        std::cout << "Init failed: " << ec << std::endl;
        return false;
    }

    // 建立连接
    if (connect(client, handler) != ErrorCode::kSuccess)
    {
        return false;
    }

    // 登录
    if (login(client) != ErrorCode::kSuccess)
    {
        return false;
    }

    // 检查是否登录成功
    if (!g_cust_logined_flag.load())
    {
        return false;
    }

    return true;
}
```

定义关闭连接并退出

```
// 关闭连接并退出
void exit(ATPTradeAPI* client, ATPTradeHandler* handler)
{
    if (g_cust_logined_flag.load())
    {
        logout(client);
    }

    if (g_connected_flag.load())
    {
        close(client);
    }
```

```
    ATPTradeAPI::Stop();
}
```

定义主函数

```
int main(int argc, char* argv[])
{
    // 获取tradeAPI客户端和回调句柄
    ATPTradeAPI* client = new ATPTradeAPI();
    CustHandler* handler = new CustHandler();

    if (init(client, handler))
    {
        if (send(client) == ErrorCode::kSuccess)
        {
            std::cout << "Wait for ack. press enter for exit." << std::endl;
            getchar();

        }
    }

    exit(client, handler);
    delete client;
    delete handler;

    return 0;
}
```

1.4. 编译运行

编写CMakeLists.txt文件

```
cmake_minimum_required(VERSION 3.7.1)

project(simple_trade_api_demo)

message(STATUS "This is BINARY dir " ${PROJECT_BINARY_DIR})
message(STATUS "This is SOURCE dir " ${PROJECT_SOURCE_DIR})

set(CMAKE_CXX_STANDARD 11)

if (MSVC)
  add_definitions(/MP)
  set(CMAKE_COMPILER_CXX_FLAG "${CMAKE_COMPILER_CXX_FLAG}   -std=c++11")
  add_compile_options("$<$<CXX_COMPILER_ID:MSVC>:/utf-8>")
endif (MSVC)


file(GLOB_RECURSE DEMO_INCLUDE "${PROJECT_SOURCE_DIR}/../../include/*.h")
source_group("simple_trade_api_demo" FILES  ${DEMO_INCLUDE})
AUX_SOURCE_DIRECTORY(. simple_trade_api_demo_SRC_LIST)
add_executable(simple_trade_api_demo  ${DEMO_INCLUDE}
 ${simple_trade_api_demo_SRC_LIST})
```

```
if(CMAKE_SYSTEM_NAME STREQUAL "Windows")
     file(GLOB_RECURSE LINK_SO "${PROJECT_SOURCE_DIR}/../../lib/*.lib")
elseif(CMAKE_SYSTEM_NAME MATCHES "Linux")
     file(GLOB_RECURSE LINK_SO "${PROJECT_SOURCE_DIR}/../../lib/*.so")
endif()

target_link_libraries(simple_trade_api_demo  ${LINK_SO})
target_include_directories(simple_trade_api_demo PRIVATE
${PROJECT_SOURCE_DIR}/../../include/  ${PROJECT_SOURCE_DIR}/../../include/trade
)
target_compile_options(simple_trade_api_demo PRIVATE $<$<CXX_COMPILER_ID:MSVC>:
/Zi /Od>)
target_link_options(simple_trade_api_demo PRIVATE $<$<CXX_COMPILER_ID:MSVC>:
/DEBUG >)
```

执行编译

```
> cmake .
> make
```

2. 参数说明

2.1. 连接参数

ATPTradeAPI提供**ATPConnectProperty**结构体，可对ATPTradeAPI功能做相关设置。

ATPConnectProperty的结构

形参	类型	默认值	描述
user	std::string		登录AGW的账户
password	std::string		登录AGW的密码
locations	std::vector<std::string>		AGW主备 IP地址和端口 格式为："ip:port"
heartbeat_interval_milli	int32_t	5000	发送心跳的时间间隔，单位：毫秒
connect_timeout_milli	int32_t	5000	连接超时时间，单位：毫秒
reconnect_time	int32_t	10	重试连接次数
report_sync	std::map<int32_t, int32_t>	std::map<int32_t, int32_t>()	回报同步数据 分区号和序号
client_name	std::string		客户端程序名称
client_version	std::string		客户端程序版本号
mode	int32_t	0	0-正常登录 1-快速登陆 快速登录模式不回同步回报请求

2.2. 环境变量参数

ATPTradeAPI提供了少量环境变量参数

3. 功能介绍

3.1. 初始化与退出

ATPTradeAPI提供了初始化**Init**和退出函数**Stop**，分别需要在**客户程序初始化时**，和**客户程序正常退出时**调用。

3.2. 会话管理

ATPTradeAPI提供了完整的且易于客户端开发的会话管理功能。

基本流程

- 调用**Connect**实现**异步建立连接**；
- 当收到**OnLogin**回调时，代表**连接建立成功**
- 调用**ReqCustLoginOther** 实现**异步登录**，并在收到**OnRspCustLoginResp**回调后，代表**登录成功**，可以进行报单；

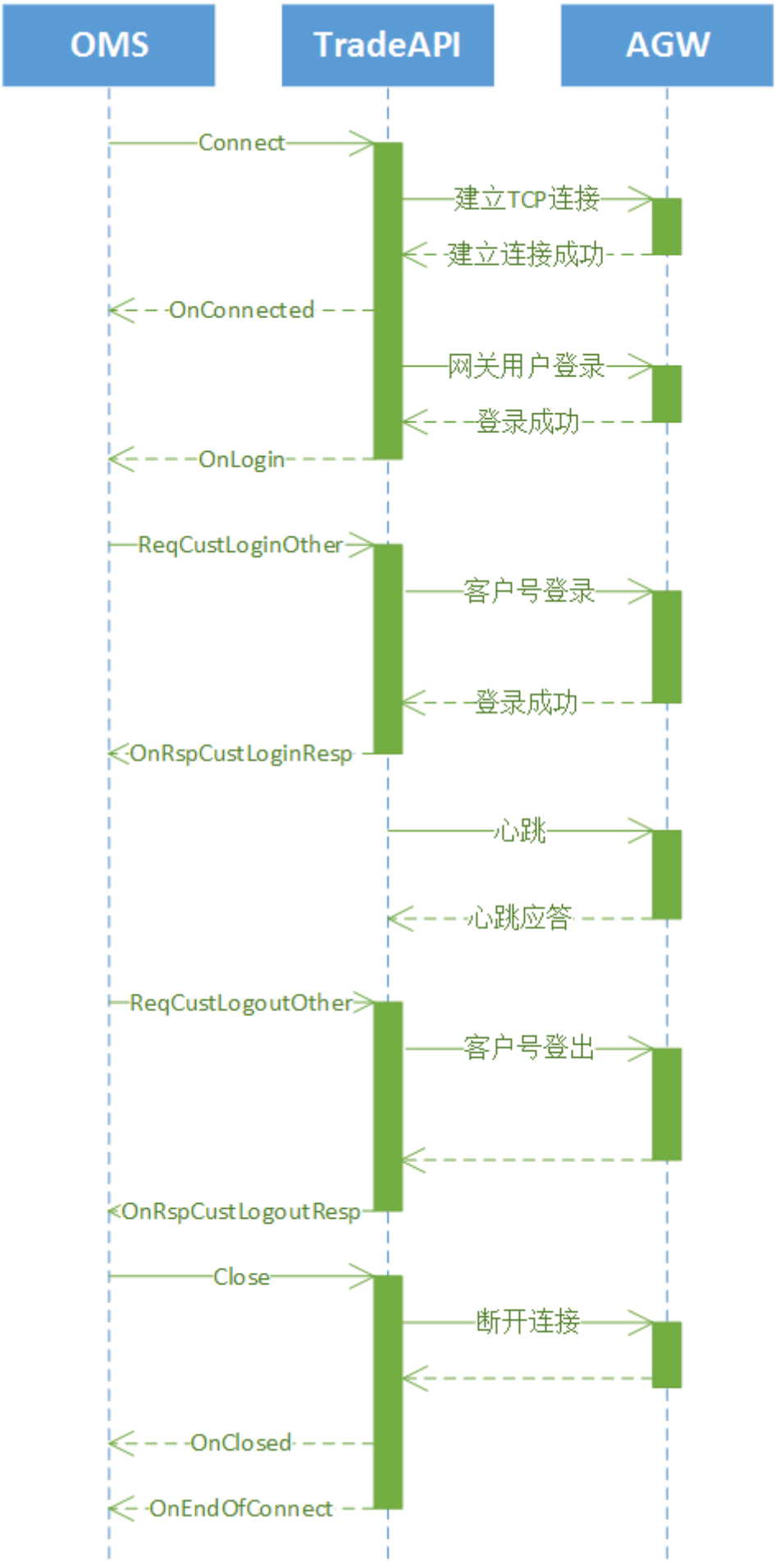
- 当需要退出时，可调用**ReqCustLogoutOther**实现**异步登出**，并在收到**OnRspCustLogoutResp**回调代表登出成功；
- 最终调用**Close**实现**异步关闭连接**；
- 收到**OnEndOfConnection**回调，代表连接断开；
- 在调用**Connect**实现**异步建立连接**时，收到**OnConnectFailure**、**OnClosed**回调，代表连接失败，此时会自动重连，该过程无需干预，直到重试次数用完，收到**OnEndOfConnection**回调，此后可根据发生的异常情况，处理后再次调用**Connect**发起连接；
- 在连接建立成功，并未主动调用Close断开连接的情况下，ATPTradeAPI会定时向AGW发送心跳包，若在**三倍心跳时长**内未收到AGW的任何响应，则视为**心跳超时**，会发起自动重连。

相关参数

- **user/password** 需要连接的网关用户名密码
- **locations** 主备网关的IP和端口 格式为："ip:port"
- **connect_timeout_milli** 超时连接时间
- **reconnect_time** 自动重试次数

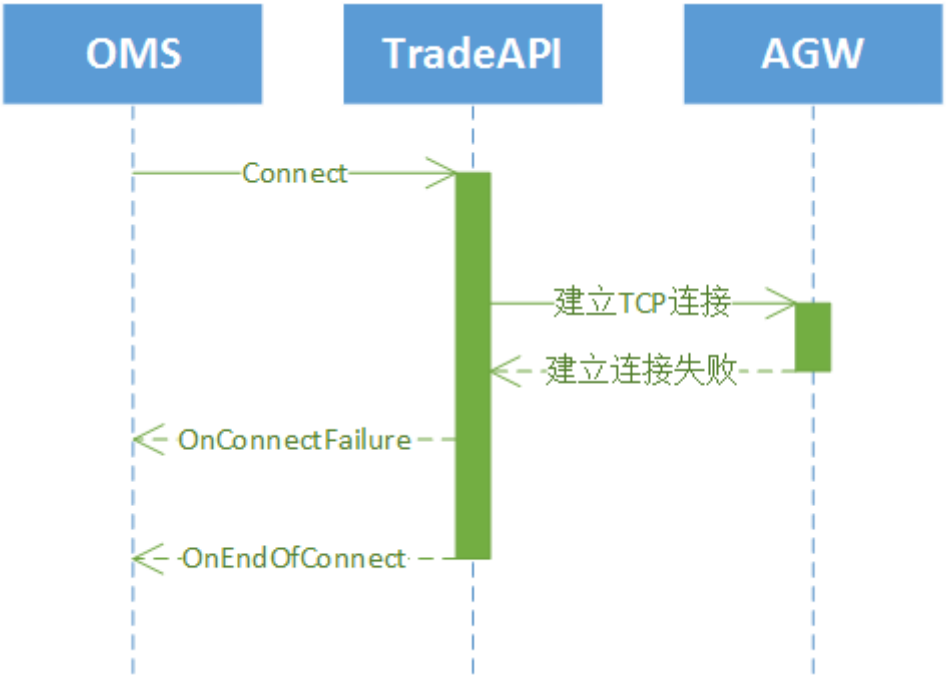
详细流程

正常流程


OMS
TradeAPI
AGW
Connect
建立TCP连接
建立连接成功
OnConnected
网关用户登录
登录成功
OnLogin
ReqCustLoginOther
客户号登录
登录成功
OnRspCustLoginResp
心跳
心跳应答
ReqCustLogoutOther
客户号登出
OnRspCustLogoutResp
Close
断开连接
OnClosed
OnEndOfConnect

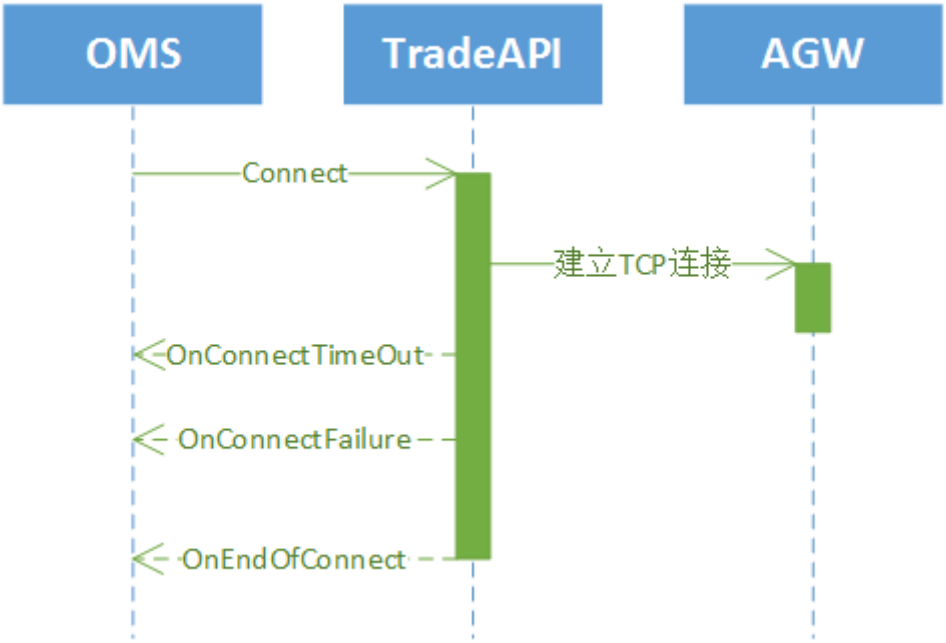
异常流程

- 连接失败



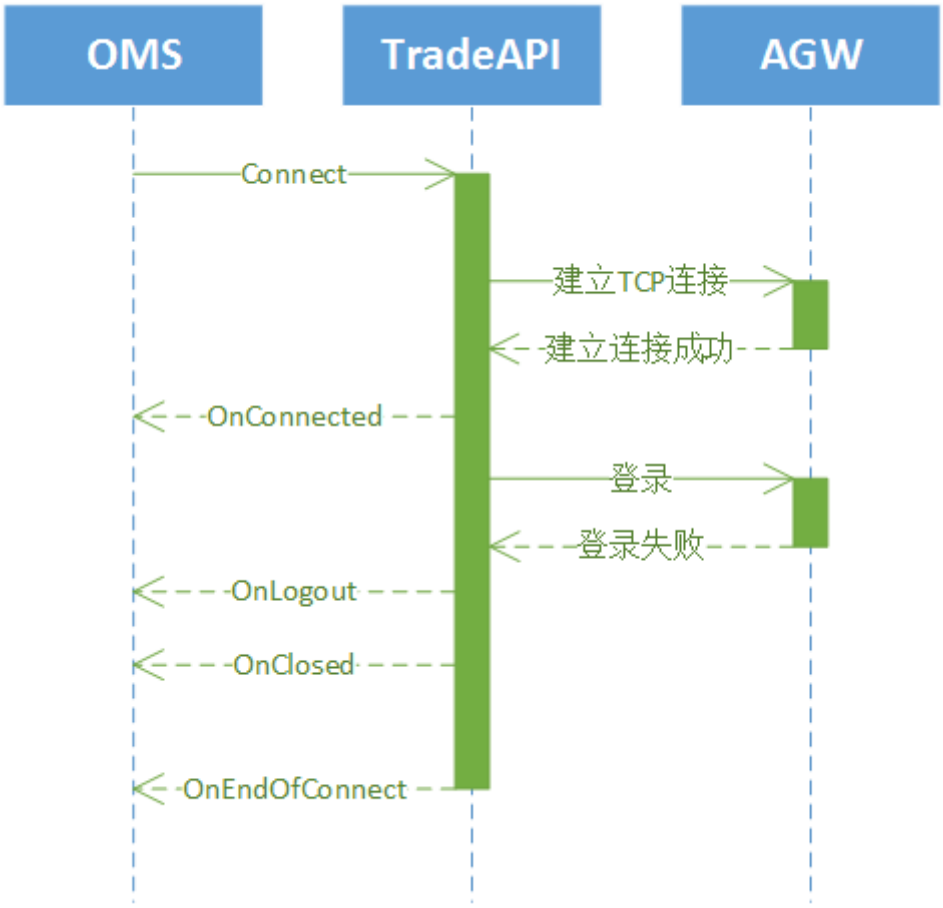
> **解决方法**：可能连接信息配置错误，或AGW尚未启动，请检查配置或确认环境情况后，再次调用connect

- 连接超时



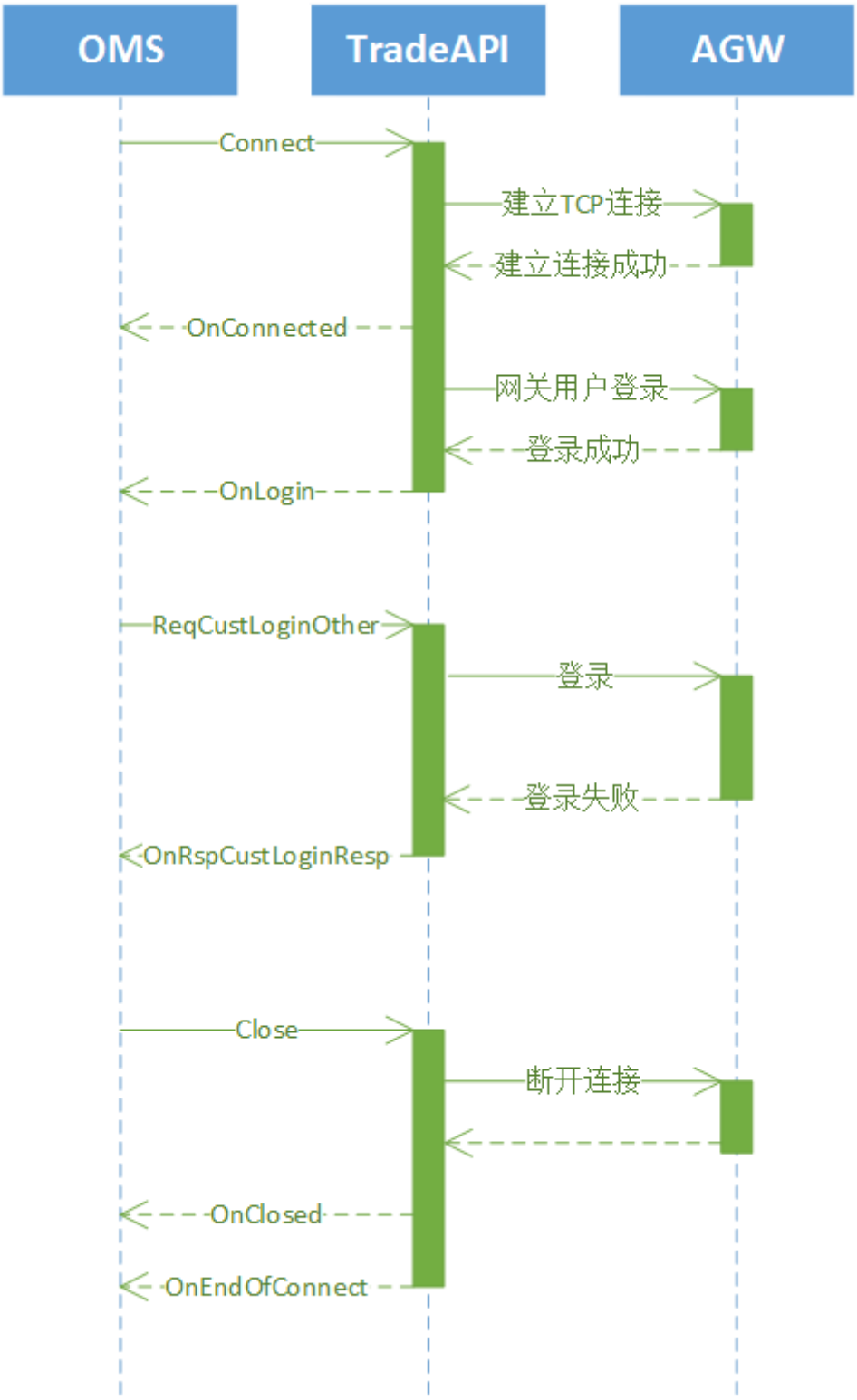
> **解决方法**：可能网络不稳定，或AGW异常，请检查环境情况后，再次调用connect

- 网关用户登录失败



> **解决方法**：可能网关用户名或密码错误，请确认后，再次调用connect

- 客户号登录失败



> **解决方法**：可能账户或密码错误，请确认后，再次调用ReqCustLoginOther

注意事项

- **连接失败**、**连接超时**、**网关用户登录失败**都会触发**ATPTradeAPI自动重连**。具体情况详见自动重连

3.3. 心跳机制

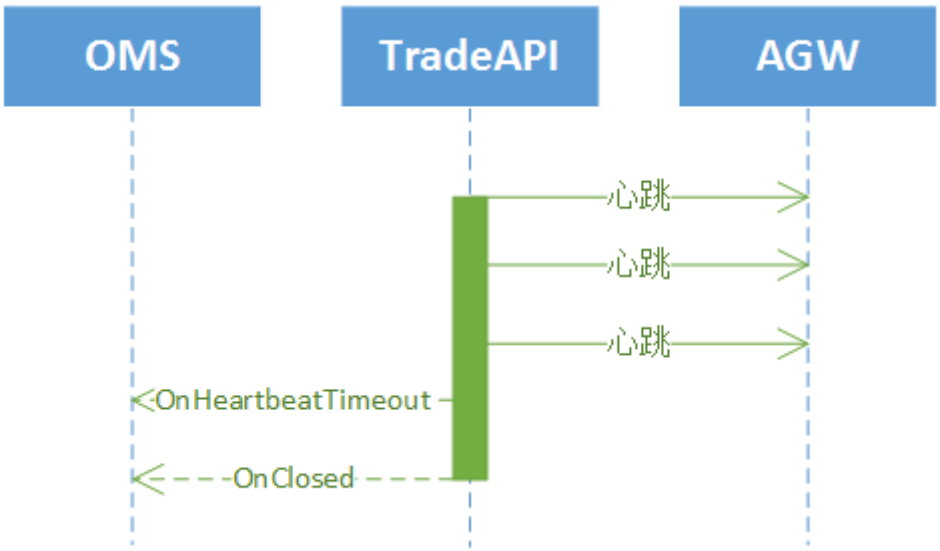
ATPTradeAPI和网关建立连接后，ATPTradeAPI和网关会双向发送心跳包，当任何一方超过三次心跳间隔未收到对方心跳包，会主动关闭连接

相关参数

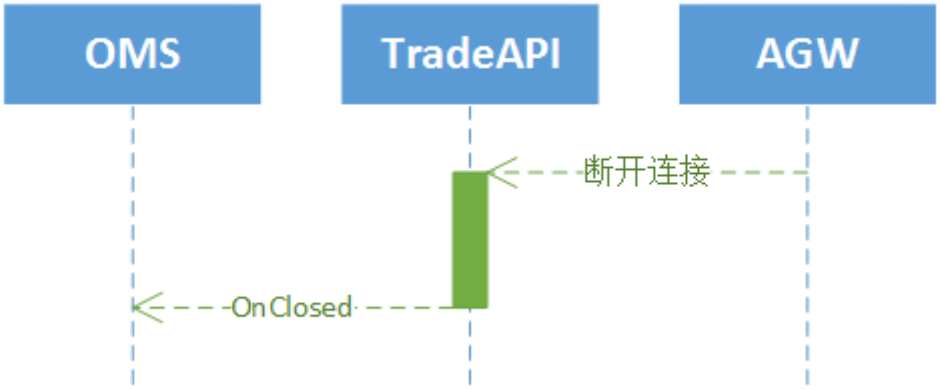
- **heartbeat_interval_milli** 心跳间隔时间

详细流程

- ATPTradeAPI检测到超时



- 网关检测到超时



注意事项

- **心跳超时**后会触发**ATPTradeAPI自动重连**。具体情况详见自动重连

3.4. 自动重连

连接异常，网关登录失败，以及心跳超时的情况下，都会触发ATPTradeAPI自动重连机制

基本流程

- ATPTradeAPI与AGW之间的连接断开后，ATPTradeAPI会重连当前断开的地址，如果没有连接成功会连接ATPConnectProperty.Locations中下一个地址；
- 如果到最后的位置会从头开始重连；
- 直到收到OnLogin回调，代表重连成功；

- 如重连次数使用完毕，则会触发OnEndOfConnection回调函数，自动重连结束；此时客户端可调用Connect重新发起连接。

相关参数

- **user** 网关用户名
- **locations** 主备网关的IP和端口，重连时
- **connect_timeout_milli** 超时连接时间
- **reconnect_time** 自动重试次数

注意事项

- 只有触发**OnEndOfConnection**回调后，才能再次发起连接调用

3.5. 回报同步

回报同步是用于客户端断线重连时，同步在断线期间系统收到的委托状态响应、成交回报信息。也可用于客户端宕机后，重传当日所有委托状态响应、成交回报信息。

基本流程

- ATPTradeAPI内部在收到网关登录成功响应后，会发起回报同步请求；
- AGW收到ATPTradeAPI的回报同步请求，根据回报同步中的**分区号和序号**信息确认下行断点，并将断点后的回报信息推送给ATPTradeAPI，如果传入**的分区号为0，序号为-1**，表示推送AGW该user下所有回报；
- 同步结束后，客户程序会收到OnLogin回调；

相关参数

- **user/password** 需要连接的网关用户名密码，在断线重连时要保证跟首次连接一致；
- **report_sync** 回报同步数据 分区号和序号；
- **mode** 为1时快速登陆，**关闭回报同步功能。**为0进行回报同步处理，

注意事项

- 客户端需记录**委托状态响应、成交回报**中的分区号和序号信息，断线重连时重新设置report_sync
- AGW缓存是按照连接用户名user分类的；所以需要客户端在调用Connect时需要**保持网关用户名不变**，否则会出现回报丢失的情况。

3.6. 订单管理

订单管理是ATPTradeAPI核心功能，提供**委托申报、委托响应、成交**全流程接口。以下规则是通用规则，对于**特殊业务或特殊功能**可能会新增接口，具体情况还得详见API

接口样式

- 每种业务类型使用不同的申报接口和同一个内部响应接口

```
// 委托申报，XXX代表任意多个字母，一般为具体业务或功能的英文名
ATPRetCodeType ReqXXXOrder(const ATPReqXXXOrderMsg* order);
// 响应接口
void OnRspOrderStatusInternalAck(const ATPRspOrderStatusAckMsg&
order_status_ack);
```

- 全业务提供同一个确认接口

```
void OnRspOrderStatusAck(const ATPRspOrderStatusAckMsg& order_status_ack);
```

- 每种业务类型使用不同的成交接口

```
// 成交接口，XXX代表任意多个字母，一般为具体业务或功能的英文名
void OnRspXXXTradeER(const ATPRspXXXTradeERMsg& trade_er);
```

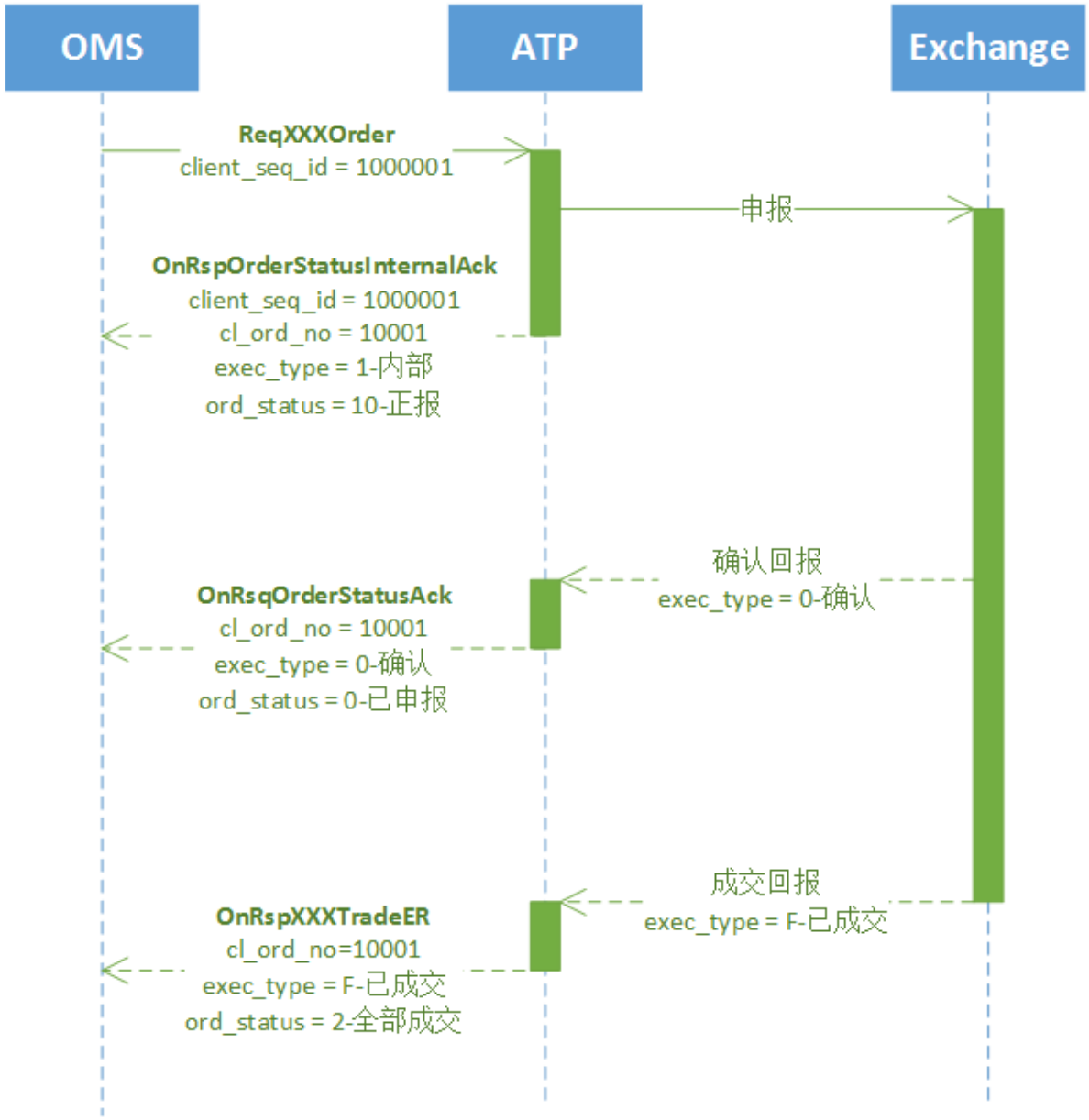
- 全业务提供同一个撤单接口

```
ATPRetCodeType ReqCancelOrder(const ATPReqCancelOrderMsg* cancel_order);
```

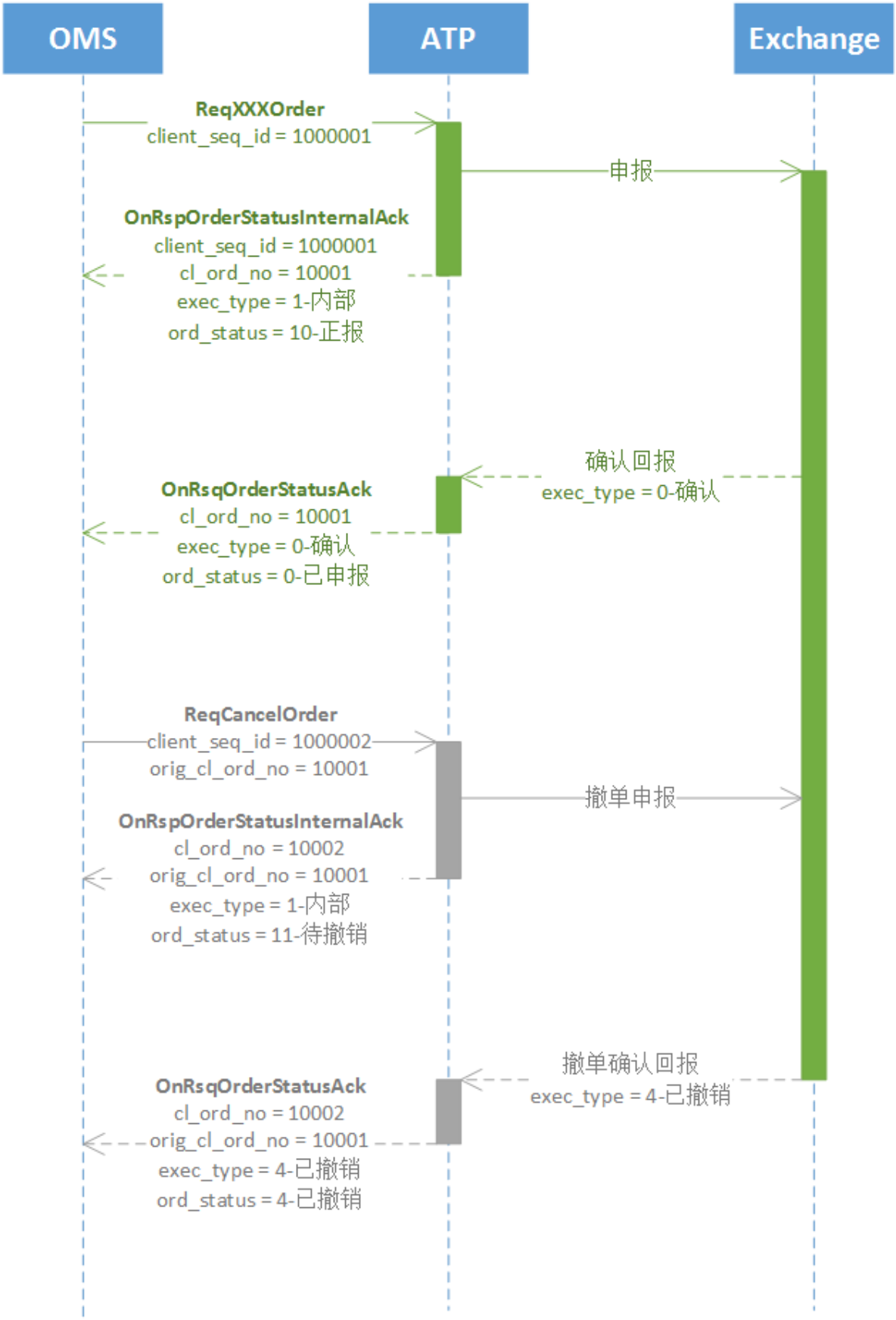
业务流程

交易时间处理流程

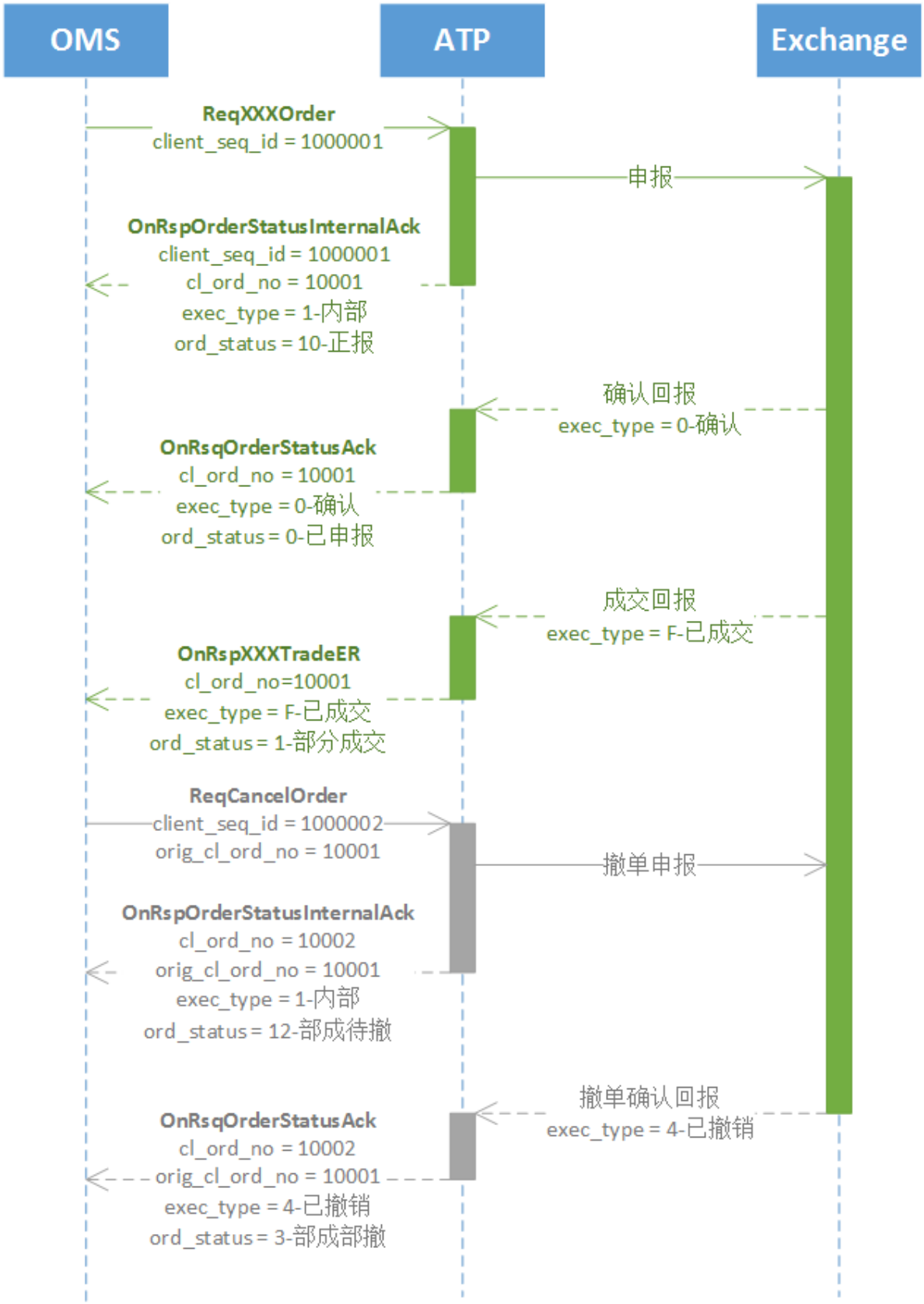
- 订单申报成交流程



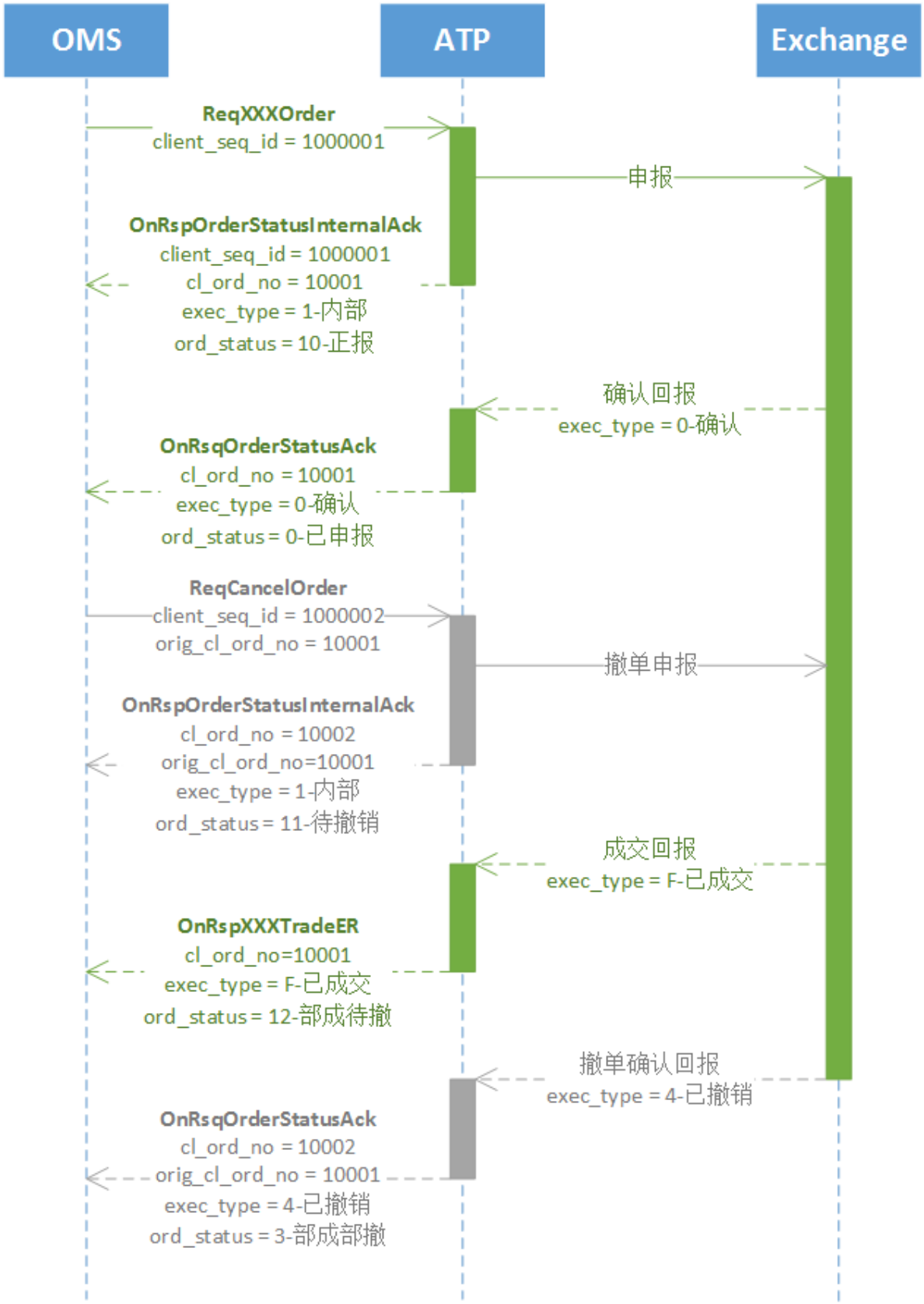
- 订单撤销流程



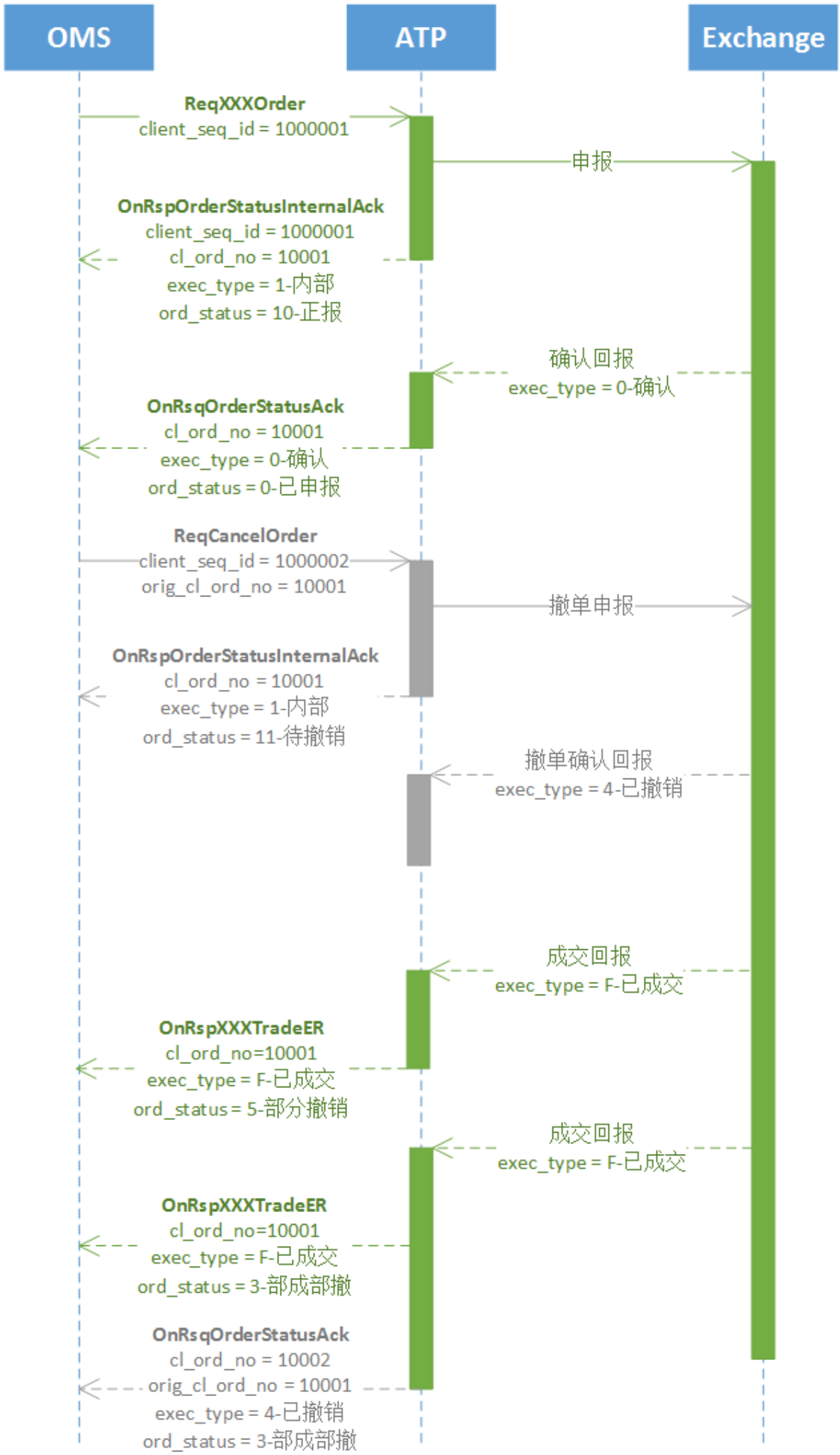
- 订单部撤流程（撤单申报在成交响应后）



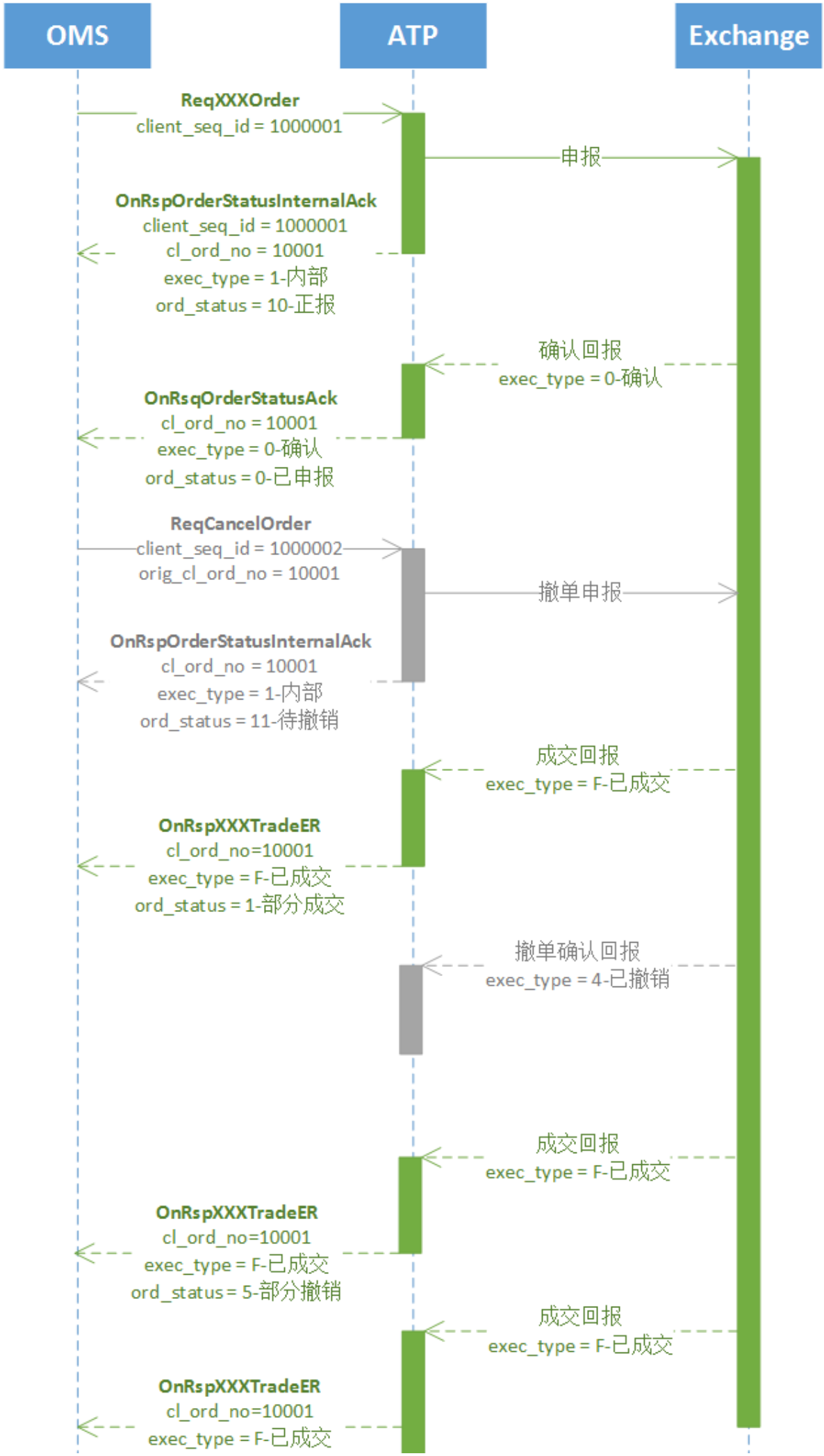
- 订单部撤流程（撤单申报在成交响应前）



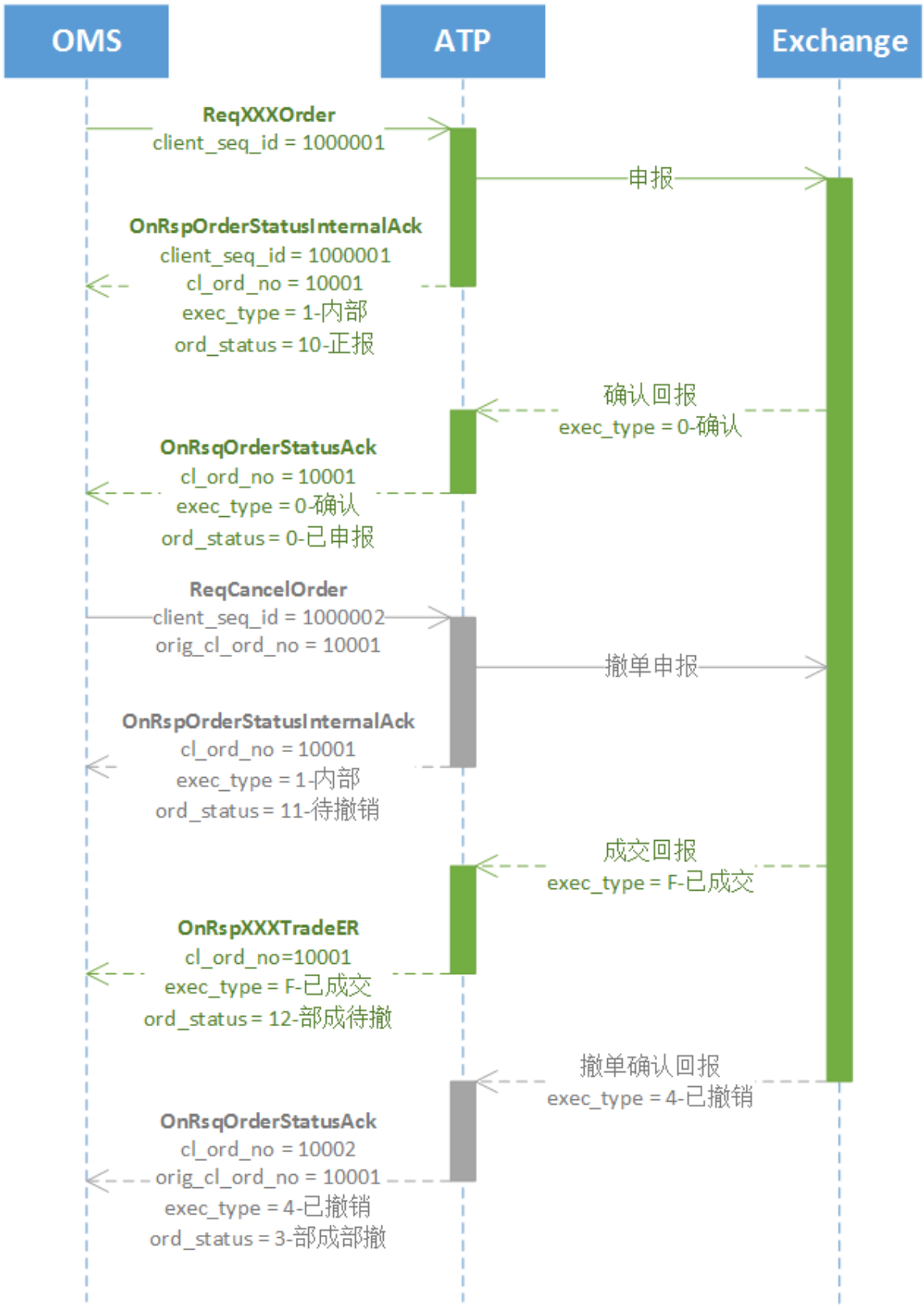
- 订单部撤流程（撤单响应在成交回报前）


OMS
ATP
Exchange
ReqXXXOrder
client_seq_id = 1000001
申报
OnRspOrderStatusInternalAck
client_seq_id = 1000001
cl_ord_no = 10001
exec_type = 1-内部
ord_status = 10-正报
确认回报
exec_type = 0-确认
OnRsqOrderStatusAck
cl_ord_no = 10001
exec_type = 0-确认
ord_status = 0-已申报
ReqCancelOrder
client_seq_id = 1000002
orig_cl_ord_no = 10001
撤单申报
OnRspOrderStatusInternalAck
cl_ord_no = 10001
exec_type = 1-内部
ord_status = 11-待撤销
撤单确认回报
exec_type = 4-已撤销
成交回报
exec_type = F-已成交
OnRspXXXTradeER
cl_ord_no=10001
exec_type = F-已成交
ord_status = 5-部分撤销
成交回报
exec_type = F-已成交
OnRspXXXTradeER
cl_ord_no=10001
exec_type = F-已成交
ord_status = 3-部成部撤
OnRsqOrderStatusAck
cl_ord_no = 10002
orig_cl_ord_no = 10001
exec_type = 4-已撤销
ord_status = 3-部成部撤

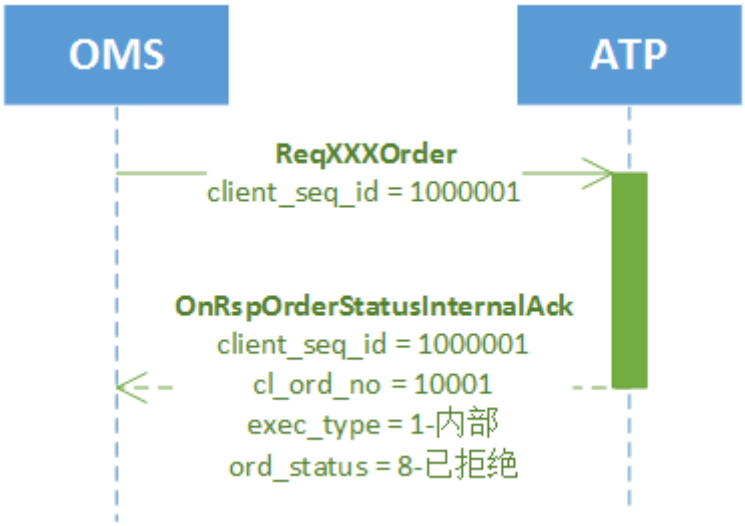
- 订单部撤流程（撤单响应在多笔成交回报中）


OMS
ATP
Exchange
ReqXXXOrder
client_seq_id = 1000001
申报
OnRspOrderStatusInternalAck
client_seq_id = 1000001
cl_ord_no = 10001
exec_type = 1-内部
ord_status = 10-正报
确认回报
exec_type = 0-确认
OnRsqOrderStatusAck
cl_ord_no = 10001
exec_type = 0-确认
ord_status = 0-已申报
ReqCancelOrder
client_seq_id = 1000002
orig_cl_ord_no = 10001
撤单申报
OnRspOrderStatusInternalAck
cl_ord_no = 10001
exec_type = 1-内部
ord_status = 11-待撤销
成交回报
exec_type = F-已成交
OnRspXXXTradeER
cl_ord_no=10001
exec_type = F-已成交
ord_status = 1-部分成交
撤单确认回报
exec_type = 4-已撤销
成交回报
exec_type = F-已成交
OnRspXXXTradeER
cl_ord_no=10001
exec_type = F-已成交
ord_status = 5-部分撤销
成交回报
exec_type = F-已成交
OnRspXXXTradeER
cl_ord_no=10001
exec_type = F-已成交

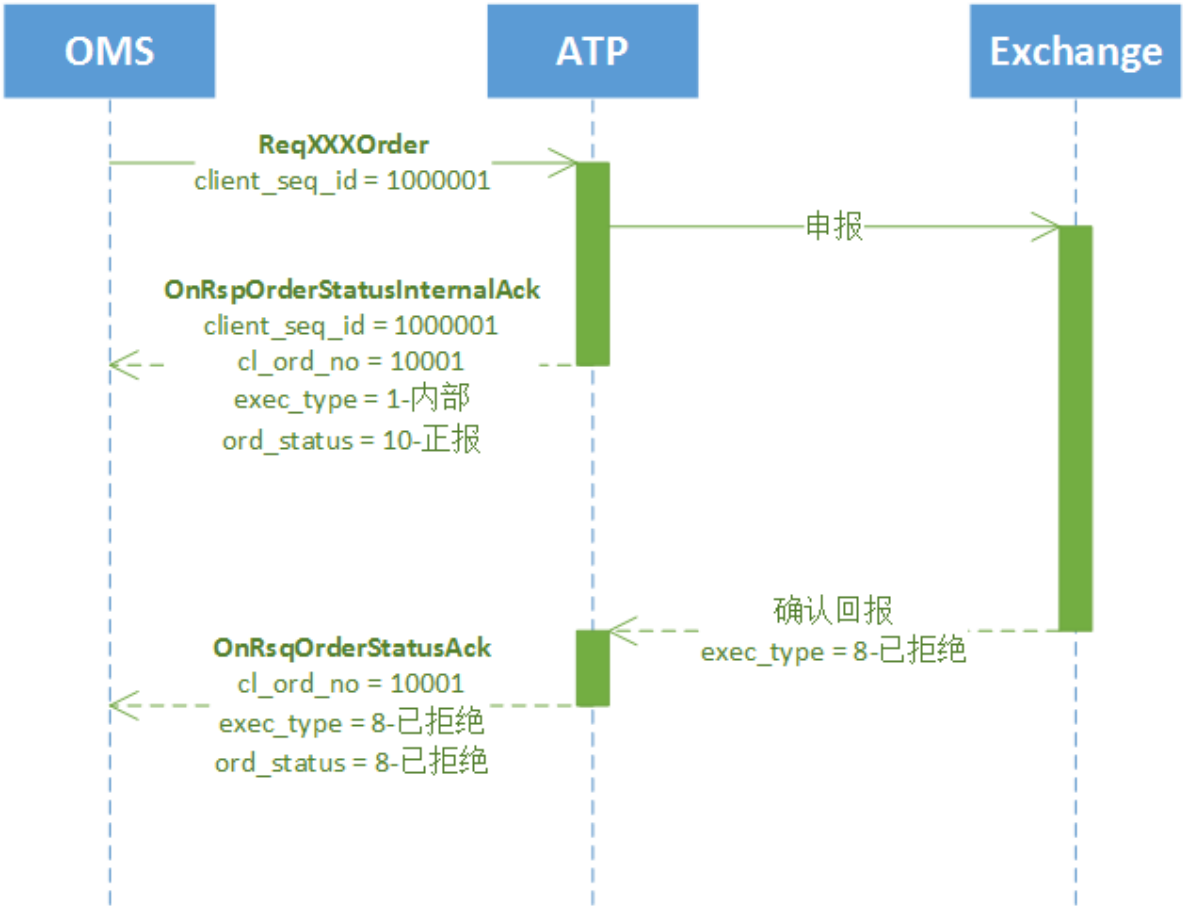
- 订单部撤流程（撤单申报在多笔成交响应中）



- 订单内部拒绝流程

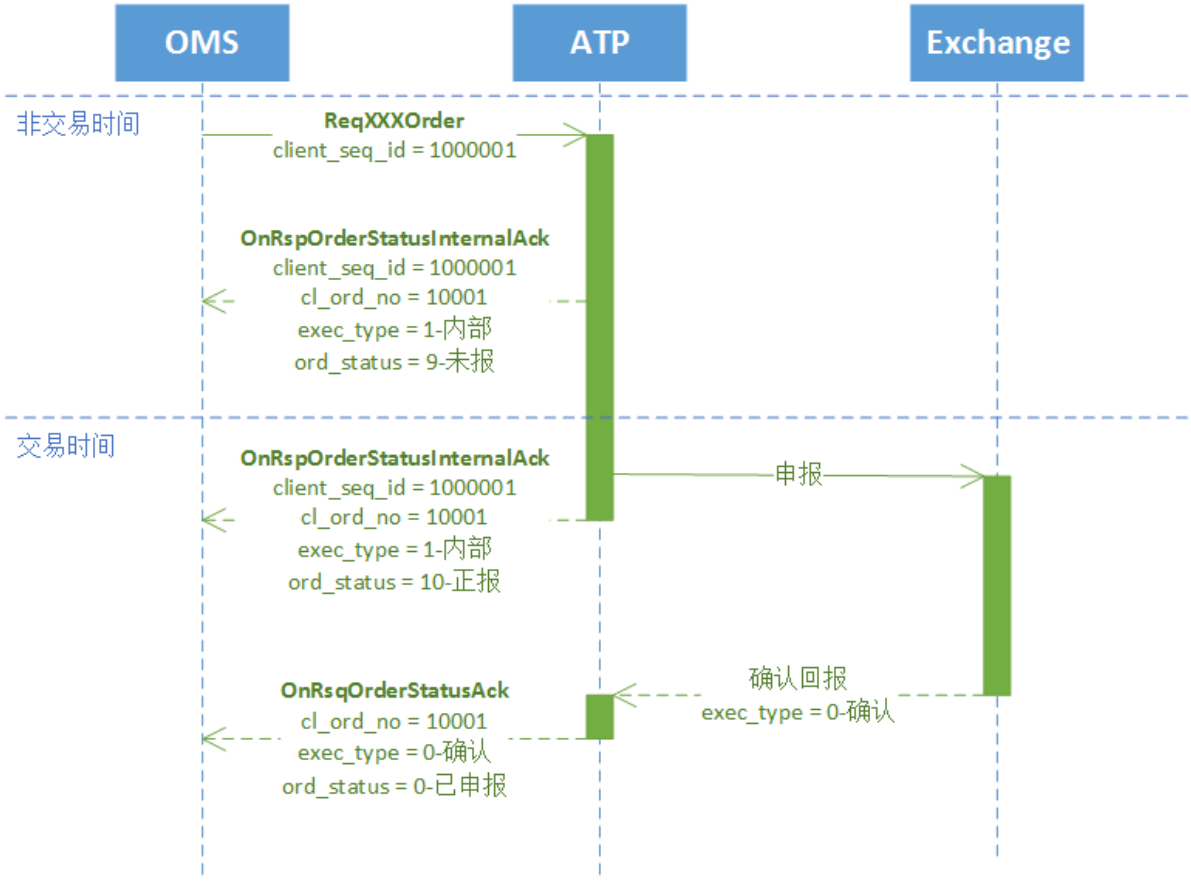


- 订单拒绝流程



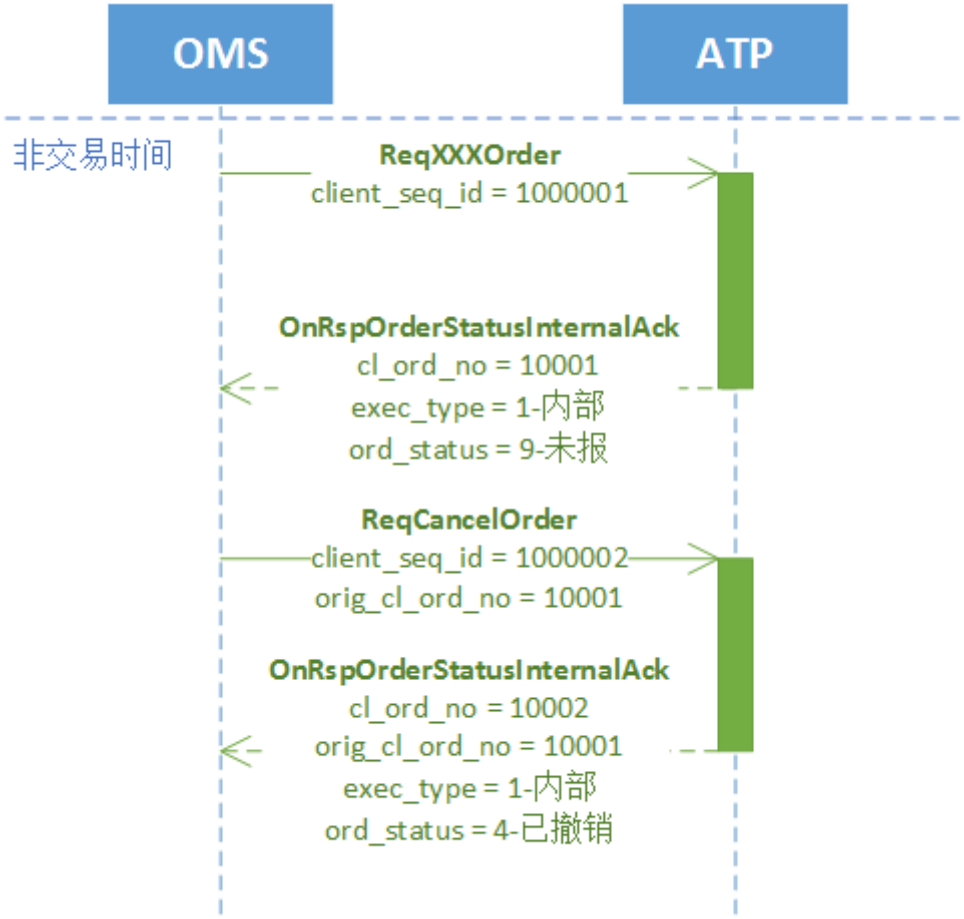
非交易时间处理流程

- 订单申报流程



到达交易时间之后，订单处理流程同上

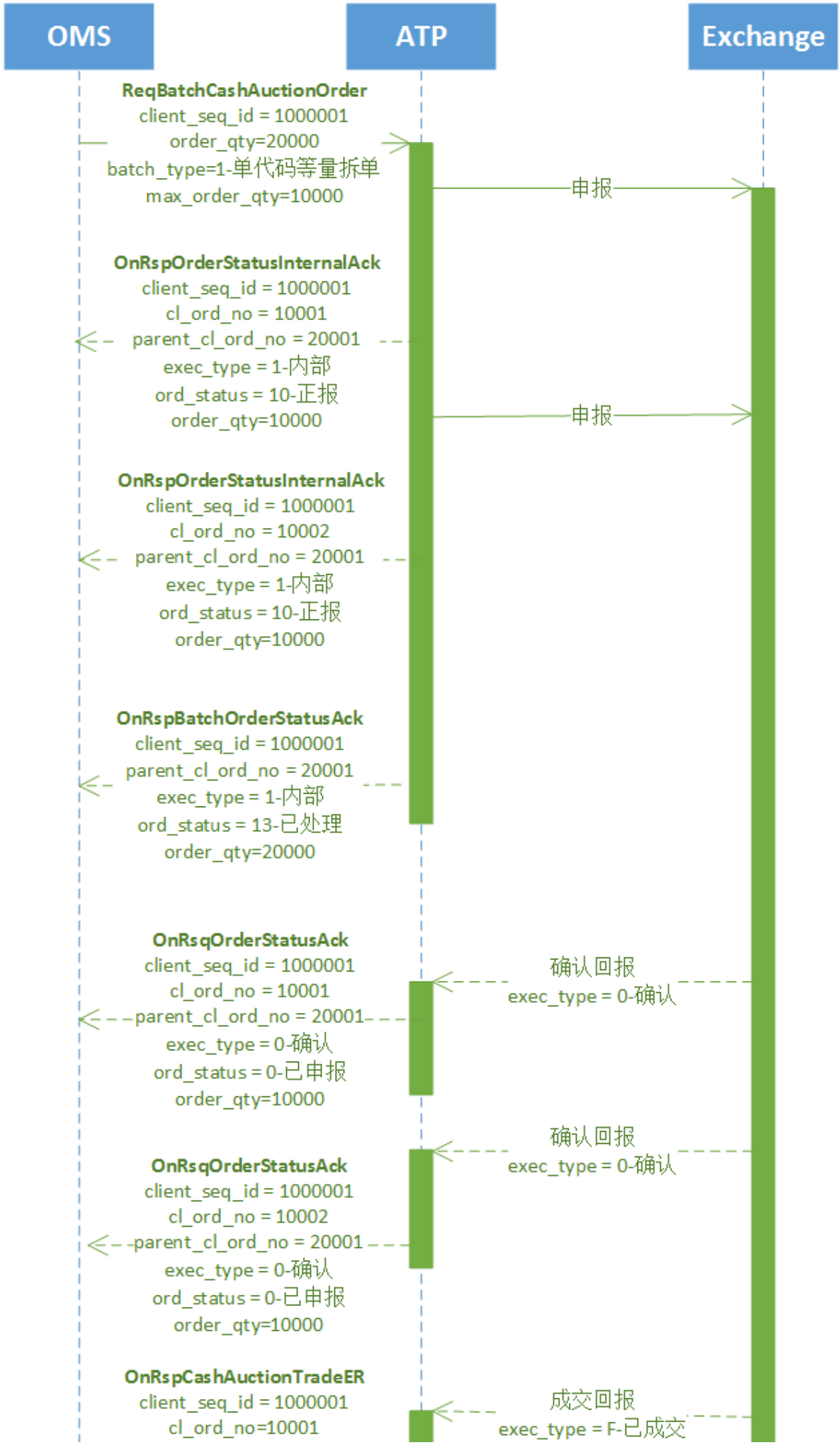
- 订单撤销流程

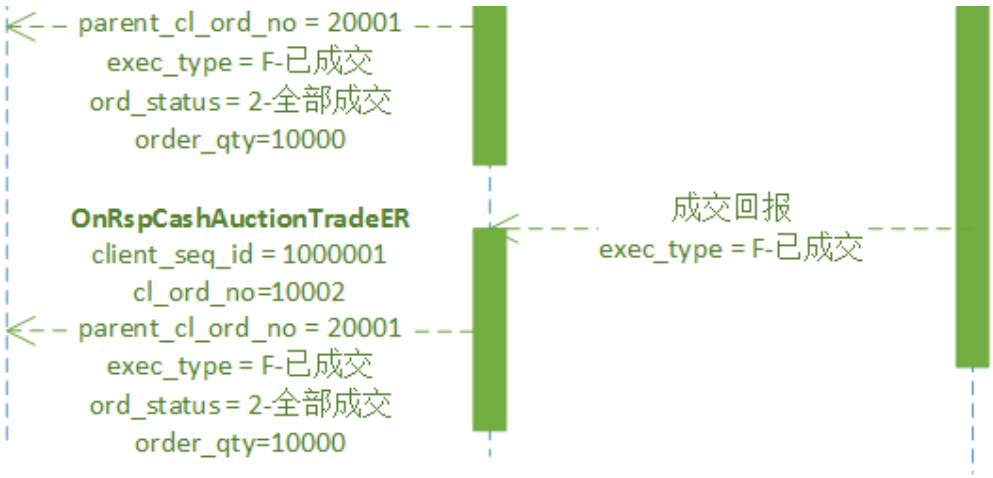


批量委托处理流程

- 订单申报流程

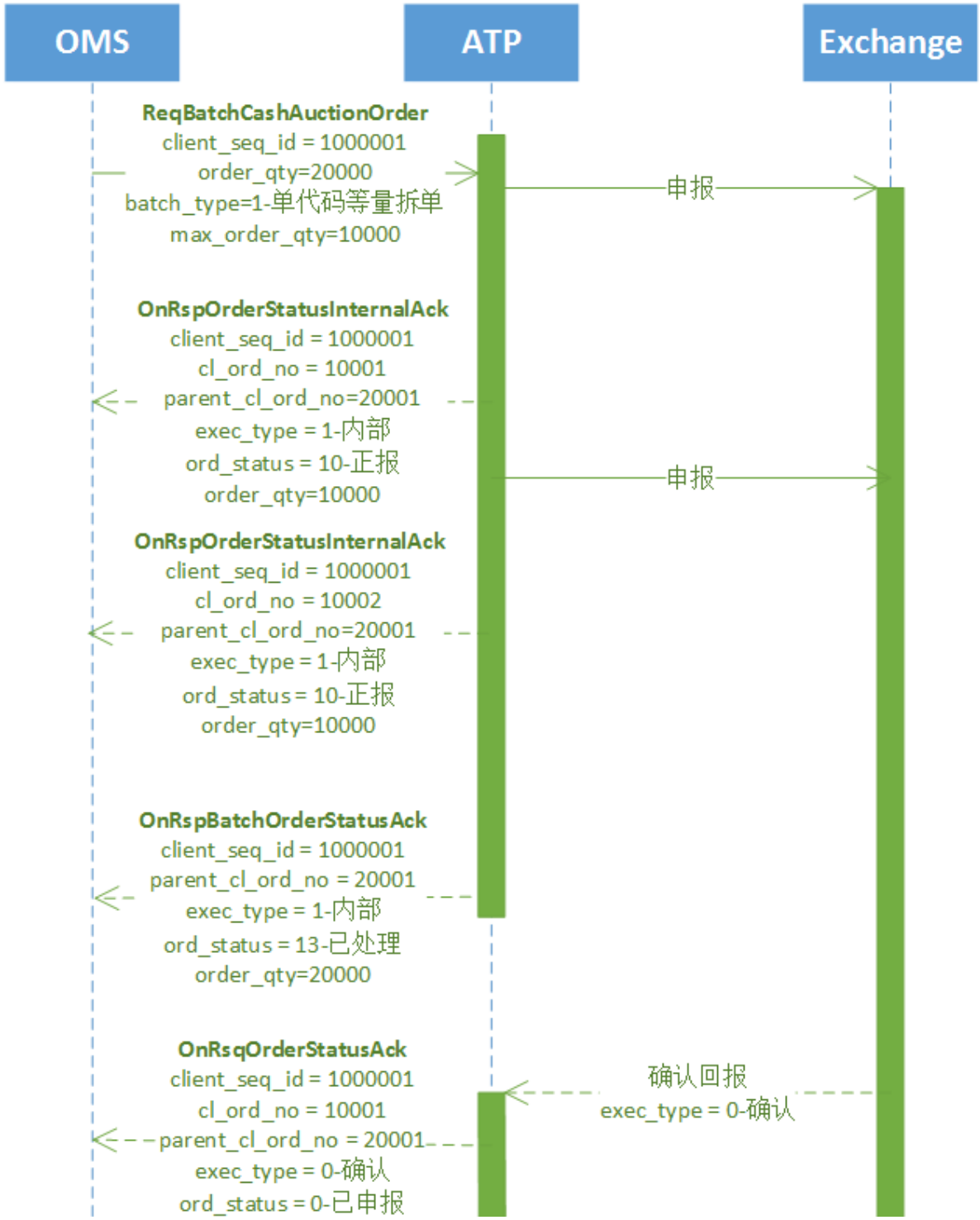
以单代码等量拆单为例

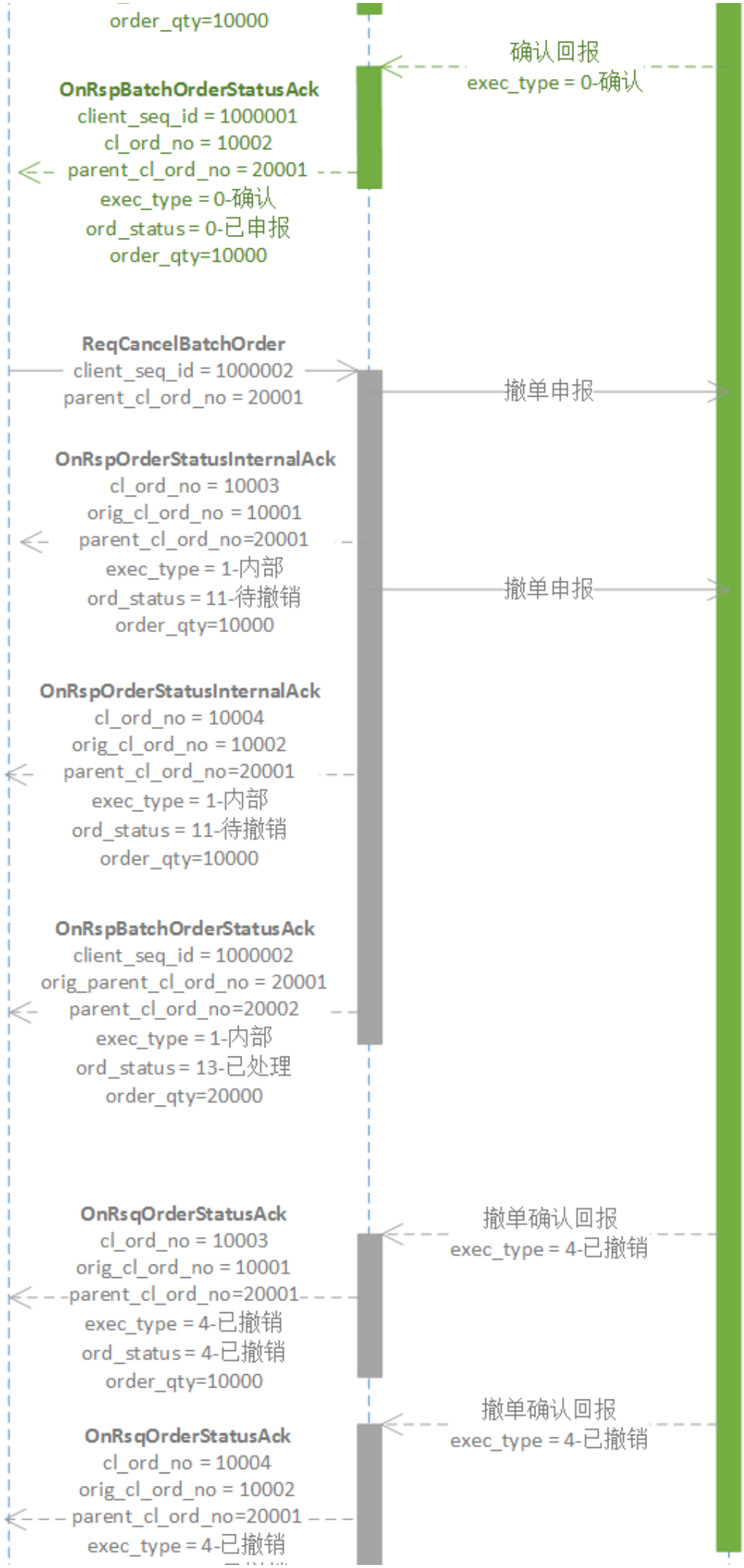

OMS
ATP
Exchange
ReqBatchCashAuctionOrder
client_seq_id = 1000001
order_qty=20000
batch_type=1-单代码等量拆单
max_order_qty=10000
申报
OnRspOrderStatusInternalAck
client_seq_id = 1000001
cl_ord_no = 10001
parent_cl_ord_no = 20001
exec_type = 1-内部
ord_status = 10-正报
order_qty=10000
申报
OnRspOrderStatusInternalAck
client_seq_id = 1000001
cl_ord_no = 10002
parent_cl_ord_no = 20001
exec_type = 1-内部
ord_status = 10-正报
order_qty=10000
OnRspBatchOrderStatusAck
client_seq_id = 1000001
parent_cl_ord_no = 20001
exec_type = 1-内部
ord_status = 13-已处理
order_qty=20000
OnRsqOrderStatusAck
client_seq_id = 1000001
cl_ord_no = 10001
parent_cl_ord_no = 20001
exec_type = 0-确认
ord_status = 0-已申报
order_qty=10000
确认回报
exec_type = 0-确认
确认回报
exec_type = 0-确认
OnRsqOrderStatusAck
client_seq_id = 1000001
cl_ord_no = 10002
parent_cl_ord_no = 20001
exec_type = 0-确认
ord_status = 0-已申报
order_qty=10000
OnRspCashAuctionTradeER
client_seq_id = 1000001
cl_ord_no=10001
成交回报
exec_type = F-已成交



- 订单撤销流程

以单代码等量拆单为例




order_qty=10000
确认回报
exec_type = 0-确认
OnRspBatchOrderStatusAck
client_seq_id = 1000001
cl_ord_no = 10002
parent_cl_ord_no = 20001
exec_type = 0-确认
ord_status = 0-已申报
order_qty=10000
ReqCancelBatchOrder
client_seq_id = 1000002
parent_cl_ord_no = 20001
撤单申报
OnRspOrderStatusInternalAck
cl_ord_no = 10003
orig_cl_ord_no = 10001
parent_cl_ord_no=20001
exec_type = 1-内部
ord_status = 11-待撤销
order_qty=10000
撤单申报
OnRspOrderStatusInternalAck
cl_ord_no = 10004
orig_cl_ord_no = 10002
parent_cl_ord_no=20001
exec_type = 1-内部
ord_status = 11-待撤销
order_qty=10000
OnRspBatchOrderStatusAck
client_seq_id = 1000002
orig_parent_cl_ord_no = 20001
parent_cl_ord_no=20002
exec_type = 1-内部
ord_status = 13-已处理
order_qty=20000
OnRsqOrderStatusAck
cl_ord_no = 10003
orig_cl_ord_no = 10001
parent_cl_ord_no=20001
exec_type = 4-已撤销
ord_status = 4-已撤销
order_qty=10000
撤单确认回报
exec_type = 4-已撤销
OnRsqOrderStatusAck
cl_ord_no = 10004
orig_cl_ord_no = 10002
parent_cl_ord_no=20001
exec_type = 4-已撤销
撤单确认回报
exec_type = 4-已撤销

批量子单的撤销，可使用通用撤单函数（ReqCancelOrder）来撤销；具体撤单流程同上

执行状态与订单状态说明

- 原单是订单

exec_type	ord_status	描述
1-内部确认	10-正报	交易时段订单申报交易所；非交易时段转交易时段时订单报到交易所
1-内部确认	9-未报	非交易时间段缓存在ATP
1-内部确认	8-已拒绝	ATP拒单
0-交易所确认	0-已报	订单被交易所确认
0-交易所确认	11-待撤销	订单被交易所确认，该单已被撤单关联，但撤单尚未撤单成功
4-已撤销	3-部成部撤	订单部分成交部分撤销，可查看reject_reason_code字段，获取交易所撤单原因
4-已撤销	4-撤成	订单已全部撤销，可查看reject_reason_code字段，获取交易所撤单原因
8-已拒绝	8-已拒绝	订单被交易所拒绝，可查看reject_reason_code字段，获取交易所拒绝原因

- 原单是撤单

exec_type	ord_status	描述
1-内部确认	11-待撤销	原单待撤销，撤单已申报交易所
1-内部确认	12-部成待撤	原单部成待撤，撤单已申报交易所
1-内部确认	10-正报	原单正报，撤单失败，需要查看拒绝原因代码
1-内部确认	8-已拒绝	原单已是拒绝状态，无法被撤单
1-内部确认	5-部分撤销	原单当前为部分撤销状态，无法再被撤单
1-内部确认	4-撤成	原单已是撤销状态，无法再被撤单
1-内部确认	3-部成部撤	原单已是部成部撤状态，无法再被撤单
1-内部确认	2-全成	原单已是全部成交状态，无法再被撤单
1-内部确认	1-部成	原单部分成交，撤单失败，需要查看拒绝原因代码
1-内部确认	0-已报	原单已被交易所确认，撤单失败，需要查看拒绝原因代码
4-已撤销	4-撤成	原单已全部撤销
4-已撤销	5-部撤	原单被部分撤销，当交易所撤单响应比成交先到时出现(中间状态)
4-已撤销	3-部成部撤	原单部分已成交，剩余部分已全部撤销
8-已拒绝	0-已报	交易所拒单，原单已被交易所确认，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	1-部成	交易所拒单，原单已是部分成交状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	2-全成	交易所拒单，原单已是全部成交状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	3-部成部撤	交易所拒单，原单已是部成部撤状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	4-撤成	交易所拒单，原单已是撤销状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	5-部分撤销	交易所拒单，原单已是部分撤销状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	8-已拒绝	交易所拒单，原单已是拒绝状态，可查看reject_reason_code字段，获取交易所拒绝原因
8-已拒绝	10-正报	交易所拒单，原单已是被交易所确认状态，可查看RejectReasonCode字段，获取交易所拒绝原因

- 成交状态

exec_type	ord_status	描述
F-成交	1-部成	部分成交
F-成交	2-全成	全部成交
F-成交	3-部成部撤	部成部撤
F-成交	12-部成待撤	部成待撤
F-成交	5-部撤	部分撤销，当交易所撤单响应比成交先到时出现(中间状态)

3.7. 查询服务

ATPTradeAPI提供了一些交易数据查询服务。并支持分页查询功能，每种不同的查询使用不同的查询接口并对应唯一的查询结果回调，具体接口定义详见API文档。

接口样式

```
// 查询请求，XXX代表任意多个字母，一般为具体业务或功能的英文名
ATPRetCodeType ReqXXXQuery(const ATPReqXXXQueryMsg* query_msg);
ATPRetCodeType ReqExtQueryXXX(const ATPReqExtQueryXXXMsg* ext_query_msg);
// 查询响应，XXX代表任意多个字母，一般为具体业务或功能的英文名
void OnRspXXXQueryResult(const ATPRspXXXQueryResultMsg& query_result);
void OnRspQueryXXXResult(const ATPRspQueryXXXResultMsg& query_result);
```

分页查询

为了控制查询结果包大小，ATPTradeAPI支持分页查询，客户端可以指定每页的记录数（若填0，则以券商限定的最大记录数返回），并指定起始查询索引号查询非起始页的数据，最终我们可以通过比较返回的last_index+1=total_num来判断是否查询到所有数据

```
// 请求参数，XXX代表任意多个字母，一般为具体业务或功能的英文名
struct TRADE_API ATPReqXXXQueryMsg
{
    ......
    ATPReportIndexType  query_index;    ///< 指定起始查询索引号 （必填）,将返回索引号以后的订单，填0表示从头开始查
    ATPTotalNumType     return_num;     ///< 指定查询返回数量 （必填）,填0表示不指定
    ATPReturnSeqType    return_seq;     ///< 返回顺序，按时间排序 （必填）
    ......
};

// 响应结果，XXX代表任意多个字母，一般为具体业务或功能的英文名
struct TRADE_API ATPRspXXXResultMsg
{
    ......
    ATPReportIndexType  last_index;     ///< 最后一笔查询结果的索引号，组合个数为0时，该值为-1
    ATPTotalNumType     total_num;      ///< 订单记录总数
    ......
};
```

例如：

某次查询总共有210条记录，每次查询100，则每次查询的请求和结果如下：

查询次数	query_index	return_num	last_index	total_num
1	0	100	99	210
2	100	100	199	210
3	200	100	209	210

3.8. 算法交易

ATPTradeAPI提供了算法类交易接口，具体接口定义详见API文档。

支持品种

现阶段支持沪深市场主板、科创板、创业板的股票、债券、基金的竞价买卖业务。

策略介绍

- VWAP
    - Volume Weighted Average Price, 成交量加权平均价格算法，旨在指定的时间范围内, 根据市场成交量分布进行下单，降低市场冲击，最小化与市场VWAP的偏差。
    - 从模型而言，VWAP策略是使在指定时间段所执行母单的成交均价贴近相应时间段的市场成交均价。
    - 华锐算法对VWAP策略模型进行了升级，引入股票特性、量化数据、时间随机、数量随机、对市场趋势预测等指标因子。有效保证母单均价更贴近甚至优于市场成交均价。
- TWAP
    - Time Weighted Average Price, 时间加权平均价格算法，旨在指定的时间范围内均匀执行，降低市场冲击，最小化与市场TWAP的偏差。
    - 从模型而言，TWAP策略是将交易时间进行均匀分割，并在每个分割节点上等量拆分订单进行成交。
    - 华锐算法对TWAP策略模型进行了升级，引入股票特性、量化数据、时间随机、数量随机等指标因子。有效解决TWAP过于机械，但同时又保留时间均匀的特性。
- POV
    - Percent of Volume，比例成交算法，也称为跟量。根据用户设置的量比比例来参与市场成交的算法，市场放量时多成交，缩量时减少成交，整个执行过程中，订单成交量与对应时间内的市场成交量之比接近用户设置的量比比例。
    - 华锐算法对POV策略模型进行了升级，在跟量的同时，大大降低对市场的冲击。
- MOO
    - 在考虑市场冲击的前提下,结合历史和实时开盘的情况,使成交均价尽可能贴近开盘价格。

策略	描述	应用场景
VWAP	成交量加权平均价格算法，旨在指定的时间范围内，根据市场成交分布进行下单，降低市场冲击，最小化与市场VWAP的偏差。	在指定的时间范围内，参考成交分布，相对均匀完成订单，时间范围如分钟级、1小时或全天 篮子组合交易 调仓 二级市场增持或减持 追求成交均价贴近市场均价
TWAP	时间加权平均价格算法，旨在指定的时间范围内均匀下单，降低市场冲击，最小化与市场TWAP的偏差。	在指定的时间范围内，按时间进度均匀完成订单，时间范围如分钟级、1小时或全天 篮子组合交易 调仓 二级市场增持或减持 追求成交均价贴近市场均价
POV	成交量比例算法，也称跟量。根据用户设置的量比比例来参与市场成交的算法，市场放量时多成交，缩量时少成交，整个执行过程中，订单成交量与对应时间内的市场成交量之比接近用户设置的比例。	跟随市场节奏，尽快完成订单 减少市场冲击，按较低的市场参与比例持续执行
MOO	开盘算法，在考虑市场冲击的前提下，结合历史分析数据和实时行情，使成交均价尽可能贴近开盘价格	追求跟踪开盘价的场景 在开盘后尽快完成

参数介绍

- 母单限价/市价
    - 限价母单：
    - 对于买入，控制子单委托价格的上限，即子单委托价格<=母单限价；
    - 对于卖出，控制子单委托价格的下限，即子单委托价格>=母单限价。
    - 市价母单：
    - 子单委托价格没有买卖上下限的控制。算法子单委托价格一般在最新价左右，且符合涨跌停等交易规则。
- 母单数量
    - 母单数量为目标数量，指算法母单需要完成成交的数量。
- 开始时间
    - 算法母单开始执行的时间。
    - 对于期望立即执行的母单，可设置开始时间在当前时间之前，比如开始时间使用默认的9:30:00。
- 结束时间
    - 算法母单停止执行的时间。
    - 在数量、价格、行情合理的情况下，通常母单会100%完成执行。如果存在极端情况，比如股票涨停而无法100%买入，那么过了结束时间后，策略会自动撤销所有已报未全成委托，待所有委托撤销之后，策略状态更新为已过期。
- 交易时长

    - 算法母单执行的交易时间长度，是对于策略时间设置的一个补充。使用场景：每天下母单次数多、下母单时间点不确定、而母单交易时长相对明确，比如根据量化模拟，信号触发后立即执行5分钟。
    - 交易时长参数会考虑有效交易时间，比如在11:00:00时执行1小时交易时长，那么对于A股股票而言，算法实际执行时间为11:00:00~13:30:00。
    - 如果设置开始时间为10:00:00、交易时长10分钟，那么母单的执行时间为10:00:00~10:10:00。
    - 如果设置开始时间为0、交易时长10分钟，那么母单的执行时间为从当前时间点开始立即执行10分钟。
    - 如果同时设置了结束时间与交易时长，那么算法会以结束时间为准。
- 量比比例
    - 量比指母单的成交数量与母单执行期间市场的总成交数量之比。对于跟量、跟价策略，此参数是作为目标比例来执行；而对于其他策略，如果TWAP、VWAP，则作为量比上限来控制控制。

4. 其他事项

网关拒绝响应

- 当请求字段非法、所收到接入管控时，会收到OnRspBizRejection网关拒绝响应回调，具体错误原因见**reject_reason_code**

一户一地账户与一户两地账户

- ATP支持单中心和双中心两种部署方案，在双中心部署方案里支持一户两地账户；
- 双中心部署，两个中心分别部署在上交所或深交所机房附近，ATP到交易所时延最低；
- 一户两地账户，在上海中心或深圳中心只存放**该账户**下上海证券账户或深圳证券账户的**资金、持仓、订单数据**；
- 使用一户两地账户的投资者，每笔请求需要填写**正确的证券账户**，否则就会出现*219-分区号没找到*的网关拒绝响应；
- 使用一户两地账户的投资者，根据双中心部署就近报盘的特性，可以同时部署两个策略程序，分别连接上海中心和深圳中心的网关，来达到链路时延最低的效果；
- 投资者可以根据客户号登录响应里**是否一户两地账户**字段，来判断当前账户是否已开通一户两地账户；
- 目前使用的ATP是否支持一户两地账户，以及需要开通一户两地账户，可向券商咨询。

客户端故障重启

- 当遇到客户端因故障重启，客户端所有的订单信息全部丢失情况，可通过以下方法重建订单信息
    - 按照ATPTradeAPI调用流程进行登录，登录成功后再进行分页做委托查询来重构委托列表；
    - 调用Connect时可使用特殊的分区索引信息即：ATPConnectProperty.report_sync中只设置(0，-1)；此时在登录的过程中AGW将推送所有的回报信息给客户端，由客户端重构委托列表；

> **注意** 因客户端故障重启存在委托丢失的情况；即发送给AGW（含未发送或者在途中的）的委托

多点登录

- ATP支持同一客户号建立多个连接
- 同一客户号的多个连接均可收到跟该客户号相关的所有响应（需券商开启此功能）

线程模型

- API包含一个发送线程、一个回调线程和一个日志打印线程；
- 所有客户线程发起的请求，都是放到发送队列，再由发送线程发出；
- 回调线程会处理网络上的响应包，并调用客户注册的回调函数；

> **注意** 禁止在回调函数里做其他调用API函数操作，如调用连接函数、订单下达函数、查询函数等

> **注意** 禁止在回调函数里做任何耗时的操作

> **注意** 建议在回调函数只做简单赋值、内存拷贝操作，由客户线程处理拷贝出的数据

5. 常见问题

1. 对于客户端如何应对AGW集群的主备高可用？

- AGW主备模式下，每个Agw登录用户如user1都需要维护各自的分区索引信息；如主AGW挂了，客户端在连接备时需要使用各自登录用户名并传入与之对应的分区索引信息；
- 如果整个集群都挂了，之后集群走应急启动；建议客户端做全量接收即将分区索引信息设置为(0,-1)或者将分区索引信息填空；

2. 客户端什么情况下需要重新发起Connect?

客户端收到事件EndofConnect,为内部重连失败，需要开发者主动再次发起connect;

3. 客户端能否使用与编译API不一致的GCC版本？

使用时需注意gcc版本与API编译gcc版本一致。否则会存在编译或者运行时core问题。

4. 主动调用Close接口那个事件是真实的连接彻底关闭不会内部重连？

以EndofConnect事件为准，Closed事件可认为是通知客户端内部的状态；

5. 如客户端使用多个api实例，应该为不同的api实例创建不同的ClientHandler对象吗，ClientHandler的生命周期如何？

建议不同的API实例使用不同的clienthandler；它的生命周期长于api实例，因为api实例内部引用了clienthandler的指针；

6. 回报消息中的索引记录为什么是间断不连续的？

此索引号是非严格递增的即递增但非连续；

7. 客户端发起Connect后立即发委托，造成关闭连接是什么原因？

需要等login事件后才能进行委托；

8. 客户使用boost环境开发出现core?

boost版本需要与API一致(1.62.0),否则存在运行时core或者编译问题；(API_rc1.7版本以后不依赖boost环境)

9. API能否前向兼容？

如发生增加/删除/修改字段定义等需要重新编译客户端；

10. 主备切换客户端如何处理？

- 当主AGW_11退出时客户端与AGW_11连接断开，此时客户端需主动连接备AGW_12；
- 客户端在连接备AGW_12时需要使用与连接主AGW_11相同的网关登陆用户名（user1）以及与之对应的分区索引信息， - 否则可能会出现登陆不成功现象；
- 注意：如果客户端需要接收该网关登陆用户名下所有的成交回报信息，则在连接时使用该登陆用户名以及特殊分区索引信息(0,-1)即可；

11. 可以在回调函数中进行耗时的处理么？

建议回调函数中仅做简单的赋值处理；下行单个API实例的下行仅一个独立线程。

12. API是否线程安全？

Connect不是线程安全,回调的接口是线程安全的。

13. 一个客户端可以与AGW建立多个连接吗？

一个客户端可以与AGW建立多个连接，但一个API对象只与AGW建立一个连接，要建立多个连接则需要创建多个API对象实例；

14. Connect的函数ReportSync同步回报数据map默认值为空，这个作用是什么？客户端需要维护它？

对于首次登录时，默认填空；之后接收成交回报、委托状态响应时需要更新该map,当客户端收到EndOfConnect事件主动调用Connect时需要把该reportsync参数置为客户端记录的map值，如此客户端能收到断开连接后未收到的回报数据；

15. Connect的函数参数Locations连接地址Vector<std::string>每个地址为"IP:PORT"形式，多个地址是主备还是负载均衡？

主备关系，默认第一个地址为主，如断开连接时API将会自动在主备之间轮询连接直到连接成功或者重连次数使用完毕为止。

16. Connect函数的用户是客户账户还是类似营业部柜员的概念

这个账户是连接AGW的登录账户，而非客户相关账户；

17. 编译报错：error LNK2038 'MT_xxx' doesn't match value 'MD_xxx' in xxxxxx.obj

解决方案属性页：“配置属性” -> “C/C++” -> “代码生成”：运行库 选项选择 多线程DLL（/MT）

18. 网关登陆密码的加密算法是啥，客户登陆的密码也要按这个算法加密吗?

网关密码是sha256加密后在用base64编码转换；客户号登陆的密码也会加密，具体算法可咨询券商

19. api使用什么编码？

UTF-8

20. api使用的CPU为什么这么高？

为了保证报单时延，发送线程采用无等待发送方式，导致发送线程始终占用CPU，如投资者对时延追求不高，可设置环境变量ATP_AGW_API_ENV=no_low_latency

21. 订单查询接口如何区分是下单委托还是撤单委托？

发起撤单委托时需要填写原委托客户合同号（orig_cl_ord_no），而下单委托不需要填写原委托客户合同号（orig_cl_ord_no）。所以可以根据orig_cl_ord_no是否为0来判断是否为撤单委托，但是前提条件是撤单传入的orig_cl_ord_no不能为0