

NetSDK_C#编程指导手册 (智能楼宇分册)



前言

概述

欢迎使用 NetSDK（以下简称 SDK）C#编程指导手册（智能楼宇分册）。

SDK 是软件开发者在开发网络硬盘录像机、网络视频服务器、网络摄像机、网络球机和智能设备等产品监控联网应用时的开发套件。

本文档描述了智能楼宇产品的通用业务涉及的 SDK 接口以及调用流程，更多功能接口、结构体等请参考《网络 SDK 开发手册》。

本文档提供的示例代码仅为演示接口调用方法，不保证能直接拷贝编译。

读者对象

使用 SDK 的软件开发工程师、项目经理和产品经理。

符号约定

在本文档中可能出现下列标识，代表的含义如下。

标识	说明
 危险	表示有高度潜在危险，如果不能避免，会导致人员伤亡或严重伤害。
 警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 防静电	表示静电敏感的设备。
 当心触电	表示高压危险。
 激光辐射	表示强激光辐射。
 风扇警告	表示危险运动部件，请远离运动风扇叶片。
 当心机械伤人	表示设备部件机械伤人。
 窍门	表示能帮助您解决某个问题或节省您的时间。
 说明	表示是正文的附加信息，是对正文的强调和补充。

修订记录

版本号	修订内容	发布日期
V1.0.1	更新依赖库信息。	2021.04

版本号	修订内容	发布日期
V1.0.0	首次发布。	2020.10

名词解释

以下对本文档中使用的专业名词分别说明，帮助您更好的理解各个业务功能。

名词	说明
情景模式	报警主机现有情景模式两种，分别为“外出模式”和“在家模式”，各个模式会有相关的配置，用户选择哪种情景模式，即可使相应配置生效。
在家模式/外出模式	当情景模式切换到“在家模式”或“外出模式”时，会使用该情景模式下预设的防区布防，其余防区变为旁路状态。
隔离	一种对入侵报警探测回路的设置，处于此状态的入侵报警探测回路不能通告报警。此状态一直保持到人为复位。
模拟量报警通道(模拟量防区)	设备有多个报警输入通道，接收外部探测的信号。通道类型为模拟量时，则称为模拟量报警通道，可连接模拟量探测器，用于采集模拟量数据。
胁迫卡	门禁卡的一种，用户被胁迫时使用胁迫卡开门时，门禁系统识别胁迫卡，并上报报警信号。

目录

前言	I
名词解释.....	III
第 1 章 产品概述.....	1
1.1 概述	1
1.2 适用性.....	1
1.2.1 适用系统	1
1.2.2 支持设备	2
1.3 应用场景.....	2
第 2 章 主要功能.....	5
2.1 通用	5
2.1.1 SDK 初始化	5
2.1.2 设备初始化.....	7
2.1.3 设备登录	9
2.1.4 实时监控	11
2.1.5 语音对讲	16
2.1.6 报警上报	20
2.2 门禁控制器/指纹一体机（一代）	23
2.2.1 门禁控制	23
2.2.2 报警事件	27
2.2.3 设备信息查看.....	35
2.2.4 网络设置	42
2.2.5 设备时间获取和设置.....	47
2.2.6 维护配置	54
2.2.7 人员管理	64
2.2.8 门配置	82
2.2.9 门时间配置.....	85
2.2.10 门高级配置.....	98
2.2.11 记录查询.....	118
2.3 门禁控制器/人脸一体机（二代）	126
2.3.1 门禁控制	127
2.3.2 报警事件	127
2.3.3 设备信息查看.....	127
2.3.4 网络设置	129
2.3.5 设备时间设置.....	129
2.3.6 维护配置	129
2.3.7 人员管理	129
2.3.8 门配置	149
2.3.9 门时间管理.....	149
2.3.10 门高级配置.....	153
2.3.11 记录查询.....	153
第 3 章 接口函数.....	157
3.1 通用接口.....	157
3.1.1 SDK 初始化	157

3.1.2	设备初始化.....	158
3.1.3	设备登录.....	159
3.1.4	实时监视.....	160
3.1.5	设备控制.....	163
3.1.6	报警侦听.....	163
3.1.7	获取设备状态.....	164
3.1.8	语音对讲.....	165
3.2	门禁控制器/指纹一体机（一代）.....	167
3.2.1	门禁控制.....	167
3.2.2	报警事件.....	167
3.2.3	设备信息查看.....	167
3.2.4	网络设置.....	169
3.2.5	时间设置.....	171
3.2.6	维护配置.....	172
3.2.7	人员管理.....	176
3.2.8	门配置.....	176
3.2.9	门时间配置.....	177
3.2.10	门高级配置.....	177
3.2.11	记录查询.....	180
3.3	门禁控制器/人脸一体机（二代）.....	182
3.3.1	门禁控制.....	182
3.3.2	报警事件.....	182
3.3.3	设备信息查看.....	182
3.3.4	网络设置.....	183
3.3.5	时间设置.....	183
3.3.6	维护配置.....	183
3.3.7	人员管理.....	183
3.3.8	门配置.....	191
3.3.9	门时间配置.....	192
3.3.10	门高级配置.....	194
3.3.11	记录查询.....	194
第 4 章	回调函数.....	195
4.1	搜索设备回调函数 fSearchDevicesCB.....	195
4.2	异步搜索设备回调函数 fSearchDevicesCBEx.....	195
4.3	断线回调函数 fDisConnectCallBack.....	195
4.4	断线重连回调函数 fHaveReConnectCallBack.....	196
4.5	实时监视数据回调函数 fRealDataCallBackEx2.....	196
4.6	音频数据回调函数 pfAudioDataCallBack.....	197
4.7	报警回调函数 fMessCallBackEx.....	198
4.8	升级进度回调函数 fUpgradeCallBackEx.....	200
附录 1	法律声明.....	202
附录 2	网络安全建议.....	203

第 1 章 产品概述

1.1 概述

本文档主要介绍关于 C#的 NetSDK 库的封装工程 NetSDKCS 的参考信息，包括主要功能、接口函数和回调函数。主要功能包括：NetSDK 初始化、设备登录、实时监控、语音对讲、报警上报以及智能楼宇的相关功能等。

- C#的 NetSDK 库所包含的文件，请参见表 1-1 和表 1-2。

表1-1 Windows 下 NetSDK 库包括的文件

库类型	Window 系统下库文件名称	Linux 系统下库文件名称	库文件说明
功能库	dhnet sdk.dll	libdhnet sdk.so	库文件
	avnetsdk.dll	libavnetsdk.so	库文件
配置库	dhconfig sdk.dll	libdhconfig sdk.so	库文件
播放（编码解码） 辅助库	dhplay.dll	libdhplay.so	播放库
	fisheye.dll	未包含	鱼眼矫正库
dhnet sdk 辅助库	lvsDrawer.dll	未包含	图像显示库
	StreamConvertor.dll	libStreamConvertor.so	转码库

- C#封装工程所包含的文件，请参见表 1-2。

表1-2 NetSDKCS 工程包括的文件

文件名称	文件说明
NetSDK.cs	封装 C++接口，提供调用的 C#接口
NetSDKStruct.cs	存放所用到的结构体枚举等
OriginalSDK.cs	将 NetSDK 库中的 C 接口引入到 C#工程中

说明

- NetSDK 的功能库和配置库是必备库。
- 功能库是设备网络 NetSDK 的主体，主要用于网络客户端与各类产品之间的通讯交互，负责远程控制、查询、配置及码流数据的获取和处理等。
- NetSDK 库是 NetSDKCS 工程的基础，工程中 OriginalSDK.cs 文件内定义了 NetSDK 库的引用路径，最终使用时请将 NetSDK 库放到相应路径下。用户可自定义引用路径。
- 客户可以在自己的工程内直接引用本封装工程，也可以将封装工程内的文件放到自己工程内使用，还可以参考本封装工程自己封装。

1.2 适用性

1.2.1 适用系统

- 推荐内存：不低于 512M。
- 操作系统：
 - ◇ Windows: Windows 10/Windows 8.1/Windows 7 以及 Windows Server 2008/2003。
 - ◇ Linux: Red Hat/SUSE 等通用 Linux 系 AAA 统。

1.2.2 支持设备



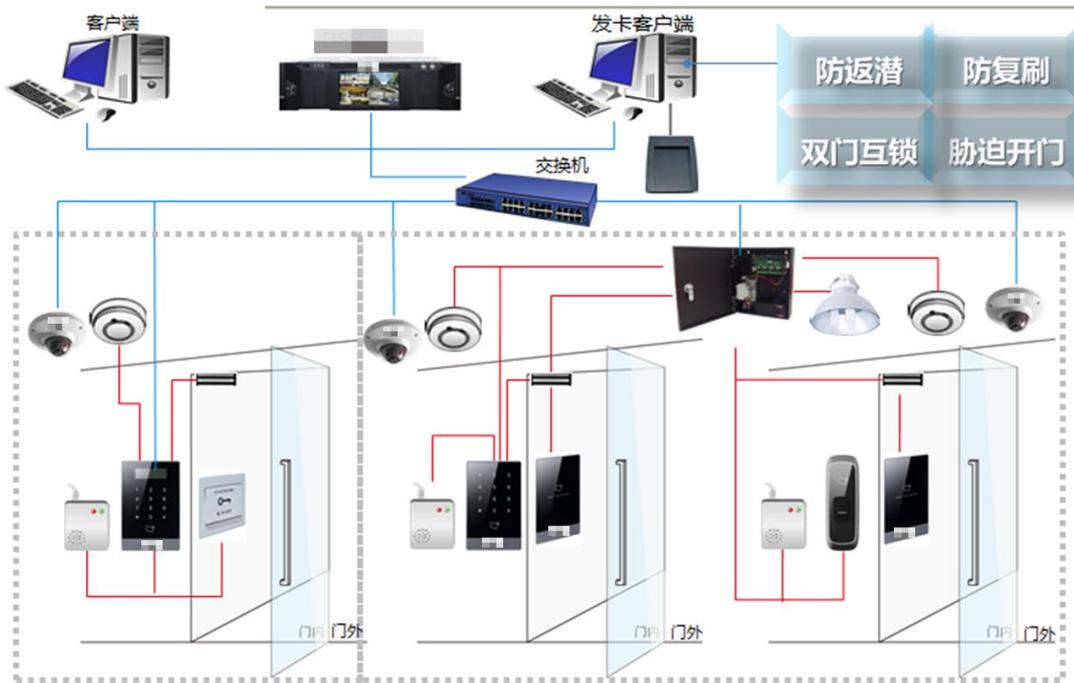
本文仅列出部分设备型号。

- 门禁设备（一代设备）
 - ◇ ASC1201B-D、ASC1201C-D
 - ◇ ASC1202B-S、ASC1202B-D、ASC1202C-S、ASC1202C-D
 - ◇ ASC1204B-S、ASC1204C-S、ASC1204C-D
 - ◇ ASC1208C-S
 - ◇ ASI1201A、ASI1201A-D、ASI1201E、ASI1201E-D
 - ◇ ASI1212A(V2)、ASI1212A-D(V2)、ASI1A212D、ASI1212D-D
- 门禁设备（二代设备）
 - ◇ ASI1202M、ASI1202M-D
 - ◇ ASI7213X、ASI7213Y-D、ASI7213Y-V3
 - ◇ ASI7214X、ASI7214Y、ASI7214Y-D、ASI7214Y-V3
 - ◇ ASI7223X-A、ASI7223Y-A、ASI7223Y-A-V3
 - ◇ ASI8213Y-V3
 - ◇ ASI8214Y、ASI8214Y(V2)、ASI8214Y-V3
 - ◇ ASI8223Y、ASI8223Y(V2)、ASI8223Y-A(V2)、ASI8223Y-A-V3
- 可视对讲设备
 - ◇ VTA8111A
 - ◇ VTO1210B-X、VTO1210C-X
 - ◇ VTO1220B
 - ◇ VTO2000A、VTO2111D
 - ◇ VTO6210B、VTO6100C
 - ◇ VTO9231D、VTO9241D
 - ◇ VTH1510CH、VTH1510A、VTH1550CH
 - ◇ VTH5221D、VTH5241D
 - ◇ VTS1500A、VTS5420B、VTS8240B、VTS8420B
 - ◇ VTT201、VTT2610C

1.3 应用场景

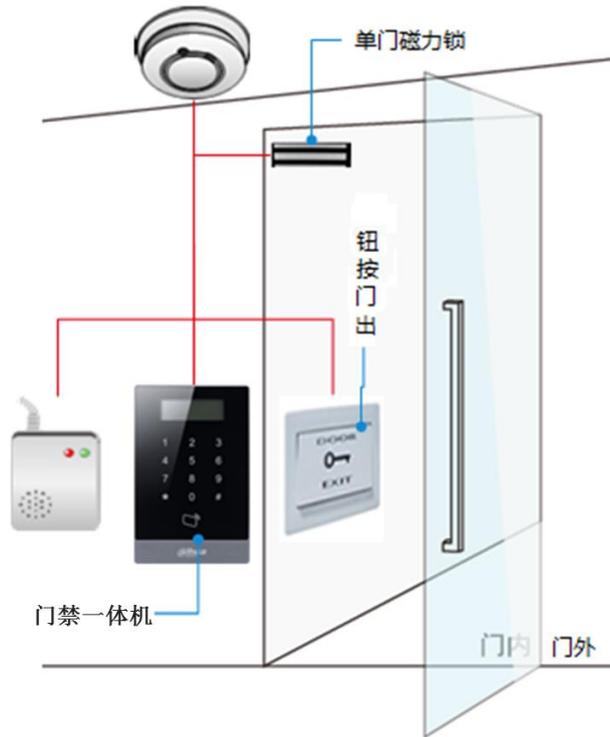
- 典型场景。

图1-1 典型场景



- 微型门禁适用于小型办公。

图1-2 微型门禁



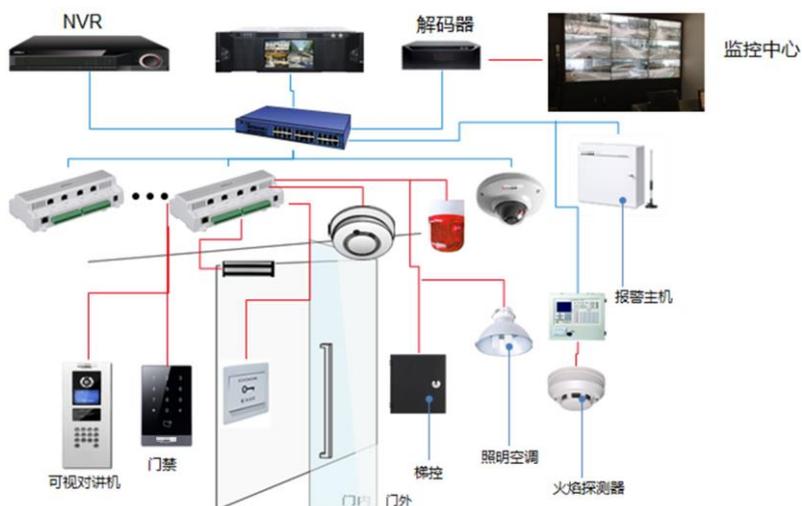
- 网络型门禁适用于中小型以上智能楼宇项目及金库和监所项目。

图1-3 网络型门禁



- 增强型门禁。

图1-4 增强型门禁



第 2 章 主要功能

2.1 通用

2.1.1 SDK 初始化

2.1.1.1 简介

初始化是 NetSDK 进行各种业务的第一步。初始化本身不包含监控业务，但会设置一些影响全局业务的参数。

- NetSDK 的初始化将会占用一定的内存。
- 同一个进程内，只有第一次初始化有效。
- 使用完毕后需要调用 NetSDK 清理接口以释放资源。

2.1.1.2 接口总览

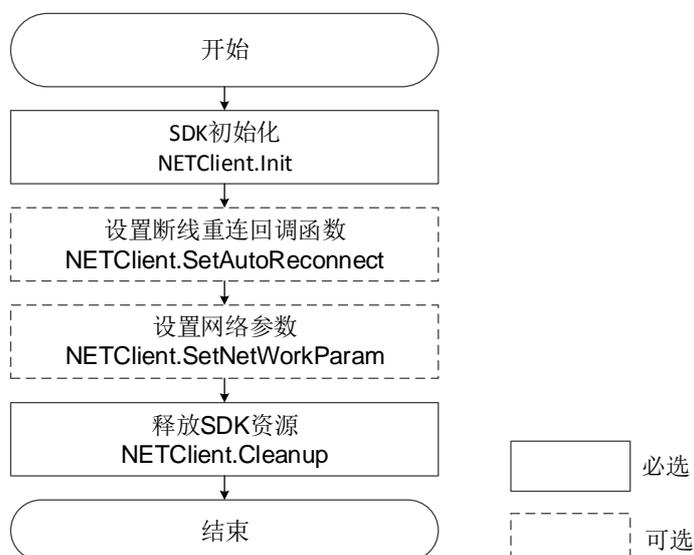
表2-1 SDK 初始化接口说明

接口	说明
NETClient.Init	NetSDK 初始化接口
NETClient.Cleanup	NetSDK 清理接口
NETClient.SetAutoReconnect	设置断线重连回调接口
NETClient.SetNetworkParam	设置登录网络环境接口

2.1.1.3 流程说明

NetSDK 初始化业务流程如图 2-1 所示。

图2-1 SDK 初始化业务流程



流程说明

- 步骤1 调用 `NETClient.Init` 完成 NetSDK 初始化流程。
- 步骤2 （可选）调用 `NETClient.SetAutoReconnect` 设置断线重连回调函数，设置后 NetSDK 内部断线自动重连。
- 步骤3 （可选）调用 `NETClient.SetNetworkParam` 设置网络登录参数，包括登录设备超时时间和尝试次数等。
- 步骤4 NetSDK 所有功能使用完后，调用 `NETClient.Cleanup` 释放 NetSDK 资源。

注意事项

- NetSDK 的 `NETClient.Init` 和 `NETClient.Cleanup` 接口需成对调用，支持单线程多次成对调用，但建议全局调用一次。
- 初始化：`NETClient.Init` 接口多次调用时，仅在内部用做计数，不会重复申请资源。
- 清理：`NETClient.Cleanup` 接口内会清理所有已开启的业务，如登录、实时监视和报警订阅等。
- 断线重连：NetSDK 可以设置断线重连功能，当遇到一些特殊情况（例如断网、断电等）设备断线时，在 NetSDK 内部会定时持续不断地进行登录操作，直至成功登录设备。断线重连后可以恢复实时监视、录像回放业务、智能事件订阅和报警订阅，其他业务无法恢复。

2.1.1.4 示例代码

```
// 声明静态回调委托（普通委托可能会出现回调之前已经将其释放了的错误）
private static fDisconnectCallBack m_DisConnectCallBack; //断线回调
private static fHaveReConnectCallBack m_ReConnectCallBack; //断线重连回调

//实现委托
m_DisConnectCallBack = new fDisconnectCallBack(DisconnectCallBack);
m_ReConnectCallBack = new fHaveReConnectCallBack(ReConnectCallBack);

// 初始化 NetSDK，在初始化时实现断线回调
bool result = NETClient.Init(m_DisConnectCallBack, IntPtr.Zero, null);
if (!result)
{
    MessageBox.Show(NETClient.GetLastError()); //显示错误信息
    return;
}

//设置断线重连回调
NETClient.SetAutoReconnect(m_ReConnectCallBack, IntPtr.Zero);

//设置网络参数
NET_PARAM param = new NET_PARAM()
{
    nWaittime = 10000, // 等待超时时间(毫秒)
    nConnectTime = 5000, // 连接超时时间(毫秒)
};
NETClient.SetNetworkParam(param);
```

```
// 清理初始化资源
NETClient.Cleanup();
```

2.1.2 设备初始化

2.1.2.1 简介

设备在出厂时处于未初始化的状态，使用设备前需要初始化设备。

- 未初始化的设备不能登录。
- 初始化相当于给默认的 **admin** 帐户设置一个密码。
- 当忘记密码时，也可以重置密码。

2.1.2.2 接口总览

表2-2 设备初始化接口说明

接口	说明
NETClient.StartSearchDevicesEx	搜索局域网内的设备，找到未初始化设备
NETClient.InitDevAccount	设备初始化接口
NETClient.StopSearchDevices	停止搜索设备

2.1.2.3 流程说明

图2-2 设备初始化流程



流程说明

步骤1 调用 NETClient.Init 完成 SDK 初始化流程。

- 步骤2 调用 NETClient.StartSearchDevicesEx 搜索局域网内的设备，获取设备信息（不支持多线程调用）。
- 步骤3 调用 NETClient.InitDevAccount 初始化设备。
- 步骤4 调用 NETClient.StopSearchDevices 停止设备的搜索。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 释放 SDK 资源。

注意事项

此接口的工作方式为组播，因此主机和设备必须在同一个组播组。

2.1.2.4 示例代码

```
Task task = new Task(() =>
{
    NET_IN_INIT_DEVICE_ACCOUNT inParam = new NET_IN_INIT_DEVICE_ACCOUNT();
    inParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_INIT_DEVICE_ACCOUNT));
    if (initDeviceDialog.IsEmail)
    {
        inParam.szMail = initDeviceDialog.RestStr;
    }
    else
    {
        inParam.szCellPhone = initDeviceDialog.RestStr;
    }
    inParam.szMac = selectInfo.stuDevInfo.szMac;
    inParam.szUserName = initDeviceDialog.UserName;
    if (initDeviceDialog.Passwrod.Length > 127)
    {
        string password = initDeviceDialog.Passwrod.Substring(0, 127);
        inParam.szPwd = password;
    }
    else
    {
        inParam.szPwd = initDeviceDialog.Passwrod;
    }
    inParam.byPwdResetWay = selectInfo.stuDevInfo.byPwdResetWay;
    NET_OUT_INIT_DEVICE_ACCOUNT outParam = new
NET_OUT_INIT_DEVICE_ACCOUNT();
    outParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_INIT_DEVICE_ACCOUNT));
    bool res = NETClient.InitDevAccount(inParam, ref outParam, 5000,
selectInfo.szLocallp);
    if (!res)
```

```

        {
            string errorMsg = NETClient.GetLastError();
            this.BeginInvoke(new Action(() =>
            {
                MessageBox.Show(this, errorMsg);
            }));
        }
        else
        {
            this.BeginInvoke(new Action(() =>
            {
                listView_device.SelectedItems[0].BackColor = Color.White;
                listView_device.SelectedItems[0].SubItems[1].Text = "Initialized(已初始化)";
            }));

            selectInfo.stuDevInfo.byInitStatus = 2;
        }
    });
    task.Start();

```

2.1.3 设备登录

2.1.3.1 简介

设备登录，即用户鉴权，是进行其他业务的前提。

用户登录设备产生唯一的登录 ID，其他功能的 NetSDK 接口需要传入登录 ID 才可执行。登出设备后，登录 ID 失效。

2.1.3.2 接口总览

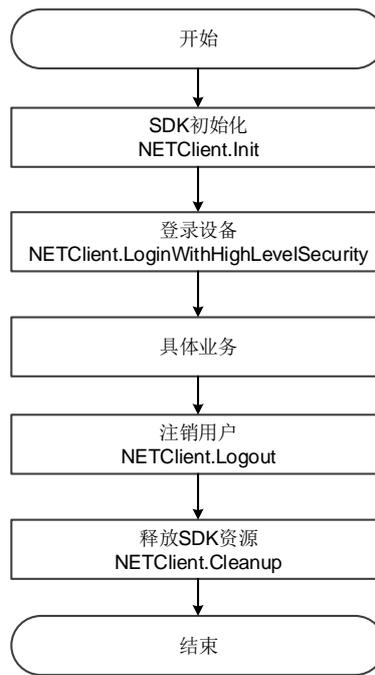
表2-3 设备登录接口说明

接口	说明
NETClient.LoginWithHighLevelSecurity	登录接口
NETClient.Logout	登出接口

2.1.3.3 流程说明

登录业务流程如图 2-3 所示。

图2-3 登录业务流程



流程说明

- 步骤1 调用 NETClient.Init 完成 NetSDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 登录设备。
- 步骤3 登录成功后，用户可以实现需要的业务功能。
- 步骤4 业务使用完后，调用 NETClient.Logout 登出设备。
- 步骤5 NetSDK 功能使用完后，调用 NETClient.Cleanup 释放 NetSDK 资源。

注意事项

- 登录句柄：登录成功时接口返回值非 0（即句柄可能小于 0，也属于登录成功）；同一设备登录多次，每次的登录句柄不一样。如果无特殊业务，建议只登录一次，登录的句柄可以重复用于其他各种业务。
- 句柄重复：登录句柄有可能与存在过的句柄相同，属于正常现象。例如登录设备 A 获得 loginIDA，将 loginIDA 注销，再次进行登录操作，可能又获取到 LoginIDA。但是在句柄的整个生命周期内，不会出现重复的句柄。
- 登出：接口内部会释放登录会话中已打开的业务，但建议用户不要依赖登出接口的清理功能。例如打开监视后，在不需要使用监视时，用户应该调用结束监视的接口。
- 登录与登出配对使用，登录会消耗一定的内存和 socket 信息，在登出后释放资源。
- 登录失败：可以通过 NETClient.GetLastError 接口获取失败信息。
- 设备断线以后，设备的登录 ID 会失效，待设备重连后该登录 ID 会重新生效。

2.1.3.4 示例代码

```
//登录设备
NET_DEVICEINFO_Ex m_DeviceInfo = new NET_DEVICEINFO_Ex();
IntPtr m_LoginID = NETClient.LoginWithHighLevelSecurity(ip, port, name, password,
EM_LOGIN_SPAC_CAP_TYPE.TCP, IntPtr.Zero, ref m_DeviceInfo);
```

```

if (IntPtr.Zero == m_LoginID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// 退出设备
if (IntPtr.Zero != m_LoginID)
{
    bool result = NETClient.Logout(m_LoginID);
    if (!result)
    {
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
}
m_LoginID = IntPtr.Zero;
}

```

2.1.4 实时监视

2.1.4.1 简介

实时监视，即向存储设备或前端设备获取实时码流的功能，是监控系统的重要组成部分。

SDK 登录设备后，可向设备获取主码流和辅码流。

- 支持用户传入窗口句柄，SDK 直接进行码流解析及播放（此功能仅限 Windows 版本）。
- 支持回调实时码流数据给用户，让用户自己处理。
- 支持保存实时录像到指定文件，用户可通过自行保存回调码流实现，也可以通过调用 SDK 接口实现。

2.1.4.2 接口总览

表2-4 实时监视接口说明

接口	说明
NETClient.RealPlay	开始实时监视扩展接口
NETClient.StopRealPlay	停止实时监视扩展接口
NETClient.SaveRealData	开始本地保存实时监视数据
NETClient.StopSaveRealData	停止本地保存实时监视数据
NETClient.SetRealDataCallBack	设置实时监视数据回调函数扩展接口

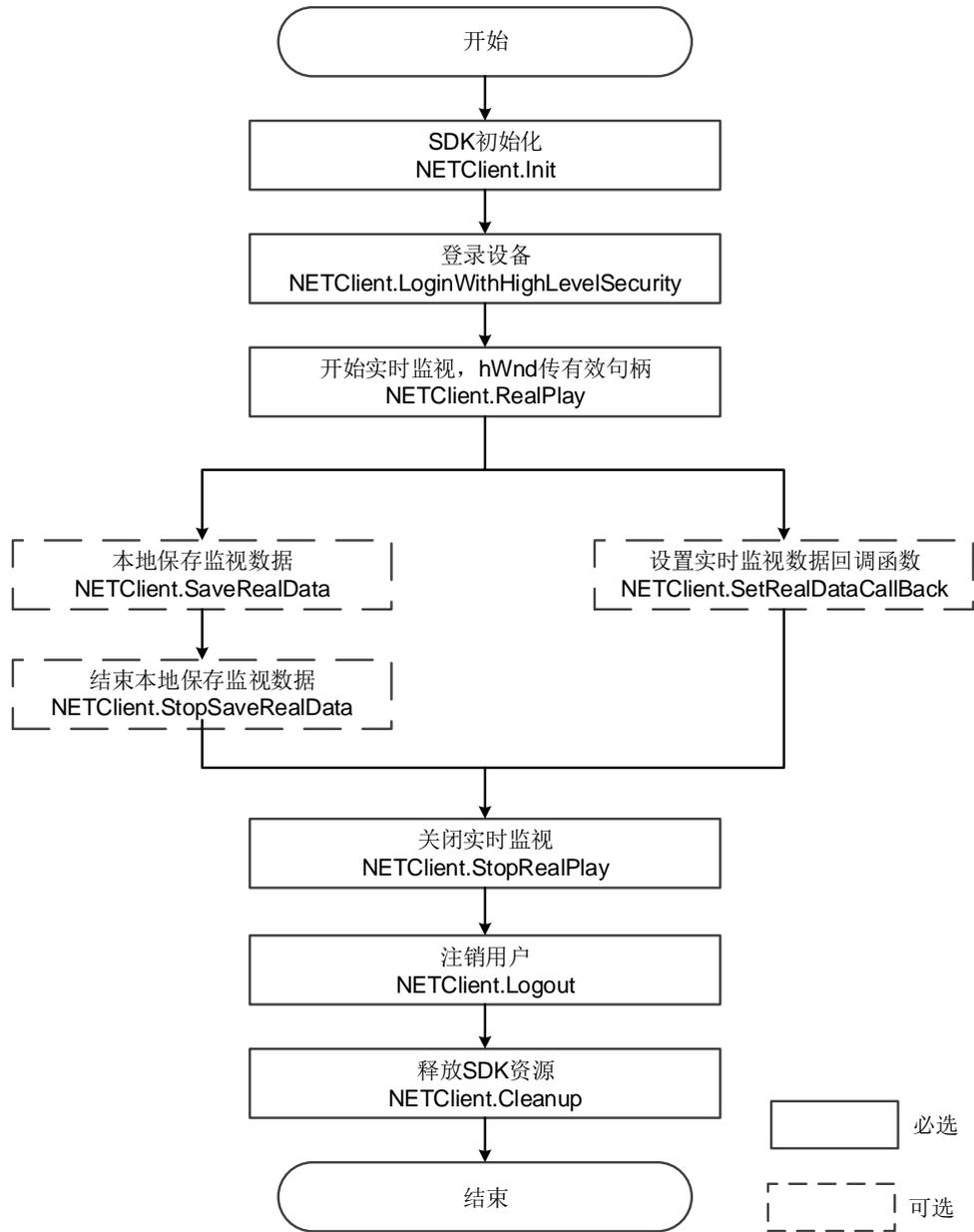
2.1.4.3 流程说明

实时监控的实现方式有两种，分别为 SDK 集成播放库进行播放及用户自己调用播放库播放码流方式进行播放。

2.1.4.3.1 SDK 解码播放

NetSDK 内部调用辅助库里的 PlaySDK 库实现实时播放。NetSDK 解码播放流程如图 2-4 所示。

图2-4 SDK 解码播放流程



流程说明

- 步骤1 调用 NETClient.Init 完成 NetSDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 NETClient.RealPlay 启动实时监控，参数 **hWnd** 为有效窗口句柄。
- 步骤4 （可选）调用 NETClient.SaveRealData 开始保存监视数据。
- 步骤5 （可选）调用 NETClient.StopSaveRealData 结束保存，生成本地视频文件。
- 步骤6 （可选）调用 NETClient.SetRealDataCallBack，用户可将视频数据选择保存或转发。
- 步骤7 实时监控使用完毕后，调用 NETClient.StopRealPlay 停止实时监控。
- 步骤8 业务使用完后，调用 NETClient.Logout 登出设备。
- 步骤9 NetSDK 功能使用完后，调用 NETClient.Cleanup 释放 NetSDK 资源。

注意事项

- NetSDK 解码播放只支持 Windows 系统，非 windows 系统需要用户获取码流后自己调用解码显示。
- 多线程调用：同一个登录会话内的业务，不支持多线程调用；可以多个线程处理不同的登录会话中的业务，但不建议这样调用。
- 超时：接口内申请监视资源需和设备做一些约定，然后才请求监视数据，过程中有一些超时的设定（请参见 NET_PARAM 结构体），其中与监视相关的字段为 **nGetConnInfoTime**。如果实际使用中（如网络状况不良）有超时现象，可将 **nGetConnInfoTime** 的值修改大一些。示例代码如下，在 NETClient.Init 函数后调用，调用一次即可。

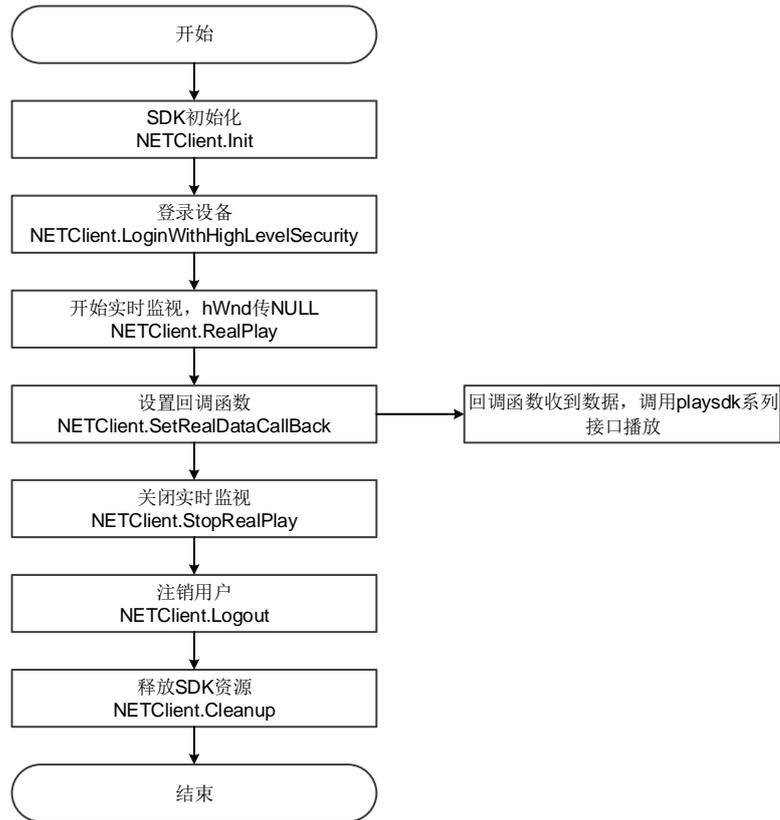
```
NET_PARAM param = new NET_PARAM()
{
    nGetConnInfoTime = 5000, // 获取子连接信息超时时间(毫秒)
};
NETClient.SetNetworkParam(param);
```

- 重复打开失败：部分设备不支持同一个通道多次打开，当重复打开同一通道的监视，可能会出现第一次打开成功，后续打开失败的现象。建议：**A**
 - ◇ 将已打开的通道先关闭。例如已经开启通道一的主码流视频，希望再打开通道一的辅码流视频时，可先关闭通道一的主码流视频，再开启通道一的辅码流视频。
 - ◇ 登录两次设备获取两个登录句柄，分别处理主码流和辅码流业务。
- 接口成功无画面：NetSDK 内部解码需要使用到 **dhplay.dll**，建议查看运行目录下是否缺少 **dhplay.dll** 及其依赖的辅助库，具体请参见表 1-1 和表 1-2。

2.1.4.3.2 调用私有播放库

NetSDK 回调实时监视码流给用户，用户调用 PlaySDK 进行解码播放。用户调用私有播放库解码播放流程如图 2-5 所示。

图2-5 第三方解码播放流程



流程说明

- 步骤1 调用 NETClient.Init 完成 NetSDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 登录设备。
- 步骤3 登录成功后, 调用 NETClient.RealPlay 启动实时监控, 参数 hWnd 为 NULL。
- 步骤4 调用 NETClient.SetRealDataCallBack 设置实时数据回调函数。
- 步骤5 在回调函数中将数据传给 PlaySDK 完成解码。
- 步骤6 实时监控使用完毕后, 调用 NETClient.StopRealPlay 停止实时监控。
- 步骤7 业务使用完后, 调用 NETClient.Logout 登出设备。
- 步骤8 NetSDK 功能使用完后, 调用 NETClient.Cleanup 释放 NetSDK 资源。

注意事项

- 码流格式: 推荐使用 PlaySDK 解码。
- 画面卡顿:
 - ◇ 使用 PlaySDK 解码时, 解码通道缓存大小有默认(PlaySDK 中的 PLAY_OpenStream 接口)。如果码流的分辨率很大, 建议修改参数值, 例如改为 3*1024*1024。
 - ◇ NetSDK 回调函数需用户返回后才能回调下一段, 建议用户在回调中不要做耗时操作, 否则会严重影响性能。

2.1.4.4 示例代码

2.1.4.4.1 NetSDK 解码播放

```
//开启第一路的主码流监视为例,hWnd 为界面窗口句柄
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, 0, hWnd, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// 关闭监视
bool ret = NETClient.StopRealPlay(m_RealPlayID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
m_RealPlayID = IntPtr.Zero;
```

2.1.4.4.2 调用播放库

```
//开启第一路的主码流监视为例
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, 0, null, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

//设置实时监视回调函数
private static fRealDataCallBackEx2 m_RealDataCallBackEx2;
m_RealDataCallBackEx2 = new fRealDataCallBackEx2(RealDataCallBackEx);
NETClient.SetRealDataCallBack(m_RealPlayID, m_RealDataCallBackEx2, IntPtr.Zero,
EM_REALDATA_FLAG.DATA_WITH_FRAME_INFO | EM_REALDATA_FLAG.PCM_AUDIO_DATA |
EM_REALDATA_FLAG.RAW_DATA | EM_REALDATA_FLAG.YUV_DATA);
private void RealDataCallBackEx(IntPtr lRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize,
IntPtr param, IntPtr dwUser)
{
    // 从设备获取的码流数据，需调用 PlaySDK 的接口，详见 NetSDK 监视 demo 源码
    // 比如保存数据、发送数据、转成 YUV 等。
    EM_REALDATA_FLAG type = (EM_REALDATA_FLAG)dwDataType;
    switch (type)
    {
        case EM_REALDATA_FLAG.RAW_DATA:
            //处理操作
            break;
```

```

    }
}

// 关闭监视
bool ret = NETClient.StopRealPlay(m_RealPlayID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
m_RealPlayID = IntPtr.Zero;

```

2.1.5 语音对讲

2.1.5.1 简介

语音对讲主要用于实现本地平台与前端设备所处环境间的语音交互，解决本地平台需要与现场环境语音交流的需求。

本章主要介绍用户如何使用 NetSDK 实现与前端设备的语音对讲。

2.1.5.2 接口总览

表2-5 语音对讲接口说明

接口	说明
NETClient.StartTalk	打开语音对讲扩展接口
NETClient.StopTalk	停止语音对讲扩展接口
NETClient.RecordStart	开始客户端录音扩展接口（只在 Windows 平台下有效）
NETClient.RecordStop	结束客户端录音扩展接口（只在 Windows 平台下有效）
NETClient.TalkSendData	发送语音数据到设备
NETClient.AudioDec	解码音频数据扩展接口（只在 Windows 平台下有效）

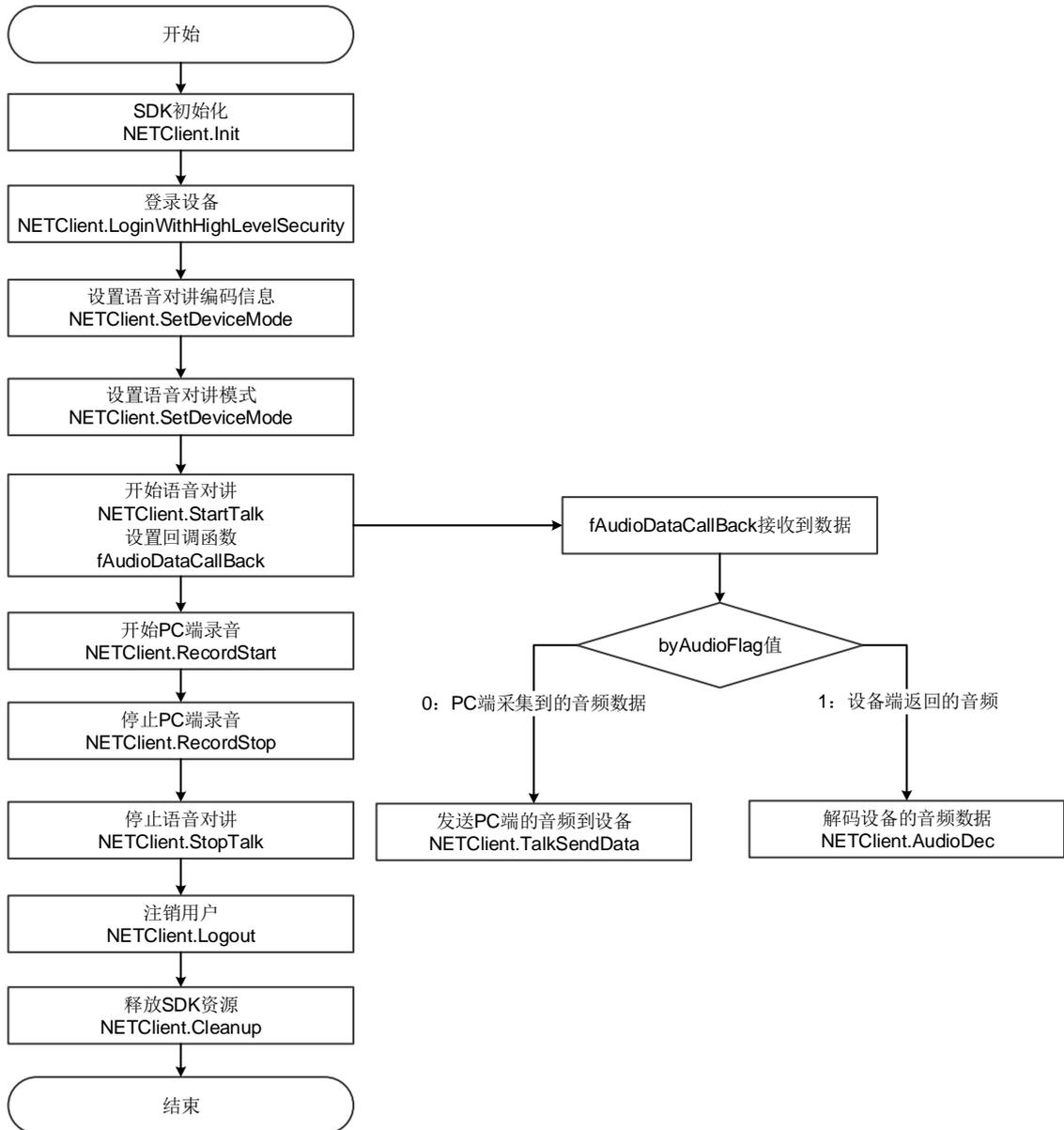
2.1.5.3 流程说明

当 NetSDK 从本地声卡采集到音频数据或 NetSDK 接收到前端发送过来的音频数据时，会调用音频数据回调函数。

用户可在回调函数中调用 NetSDK 接口将采集到的本地音频数据发送到前端设备，也可以调用 NetSDK 接口将接收到的前端设备的音频数据进行解码播放。

该模式只在 Windows 平台下有效。流程如图 2-6 所示。

图2-6 二代语音对讲流程图



流程说明

- 步骤1 调用 NETClient.Init 完成 NetSDK 初始化流程。
- 步骤2 初始化成功后，调用 NETClient.LoginWithHighLevelSecurity 登录设备。
- 步骤3 调用 NETClient.SetDeviceMode 设置语音对讲编码信息，参数 emType 设置为 EM_USEDEV_MODE.TALK_ENCODE_TYPE。
- 步骤4 调用 NETClient.SetDeviceMode 设置语音对讲模式，参数 emType 设置为 EM_USEDEV_MODE.TALK_SPEAK_PARAM。
- 步骤5 调用 NETClient.StartTalk 设置回调函数并开始语音对讲。在回调函数中，调用 NETClient.AudioDec，解码设备发送过来的音频数据；调用 NETClient.TalkSendData，发送 PC 端的音频数据到设备。
- 步骤6 调用 NETClient.RecordStart 开始 PC 端录音，该接口调用后，NETClient.StartTalk 设置的语音对讲回调函数中才会收到本地音频数据。
- 步骤7 对讲功能使用完毕后，调用 NETClient.RecordStop 停止 PC 端录音。
- 步骤8 调用 NETClient.StopTalk 停止语音对讲。

步骤9 调用 NETClient.Logout 登出设备。

步骤10 NetSDK 功能使用完后，调用 NETClient.Cleanup 释放 NetSDK 资源。

注意事项

- 语音编码格式：示例采用了常用的 PCM 格式，NetSDK 支持获取设备支持的语音编码格式。如果默认 PCM 不能满足需求，建议先获取设备支持的语音编码格式，再设置设备支持的编码格式。
- 设备端无声音：需要从麦克风等设备采集音频数据，建议检查是否插上麦克风等音频采集设备，同时检查 NETClient.RecordStart 接口是否返回成功。

2.1.5.4 示例代码

```
if (IntPtr.Zero == m_TalkID)
{
    IntPtr talkEncodePointer = IntPtr.Zero;
    IntPtr talkSpeakPointer = IntPtr.Zero;
    IntPtr talkTransferPointer = IntPtr.Zero;
    IntPtr channelPointer = IntPtr.Zero;

    NET_DEV_TALKDECODE_INFO talkCodeInfo = new
NET_DEV_TALKDECODE_INFO();
    talkCodeInfo.encodeType = EM_TALK_CODING_TYPE.PCM;
    talkCodeInfo.dwSampleRate = SampleRate;
    talkCodeInfo.nAudioBit = AudioBit;
    talkCodeInfo.nPacketPeriod = PacketPeriod;
    talkCodeInfo.reserved = new byte[60];

    talkEncodePointer =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_TALKDECODE_INFO)));
    Marshal.StructureToPtr(talkCodeInfo, talkEncodePointer, true);
    // set talk encode type 设置对讲编码类型
    NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.TALK_ENCODE_TYPE,
talkEncodePointer);

    NET_SPEAK_PARAM speak = new NET_SPEAK_PARAM();
    speak.dwSize = (uint)Marshal.SizeOf(typeof(NET_SPEAK_PARAM));
    speak.nMode = 0;
    speak.bEnableWait = false;
    speak.nSpeakerChannel = 0;
    talkSpeakPointer =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_SPEAK_PARAM)));
    Marshal.StructureToPtr(speak, talkSpeakPointer, true);
    //set talk speak mode 设置对讲模式
    NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.TALK_SPEAK_PARAM,
talkSpeakPointer);
```

```

NET_TALK_TRANSFER_PARAM transfer = new NET_TALK_TRANSFER_PARAM();
transfer.dwSize = (uint)Marshal.SizeOf(typeof(NET_TALK_TRANSFER_PARAM));
if (local_radioButton.Checked)
{
    transfer.bTransfer = false;
}
else
{
    transfer.bTransfer = true;
    channelPointer = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
    Marshal.WriteInt32(channelPointer, channel_comboBox.SelectedIndex);
    NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_TALK_CHANNEL, channelPointer);
}
    talkTransferPointer =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_TALK_TRANSFER_PARAM)));
    Marshal.StructureToPtr(transfer, talkTransferPointer, true);
    //set talk transfer mode 设置对讲是否转发模式
    NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_TRANSFER_MODE, talkTransferPointer);

    m_TalkID = NETClient.StartTalk(m_LoginID, m_AudioDataCallBack, IntPtr.Zero);
    Marshal.FreeHGlobal(talkEncodePointer);
    Marshal.FreeHGlobal(talkSpeakPointer);
    Marshal.FreeHGlobal(talkTransferPointer);
    Marshal.FreeHGlobal(channelPointer);
    if(IntPtr.Zero == m_TalkID)
    {
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
    bool ret = NETClient.RecordStart(m_LoginID);
    if(!ret)
    {
        NETClient.StopTalk(m_TalkID);
        m_TalkID = IntPtr.Zero;
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
    talk_button.Text = "StopTalk(停止对讲)";
}
else
{
    NETClient.RecordStop(m_LoginID);
    NETClient.StopTalk(m_TalkID);
    m_TalkID = IntPtr.Zero;
    talk_button.Text = "StartTalk(开始对讲)";
}

```

```

    }
    private void AudioDataCallBack(IntPtr ITalkHandle, IntPtr pDataBuf, uint dwBufSize, byte
byAudioFlag, IntPtr dwUser)
    {
        if (ITalkHandle == m_TalkID)
        {
            if (SendAudio == byAudioFlag)
            {
                //send talk data 发送语音数据
                NETClient.TalkSendData(ITalkHandle, pDataBuf, dwBufSize);
            }
            else if (ReviceAudio == byAudioFlag)
            {
                //here call netsdk decode audio,or can send data to other user.这里调用
netsdk 解码语音数据, 或也可以把语音数据发送给另外的用户
                try
                {
                    NETClient.AudioDec(pDataBuf, dwBufSize);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }
        }
    }
}

```

2.1.6 报警上报

2.1.6.1 简介

报警上报实现方式为：通过 NetSDK 登录设备并向设备订阅报警功能，设备检测到报警事件立即发送给 NetSDK，通过报警回调函数即可获取相应报警信息。

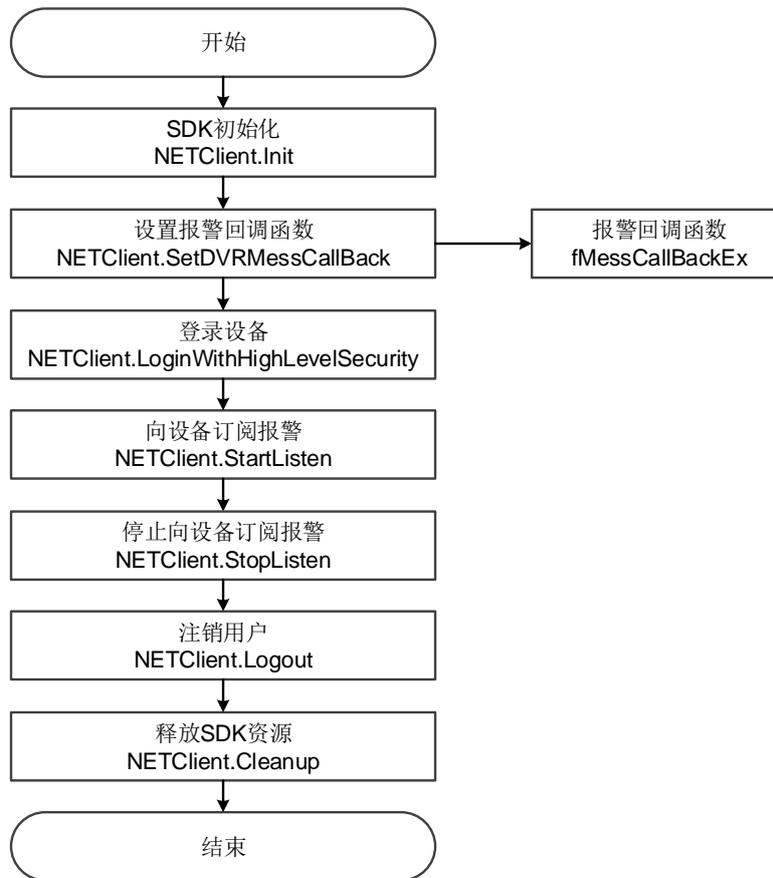
2.1.6.2 接口总览

表2-6 报警上报的接口说明

接口	说明
NETClient.SetDVRMessCallBack	设置报警回调函数接口
NETClient.StartListen	订阅报警扩展接口
NETClient.StopListen	停止订阅报警

2.1.6.3 流程说明

图2-7 报警上报流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 NetSDK 初始化流程。
- 步骤2 调用 NETClient.SetDVRMessCallBack 函数，设置报警回调函数，该接口需在报警订阅之前调用。
- 步骤3 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤4 调用 NETClient.StartListen 函数，向设备订阅报警。订阅成功后，设备上报的报警事件通过 NETClient.SetDVRMessCallBack 函数设置的回调函数通知用户。
- 步骤5 报警上报功能使用完毕后，调用 NETClient.StopListen 函数停止向设备订阅报警。
- 步骤6 调用 NETClient.Logout 函数登出设备。
- 步骤7 NetSDK 功能使用完后，调用 NETClient.Cleanup 函数释放 NetSDK 资源。

注意事项

- 如果之前能上报的报警突然不再上报了，则需排查下设备是否断线。如果断线，设备自动重连后会自动重新订阅报警，不需要用户手动取消订阅再重新订阅。
- 报警回调函数 fMessCallBack 中的报警信息，建议异步处理，不建议在回调里做过多操作，否则会阻塞回调。

2.1.6.4 示例代码

```
// 声明静态回调委托
private static fMessCallBackEx m_AlarmCallBack;
m_AlarmCallBack = new fMessCallBackEx(AlarmCallBackEx);
// 设置报警回调
NETClient.SetDVRMessCallBack(m_AlarmCallBack, IntPtr.Zero);

// 报警回调处理
private bool AlarmCallBackEx(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr
pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser)
{
    EM_ALARM_TYPE type = (EM_ALARM_TYPE)ICommand;
    switch (type)
    {
    case EM_ALARM_TYPE.ALARM_ALARM_EX:
        data = new byte[dwBufLen];
        Marshal.Copy(pBuf, data, 0, (int)dwBufLen);
        for (int i = 0; i < dwBufLen; i++)
        {
            if (data[i] == ALARM_START) // alarm start 报警开始
                { //自定义处理
                }
            else //alarm stop 报警停止
                {
                }
        }
        break;
    case EM_ALARM_TYPE.ALARM_RECORD_SCHEDULE_CHANGE:
        {
            NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO info =
            (NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO)Marshal.PtrToStructure(pBuf,
            typeof(NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO));
            //自定义处理
        }
        break;
        default:
            Console.WriteLine(ICommand.ToString("X"));
            break;
        }
    return true;
}

// 订阅报警
bool ret = NETClient.StartListen(m_LoginID);
if (!ret)
{
```

```

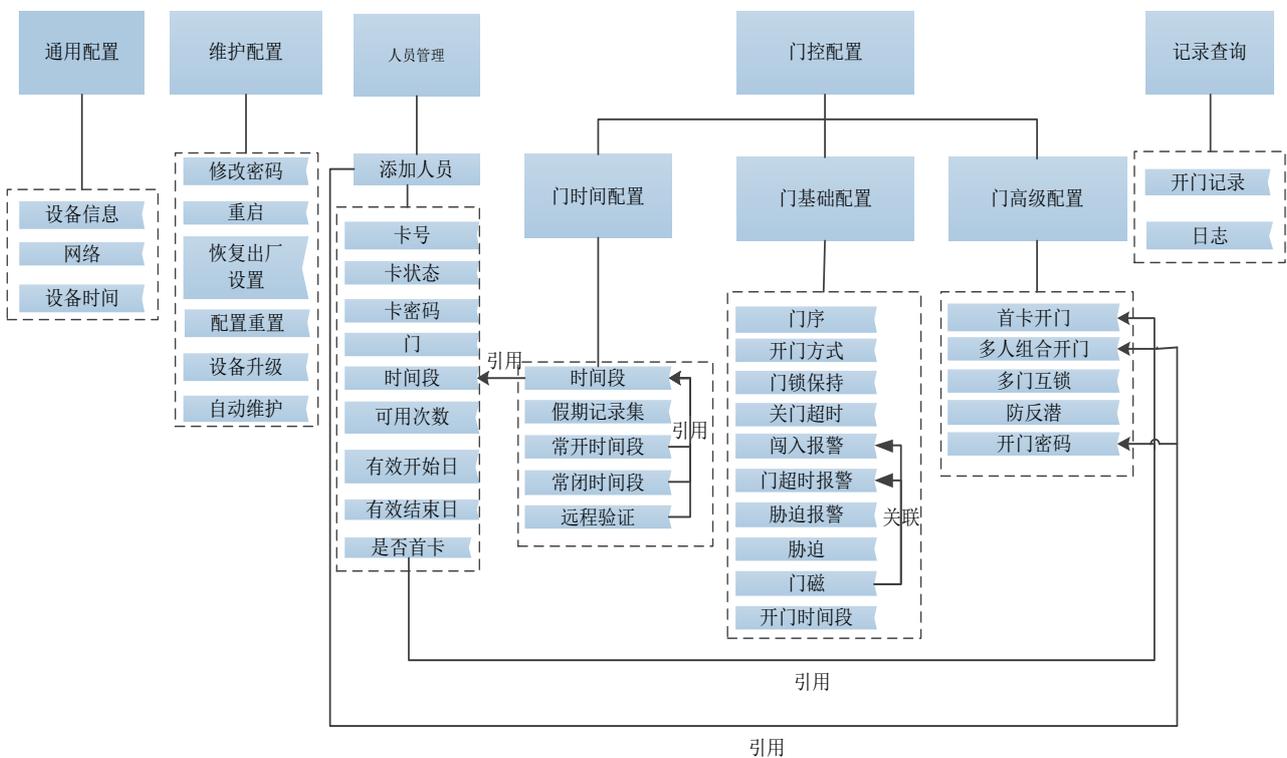
MessageBox.Show(this, NETClient.GetLastError());
return;
}
// 停止报警订阅
bool ret = NETClient.StopListen(m_LoginID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
}

```

2.2 门禁控制器/指纹一体机（一代）

门禁控制器/指纹一体机（一代）功能调用关系，如图 2-8 所示。

图2-8 功能调用关系



功能调用关系中引用和关联的含义如下：

- 引用：箭头终点指向的功能引用箭头起点指向的功能。
- 关联：箭头起始的功能是否能够正常使用，与箭头终点指向的功能配置相关。

2.2.1 门禁控制

2.2.1.1 简介

控制门禁的打开和关闭，获取门磁状态。不需要人员信息，直接远程控制门进行打开和关闭。

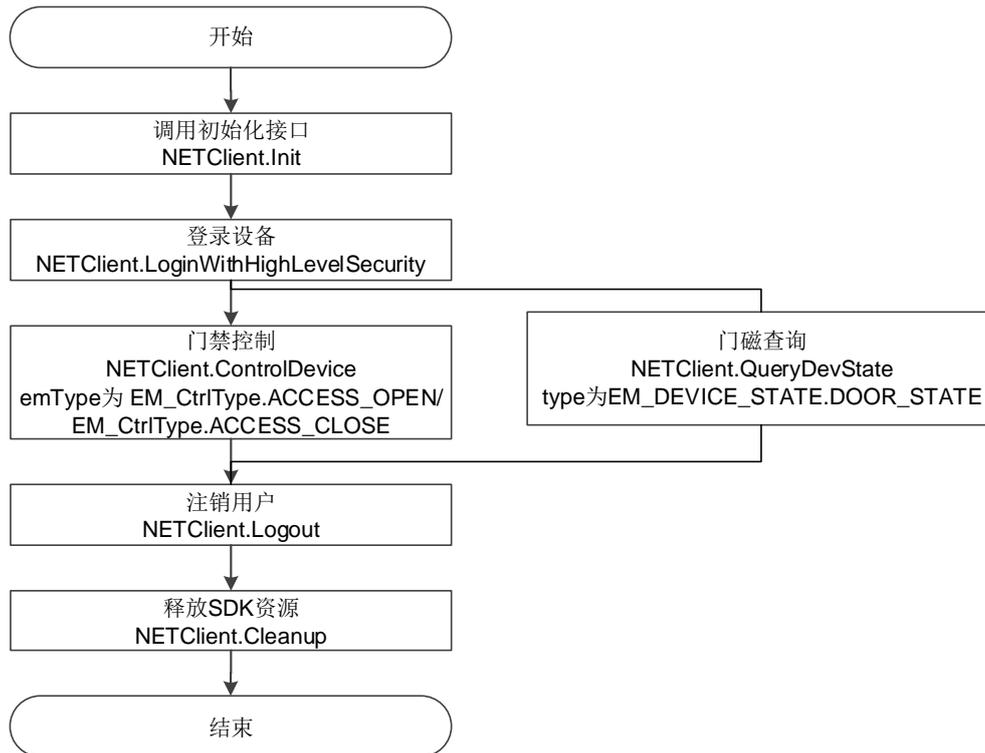
2.2.1.2 接口总览

表2-7 门禁控制接口说明

接口	说明
NETClient.ControlDevice	设备控制接口
NETClient.QueryDevState	状态查询接口

2.2.1.3 流程说明

图2-9 门禁控制业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.ControlDevice 函数来控制门禁。
 - 打开门禁：emType 值为 EM_CtrlType.ACCESS_OPEN。
 - 关闭门禁：emType 值为 EM_CtrlType.ACCESS_CLOSE。
- 步骤4 业务实现，调用 NETClient.QueryDevState 来实现门磁查询。
Type: EM_DEVICE_STATE_DOOR_STATE
pBuf: NET_DOOR_STATUS_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.1.4 示例代码

```
#region Open door 门禁控制-开门
```

```

IntPtr m_LoginHandle = IntPtr.Zero;
GetConfig();
if (cfg.emState != EM_CFG_ACCESS_STATE.NORMAL)
{
    cfg.emState = EM_CFG_ACCESS_STATE.NORMAL;
    SetConfig(cfg);
}
NET_CTRL_ACCESS_OPEN openInfo = new NET_CTRL_ACCESS_OPEN();
openInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_ACCESS_OPEN));
openInfo.nChannelID = 0;
openInfo.szTargetID = IntPtr.Zero;
openInfo.emOpenDoorType = EM_OPEN_DOOR_TYPE.REMOTE;
IntPtr inPtr = IntPtr.Zero;
try
{
    inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_ACCESS_OPEN)));
    Marshal.StructureToPtr(openInfo, inPtr, true);
    bool ret = NETClient.ControlDevice(m_LoginID, EM_CtrlType.ACCESS_OPEN, inPtr,
10000);

    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
}
finally
{
    Marshal.FreeHGlobal(inPtr);
}
MessageBox.Show("Open Door success(开门成功)");
#endregion

#region Close door 门禁控制-关门
IntPtr m_LoginHandle = IntPtr.Zero;
GetConfig();
if (cfg.emState != EM_CFG_ACCESS_STATE.NORMAL)
{
    cfg.emState = EM_CFG_ACCESS_STATE.NORMAL;
    SetConfig(cfg);
}

```

```

NET_CTRL_ACCESS_CLOSE closeInfo = new NET_CTRL_ACCESS_CLOSE();
closeInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_ACCESS_CLOSE));
closeInfo.nChannelID = 0;
IntPtr inPtr = IntPtr.Zero;
try
{
    inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_ACCESS_CLOSE)));
    Marshal.StructureToPtr(closeInfo, inPtr, true);
    bool ret = NETClient.ControlDevice(m_LoginID, EM_CtrlType.ACCESS_CLOSE,
inPtr, 10000);

    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
}
finally
{
    Marshal.FreeHGlobal(inPtr);
}
MessageBox.Show("Close door success(关门成功)");
#endregion
#region Query door state 查询门磁状态
NET_DOOR_STATUS_INFO info = new NET_DOOR_STATUS_INFO();
info.dwSize = (uint)Marshal.SizeOf(typeof(NET_DOOR_STATUS_INFO));
info.nChannel = 0;
object objInfo = info;
bool ret = NETClient.QueryDevState(m_LoginID, EM_DEVICE_STATE.DOOR_STATE, ref
objInfo, typeof(NET_DOOR_STATUS_INFO), 10000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
info = (NET_DOOR_STATUS_INFO)objInfo;

switch (info.emStateType)
{
    case EM_NET_DOOR_STATUS_TYPE.BREAK:
        MessageBox.Show("Door abnormal unlock(门异常打开)");

```

```

        break;
    case EM_NET_DOOR_STATUS_TYPE.CLOSE:
        MessageBox.Show("Door closed(门关闭)");
        break;
    case EM_NET_DOOR_STATUS_TYPE.OPEN:
        MessageBox.Show("Door opened(门打开)");
        break;
    case EM_NET_DOOR_STATUS_TYPE.UNKNOWN:
        MessageBox.Show("Unknown(未知)");
        break;
    default:
        break;
}
#endregion

```

2.2.2 报警事件

2.2.2.1 简介

事件获取，即用户调用 SDK 接口，SDK 主动连接设备，并向设备订阅事件，包括开门事件、报警事件，设备发生事件立即发送给 SDK。若要停止接收设备事件，则需要停止订阅。

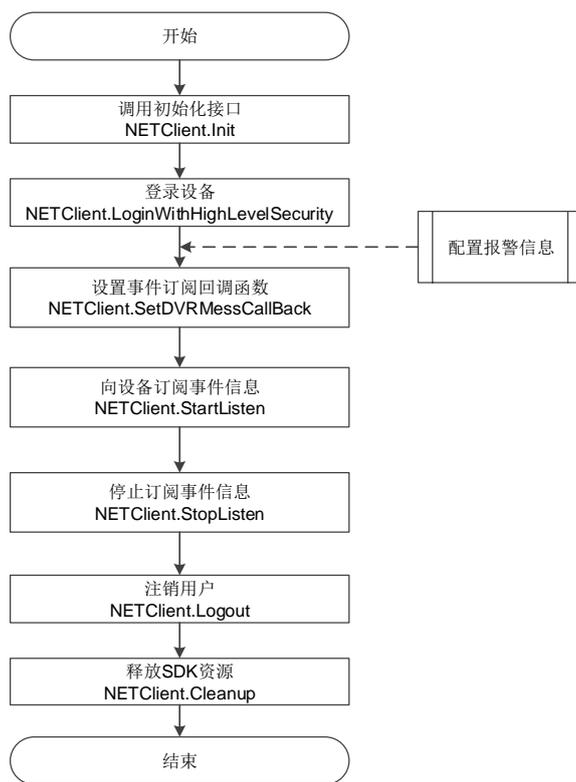
2.2.2.2 接口总览

表2-8 报警事件接口说明

接口	说明
NETClient.StartListen	向设备订阅报警
NETClient.SetDVRMessCallBack	设置设备消息回调函数，用来得到设备当前状态信息，与调用顺序无关，SDK 默认不回调，此回调函数必须先调用报警消息订阅接口 NETClient.StartListen 有效
NETClient.StopListen	停止订阅

2.2.2.3 流程说明

图2-10 报警事件业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 进行报警布防配置（如果报警布防已经配置完成，则可省略）。
- 步骤4 设置报警回调函数 NETClient.SetDVRMessCallBack。
- 步骤5 调用 NETClient.StartListen 函数向设备订阅报警信息。
- 步骤6 整个报警上传过程结束后还需要停止订阅报警接口 NETClient.StopListen。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.2.4 示例代码

```
// 声明静态回调委托
private static fMessCallBackEx m_AlarmCallBack;
m_AlarmCallBack = new fMessCallBackEx(AlarmCallBackEx);
// 设置报警回调
NETClient.SetDVRMessCallBack(m_AlarmCallBack, IntPtr.Zero);
// 报警回调处理
private bool AlarmCallBack(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr
pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser)
{
```

```

EM_ALARM_TYPE type = (EM_ALARM_TYPE)ICommand;
var item = new ListViewItem();
switch (type)
{
    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_EVENT:
        NET_ALARM_ACCESS_CTL_EVENT_INFO access_info =
(NET_ALARM_ACCESS_CTL_EVENT_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_EVENT_INFO));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(access_info.stuTime.ToString());
        item.SubItems.Add("Entry(进门)");
        item.SubItems.Add(access_info.szUserID.ToString());
        item.SubItems.Add(access_info.szCardNo.ToString());
        item.SubItems.Add(access_info.nDoor.ToString());
        switch (access_info.emOpenMethod)
        {
            case EM_ACCESS_DOOROPEN_METHOD.CARD:
                item.SubItems.Add("Card(卡)");
                break;
            case EM_ACCESS_DOOROPEN_METHOD.FACE_RECOGNITION:
                item.SubItems.Add("Face recognition(人脸识别)");
                break;
            default:
                item.SubItems.Add("Unknown(未知)");
                break;
        }
        if (access_info.bStatus)
        {
            item.SubItems.Add("Success(成功)");
        }
        else
        {
            item.SubItems.Add("Failure(失败)");
        }

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)

```

```

        {
            listView_event.Items.RemoveAt(ListViewCount);
        }
        listView_event.EndUpdate();
    });
    Alarm_Index++;
    break;
case EM_ALARM_TYPE.ALARM_ACCESS_CTL_NOT_CLOSE:
    NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO notclose_info =
(NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO));
    item.Text = Alarm_Index.ToString();
    item.SubItems.Add(notclose_info.stuTime.ToString());
    item.SubItems.Add("NotClose(门未关)");
    item.SubItems.Add("");
    item.SubItems.Add("");
    item.SubItems.Add(notclose_info.nDoor.ToString());
    item.SubItems.Add("");
    if (notclose_info.nAction == ALARM_START)
    {
        item.SubItems.Add("Start(开始)");
    }
    else if (notclose_info.nAction == ALARM_STOP)
    {
        item.SubItems.Add("Stop(结束)");
    }
    else
    {
        item.SubItems.Add("");
    }

    this.BeginInvoke(new Action(() =>
    {
        listView_event.BeginUpdate();
        listView_event.Items.Insert(0, item);
        if (listView_event.Items.Count > ListViewCount)
        {
            listView_event.Items.RemoveAt(ListViewCount);
        }
        listView_event.EndUpdate();
    }

```

```

    });
    Alarm_Index++;
    break;
    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_BREAK_IN:
        NET_ALARM_ACCESS_CTL_BREAK_IN_INFO breakin_info =
(NET_ALARM_ACCESS_CTL_BREAK_IN_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_BREAK_IN_INFO));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(breakin_info.stuTime.ToString());
        item.SubItems.Add("BreakIn(闯入)");
        item.SubItems.Add("");
        item.SubItems.Add("");
        item.SubItems.Add(breakin_info.nDoor.ToString());
        item.SubItems.Add("");
        item.SubItems.Add("");

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;
    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_REPEAT_ENTER:
        NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO repeat_info =
(NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(repeat_info.stuTime.ToString());
        item.SubItems.Add("RepeakIn(反潜)");
        item.SubItems.Add("");
        item.SubItems.Add(repeat_info.szCardNo.ToString());
        item.SubItems.Add(repeat_info.nDoor.ToString());
        item.SubItems.Add("");
        item.SubItems.Add("");

```

```

this.BeginInvoke(new Action(() =>
{
    listView_event.BeginUpdate();
    listView_event.Items.Insert(0, item);
    if (listView_event.Items.Count > ListViewCount)
    {
        listView_event.Items.RemoveAt(ListViewCount);
    }
    listView_event.EndUpdate();
}));
Alarm_Index++;
break;
case EM_ALARM_TYPE.ALARM_ACCESS_CTL_DURESS:
    NET_ALARM_ACCESS_CTL_DURESS_INFO duress_info =
(NET_ALARM_ACCESS_CTL_DURESS_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_DURESS_INFO));
    item.Text = Alarm_Index.ToString();
    item.SubItems.Add(duress_info.stuTime.ToString());
    item.SubItems.Add("Duress(胁迫)");
    item.SubItems.Add(duress_info.szUserID.ToString());
    item.SubItems.Add(duress_info.szCardNo.ToString());
    item.SubItems.Add(duress_info.nDoor.ToString());
    item.SubItems.Add("");
    item.SubItems.Add("");

this.BeginInvoke(new Action(() =>
{
    listView_event.BeginUpdate();
    listView_event.Items.Insert(0, item);
    if (listView_event.Items.Count > ListViewCount)
    {
        listView_event.Items.RemoveAt(ListViewCount);
    }
    listView_event.EndUpdate();
}));
Alarm_Index++;
break;
case EM_ALARM_TYPE.ALARM_CHASSISINTRUDED:
    NET_ALARM_CHASSISINTRUDED_INFO chassisintruded_info =

```

```

(NET_ALARM_CHASSISINTRUDED_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_CHASSISINTRUDED_INFO));

    item.Text = Alarm_Index.ToString();
    item.SubItems.Add(chassisintruded_info.stuTime.ToString());
    if (chassisintruded_info.szReaderID.Length > 0)
    {
        item.SubItems.Add("CardreaderAntidemolition(读卡器防拆)");
    }
    else
    {
        item.SubItems.Add("ChassisIntruded(本机防拆)");
    }
    item.SubItems.Add("");
    item.SubItems.Add("");
    item.SubItems.Add(chassisintruded_info.nChannelID.ToString());
    item.SubItems.Add("");
    if (chassisintruded_info.nAction == ALARM_START)
    {
        item.SubItems.Add("Start(开始)");
    }
    else if (chassisintruded_info.nAction == ALARM_STOP)
    {
        item.SubItems.Add("Stop(结束)");
    }
    else
    {
        item.SubItems.Add("");
    }

    this.BeginInvoke(new Action(() =>
    {
        listView_event.BeginUpdate();
        listView_event.Items.Insert(0, item);
        if (listView_event.Items.Count > ListViewCount)
        {
            listView_event.Items.RemoveAt(ListViewCount);
        }
        listView_event.EndUpdate();
    }));
    Alarm_Index++;

```

```

        break;
    case EM_ALARM_TYPE.ALARM_ALARM_EX2:
        NET_ALARM_ALARM_INFO_EX2 alarm_info =
(NET_ALARM_ALARM_INFO_EX2)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ALARM_INFO_EX2));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(alarm_info.stuTime.ToString());
        item.SubItems.Add("AlarmEx2(外部报警)");
        item.SubItems.Add("");
        item.SubItems.Add("");
        item.SubItems.Add(alarm_info.nChannelID.ToString());
        item.SubItems.Add("");
        if (alarm_info.nAction == ALARM_START)
        {
            item.SubItems.Add("Start(开始)");
        }
        else if (alarm_info.nAction == ALARM_STOP)
        {
            item.SubItems.Add("Stop(结束)");
        }
        else
        {
            item.SubItems.Add("");
        }

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;
    default:
        break;
}

```

```

        return true;
    }
    #endregion
// 报警订阅
    if (!m_IsListen)
    {
        bool ret = NETClient.StartListen(m_LoginID);
        if (!ret)
        {
            MessageBox.Show(this, NETClient.GetLastError());
            return;
        }
        m_IsListen = true;
        btn_StartListen.Text = "StopListen(停止订阅)";
    }
    else
    {
        bool ret = NETClient.StopListen(m_LoginID);
        if (!ret)
        {
            MessageBox.Show(this, NETClient.GetLastError());
            return;
        }
        m_IsListen = false;
        listView_Event.Items.Clear();
        btn_StartListen.Text = "StartListen(开启订阅)";
    }
}

```

2.2.3 设备信息查看

2.2.3.1 能力集查询

2.2.3.1.1 简介

设备信息查看，即用户通过 SDK 下发命令给门禁设备，来获取设备的能力集。

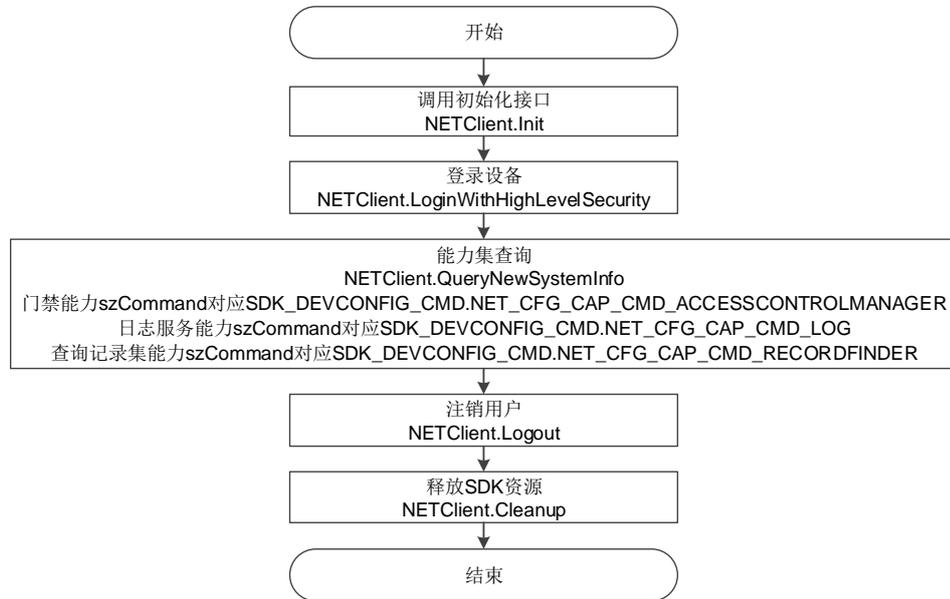
2.2.3.1.2 接口总览

表2-9 能力集查询接口说明

接口	说明
NETClient.QueryNewSystemInfo	查询系统能力信息（日志、记录集、门控能力等）

2.2.3.1.3 流程说明

图2-11 设备信息查看流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.QueryNewSystemInfo 函数，来查询门禁能力集。

表2-10 szCommand 取值对应含义和结构体

szCommand	对应含义	szOutBuffer
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_ACCESSCONTROLMANAGER	门禁能力	CFG_CAP_ACCESSCONTROL
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_LOG	获取日志服务能力	NET_CFG_CAP_LOG
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_RECORDFINDER	获取查询记录集能力	NET_CFG_CAP_RECORDFINDER_INFO

- 步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.3.1.4 示例代码

```

#region Query Access control caps 获取门禁能力
this.button_Query.Enabled = false;
textBox_Caps.Text = "";
textBox_Version.Text = "";
  
```

```

        int nCount = 0;
        CFG_CAP_ACCESSCONTROL info = new CFG_CAP_ACCESSCONTROL();
        object obj = info;
        string strCommand =
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_ACCESSCONTROLMANAGER;
        try
        {
            bool bQuery = NETClient.QueryNewSystemInfo(loginID, -1, strCommand, ref obj,
typeof(CFG_CAP_ACCESSCONTROL), 3000);
            if (bQuery)
            {
                nCount = ((CFG_CAP_ACCESSCONTROL)obj).nAccessControlGroups;
                textBox_Caps.Text = "Access Control Caps(门禁能力):" +
System.Environment.NewLine + "Access Control Num(门禁个数)=" + nCount.ToString() +
System.Environment.NewLine + System.Environment.NewLine;
                this.button_Query.Enabled = true;
            }
        }
        catch (NETClientExcetion ex)
        {
            Console.WriteLine("GetAccessCount error:" + ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetAccessCount error:" + ex.Message);
        }
        #endregion

        #region Query Record Finder caps 获取查询记录能力
        NET_CFG_CAP_RECORDFINDER_INFO RecordInfo = new
NET_CFG_CAP_RECORDFINDER_INFO();
        obj = RecordInfo;
        strCommand = SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_RECORDFINDER;
        try
        {
            bool bQuery = NETClient.QueryNewSystemInfo(loginID, 0, strCommand, ref obj,
typeof(NET_CFG_CAP_RECORDFINDER_INFO), 3000);
            if (bQuery)
            {
                int nMaxPageSize =

```

```

((NET_CFG_CAP_RECORDFINDER_INFO)obj).nMaxPageSize;
        textBox_Caps.Text += "RecordSetFinder Cap(记录集查询能力):" +
System.Environment.NewLine + "MaxPageSize(每个分页最大记录数)=" + nMaxPageSize.ToString() +
System.Environment.NewLine + System.Environment.NewLine;
        this.button_Query.Enabled = true;
    }
}
catch (NETClientExcetion ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
#endregion

#region Query log caps 获取日志服务能力
NET_CFG_CAP_LOG LogInfo = new NET_CFG_CAP_LOG();
obj = LogInfo;
strCommand = SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_LOG;
try
{
    bool bQuery = NETClient.QueryNewSystemInfo(loginID, 0, strCommand, ref obj,
typeof(NET_CFG_CAP_LOG), 3000);

    if (bQuery)
    {
        int dwMaxLogItems = (int)((NET_CFG_CAP_LOG)obj).dwMaxLogItems;
        int dwMaxPageItems = (int)((NET_CFG_CAP_LOG)obj).dwMaxPageItems;
        string strSupportStartNo = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportStartNo)
        {
            strSupportStartNo = "Yes";
        }
        else
        {
            strSupportStartNo = "No";
        }
    }
}

```

```

        string strSupportTypeFilter = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportTypeFilter)
        {
            strSupportTypeFilter = "Yes";
        }
        else
        {
            strSupportTypeFilter = "No";
        }

        string strSupportTimeFilter = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportTimeFilter)
        {
            strSupportTimeFilter = "Yes";
        }
        else
        {
            strSupportTimeFilter = "No";
        }

        textBox_Caps.Text += "Log Cap(日志服务能力):" +
System.Environment.NewLine + "LogMaxItem(最大日志条数)=" + dwMaxLogItems.ToString() +
System.Environment.NewLine;
        textBox_Caps.Text += "MaxPageLogItem(每个分页的最大日志条数)=" +
dwMaxPageItems.ToString() + System.Environment.NewLine;
        textBox_Caps.Text += "IsSupportStartNo(是否支持起始序号)=" +
strSupportStartNo + System.Environment.NewLine;
        textBox_Caps.Text += "IsSupportTypeFilter(是否支持类型过滤)=" +
strSupportTypeFilter + System.Environment.NewLine;
        textBox_Caps.Text += "IsSupportTimeFilter(是否支持时间过滤)=" +
strSupportTimeFilter + System.Environment.NewLine;
        this.button_Query.Enabled = true;
    }
}
catch (NETClientExcetion ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
}

```

2.2.3.2 设备版本、MAC 查看

2.2.3.2.1 简介

设备版本、MAC 查看，即用户通过 SDK 下发命令给门禁设备，来获取设备的序列号、版本号、Mac 地址等内容。

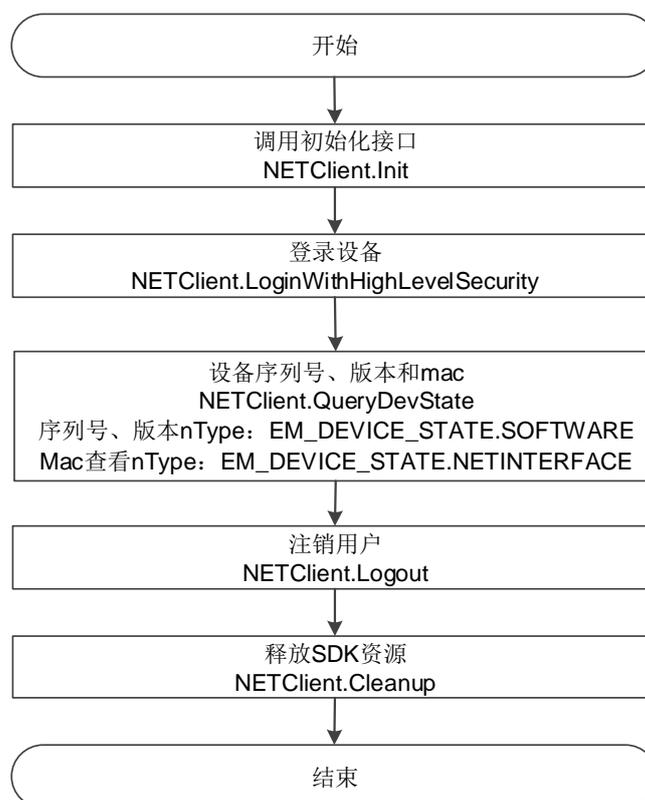
2.2.3.2.2 接口总览

表2-11 设备版本和 MAC 查看接口说明

接口	说明
NETClient.QueryDevState	查询设备状态（查询序列号、软件版本、编译时间、Mac 地址）

2.2.3.2.3 流程说明

图2-12 设备信息查看业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.QueryDevState 函数，来查询门禁设备的序列号、版本和 mac 信息。

表2-12 nType 取值对应含义和结构体

nType	对应含义	pBuf
EM_DEVICE_STATE.SOFTWARE	序列号、版本	NET_DEV_VERSION_INFO

nType	对应含义	pBuf
EM_DEVICE_STATE.NETINTERFACE	Mac 地址	NET_DEV_NETINTERFACE_INFO

步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.3.2.4 示例代码

```
#region Query Version info 获取设备版本信息
    NET_DEV_VERSION_INFO VersionInfo = new NET_DEV_VERSION_INFO();
    object objInfo = VersionInfo;
    bool ret = NETClient.QueryDevState(loginID, EM_DEVICE_STATE.SOFTWARE, ref
objInfo, typeof(NET_DEV_VERSION_INFO), 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    this.button_Query.Enabled = true;
    VersionInfo = (NET_DEV_VERSION_INFO)objInfo;

    textBox_Version.Text += "SerialNo(序列号): " + VersionInfo.szDevSerialNo +
System.Environment.NewLine;
    textBox_Version.Text += "SoftwareVersion(软件版本): " +
VersionInfo.szSoftWareVersion + System.Environment.NewLine;
    textBox_Version.Text += "ReleaseTime(编译时间): " +
((VersionInfo.dwSoftwareBuildDate >> 16) & 0xffff) + "-" + ((VersionInfo.dwSoftwareBuildDate >> 8)
& 0xff) + "-" + (VersionInfo.dwSoftwareBuildDate & 0xff) + System.Environment.NewLine;

    // Query MAC address 获取物理地址
    NET_DEV_NETINTERFACE_INFO[] stuNetInfo = new
NET_DEV_NETINTERFACE_INFO[64];

    for (int i = 0; i < 64; i++)
    {
        stuNetInfo[i].dwSize = (int)Marshal.SizeOf(stuNetInfo[i].GetType());
    }
    object[] objInfo2 = new object[64];
    for (int i = 0; i < 64; i++)
    {
        objInfo2[i] = stuNetInfo[i];
    }
    bool Macret = NETClient.QueryDevState(loginID,
```

```

(int)EM_DEVICE_STATE.NETINTERFACE, ref objInfo2, typeof(NET_DEV_NETINTERFACE_INFO), 5000);
    if (!Macret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    for (int i = 0; i < objInfo2.Length; i++)
    {
        stuNetInfo[i] = (NET_DEV_NETINTERFACE_INFO)objInfo2[i];
    }
    textBox_Version.Text += "MAC(物理地址): " + stuNetInfo[0].szMAC +
System.Environment.NewLine;
#endregion

```

2.2.4 网络设置

2.2.4.1 IP 设置

2.2.4.1.1 简介

IP 设置，即用户通过调用 SDK 接口，对设备的 IP 等信息进行获取和配置，包含 IP 地址、子网掩码、默认网关等信息。

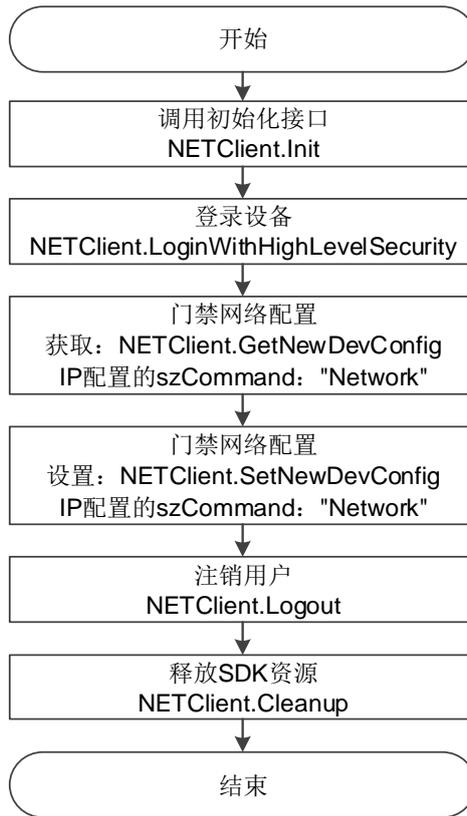
2.2.4.1.2 接口总览

表2-13 IP 设置接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.4.1.3 流程说明

图2-13 IP 设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数，查询门禁 IP 网络配置。
 - szCommand: "Network"。
 - pBuf: CFG_NETWORK_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数，设置门禁 IP 网络配置。
 - szCommand: "Network"。
 - pBuf: CFG_NETWORK_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.4.1.4 示例代码

```
// 获取 IP 网络配置信息
CFG_NETWORK_INFO cfg = new CFG_NETWORK_INFO();

public CFG_NETWORK_INFO GetConfig_Network()
{
    try
    {
        object objTemp = new object();
```

```

        bool bRet = NETClient.GetNewDevConfig(loginID, -1, "Network", ref objTemp,
typeof(CFG_NETWORK_INFO), 5000);
        if (!bRet)
        {
            MessageBox.Show(NETClient.GetLastError());
            return cfg;
        }
        cfg = (CFG_NETWORK_INFO)objTemp;
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig_Network(CFG_NETWORK_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "Network", (object)cfg,
typeof(CFG_NETWORK_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.4.2 主动注册配置

2.2.4.2.1 简介

主动注册配置,即用户通过调用 SDK 接口,对设备的主动注册信息进行配置,包括主动注册使能、设备 ID、服务器信息等。

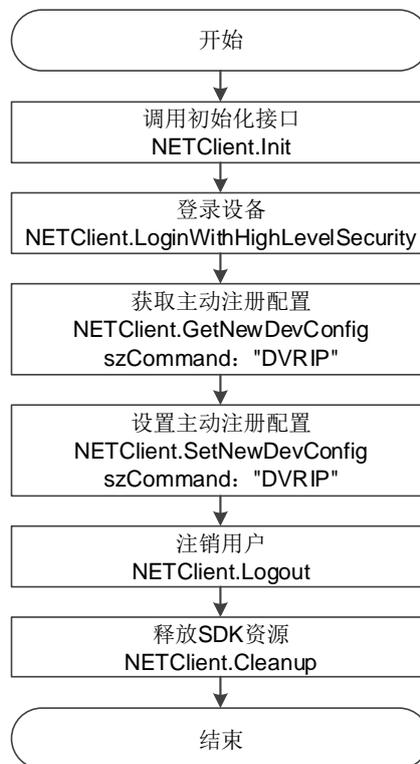
2.2.4.2.2 接口总览

表2-14 主动注册配置接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.4.2.3 流程说明

图2-14 主动注册设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数,完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数,查询门禁 IP 网络配置。
 - szCommand: "DVRIP"。
 - pBuf: NET_CFG_DVRIP_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数,设置门禁 IP 网络配置。
 - szCommand: "DVRIP"。
 - pBuf: CFG_NETWORK_INFO。
- 步骤5 业务执行完成之后,调用 NETClient.Logout 函数登出设备。

步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.4.2.4 示例代码

```
// 获取主动注册网络配置信息
NET_CFG_DVRIP_INFO cfg_Dvrip = new NET_CFG_DVRIP_INFO();
public NET_CFG_DVRIP_INFO GetConfig_Dvrip()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID, -1, "DVRIP", ref objTemp,
        typeof(NET_CFG_DVRIP_INFO), 5000);
        if (bRet)
        {
            cfg_Dvrip = (NET_CFG_DVRIP_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg_Dvrip;
}

public bool SetConfig_Dvrip(NET_CFG_DVRIP_INFO cfg_Dvrip)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "DVRIP", (object)cfg_Dvrip,
        typeof(NET_CFG_DVRIP_INFO), 5000);
    }
}
```

```

catch (NETClientExcetion nex)
{
    Console.WriteLine(nex.Message);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
return bRet;
}

```

2.2.5 设备时间获取和设置

2.2.5.1 设备时间获取和设置

2.2.5.1.1 简介

设备时间设置，即用户通过调用 SDK 接口，对设备时间进行获取和设置的操作。

2.2.5.1.2 接口总览

表2-15 时间设置接口说明

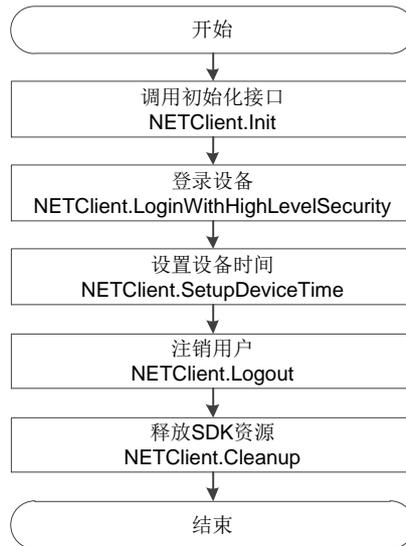
接口	说明
NETClient.QueryDeviceTime	获取设备当前时间
NETClient.SetupDeviceTime	设置设备当前时间

2.2.5.1.3 流程说明

图2-15 时间获取业务流程



图2-16 时间设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.SetupDeviceTime 函数来设置门禁时间信息。
- 步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.5.1.4 示例代码

```
#region Get Deice Time 获取设备时间
NET_TIME stuInfo = new NET_TIME();

bool ret = NETClient.QueryDeviceTime(loginID, ref stuInfo, 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
dateTimePicker_DevTime.Value = stuInfo.ToDateTime();
MessageBox.Show("Get Success(获取成功)");
#endregion

#region Set Device Time 设置设备时间
NET_TIME stuSet = new NET_TIME();
stuSet.dwYear = (uint)dateTimePicker_DevTime.Value.Year;
stuSet.dwMonth = (uint)dateTimePicker_DevTime.Value.Month;
stuSet.dwDay = (uint)dateTimePicker_DevTime.Value.Day;
stuSet.dwHour = (uint)dateTimePicker_DevTime.Value.Hour;
```

```

stuSet.dwMinute = (uint)dateTimePicker_DevTime.Value.Minute;
stuSet.dwSecond = (uint)dateTimePicker_DevTime.Value.Second;

bool ret = NETClient.SetupDeviceTime(loginID, stuSet);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
MessageBox.Show("Set Success(设置成功)");
#endregion

```

2.2.5.2 NTP 服务器和时区设置

2.2.5.2.1 简介

NTP 服务器和时区设置，即用户通过调用 SDK 接口，对 NTP 服务器和时区进行获取和设置。

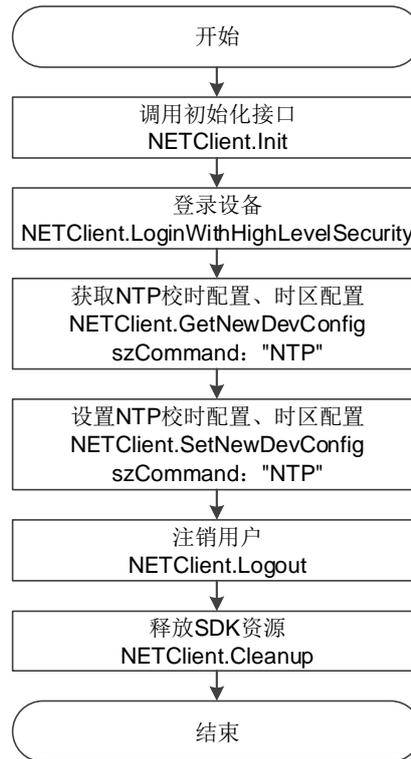
2.2.5.2.2 接口总览

表2-16 NTP 服务器和时区设置接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.5.2.3 流程说明

图2-17 NTP 服务器和时区设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数，查询门禁 NTP 校时配置、时区配置。
 - szCommand: "NTP"。
 - pBuf: NET_CFG_NTP_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数，设置门禁 NTP 校时配置、时区配置。
 - szCommand: "NTP" 。
 - pBuf: NET_CFG_NTP_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.5.2.4 示例代码

```
// NTP 配置
NET_CFG_NTP_INFO cfg = new NET_CFG_NTP_INFO();

public NET_CFG_NTP_INFO GetConfig_NTP()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID, -1, "NTP", ref objTemp,
```

```

typeof(NET_CFG_NTP_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_NTP_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig_NTP(NET_CFG_NTP_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "NTP", (object)cfg,
typeof(NET_CFG_NTP_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.5.3 夏令时设置

2.2.5.3.1 简介

夏令时设置，即用户通过调用 SDK 接口，对夏令时进行获取和设置的操作。

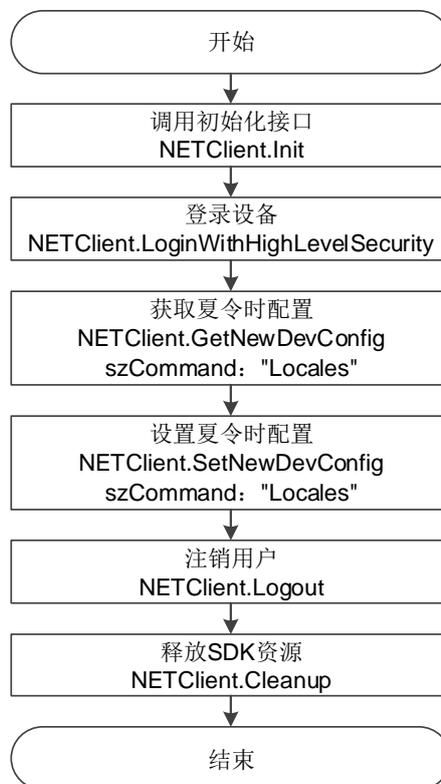
2.2.5.3.2 接口总览

表2-17 夏令时设置接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.5.3.3 流程说明

图2-18 夏令时设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数，查询门禁夏令时信息。
 - szCommand: "Locales" 。
 - pBuf: NET_AV_CFG_Locales。
- 步骤4 调用 NETClient.SetNewDevConfig 函数，设置门禁夏令时信息。
 - szCommand: "Locales" 。
 - pBuf: NET_AV_CFG_Locales。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.5.3.4 示例代码

```
// Locales 配置

public bool GetConfig_Locales()
{
    bool bRet = false;
    try
    {
        cfg_Locales.stuDstStart.nStructSize = Marshal.SizeOf(typeof(AV_CFG_DSTTime));
        cfg_Locales.stuDstEnd.nStructSize = Marshal.SizeOf(typeof(AV_CFG_DSTTime));
        object objTemp = (object)cfg_Locales;
        bRet = NETClient.GetNewDevConfig(loginID, -1, "Locales", ref objTemp,
typeof(NET_AV_CFG_Locales), 5000);
        if (bRet)
        {
            cfg_Locales = (NET_AV_CFG_Locales)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig_Locales(NET_AV_CFG_Locales cfg_Locales)
{
    bool bRet = false;
    try
    {
```

```

        bRet = NETClient.SetNewDevConfig(loginID, -1, "Locales", (object)cfg_Locales,
typeof(NET_AV_CFG_Locales), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.6 维护配置

2.2.6.1 修改登录密码

2.2.6.1.1 简介

修改登录密码，即用户通过调用 SDK 接口，对设备登录密码进行修改。

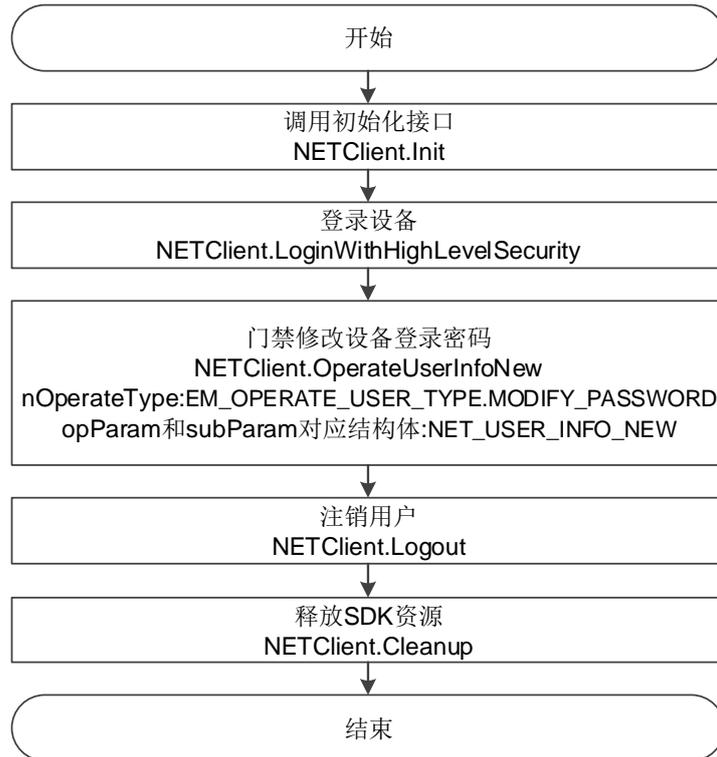
2.2.6.1.2 接口总览

表2-18 修改登录密码接口说明

接口	说明
NETClient.OperateUserInfoNew	操作设备用户

2.2.6.1.3 流程说明

图2-19 修改登录密码业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
调用 NETClient.OperateUserInfoNew 函数来操作设备用户信息，从而修改设备登录密码。
其中参数 nOperateType 为 EM_OPERATE_USER_TYPE.MODIFY_PASSWORD，opParam 和 subParam 对应结构体 NET_USER_INFO_NEW。
- 步骤3 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤4 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.6.1.4 示例代码

```
// 修改设备登录密码
NET_USER_INFO_NEW userInfo = new NET_USER_INFO_NEW();
userInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_USER_INFO_NEW));
userInfo.name = textBox_User.Text.Trim();
userInfo.passWord = textBox_OldPasswd.Text.Trim();
IntPtr inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_USER_INFO_NEW)));

NET_USER_INFO_NEW stuModifyInfo = new NET_USER_INFO_NEW();
stuModifyInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_USER_INFO_NEW));
stuModifyInfo.passWord = textBox_NewPasswd.Text.Trim();
```

```

        IntPtr inSubPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_USER_INFO_NEW)));
        try
        {

            Marshal.StructureToPtr(userInfo, inPtr, true);

            Marshal.StructureToPtr(stuModifyInfo, inSubPtr, true);

            bool ret = NETClient.OperateUserInfoNew(loginID,
EM_OPERATE_USER_TYPE.MODIFY_PASSWORD, inSubPtr, inPtr, 10000);
            if (!ret)
            {
                MessageBox.Show( NETClient.GetLastError());
                return;
            }
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(inSubPtr);
        }
        MessageBox.Show("Modify password successfully.(修改密码成功)");

```

2.2.6.2 重启设备

2.2.6.2.1 简介

重启设备，即用户通过调用 SDK 接口，对设备进行重启操作。

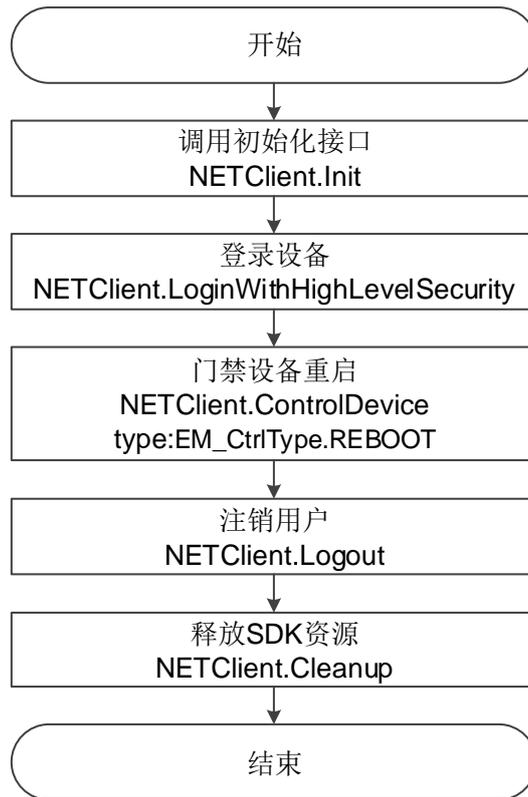
2.2.6.2.2 接口总览

表2-19 重启设备接口说明

接口	说明
NETClient.ControlDevice	设备控制

2.2.6.2.3 流程说明

图2-20 重启设备业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.ControlDevice 函数来控制设备重启。其中参数 type 为 EM_CtrlType.REBOOT。
- 步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.6.2.4 示例代码

```
#region Reboot Device 重启设备

IntPtr inPtr = IntPtr.Zero;

bool ret = NETClient.ControlDevice(loginID, EM_CtrlType.REBOOT, inPtr, 10000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

this.Hide();
```

2.2.6.3 恢复出厂设置

2.2.6.3.1 简介

恢复出厂设置，即用户通过调用 SDK 接口对设备进行恢复出厂设置操作，点击确认后会清空设备端所有配置及人员信息。

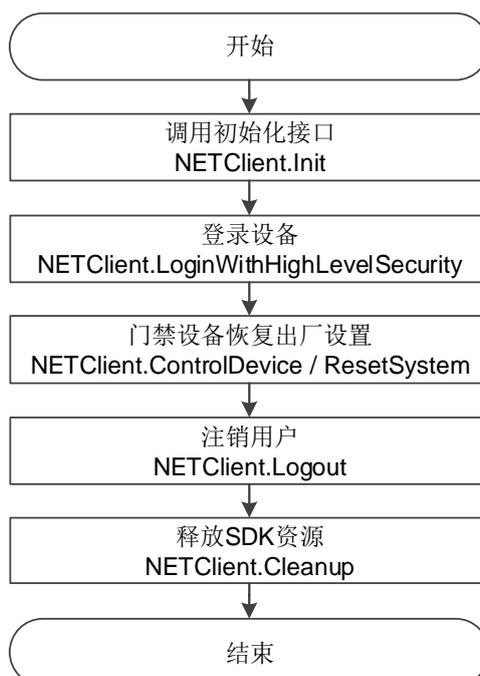
2.2.6.3.2 接口总览

表2-20 恢复出厂设置接口说明

接口	说明
NETClient.ControlDevice	控制设备（恢复出厂设置），支持一体机、控制器
NETClient.ResetSystem	控制设备（恢复出厂设置），支持一体机（推荐）

2.2.6.3.3 流程说明

图2-21 恢复出厂设置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.ResetSystem 函数来控制设备（指纹一体机）恢复出厂设置。或者调用 ControlDevice 函数来控制设备（控制器、指纹一体机）恢复出厂设置，其中参数 type 为 EM_CtrlType.RESTOREDEFAULT。
- 步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.6.3.4 示例代码

```
#region Reset Device 设备恢复出厂设置
NET_IN_RESET_SYSTEM stuResetIn = new NET_IN_RESET_SYSTEM();
stuResetIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_START_FIND));

NET_OUT_RESET_SYSTEM stuResetOut = new NET_OUT_RESET_SYSTEM();
stuResetOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_START_FIND));
bool nRet = NETClient.ResetSystem(loginID, ref stuResetIn, ref stuResetOut, 5000);
if (!nRet)
{
    IntPtr inPtr = IntPtr.Zero;
    inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RESTORE_TEMPSTRUCT)));
    NET_RESTORE_TEMPSTRUCT temp = new NET_RESTORE_TEMPSTRUCT() { value =
NET_RESTORE.ALL };
    Marshal.StructureToPtr(temp, inPtr, true);
    bool ret = NETClient.ControlDevice(loginID, EM_CtrlType.RESTOREDEFAULT, inPtr,
10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
}
this.Hide();
#endregion
```

2.2.6.4 设备升级

2.2.6.4.1 简介

设备升级，即用户通过调用 SDK 接口，对设备程序进行升级。

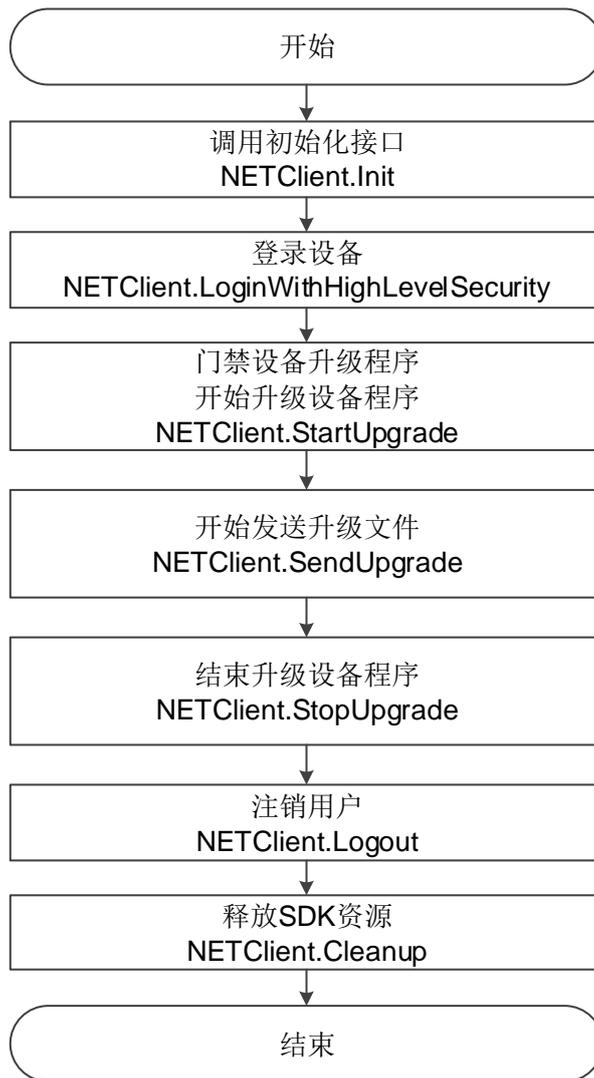
2.2.6.4.2 接口总览

表2-21 设备升级接口说明

接口	说明
NETClient.StartUpgrade	开始升级设备程序--扩展
NETClient.SendUpgrade	开始发送升级文件
NETClient.StopUpgrade	停止升级

2.2.6.4.3 流程说明

图2-22 设备升级业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.StartUpgrade 函数来开始升级设备程序。
- 步骤4 调用 NETClient.SendUpgrade 函数来发送设备升级文件。
- 步骤5 调用 NETClient.StopUpgrade 函数来停止/结束升级设备程序
- 步骤6 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤7 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.6.4.4 示例代码

```
#region upgrade device 升级设备
if (textBox_Path.Text == null || textBox_Path.Text == "")
{
    MessageBox.Show("please choose a upgrade packet file.(请选择远程升级包)");
}
```

```

        return;
    }

    m_UpgradeID = NETClient.StartUpgrade(m_LoginID, EM_UPGRADE_TYPE.BIOS_TYPE,
textBox_Path.Text, m_UpgradeCallBack, IntPtr.Zero);
    if (IntPtr.Zero != m_UpgradeID)
    {
        bool bRet = NETClient.SendUpgrade(m_UpgradeID);
        if (!bRet)
        {
            MessageBox.Show(NETClient.GetLastError());
            button_Upgrade.Enabled = true;
        }
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
        button_Upgrade.Enabled = true;
    }
#endregion

    bool ret = NETClient.StopUpgrade(m_UpgradeID);
    if (ret)
    {
        m_UpgradeID = IntPtr.Zero;
        button_Upgrade.Enabled = true;
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}

```

2.2.6.5 自动维护

2.2.6.5.1 简介

自动维护，即用户通过调用 SDK 接口对设备自动维护进行配置，自动维护包含自动重启时间等信息。

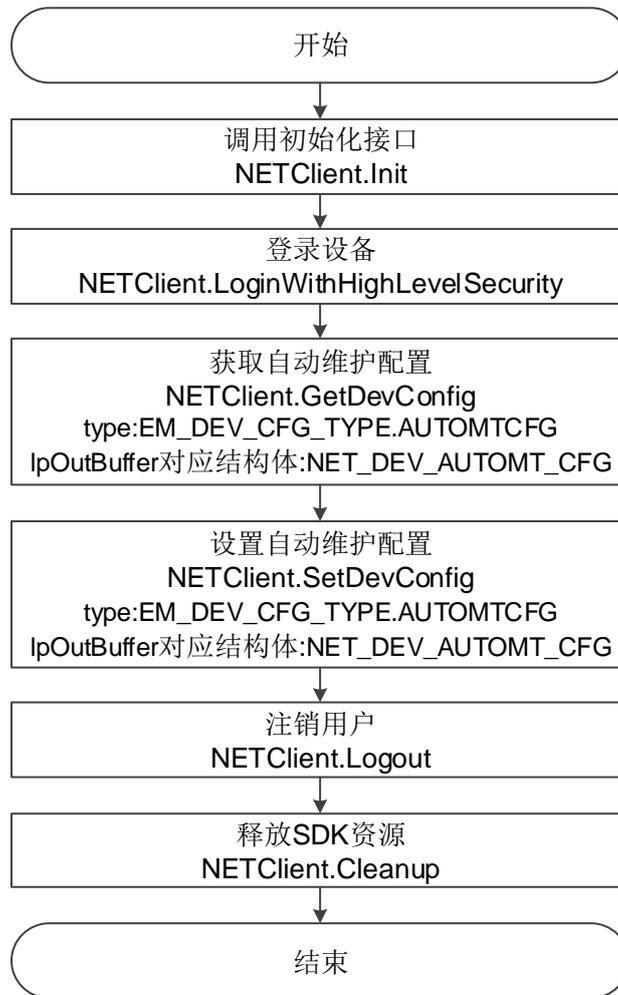
2.2.6.5.2 接口总览

表2-22 自动维护接口说明

接口	说明
NETClient.GetDevConfig	查询配置信息
NETClient.SetDevConfig	设置配置信息

2.2.6.5.3 流程说明

图2-23 自动维护业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetDevConfig 函数来查询门禁自动维护信息。其中参数 type 为 EM_DEV_CFG_TYPE.AUTOMTCFG，IpOutBuffer 对应结构体 NET_DEV_AUTOMT_CFG。
- 步骤4 调用 NETClient.SetDevConfig 函数来设置门禁自动维护信息。其中参数 type 为 EM_DEV_CFG_TYPE.AUTOMTCFG，IpOutBuffer 对应结构体 NET_DEV_AUTOMT_CFG。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.6.5.4 示例代码

```
NET_DEV_AUTOMT_CFG cfg_AutoMT = new NET_DEV_AUTOMT_CFG();
public NET_DEV_AUTOMT_CFG GetDevConfig_AutoMT()
{
    uint ret = 0;
    IntPtr inPtr = IntPtr.Zero;
    try
    {
        inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)));
        Marshal.StructureToPtr(cfg_AutoMT, inPtr, true);
        bool result = NETClient.GetDevConfig(loginID, EM_DEV_CFG_TYPE.AUTOMTCFG,
0, inPtr, (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)), ref ret, 5000);
        if (result && ret == (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)))
        {
            cfg_AutoMT = (NET_DEV_AUTOMT_CFG)Marshal.PtrToStructure(inPtr,
typeof(NET_DEV_AUTOMT_CFG));
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
    }
    return cfg_AutoMT;
}

#region Set auto matrix config 设置自动维护配置
GetDevConfig_AutoMT();
cfg_AutoMT.byAutoRebootDay = (byte)comboBox_RebootDay.SelectedIndex;
cfg_AutoMT.byAutoRebootTime = (byte)comboBox_RebootTime.SelectedIndex;
IntPtr inPtr = IntPtr.Zero;
inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)));
Marshal.StructureToPtr(cfg_AutoMT, inPtr, true);
```

```

bool result = NETClient.SetDevConfig(loginID, EM_DEV_CFG_TYPE.AUTOMTCFG, 0,
inPtr, (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)), 5000);
if (!result)
{
    MessageBox.Show(NETClient.GetLastError());
}
MessageBox.Show("Set successfully.(设置成功)");
#endregion

```

2.2.7 人员管理

2.2.7.1 简介

人员信息，即用户通过调用 SDK，可以对门禁设备的人员信息字段（包含：编号、姓名、人脸、卡、指纹、密码、用户权限、时段、假日计划、用户类型等）进行增删查改的操作。

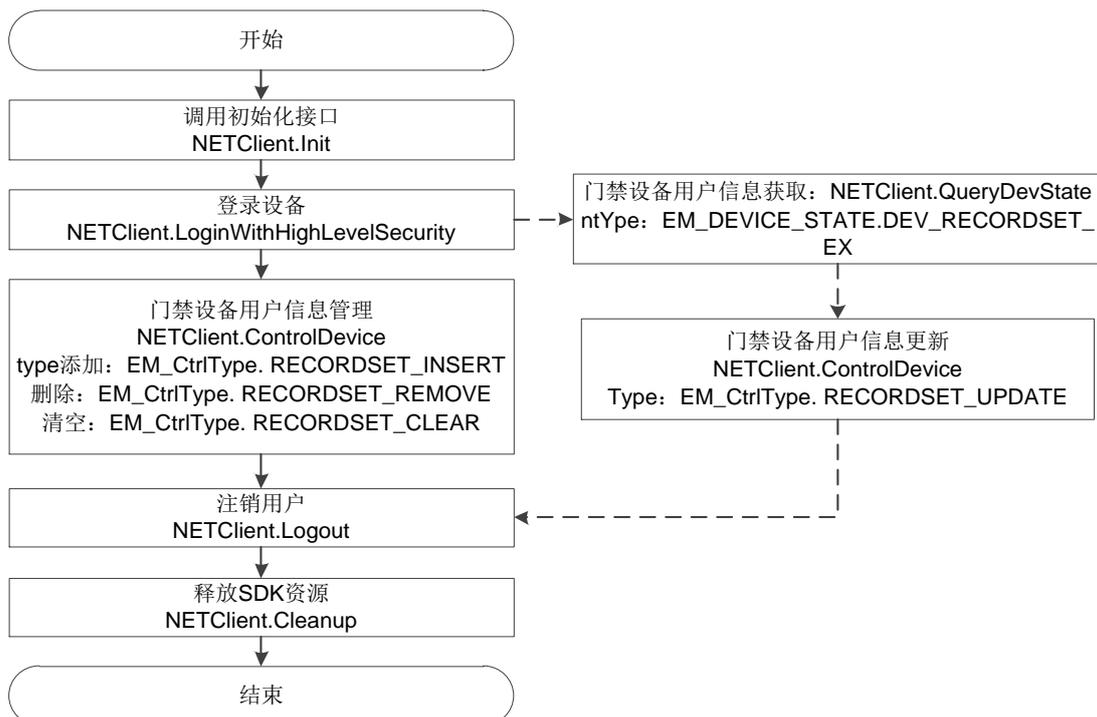
2.2.7.2 接口总览

表2-23 人员信息接口说明

接口	说明
NETClient.ControlDevice	控制设备
NETClient.QueryDevState	查询设备状态

2.2.7.3 流程说明

图2-24 用户信息管理业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
 步骤3 调用 NETClient.ControlDevice 函数来对用户信息进行操作。

表2-24 用户信息管理 type 对应含义和结构体

Type	对应含义	emType	Param
EM_CtrlType. RECORDSET_INSERT	添加用户信息	EM_NET_RECORDER_TYPE.	● NET_CTRL_RECORDSET_INSERT_PARAM
EM_CtrlType. RECORDSET_INSERTEX	添加用户信息(带指纹)	ACCESSCTLCARD	● NET_RECORDSET_ACCESS_CTL_CARD
EM_CtrlType. RECORDSET_REMOVE	删除用户信息	EM_NET_RECORDER_TYPE. ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
EM_CtrlType. RECORDSET_CLEAR	清空用户信息	EM_NET_RECORDER_TYPE. ACCESSCTLCARD	NET_CTRL_RECORDSET_PARAM

步骤4 先调用 NETClient.QueryDevState 接口来获取用户信息。

表2-25 用户信息获取 type 对应含义和结构体

Type	对应含义	emType	Param
EM_DEVICE_STATE. DEV_RECORDSET_EX	获取用户信息	ACCESSCTLCARD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_CARD

步骤5 再调用 NETClient.ControlDevice 函数更新用户信息。

表2-26 用户信息更新 type 对应含义和结构体

Type	对应含义	emType	Param
EM_CtrlType. RECORDSET_UPDATE	更新用户信息	ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
EM_CtrlType. RECORDSET_UPDATEEX	更新用户信息(带指纹)		

步骤6 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤7 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.7.4 注意事项

- 卡号：人员卡号。
- 卡类型：当设置为胁迫卡时，与卡绑定的人员通过卡密码、开门密码、指纹开门时，会触发胁迫报警。
- 卡密码：此密码适用于卡+密码开门模式。
- 时间段：选择配置时间段对应序号。如果未设置序号，请到“时间段”配置中进行配置，具体方法请参见“2.2.9.1 时段配置”。
- 开门密码：设置此密码后，无需刷卡直接输入密码即可开门，详细介绍请参见 2.2.10.5 开门密码。

- 有效次数：仅来宾用户支持设置此字段。
- 是否为首卡：根据实际情况选择。首卡信息的配置方法，请参见“2.2.10.1 分时段、首卡开门”。

2.2.7.5 示例代码

```

#region Get Card Info 获取卡记录
if (textBox_RecNo.Text == null || textBox_RecNo.Text == "")
{
    MessageBox.Show("Please input Rec Number(请输入记录集序号)");
    return;
}
NET_CTRL_RECORDSET_PARAM inp = new NET_CTRL_RECORDSET_PARAM();
inp.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
inp.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

NET_RECORDSET_ACCESS_CTL_CARD info = new
NET_RECORDSET_ACCESS_CTL_CARD();
info.dwSize = (uint)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
info.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
IntPtr infoPtr = IntPtr.Zero;

try
{
    infoPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
    Marshal.StructureToPtr(info, infoPtr, true);
    inp.pBuf = infoPtr;
    inp.nBufLen = Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
    object objInp = inp;
    bool ret = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET_EX, ref objInp, typeof(NET_CTRL_RECORDSET_PARAM),
10000);

    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }

    inp = (NET_CTRL_RECORDSET_PARAM)objInp;
    info = (NET_RECORDSET_ACCESS_CTL_CARD)Marshal.PtrToStructure(inp.pBuf,
typeof(NET_RECORDSET_ACCESS_CTL_CARD));

```

```

// dateTimePicker_CreateTime.Value = info.stuCreateTime.ToDateTime();
textBox_CardNo.Text = info.szCardNo;
textBox_UserID.Text = info.szUserID;
comboBox_CardStatus.SelectedIndex = (int)info.emStatus + 1;
comboBox_CardType.SelectedIndex = (int)info.emType + 1;
textBox_CardPwd.Text = info.szPsw;
m_SelectDoorsAry = info.nNewDoors;
m_SelectTimeAry = info.nNewTimeSectionNo;
textBox_UseTimes.Text = info.nUseTime.ToString();
dateTimePicker_ValidStart.Value = info.stuValidStartTime.ToDateTime();
dateTimePicker_ValidEnd.Value = info.stuValidEndTime.ToDateTime();

checkBox_First.Checked = info.bFirstEnter;

int nCtlType = comboBox_OperateType.SelectedIndex;

if (2 == nCtlType)
{
    OnChangeUIState(7);
}
else
{
    OnChangeUIState(8);
}

// MessageBox.Show("Query Success(获取成功)");
}
finally
{
    // Marshal.FreeHGlobal(infoPtr);
}
#endregion

#region Insert record 添加卡记录
NET_CTRL_RECORDSET_INSERT_PARAM stuInfo = new
NET_CTRL_RECORDSET_INSERT_PARAM();
stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM));

```

```

        stuInfo.stuCtrlRecordSetInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_IN));
        stuInfo.stuCtrlRecordSetInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

        stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

        stuInfo.stuCtrlRecordSetResult.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));

        m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
        m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
        m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
        m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
        m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
        m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

        m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
        m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
        m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
        m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
        m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
        m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
        m_stuInfo.bNewDoor = true;

        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
            }
        }
        m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

        if (m_selectTimesList.Count > 0)
        {
            for (int i = 0; i < m_selectTimesList.Count; i++)
            {

```

```

        m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
    }
}
m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;

m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
    m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay = (uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
            m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
                m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

m_stuInfo.stuValidEndTime.dwYear = (uint)dateTimePicker_ValidEnd.Value.Year;
m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
    m_stuInfo.stuValidEndTime.dwDay = (uint)dateTimePicker_ValidEnd.Value.Day;
    m_stuInfo.stuValidEndTime.dwHour = (uint)dateTimePicker_ValidEnd.Value.Hour;
    m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
    m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

IntPtr inPtr = IntPtr.Zero;
IntPtr ptr = IntPtr.Zero;
try
{
    inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
    Marshal.StructureToPtr(m_stuInfo, inPtr, true);

    stuInfo.stuCtrlRecordSetInfo.pBuf = inPtr;
    stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
    ptr =

```

```

Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
    Marshal.StructureToPtr(stuInfo, ptr, true);
    bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_INSERT, ptr, 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    stuInfo =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(ptr,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
        MessageBox.Show("Execute Success(操作成功)\n RetNo="+
stuInfo.stuCtrlRecordSetResult.nRecNo.ToString());
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
        Marshal.FreeHGlobal(ptr);
    }
#endregion

#region Update record 更新卡记录
NET_CTRL_RECORDSET_PARAM stuInfo = new NET_CTRL_RECORDSET_PARAM();
stuInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
    m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;

```

```

m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
m_stuInfo.bNewDoor = true;
if (m_selectDoorsList.Count > 0)
{
    for (int i = 0; i < m_selectDoorsList.Count; i++)
    {
        m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
    }
}
m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

if (m_selectTimesList.Count > 0)
{
    for (int i = 0; i < m_selectTimesList.Count; i++)
    {
        m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
    }
}
m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
m_stuInfo.stuValidStartTime.dwDay = (uint)dateTimePicker_ValidStart.Value.Day;
m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

m_stuInfo.stuValidEndTime.dwYear = (uint)dateTimePicker_ValidEnd.Value.Year;
m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
m_stuInfo.stuValidEndTime.dwDay = (uint)dateTimePicker_ValidEnd.Value.Day;
m_stuInfo.stuValidEndTime.dwHour = (uint)dateTimePicker_ValidEnd.Value.Hour;
m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;

```

```

        m_stulInfo.bFirstEnter = this.checkBox_First.Checked;

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stulInfo, inPtr, true);

            stulInfo.pBuf = inPtr;
            stulInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stulInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATE, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success(操作成功)");
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }
        OnChangeUIState(nCtlType);
    #endregion
    #region Remove record 移除卡记录
    NET_CTRL_RECORDSET_PARAM stulInfo = new NET_CTRL_RECORDSET_PARAM();
    stulInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
    stulInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;
    m_stulInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());

```

```

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
            Marshal.StructureToPtr(m_stuInfo.nRecNo, inPtr, true);

            stuInfo.pBuf = inPtr;
            stuInfo.nBufLen = (int)Marshal.SizeOf(typeof(int));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
NET_CtrlType.RECORDSET_REMOVE, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success(操作成功)");
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }

        #endregion
        #region Clear card record 清除卡记录
        NET_CTRL_RECORDSET_PARAM inParam = new
NET_CTRL_RECORDSET_PARAM();
        inParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
        inParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;
        IntPtr inPtr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));

```

```

        Marshal.StructureToPtr(inParam, inPtr, true);
        bool ret = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_CLEAR,
inPtr, 10000);

        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        MessageBox.Show("Execute Success(操作成功)");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
    }
#endregion

#region Insert card record with finger 添加带指纹
    NET_CTRL_RECORDSET_INSERT_PARAM stuInfo = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM));

    stuInfo.stuCtrlRecordSetInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_IN));
    stuInfo.stuCtrlRecordSetInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

    stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

    stuInfo.stuCtrlRecordSetResult.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));

    m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
    m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
    m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
    m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
    m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;

```

```

        m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

        m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
        m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
        m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
        m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
        m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
        m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
        m_stuInfo.bNewDoor = true;
        //m_stuInfo.nNewDoorNum;
        // m_stuInfo.sznDoors;
        // m_stuInfo.nNewTimeSectionNum;
        //m_stuInfo.nNewTimeSectionNo
        m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay = (uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear = (uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay = (uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour = (uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;

        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;

```

```

        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);
        Marshal.Copy(FingerPrintInfo, 0, m_stuInfo.stuFingerPrintInfoEx.pPacketData,
m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.stuCtrlRecordSetInfo.pBuf = inPtr;
            stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_INSERT, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            stuInfo =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(ptr,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
            MessageBox.Show("Execute Success(操作成功)\n RetNo=" +
stuInfo.stuCtrlRecordSetResult.nRecNo.ToString());
        }
        finally
        {
            Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }
    }
}
#endregion

```

```

#region Update record with finger 更新带指纹
NET_CTRL_RECORDSET_PARAM stuInfo = new NET_CTRL_RECORDSET_PARAM();
stuInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
m_stuInfo.bNewDoor = true;
if (m_selectDoorsList.Count > 0)
{
    for (int i = 0; i < m_selectDoorsList.Count; i++)
    {
        m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
    }
}
m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

if (m_selectTimesList.Count > 0)
{
    for (int i = 0; i < m_selectTimesList.Count; i++)
    {
        m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
    }
}
m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
m_stuInfo.stuValidStartTime.dwYear =

```

```

(uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay = (uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear = (uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay = (uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour = (uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);
        Marshal.Copy(FingerPrintInfo, 0, m_stuInfo.stuFingerPrintInfoEx.pPacketData,
m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.pBuf = inPtr;

```

```

        stuInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

        ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
        Marshal.StructureToPtr(stuInfo, ptr, true);
        bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATEEX, ptr, 10000);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        MessageBox.Show("Execute Success(操作成功)");
    }
    finally
    {
        Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
        Marshal.FreeHGlobal(inPtr);
        Marshal.FreeHGlobal(ptr);
    }
    OnChangeUIState(nCtlType);
#endregion

#region Update record with finger 更新带指纹
NET_CTRL_RECORDSET_PARAM stuInfo = new NET_CTRL_RECORDSET_PARAM();
stuInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
m_stuInfo.emStatus =

```

```

(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
    m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
    m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
    m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
    m_stuInfo.bNewDoor = true;
    if (m_selectDoorsList.Count > 0)
    {
        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
        }
    }
    m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

    if (m_selectTimesList.Count > 0)
    {
        for (int i = 0; i < m_selectTimesList.Count; i++)
        {
            m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
        }
    }
    m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
    m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
    m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
    m_stuInfo.stuValidStartTime.dwDay = (uint)dateTimePicker_ValidStart.Value.Day;
    m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
    m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
    m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

    m_stuInfo.stuValidEndTime.dwYear = (uint)dateTimePicker_ValidEnd.Value.Year;
    m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
    m_stuInfo.stuValidEndTime.dwDay = (uint)dateTimePicker_ValidEnd.Value.Day;
    m_stuInfo.stuValidEndTime.dwHour = (uint)dateTimePicker_ValidEnd.Value.Hour;
    m_stuInfo.stuValidEndTime.dwMinute =

```

```

(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);
        Marshal.Copy(FingerPrintInfo, 0, m_stuInfo.stuFingerPrintInfoEx.pPacketData,
m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.pBuf = inPtr;
            stuInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATEEX, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success(操作成功)");
        }
        finally
        {

```

```
Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
Marshal.FreeHGlobal(inPtr);
Marshal.FreeHGlobal(ptr);
}
OnChangeUIState(nCtlType);
#endregion
```

2.2.8 门配置

2.2.8.1 简介

门配置信息，即用户通过调用 SDK 接口，对门禁设备的门配置进行获取和设置的操作，包括开门模式、门锁保持时间、关门超时时长、假期时间段编号、开门时间段及报警使能项等。

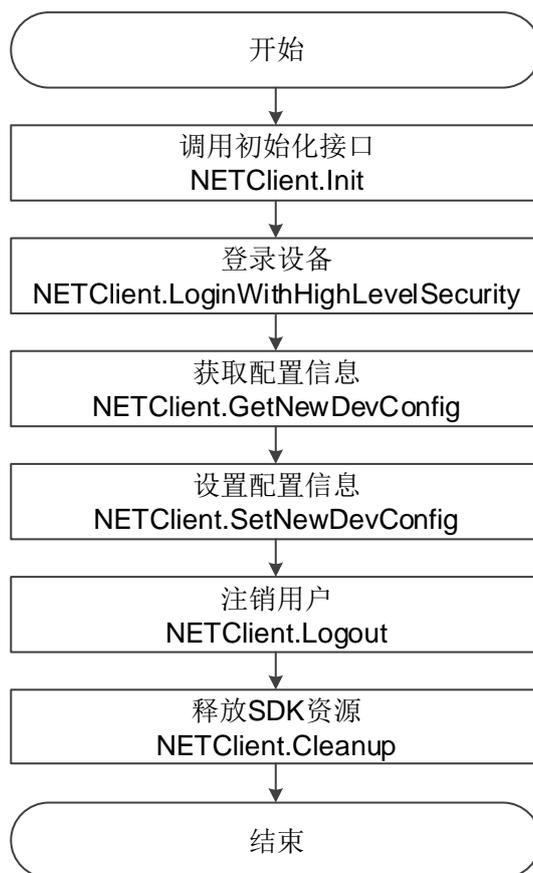
2.2.8.2 接口总览

表2-27 门配置信息接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.8.3 流程说明

图2-25 门配置信息业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数来查询门禁门配置信息。其中参数 strCommand 为"AccessControl"，obj 对应结构体 NET_CFG_ACCESS_EVENT_INFO。

表2-28 NET_CFG_ACCESS_EVENT_INFO 参数说明

CFG_ACCESS_EVENT_INFO 参数	对应含义
emState	门状态
nUnlockHoldInterval	开门时长
nCloseTimeout	关门超时时长
emDoorOpenMethod	开门模式
bDuressAlarmEnable	胁迫
bBreakInAlarmEnable	闯入报警使能
bRepeatEnterAlarm	重复进入报警使能
abDoorNotClosedAlarmEnable	互锁报警使能
abSensorEnable	门磁使能

步骤4 调用 NETClient.SetNewDevConfig 函数来设置门禁门配置信息。其中参数 strCommand 为 "AccessControl"，obj 对应结构体 NET_CFG_ACCESS_EVENT_INFO。

步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.8.4 注意事项

- 开启闯入报警、门未关报警功能时，需要同时开启门磁功能，才能实现闯入报警、门未关报警功能。
- 常开、常闭、远程验证引用的序号，通过“时间段”进行设置，具体设置方法请参见“2.2.9.1 时段配置”。

2.2.8.5 示例代码

```
// 获取门配置信息
NET_CFG_ACCESS_EVENT_INFO cfg = new NET_CFG_ACCESS_EVENT_INFO();
public NET_CFG_ACCESS_EVENT_INFO? GetConfig()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID,
cmbBox_DoorIndex.SelectedIndex, "AccessControl", ref objTemp,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
        cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        if (!bRet)
        {
            MessageBox.Show(NETClient.GetLastError());
            return cfg;
        }
        cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{

```

```

        bool bRet = false;
        try
        {
            bRet = NETClient.SetNewDevConfig(loginID, cmbBox_DoorIndex.SelectedIndex,
"AccessControl", (object)cfg, typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
        }
        catch (NETClientExcetion nex)
        {
            Console.WriteLine(nex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        return bRet;
    }

```

2.2.9 门时间配置

2.2.9.1 时段配置

2.2.9.1.1 简介

时段配置信息，即用户通过调用 SDK 接口，对门禁设备的门时间段进行获取和设置的操作。此模板的配置不能直接对设备生效，需要被其他功能模块调用。

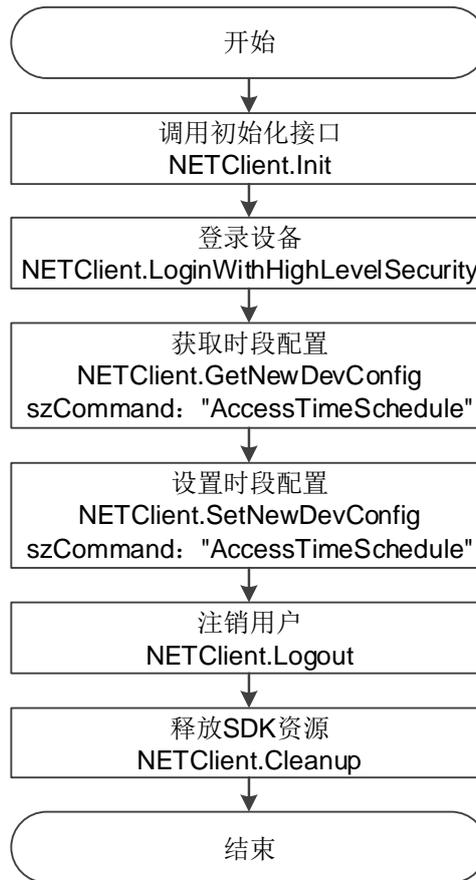
2.2.9.1.2 接口总览

表2-29 时段接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.9.1.3 流程说明

图2-26 时段业务配置流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数，查询门禁时段信息。
- szCommand: "AccessTimeSchedule"。
 - pBuf: CFG_ACCESS_TIMESCHEDULE_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数，设置门禁时段信息。
- szCommand: "AccessTimeSchedule"。
 - pBuf: CFG_ACCESS_TIMESCHEDULE_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.9.1.4 示例代码

```
// 获取时段配置信息
timeSchedule = new NET_CFG_ACCESS_TIMESCHEDULE_INFO();
object objInfo = timeSchedule;
bool ret = NETClient.GetNewDevConfig(loginID, comboBox_Index.SelectedIndex + 1,
CFG_CMD_ACCESSTIMESCHEDULE, ref objInfo, typeof(NET_CFG_ACCESS_TIMESCHEDULE_INFO),
10000);
```

```

        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        timeSchedule = (NET_CFG_ACCESS_TIMESCHEDULE_INFO)objInfo;

        textBox_Name.Text = timeSchedule.szName;

        var temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4];
        dateTimePicker_Start1.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
        dateTimePicker_End1.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

        temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 1];
        dateTimePicker_Start2.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
        dateTimePicker_End2.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

        temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 2];
        dateTimePicker_Start3.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
        dateTimePicker_End3.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

        temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 3];
        dateTimePicker_Start4.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
        dateTimePicker_End4.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

        MessageBox.Show("Get success(获取成功)");

        object objInfo = timeSchedule;
        bool ret = NETClient.SetNewDevConfig(loginID, comboBox_Index.SelectedIndex + 1,
CFG_CMD_ACCESSTIMESCHEDULE, objInfo, typeof(NET_CFG_ACCESS_TIMESCHEDULE_INFO),
10000);
        if (!ret)
        {

```

```

        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    MessageBox.Show("Set success");

```

2.2.9.2 常开常闭时间段配置

2.2.9.2.1 简介

常开常闭时间段配置，即用户通过调用 SDK 接口，对门禁设备的时间段配置进行获取和设置的操作，这里包括常开/常闭时段配置、远程验证时间段配置等信息。

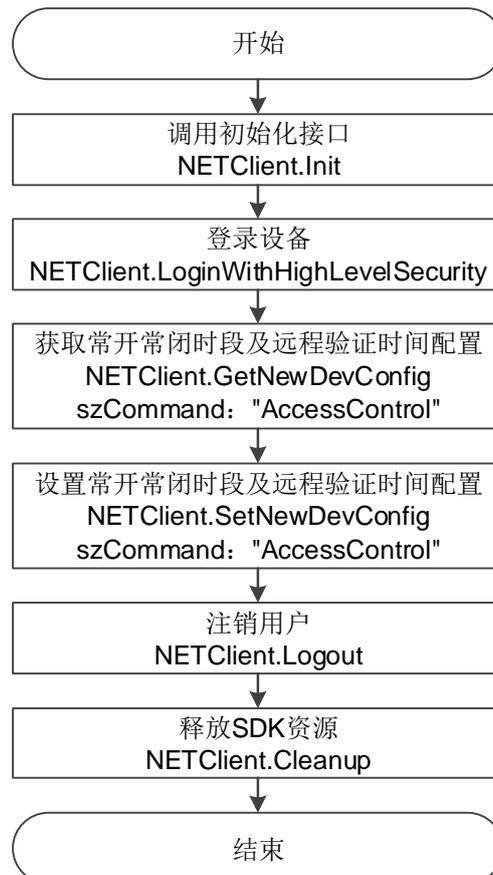
2.2.9.2.2 接口总览

表2-30 常开常闭时间段配置接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.9.2.3 流程说明

图2-27 常开常闭时间段配置流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数，查询门禁常开常闭时段配置信息以及远程验证时间段。
- szCommand: "AccessControl"。
 - pBuf: NET_CFG_ACCESS_EVENT_INFO。

表2-31 NET_CFG_ACCESS_EVENT_INFO 参数说明

NET_CFG_ACCESS_EVENT_INFO 参数	对应含义
nOpenAlwaysTimeIndex	常开时间段配置
nCloseAlwaysTimeIndex	常闭时间段配置
stuAutoRemoteCheck	远程验证时间段

- 步骤4 调用 NETClient.SetNewDevConfig 函数，设置门禁常开常闭时段配置信息以及远程验证时间段。
- szCommand: "AccessControl"。
 - pBuf: NET_CFG_ACCESS_EVENT_INFO。

表2-32 NET_CFG_ACCESS_EVENT_INFO 参数说明

NET_CFG_ACCESS_EVENT_INFO 参数	对应含义
nOpenAlwaysTimeIndex	常开时间段配置
nCloseAlwaysTimeIndex	常闭时间段配置
stuAutoRemoteCheck	远程验证时间段

- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.9.2.4 注意事项

常开、常闭、远程验证引用的序号，通过“时间段”进行设置，具体设置方法请参见“2.2.9.1 时段配置”。

2.2.9.2.5 示例代码

```
// 获取常开常闭、远程验证时段配置信息
NET_CFG_ACCESS_EVENT_INFO cfg = new NET_CFG_ACCESS_EVENT_INFO();
public NET_CFG_ACCESS_EVENT_INFO? GetConfig()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID,
comboBox_DoorNo.SelectedIndex, "AccessControl", ref objTemp,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        }
    }
}
```

```

        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, comboBox_DoorNo.SelectedIndex,
"AccessControl", (object)cfg, typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.9.3 假期配置

2.2.9.3.1 简介

假期配置，即用户通过调用 SDK 接口，对门禁设备的假期进行获取和配置的操作。

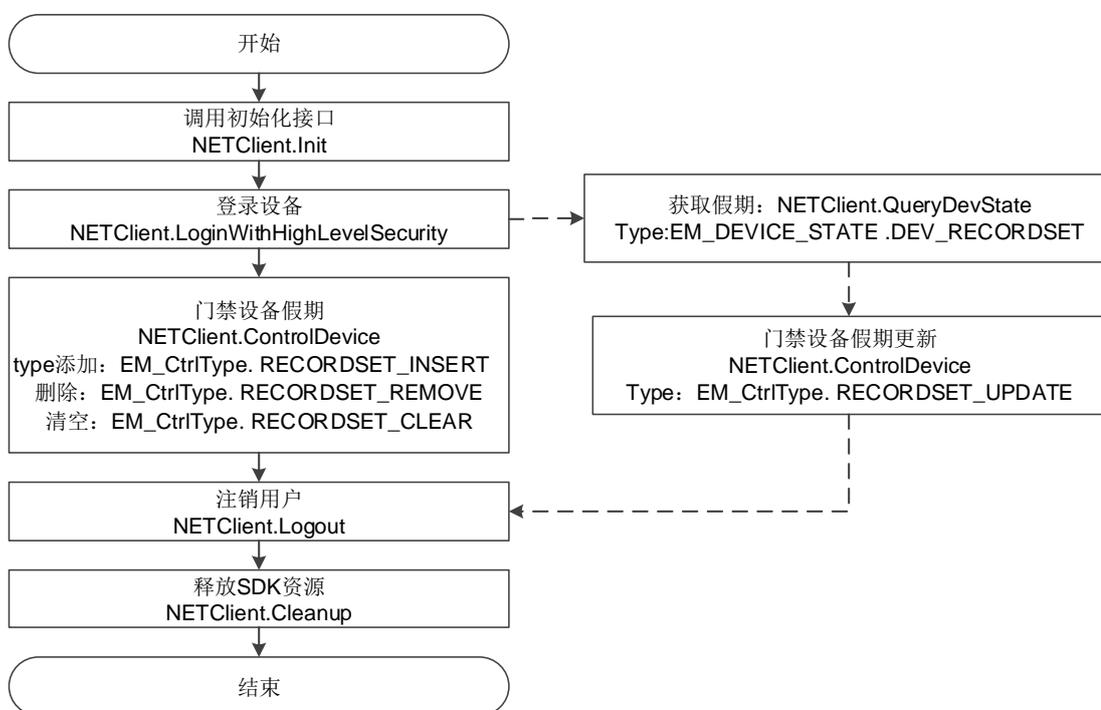
2.2.9.3.2 接口总览

表2-33 假期配置接口说明

接口	说明
NETClient.ControlDevice	控制设备
NETClient.QueryDevState	查询设备状态

2.2.9.3.3 流程说明

图2-28 假期配置流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.ControlDevice 函数来对假期信息进行操作。

表2-34 type 对应操作类型和结构体

type	对应含义	emType	Param
EM_CtrlType. RECORDSET_INSERT	添加假期	EM_NET_RECO RD_TYPE. ACCESSCTLHO LIDAY	NET_CTRL_RECORDSET_INSERT_ PARAM NET_RECORDSET_HOLIDAY

type	对应含义	emType	Param
EM_CtrlType. RECORDSET_REMOVE	删除假期	EM_NET_RECO RD_TYPE. ACCESSCTLHO OLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY
EM_CtrlType. RECORDSET_CLEAR	清空假期	EM_NET_RECO RD_TYPE. ACCESSCTLHO OLIDAY	NET_CTRL_RECORDSET_PARAM

步骤4 先调用 NETClient.QueryDevState 接口来获取假期信息。

表2-35 type 对应操作类型和结构体

type	对应含义	emType	Param
EM_DEVICE_STATE.DEV_REC ORDSET	获取假期	EM_NET_RECORD _TYPE.ACCESSCTL HOLIDAY	NET_CTRL_RECORDSET_PA RAM NET_RECORDSET_HOLIDAY

步骤5 再调用 NETClient.ControlDevice 函数更新假期信息。

表2-36 type 对应操作类型和结构体

type	对应含义	emType	Param
EM_CtrlType. RECORDSET_UPDATE	更新假期	EM_NET_RECORD_ TYPE.ACCESSCTLH OLIDAY	NET_CTRL_RECORDSET_PAR AM NET_RECORDSET_HOLIDAY

步骤6 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤7 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.9.3.4 示例代码

```
// 获取假期
IntPtr pBuf = IntPtr.Zero;

NET_RECORDSET_HOLIDAY result = new NET_RECORDSET_HOLIDAY();
NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();

try
{
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(result));

    //package for pwd
    result.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
    result.dwSize = (uint)Marshal.SizeOf(result);
    Marshal.StructureToPtr(result, pBuf, true);

    //package stuParam
    stuParam.pBuf = pBuf;
    stuParam.nBufLen = Marshal.SizeOf(result);
}
```

```

        stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
        object obj = stuParam;

        bool bRet = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET, ref obj, typeof(NET_CTRL_RECORDSET_PARAM), 3000);
        if (bRet)
        {
            update_holiday = (NET_RECORDSET_HOLIDAY)Marshal.PtrToStructure(pBuf,
typeof(NET_RECORDSET_HOLIDAY));

            dateTimePicker_StartTime.Value =
update_holiday.stuStartTime.ToDateTime();
            dateTimePicker_EndTime.Value = update_holiday.stuEndTime.ToDateTime();
            textBox_HolidayNo.Text = update_holiday.szHolidayNo;
            MessageBox.Show("Get succeed(获取成功)。");
            OnChangeUIState(5);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
    }

// 添加假期
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_INSERT_PARAM stuInsertParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    NET_CTRL_RECORDSET_INSERT_PARAM stuOutParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();

```

```

NET_RECORDSET_HOLIDAY stuHoliday = new NET_RECORDSET_HOLIDAY();
object obj = stuHoliday;
InitStruct(ref obj);
stuHoliday = (NET_RECORDSET_HOLIDAY)obj;
stuHoliday.dwSize = (uint)Marshal.SizeOf(stuHoliday);

stuHoliday.stuStartTime.dwYear = (uint)dateTimePicker_StartTime.Value.Year;
stuHoliday.stuStartTime.dwMonth = (uint)dateTimePicker_StartTime.Value.Month;
stuHoliday.stuStartTime.dwDay = (uint)dateTimePicker_StartTime.Value.Day;

stuHoliday.stuEndTime.dwYear = (uint)dateTimePicker_EndTime.Value.Year;
stuHoliday.stuEndTime.dwMonth = (uint)dateTimePicker_EndTime.Value.Month;
stuHoliday.stuEndTime.dwDay = (uint)dateTimePicker_EndTime.Value.Day;

stuHoliday.szHolidayNo = textBox_HolidayNo.Text;

if (m_selectDoorsList.Count > 0)
{
    for (int i = 0; i < m_selectDoorsList.Count; i++)
    {
        stuHoliday.sznDoors[i] = m_selectDoorsList[i];
    }
}
stuHoliday.nDoorNum = m_selectDoorsList.Count;

try
{
    pParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
    pBuffer =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_HOLIDAY)));

    Marshal.StructureToPtr(stuHoliday, pBuffer, true);

    //package stuInsertParam
    stuInsertParam.stuCtrlRecordSetInfo.pBuf = pBuffer;
    stuInsertParam.stuCtrlRecordSetInfo.nBufLen = Marshal.SizeOf(stuHoliday);
    stuInsertParam.dwSize = (uint)Marshal.SizeOf(stuInsertParam);
    stuInsertParam.stuCtrlRecordSetInfo.dwSize =
(uint)Marshal.SizeOf(stuInsertParam.stuCtrlRecordSetInfo);

```

```

        stuInsertParam.stuCtrlRecordSetInfo.emType =
EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuInsertParam.stuCtrlRecordSetResult.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));
        Marshal.StructureToPtr(stuInsertParam, pParam, true);

        bool bRet = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_INSERT,
pParam, 3000);

        stuOutParam =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(pParam,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
        if (bRet && stuOutParam.stuCtrlRecordSetResult.nRecNo > 0)
        {
            MessageBox.Show("Instert succeed(添加成功)。RecNO(记录号):" +
stuOutParam.stuCtrlRecordSetResult.nRecNo);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    //free resource
    finally
    {
        Marshal.FreeHGlobal(pParam);
        Marshal.FreeHGlobal(pBuf);
    }

// 假期更新
    update_holiday.stuEndTime.dwDay = (uint)dateTimePicker_EndTime.Value.Day;

    update_holiday.szHolidayNo = textBox_HolidayNo.Text;
    if (m_selectDoorsList.Count > 0)
    {

```

```

        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            update_holiday.sznDoors[i] = m_selectDoorsList[i];
        }
    }
    update_holiday.nDoorNum = m_selectDoorsList.Count;

    bool bRet = false;
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();
    try
    {
        pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
        pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(update_holiday));

        Marshal.StructureToPtr(update_holiday, pBuf, true);
        stuParam.pBuf = pBuf;
        stuParam.nBufLen = Marshal.SizeOf(update_holiday);
        stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
        Marshal.StructureToPtr(stuParam, pParam, true);

        bRet = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_UPDATE,
pParam, 3000);

        if (bRet)
        {
            MessageBox.Show("Update succeed(更新成功)。");
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

```

        finally
        {
            Marshal.FreeHGlobal(pParam);
            Marshal.FreeHGlobal(pBuf);
        }

// 移除假期
IntPtr pParam = IntPtr.Zero;
IntPtr pBuf = IntPtr.Zero;
NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();
stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
stuParam.pBuf = IntPtr.Zero;
stuParam.nBufLen = 0;
try
{
    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(int.Parse(textBox_RecNo.Text)));
    Marshal.StructureToPtr(int.Parse(textBox_RecNo.Text), pBuf, true);
    stuParam.pBuf = pBuf;
    stuParam.nBufLen = Marshal.SizeOf(int.Parse(textBox_RecNo.Text));
    Marshal.StructureToPtr(stuParam, pParam, true);

    result = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_REMOVE,
pParam, 3000);

    if (result)
    {
        MessageBox.Show("Remove succeed(删除成功。)");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{

```

```

        Marshal.FreeHGlobal(pBuf);
        Marshal.FreeHGlobal(pParam);
    }

// 清空假期
    IntPtr pParam = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);

    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    Marshal.StructureToPtr(stuParam, pParam, true);

    bool result = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_CLEAR,
pParam, 3000);
    if (result)
    {
        MessageBox.Show("Clear succeed(清空成功)。");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}

```

2.2.10 门高级配置

2.2.10.1 分时段、首卡开门

2.2.10.1.1 简介

分时段、首卡开门，即用户通过调用 SDK 接口，对门禁设备的分时段、首卡、首用户开门配置进行获取和设置的操作。

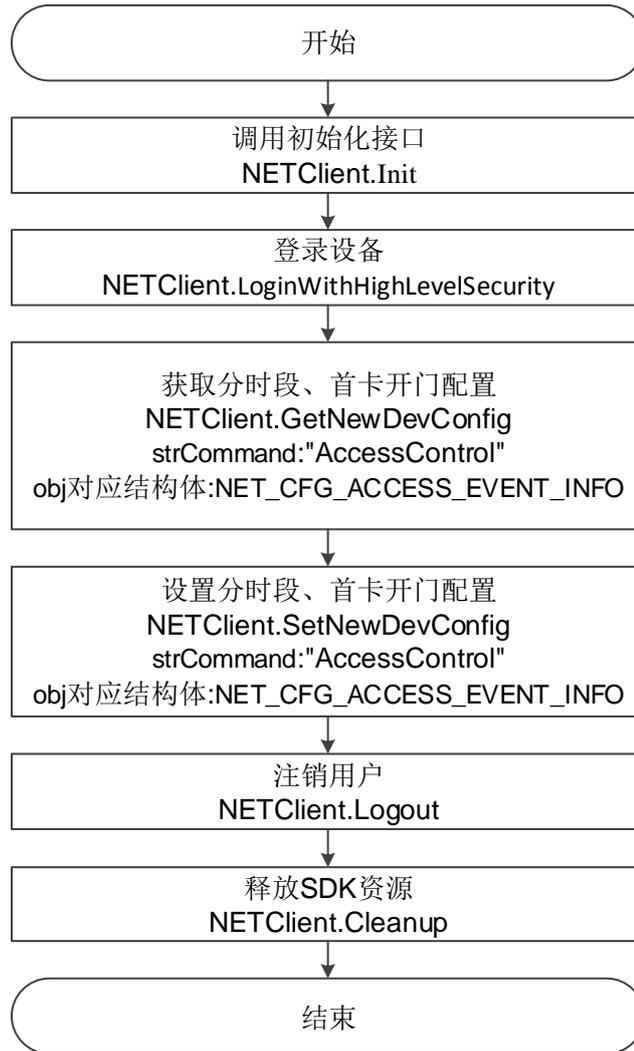
2.2.10.1.2 接口总览

表2-37 分时段和首卡开门接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.10.1.3 流程说明

图2-29 分时段和首卡开门业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数来查询门禁分时段、首卡开门配置信息。其中参数 strCommand 为"AccessControl"，obj 对应结构体 NET_CFG_ACCESS_EVENT_INFO。

表2-38 CFG_ACCESS_EVENT_INFO 结构体参数说明

参数	含义
stuDoorTimeSection	分时段开门配置
stuFirstEnterInfo	首用户/卡开门配置

- 步骤4 调用 NETClient.SetNewDevConfig 函数来设置门禁分时段、首卡开门配置信息。其中参数 strCommand 为"AccessControl"，obj 对应结构体 NET_CFG_ACCESS_EVENT_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.10.1.4 注意事项

- 首卡引用的用户 ID 为卡号。

- 为实现首卡开门功能，添加该用户 ID 对应人员到设备时，需要选择“首卡”，否则，首卡开门功能无法使用。

2.2.10.1.5 示例代码

```
// 获取分时段开门、首卡/用户开门配置信息
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID, comboBox_Channel.SelectedIndex,
CFG_CMD_ACCESS_EVENT, ref objTemp, typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, comboBox_Channel.SelectedIndex,
CFG_CMD_ACCESS_EVENT, (object)cfg, typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
    }
}
```

```

    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

```

2.2.10.2 多人组合开门

2.2.10.2.1 简介

多人组合开门，即用户通过调用 SDK 接口，对门禁设备的多人组合开门配置信息进行获取和设置的操作。

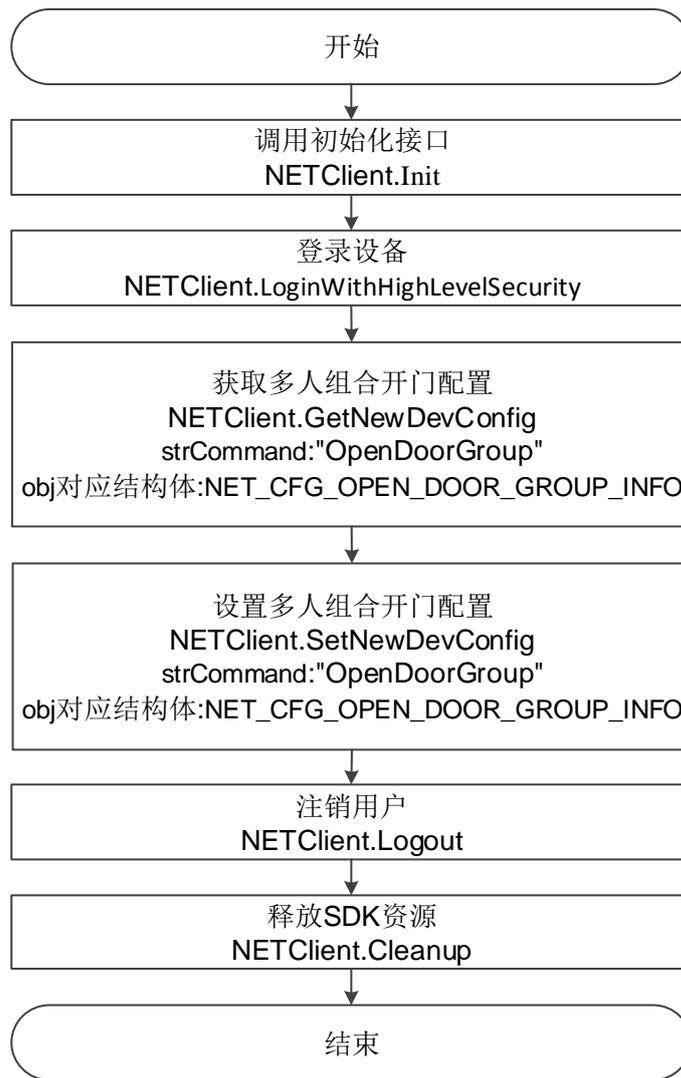
2.2.10.2.2 接口总览

表2-39 多人组合开门接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.10.2.3 流程说明

图2-30 多人组合开门业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数来查询门禁多人组合开门配置信息。其中参数 strCommand 为"OpenDoorGroup", obj 对应结构体 NET_CFG_OPEN_DOOR_GROUP_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数来设置门禁多人组合开门配置信息。其中参数 strCommand 为"OpenDoorGroup", obj 对应结构体 NET_CFG_OPEN_DOOR_GROUP_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.10.2.4 注意事项

- 配置多人组合开门功能之前，需要先将人员添加到设备。
- 组合序号：将人员进行分组，1 个门最多可以配置 4 个人员组。
- 人员组：序号组内的人员。1 个组内人员人数最多为 50 人。人员需要提前添加到设备。
- 有效人数：每个组内的有效人数≤组内当前人数，1 个门的有效人员总数≤5 人。

- 设置人员组开门方式：仅支持卡、指纹，二选一。

2.2.10.2.5 示例代码

```
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        cfg_info.stuGroupInfo = new NET_CFG_OPEN_DOOR_GROUP[4];
        for (int i = 0; i < cfg_info.stuGroupInfo.Length; i++)
        {
            cfg_info.stuGroupInfo[i].stuGroupDetail = new
NET_CFG_OPEN_DOOR_GROUP_DETAIL[64];
        }

        object objTemp = cfg_info;
        bRet = NETClient.GetNewDevConfig(loginID, comboBox_Door.SelectedIndex,
CFG_CMD_OPEN_DOOR_GROUP, ref objTemp, typeof(NET_CFG_OPEN_DOOR_GROUP_INFO),
10000);

        if (bRet)
        {
            cfg_info = (NET_CFG_OPEN_DOOR_GROUP_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_OPEN_DOOR_GROUP_INFO cfg)
```

```

    {
        bool bRet = false;
        try
        {
            bRet = NETClient.SetNewDevConfig(loginID, comboBox_Door.SelectedIndex,
            CFG_CMD_OPEN_DOOR_GROUP, (object)cfg, typeof(NET_CFG_OPEN_DOOR_GROUP_INFO), 5000);
            if (!bRet)
            {
                MessageBox.Show(NETClient.GetLastError());
            }
        }
        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }

```

2.2.10.3 多门互锁

2.2.10.3.1 简介

多门互锁配置，即用户通过调用 SDK 接口，对门禁设备的多门互锁配置进行获取和设置的操作。

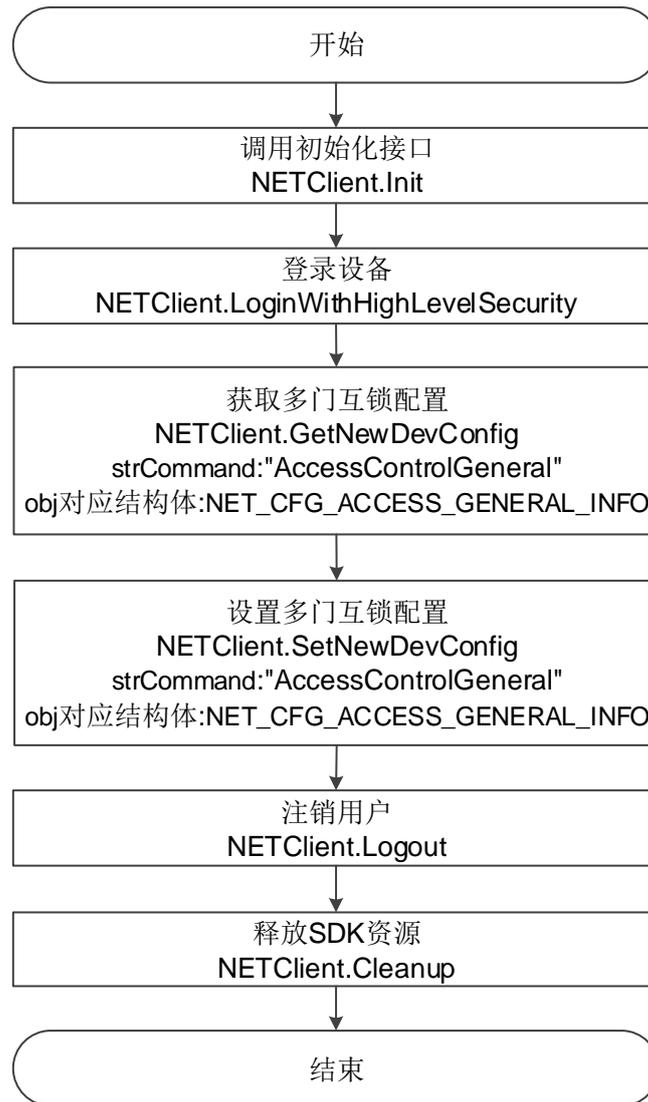
2.2.10.3.2 接口总览

表2-40 多门互锁接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.10.3.3 流程说明

图2-31 多门互锁配置业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数来查询门禁多门互锁配置信息。其中参数 strCommand 为"AccessControlGeneral"，obj 对应结构体 NET_CFG_ACCESS_GENERAL_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数来设置门禁多门互锁配置信息。其中参数 strCommand 为"AccessControlGeneral"，obj 对应结构体 NET_CFG_ACCESS_GENERAL_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.10.3.4 注意事项

1 个设备只能支持 1 种多门互锁方案。

2.2.10.3.5 示例代码

```
//获取多门互锁配置信息
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID, -1, CFG_CMD_ACCESS_GENERAL, ref
objTemp, typeof(NET_CFG_ACCESS_GENERAL_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_ACCESS_GENERAL_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcection nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_ACCESS_GENERAL_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, CFG_CMD_ACCESS_GENERAL,
(object)cfg, typeof(NET_CFG_ACCESS_GENERAL_INFO), 5000);
        if (!bRet)
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
}
```

```

        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

```

2.2.10.4 防反潜

2.2.10.4.1 简介

防反潜配置，即用户通过调用 SDK 接口，对门禁设备的防反潜配置进行获取和设置的操作。

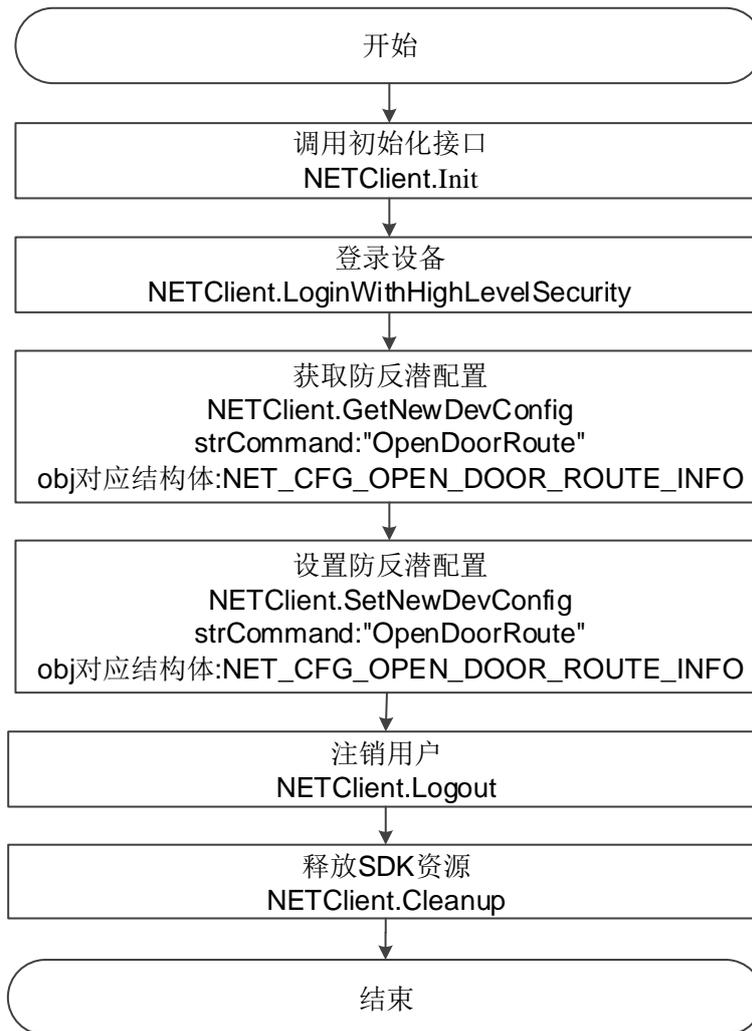
2.2.10.4.2 接口总览

表2-41 防反潜接口说明

接口	说明
NETClient.GetNewDevConfig	查询配置信息
NETClient.SetNewDevConfig	设置配置信息

2.2.10.4.3 流程说明

图2-32 防反潜配置流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetNewDevConfig 函数来查询门禁防反潜配置信息。其中参数 strCommand 为"OpenDoorRoute", obj 对应结构体 NET_CFG_OPEN_DOOR_ROUTE_INFO。
- 步骤4 调用 NETClient.SetNewDevConfig 函数来设置门禁防反潜配置信息。其中参数 strCommand 为"OpenDoorRoute", obj 对应结构体 NET_CFG_OPEN_DOOR_ROUTE_INFO。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.10.4.4 注意事项

1 个设备只能支持 1 种防反潜方案。

2.2.10.4.5 示例代码

```
//获取防反潜配置信息
```

```

public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID, -1, CFG_CMD_OPEN_DOOR_ROUTE,
ref objTemp, typeof(NET_CFG_OPEN_DOOR_ROUTE_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_OPEN_DOOR_ROUTE_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_OPEN_DOOR_ROUTE_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, CFG_CMD_OPEN_DOOR_ROUTE,
(object)cfg, typeof(NET_CFG_OPEN_DOOR_ROUTE_INFO), 5000);
        if (!bRet)
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
}

```

```

        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }

```

2.2.10.5 开门密码

2.2.10.5.1 简介

开门密码，即用户通过调用 SDK 接口，对门禁设备开门密码进行增删查改等操作。

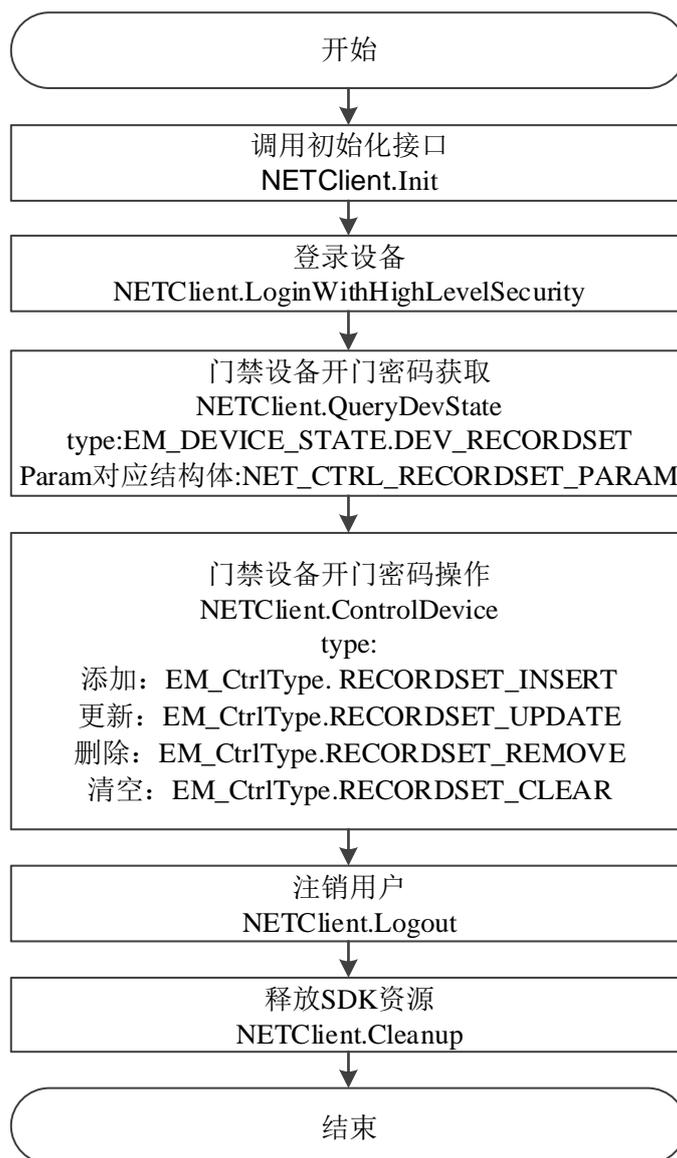
2.2.10.5.2 接口总览

表2-42 开门密码接口说明

接口	说明
NETClient.ControlDevice	设备控制

2.2.10.5.3 流程说明

图2-33 开门密码配置流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 先调用 NETClient.QueryDevState 接口来获取开门密码信息。

表2-43 Type 对应操作和结构体

type	含义	Param
EM_DEVICE_STATE.DEV_RECORDSET	获取开门密码	NET_CTRL_RECORDSET_PARAM

- 步骤4 调用 NETClient.ControlDevice 函数来对开门密码信息进行操作。

表2-44 type 对应操作和结构体

type	含义	Param
EM_CtrlType.RECORDSET_INSERT	添加开门密码	NET_CTRL_RECORDSET_INSERT_PARAM NET_RECORDSET_ACCESS_CTL_PWD

type	含义	Param
EM_CtrlType.RECORDSET_UPD ATE	更新开门密码	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD
EM_CtrlType.RECORDSET_REM OVE	删除开门密码	NET_CTRL_RECORDSET_PARAM
EM_CtrlType.RECORDSET_CLE AR	清空开门密码	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD

步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.10.5.4 注意事项

- 配置多人组合开门功能之前，需要先将人员添加到设备。
- 用户编号：人员卡号。

2.2.10.5.5 示例代码

```
// 获取密码
IntPtr pBuf = IntPtr.Zero;

NET_RECORDSET_ACCESS_CTL_PWD result = new
NET_RECORDSET_ACCESS_CTL_PWD();
NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();

try
{
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(result));

    //package for pwd
    result.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());

    result.dwSize = (uint)Marshal.SizeOf(result);
    Marshal.StructureToPtr(result, pBuf, true);

    //package stuParam
    stuParam.pBuf = pBuf;
    stuParam.nBufLen = Marshal.SizeOf(result);
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
    object obj = stuParam;

    bool bRet = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET, ref obj, typeof(NET_CTRL_RECORDSET_PARAM), 3000);
```

```

        if (bRet)
        {
            update_pwd =
(NET_RECORDSET_ACCESS_CTL_PWD)Marshal.PtrToStructure(pBuf,
typeof(NET_RECORDSET_ACCESS_CTL_PWD));
            m_SelectDoorsAry = update_pwd.sznDoors;
            textBox_OpenPwd.Text =
Encoding.UTF8.GetString(update_pwd.szDoorOpenPwd);
            textBox_UserID.Text = Encoding.UTF8.GetString(update_pwd.szUserID);
            MessageBox.Show("Get succeed(获取成功)。");
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
    }

// 添加密码
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_INSERT_PARAM stuInsertParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    NET_CTRL_RECORDSET_INSERT_PARAM stuOutParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();

    NET_RECORDSET_ACCESS_CTL_PWD stuPwd = new
NET_RECORDSET_ACCESS_CTL_PWD();
    object obj = stuPwd;
    InitStruct(ref obj);
    stuPwd = (NET_RECORDSET_ACCESS_CTL_PWD)obj;
    stuPwd.dwSize = (uint)Marshal.SizeOf(stuPwd);

    Encoding.Default.GetBytes( textBox_UserID.Text, 0, textBox_UserID.Text.Length,

```

```

stuPwd.szUserID, 0);
        Encoding.Default.GetBytes(textBox_OpenPwd.Text, 0, textBox_OpenPwd.Text.Length,
stuPwd.szDoorOpenPwd, 0);
        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                stuPwd.sznDoors[i] = m_selectDoorsList[i];
            }
        }
        stuPwd.nDoorNum = m_selectDoorsList.Count;

        try
        {
            pParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
            pBuf =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_PWD)));

            Marshal.StructureToPtr(stuPwd, pBuf, true);

            //package stuInsertParam
            stuInsertParam.stuCtrlRecordSetInfo.pBuf = pBuf;
            stuInsertParam.stuCtrlRecordSetInfo.nBufLen = Marshal.SizeOf(stuPwd);
            stuInsertParam.dwSize = (uint)Marshal.SizeOf(stuInsertParam);
            stuInsertParam.stuCtrlRecordSetInfo.dwSize =
(uint)Marshal.SizeOf(stuInsertParam.stuCtrlRecordSetInfo);
            stuInsertParam.stuCtrlRecordSetInfo.emType =
EM_NET_RECORD_TYPE.ACCESSCTLPWD;
            stuInsertParam.stuCtrlRecordSetResult.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));
            Marshal.StructureToPtr(stuInsertParam, pParam, true);

            bool bRet = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_INSERT,
pParam, 3000);

            stuOutParam =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(pParam,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
            if (bRet && stuOutParam.stuCtrlRecordSetResult.nRecNo > 0)

```

```

        {
            MessageBox.Show("Inster succeed(添加成功)。RecNO(记录号):" +
stuOutParam.stuCtrlRecordSetResult.nRecNo);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    //free resource
    finally
    {
        Marshal.FreeHGlobal(pParam);
        Marshal.FreeHGlobal(pBuf);
    }

// 获取再更新
    Encoding.Default.GetBytes(textBox_UserID.Text, 0, textBox_UserID.Text.Length,
update_pwd.szUserID, 0);
    Encoding.Default.GetBytes(textBox_OpenPwd.Text, 0, textBox_OpenPwd.Text.Length,
update_pwd.szDoorOpenPwd, 0);
    if (m_selectDoorsList.Count > 0)
    {
        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            update_pwd.sznDoors[i] = m_selectDoorsList[i];
        }
    }
    update_pwd.nDoorNum = m_selectDoorsList.Count;

    bool bRet = false;
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();

```

```

try
{
    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    pBuffer = Marshal.AllocHGlobal(Marshal.SizeOf(update_pwd));

    Marshal.StructureToPtr(update_pwd, pBuffer, true);
    stuParam.pBuf = pBuffer;
    stuParam.nBufLen = Marshal.SizeOf(update_pwd);
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
    Marshal.StructureToPtr(stuParam, pParam, true);

    bRet = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_UPDATE,
pParam, 3000);
    if (bRet)
    {
        MessageBox.Show("Update succeed(更新成功)。");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    Marshal.FreeHGlobal(pParam);
    Marshal.FreeHGlobal(pBuf);
}

// 移除密码
bool result = false;
IntPtr pParam = IntPtr.Zero;
IntPtr pBuffer = IntPtr.Zero;
NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();
stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;

```

```

stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
stuParam.pBuf = IntPtr.Zero;
stuParam.nBufLen = 0;
try
{
    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(int.Parse(textBox_RecNo.Text)));
    Marshal.StructureToPtr(int.Parse(textBox_RecNo.Text), pBuf, true);
    stuParam.pBuf = pBuf;
    stuParam.nBufLen = Marshal.SizeOf(int.Parse(textBox_RecNo.Text));
    Marshal.StructureToPtr(stuParam, pParam, true);

    result = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_REMOVE,
pParam, 3000);

    if (result)
    {
        MessageBox.Show("Remove succeed(删除成功。)");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    Marshal.FreeHGlobal(pBuf);
    Marshal.FreeHGlobal(pParam);
}

// 清空密码
IntPtr pParam = IntPtr.Zero;
NET_CTRL_RECORDSET_PARAM stuParam = new NET_CTRL_RECORDSET_PARAM();
stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);

pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));

```

```

        Marshal.StructureToPtr(stuParam, pParam, true);

        bool result = NETClient.ControlDevice(loginID, EM_CtrlType.RECORDSET_CLEAR,
pParam, 3000);
        if (result)
        {
            MessageBox.Show("Clear succeed(清空成功)。");
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }

//

```

2.2.11 记录查询

2.2.11.1 开门记录

2.2.11.1.1 简介

开门记录查询，即用户调用 SDK 接口对门禁设备的开门进行查询。查询可以设置查询条件，查询条目数。

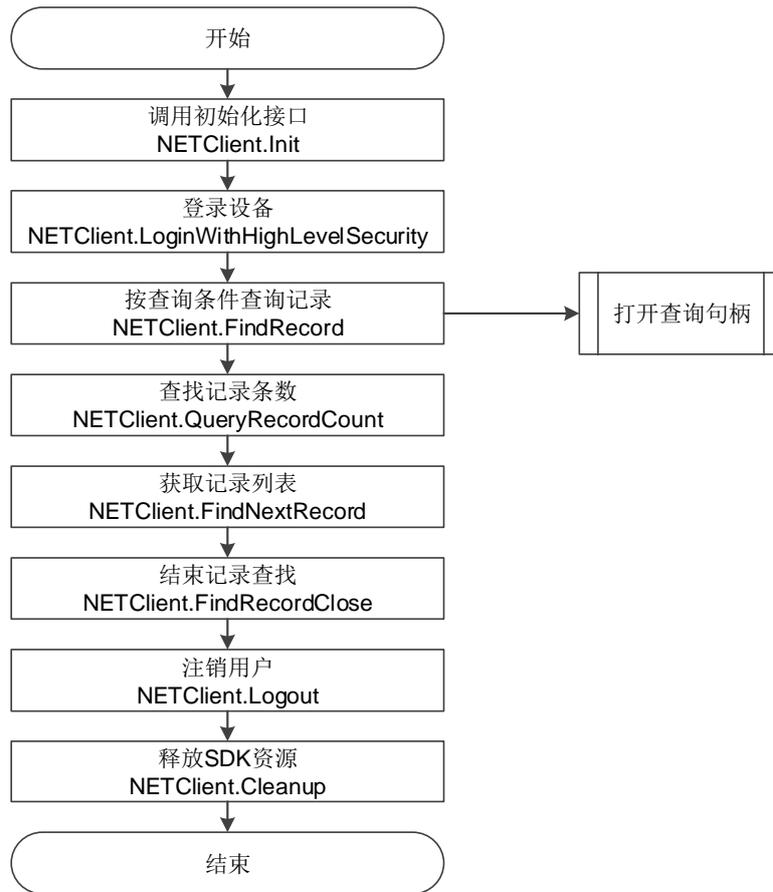
2.2.11.1.2 接口总览

表2-45 记录查询接口说明

接口	说明
NETClient.QueryRecordCount	查找记录条数
NETClient.FindRecord	按查询条件查询记录
NETClient.FindNextRecord	查找记录。nFilecount:需要查询的条数，出参中 nRetRecordNum 为查询到的记录条数，当查询到的条数小于 nFilecount 时查询结束
NETClient.FindRecordClose	结束记录查询

2.2.11.1.3 流程说明

图2-34 记录查询业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 先调用 NETClient.FindRecord 函数获得查询句柄。
emType 开门记录：EM_NET_RECORD_TYPE_ACCESSCTLCARDREC。
- 步骤4 再调用 NETClient.QueryRecordCount 函数查找记录条数。
- 步骤5 再调用 NETClient.FindNextRecord 函数获取记录列表。
- 步骤6 查询完毕可以调用 NETClient.FindRecordClose 关闭查询句柄。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.11.1.4 示例代码

```
// 开始查询
m_LogNum = 0;
if (IntPtr.Zero == m_FindDoorRecordID)
{
    NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX condition = new
NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX();
```

```

        condition.dwSize =
(uint)Marshal.SizeOf(typeof(NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX));
        condition.bTimeEnable = true;
        condition.stStartTime = NET_TIME.FromDateTime(dateTimePicker_Start.Value);
        condition.stEndTime = NET_TIME.FromDateTime(dateTimePicker_End.Value);
        object obj = condition;

        bool ret = NETClient.FindRecord(loginID,
EM_NET_RECORD_TYPE.ACCESSCTLCARDREC_EX, obj,
typeof(NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX), ref m_FindDoorRecordID,
10000);

        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        btn_StartQuery.Text = "StopQuery(停止查询)";
        btn_NextFind.Enabled = true;
        btn_GetRecordCount.Enabled = true;
        dateTimePicker_Start.Enabled = false;
        dateTimePicker_End.Enabled = false;
    }
    else
    {
        NETClient.FindRecordClose(m_FindDoorRecordID);
        m_FindDoorRecordID = IntPtr.Zero;
        btn_StartQuery.Text = "StartQuery(开始查询)";
        btn_NextFind.Enabled = false;
        btn_GetRecordCount.Enabled = false;
        dateTimePicker_Start.Enabled = true;
        dateTimePicker_End.Enabled = true;
        textBox_Count.Text = "";

        listView_DoorRecord.Items.Clear();
    }

// 获取查询条数
    if (IntPtr.Zero == m_FindDoorRecordID)
    {
        return;
    }

```

```

    }

    int nCount = 0;
    try
    {
        if (NETClient.QueryRecordCount(m_FindDoorRecordID, ref nCount, 3000))
        {
            textBox_Count.Text = nCount.ToString();
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
    }
    catch (NETClientExcetion ex)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

// 查询下一条记录
listView_DoorRecord.Items.Clear();
int max = 10;
int retNum = 0;
List<object> ls = new List<object>();
for (int i = 0; i < max; i++)
{
    NET_RECORDSET_ACCESS_CTL_CARDREC cardrec = new
NET_RECORDSET_ACCESS_CTL_CARDREC();
    cardrec.dwSize =
(uint)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARDREC));
    ls.Add(cardrec);
}
NETClient.FindNextRecord(m_FindDoorRecordID, max, ref retNum, ref ls,

```

```

typeof(NET_RECORDSET_ACCESS_CTL_CARDREC), 10000);
    BeginInvoke(new Action() =>
    {
        foreach (var item in ls)
        {
            NET_RECORDSET_ACCESS_CTL_CARDREC info =
(NET_RECORDSET_ACCESS_CTL_CARDREC)item;
            var listitem = new ListViewItem();
            listitem.Text = info.nRecNo.ToString();
            listitem.SubItems.Add(info.stuTime.ToString());
            listitem.SubItems.Add(info.szCardNo);
            listitem.SubItems.Add(info.bStatus.ToString());
            listitem.SubItems.Add(info.nDoor.ToString());
            listitem.SubItems.Add(info.emMethod.ToString());
            if (listView_DoorRecord != null)
            {
                listView_DoorRecord.BeginUpdate();
                listView_DoorRecord.Items.Add(listitem);
                listView_DoorRecord.EndUpdate();
            }
        }
    });

```

2.2.11.2 设备日志

2.2.11.2.1 简介

设备日志，即用户通过调用 SDK 接口，可以对门禁设备的操作日志进行查询，查询可以指定日志类型，可以分页查询，可以指定查询数目等信息。

2.2.11.2.2 接口总览

表2-46 设备日志接口说明

接口	说明
NETClient.QueryDevLogCount	查询设备日志条数
NETClient.StartQueryLog	开始查询日志
NETClient.QueryNextLog	获取日志
NETClient.StopQueryLog	结束查询日志

2.2.11.2.3 流程说明

图2-35 设备日志业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.QueryDevLogCount 函数设置查询日志条数。
- 步骤4 调用 NETClient.StartQueryLog 函数开始查询日志信息。
pInParam: NET_IN_START_QUERYLOG。
pOutParam: NET_OUT_START_QUERYLOG。
- 步骤5 调用 NETClient.QueryNextLog 函数获取日志信息。
pInParam: NET_IN_QUERYNEXTLOG。
pOutParam: NET_OUT_QUERYNEXTLOG。
- 步骤6 调用 NETClient.StopQueryLog 函数停止日志查询。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.2.11.2.4 示例代码

```
// 开始查询日志信息
m_LogNum = 0;
if (IntPtr.Zero == m_FindLogID)
{
```

```

NET_IN_START_QUERYLOG stuIn = new NET_IN_START_QUERYLOG();
stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_START_QUERYLOG));
NET_OUT_START_QUERYLOG stuOut = new NET_OUT_START_QUERYLOG();
stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_START_QUERYLOG));

m_FindLogID = NETClient.StartQueryLog(loginID, ref stuIn, ref stuOut, 5000);
//CLIENT_StartQueryLog(m_LoginID, &stuIn, &stuOut, SDK_API_WAIT);
if (IntPtr.Zero == m_FindLogID)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

btn_StartQuery.Text = "StopQuery(停止查询)";

btn_NextLog.Enabled = true;
btn_GetLogCount.Enabled = true;
}
else
{
    NETClient.StopQueryLog(m_FindLogID);
    m_FindLogID = IntPtr.Zero;
    btn_StartQuery.Text = "StartQuery(开始查询)";
    btn_NextLog.Enabled = false;
    btn_GetLogCount.Enabled = false;
    textBox_LogCount.Text = "";

    listView_Log.Items.Clear();
}
//获取记录条数
NET_IN_GETCOUNT_LOG_PARAM stuIn = new NET_IN_GETCOUNT_LOG_PARAM();
stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_GETCOUNT_LOG_PARAM));
NET_OUT_GETCOUNT_LOG_PARAM stuOut = new
NET_OUT_GETCOUNT_LOG_PARAM();
stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_GETCOUNT_LOG_PARAM));
if (NETClient.QueryDevLogCount(loginID, ref stuIn, ref stuOut, 5000))
{
    textBox_LogCount.Text = stuOut.nLogCount.ToString();
}
else

```

```

    {
        MessageBox.Show(NETClient.GetLastError());
    }
// 查询下一条记录
    listView_Log.Items.Clear();
    int max = 10;
    NET_IN_QUERYNEXTLOG stuIn = new NET_IN_QUERYNEXTLOG();
    stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_QUERYNEXTLOG));
    stuIn.nGetCount = max;

    NET_OUT_QUERYNEXTLOG stuOut = new NET_OUT_QUERYNEXTLOG();
    stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_QUERYNEXTLOG));
    stuOut.nMaxCount = max;
    stuOut.pstuLogInfo = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_LOG_INFO)) *
stuOut.nMaxCount);

    NET_LOG_INFO[] logInfo = new NET_LOG_INFO[stuOut.nMaxCount];
    for (int i = 0; i < stuOut.nMaxCount; i++)
    {
        logInfo[i].dwSize = (uint)Marshal.SizeOf(typeof(NET_LOG_INFO));
        logInfo[i].stuLogMsg.dwSize = (uint)Marshal.SizeOf(typeof(NET_LOG_MESSAGE));
        IntPtr pDst = IntPtr.Add(stuOut.pstuLogInfo,
Marshal.SizeOf(typeof(NET_LOG_INFO)) * i);
        Marshal.StructureToPtr(logInfo[i], pDst, true);
    }

    if (NETClient.QueryNextLog(m_FindLogID, ref stuIn, ref stuOut, 5000))
    {
        if (stuOut.nRetCount > 0)
        {
            BeginInvoke(new Action(() =>
            {
                for (int i = 0; i < stuOut.nRetCount; i++)
                {
                    IntPtr pDst = IntPtr.Add(stuOut.pstuLogInfo,
Marshal.SizeOf(typeof(NET_LOG_INFO)) * i);
                    NET_LOG_INFO retInfo =
(NET_LOG_INFO)Marshal.PtrToStructure(pDst, typeof(NET_LOG_INFO));

                    m_LogNum += 1;
                }
            }
            ));
        }
    }
}

```

```

        var listitem = new ListViewItem();
        listitem.Text = m_LogNum.ToString();
        listitem.SubItems.Add(retInfo.stuTime.ToString());
        listitem.SubItems.Add(retInfo.szUserName);
        listitem.SubItems.Add(retInfo.szLogType);
        listitem.SubItems.Add(retInfo.stuLogMsg.szLogMessage);
        if (listView_Log != null)
        {
            listView_Log.BeginUpdate();
            listView_Log.Items.Add(listitem);
            listView_Log.EndUpdate();
        }
    });

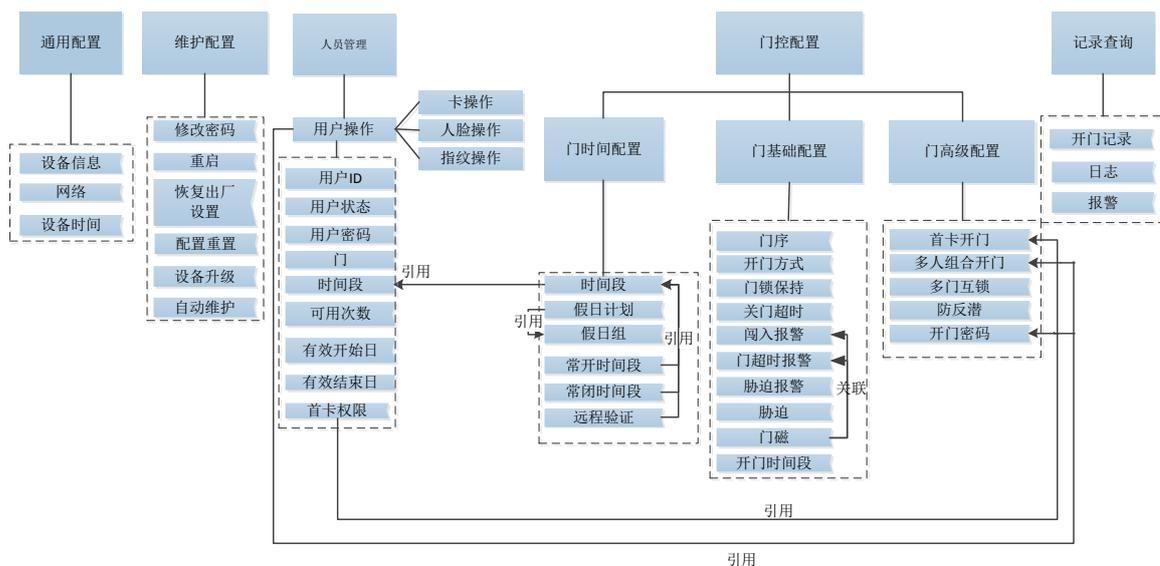
}
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}
}
}

```

2.3 门禁控制器/人脸一体机（二代）

门禁控制器/人脸一体机（二代）功能调用关系，如图 2-36 所示。

图2-36 功能调用关系



功能调用关系中引用和关联的含义如下：

- 引用：箭头终点指向的功能引用箭头起点指向的功能。
- 关联：箭头起始的功能是否能够正常使用，与箭头终点指向的功能配置相关。

2.3.1 门禁控制

请参见“2.2.1 门禁控制”。

2.3.2 报警事件

请参见“2.2.2 报警事件”。

2.3.3 设备信息查看

2.3.3.1 能力集查询

2.3.3.1.1 简介

设备信息查看，即用户通过 SDK 下发命令给门禁设备，来获取设备的能力集。

2.3.3.1.2 接口总览

表2-47 能力集查询接口说明

接口	说明
NETClient.GetDevCaps	获取门禁能力（包括门禁、用户、卡、人脸、指纹能力）

2.3.3.1.3 流程说明

图2-37 设备信息查看流程图



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetDevCaps 函数，nType 赋值为 NET_ACCESSCONTROL_CAPS，可以获取门禁能力。
- 步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.3.1.4 示例代码

```
NET_IN_AC_CAPS stuIn = new NET_IN_AC_CAPS();
stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_AC_CAPS));
NET_OUT_AC_CAPS stuOut = new NET_OUT_AC_CAPS();
stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_AC_CAPS));
stuOut.stuACCaps = new NET_AC_CAPS();
stuOut.stuUserCaps = new NET_ACCESS_USER_CAPS();
stuOut.stuCardCaps = new NET_ACCESS_CARD_CAPS();
stuOut.stuFingerprintCaps = new NET_ACCESS_FINGERPRINT_CAPS();
stuOut.stuFaceCaps = new NET_ACCESS_FACE_CAPS();

IntPtr ptrIn = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_AC_CAPS)));
IntPtr ptrOut = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_AC_CAPS)));
Marshal.StructureToPtr(stuIn, ptrIn, true);
```

```
Marshal.StructureToPtr(stuOut, ptrOut, true);
bool bRet = NETClient.GetDevCaps(m_LoginID, EM_DEVCAP_TYPE.ACCESSCONTROL_CAPS, ptrIn,
ptrOut, 5000);
if (bRet)
{
    stuOut = (NET_OUT_AC_CAPS)Marshal.PtrToStructure(ptrOut, typeof(NET_OUT_AC_CAPS));
    m_AccessCount = stuOut.stuACCaps.nChannels;
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}
```

2.3.3.2 设备版本、MAC 查看

请参见“2.2.3.2 设备版本、MAC 查看”。

2.3.4 网络设置

请参见“2.2.4 网络设置”。

2.3.5 设备时间设置

请参见“2.2.5 设备时间获取和设置”。

2.3.6 维护配置

请参见“2.2.6 维护配置”。

2.3.7 人员管理

2.3.7.1 用户管理

2.3.7.1.1 简介

用户通过调用 SDK，可以对门禁设备的用户信息字段（包含：用户 ID、人员名称、类型、状态、身份证号码、有效时间段、假日计划、权限等）进行增删查的操作。

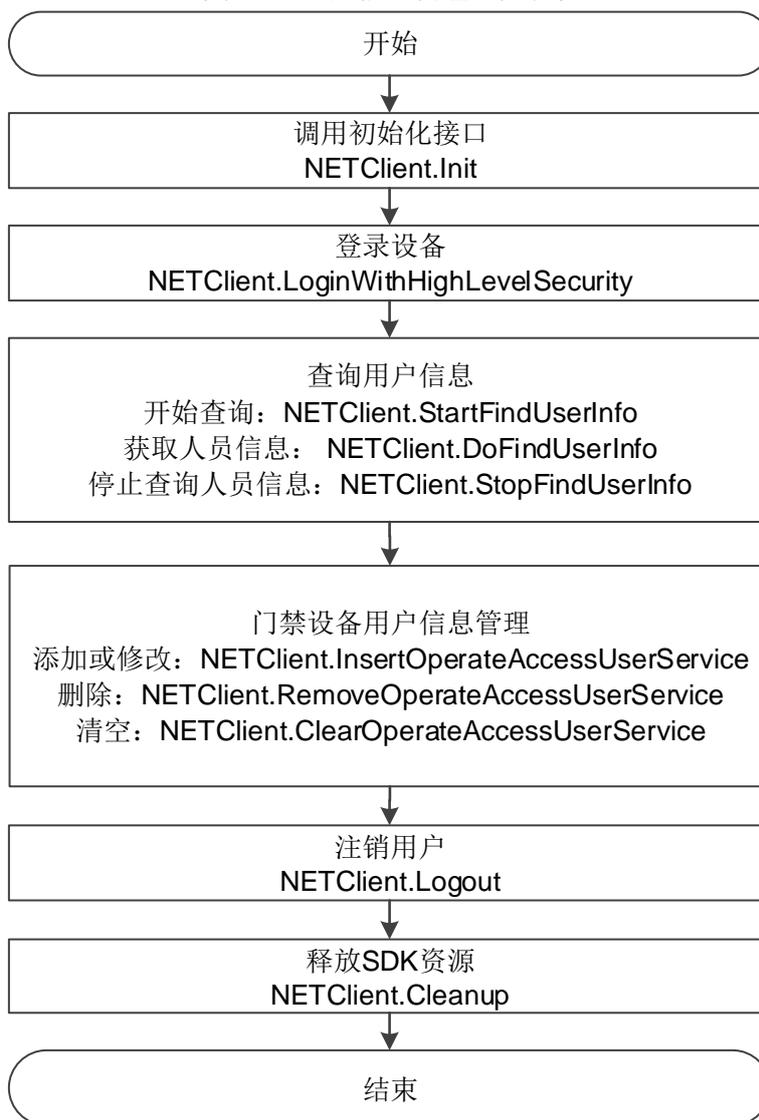
2.3.7.1.2 接口总览

表2-48 用户信息接口说明

接口	说明
NETClient.StartFindUserInfo	开始查询用户信息
NETClient.DoFindUserInfo	获取用户信息
NETClient.StopFindUserInfo	停止查询用户信息
NETClient.InsertOperateAccessUserService	门禁用户信息添加或修改
NETClient.RemoveOperateAccessUserService	门禁用户信息删除
NETClient.ClearOperateAccessUserService	门禁用户信息清空

2.3.7.1.3 流程说明

图2-38 用户信息管理业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.StartFindUserInfo 开始查询用户信息。
- 步骤4 调用 NETClient.DoFindUserInfo 获取用户信息。

- 步骤5 调用 NETClient.StopFindUserInfo 停止查询用户信息。
- 步骤6 调用 NETClient.InsertOperateAccessUserService 函数来对用户信息进行添加或修改操作、调用 NETClient.RemoveOperateAccessUserService 函数来对用户信息进行删除操作、调用 NETClient.ClearOperateAccessUserService 函数来对用户信息进行清空操作。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.7.1.4 示例代码

```
private IntPtr m_FindUserID = IntPtr.Zero;
private List<NET_ACCESS_USER_INFO> userInfoList = new List<NET_ACCESS_USER_INFO>();

//获取所有用户信息
NET_IN_USERINFO_START_FIND stuStartIn = new NET_IN_USERINFO_START_FIND();
stuStartIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_START_FIND));

NET_OUT_USERINFO_START_FIND stuStartOut = new NET_OUT_USERINFO_START_FIND();
stuStartOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_START_FIND));
stuStartOut.nTotalCount = 0;
stuStartOut.nCapNum = 50;
m_FindUserID = NETClient.StartFindUserInfo(m_LoginID, ref stuStartIn, ref stuStartOut, 5000);
if (IntPtr.Zero == m_FindUserID)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

userInfoList.Clear();

NET_IN_USERINFO_DO_FIND stuFindIn = new NET_IN_USERINFO_DO_FIND();
stuFindIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_DO_FIND));
stuFindIn.nCount = QueryNum;

NET_OUT_USERINFO_DO_FIND stuFindOut = new NET_OUT_USERINFO_DO_FIND();
stuFindOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_DO_FIND));
stuFindOut.nMaxNum = QueryNum;

NET_ACCESS_USER_INFO[] stuOutUserInfo = new NET_ACCESS_USER_INFO[stuFindOut.nMaxNum];
IntPtr outInfo = IntPtr.Zero;
outInfo = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)) *
stuFindOut.nMaxNum);
for (int index = 0; index < stuFindOut.nMaxNum; index++)
```

```

{
    IntPtr outInfoIndex = outInfo + index * Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO));
    if (stuOutUserInfo[index].GetType() == typeof(NET_ACCESS_USER_INFO))
//if obj is boxed type of typeName, some param(ex. dwsize) need trans to unmanaged memory
    {
        Marshal.StructureToPtr(stuOutUserInfo[index], outInfoIndex, true);
    }
    else
    {
        for (int i = 0; i < Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)); i++)
        {
            Marshal.WriteByte(outInfoIndex, i, 0);
        }
    }
}
stuFindOut.pstuInfo = outInfo;

int startNum = 0;
while (true)
{
    stuFindIn.nStartNo = startNum;

    bool result = NETClient.DoFindUserInfo(m_FindUserID, ref stuFindIn, ref stuFindOut, 5000);
    if (!result)
    {
        break;
    }

    if (stuFindOut.nRetNum > 0)
    {
        startNum += stuFindOut.nRetNum;
        for (int i = 0; i < stuFindOut.nRetNum; i++)
        {
            var userinfo =
(NET_ACCESS_USER_INFO)Marshal.PtrToStructure(IntPtr.Add(stuFindOut.pstuInfo,
Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)) * i), typeof(NET_ACCESS_USER_INFO));
            userInfoList.Add(userinfo);
        }
    }
}
}

```

```

NETClient.StopFindUserInfo(m_FindUserID);

//新增或修改
private NET_ACCESS_USER_INFO m_UserInfo = new NET_ACCESS_USER_INFO();
m_UserInfo.szUserID = txt_UserID.Text.Trim();
m_UserInfo.szName = txt_Name.Text.Trim();
m_UserInfo.szPsw = txt_Pwd.Text.Trim();
bool result = false;
NET_ACCESS_USER_INFO[] stuInArray = new NET_ACCESS_USER_INFO[1] { m_UserInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
result = NETClient.InsertOperateAccessUserService(m_LoginID, stuInArray, out stuOutErrArray, 5000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
//删除
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
string[] InUserid = new string[] { szUserID }; // szUserID: 要删除的用户的 szUserID
bool result = NETClient.RemoveOperateAccessUserService(m_LoginID, InUserid, out stuOutErrArray, 3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

```

2.3.7.2 卡片管理

2.3.7.2.1 简介

用户通过调用 SDK，可以对门禁设备的卡片信息字段（包含：卡号、用户 ID、卡类型等）进行增删查改的操作。

2.3.7.2.2 接口总览

表2-49 卡片信息接口说明

接口	说明
NETClient.StartFindCardInfo	开始查询卡片信息
NETClient.DoFindCardInfo	获取卡片信息
NETClient.StopFindCardInfo	停止查询卡片信息
NETClient.InsertOperateAccessCardService	门禁卡片信息添加
NETClient.RemoveOperateAccessCardService	门禁卡片信息删除
NETClient.UpdateOperateAccessCardService	门禁卡片信息更新

2.3.7.2.3 流程说明

图2-39 卡片信息管理业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.StartFindCardInfo 开始查询卡片信息。
- 步骤4 调用 NETClient.DoFindCardInfo 获取卡片信息。
- 步骤5 调用 NETClient.StopFindCardInfo 停止查询卡片信息。
- 步骤6 调用 NETClient.InsertOperateAccessCardService 函数来对卡片信息进行添加、调用 NETClient.UpdateOperateAccessCardService 函数来对卡片信息进行更新、调用 NETClient.RemoveOperateAccessCardService 函数来对卡片信息进行删除操作。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.7.2.4 示例代码

```
//获取
private List<NET_ACCESS_CARD_INFO> cardInfoList = new List<NET_ACCESS_CARD_INFO>();
NET_IN_CARDINFO_START_FIND stuStartIn = new NET_IN_CARDINFO_START_FIND();
stuStartIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_CARDINFO_START_FIND));
stuStartIn.szUserID = m_UserInfo.szUserID;

NET_OUT_CARDINFO_START_FIND stuStartOut = new NET_OUT_CARDINFO_START_FIND();
stuStartOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_CARDINFO_START_FIND));
stuStartOut.nTotalCount = 0;
stuStartOut.nCapNum = 10;

IntPtr cardFindId = NETClient.StartFindCardInfo(m_LoginID, ref stuStartIn, ref stuStartOut, 5000);
if (IntPtr.Zero != cardFindId)
{
    int nStartNo = 0;
    bool m_bIsDoFindNextCard = true;
    while (m_bIsDoFindNextCard)
    {
        int nRecordNum = 0;

        NET_IN_CARDINFO_DO_FIND stuFindIn = new NET_IN_CARDINFO_DO_FIND();
        stuFindIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_CARDINFO_DO_FIND));
        stuFindIn.nStartNo = nStartNo;
        stuFindIn.nCount = 10;

        NET_OUT_CARDINFO_DO_FIND stuFindOut = new NET_OUT_CARDINFO_DO_FIND();
        stuFindOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_CARDINFO_DO_FIND));
        stuFindOut.nMaxNum = 10;
        stuFindOut.pstulInfo =
        Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) * stuFindOut.nMaxNum);

        NET_ACCESS_CARD_INFO[] pCardInfo = new
        NET_ACCESS_CARD_INFO[stuFindOut.nMaxNum];
        for (int i = 0; i < stuFindOut.nMaxNum; i++)
        {
            IntPtr pDst = IntPtr.Add(stuFindOut.pstulInfo,
            Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) * i);
            Marshal.StructureToPtr(pCardInfo[i], pDst, true);
        }
    }
}
```

```

bool ret = NETClient.DoFindCardInfo(cardFindId, ref stuFindIn, ref stuFindOut, 5000);
if (ret)
{
    if (stuFindOut.nRetNum > 0)
    {
        nRecordNum = stuFindOut.nRetNum;
        for (int i = 0; i < nRecordNum; i++)
        {
            IntPtr pDst = IntPtr.Add(stuFindOut.pstuInfo,
Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) * i);
            NET_ACCESS_CARD_INFO stuInfo =
(NET_ACCESS_CARD_INFO)Marshal.PtrToStructure(pDst, typeof(NET_ACCESS_CARD_INFO));
            cardInfoList.Add(stuInfo);
        }
    }

    if (nRecordNum < 10)
    {
        break;
    }
    else
    {
        nStartNo += nRecordNum;
    }
}
else
{
    break;
}
}

NETClient.StopFindCardInfo(cardFindId);
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}
}

//新增

```

```

NET_ACCESS_CARD_INFO[] stuInArray = new NET_ACCESS_CARD_INFO[1] { m_CardInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
stuInArray[0].emType = (EM_ACCESSCTLCARD_TYPE)cmb_CardType.SelectedIndex;
stuInArray[0].szCardNo = txt_CardNum.Text;
result = NETClient.InsertOperateAccessCardService(m_LoginID, stuInArray, out stuOutErrArray, 5000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
//更新
NET_ACCESS_CARD_INFO[] stuInArray = new NET_ACCESS_CARD_INFO[1] { m_CardInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];

stuInArray[0].emType = (EM_ACCESSCTLCARD_TYPE)cmb_CardType.SelectedIndex;
result = NETClient.UpdateOperateAccessCardService(m_LoginID, stuInArray, out stuOutErrArray,
3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

//删除
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
string[] InCardid = new string[] { szCardNo };// szCardNo: 要删除的用户的 szCardNo
bool result = NETClient.RemoveOperateAccessCardService(m_LoginID, InCardid, out stuOutErrArray,
3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

```

2.3.7.3 人脸管理

2.3.7.3.1 简介

用户通过调用 SDK，可以对门禁设备的人脸信息字段（包含：用户 ID、人脸图片数据等）进行增删查改的操作。

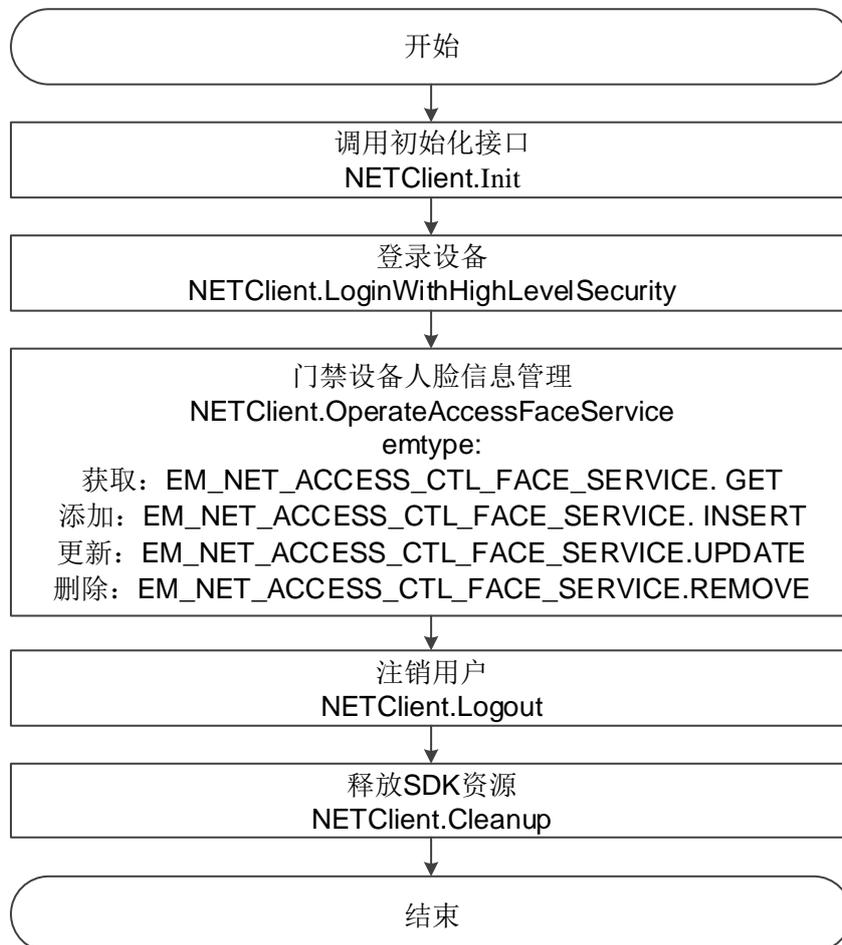
2.3.7.3.2 接口总览

表2-50 人脸信息接口说明

接口	说明
NETClient.OperateAccessFaceService	门禁人脸信息管理接口

2.3.7.3.3 流程说明

图2-40 人脸信息管理业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.OperateAccessFaceService 函数来对人脸信息进行添加、获取、更新、删除操作。其中参数 emtype 参考下表 EM_NET_ACCESS_CTL_FACE_SERVICE 枚举值。

表2-51 EM_NET_ACCESS_CTL_FACE_SERVICE 枚举值说明

参数	含义
INSERT	添加
GET	获取
UPDATE	更新
REMOVE	删除

步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.7.3.4 示例代码

```
//获取
NET_IN_ACCESS_FACE_SERVICE_GET stuFaceGetIn = new NET_IN_ACCESS_FACE_SERVICE_GET();
stuFaceGetIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_GET));
stuFaceGetIn.nUserNum = 1;
stuFaceGetIn.szUserID = new NET_IN_ACCESS_FACE_SERVICE_UserID[100];
stuFaceGetIn.szUserID[0] = new NET_IN_ACCESS_FACE_SERVICE_UserID() { userID =
m_UserInfo.szUserID };//m_UserInfo.szUserID;

NET_OUT_ACCESS_FACE_SERVICE_GET stuFaceGetOut = new
NET_OUT_ACCESS_FACE_SERVICE_GET();
stuFaceGetOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_GET));
stuFaceGetOut.nMaxRetNum = 1;
stuFaceGetOut.pFaceInfo = IntPtr.Zero;
stuFaceGetOut.pFaceInfo = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));
stuFaceGetOut.pFailCode = IntPtr.Zero;
stuFaceGetOut.pFailCode = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.pFacePhoto = new IntPtr[5];
for (int i = 0; i < 5; i++)
{
    stuFaceInfo.nInFacePhotoLen[i] = 100 * 1024;
    IntPtr tempPtr = IntPtr.Zero;
    tempPtr = Marshal.AllocHGlobal(100 * 1024);
    for (int j = 0; j < 100 * 1024; j++)
    {
        Marshal.WriteByte(tempPtr, j, 0);
    }
    stuFaceInfo.pFacePhoto[i] = tempPtr;
}
}
```

```

Marshal.StructureToPtr(stuFaceInfo, stuFaceGetOut.pFaceInfo, true);

NET_EM_FAILCODE stuFailCode = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCode, stuFaceGetOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_GET)));
Marshal.StructureToPtr(stuFaceGetIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_GET)));
Marshal.StructureToPtr(stuFaceGetOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.GET, pstInParam, pstOutParam, 5000);
var get_face_service = (NET_OUT_ACCESS_FACE_SERVICE_GET)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_GET));
if (!result)
{
    stuFailCode = (NET_EM_FAILCODE)Marshal.PtrToStructure(get_face_service.pFailCode,
typeof(NET_EM_FAILCODE));

    if (stuFailCode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else if (stuFailCode.emCode != EM_FAILCODE.UNKNOWN)
    {
        MessageBox.Show(stuFailCode.emCode.ToString());
    }
}
else
{
    stuFaceInfo = (NET_ACCESS_FACE_INFO)Marshal.PtrToStructure(get_face_service.pFaceInfo,
typeof(NET_ACCESS_FACE_INFO));
    if (stuFaceInfo.nFacePhoto > 0)
    {
        m_ImageData = new byte[stuFaceInfo.nOutFacePhotoLen[0]];
        Marshal.Copy(stuFaceInfo.pFacePhoto[0], m_ImageData, 0,
stuFaceInfo.nOutFacePhotoLen[0]);
    }
}

```

```

        using (MemoryStream stream = new MemoryStream(m_ImageData))
        {
            Image image = Image.FromStream(stream);
            pictureBox_face.Image = image;
            pictureBox_face.Refresh();
        }
    }
}
//添加
NET_IN_ACCESS_FACE_SERVICE_INSERT stuFaceInsertIn = new
NET_IN_ACCESS_FACE_SERVICE_INSERT();
stuFaceInsertIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_INSERT));
stuFaceInsertIn.nFaceInfoNum = 1;
stuFaceInsertIn.pFaceInfo = IntPtr.Zero;
stuFaceInsertIn.pFaceInfo = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();
stuFaceInfo.szUserID = m_UserInfo.szUserID;
stuFaceInfo.nFacePhoto = 1;
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.nOutFacePhotoLen = new int[5];
stuFaceInfo.nInFacePhotoLen[0] = stuFaceInfo.nOutFacePhotoLen[0] = m_ImageData.Length;
stuFaceInfo.pFacePhoto = new IntPtr[5];
stuFaceInfo.pFacePhoto[0] = Marshal.AllocHGlobal(stuFaceInfo.nInFacePhotoLen[0]);
Marshal.Copy(m_ImageData, 0, stuFaceInfo.pFacePhoto[0], stuFaceInfo.nInFacePhotoLen[0]);

Marshal.StructureToPtr(stuFaceInfo, stuFaceInsertIn.pFaceInfo, true);

NET_OUT_ACCESS_FACE_SERVICE_INSERT stuFaceInsertOut = new
NET_OUT_ACCESS_FACE_SERVICE_INSERT();
stuFaceInsertOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT));
stuFaceInsertOut.nMaxRetNum = 1;
stuFaceInsertOut.pFailCode = IntPtr.Zero;
stuFaceInsertOut.pFailCode = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceInsertOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =

```

```

Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_INSERT)));
Marshal.StructureToPtr(stuFaceInsertIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT)));
Marshal.StructureToPtr(stuFaceInsertOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.INSERT, pstInParam, pstOutParam, 5000);
var faceinfo = (NET_OUT_ACCESS_FACE_SERVICE_INSERT)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT));
if (!result)
{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else
    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

//更新
NET_IN_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateIn = new
NET_IN_ACCESS_FACE_SERVICE_UPDATE();
stuFaceUpdateIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_UPDATE));
stuFaceUpdateIn.nFaceInfoNum = 1;
stuFaceUpdateIn.pFaceInfo = IntPtr.Zero;
stuFaceUpdateIn.pFaceInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();
stuFaceInfo.szUserID = m_UserInfo.szUserID;
stuFaceInfo.nFacePhoto = 1;
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.nOutFacePhotoLen = new int[5];
stuFaceInfo.nInFacePhotoLen[0] = stuFaceInfo.nOutFacePhotoLen[0] = m_ImageData.Length;

```

```

stuFaceInfo.pFacePhoto = new IntPtr[5];
stuFaceInfo.pFacePhoto[0] = Marshal.AllocHGlobal(stuFaceInfo.nInFacePhotoLen[0]);
Marshal.Copy(m_ImageData, 0, stuFaceInfo.pFacePhoto[0], stuFaceInfo.nInFacePhotoLen[0]);

Marshal.StructureToPtr(stuFaceInfo, stuFaceUpdateIn.pFaceInfo, true);

NET_OUT_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateOut = new
NET_OUT_ACCESS_FACE_SERVICE_UPDATE(); //{ sizeof(stuFaceUpdateOut) };
stuFaceUpdateOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDATE));
stuFaceUpdateOut.nMaxRetNum = 1;
stuFaceUpdateOut.pFailCode = IntPtr.Zero;
stuFaceUpdateOut.pFailCode = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceUpdateOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_UPDATE)));
Marshal.StructureToPtr(stuFaceUpdateIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDATE)));
Marshal.StructureToPtr(stuFaceUpdateOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.UPDATE, pstInParam, pstOutParam, 5000);
var faceinfo = (NET_OUT_ACCESS_FACE_SERVICE_UPDATE)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDATE));

if (!result)
{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else

```

```

    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

//删除
NET_IN_ACCESS_FACE_SERVICE_REMOVE stuFaceRemoveIn = new
NET_IN_ACCESS_FACE_SERVICE_REMOVE();
stuFaceRemoveIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_REMOVE));
stuFaceRemoveIn.nUserNum = 1;
stuFaceRemoveIn.szUserID = new NET_IN_ACCESS_FACE_SERVICE_UserID[100];
stuFaceRemoveIn.szUserID[0] = new NET_IN_ACCESS_FACE_SERVICE_UserID() { userID =
m_UserInfo.szUserID };

NET_OUT_ACCESS_FACE_SERVICE_REMOVE stuFaceRemoveOut = new
NET_OUT_ACCESS_FACE_SERVICE_REMOVE();//{ sizeof(stuFaceROut) };
stuFaceRemoveOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE));
stuFaceRemoveOut.nMaxRetNum = 1;
stuFaceRemoveOut.pFailCode = IntPtr.Zero;
stuFaceRemoveOut.pFailCode = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceRemoveOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_REMOVE)));
Marshal.StructureToPtr(stuFaceRemoveIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE)));
Marshal.StructureToPtr(stuFaceRemoveOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.REMOVE, pstInParam, pstOutParam, 5000);
var faceinfo = (NET_OUT_ACCESS_FACE_SERVICE_REMOVE)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE));

if (!result)

```

```

{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else
    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

```

2.3.7.4 指纹管理

2.3.7.4.1 简介

用户通过调用 SDK，可以对门禁设备的指纹信息字段（包含：用户 ID、指纹数据包、胁迫指纹序号等）进行增删查改的操作。

2.3.7.4.2 接口总览

表2-52 指纹信息接口说明

接口	说明
NETClient.OperateAccessFingerprintService	指纹信息管理接口

2.3.7.4.3 流程说明

图2-41 指纹信息管理业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.OperateAccessFingerprintService 函数来对指纹信息进行添加、获取、更新、删除、清空操作。其中参数 emtype 参考下表 EM_ACCESS_CTL_FINGERPRINT_SERVICE 枚举值。

表2-53 type 对应操作和结构体

emtype	含义	Param
INSERT	添加指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT
GET	获取指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_GET NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET
UPDATE	更新指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE
REMOVE	删除指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE
CLEAR	清空指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR

步骤4 业务执行完成之后，调用 NETClient.Logout 函数登出设备。

步骤5 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.7.4.4 示例代码

```
private NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET m_FingerprintInfo = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET();
//获取
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET();
stuFingerPrintGetIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FINGERPRINT_SERVICE_GET));
stuFingerPrintGetIn.szUserID = m_UserInfo.szUserID;

NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET();
stuFingerPrintGetOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET));
stuFingerPrintGetOut.nMaxFingerDataLength = 10000;
stuFingerPrintGetOut.pbyFingerData = IntPtr.Zero;
stuFingerPrintGetOut.pbyFingerData = Marshal.AllocHGlobal(10000);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FINGERPRINT_SERVICE_GET)));
Marshal.StructureToPtr(stuFingerPrintGetIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET)));
Marshal.StructureToPtr(stuFingerPrintGetOut, pstOutParam, true);

bool result = NETClient.OperateAccessFingerprintService(m_LoginID,
EM_ACCESS_CTL_FINGERPRINT_SERVICE.GET, pstInParam, pstOutParam, 5000);
m_FingerprintInfo =
(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET));

//新增
m_FingerprintInfo.nPacketNum = 1;
m_FingerprintInfo.nPacketLen = m_PacketLen;
m_FingerprintInfo.szFingerPrintInfo = Marshal.AllocHGlobal(m_PacketLen);
for (int i = 0; i < m_PacketLen; i++)
```

```

{
    Marshal.WriteByte(m_FingerprintInfo.szFingerPrintInfo, i, FingerPrintInfo[i]);
}
m_FingerprintInfo.nDuressIndex = 0;
if (cckb_Duress.Checked)
{
    m_FingerprintInfo.nDuressIndex = 1;
}

NET_ACCESS_FINGERPRINT_INFO[] stuInArray = new NET_ACCESS_FINGERPRINT_INFO[1]
{ m_FingerprintInfo };
NET_EM_FAILCODE[] stuOutArray;

bRet = NETClient.InsertOperateAccessFingerprintService(m_LoginID, stuInArray, out stuOutArray,
3000);
if (!bRet)
{
    for (int i = 0; i < stuOutArray.Length; i++)
    {
        MessageBox.Show(stuOutArray[i].emCode.ToString());
    }
}

//更新
for (int i = 0; i < m_PacketLen; i++)
{
    Marshal.WriteByte(m_FingerprintInfo.szFingerPrintInfo, (m_FingerprintNum - 1) * m_PacketLen
+ i, FingerPrintInfo[i]);
}
if (cckb_Duress.Checked)
{
    m_FingerprintInfo.nDuressIndex = m_FingerprintNum;
}

NET_ACCESS_FINGERPRINT_INFO[] stuInArray = new NET_ACCESS_FINGERPRINT_INFO[1]
{ m_FingerprintInfo };
NET_EM_FAILCODE[] stuOutArray;

bRet = NETClient.UpdateOperateAccessFingerprintService(m_LoginID, stuInArray, out stuOutArray,
3000);

```

```
if (!bRet)
{
    for (int i = 0; i < stuOutArray.Length; i++)
    {
        MessageBox.Show(stuOutArray[i].emCode.ToString());
    }
}
//删除
NET_EM_FAILCODE[] stuOutErrArray;
string[] userid = new string[] { m_UserInfo.szUserID };
result = NETClient.RemoveOperateAccessFingerprintService(m_LoginID, userid, out stuOutErrArray,
3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
```

2.3.8 门配置

请参见“2.2.8 门配置”。

2.3.9 门时间管理

2.3.9.1 时段配置

请参见“2.2.9.1 时段配置”。

2.3.9.2 常开常闭时间段配置

请参见“2.2.9.2 常开常闭时间段配置”。

2.3.9.3 假日组

2.3.9.3.1 简介

假日组，即用户通过调用 SDK 接口对设备假日组进行配置，包含假日组名称、开始结束时间、组使能等信息。

2.3.9.3.2 接口总览

表2-54 假日组接口说明

接口	说明
NETClient.GetOperateConfig	查询配置信息
NETClient.SetOperateConfig	设置配置信息

2.3.9.3.3 流程说明

图2-42 假日组业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetOperateConfig 函数来查询门禁假日组配置信息。

表2-55 cfg_type 参数赋值说明

emCfgOpType	描述	对应结构体
EM_CFG_OPERATE_TY PE.SPECIALDAY_GROU P	获取假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

- 步骤4 调用 NETClient.SetOperateConfig 函数来设置门禁假日组配置信息。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.9.3.4 示例代码

```
// 获取
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuln = new
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO();
stuln.dwSize = (uint)Marshal.SizeOf(typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));
object obj = stuln;
bool ret = NETClient.OperateConfig(m_LoginID, EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP,
cmb_Index.SelectedIndex, ref obj, typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
m_SpecialdayGroupInfo = (NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO)obj;

//设置
m_SpecialdayGroupInfo.bGroupEnable = chb_Enable.Checked;
m_SpecialdayGroupInfo.szGroupName = txt_Name.Text;
object obj = m_SpecialdayGroupInfo;
bool ret = NETClient.SetOperateConfig(m_LoginID, EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP,
cmb_Index.SelectedIndex, obj, typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
```

2.3.9.4 假日计划

2.3.9.4.1 简介

假日计划，即用户通过调用 SDK 接口对设备假日计划进行配置，包含假日计划名称、使能、时间段、有效的门通道等信息。

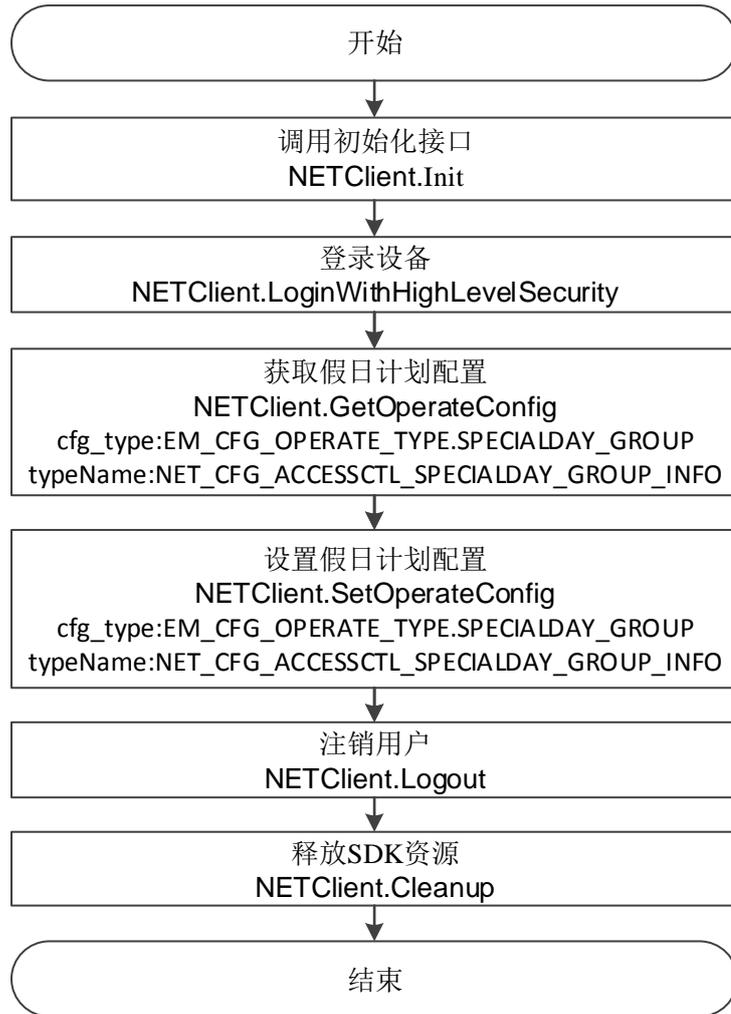
2.3.9.4.2 接口总览

表2-56 假日计划接口说明

接口	说明
NETClient.OperateConfig	查询配置信息
NETClient.SetOperateConfig	设置配置信息

2.3.9.4.3 流程说明

图2-43 假日计划业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 调用 NETClient.GetOperateConfig 函数来查询门禁假日计划配置信息。

表2-57 cfg_type 参数赋值说明

emCfgOpType	描述	对应结构体
EM_CFG_OPERATE_TY PE.SPECIALDAYS_SCHE DULE	获取假日计划信息	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE _INFO

- 步骤4 调用 NETClient.SetOperateConfig 函数来设置门禁假日计划配置信息。
- 步骤5 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤6 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.9.4.4 示例代码

```

// 获取
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuln = new
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO();
  
```

```

stuln.dwSize = (uint)Marshal.SizeOf(typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
object obj = stuln;
bool ret = NETClient.OperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE, cmb_ScheduleGroup.SelectedIndex, ref obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
m_ScheduleInfo = (NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO)obj;

//设置
m_ScheduleInfo.szSchduleName = txt_ScheduleName.Text;
m_ScheduleInfo.bSchdule = chb_ScheduleEnable.Checked;
m_ScheduleInfo.nGroupNo = int.Parse(txt_GroupNum.Text);

object obj = m_ScheduleInfo;
bool ret = NETClient.SetOperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE, cmb_ScheduleGroup.SelectedIndex, obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}

```

2.3.10 门高级配置

请参见“2.2.10 门高级配置”。

2.3.11 记录查询

2.3.11.1 开门记录

请参见“2.2.11.1 开门记录”。

2.3.11.2 设备日志

请参见“2.2.11.2 设备日志”。

2.3.11.3 报警记录

2.3.11.3.1 简介

报警记录查询，即用户调用 SDK 接口对门禁设备的报警记录进行查询。

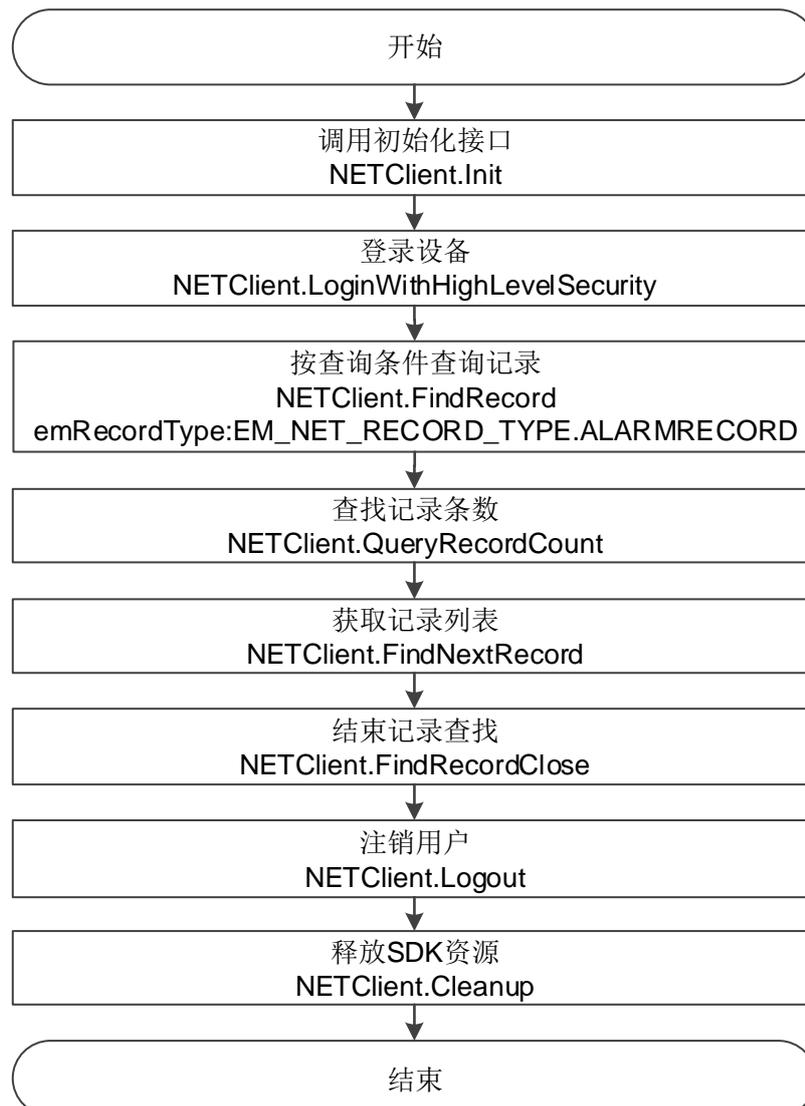
2.3.11.3.2 接口总览

表2-58 记录查询接口说明

接口	说明
NETClient.QueryRecordCount	查找记录条数
NETClient.FindRecord	按查询条件查询记录
NETClient.FindNextRecord	查找记录。nFilecount:需要查询的条数，出参中 nRetRecordNum 为查询到的记录条数，当查询到的条数小于 nFilecount 时查询结束
NETClient.FindRecordClose	结束记录查询

2.3.11.3.3 流程说明

图2-44 记录查询业务流程



流程说明

- 步骤1 调用 NETClient.Init 函数，完成 SDK 初始化流程。
- 步骤2 调用 NETClient.LoginWithHighLevelSecurity 函数登录设备。
- 步骤3 先调用 NETClient.FindRecord 函数获得查询句柄。参数 emRecordType 赋值为 EM_NET_RECORD_TYPE.ALARMRECORD。
- 步骤4 再调用 NETClient.QueryRecordCount 函数查找记录条数。
- 步骤5 再调用 NETClient.FindNextRecord 函数获取记录列表。
- 步骤6 查询完毕可以调用 NETClient.FindRecordClose 关闭查询句柄。
- 步骤7 业务执行完成之后，调用 NETClient.Logout 函数登出设备。
- 步骤8 SDK 功能使用完后，调用 NETClient.Cleanup 函数释放 SDK 资源。

2.3.11.3.4 示例代码

```
//开始查询
NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION condition = new
NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION();
condition.dwSize =
(uint)Marshal.SizeOf(typeof(NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION));
condition.stStartTime = NET_TIME.FromDateTime(dateTimePicker_DoorStart.Value);
condition.stEndTime = NET_TIME.FromDateTime(dateTimePicker_DoorEnd.Value);
object obj = condition;

bool ret = NETClient.FindRecord(m_LoginID, EM_NET_RECORD_TYPE.ACCESS_ALARMRECORD, obj,
typeof(NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION), ref
m_FindAlarmRecordID, 10000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

//查询记录数
int nCount = 0;
if (NETClient.QueryRecordCount(m_FindAlarmRecordID, ref nCount, 3000))
{
    txt_AlarmRecordCount.Text = nCount.ToString();
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}

//查找记录
```

```

int max = 20;
int retNum = 0;
List<object> ls = new List<object>();
for (int i = 0; i < max; i++)
{
    NET_RECORD_ACCESS_ALARMRECORD_INFO alarm_rec = new
NET_RECORD_ACCESS_ALARMRECORD_INFO();
    alarm_rec.dwSize = (uint)Marshal.SizeOf(typeof(NET_RECORD_ACCESS_ALARMRECORD_INFO));
    ls.Add(alarm_rec);
}
NETClient.FindNextRecord(m_FindAlarmRecordID, max, ref retNum, ref ls,
typeof(NET_RECORD_ACCESS_ALARMRECORD_INFO), 10000);
foreach (var item in ls)
{
    NET_RECORD_ACCESS_ALARMRECORD_INFO info =
(NET_RECORD_ACCESS_ALARMRECORD_INFO)item;
}
//结束记录查找
NETClient.FindRecordClose(m_FindAlarmRecordID);
m_FindAlarmRecordID = IntPtr.Zero;

```

第 3 章 接口函数

3.1 通用接口

3.1.1 SDK 初始化

3.1.1.1 SDK 初始化 Init

表3-1 NetSDK 初始化 Init 说明

选项	说明	
描述	对整个 NetSDK 进行初始化	
函数	bool Init(fDisconnectCallBack cbDisconnect, IntPtr dwUser, NETSDK_INIT_PARAM? stulnitParam);	
参数	[in]cbDisconnect	断线回调函数
	[in]dwUser	断线回调函数的用户参数
	[in]stulnitParam	NetSDK 初始化参数
返回值	失败返回 false, 成功返回 true	
说明	<ul style="list-style-type: none">● 调用网络 NetSDK 其他函数的前提● 回调函数设置成 null 时, 设备断线后不会回调给用户● Init 传入的 dwUser 参数, 会在回调函数 cbDisconnect 内的同字段 dwUser 返回。这样便于客户定位。其他函数同理。	

3.1.1.2 SDK 清理 Cleanup

表3-2 NetSDK 清理 Cleanup 说明

选项	说明
描述	清理 NetSDK
函数	void Cleanup()
参数	无
返回值	无
说明	NetSDK 清理接口, 在结束前最后调用

3.1.1.3 设置断线重连回调函数 SetAutoReconnect

表3-3 设置断线重连回调函数 SetAutoReconnect 说明

选项	说明
描述	设置自动重连回调函数

选项	说明	
函数	void SetAutoReconnect(fHaveReConnectCallBack cbAutoConnect, IntPtr dwUser);	
参数	[in]cbAutoConnect	断线重连回调函数
	[in]dwUser	断线重连回调函数的用户参数
返回值	无	
说明	设置断线重连回调接口。如果回调函数设置为 null，则不自动重连	

3.1.1.4 设置网络参数 SetNetworkParam

表3-4 设置网络参数 SetNetworkParam 说明

选项	说明	
描述	设置网络环境相关参数	
函数	void SetNetworkParam(NET_PARAM? netParam);	
参数	[in]netParam	网络延迟、重连次数、缓存大小等参数
返回值	无	
说明	可根据实际网络环境，调整参数	

3.1.2 设备初始化

3.1.2.1 搜索设备 StartSearchDevices

表3-5 搜索设备 StartSearchDevices 说明

选项	说明	
描述	搜索设备信息	
函数	IntPtr StartSearchDevice(fSearchDevicesCB cbSearchDevice, IntPtr pUserData, IntPtr szLocalIp)	
参数	[in] cbSearchDevice	异步搜索设备毁掉函数
	[in] pUserData	用户数据
	[in] szLocalIp	在单网卡的情况下，szLocalIp 可不填 在多网卡的情况下，szLocalIp 填主机 IP
返回值	失败返回 0，成功返回非 0 的值	
说明	不支持多线程调用	

3.1.2.2 设备初始化 InitDevAccount

表3-6 设备初始化说明

选项	说明
描述	初始化设备

选项	说明	
函数	<pre>bool InitDevAccount(NET_IN_INIT_DEVICE_ACCOUNT pInitAccountIn, ref NET_OUT_INIT_DEVICE_ACCOUNT pInitAccountOut, uint dwWaitTime, string szLocalIp)</pre>	
参数	[in]pInitAccountIn	输入参数，对应 NET_IN_INIT_DEVICE_ACCOUNT 结构体
	[out]pInitAccountOut	输出参数，对应 NET_OUT_INIT_DEVICE_ACCOUNT 结构体
	[in]dwWaitTime	超时时间
	[in]szLocalIp	<ul style="list-style-type: none"> 在单网卡的情况下，szLocalIp 可不填 在多网卡的情况下，szLocalIp 填主机 IP
返回值	失败返回 false，成功返回 true	
说明	无	

3.1.2.3 停止搜索设备 StopSearchDevices

表3-7 停止搜索设备说明

选项	说明	
描述	停止搜索设备信息	
函数	<pre>bool StopSearchDevice(IntPtr ISearchHandle)</pre>	
参数	[in] ISearchHandle	输入参数，搜索句柄
返回值	失败返回 false，成功返回 true	
说明	不支持多线程调用	

3.1.3 设备登录

3.1.3.1 用户登录设备 LoginWithHighLevelSecurity

表3-8 用户登录设备 LoginWithHighLevelSecurity 说明

选项	说明	
描述	用户登录设备	
函数	<pre>IntPtr LoginWithHighLevelSecurity(string pchDVRIP, ushort wDVRPort, string pchUserName, string pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, IntPtr pCapParam, ref NET_DEVICEINFO_Ex deviceInfo);</pre>	
参数	[in]pchDVRIP	设备 IP
	[in]wDVRPort	设备端口

选项	说明	
	[in]pchUserName	用户名
	[in]pchPassword	密码
	[in]emSpecCap	登录类别
	[in]pCapParam	登录类别参数
	[out]deviceInfo	设备信息
返回值	失败返回 0，成功返回非 0 的值	
说明	高安全级别登录接口。  说明 LoginEx2 仍然可以使用，但存在安全风险，所以强烈推荐使用最新接口 LoginWithHighLevelSecurity 登录设备。	

3.1.3.2 用户登出设备 Logout

表3-9 用户登出设备 Logout 说明

选项	说明	
描述	用户登出设备	
函数	<pre>bool Logout(IntPtr ILoginID);</pre>	
参数	[in]ILoginID	LoginWithHighLevelSecurity 的返回值
返回值	失败返回 false，成功返回 true	
说明	无	

3.1.4 实时监视

3.1.4.1 打开监视 RealPlay

表3-10 打开监视 RealPlay 说明

选项	说明	
描述	打开实时监视	
函数	<pre>IntPtr RealPlay(IntPtr ILoginID, int nChannelID, IntPtr hWnd, EM_RealPlayType rType = EM_RealPlayType.Realplay);</pre>	
参数	[in]ILoginID	LoginWithHighLevelSecurity 的返回值
	[in]nChannelID	视频通道号，从 0 开始递增的整数
	[in]hWnd	窗口句柄，仅在 Windows 系统下有效
	[in]rType	预览类型
返回值	失败返回 0，成功返回非 0 的值	

选项	说明
说明	在 Windows 环境下： <ul style="list-style-type: none"> • hWnd 为有效值时，在对应窗口显示画面 • hWnd 为 NULL 时，表示取流方式，通过设置回调函数来获取视频数据，交由用户处理

预览类型及含义，请参见表 3-11。

表3-11 预览类型说明

预览类型	含义
Realplay	实时预览
Multiply	多画面预览
Realplay_0	实时监视-主码流，等同于 DH_RType_Realplay
Realplay_1	实时监视-辅码流 1
Realplay_2	实时监视-辅码流 2
Realplay_3	实时监视-辅码流 3
Multiply_1	多画面预览—1 画面
Multiply_4	多画面预览—4 画面
Multiply_8	多画面预览—8 画面
Multiply_9	多画面预览—9 画面
Multiply_16	多画面预览—16 画面
Multiply_6	多画面预览—6 画面
Multiply_12	多画面预览—12 画面
Multiply_25	多画面预览—25 画面
Multiply_36	多画面预览—36 画面

3.1.4.2 关闭监视 StopRealPlay

表3-12 关闭监视 StopRealPlay 说明

选项	说明
描述	关闭实时监视
函数	bool StopRealPlay(IntPtr IRealHandle);
参数	[in]IRealHandle RealPlay 的返回值
返回值	失败返回 false，成功返回 true
说明	无

3.1.4.3 保存监视数据 SaveRealData

表3-13 保存监视数据 SaveRealData 说明

选项	说明
描述	保存实时监视数据为文件
函数	bool SaveRealData(IntPtr IRealHandle, string pchFileName);

选项	说明	
参数	[in]IRealHandle	RealPlay 的返回值
	[in]pchFileName	需要保存的文件路径
返回值	失败返回 false, 成功返回 true	
说明	无	

3.1.4.4 停止保存监视数据 StopSaveRealData

表3-14 停止保存监视数据 StopSaveRealData 说明

选项	说明	
描述	停止保存实时监视数据为文件	
函数	<pre>bool StopSaveRealData(IntPtr IRealHandle);</pre>	
参数	[in]IRealHandle	RealPlay 的返回值
返回值	失败返回 false, 成功返回 true	
说明	无	

3.1.4.5 设置监视数据回调 SetRealDataCallBack

表3-15 设置监视数据回调 SetRealDataCallBack 说明

选项	说明	
描述	设置实时监视数据回调	
函数	<pre>bool SetRealDataCallBack(IntPtr IRealHandle, fRealDataCallBackEx2 cbRealData, IntPtr dwUser, EM_REALDATA_FLAG dwFlag);</pre>	
参数	[in]IRealHandle	RealPlay 的返回值
	[in]cbRealData	监视数据流回调函数
	[in]dwUser	监视数据流回调函数的参数
	[in]dwFlag	回调中监视数据的类型
返回值	失败返回 false, 成功返回 true	
说明	无	

表3-16 dwFlag 类型及含义

dwFlag	含义
0x00000001	设备的原始数据
0x00000004	转成 YUV 格式的数据

3.1.5 设备控制

3.1.5.1 设备控制 ControlDevice

表3-17 设备控制说明

选项	说明	
描述	设备控制	
函数	<pre>bool ControlDevice(IntPtr ILoginID, EM_CtrlType type, IntPtr param, int waittime)</pre>	
参数	[in]ILoginID	LoginWithHighLevelSecurity 的返回值
	[in]type	控制类型
	[in]param	输入参数，因 emType 不同而不同
	[in]waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	失败返回 false，成功返回 true	
说明	无	

3.1.6 报警侦听

3.1.6.1 设置报警回调函数 SetDVRMessCallBack

表3-18 设置报警回调函数 SetDVRMessCallBack 说明

选项	说明	
描述	设置报警回调函数	
函数	<pre>void SetDVRMessCallBack(fMessCallBackEx cbMessage, IntPtr dwUser);</pre>	
参数	[in]cbMessage	<ul style="list-style-type: none"> 消息回调函数，可以回调设备的状态，如报警状态 当设置为 0 时表示禁止回调
	[in]dwUser	用户自定义数据
返回值	无	
说明	<ul style="list-style-type: none"> 设置设备消息回调函数，用来得到设备当前状态信息，与调用顺序无关，NetSDK 默认不回调 此回调函数 fMessCallBack 必须先调用报警消息订阅接口 StartListen 才生效 	

3.1.6.2 订阅报警 StartListen

表3-19 订阅报警 StartListen 说明

选项	说明
描述	订阅报警

选项	说明	
函数	bool StartListen(IntPtr ILoginID);	
参数	[in]ILoginID	LoginWithHighLevelSecurity 返回值
返回值	失败返回 false, 成功返回 true	
说明	订阅设备消息, 得到的消息从 SetDVRMessCallBack 的设置值回调出来	

3.1.6.3 停止订阅报警 StopListen

表3-20 停止订阅报警 StopListen 说明

选项	说明	
描述	停止订阅报警	
函数	bool StopListen(IntPtr ILoginID);	
参数	[in]ILoginID	LoginWithHighLevelSecurity 返回值
返回值	失败返回 false, 成功返回 true	
说明	无	

3.1.7 获取设备状态

3.1.7.1 获取设备状态 QueryDevState

表3-21 获取设备状态 QueryDevState 说明

选项	说明	
描述	直接获取远程设备连接状态	
函数	bool QueryDevState(IntPtr ILoginID, int nType, ref object obj, Type typeName, int waittime);	
参数	[in]ILoginID	LoginWithHighLevelSecurity 返回值
	[in]nType	查询信息类型, 当获取远程设备连接状态时 nType 为 EM_DEVICE_STATE.VIRTUALCAMERA
	[out]obj	用于接收查询返回的数据的缓存, 对应 NET_VIRTUALCAMERA_STATE_INFO 结构体
	[in]typeName	查询结构体的类型
	[in]waittime	查询状态等待时间
返回值	失败返回 false, 成功返回 true	
说明	无	

3.1.8 语音对讲

3.1.8.1 设置设备语音对讲工作模式 SetDeviceMode

表3-22 设置工作模式 SetDeviceMode 说明

选项	说明	
描述	设置工作模式	
函数	<pre>bool SetDeviceMode(IntPtr lLoginID, EM_USEDEV_MODE emType, IntPtr pValue);</pre>	
参数	[in] lLoginID	LoginWithHighLevelSecurity 返回值
	[in]emType	工作模式枚举
	[in]pValue	相应工作模式对应的结构体
返回值	失败返回 false, 成功返回 true	
说明	无	

表3-23 工作模式枚举及结构体对照表

emType 枚举	含义	结构体
EM_USEDEV_MODE.TALK_ENCODE_TYPE	指定某种格式进行对讲	NET_DEV_TALKDECODE_INFO
EM_USEDEV_MODE.TALK_CLIENT_MODE	设置语音对讲客户端方式	无
EM_USEDEV_MODE.TALK_SPEAK_PARAM	设置语音对讲喊话参数	NET_SPEAK_PARAM
EM_USEDEV_MODE.TALK_MODE3	设置三代设备的语音对讲参数	NET_TALK_EX

3.1.8.2 开启对讲 StartTalk

表3-24 开启对讲 StartTalk 说明

选项	说明	
描述	打开语音对讲	
函数	<pre>IntPtr StartTalk(IntPtr lLoginID, fAudioDataCallBack pfcb, IntPtr dwUser);</pre>	
参数	[in]lLoginID	LoginWithHighLevelSecurity 的返回值
	[in]pfcb	音频数据回调函数
	[in]dwUser	音频数据回调函数的参数
返回值	失败返回 0, 成功返回非 0 的值	
说明	无	

3.1.8.3 关闭对讲 StopTalk

表3-25 关闭对讲 StopTalk 说明

选项	说明
描述	关闭语音对讲
函数	bool StopTalk(IntPtr ITalkHandle);
参数	[in]ITalkHandle StartTalk 的返回值
返回值	失败返回 false, 成功返回 true
说明	无

3.1.8.4 开启录音 RecordStart

表3-26 开启录音 RecordStart 说明

选项	说明
描述	开启本地录音
函数	bool RecordStart(IntPtr ILoginID);
参数	[in]ILoginID LoginWithHighLevelSecurity 的返回值
返回值	失败返回 false, 成功返回 true
说明	此接口只在 Windows 下有效

3.1.8.5 关闭录音 RecordStop

表3-27 关闭录音 RecordStop 说明

选项	说明
描述	关闭本地录音
函数	bool RecordStop(IntPtr ILoginID);
参数	[in]ILoginID LoginWithHighLevelSecurity 的返回值
返回值	失败返回 false, 成功返回 true
说明	此接口只在 Windows 下有效

3.1.8.6 发送语音 TalkSendData

表3-28 发送语音 TalkSendData 说明

选项	说明
描述	发送音频数据给设备
函数	int TalkSendData(IntPtr ITalkHandle, IntPtr pSendBuf, uint dwBufSize);

参数	[in]ITalkHandle	StartTalk 的返回值
	[in]pSendBuf	需要发送的音频数据块的指针
	[in]dwBufSize	需要发送的音频数据块的长度，单位：字节
返回值	成功返回音频数据块的长度，失败返回-1	
说明	无	

3.1.8.7 解码语音 AudioDec

表3-29 解码语音 AudioDec 说明

选项	说明	
描述	解码音频数据	
函数	void AudioDec(IntPtr pAudioDataBuf, uint dwBufSize);	
参数	[in]pAudioDataBuf	需要解码的音频数据块的指针
	[in]dwBufSize	需要解码的音频数据块的长度，单位：字节
返回值	无	
说明	无	

3.2 门禁控制器/指纹一体机（一代）

3.2.1 门禁控制

关于门控制接口的详细介绍，请参见“3.1.5.1 设备控制 ControlDevice”。

关于门磁状态接口的详细介绍，请参见“3.2.3.3 查询设备状态 QueryDevState”。

3.2.2 报警事件

请参见“3.1.6 报警侦听”。

3.2.3 设备信息查看

3.2.3.1 查询系统能力信息 QueryNewSystemInfo

表3-30 查询系统能力信息说明

选项	说明
描述	查询系统能力信息

选项	说明	
函数	<pre>bool QueryNewSystemInfo(IntPtr ILoginID, Int32 IChannel, string strCommand, ref object obj, Type typeName, int waittime)</pre>	
参数	[in] ILoginID	LoginWithHighLevelSecurity 的返回值
	[in] IChannel	通道号
	[in] szCommand	命令参数
	[out] obj	接收的协议缓冲区
	[in] typeName	对应结构体类型
	[in] waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.3.2 获取设备能力 GetDevCaps

表3-31 获取设备能力说明

选项	说明	
描述	获取设备能力	
函数	<pre>bool GetDevCaps(IntPtr ILoginID, EM_DEVCAP_TYPE nType, IntPtr pInBuf, IntPtr pOutBuf, int nWaitTime)</pre>	
参数	[in] ILoginID	登录句柄
	[in] nType	设备类型 控制参数根据 type 不同而不同
	[in] pInBuf	获取设备能力（入参）
	[out] pOutBuf	获取设备能力（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

nType、pInBuf 和 pOutBuf 的对照关系，请参见表 3-32。

表3-32 nType、pInBuf 和 pOutBuf 对照关系

nType	描述	pInBuf	pOutBuf
EM_DEVCAP_TYPE. FACEINFO_CAPS	获得人脸门禁控制 器能力集	NET_IN_GET_FACEIN FO_CAPS	NET_OUT_GET_FACEINF O_CAPS

3.2.3.3 查询设备状态 QueryDevState

表3-33 查询设备状态 QueryDevState 说明

选项	说明	
描述	直接获取远程设备连接状态	
函数	<pre>bool QueryDevState(IntPtr lLoginID, int nType, ref object obj, Type typeName, int waittime);</pre>	
参数	[in]lLoginID	LoginWithHighLevelSecurity 返回值
	[in]nType	查询信息类型，当获取远程设备连接状态时 nType 为 EM_DEVICE_STATE.VIRTUALCAMERA
	[out]obj	用于接收查询返回的数据的缓存，对应 NET_VIRTUALCAMERA_STATE_INFO 结构体
	[in]typeName	查询结构体的类型
	[in]waittime	查询状态等待时间
返回值	失败返回 false，成功返回 true	
说明	无	

nType、查询类型和结构体的对应关系，请参见表 3-34。

表3-34 nType、查询类型和结构体的对应关系

nType	描述	pBuf
EM_DEVICE_STATE.SOFTWARE	查询设备软件版本信息	NET_DEV_VERSION_INFO
EM_DEVICE_STATE.NETINTERFACE	查询网络接口信息	NET_DEV_NETINTERFACE_INFO
EM_DEVICE_STATE.RECORDSET	查询设备记录集信息	NET_CTRL_RECORDSET_PARAMETER
EM_DEVICE_STATE.DOOR_STATE	查询门禁状态（门磁）	NET_DOOR_STATUS_INFO

3.2.4 网络设置

3.2.4.1 IP 设置

3.2.4.1.1 查询配置信息 GetNewDevConfig

表3-35 查询配置信息 GetNewDevConfig 说明

选项	说明
描述	获取通道名称

选项	说明	
函数	<pre>bool GetNewDevConfig(IntPtr ILoginID, Int32 IChannel, string strCommand, ref object obj, Type typeName, int waittime);</pre>	
参数	[in]ILoginID	LoginWithHighLevelSecurity 返回值
	[in]IChannel	设备通道号，从 0 开始
	[in]strCommand	命令参数，查询通道名称 strCommand 为“ChannelTitle”
	[out]obj	查询到的信息数组
	[in]typeName	查询结构体的类型
	[in]waittime	等待超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.4.1.2 设置配置信息 SetNewDevConfig

表3-36 设置配置信息说明

选项	说明	
描述	获取配置，按照字符串格式	
函数	<pre>bool SetNewDevConfig(IntPtr ILoginID, int IChannel, string strCommand, object obj, Type typeName, int waittime);</pre>	
参数	[in] ILoginID	登录句柄
	[in] nChannelID	通道号
	[in] szCommand	命令参数信息
	[in]obj	设置的内容
	[in]typeName	设置结构体的类型
	[in]waittime	等待超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.4.2 主动注册设置

3.2.4.2.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.4.2.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.5 时间设置

3.2.5.1 时间设置

表3-37 时间设置说明

选项	说明	
描述	设置设备当前时间	
函数	bool SetupDeviceTime(IntPtr ILoginID, NET_TIME DeviceTime)	
参数	[in] ILoginID	登录句柄
	[in] DeviceTime	设置的设备时间
返回值	失败返回 false，成功返回 true	
说明	应用于系统校时时更改当前前端设备系统时间与本机系统时间同步	

3.2.5.2 NTP 校时、时区配置

3.2.5.2.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.5.2.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.5.3 夏令时设置

3.2.5.3.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.5.3.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.6 维护配置

3.2.6.1 修改登录密码

3.2.6.1.1 操作设备用户 OperateUserInfoNew

表3-38 操作设备用户说明

选项	说明	
描述	操作设备用户,最大支持 64 通道设备	
函数	<pre>bool OperateUserInfoNew(IntPtr lLoginID, EM_OPERATE_USER_TYPE nOperateType, IntPtr opParam, IntPtr subParam, int waittime)</pre>	
参数	[in]lLoginID	LoginWithHighLevelSecurity 的返回值
	[in] nOperateType	操作类型, 具体请参见表 3-39
	[in] opParam	设置用户信息的输入缓冲, 具体请参见表 3-39
	[in] subParam	设置用户信息的辅助输入缓冲, 当设置类型为修改信息的时候, 此处应传进来部分原始用户信息, 具体请参见表 3-39
	[in]waittime	超时时间, 默认 1000ms, 可根据需要自行设置
返回值	失败返回 false, 成功返回 true	
说明	为实现要求的功能, 设置更改设备的用户信息	

表3-39 nOperateType、opParam 和 subParam 之间的对应关系

nOperateType	opParam	subParam
EM_OPERATE_USER_TYPE.MODIFY_PASS WORD	NET_USER_INFO_NEW	NET_USER_INFO_NE W

3.2.6.2 重启设备

3.2.6.2.1 设备控制 ControlDevice

表3-40 设备控制说明

选项	说明	
描述	设备控制	
函数	<pre>bool ControlDevice(IntPtr lLoginID, EM_CtrlType type, IntPtr param, int waittime)</pre>	
参数	[in]lLoginID	LoginWithHighLevelSecurity 的返回值

选项	说明	
	[in]type	控制类型
	[in]param	控制参数，因 type 不同而不同
	[in]waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	失败返回 false，成功返回 true	
说明	无	

表3-41 Type 和 param 的对照关系

Type	描述	Param
REBOOT	重启设备	无
RECORDSET_INSERT	添加记录，获得记录集编号	NET_CTRL_RECORDSET_INSERT_PARAM
RECORDSET_INSERTEX	添加指纹记录，获得记录集编号	NET_CTRL_RECORDSET_INSERT_PARAM
RECORDSET_REMOVE	根据记录集编号删除某记录	NET_CTRL_RECORDSET_PARAM
RECORDSET_CLEAR	清除所有记录集信息	NET_CTRL_RECORDSET_PARAM
RECORDSET_UPDATE	更新某记录集编号的记录	NET_CTRL_RECORDSET_PARAM
RECORDSET_UPDATEEX	更新指纹记录集编号的记录	NET_CTRL_RECORDSET_PARAM
ACCESS_OPEN	门禁控制-开门	NET_CTRL_ARM_DISARM_PARAM
RESTOREDEFAULT	恢复设备的默认设置	NET_RESTORE_COMMON

3.2.6.3 恢复出厂设置

3.2.6.3.1 恢复出厂设置 ControlDevice、ResetSystem

- 关于 ControlDevice 的详细介绍，请参见“3.2.6.2.1 设备控制 ControlDevice”。
- 关于 ResetSystem 的详细介绍，请参见表 3-42。

表3-42 恢复出厂设置说明

选项	说明	
描述	恢复出厂设置	
函数	<pre>bool ResetSystem(IntPtr lLoginID, ref NET_IN_RESET_SYSTEM pInParam, ref NET_IN_RESET_SYSTEM pOutParam, int nWaitTime)</pre>	
参数	[in]lLoginID	LoginWithHighLevelSecurity 的返回值
	[in] pstInParam	恢复出厂设置入参
	[out] pstOutParam	恢复出厂设置出参
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	

3.2.6.4 设备升级

3.2.6.4.1 开始升级设备程序 StartUpgrade

表3-43 开始升级设备程序说明

选项	说明	
描述	开始升级设备程序--扩展	
函数	<pre> IntPtr StartUpgrade(IntPtr lLoginID, EM_UPGRADE_TYPE emType, string pchFileName, fUpgradeCallBack cbUpgrade, IntPtr dwUser) </pre>	
参数	[in] lLoginID	LoginWithHighLevelSecurity 的返回值
	[in] emType	枚举值，详细介绍请参见表 3-44
	[in] pchFileName	要升级的文件名
	[in] cbUpgrade	升级进度回调函数，详细介绍请参见 4.8 升级进度回调函数 fUpgradeCallBackEx
	[in] dwUser	用户自定义数据
返回值	失败返回 0，成功返回非 0 的值	
说明	设置远程程序的升级，返回程序升级句柄，调用本接口还没有发送升级程序数据，需要调用 SendUpgrade 接口，数据才能发送。	

表3-44 枚举值

emType	意义
BIOS_TYPE	BIOS 升级
WEB_TYPE	WEB 升级
BOOT_TYPE	BOOT 升级
CHARACTER_TYPE	汉字库
LOGO_TYPE	LOGO
EXE_TYPE	EXE,例如播放器等
DEVCONSTINFO_TYPE	设备固有信息设置(如：硬件 ID、MAC、序列号)
PERIPHERAL_TYPE	外设接入从片（如车载芯片）
GEOINFO_TYPE	地理信息定位芯片
MENU	菜单（设备操作界面的图片）
ROUTE	线路文件（如公交线路）
ROUTE_STATE_AUTO	报站音频（与线路配套的报站音频）
SCREEN	调度屏（如公交操作屏）

3.2.6.4.2 开始发送升级文件 SendUpgrade

表3-45 开始发送升级文件说明

选项	说明
描述	开始发送升级文件

选项	说明	
函数	bool SendUpgrade(IntPtr IUpgradeID)	
参数	[in] IUpgradeID	升级句柄 ID
返回值	失败返回 false, 成功返回 true	
说明	发送升级程序数据	

3.2.6.4.3 停止升级 StopUpgrade

表3-46 停止升级说明

选项	说明	
描述	停止升级	
函数	bool StopUpgrade(IntPtr IUpgradeID)	
参数	[in] IUpgradeID	升级句柄 ID
返回值	失败返回 false, 成功返回 true	
说明	不要在回调函数里调用此接口	

3.2.6.5 自动维护

3.2.6.5.1 查询配置信息 GetDevConfig

表3-47 查询配置信息说明

选项	说明	
描述	读取设备的配置信息	
函数	bool GetDevConfig(IntPtr ILoginID, EM_DEV_CFG_TYPE type, int IChannel, IntPtr lpOutBuffer, uint dwOutBufferSize, ref uint bytesReturned, int waittime)	
参数	[in] ILoginID	设备登录句柄
	[in] type	设备配置命令, 具体请参见 EM_DEV_CFG_TYPE 枚举
	[in] IChannel	通道号, 如果获取全部通道数据为 0xFFFFFFFF, 如果命令不需要通道号, 该参数无效
	[out] lpOutBuffer	接受数据缓冲指针
	[in] dwOutBufferSize	接收数据缓冲长度(以字节为单位)
	[out] bytesReturned	实际收到数据的长度
	[in] waittime	等待超时时间
返回值	失败返回 false, 成功返回 true	

选项	说明
说明	无

表3-48 dwCommand、lpOutBuffer 对应关系

dwCommand	查询类型	对应结构体 lpOutBuffer
DST_CFG	夏令时配置	NET_CFG_NTP_INFO
AUTOMTCFG	自动维护配置	NET_DEV_AUTOMT_CFG

3.2.6.5.2 设置配置信息 SetDevConfig

表3-49 设置配置信息说明

选项	说明	
描述	设置设备的配置信息	
函数	<pre>bool SetDevConfig(IntPtr lLoginID, EM_DEV_CFG_TYPE type, int lChannel, IntPtr lpInBuffer, uint dwInBufferSize, int waittime)</pre>	
参数	[in] lLoginID	设备登录句柄
	[in] type	设备配置命令
	[in] lChannel	通道号，如果获取全部通道数据为 0xFFFFFFFF，如果命令不需要通道号，该参数无效
	[in] lpInBuffer	数据缓冲指针
	[in] dwInBufferSize	数据缓冲长度(以字节为单位)
	[in] waittime	等待超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.7 人员管理

3.2.7.1 人员信息字段集合

请参见“3.2.6.2.1 设备控制 ControlDevice”和“3.2.3.3 查询设备状态 QueryDevState”。

3.2.8 门配置

3.2.8.1 门配置信息

3.2.8.1.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.8.1.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.9 门时间配置

3.2.9.1 时段配置

3.2.9.1.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.9.1.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.9.2 常开常闭时段配置

3.2.9.2.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.9.2.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.9.3 假日配置

请参见“3.2.6.2.1 设备控制 ControlDevice”和“3.2.3.3 查询设备状态 QueryDevState”。

3.2.10 门高级配置

3.2.10.1 分时段、首卡开门

3.2.10.1.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.10.1.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.10.2 多人组合开门

3.2.10.2.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.10.2.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.10.3 多门互锁

3.2.10.3.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.10.3.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.10.4 防反潜

3.2.10.4.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.2.10.4.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.2.10.5 开门密码

关于 ControlDevice 的详细介绍，请参见“3.2.6.2.1 设备控制 ControlDevice”。

3.2.10.6 设备日志

3.2.10.6.1 查询设备日志条数 QueryDevLogCount

表3-50 查询设备日志条数说明

选项	说明
描述	查询设备日志条数

选项	说明	
函数	bool QueryDevLogCount(IntPtr lLoginID, ref NET_IN_GETCOUNT_LOG_PARAM pInParam, ref NET_OUT_GETCOUNT_LOG_PARAM pOutParam, int nWaitTime)	
参数	[in] lLoginID	设备登录句柄
	[in] pInParam	查询日志的参数, 详细介绍请参见 NET_IN_GETCOUNT_LOG_PARAM
	[out] pOutParam	返回日志条数, 详细介绍请参见 NET_OUT_GETCOUNT_LOG_PARAM
	[in] waittime	查询超时时间
返回值	返回查询日志条数	
说明	无	

3.2.10.6.2 开始查询日志 StartQueryLog

表3-51 开始查询日志说明

选项	说明	
描述	开始查询设备日志	
函数	IntPtr StartQueryLog(IntPtr lLoginID, ref NET_IN_START_QUERYLOG pInParam, ref NET_OUT_START_QUERYLOG pOutParam, int nWaitTime)	
参数	[in] lLoginID	设备登录句柄
	[in] pInParam	开始查询日志的参数, 详细介绍请参见 NET_IN_START_QUERYLOG
	[out] pOutParam	开始查询日志输出参数, 详细介绍请参见 NET_OUT_START_QUERYLOG
	[in] nWaitTime	查询超时时间
返回值	失败返回 0, 成功返回非 0 的值	
说明	无	

3.2.10.6.3 获取日志 QueryNextLog

表3-52 获取日志说明

选项	说明	
描述	获取日志	
函数	bool QueryNextLog(IntPtr lFindLogID, ref NET_IN_QUERYNEXTLOG pInParam, ref NET_OUT_QUERYNEXTLOG pOutParam, int nWaitTime)	

选项	说明	
参数	[in] IFindLogID	查询日志句柄
	[in] pInParam	获取日志的输入参数，详细介绍请参见 NET_IN_QUERYNEXTLOG
	[out] pOutParam	获取日志的输出参数，详细介绍请参见 NET_OUT_QUERYNEXTLOG
	[in] nWaitTime	查询超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.10.6.4 结束查询日志 StopQueryLog

表3-53 结束查询日志说明

选项	说明	
描述	停止查询设备日志	
函数	bool StopQueryLog(IntPtr IFindLogID)	
参数	[in] IFindLogID	查询日志句柄
返回值	失败返回 false，成功返回 true	
说明	无	

3.2.11 记录查询

3.2.11.1 开门记录

3.2.11.1.1 查找记录条数 QueryRecordCount

表3-54 查询记录条数说明

选项	说明	
描述	查找记录条数	
函数	bool QueryRecordCount(IntPtr IFindHandle, ref int nRecordCount, int waittime)	
参数	[in] IFindHandle	查询日志句柄
	[out] nRecordCount	查询记录条数
	[in] waittime	查询超时时间
返回值	失败返回 false，成功返回 true	
说明	调用本接口之前应先调用 FindRecord 以打开查询句柄	

3.2.11.1.2 按查询条件查询记录 FindRecord

表3-55 按查询条件查询记录说明

选项	说明	
描述	按查询条件查询记录	
函数	<pre>bool FindRecord(IntPtr ILoginID, EM_NET_RECORD_TYPE emRecordType, object oCondition, Type tyCondition, ref IntPtr IFindID, int waittime)</pre>	
参数	[in] ILoginID	设备登录句柄
	[in] emRecordType	查询录像的类型
	[in] oCondition	查询日志的参数
	[in] tyCondition	查询的结构体类型
	[out] IFindID	返回查询句柄
	[in] waittime	等待超时时间
返回值	失败返回 false，成功返回 true	
说明	可以先调用本接口获得查询句柄，再调用 FindNextRecord 函数获取记录列表，查询完毕可以调用 FindRecordClose 关闭查询句柄	

表3-56 查询开门记录入参数说明

emRecordType 取值	结构体	说明
EM_NET_RECORD_TYPE. ACCESSCTLCARDREC_EX	NET_FIND_RECORD_ACCESSCTLCARDREC_C ONDITION_EX	用于开门记录查询

3.2.11.1.3 查找记录 FindNextRecord

表3-57 查找记录说明

选项	说明	
描述	查找记录:nFilecount:需要查询的条数,返回值 nRetNum 为媒体文件条数, nRetNum 小于 nFilecount 则相应时间段内的文件查询完毕	
函数	<pre>int FindNextRecord(IntPtr IFindeHandle, int nMaxNum, ref int nRetNum, ref List<object> ls, Type tyRecord, int waittime)</pre>	
	[in] IFindeHandle	设备查询句柄
	[in] nMaxNum	查询的最大数量
	[out] nRetNum	查询到的最大数量
	[out] ls	查询到的结构体列表
参数	[in] tyRecord	结构体对应类型
	[in] waittime	等待超时时间

选项	说明
返回值	成功返回 1，失败返回 0
说明	无

3.2.11.1.4 结束记录查询 FindRecordClose

表3-58 结束记录查询说明

选项	说明
描述	结束记录查询
函数	bool FindRecordClose(IntPtr IFindHandle)
参数	[in] IFindHandle FindRecord 的返回值
返回值	失败返回 false，成功返回 true
说明	调用 FindRecord 打开查询句柄，查询完毕后应调用本函数以关闭查询句柄

3.3 门禁控制器/人脸一体机（二代）

3.3.1 门禁控制

关于门控制接口的详细介绍，请参见“3.1.5.1 设备控制 ControlDevice”。

关于门磁状态接口的详细介绍，请参见“3.2.3.3 查询设备状态 QueryDevState”。

3.3.2 报警事件

请参见“3.1.6 报警侦听”。

3.3.3 设备信息查看

3.3.3.1 获取设备能力 GetDevCaps

表3-59 获取设备能力说明

选项	说明
描述	获取设备能力
函数	bool GetDevCaps(IntPtr ILoginID, EM_DEVCAP_TYPE nType, IntPtr pInBuf, IntPtr pOutBuf, int nWaitTime)
参数	[in] ILoginID 登录句柄
	[in] nType 设备类型 控制参数根据 type 不同而不同

选项	说明	
	[in] pInBuf	获取设备能力（入参）
	[out] pOutBuf	获取设备能力（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false ，成功返回 true	
说明	无	

nType、pInBuf 和 pOutBuf 的对照关系，请参见表 3-32。

表3-60 nType、pInBuf 和 pOutBuf 对照关系

nType	描述	pInBuf	pOutBuf
EM_DEVCAP_TYPE. ACCESSCONTROL_CAP S	获取门禁能力	NET_IN_AC_CAPS	NET_OUT_AC_CAPS

3.3.3.2 查询设备状态 QueryDevState

关于 QueryDevState 详细介绍，请参见“3.2.3.3 查询设备状态 QueryDevState”。

3.3.4 网络设置

请参见“3.2.4 网络设置”。

3.3.5 时间设置

请参见“3.2.5 时间设置”。

3.3.6 维护配置

请参见“3.2.6 维护配置”。

3.3.7 人员管理

3.3.7.1 用户管理

3.3.7.1.1 开始查询人员信息接口 StartFindUserInfo

表3-61 开始查询人员信息接口说明

选项	说明
描述	开始查询人员信息接口

选项	说明
函数	IntPtr StartFindUserInfo(IntPtr ILoginID, ref NET_IN_USERINFO_START_FIND pstIn, ref NET_OUT_USERINFO_START_FIND pstOut, int nWaitTime)
参数	[in] ILoginID 登录句柄
	[in] pstIn 开始查询人员信息接口（入参）
	[out] pstOut 开始查询人员信息接口（出参）
	[in] nWaitTime 超时时间
返回值	失败返回 0，成功返回非 0 的值
说明	无

3.3.7.1.2 获取人员信息接口 DoFindUserInfo

表3-62 获取人员信息接口说明

选项	说明
描述	获取人员信息接口
函数	bool DoFindUserInfo(IntPtr IFindHandle, ref NET_IN_USERINFO_DO_FIND pstIn, ref NET_OUT_USERINFO_DO_FIND pstOut, int nWaitTime)
参数	[in] IFindHandle StartFindUserInfo 返回值
	[in] pstIn 获取人员信息接口（入参）
	[out] pstOut 获取人员信息接口（出参）
	[in] nWaitTime 超时时间
返回值	失败返回 false，成功返回 true
说明	无

3.3.7.1.3 停止查询人员信息接口 StopFindUserInfo

表3-63 停止查询人员信息接口说明

选项	说明
描述	停止查询人员信息接口
函数	bool StopFindUserInfo(IntPtr IFindHandle)
参数	[in] IFindHandle StartFindUserInfo 返回值
返回值	失败返回 false，成功返回 true
说明	无

3.3.7.1.4 门禁用户信息获取接口 GetOperateAccessUserService

表3-64 门禁用户信息获取接口说明

选项	说明	
描述	门禁人员信息管理接口	
函数	<pre>bool GetOperateAccessUserService(IntPtr lLoginID, string[] userid, out NET_ACCESS_USER_INFO[] stOutParam1, out NET_EM_FAILCODE[] stOutParam2, int nWaitTime)</pre>	
参数	[in] lLoginID	登录句柄
	[in] userid	要查询的用户的 userid
	[in] stOutParam1	用户信息管理（出参）
	[out] stOutParam2	用户信息查询错误类型（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.1.5 门禁用户信息添加接口 InsertOperateAccessUserService

表3-65 门禁用户信息添加接口说明

选项	说明	
描述	门禁用户信息添加接口	
函数	<pre>bool InsertOperateAccessUserService(IntPtr lLoginID, NET_ACCESS_USER_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>	
参数	[in] lLoginID	登录句柄
	[in] stInParam	用户信息管理（入参）
	[out] stOutParam	用户信息管理（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.1.6 门禁用户信息删除接口 RemoveOperateAccessUserService

表3-66 门禁用户信息删除接口说明

选项	说明	
描述	门禁用户信息删除接口	

选项	说明	
函数	bool RemoveOperateAccessUserService(IntPtr ILoginID, string[] userid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
参数	[in] ILoginID	登录句柄
	[in] userid	要删除的用户的 userid
	[out] stOutParam	用户信息查询错误类型（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.1.7 门禁用户信息清空接口 ClearOperateAccessUserService

表3-67 门禁用户信息清空接口说明

选项	说明	
描述	门禁用户信息清空接口	
函数	bool ClearOperateAccessUserService(IntPtr ILoginID, int nWaitTime)	
参数	[in] ILoginID	登录句柄
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.2 卡片管理

3.3.7.2.1 开始查询卡片信息接口 StartFindCardInfo

表3-68 开始查询卡片信息接口说明

选项	说明	
描述	开始查询卡片信息接口	
函数	IntPtr StartFindCardInfo(IntPtr ILoginID, ref NET_IN_CARDINFO_START_FIND pstIn, ref NET_OUT_CARDINFO_START_FIND pstOut, int nWaitTime)	
参数	[in] ILoginID	登录句柄
	[in] pstIn	开始查询卡片信息接口（入参）
	[out] pstOut	开始查询卡片信息接口（出参）
	[in] nWaitTime	超时时间
返回值	成功返回查询句柄，失败返回 0	

选项	说明
说明	无

3.3.7.2.2 获取卡片信息接口 DoFindCardInfo

表3-69 获取卡片信息接口说明

选项	说明
描述	获取卡片信息接口
函数	bool DoFindCardInfo(IntPtr IFindHandle, ref NET_IN_CARDINFO_DO_FIND pstIn, ref NET_OUT_CARDINFO_DO_FIND pstOut, int nWaitTime)
参数	[in] IFindHandle StartFindCardInfo 返回值
	[in] pstIn 获取卡片信息接口（入参）
	[out] pstOut 获取卡片信息接口（出参）
	[in] nWaitTime 超时时间
返回值	失败返回 false，成功返回 true
说明	无

3.3.7.2.3 停止查询卡片信息接口 StopFindUserInfo

表3-70 停止查询卡片信息接口说明

选项	说明
描述	停止查询卡片信息接口
函数	bool StopFindCardInfo(IntPtr IFindHandle)
参数	[in] IFindHandle StartFindCardInfo 返回值
返回值	失败返回 false，成功返回 true
说明	无

3.3.7.2.4 门禁卡片信息获取接口 GetOperateAccessCardService

表3-71 门禁卡片信息获取接口说明

选项	说明
描述	门禁卡片信息获取接口
函数	bool GetOperateAccessCardService(IntPtr ILoginID, string[] Cardid, out NET_ACCESS_CARD_INFO[] stOutParam1, out NET_EM_FAILCODE[] stOutParam2, int nWaitTime)
参数	[in] ILoginID 登录句柄
	[in] Cardid 要查询的卡片的 Cardid
	[in] stOutParam1 卡片信息管理（出参）

选项	说明	
	[out] stOutParam2	卡片信息查询错误类型（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.2.5 门禁卡片信息添加接口 InsertOperateAccessCardService

表3-72 门禁卡片信息添加接口说明

选项	说明	
描述	门禁卡片信息添加接口	
函数	<pre>bool InsertOperateAccessCardService(IntPtr lLoginID, NET_ACCESS_CARD_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>	
参数	[in] lLoginID	登录句柄
	[in] stInParam	卡片信息管理（入参）
	[out] stOutParam	卡片信息管理（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.2.6 门禁卡片信息删除接口 RemoveOperateAccessCardService

表3-73 门禁卡片信息删除接口说明

选项	说明	
描述	门禁卡片信息删除接口	
函数	<pre>bool RemoveOperateAccessCardService(IntPtr lLoginID, string[] Cardid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>	
参数	[in] lLoginID	登录句柄
	[in] Cardid	要删除的卡片的 Cardid
	[out] stOutParam	卡片信息查询错误类型（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.2.7 门禁卡片信息更新接口 ClearOperateAccessCardService

表3-74 门禁卡片信息更新接口说明

选项	说明	
描述	门禁卡片信息更新接口	

选项	说明	
函数	bool UpdateOperateAccessCardService(IntPtr ILoginID, NET_ACCESS_CARD_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
参数	[in] ILoginID	登录句柄
	[in] stInParam	卡片信息管理（入参）
	[out] stOutParam	卡片信息管理（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.3 人脸管理

3.3.7.3.1 门禁人脸信息管理接口 OperateAccessFaceService

表3-75 门禁人脸信息管理接口说明

选项	说明	
描述	门禁人脸信息管理接口	
函数	bool OperateAccessFaceService(IntPtr ILoginID, EM_NET_ACCESS_CTL_FACE_SERVICE emtype, IntPtr pstInParam, IntPtr pstOutParam, int nWaitTime)	
参数	[in] ILoginID	登录句柄
	[in] emtype	人脸信息操作类型
	[in] pstInParam	人脸信息管理（入参）
	[out] pstOutParam	人脸信息管理（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

emtype、pInBuf 和 pOutBuf 的对照关系，请参见表 3-76。

表3-76 nType、pInBuf 和 pOutBuf 对照关系

emtype	描述	pInBuf	pOutBuf
EM_NET_ACCESS_CTL_FACE_SERVICE.INSERT	添加人脸信息	NET_IN_ACCESS_FACE_SERVICE_INSERT	NET_OUT_ACCESS_FACE_SERVICE_INSERT
EM_NET_ACCESS_CTL_FACE_SERVICE.GET	获取人脸信息	NET_IN_ACCESS_FACE_SERVICE_GET	NET_OUT_ACCESS_FACE_SERVICE_GET
EM_NET_ACCESS_CTL_FACE_SERVICE.UPDATE	更新人脸信息	NET_IN_ACCESS_FACE_SERVICE_UPDATE	NET_OUT_ACCESS_FACE_SERVICE_UPDATE

emtype	描述	pInBuf	pOutBuf
EM_NET_ACCESS_CTL_FACE_SERVICE.REMOVE	删除人脸信息	NET_IN_ACCESS_FACE_SERVICE.REMOVE	NET_OUT_ACCESS_FACE_SERVICE.REMOVE
EM_NET_ACCESS_CTL_FACE_SERVICE.CLEAR	清空人脸信息	NET_IN_ACCESS_FACE_SERVICE.CLEAR	NET_OUT_ACCESS_FACE_SERVICE.CLEAR

3.3.7.4 指纹管理

3.3.7.4.1 门禁指纹信息获取接口 GetOperateAccessFingerprintService

表3-77 门禁指纹信息获取接口说明

选项	说明	
描述	门禁指纹信息管理接口	
函数	<pre>bool GetOperateAccessFingerprintService(IntPtr ILoginID, string userid, IntPtr pFingerprintData, int dataLen, out NET_ACCESS_FINGERPRINT_INFO stOutParam1, int nWaitTime)</pre>	
参数	[in] ILoginID	登录句柄
	[in] userid	要查询的用户的 userid
	[out] pFingerprintData	指纹信息数据（出参）
	[out] dataLen	指纹信息数据长度（出参）
	[out] stOutParam1	指纹信息查询错误类型（出参）
	[in] nWaitTime	超时时间
返回值	失败返回 false，成功返回 true	
说明	无	

3.3.7.4.2 门禁指纹信息添加接口 InsertOperateAccessFingerprintService

表3-78 门禁指纹信息添加接口说明

选项	说明	
描述	门禁指纹信息管理接口	
函数	<pre>bool InsertOperateAccessFingerprintService(IntPtr ILoginID, NET_ACCESS_FINGERPRINT_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>	
参数	[in] ILoginID	登录句柄
	[in] stInParam	指纹信息管理（入参）
	[out] stOutParam	指纹信息管理（出参）
	[in] nWaitTime	超时时间

选项	说明
返回值	失败返回 false, 成功返回 true
说明	无

3.3.7.4.3 门禁指纹信息删除接口 RemoveOperateAccessFingerprintService

表3-79 门禁指纹信息删除接口说明

选项	说明
描述	门禁指纹信息管理接口
函数	bool RemoveOperateAccessFingerprintService(IntPtr ILoginID, string[] userid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)
参数	[in] ILoginID 登录句柄
	[in] userid 要删除的用户的 userid
	[out] stOutParam 指纹信息查询错误类型（出参）
	[in] nWaitTime 超时时间
返回值	失败返回 false, 成功返回 true
说明	无

3.3.7.4.4 门禁指纹信息清空接口 ClearOperateAccessFingerprintService

表3-80 门禁指纹信息清空接口说明

选项	说明
描述	门禁指纹信息管理接口
函数	bool ClearOperateAccessFingerprintService(IntPtr ILoginID, int nWaitTime)
参数	[in] ILoginID 登录句柄
	[in] nWaitTime 超时时间
返回值	失败返回 false, 成功返回 true
说明	无

3.3.8 门配置

3.3.8.1 门配置信息

3.3.8.1.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍, 请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.3.8.1.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.3.9 门时间配置

3.3.9.1 时段配置

3.3.9.1.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.3.9.1.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.3.9.2 常开常闭时段配置

3.3.9.2.1 查询配置信息 GetNewDevConfig

关于 GetNewDevConfig 的详细介绍，请参见“3.2.4.1.1 查询配置信息 GetNewDevConfig”。

3.3.9.2.2 设置配置信息 SetNewDevConfig

关于 SetNewDevConfig 的详细介绍，请参见“3.2.4.1.2 设置配置信息 SetNewDevConfig”。

3.3.9.3 假日组

3.3.9.3.1 获取假日组接口 GetOperateConfig

表3-81 获取假日组接口说明

选项	说明	
描述	获取假日组接口	
函数	bool GetOperateConfig(IntPtr ILoginID, EM_CFG_OPERATE_TYPE cfg_type, int IChannel, ref object obj, Type typeName, int waittime)	
参数	[in] ILoginID	登录句柄
	[in] cfg_type	设置配置信息的类型
	[in] IChannel	通道号
	[out] obj	返回的数据结构体
	[in] typeName	结构体类型

选项	说明	
	[in] waittime	超时时间
返回值	失败返回 false, 成功返回 true	
说明	无	

表3-82 获取假日组信息 cfg_type 参数赋值说明

cfg_type	描述	typeName
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP	获取假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

3.3.9.3.2 设置假日组接口 SetOperateConfig

表3-83 设置假日组接口说明

选项	说明	
描述	设置假日组接口	
函数	<pre>bool SetOperateConfig(IntPtr ILoginID, EM_CFG_OPERATE_TYPE cfg_type, int IChannel, object obj, Type typeName, int waittime)</pre>	
参数	[in] ILoginID	登录句柄
	[in] cfg_type	设置配置的类型
	[in] IChannel	通道号
	[in] obj	设置配置的结构体
	[in] typeName	结构体类型
	[in] waittime	超时时间
返回值	失败返回 false, 成功返回 true	
说明	无	

表3-84 设置假日组信息 cfg_type 参数赋值说明

cfg_type	描述	typeName
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP	设置假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

3.3.9.4 假日计划

关于接口的详细介绍, 请参见“3.3.9.3 假日组”。

表3-85 获取假日计划信息 emCfgOpType 参数赋值说明

cfg_type	描述	typeName
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE	获取计划信息	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

表3-86 设置假日计划信息 emCfgOpType 参数赋值说明

cfg_type	描述	typeName
EM_CFG_OPERATE_TYPE. SPECIALDAYS_SCHEDULE	设置假日计划信息	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

3.3.10 门高级配置

请参见“3.2.10 门高级配置”。

3.3.11 记录查询

3.3.11.1 开门记录

请参见“3.2.11.1 开门记录”。

3.3.11.2 报警记录

3.3.11.2.1 查找记录条数 QueryRecordCount

请参见“3.2.11.1.1 查询记录条数”。

3.3.11.2.2 按查询条件查询记录 FindRecord

请参见“3.2.11.1.2 按查询条件查询记录”。

表3-87 查询开门记录入参数说明

emRecordType 取值	结构体	说明
EM_NET_RECORD_TYPE. ACCESS_ALARMRECORD	NET_RECORD_ACCESS_ALARMRECORD_INF 0	用于门禁报警记录查询

3.3.11.2.3 查找记录 FindNextRecord

请参见“3.2.11.1.3 查询记录”。

3.3.11.2.4 结束记录查询 FindRecordClose

请参见“3.2.11.1.4 结束记录查询”。

第 4 章 回调函数

4.1 搜索设备回调函数 fSearchDevicesCB

表4-1 搜索设备回调函数说明

选项	说明	
描述	搜索设备回调函数	
函数	public delegate void fSearchDevicesCB(IntPtr pDevNetInfo, IntPtr pUserData);	
参数	[out]pDevNetInfo	搜索的设备信息
	[out]pUserData	用户数据
返回值	无	
说明	无	

4.2 异步搜索设备回调函数 fSearchDevicesCBEx

表4-2 异步搜索设备回调函数 fSearchDevicesCBEx 说明

选项	说明	
描述	异步搜索设备回调	
函数	public delegate void fSearchDevicesCBEx(IntPtr ISearchHandle, IntPtr pDevNetInfo, IntPtr pUserData);	
参数	[out]ISearchHandle	搜索句柄
	[out]pDevNetInfo	搜索的设备信息
	[out]pUserData	用户数据
返回值	无	
说明	无	

4.3 断线回调函数 fDisconnectCallBack

表4-3 断线回调函数 fDisconnectCallBack 说明

选项	说明
描述	断线回调函数

选项	说明	
函数	<pre>public delegate void fDisconnectCallBack(IntPtr lLoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);</pre>	
参数	[out]lLoginID	LoginWithHighLevelSecurity 的返回值
	[out]pchDVRIP	断线的设备 IP
	[out]nDVRPort	断线的设备端口
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.4 断线重连回调函数 fHaveReConnectCallBack

表4-4 断线重连回调函数 fHaveReConnectCallBack 说明

选项	说明	
描述	断线重连回调函数	
函数	<pre>public delegate void fHaveReConnectCallBack(IntPtr lLoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);</pre>	
参数	[out]lLoginID	LoginWithHighLevelSecurity 的返回值
	[out]pchDVRIP	断线后重连成功的设备 IP
	[out]nDVRPort	断线后重连成功的设备端口
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.5 实时监视数据回调函数 fRealDataCallBackEx2

表4-5 实时监视数据回调函数说明

选项	说明
描述	实时监视数据回调函数

选项	说明	
函数	<pre>public delegate void fRealDataCallBackEx2(IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, IntPtr param, IntPtr dwUser);</pre>	
参数	[out]IRealHandle	RealPlay 的返回值
	[out]dwDataType	数据类型 <ul style="list-style-type: none"> • 0 表示原始数据 • 1 表示带有帧信息的数据 • 2 表示 YUV 数据 • 3 表示 PCM 音频数据
	[out]pBuffer	监视数据块地址
	[out]dwBufSize	监视数据块的长度，单位：字节
	[out]param	回调数据参数结构体，dwDataType 值不同类型不同。 <ul style="list-style-type: none"> • dwDataType 为 0 时，param 为空指针 • dwDataType 为 1 时，param 为 NET_VideoFrameParam 结构体指针 • dwDataType 为 3 时，param 为 NET_CBPCMDDataParam 结构体指针
[out]dwUser	回调函数的用户参数	
返回值	无	
说明	无	

4.6 音频数据回调函数 pfAudioDataCallBack

表4-6 音频数据回调函数 fAudioDataCallBack 说明

选项	说明	
描述	语音对讲的音频数据回调函数	
函数	<pre>public delegate void fAudioDataCallBack(IntPtr ITalkHandle, IntPtr pDataBuf, uint dwBufSize, byte byAudioFlag, IntPtr dwUser);</pre>	
参数	[out]ITalkHandle	StartTalk 的返回值
	[out]pDataBuf	音频数据块地址
	[out]dwBufSize	音频数据块的长度，单位：字节
	[out]byAudioFlag	数据类型标志，0 表示来自本地采集，1 表示来自设备发送

选项	说明	
	[out]dwUser	回调函数的用户参数
返回值	无	
说明	无	

4.7 报警回调函数 fMessCallBackEx

表4-7 报警回调函数 fMessCallBackEx 说明

选项	说明	
描述	报警回调函数	
函数	<pre>public delegate bool fMessCallBackEx(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser);</pre>	
参数	[out]ICommand	报警类型，具体请参见表 4-8
	[out]ILoginID	登录接口返回值
	[out]pBuf	接收报警数据的缓存，根据调用的侦听接口和 ICommand 值不同，填充的数据不同
	[out]dwBufLen	pBuf 的长度，单位：字节
	[out]pchDVRIP	设备 ip
	[out]nDVRPort	端口
	[out]bAlarmAckFlag	TRUE，该事件为可以进行确认的事件；FALSE，该事件无法进行确认
	[out]nEventID	用于对 AlarmAck 接口的入参进行赋值，当 bAlarmAckFlag 为 TRUE 时，该数据有效
	[out]dwUser	用户自定义数据
返回值	TRUE 回调函数执行正确，FALSE 执行错误	
说明	一般在应用程序初始化时调用设置回调，在回调函数中根据不同的设备 ID 和命令值做出不同的处理	

表4-8 报警类型和结构体对应关系

报警所属业务	报警类型名称	ICommand	pBuf
报警主机	本地报警事件	ALARM_ALARM_EX2	NET_ALARM_ALARM_INFO_EX2
	电源故障事件	ALARM_POWERFAULT	NET_ALARM_POWERFAULT_INFO
	防拆事件	ALARM_CHASSISINTRUDED	NET_ALARM_CHASSISINTRUDED_INFO

报警所属业务	报警类型名称	lCommand	pBuf
	扩展报警输入通道事件	ALARM_ALARMEXTENDED	NET_ALARM_ALARMEXTENDED_INFO
	紧急事件	URGENCY_ALARM_EX	数据为 16 个字节数组，每个字节表示一个通道状态 <ul style="list-style-type: none"> ● 1 为有报警 ● 0 为无报警
	蓄电池低电压事件	ALARM_BATTERYLOWPOWER	NET_ALARM_BATTERYLOWPOWER_INFO
	设备邀请平台对讲事件	ALARM_TALKING_INVITE	NET_ALARM_TALKING_INVITE_INFO
	设备布防模式变化事件	ALARM_ARMMODE_CHANGE_EVENT	NET_ALARM_ARMMODE_CHANGE_INFO
	防区旁路状态变化事件	ALARM_BYPASSMODE_CHANGE_EVENT	NET_ALARM_BYPASSMODE_CHANGE_INFO
	报警输入源信号事件	ALARM_INPUT_SOURCE_SIGNAL	NET_ALARM_INPUT_SOURCE_SIGNAL_INFO
	消警事件	ALARM_ALARMCLEAR	NET_ALARM_ALARMCLEAR_INFO
	子系统状态改变事件	ALARM_SUBSYSTEM_STATE_CHANGE	NET_ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	扩展模块掉线事件	ALARM_MODULE_LOST	NET_ALARM_MODULE_LOST_INFO
	PSTN 掉线事件	ALARM_PSTN_BREAK_LINE	NET_ALARM_PSTN_BREAK_LINE_INFO
	模拟量报警事件	ALARM_ANALOG_PULSE	NET_ALARM_ANALOGPULSE_INFO
	报警传输事件	ALARM_PROFILE_ALARM_TRANSMIT	NET_ALARM_PROFILE_ALARM_TRANSMIT_INFO
	无线设备低电量报警事件	ALARM_WIRELESSDEV_LOWPPOWER	NET_ALARM_WIRELESSDEV_LOWPPOWER_INFO
	防区布撤防状态改变事件	ALARM_DEFENCE_ARMMODE_CHANGE	NET_ALARM_DEFENCE_ARMMODECHANGE_INFO
	子系统布撤防状态改变事件	ALARM_SUBSYSTEM_ARMMODE_CHANGE	NET_ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	探测器异常报警	ALARM_SENSOR_ABNORMAL	NET_ALARM_SENSOR_ABNORMAL_INFO
病人活动状态报警事件	ALARM_PATIENTDETECTION	NET_ALARM_PATIENTDETECTION_INFO	
门禁	门禁事件	ALARM_ACCESS_CTL_EVENT	NET_ALARM_ACCESS_CTL_EVENT_INFO
	门禁未关事件详细信息	ALARM_ACCESS_CTL_NOT_CLOSE	NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	闯入事件详细信息	ALARM_ACCESS_CTL_BREAK_IN	NET_ALARM_ACCESS_CTL_BREAK_IN_INFO

报警所属业务	报警类型名称	lCommand	pBuf
	反复进入事件详细信息	ALARM_ACCESS_CTL_REPEAT_ENTER	NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	恶意开门事件	ALARM_ACCESS_CTL_MALICIOUS	NET_ALARM_ACCESS_CTL_MALICIOUS
	胁迫卡刷卡事件详细信息	ALARM_ACCESS_CTL_DURESS	NET_ALARM_ACCESS_CTL_DURESS_INFO
	多人组合开门事件	ALARM_OPENDOORGROUP	NET_ALARM_OPEN_DOOR_GROUP_INFO
	防拆事件	ALARM_CHASSISINTRUDED	NET_ALARM_CHASSISINTRUDED_INFO
	本地报警事件	ALARM_ALARM_EX2	NET_ALARM_ALARM_INFO_EX2
	门禁状态事件	ALARM_ACCESS_CTL_STATUS	NET_ALARM_ACCESS_CTL_STATUS_INFO
	锁舌报警	ALARM_ACCESS_CTL_STATUS	NET_ALARM_ACCESS_CTL_STATUS_INFO
	获取指纹事件	ALARM_FINGER_PRINT	NET_ALARM_CAPTURE_FINGER_PRINT_INFO
可视对讲	直连情况下，呼叫无答应事件	ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO
	手机号码上报事件	ALARM_TELEPHONE_CHECK	NET_ALARM_TELEPHONE_CHECK_INFO
	VTS 状态上报	ALARM_VTSTATE_UPDATE	NET_ALARM_VTSTATE_UPDATE_INFO
	VTO 人脸识别	ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	设备请求对方发起对讲事件	ALARM_TALKING_INVITE	NET_ALARM_TALKING_INVITE_INFO
	设备取消对讲请求事件	ALARM_TALKING_IGNORE_INVITE	NET_ALARM_TALKING_IGNORE_INVITE_INFO
	设备主动挂断对讲事件	ALARM_TALKING_HANGUP	NET_ALARM_TALKING_HANGUP_INFO
	雷达监测超速报警事件	ALARM_RADAR_HIGH_SPEED	NET_ALARM_RADAR_HIGH_SPEED_INFO

4.8 升级进度回调函数 fUpgradeCallBackEx

表4-9 升级进度回调函数说明

选项	说明
描述	升级设备程序回调函数原形，支持 G 以上升级文件

选项	说明	
函数	<pre>public delegate void fUpgradeCallBackEx(IntPtr ILoginID, IntPtr IUpgradechannel, long nTotalSize, long nSendSize, IntPtr dwUser);</pre>	
参数	[out]ILoginID	登录接口返回值
	[out] IUpgradechannel	StartUpgrade 返回的升级句柄 ID
	[out] nTotalSize	升级文件的总长度(单位:字节)
	[out] nSendSize	已发送的文件长度(单位:字节), 为-1 时, 表示发送升级文件结束
	[out]dwUser	用户自定义数据
返回值	无	
说明	升级设备程序回调函数原形支持 G 以上升级文件 nTotalSize = 0, nSendSize = -1 表示升级完成 nTotalSize = 0, nSendSize = -2 表示升级出错 nTotalSize = 0, nSendSize = -3 用户没有权限升级 nTotalSize = 0, nSendSize = -4 升级程序版本过低 nTotalSize = -1, nSendSize = XX 表示升级进度 nTotalSize = XX, nSendSize = XX 表示升级文件发送进度	

附录1 法律声明

商标声明

- VGA 是 IBM 公司的商标。
- Windows 标识和 Windows 是微软公司的商标或注册商标。
- 在本文档中可能提及的其他商标或公司的名称，由其各自所有者拥有。

责任声明

- 在适用法律允许的范围内，在任何情况下，本公司都不对因本文档中相关内容及描述的产品而产生任何特殊的、附随的、间接的、继发性的损害进行赔偿，也不对任何利润、数据、商誉、文档丢失或预期节约的损失进行赔偿。
- 本文档中描述的产品均“按照现状”提供，除非适用法律要求，本公司对文档中的所有内容不提供任何明示或暗示的保证，包括但不限于适销性、质量满意度、适合特定目的、不侵犯第三方权利等保证。

隐私保护提醒

您安装了我们的产品，您可能会采集人脸、指纹、车牌、邮箱、电话、GPS 等个人信息。在使用产品过程中，您需要遵守所在地区或国家的隐私保护法律法规要求，保障他人的合法权益。如，提供清晰、可见的标牌，告知相关权利人视频监控区域的存在，并提供相应的联系方式。

关于本文档

- 产品请以实物为准，本文档仅供参考。
- 本公司保留随时维护本文档中任何信息的权利，维护的内容将会在本文档的新版本中加入，恕不另行通知。
- 本文档如有不准确或不详尽的地方，或印刷错误，请以公司最终解释为准。
- 本文档供多个型号产品做参考，每个产品的具体操作不逐一例举，请用户根据实际产品自行对照操作。
- 如不按照本文档中的指导进行操作，因此而造成的任何损失由使用方自行承担。
- 如获取到的 PDF 文档无法打开，请将阅读工具升级到最新版本或使用其他主流阅读工具。

附录2 网络安全建议

保障设备基本网络安全的必须措施：

1. 使用复杂密码

请参考如下建议进行密码设置：

- 长度不小于 8 个字符。
- 至少包含两种字符类型，字符类型包括大小写字母、数字和符号。
- 不包含帐户名称或帐户名称的倒序。
- 不要使用连续字符，如 123、abc 等。
- 不要使用重叠字符，如 111、aaa 等。

2. 及时更新固件和客户端软件

- 按科技行业的标准作业规范，设备的固件需要及时更新至最新版本，以保证设备具有最新的功能和安全性。设备接入公网情况下，建议开启在线升级自动检测功能，便于及时获知厂商发布的固件更新信息。
- 建议您下载和使用最新版本客户端软件。

增强设备网络安全的建议措施：

1. 物理防护

建议您对设备（尤其是存储类设备）进行物理防护，比如将设备放置在专用机房、机柜，并做好门禁权限和钥匙管理，防止未经授权的人员进行破坏硬件、外接设备（例如 U 盘、串口）等物理接触行为。

2. 定期修改密码

建议您定期修改密码，以降低被猜测或破解的风险。

3. 及时设置、更新密码重置信息

设备支持密码重置功能，为了降低该功能被攻击者利用的风险，请您及时设置密码重置相关信息，包含预留手机号/邮箱、密保问题，如有信息变更，请及时修改。设置密保问题时，建议不要使用容易猜测的答案。

4. 开启帐户锁定

出厂默认开启帐户锁定功能，建议您保持开启状态，以保护帐户安全。在攻击者多次密码尝试失败后，其对应帐户及源 IP 将会被锁定。

5. 更改 HTTP 及其他服务默认端口

建议您将 HTTP 及其他服务默认端口更改为 1024~65535 间的任意端口，以减小被攻击者猜测服务端口的风险。

6. 使能 HTTPS

建议您开启 HTTPS，通过安全的通道访问 Web 服务。

7. 启用白名单

建议您开启白名单功能，开启后仅允许白名单列表中的 IP 访问设备。因此，请务必将您的电脑 IP 地址，以及配套的设备 IP 地址加入白名单列表中。

8. MAC 地址绑定

建议在设备端将其网关设备的 IP 与 MAC 地址进行绑定，以降低 ARP 欺骗风险。

9. 合理分配帐户及权限

根据业务和管理需要，合理新增用户，并合理为其分配最小权限集合。

10. 关闭非必需服务，使用安全的模式

如果没有需要，建议您关闭 SNMP、SMTP、UPnP 等功能，以降低设备面临的风险。

如果有需要，强烈建议您使用安全的模式，包括但不限于：

- SNMP：选择 SNMP v3，并设置复杂的加密密码和鉴权密码。
- SMTP：选择 TLS 方式接入邮箱服务器。
- FTP：选择 SFTP，并设置复杂密码。
- AP 热点：选择 WPA2-PSK 加密模式，并设置复杂密码。

11. 音视频加密传输

如果您的音视频数据包含重要或敏感内容，建议启用加密传输功能，以降低音视频数据传输过程中被窃取的风险。

12. 使用 PoE 方式连接设备

如果设备支持 PoE 功能，建议采用 PoE 方式连接设备，使摄像机与其他网络隔离。

13. 安全审计

- 查看在线用户：建议您不定期查看在线用户，识别是否有非法用户登录。
- 查看设备日志：通过查看日志，可以获知尝试登录设备的 IP 信息，以及已登录用户的关键操作信息。

14. 网络日志

由于设备存储容量限制，日志存储能力有限，如果您需要长期保存日志，建议您启用网络日志功能，确保关键日志同步至网络日志服务器，便于问题回溯。

15. 安全网络环境的搭建

为了更好地保障设备的安全性，降低网络安全风险，建议您：

- 关闭路由器端口映射功能，避免外部网络直接访问路由器内网设备的服务。
- 根据实际网络需要，对网络进行划区隔离：若两个子网间没有通信需求，建议使用 VLAN、网闸等方式对其进行网络分割，达到网络隔离效果。
- 建立 802.1x 接入认证体系，以降低非法终端接入专网的风险。
- 启用设备的防火墙或者黑白名单功能，降低设备可能遭受攻击的风险。