api_demo 功能说明

目录

第一章 Demo 功能详细说明

1.1 功能总介

功能序号	功能说明
1	交易所选择
2	记录控制：是否退出时记录；过滤合约功能；逐笔通道是否分开记录
3	接收方式选择
4	接收通道配置
5	运行时统计信息
6	版本 1.2.X 以上版本只维护 onload 7

1.2 交易所选择

1.2.1 功能说明

当前 API 支持接收上交所和深交所的输出 EFH 行情，可通过配置选择只接收上交所或深交所行情，或者同时接收，或都不接收。

1.2.2 配置说明

```
[EFH_QUOTE_TYPE]
enable_sze=0     #使用深交所通道
enable_sse=1     #使用上交所通道
```

enable_sze=1	使用深交所通道接收行情
enable_sze=0	不启用深交所通道
enable_sse=1	使用上交所通道接收行情
enable_sse=0	不启用上交所通道

1.2.3 代码说明

main.cpp 文件中控制

```
int sse_enable = ini.ReadInt( "EFH_QUOTE_TYPE" , "enable_sse" , 0 );
int sze_enable = ini.ReadInt( "EFH_QUOTE_TYPE" , "enable_sze" , 0 );
if ( sse_enable == 0 && sze_enable == 0 )
 {
```

```
        printf( "sze enable and sse enable is all is 0\n" );
        return 0;
    }
    sse_report* sse = NULL;
    sze_report* sze = NULL;
    if ( sze_enable == 1 )
    {
        sze = run_sze();
    }
    if ( sse_enable == 1 )
    {
        sse = run_sse();
    }
```

1.3 记录控制

1.3.1 功能说明

记录控制中通过配置文件控制，包括收到的数据是否在最后程序退出时记录；接收行情数据时，过滤合约；逐笔通道是否分开记录；

1.3.2 配置说明

```
[REPORT]
#是否退出时才写文件，1：是，打印文本文件，会过滤合约；0：不是，打印二进制文件，不过滤合约
report_when_quit=0
#需要的合约，以","分割
symbol="002835"
#逐笔通道分开写入文件开关,0:合并写入,1:分开写入
tick_detach_enable=0
```

report_when_quit=0	接收数据就会记录数据，并且记录为二进制接收的原始数据，此时不会过滤合约。
report_when_quit=1	接收数据放到节点缓存中，每种类型事先开辟 800W 个节点，循环存储，存储为 csv 格式的人眼可识字段数据格式。
symbol=””	收到行情时需要记录的合约，只有在 report_when_quit=1 时生效，合约分割符为逗号“，”分割，格式如：002835;002836;002837
tick_detach_enable=0	逐笔通道中订单和成交合并记录到文件，分别为日期_sse/sze_order.dat 和日期_sse/sze_exe.dat

tick_detach_enable=1	逐笔通道中订单行情和成交行情合并记录到日期_sse/sze_tick.dat。但是在 report_when_quit=1 时不生效。

1.3.3 代码说明

以 sze_report.cpp 为例：

```
void sze_report::on_report_efh_sze_lev2_tick(int msg_type, sze_hpf_order* p_order, sze_hpf_exe* p_exe)
{
    if (msg_type == SZE_LEV2_ORDER_MSG_TYPE)
    {
        if (!p_order)
        {
            string msg = format_str("write lev2 tick error: p_order == NULL\n");
            efh_sze_lev2_error(msg.c_str(), msg.length());
            return;
        }
        if(m_i_tick_seq!=0)
        {
            m_ll_tick_count_lost += p_order->m_head.m_sequence - m_i_tick_seq - 1;
            if (p_order->m_head.m_sequence <= m_i_tick_seq)
            {
                m_b_tick_rollback_flag = true;
                printf("SZE tick out of seq, last seq[%u], cur seq[%u].\n", m_i_tick_seq, p_order->m_head.m_sequence);
            }
        }
        m_i_tick_seq = p_order->m_head.m_sequence;

        if (!m_b_report_quit)
        {
            if(m_b_tick_detach)
            {
                if (m_fp_ord == NULL)
                {
                    time_t now = time(NULL);
                    tm* ltm = localtime(&now);

                    char    str_full_name[1024];
                    memset(str_full_name, 0, sizeof(str_full_name));
```

```
                    sprintf(str_full_name, "%04d%02d%02d_sze_order.dat"
, ltm->tm_year + 1900, ltm->tm_mon + 1, ltm->tm_mday);

                    m_fp_ord = fopen(str_full_name, "wb+");
                    if (m_fp_ord == NULL)
                    {
                        return;
                    }
                }

                int ret = fwrite((char*)p_order, 1, sizeof(*p_order), m
_fp_ord);
                fflush(m_fp_ord);
                if (ret <= 0)
                {
                    printf("write sze_order.dat error!\t");
                }
            }
            else
            {
                if (m_fp_tick == NULL)
                {
                    time_t now = time(NULL);
                    tm* ltm = localtime(&now);

                    char    str_full_name[1024];
                    memset(str_full_name, 0, sizeof(str_full_name));
                    sprintf(str_full_name, "%04d%02d%02d_sze_tick.dat",
 ltm->tm_year + 1900, ltm->tm_mon + 1, ltm->tm_mday);

                    m_fp_tick = fopen(str_full_name, "wb+");
                    if (m_fp_tick == NULL)
                    {
                        return;
                    }
                }

                int ret = fwrite((char*)p_order, 1, sizeof(*p_order), m
_fp_tick);
                fflush(m_fp_tick);
                if (ret <= 0)
                {
                    printf("write sze_tick.dat error!\t");
                }
```

```
            }

            ++m_ll_order_count;
            return;
        }

        if (!is_vaild_symbol((char*)(p_order->m_head.m_symbol)))
        {
            return;
        }

        timespec                ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        int node = m_ll_order_count % QT_SZE_QUOTE_COUNT;

        memcpy(m_order[node].m_symbol, p_order->m_head.m_symbol, 9);
        m_order[node].m_local_time        = ((long long)(ts.tv_sec)) *
1000000000 + ts.tv_nsec;
        m_order[node].m_channel_num       = p_order->m_head.m_channel_n
um;
        m_order[node].m_sequence_num      = p_order->m_head.m_sequence_
num;
        m_order[node].m_quote_update_time = p_order->m_head.m_quote_upd
ate_time;
        m_order[node].m_order_quantity    = p_order->m_order_quantity;
        m_order[node].m_order_price       = p_order->m_order_price;
        m_order[node].m_order_type        = p_order->m_order_type;

        ++m_ll_order_count;
    }
    else if (msg_type == SZE_LEV2_EXE_MSG_TYPE)
    {
        if (!p_exe)
        {
            string msg = format_str("write lev2 tick error: p_exe == NU
LL\n");
            efh_sze_lev2_error(msg.c_str(), msg.length());
            return;
        }
        if(m_i_tick_seq!=0)
        {
            m_ll_tick_count_lost += p_exe->m_head.m_sequence - m_i_tick
_seq - 1;
            if(p_exe->m_head.m_sequence <= m_i_tick_seq)
```

```
            {
                m_b_tick_rollback_flag = true;
                printf("SZE tick out of seq, last seq[%u], cur seq[%u].
\n", m_i_tick_seq, p_exe->m_head.m_sequence);
            }
        }
        m_i_tick_seq = p_exe->m_head.m_sequence;
        if (!m_b_report_quit)
        {
            if(m_b_tick_detach)
            {
                if (m_fp_exe == NULL)
                {
                    time_t now = time(NULL);
                    tm* ltm = localtime(&now);

                    char    str_full_name[1024];
                    memset(str_full_name, 0, sizeof(str_full_name));
                    sprintf(str_full_name, "%04d%02d%02d_sze_exe.dat",
ltm->tm_year + 1900, ltm->tm_mon + 1, ltm->tm_mday);

                    m_fp_exe = fopen(str_full_name, "wb+");
                    if (m_fp_exe == NULL)
                    {
                        return;
                    }
                }

                int ret = fwrite((char*)p_exe, 1, sizeof(*p_exe), m_fp_
exe);
                fflush(m_fp_exe);
                if (ret <= 0)
                {
                    printf("write sze_exe.dat error!\n");
                }
            }
            else
            {
                if (m_fp_tick == NULL)
                {
                    time_t now = time(NULL);
                    tm* ltm = localtime(&now);

                    char    str_full_name[1024];
```

```
                    memset(str_full_name, 0, sizeof(str_full_name));
                    sprintf(str_full_name, "%04d%02d%02d_sze_tick.dat",
 ltm->tm_year + 1900, ltm->tm_mon + 1, ltm->tm_mday);

                    m_fp_tick = fopen(str_full_name, "wb+");
                    if (m_fp_tick == NULL)
                    {
                        return;
                    }
                }

                int ret = fwrite((char*)p_exe, 1, sizeof(*p_exe), m_fp_
tick);
                fflush(m_fp_tick);
                if (ret <= 0)
                {
                    printf("write sze_tick.dat error!\t");
                }
            }
            ++m_ll_exe_count;

            return;
        }

        if (!is_vaild_symbol((char*)(p_exe->m_head.m_symbol)))
        {
            return;
        }

        timespec            ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        int node = m_ll_exe_count % QT_SZE_QUOTE_COUNT;

        memcpy(m_exe[node].m_symbol, p_exe->m_head.m_symbol, 9);
        m_exe[node].m_local_time        = ((long long)(ts.tv_sec)) * 10
00000000 + ts.tv_nsec;
        m_exe[node].m_channel_num       = p_exe->m_head.m_channel_num;
        m_exe[node].m_sequence_num      = p_exe->m_head.m_sequence_num;
        m_exe[node].m_quote_update_time = p_exe->m_head.m_quote_update_
time;
        m_exe[node].m_trade_price       = p_exe->m_trade_price;
        m_exe[node].m_trade_quantity    = p_exe->m_trade_quantity;
        m_exe[node].m_trade_type        = p_exe->m_trade_type;
```

```
        ++m_ll_exe_count;
    }
}
```

1.4 接收方式控制

1.4.1 功能说明

api 提供多种接收方式，使用不同的接收方式来适配不同的接收场景。

1.4.2 配置说明

```
[NIC_TYPE]
nic_type=1                    #网卡模式，0：普通模式，1：efvi 模式，2：x25 模式，3：win 高速模式(x710 网卡)，4：win 高速模式(solarflare 网卡)
```

nic_type=0	使用系统 socket 方法接收网络行情数据
nic_type=1	使用 solarflare 网卡提供的 efvi 接收网络行情数据
nic_type=2	使用 x25 模式接收网络行情数据
nic_type=3	使用 x710 网卡接收网络行情数据，只适配 windows 系统
nic_type=4	使用 solarflare 网卡接收网络行情数据，只适配 windows 系统

1.4.3 代码说明

主要代码在 sze_report.cpp 和 sse_report.cpp 中。以 sze_report.cpp 为例

```
    switch (nic_type)
    {
    case enum_nic_normal:
    case enum_nic_solarflare_efvi:
    case enum_nic_exablaze_exanic:
    case enum_nic_x710_win_speed:
    case enum_nic_solarflare_win_speed:
    {
        if (!m_p_quote->start_sze( enum_efh_nic_type(nic_type) ))
        {
            string msg = format_str( "start parse error\n" );
            efh_sze_lev2_error( msg.c_str( ) , msg.length( ) );
        }
    }
    break;
```

1.5 接收通道配置

1.5.1 功能说明

主要配置每个网络类型通道，包括通道是否开启，通道绑定的 cpu，session 四要素，接收用的网口名，使用缓存接收，强制使用普通 socket 接收等。

1.5.2 配置说明

```
[EFH_SZE_LEV2_SNAP]
enable=1                        # 是否使用此通道，1:使用，0 为不使用
cpu_id=1                        # 用于接收的 cpu 序号
multicast_ip="233.57.1.100"     # 组播 IP 地址（通道 IP 地址）
multicast_port=37100            # 组播端口号（通道端口号）
data_ip="4.4.4.63"              # 普通模式下为本机 IP 地址,win 高速模式下为发送方 IP 地址,(EFVI 模式与 x25 模式不使用)
data_port=0                     # 普通模式下为本机端口号,win 高速模式下为发送方端口号，(EFVI 模式与 x25 模式不使用)
eth_name="eth7"                 # 网卡名称,对于 Windows 平台， 形如 0000:af:00.0
cache_size=256                  # cache 大小，单位为 M,设置为小于等于 0 的值表示不使用缓存
force_normal_socket=0           # 强制使用普通的 socket 接收(低性能模式)，1:使用低性能接收，0:禁用
```

enable=1	使用此通道接收行情
enable=0	不使用此通道接收行情，后面的配置都将不使用
cpu_id=1	用于接收绑定的 cpu 核序号。当小于 0 时不绑定
multicast_ip	接收行情数据用的组播地址
multicast_port	接收行情数据用的组播端口
data_ip	普通模式下为本机 IP 地址,windows 系统高速模式下为发送方 IP 地址，EFVI 模式和 X25 模式下可不配置
data_port	普通模式下为本机端口号，windows 系统高速模式下为发送方端口号，EFVI 模式和 X25 模式可不配置
eth_name	接收数据时用的网卡名称
cache_size	设置的缓存大小，当设置为 0 是表示不使用缓存
force_normal_socket	强制使用普通 socket 接收

1.5.3 代码说明

主要是将配置参数传输给 API。

1.6 运行时统计信息

1.6.1 功能说明

在运行时通过在界面输入 show 命令，将各个类型和通道的接收数据量，丢包数据量，是否有回滚标志打印的界面上。

接收数据量的统计方法是当收到数据后就会累加 1

丢包数据量的统计方法是根据盛立行情包头的 sequence，记录上次的序号和本次数据包的序号对比，若是相差大于 1 则表示丢包，并将丢包数据量累加差值。

回滚标志的计算方法是根据盛立行情包头的 sequence，当本次收到序号小于等于上次收到的数据包序号，则表示出现回滚。

1.6.2 配置说明

不需要在配置文件中

1.6.3 代码说明

在 sze_report.cpp 和 sse_report.cpp 文件的 show 方法中。sze_report.cpp 为例

```
    void sze_report::show()
{
    printf( "--------------------< SZE Info >--------------------\n" );
    printf("SZE_snap count: %lld, lost_count:%lld, rollback_flag:%d.\n"
, m_ll_snap_count, m_ll_snap_count_lost, (int)m_b_snap_rollback_flag);
    printf("SZE_idx count: %lld, lost_count:%lld, rollback_flag:%d.\n",
 m_ll_idx_count, m_ll_idx_count_lost, (int)m_b_idx_rollback_flag);
    printf("SZE_after count: %lld, lost_count:%lld, rollback_flag:%d.\n
",m_ll_after_count, m_ll_after_count_lost, (int)m_b_after_rollback_flag
);
    printf("SZE_tree count: %lld, lost_count:%lld, rollback_flag:%d.\n"
, m_ll_tree_count, m_ll_tree_count_lost, (int)m_b_tree_rollback_flag);
    printf("SZE_order count: %lld\n", m_ll_order_count);
    printf("SZE_exe count: %lld\n", m_ll_exe_count);
    printf("SZE_tick count: %lld, lost_count:%lld, rollback_flag:%d.\n"
, m_ll_order_count + m_ll_exe_count, m_ll_tick_count_lost, (int)m_b_tic
k_rollback_flag);
```

```
    printf("SZE_ibr_tree count: %lld, lost_count:%lld, rollback_flag:%d
.\n",m_ll_ibr_tree_count, m_ll_ibr_tree_count_lost, (int)m_b_ibr_tree_r
ollback_flag);
    printf("SZE_turnover count: %lld, lost_count:%lld, rollback_flag:%d
.\n",m_ll_turnover_count, m_ll_turnover_count_lost, (int)m_b_turnover_r
ollback_flag);
    printf("SZE_bond_snap count: %lld, lost_count:%lld, rollback_flag:%
d.\n",m_ll_bond_snap_count, m_ll_bond_snap_count_lost, (int)m_b_bond_sn
ap_rollback_flag);
    printf("SZE_bond_order count: %lld\n",m_ll_bond_order_count);
    printf("SZE_bond_exe count: %lld\n",m_ll_bond_exe_count);
    printf("SZE_bond_tick count: %lld, lost_count:%lld, rollback_flag:%
d.\n",m_ll_bond_order_count + m_ll_bond_exe_count, m_ll_bond_tick_count
_lost, (int)m_b_bond_tick_rollback_flag);
    fflush(stdout);
}
```

1.7 版本 1.2.X 以上版本只维护 onload 7

1.7.1 功能说明

在 1.2.X 以上的版本使用 onload7 版本

第二章 API 使用注意事项

2.1 在调用 API 的同进程中，不要使用 system(“ifconfig”)或 system(“netstat –s”)等系统调用，会影响 EFVI 的接收

2.2 普通 socket 不要使用不同网卡接收同组播 IP 会接收，否则会接收到两份一样的数据。