NetSDK 编程指导手册

（智能楼宇分册）



V2.0.0

前言

概述

欢迎使用 NetSDK（以下简称 SDK）编程指导手册。

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》，基础核心业务流程（例如初始化、登录、普通报警和智能报警等）详见《NetSDK 编程指导手册》。

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志，代表的含义如下。

符号	说明
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
V2.0.0	基础核心内容参考《NetSDK 编程指导手册》	2025.02
V1.0.5	全文优化语言。	2023.02
V1.0.4	删除 avnetsdk、fisheye 依赖库信息。 新增 StreamConvertor 依赖库	2021.05
V1.0.3	合入门禁控制器/人脸一体机（二代）功能和接口说明。	2020.07
V1.0.2	修改门禁设备型号、新增一代门禁相关功能、 设备登录接口全部替换成安全登录接口。	2020.04
V1.0.1	删除表 1-1 中的一部分内容。	2019.01
V1.0.0	首次发布。	2017.12

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
防区	报警输入通道可接收外部探测的信号，每一路报警输入通道即成为一个防区。
布防和撤防	● 布防：设备已布防的防区处于警戒状态，接收外部信号、处理、记录和转发。 ● 撤防：设备已撤防的防区不会接收外部报警信号、也不会处理、记录和转发。
旁路	在设备处于布防时，某个防区对外部探测器仍然监听，也会做记录，但不会转发给用户。在设备撤防后，原来旁路的防区会转变成正常状态，当再次布防时，该防区布防成功。
消警	设备发出报警后，通常会执行一些联动操作，比如蜂鸣、报警键盘提示等，这些操作往往会持续一段时间，消警是使这些联动操作提前结束。
即时防区	在设备布防状态下，该防区触发报警时，立刻记录并转发报警信号。
延时防区	防区类型为延时防区时，可设置进入延时或者退出延时。 进入延时，即用户在延时时间内从非布防区域进入布防区域时产生报警，但不联动报警，延时时间结束后，如果没撤防，此时会联动报警；如果撤防了，就不会联动报警。设置退出延时后，设备会在退出延时结束后再进入布防状态。
24 小时防区	一经配置成 24 小时防区，立即生效，且布撤防动作对其无效，可应用于火警等场景。
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”，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，门禁系统识别出是胁迫卡刷卡，并发出报警信号。

目录

前言 I
名词解释 II
第 1 章 产品概述 1
1.1 概述 1
1.2 适用性 2
1.2.1 适用系统 2
1.2.2 支持设备 2
1.3 应用场景 3
第 2 章 主要功能 6
2.1 通用 6
2.1.1 SDK 初始化 6
2.1.2 设备初始化 7
2.1.3 设备登录 12
2.1.4 实时预览 14
2.1.5 语音对讲 19
2.1.6 事件侦听 22
2.1.7 智能事件订阅 24
2.2 报警主机 26
2.2.1 布撤防 26
2.2.2 设置防区状态 28
2.2.3 防区状态查询 29
2.3 门禁控制器/指纹一体机（一代） 31
2.3.1 门禁控制 32
2.3.2 报警事件 34
2.3.3 设备信息查看 38
2.3.4 网络配置 42
2.3.5 设备时间设置 45
2.3.6 维护配置 50
2.3.7 人员管理 59
2.3.8 门配置 62
2.3.9 门时间配置 65
2.3.10 门高级配置 74
2.3.11 记录查询 89
2.4 门禁控制器/人脸一体机（二代） 95
2.4.1 门禁控制 96
2.4.2 报警事件 96
2.4.3 设备信息查看 96
2.4.4 网络配置 98
2.4.5 设备时间设置 98
2.4.6 维护配置 98
2.4.7 人员管理 98
2.4.8 门配置 117
2.4.9 门时间管理 117

2.4.10 门高级配置 121
2.4.11 记录查询 121
第 3 章 接口函数 125
3.1 通用接口 125
3.1.1 SDK 初始化 125
3.1.2 设备初始化 126
3.1.3 设备登录 129
3.1.4 实时预览 130
3.1.5 设备控制 132
3.1.6 报警侦听 133
3.1.7 智能事件订阅 134
3.1.8 获取设备状态 135
3.1.9 语音对讲 136
3.2 报警主机 139
3.3 门禁控制器/指纹一体机（一代） 139
3.3.1 门禁控制 139
3.3.2 报警事件 139
3.3.3 设备信息查看 139
3.3.4 网络配置 142
3.3.5 时间设置 145
3.3.6 维护配置 146
3.3.7 人员管理 151
3.3.8 门配置 151
3.3.9 门时间配置 152
3.3.10 门高级配置 153
3.3.11 记录查询 156
3.4 门禁控制器/人脸一体机（二代） 158
3.4.1 门禁控制 158
3.4.2 报警事件 158
3.4.3 设备信息查看 158
3.4.4 网络配置 159
3.4.5 时间设置 159
3.4.6 维护配置 159
3.4.7 人员管理 160
3.4.8 门配置 165
3.4.9 门时间配置 166
3.4.10 门高级配置 169
3.4.11 记录查询 169
第 4 章 回调函数 171
4.1 搜索设备回调函数 fSearchDevicesCB 171
4.2 异步搜索设备回调函数 fSearchDevicesCBEx 171
4.3 断线回调函数 fDisConnect 171
4.4 断线重连回调函数 fHaveReConnect 172
4.5 实时预览数据回调函数 fRealDataCallBackEx2 172
4.6 音频数据回调函数 pfAudioDataCallBack 173
4.7 报警回调函数 fMessCallBack 174
4.8 升级进度回调函数 fUpgradeCallBackEx 176

附录 1 法律声明 178
附录 2 网络安全建议 179

第 1 章 产品概述

1.1 概述

本文档主要介绍 SDK 接口参考信息，包括主要功能、接口函数和回调函数。

主要功能包括：通用功能、报警主机、门禁等功能。

根据环境不同，开发包包含的文件会不同，具体如下所示。

- Windows 开发包所包含的文件，请参见表 1-1。

表1-1 Windows 开发包包括的文件

库类型	库文件名称	库文件说明
功能库	dhnetsdk.h	头文件
	dhnetsdk.lib	Lib 文件
	dhnetsdk.dll	库文件
	avnetsdk.dll	库文件
配置库	avglobal.h	头文件
	dhconfigsdk.h	配置头文件
	dhconfigsdk.lib	Lib 文件
	dhconfigsdk.dll	库文件
播放（编码解码）辅助库	dhplay.dll	大华播放库
dhnetsdk 辅助库	IvsDrawer.dll	图像显示库
	StreamConvertor.dll	转码库

- Linux 开发包包含的文件，请参见表 1-2。

表1-2 Linux 开发包包括的文件

库类型	库文件名称	库文件说明
功能库	dhnetsdk.h	头文件
	libdhnetsdk.so	库文件
	libavnetsdk.so	库文件
配置库	avglobal.h	头文件
	dhconfigsdk.h	配置头文件
	libdhconfigsdk.so	配置库
libdhnetsdk.so 辅助库	libStreamConvertor.so	转码库

说明

- SDK 的功能库和配置库是必备库。
- 功能库是设备网络 SDK 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。
- 配置库针对配置功能的结构体进行打包和解析。
- 推荐使用播放库进行码流解析和播放。
- 辅助库用于预览、回放、对讲等功能的音视频码流解码以及本地音频采集。

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 系统。

1.2.2 支持设备

- 门禁设备（一代设备）
  - ◇ DH-ASC1201C-D
  - ◇ DH-ASC1202B-D、DH-ASC1202B-S、DH-ASC1202C-D、DH-ASC1202C-S
  - ◇ DH-ASC1204B-S、DH-ASC1204C-D、DH-ASC1204C-S
  - ◇ DH-ASC1208C-S
  - ◇ DH-ASI1201A、DH-ASI1201A-D、DH-ASI1201E-D、DH-ASI1201E
  - ◇ DH-ASI1212A(V2) 、 DH-ASI1212A-C(V2) 、 DH-ASI1212A-D(V2) 、 DH-ASI1212D 、 DH-ASI1212D-D
  - ◇ DHI-ASC1201B-D、DHI-ASC1201C-D
  - ◇ DHI-ASC1202B-D、DHI-ASC1202B-S、DHI-ASC1202C-D、DHI-ASC1202C-S
  - ◇ DHI-ASC1204B-S、DHI-ASC1204C-D、DHI-ASC1204C-S
  - ◇ DHI-ASC1208C-S
  - ◇ DHI-ASI1201A、DHI-ASI1201A-D、DHI-ASI1201E-D、DHI-ASI1201E
  - ◇ DHI-ASI1212A(V2)、DHI-ASI1212A-D(V2)、DHI-ASI1212D、DHI-ASI1212D-D
  - ◇ 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)、ASI1212D、ASI1212D-D
- 门禁设备（二代设备）
  - ◇ DH-ASI4213Y
  - ◇ DH-ASI4214Y
  - ◇ DH-ASI7213X、DH-ASI7213X-C、DH-ASI7213Y、DH-ASI7213Y-V3
  - ◇ DH-ASI7214X、DH-ASI7214X-C、DH-ASI7214Y、DH-ASI7214Y-V3
  - ◇ DH-ASI7223X-A、DH-ASI7223Y-A、DH-ASI7223Y-A-V3
  - ◇ DH-ASI8213Y(V2)、DH-ASI8213Y-C(V2)、DH-ASI8213Y-V3
  - ◇ DH-ASI8214Y、DH-ASI8214Y(V2)、DH-ASI8214Y-C(V2)、DH-ASI8214Y-V3
  - ◇ DH-ASI8215Y、DH-ASI8215Y(V2)、DH-ASI8215Y-V3
  - ◇ DH-ASI8223Y(V2)、DH-ASI8223Y-A(V2)、DH-ASI8223Y、DH-ASI8233Y-A-V3
  - ◇ DHI-ASI1202M、DHI-ASI1202M-D
  - ◇ DHI-ASI4213Y、DHI-ASI4214Y

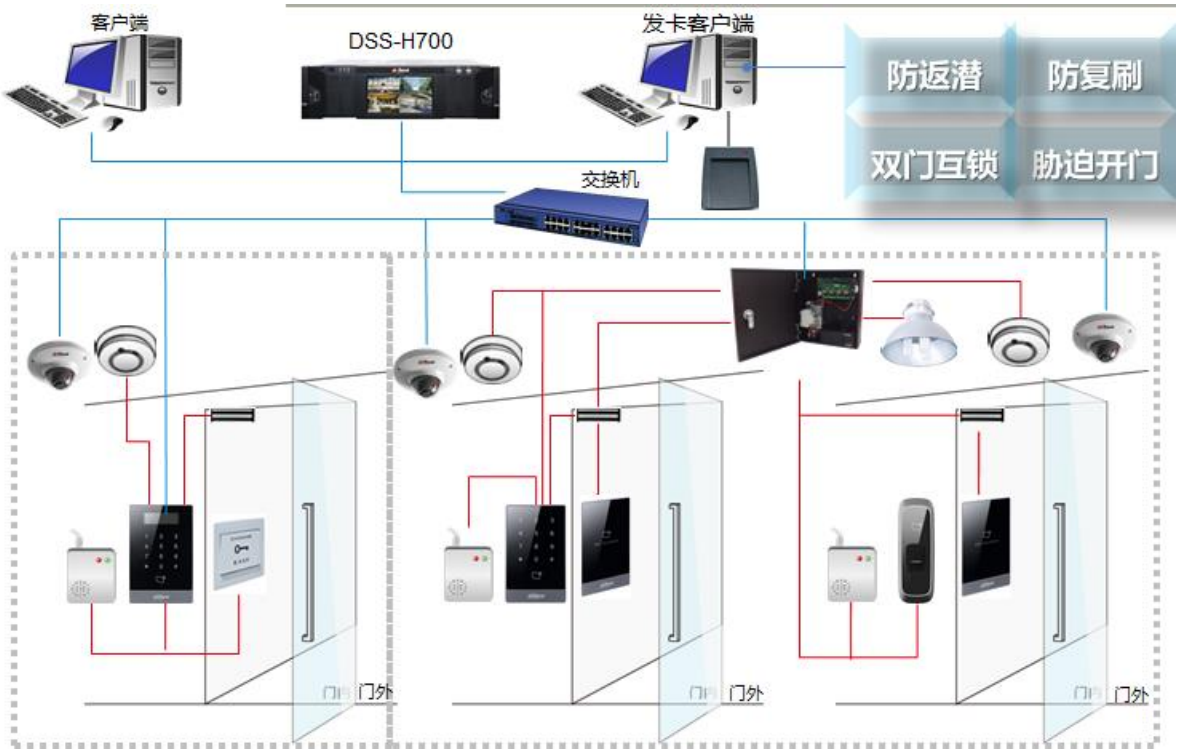
- ◇ DHI-ASI7213X、DHI-ASI7213Y、DHI-ASI7213Y-D、DHI-ASI7213Y-V3
- ◇ DHI-ASI7214X、DHI-ASI7214Y、DHI-ASI7214Y-D、DHI-ASI7214Y-V3
- ◇ DHI-ASI7223X-A、DHI-ASI7223Y-A、DHI-ASI7223Y-A-V3
- ◇ DHI-ASI8213Y-V3
- ◇ DHI-ASI8214Y、DHI-ASI8214Y(V2)、DHI-ASI8214Y-V3
- ◇ DHI-ASI8223Y、ASI8223Y(V2)、DHI-ASI8223Y-A(V2)、DHI-ASI8223Y-A-V3
- ◇ 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
- 报警主机
  - ◇ ARC2008C、ARC2008C-G、ARC2016C、ARC2016C-G、ARC5408C、ARC5408C-C、ARC5808C、ARC5808C-C、ARC9016C、ARC9016C-G
  - ◇ DH-ARC2008C、DH-ARC2008C-G、DH-ARC2016C、DH-ARC2016C-G、DH-ARC5408C、DH-ARC5408C-C、DH-ARC5408C-E、DH-ARC5808C、DH-ARC5808C-C、DH-ARC5808C-E、DH-ARC9016C、DH-ARC9016C-G、
  - ◇ DHI-ARC2008C、DHI-ARC2008C-G、DHI-ARC2016C、DHI-ARC2016C-G、DHI-ARC5808C、DHI-ARC5808C-C、DHI-ARC5408C、DHI-ARC5408C-C、DHI-ARC9016C、DHI-ARC9016C-G、
  - ◇ ARC2008C、ARC2008C-G、ARC2016C、ARC2016C-G、ARC5408C、ARC5408C-C、ARC5408C-E、ARC5808C-C、ARC5808C、ARC5808C-E、ARC9016C、ARC9016C-G

1.3 应用场景

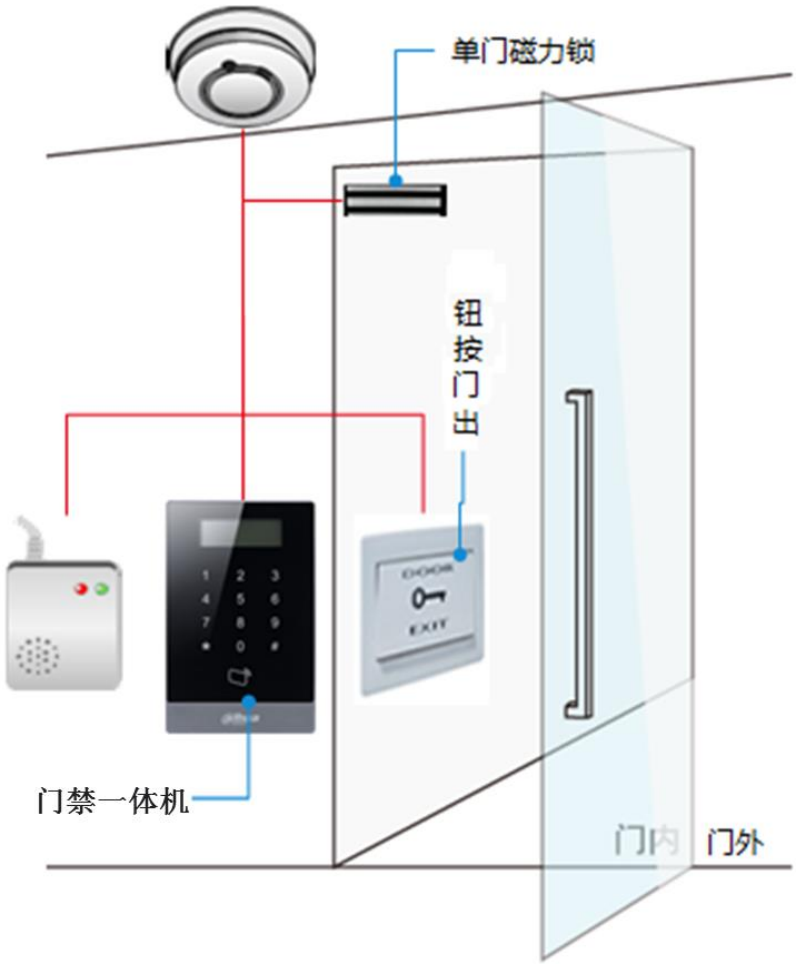
- 典型场景。

图1-1 典型场景



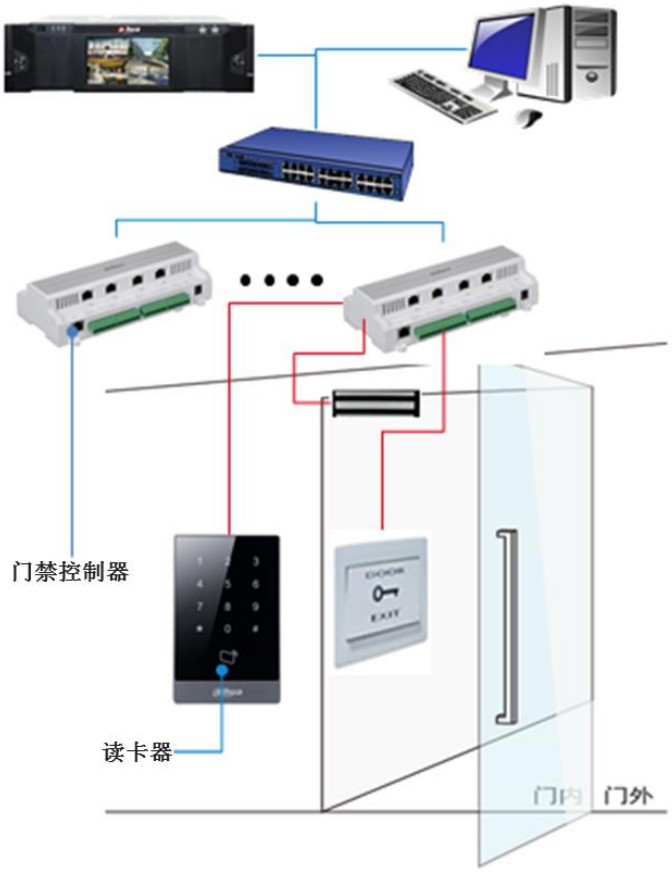
- 微型门禁适用于小型办公。

图1-2 微型门禁



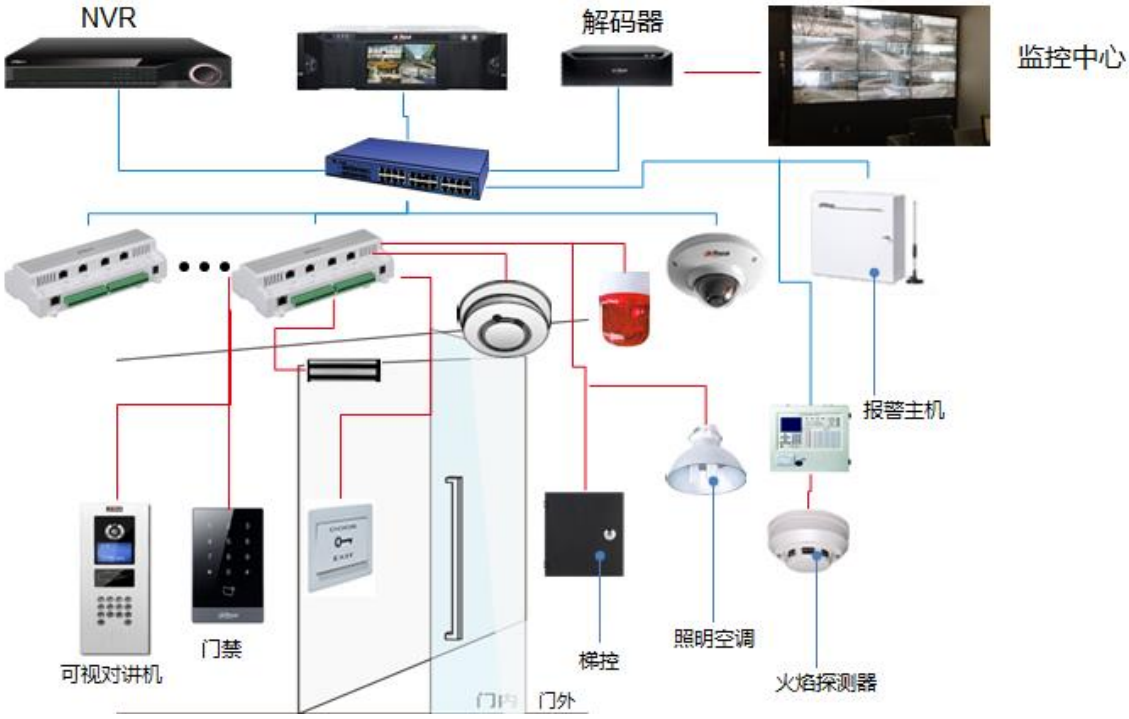
- 网络型门禁适用于中小型以上智能楼宇项目及金库和监所项目。

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 SDK 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。

- SDK 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 CLIENT_Cleanup 释放资源。

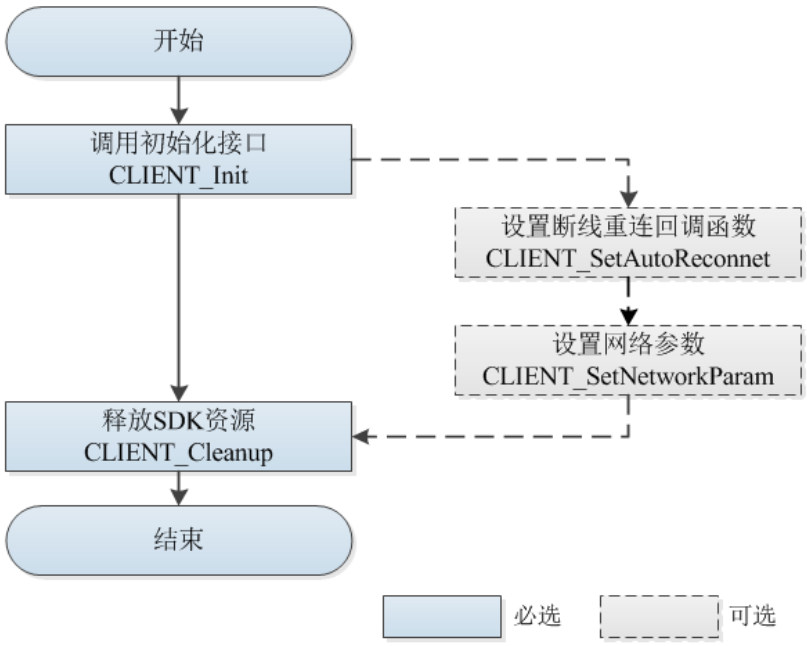
2.1.1.2 接口总览

表2-1 SDK 初始化接口说明

接口	说明
CLIENT_Init	SDK 初始化接口
CLIENT_Cleanup	SDK 清理接口
CLIENT_SetAutoReconnect	设置断线重连回调接口
CLIENT_SetNetworkParam	设置登录网络环境接口

2.1.1.3 流程说明

图2-1 SDK 初始化业务流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。

步骤2　（可选）调用 CLIENT_SetAutoReconnect 设置断线重连回调函数，设置后 SDK 内部断线自动重连。

步骤3　（可选）调用 CLIENT_SetNetworkParam 设置网络登录参数，参数中包含登录设备超时时间和尝试次数。

步骤4　SDK 所有功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- SDK 的 CLIENT_Init 和 CLIENT_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：CLIENT_Init 接口内部多次调用时，仅在内部用做计数，不会重复申请资源。
- 清理：CLIENT_Cleanup 接口内会清理所有已开启的业务，如登录、实时预览和报警订阅等。
- 断线重连：SDK 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 SDK 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预览、报警和智能图片订阅业务，其他业务无法恢复。

2.1.1.4 示例代码

```
//通过 CLIENT_Init 设置该回调函数，当设备出现断线时，SDK 通过该函数通知用户
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
{
    printf("Call DisConnectFunc: lLoginID[0x%x]\n", lLoginID);
}
//初始化 SDK
CLIENT_Init(DisConnectFunc, 0);

//调用功能接口处理业务

//清理 SDK 资源
CLIENT_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

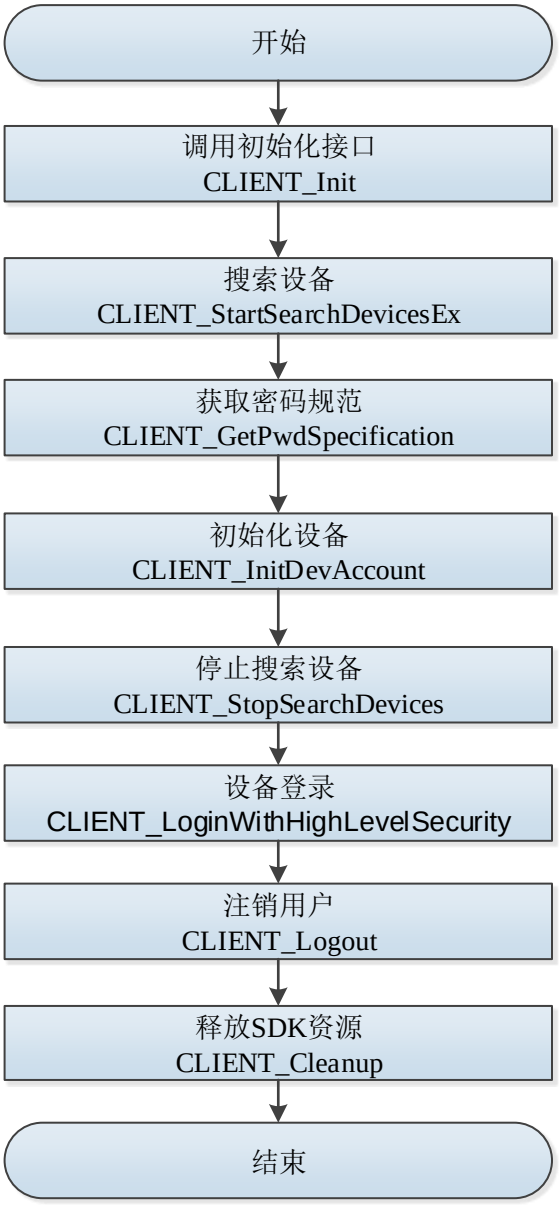
表2-2 设备初始化接口说明

接口	说明
CLIENT_StartSearchDevicesEx	搜索局域网内的设备，找到未初始化设备
CLIENT_InitDevAccount	设备初始化接口。
CLIENT_GetDescriptionForResetPwd	获取密码重置信息：手机号、邮箱和二维码信息。
CLIENT_CheckAuthCode	校验安全码是否有效。
CLIENT_ResetPwd	重置密码。
CLIENT_GetPwdSpecification	获取密码规则。
CLIENT_StopSearchDevices	停止搜索设备。

2.1.2.3 流程说明

2.1.2.3.1 设备初始化

图2-2 设备初始化流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。

步骤2　调用 CLIENT_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 CLIENT_GetPwdSpecification 接口获取设备的密码规则，依照规则确定需要设置的密码格式。

步骤4　调用 CLIENT_InitDevAccount 初始化设备。

步骤5　调用 CLIENT_StopSearchDevices 停止设备的搜索。

步骤6　调用 CLIENT_LoginWithHighLevelSecurity，使用 admin 帐户和设置的密码登录设备。

步骤7　业务使用完后，调用 CLIENT_Logout 登出设备。

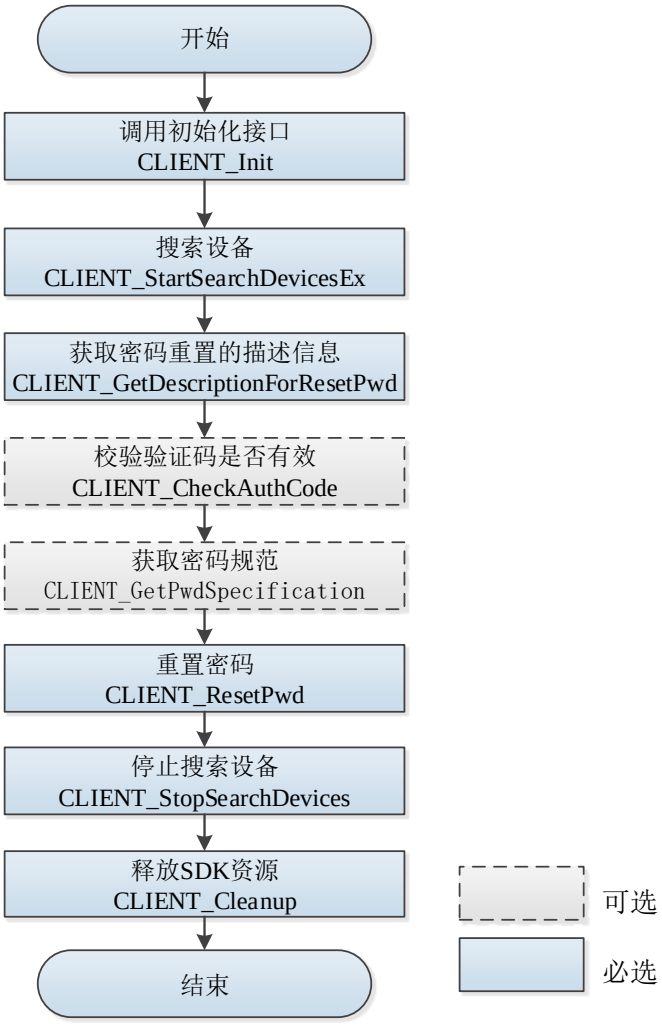
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

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。

2.1.2.3.2 重置密码

图2-3 重置密码及验证流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。
步骤2　调用 CLIENT_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 CLIENT_GetDescriptionForResetPwd 获取重置密码的描述信息。
步骤4　（可选）指定方式扫描上一步骤中获取的二维码，获取重置密码的安全码，通过 CLIENT_CheckAuthCode 校验安全码。
步骤5　（可选）使用 CLIENT_GetPwdSpecification 获取密码规则。
步骤6　使用 CLIENT_ResetPwd 重置密码。
步骤7　调用 CLIENT_StopSearchDevices 停止设备的搜索。
步骤8　调用 CLIENT_LoginWithHighLevelSecurity，使用 admin 帐户和已重置的密码登录设备。
步骤9　业务使用完后，调用 CLIENT_Logout 登出设备。
步骤10　SDK 功能使用完后，调用 CLIENT_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

2.1.2.4.1 设备初始化示例代码

```
//首先调用接口 CLIENT_StartSearchDevicesEx，在回调函数中获取设备信息
//获取密码规则
NET_IN_PWD_SPECI stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1);
NET_OUT_PWD_SPECI stOut = {sizeof(stOut)};
CLIENT_GetPwdSpecification(&stIn, &stOut, 3000, NULL);//在单网卡的情况下最后一个参数可以不填；在多网卡的情况下，最后一个参数填主机 IP。可根据已获取的设备密码规则，设置符合规则的密码，此步骤主要是防止客户设置一些设备不支持的密码格式。


//设备初始化
NET_IN_INIT_DEVICE_ACCOUNT sInitAccountIn = {sizeof(sInitAccountIn)};
NET_OUT_INIT_DEVICE_ACCOUNT sInitAccountOut = {sizeof(sInitAccountOut)};
sInitAccountIn.byPwdResetWay = 1;//1 为手机号重置方式，2 为邮箱重置方式
strncpy(sInitAccountIn.szMac, szMac, sizeof(sInitAccountIn.szMac) - 1);//设置 mac
strncpy(sInitAccountIn.szUserName, szUserName, sizeof(sInitAccountIn.szUserName) - 1);//设置用户名
strncpy(sInitAccountIn.szPwd, szPwd, sizeof(sInitAccountIn.szPwd) - 1);//设置密码
strncpy(sInitAccountIn.szCellPhone, szRig, sizeof(sInitAccountIn.szCellPhone) - 1);//由于 byPwdResetWay 设置为 1，此处需要设置 szCellPhone 字段；如果 byPwdResetWay 设置为 2，则需要设置 sInitAccountIn.szMail。
CLIENT_InitDevAccount(&sInitAccountIn, &sInitAccountOut, 5000, NULL);
```

2.1.2.4.2 重置密码示例代码

```
//首先调用接口 CLIENT_StartSearchDevicesEx，在回调函数中获取设备信息
//获取密码重置的描述信息
NET_IN_DESCRIPTION_FOR_RESET_PWD stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1); //设置 mac 值
strncpy(stIn.szUserName, szUserName, sizeof(stIn.szUserName) - 1);//设置用户名
stIn.byInitStatus = bStstus; //bStstus 为搜索设备接口(CLIENT_SearchDevices、CLIENT_StartSearchDevices、CLIENT_StartSearchDevicesEx 的回调函数和 CLIENT_SearchDevicesByIPs)返回字段 byInitStatus 的值
NET_OUT_DESCRIPTION_FOR_RESET_PWD stOut = {sizeof(stOut)};
char szTemp[360];
stOut.pQrCode = szTemp;
CLIENT_GetDescriptionForResetPwd(&stIn, &stOut, 3000, NULL);//在单网卡的情况下最后一个参数可以不填；在多网卡的情况下，最后一个参数填主机 IP。接口执行成功后，stOut 会输出一个二维码，二维码信息地址为 stOut.pQrCode，扫描此二维码，获取重置密码的安全码，此安全码会发送到预留手机号或者邮箱里
//(可选)校验安全码
NET_IN_CHECK_AUTHCODE stIn1 = {sizeof(stIn1)};
strncpy(stIn1.szMac, szMac, sizeof(stIn1.szMac) - 1); //设置 mac
strncpy(stIn1.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); // szSecu 为上一步骤中发送到预留手机号或者邮箱里的安全码
NET_OUT_CHECK_AUTHCODE stOut1 = {sizeof(stOut1)};
bRet = CLIENT_CheckAuthCode(&stIn1, &stOut1, 3000, NULL); //在单网卡的情况下最后一个参数可以不填；在多网卡的情况下，最后一个参数填主机 IP
//获取密码规则
NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};
strncpy(stIn2.szMac, szMac, sizeof(stIn2.szMac) - 1); //设置 mac
NET_OUT_PWD_SPECI stOut2 = {sizeof(stOut2)};
CLIENT_GetPwdSpecification(&stIn2, &stOut2, 3000, NULL);//在单网卡的情况下最后一个参数可以不填；在多网卡的情况下，最后一个参数填主机 IP。获取成功的情况下，可根据获取出的设备密码规则设置符合规则的密码，此步骤主要是防止客户设置一些设备不支持的密码格式
//重置密码
NET_IN_RESET_PWD stIn3 = {sizeof(stIn3)};
strncpy(stIn3.szMac, szMac, sizeof(stIn3.szMac) - 1); //设置 mac 值
strncpy(stIn3.szUserName, szUserName, sizeof(stIn3.szUserName) - 1); //设置用户名
strncpy(stIn3.szPwd, szPassWd, sizeof(stIn3.szPwd) - 1); //szPassWd 为符合密码规则的重置密码
strncpy(stIn3.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); // szSecu 为扫描二维码后发送到预留手机号或者邮箱里的安全码
stIn3.byInitStaus = bStstus; //bStstus 为搜索设备接口(CLIENT_SearchDevices、CLIENT_StartSearchDevices、CLIENT_StartSearchDevicesEx 的回调函数和 CLIENT_SearchDevicesByIPs)返回字段 byInitStatus 的值
stIn3.byPwdResetWay = bPwdResetWay; //bPwdResetWay 为搜索设备接口(CLIENT_SearchDevices、CLIENT_StartSearchDevices、CLIENT_StartSearchDevicesEx 的回调函数和 CLIENT_SearchDevicesByIPs)返
```

回字段 byPwdResetWay 的值

NET_OUT_RESET_PWD stOut3 = {sizeof(stOut3)};

CLIENT_ResetPwd(&stIn3, &stOut3, 3000, NULL);// 在单网卡的情况下最后一个参数可以不填；在多网卡的情况下，最后一个参数填主机 IP

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 SDK 接口需要传入登录 ID 才可执行。登出设备后，登录 ID 失效。

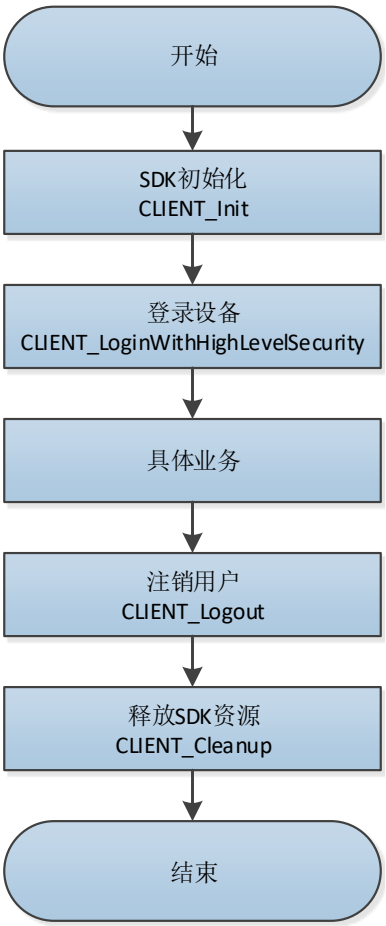
2.1.3.2 接口总览

表2-3 设备登录接口说明

接口	说明
CLIENT_LoginWithHighLevelSecurity	高安全级别登录接口。 **说明** CLINET_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 CLIENT_LoginWithHighLevelSecurity 登录设备。
CLIENT_Logout	登出接口。

2.1.3.3 流程说明

图2-4 登录业务流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。
步骤2　调用 CLIENT_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 CLIENT_Logout 登出设备。
步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 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。
- 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。例如打开预览后，在不需要使用预览时，用户应该调用结束预览的接口。
- 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：建议通过登录接口的 error 参数（登录错误码）初步排查。常见错误码如表 2-4 所示。

表2-4 常见错误码

error 错误码	对应的含义
1	密码不正确。
2	用户名不存在。
3	登录超时。
4	账号已登录。
5	账号已被锁定。
6	账号被列为黑名单。
7	资源不足，设备系统忙。
8	子连接失败。
9	主连接失败。
10	超过最大用户连接数。
11	缺少 avnetsdk 或 avnetsdk 的依赖库。
12	设备未插入 U 盘或 U 盘信息错误。
13	客户端 IP 地址没有登录权限。

更多错误码信息，请参见《网络 SDK 开发手册》中的“CLIENT_LoginWithHighLevelSecurity 接口”描述。其中错误码 3 规避示例代码如下：

```
NET_PARAM stuNetParam = {0};
stuNetParam.nWaittime = 8000;
CLIENT_SetNetworkParam (&stuNetParam);
```

2.1.3.4 示例代码

```
NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stuIn = {sizeof(stuIn)};
strncpy(stuIn.szIP, pchDVRIP, 63);
stuIn.nPort = wDVRPort;
strncpy(stuIn.szUserName, pchUserName, 63);
strncpy(stuIn.szPassword, pchPassword, 63);
stuIn.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
stuIn.pCapParam = NULL;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stuOut = {sizeof(stuOut)};
// 登录设备
LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stuIn, &stuOut);
// 登出设备
CLIENT_Logout(lLoginHandle);
```

2.1.4 实时预览

2.1.4.1 简介

实时预览，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2-5 实时预览接口说明

接口	说明
CLIENT_RealPlayEx	开始实时预览扩展接口。
CLIENT_StopRealPlayEx	停止实时预览扩展接口。
CLIENT_SaveRealData	开始本地保存实时预览数据。
CLIENT_StopSaveRealData	停止本地保存实时预览数据。
CLIENT_SetRealDataCallBackEx2	设置实时预览数据回调函数扩展接口。

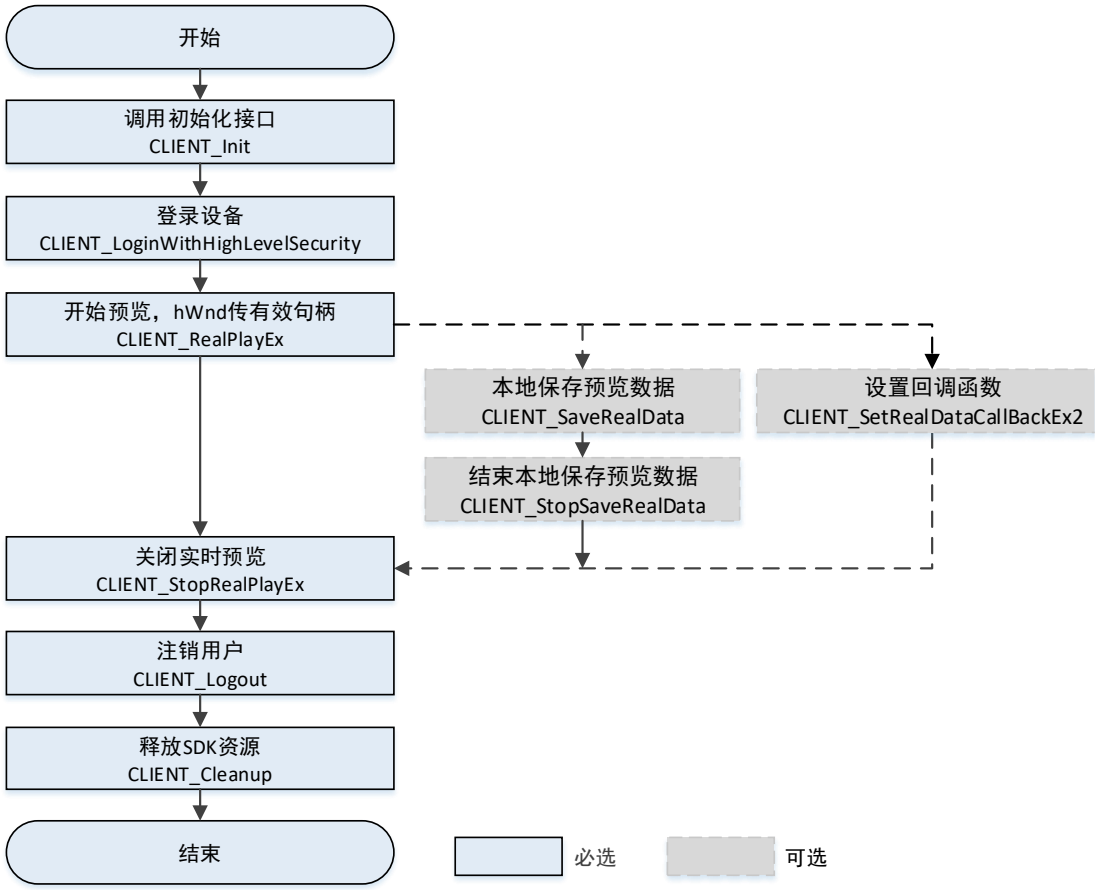
2.1.4.3 流程说明

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

SDK 内部调用辅助库里的 PlaySDK 库实现实时播放。

图2-5 SDK 解码播放流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
步骤3　调用 CLIENT_RealPlayEx 启动实时预览，参数 **hWnd** 为有效窗口句柄。
步骤4　（可选）调用 CLIENT_SaveRealData 开始保存预览数据。
步骤5　（可选）调用 CLIENT_StopSaveRealData 结束保存，生成本地视频文件。
步骤6　（可选）若调用 CLIENT_SetRealDataCallBackEx2，用户可将视频数据选择保存或转发。若保存成文件，与步骤 4、5 效果相同。
步骤7　实时预览使用完毕后，调用 CLIENT_StopRealPlayEx 停止实时预览。
步骤8　业务使用完后，调用 CLIENT_Logout 登出设备。
步骤9　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- SDK 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：接口内申请预览资源需和设备做一些约定，然后才请求预览数据，过程中有一些超时时间的设定（请参见 NET_PARAM 结构体），其中与预览相关的字段为 **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 CLIENT_Init 函数后调用，调用一次即可：

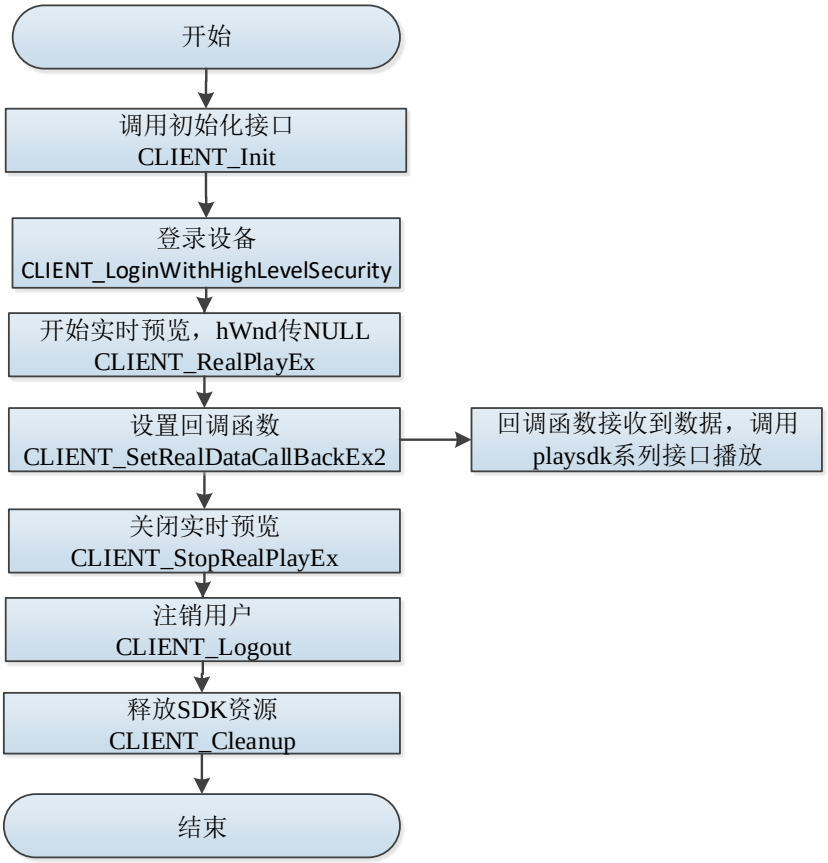
```
NET_PARAM stuNetParam = {0};
stuNetParam. nGetConnInfoTime = 5000; // 单位 ms
CLIENT_SetNetworkParam (&stuNetParam);
```

- 重复打开失败：部分设备不支持同一次登录下同一个通道多次打开，当重复打开同一通道的预览，可能会出现第一次打开成功，后续打开失败的现象。建议：
  - ◇ 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：SDK 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。
- 系统资源不足的情况下，设备可能返回错误而不恢复码流，可以在报警回调函数（即 CLIENT_SetDVRMessCallBack 中设置的回调函数）收到事件 DH_REALPLAY_FAILD_EVENT，该事件包含了详细的错误码，请参见《网络 SDK 开发手册》中的“DEV_PLAY_RESULT 结构体”。
- 32 路限制：解码显示比较消耗资源，特别是高分辨率视频，考虑到客户端硬件资源有限，一般同时解码显示的通道数有限，所以该方式暂时限定为最多 32 路，如超过 32 路，相关信息请参见“2.1.4.3.2 调用第三方解码播放库”。

2.1.4.3.2 调用第三方解码播放库

SDK 回调实时预览码流给用户，用户调用 PlaySDK 进行解码播放。

图2-6 第三方解码播放流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 登录设备。
步骤3　登录成功后，调用 CLIENT_RealPlayEx 启动实时预览，参数 **hWnd** 为 **NULL**。
步骤4　调用 CLIENT_SetRealDataCallBackEx2 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　实时预览使用完毕后，调用 CLIENT_StopRealPlayEx 停止实时预览。
步骤7　业务使用完后，调用 CLIENT_Logout 登出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 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值（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 3M。
  - ◇ SDK 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 SDK 解码播放

```
//以开启第一路的主码流预览为例，hWnd 为界面窗口句柄
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, hWnd, DH_RType_Realplay);
if (NULL == lRealHandle)
{
printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();
// 关闭预览
if (NULL != lRealHandle)
{
CLIENT_StopRealPlayEx(lRealHandle);
}
```

2.1.4.4.2 调用播放库

```
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LLONG param, LDWORD dwUser);
//以开启第一路的主码流预览为例
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, NULL, DH_RType_Realplay);
if (NULL == lRealHandle)
{
printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
else
{
DWORD dwFlag = REALDATA_FLAG_RAW_DATA; //原始数据标志
CLIENT_SetRealDataCallBackEx2(lRealHandle, &RealDataCallBackEx, NULL, dwFlag);
}

printf("input any key to quit!\n");
getchar();
// 关闭预览
if (0 != lRealHandle)
{
    CLIENT_StopRealPlayEx(lRealHandle);
}
```

```
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser)
{
    // 从设备获取的码流数据，需调用 PlaySDK 的接口，详细介绍请参见 SDK 预览 demo 源码
printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d]\n", lRealHandle, dwDataType, pBuffer, dwBufSize);
}
```

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 SDK 实现与设备的语音对讲。

2.1.5.2 接口总览

表2-6 语音对讲接口说明

接口	说明
CLIENT_StartTalkEx	打开语音对讲扩展接口
CLIENT_StopTalkEx	停止语音对讲扩展接口
CLIENT_RecordStartEx	开始客户端录音扩展接口（只在 Windows 平台下有效）
CLIENT_RecordStopEx	结束客户端录音扩展接口（只在 Windows 平台下有效）
CLIENT_TalkSendData	发送语音数据到设备
CLIENT_AudioDecEx	解码音频数据扩展接口（只在 Windows 平台下有效）
CLIENT_SetDeviceMode	设置设备语音对讲工作模式

2.1.5.3 流程说明

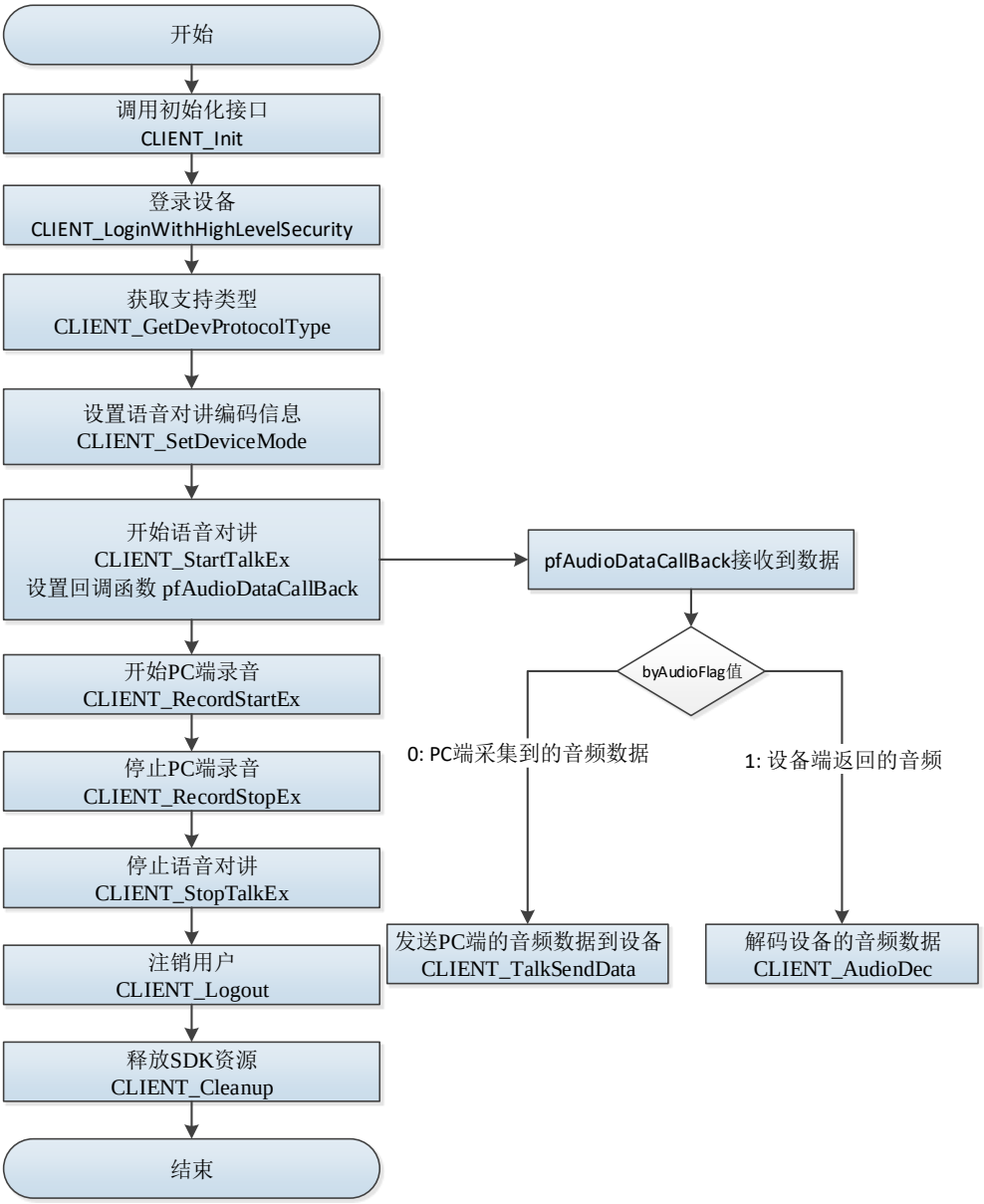
当 SDK 从本地声卡采集到音频数据或 SDK 接收到前端发送过来的音频数据时，会调用音频数据回调函数。用户可在回调函数中调用 SDK 接口将采集到的本地音频数据发送到前端设备，也可以调用 SDK 接口将接收到的前端设备的音频数据进行解码播放。语音对讲流程如图 2-7 所示。

该模式只在 Windows 平台下有效。

说明

目前语音对讲分为二代和三代语音对讲，可以使用接口 CLIENT_GetDevProtocolType 获取设备支持的语音对讲方式，二代和三代对讲流程相同，区别在于 CLIENT_SetDeviceMode 设置的对讲参数不同。

图2-7 二代语音对讲流程图



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。

步骤2　初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。

步骤3　调用 CLIENT_GetDevProtocolType 获取支持二代对讲或者三代对讲。

步骤4　调用 CLIENT_SetDeviceMode 设置语音对讲参数。

- 如果支持二代对讲：设置编码模式、客户端模式和喊话模式，参数 **emType** 设置为 **DH_TALK_ENCODE_TYPE**、**DH_TALK_CLIENT_MODE** 和 **DH_TALK_SPEAK_PARAM**。
- 如果支持三代对讲：设置编码模式、客户端模式和三代语音对讲参数，参数 **emType** 设置为 **DH_TALK_ENCODE_TYPE**、**DH_TALK_CLIENT_MODE** 和 **DH_TALK_MODE3**。

步骤5　调用 CLIENT_StartTalkEx 设置回调函数并开始语音对讲。在回调函数中，调用 CLIENT_AudioDec，解码设备发送过来的音频数据；调用 CLIENT_TalkSendData，发送 PC 端的音频数据到设备。

步骤6　调用 CLIENT_RecordStartEx 开始 PC 端录音，该接口调用后，CLIENT_StartTalkEx 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 CLIENT_RecordStopEx 停止 PC 端录音。

步骤8 调用 CLIENT_StopTalkEx 停止语音对讲。

步骤9 调用 CLIENT_Logout 登出设备。

步骤10 SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 语音编码格式：示例采用了常用的 PCM 格式，SDK 支持获取设备支持的语音编码格式，示例源码详细介绍请参见官网发布包。如果默认 PCM 能满足需求，建议不用获取设备支持的语音编码格式。
- 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 CLIENT_RecordStartEx 接口是否返回成功。

2.1.5.4 示例代码

```
// 获取设备支持二代还是三代语音对讲。
EM_DEV_PROTOCOL_TYPE emTpye = EM_DEV_PROTOCOL_UNKNOWN;
CLIENT_GetDevProtocolType(g_lLoginHandle, &emTpye);

DHDEV_TALKDECODE_INFO curTalkMode = {0};
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode); //设置对讲编码格式
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_CLIENT_MODE, NULL);// 设置客户端方式语音对讲

// 根据获取出的对讲类型，设置对讲参数
if (emTpye == EM_DEV_PROTOCOL_V3) //三代对讲需要设备此类参数，而二代设备不需要设置此类参数
{
NET_TALK_EX stuTalk = {sizeof(stuTalk)};
        stuTalk.nAudioPort    = RECEIVER_AUDIO_PORT; // 用户自定义接收端口
        stuTalk.nChannel  = 0;
        stuTalk.nWaitTime= 5000;
        CLIENT_SetDeviceMode(m_lLoginHandle, DH_TALK_MODE3, &stuTalk)
}
// 开始语音对讲
lTalkHandle = CLIENT_StartTalkEx(lLoginHandle, AudioDataCallBack, (LDWORD)NULL);
// 开始本地录音
CLIENT_RecordStartEx(lLoginHandle);
// 停止本地录音
```

```
CLIENT_RecordStopEx(lLoginHandle)
// 停止语音对讲
CLIENT_StopTalkEx(lTalkHandle);
// 语音对讲回调数据处理
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE
byAudioFlag, LDWORD dwUser)
{
if(0 == byAudioFlag)
{
        // 将收到的本地 PC 端检测到的声卡数据发送给设备端
        CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
    }
else if(1 == byAudioFlag)
    {
        // 将收到的设备端发送过来的语音数据传给 SDK 解码播放
        CLIENT_AudioDec(pDataBuf, dwBufSize);
    }
}
```

2.1.6 事件侦听

2.1.6.1 简介

报警上报实现方式为：通过 SDK 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 SDK，用户可通过报警回调函数获取相应的报警信息。

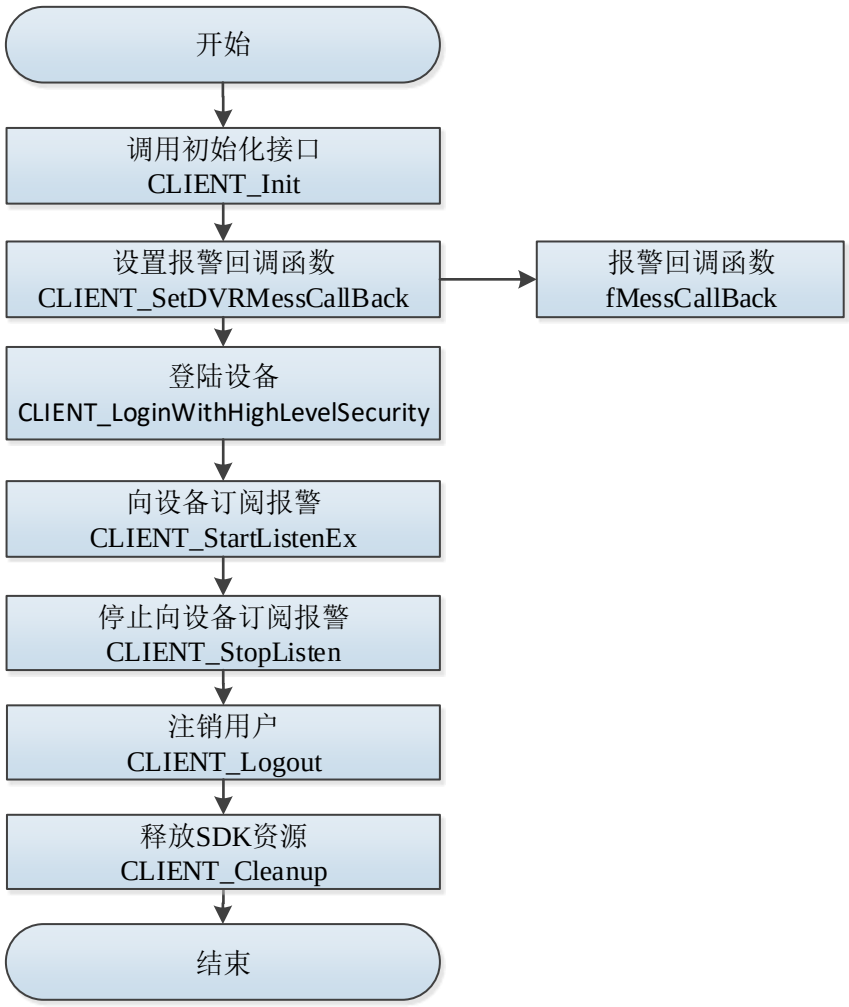
2.1.6.2 接口总览

表2-7 报警侦听上报的接口说明

接口	说明
CLIENT_SetDVRMessCallBack	设置报警回调函数接口。
CLIENT_StartListenEx	订阅报警扩展接口。
CLIENT_StopListen	停止订阅报警。

2.1.6.3 流程说明

图2-8 报警上报流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_SetDVRMessCallBack 函数，设置报警回调函数。

说明

该接口需在报警订阅之前调用。

步骤3　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤4　调用 CLIENT_StartListenEx 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 CLIENT_SetDVRMessCallBack 设置的回调函数通知用户。

说明

报警主机、门禁和语音对讲相关的报警事件，具体请参见“4.7 报警回调函数 fMessCallBack”。

步骤5　报警上报功能使用完毕后，调用 CLIENT_StopListen 函数停止向设备订阅报警。

步骤6　调用 CLIENT_Logout 函数登出设备。

步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 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，设备自动重连后是不会再有报警上报的，需要取消订阅后重新订阅。
- 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

```
// 报警回调
int CALLBACK    afMessCallBack(LONG lCommand, LLONG lLinID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
if(lCommand == DH_ALARM_ACCESS_CTL_EVENT) // 门禁事件，更多门禁相关事件请参见“4.7 报警回调函数 fMessCallBack”。
{
ALARM_ACCESS_CTL_EVENT_INFO* pstAccessInfo =
  (ALARM_ACCESS_CTL_EVENT_INFO*)pBuf;
      //接下来可以通过 pstAccessInfo 获取对应的报警信息。
}
……..
}
// 设置报警回调
CLIENT_SetDVRMessCallBack(afMessCallBack,0);
// 订阅报警
CLIENT_StartListenEx(lLoginHandle);
// 停止报警订阅
CLIENT_StopListen(lLoginHandle);
```

2.1.7 智能事件订阅

2.1.7.1 简介

智能事件订阅，即前端设备智能算法或者后端智能算法对码流实时进行分析，当检测到预先设定好的智能事件时，就将该事件和事件信息上报给用户。在本文档中的智能事件包括，通用行为分析智能事件（绊线入侵、区域入侵等智能事件）、目标检测、人体检测、智能交通的智能事件（卡口、超速、低速、交通拥堵等智能事件）。

2.1.7.2 接口总览

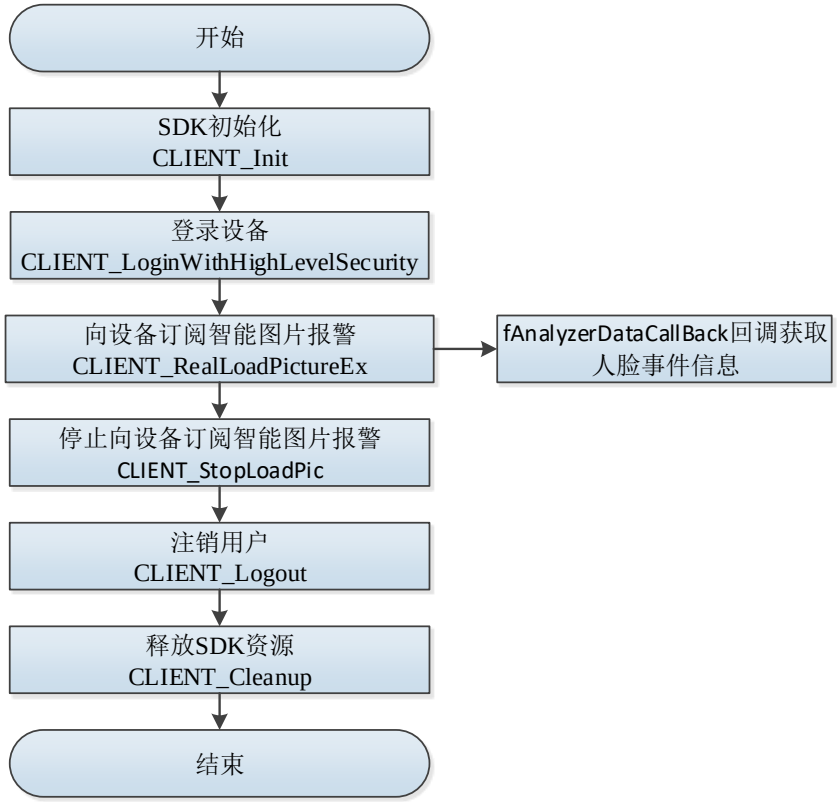
表2-8 智能事件订阅接口说明

接口	说明
CLIENT_RealLoadPictureEx	订阅智能事件
CLIENT_StopLoadPic	取消订阅智能事件

2.1.7.3 流程说明

人脸事件上报流程，如图 2-9 所示。

图2-9 人脸事件上报流程



流程说明

步骤1　调用 CLIENT_Init 完成 SDK 初始化流程。

步骤2　初始化成功后，调用 CLIENT_LoginWithHighLevelSecurity 登录设备。

步骤3　调用 CLIENT_RealLoadPictureEx 向设备订阅智能事件。

步骤4　订阅成功后，设备上报的智能事件通过 fAnalyzerDataCallBack 回调函数上报智能事件。通过该函数，可以根据报警类型过滤出您需要的智能报警事件。

步骤5　智能事件上报功能使用完毕后，调用 CLIENT_StopLoadPic 停止订阅智能事件。

步骤6　业务使用完后，调用 CLIENT_Logout 登出设备。

步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 释放 SDK 资源。

注意事项

- 订阅事件类型：如果需要同时上报不同智能事件时，支持订阅所有智能事件

（EVENT_IVS_ALL）；也支持订阅单个智能事件。

- 设置是否接收图片：由于一些设备所在网络环境是 3G 或 4G 网络，当 SDK 连接设备时，如不需要接收图片可以把 CLIENT_RealLoadPictureEx 接口中 bNeedPicFile 参数设置为 False，只接收人脸事件信息，不带图片。

2.1.7.4 示例代码

```
// 智能事件上报回调函数
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
switch(dwAlarmType)
{
// 过滤出你想要的智能事件
……
case EVENT_IVS_ACCESS_CTL:
……
default:
break;
}
}

// 订阅智能事件上报
LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, NULL, NULL);
if(NULL == lAnalyerHandle)
{
printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
return -1;
}

// 取消订阅智能事件上报
CLIENT_StopLoadPic(lAnalyerHandle);
```

2.2 报警主机

2.2.1 布撤防

2.2.1.1 简介

- 布防：设备所有防区处于警戒状态，接收外部信号，并可处理、记录和转发信号。
- 撤防：设备所有防区都不会接收外部报警信号，也不会处理、记录和转发信号。

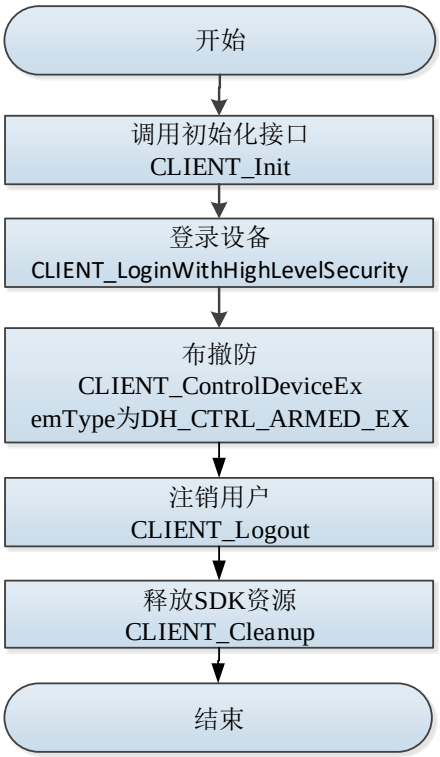
2.2.1.2 接口总览

表2-9 布撤防接口说明

接口	说明
CLIENT_ControlDeviceEx	设备控制扩展接口。

2.2.1.3 流程说明

图2-10 布撤防业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDeviceEx 函数来控制设备布防撤防，函数参数 **emType** 值为 **DH_CTRL_ARMED_EX**。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.2.1.4 示例代码

```
CTRL_ARM_DISARM_PARAM_EX stuParam = {sizeof(stuParam)};
stuParam.stuIn.dwSize = sizeof(stuParam.stuIn);
stuParam.stuOut.dwSize = sizeof(stuParam.stuOut);

stuParam.stuIn.emState = NET_ALARM_MODE_ARMING;
stuParam.stuIn.emSceneMode = NET_SCENE_MODE_OUTDOOR;
```

```
stuParam.stuIn.szDevPwd = "admin";
CLIENT_ControlDeviceEx(g_lLoginHandle, DH_CTRL_ARMED_EX, &stuParam, NULL,3000);
```

2.2.2 设置防区状态

2.2.2.1 简介

设置防区状态，控制设备报警通道的正常、旁路和隔离状态。

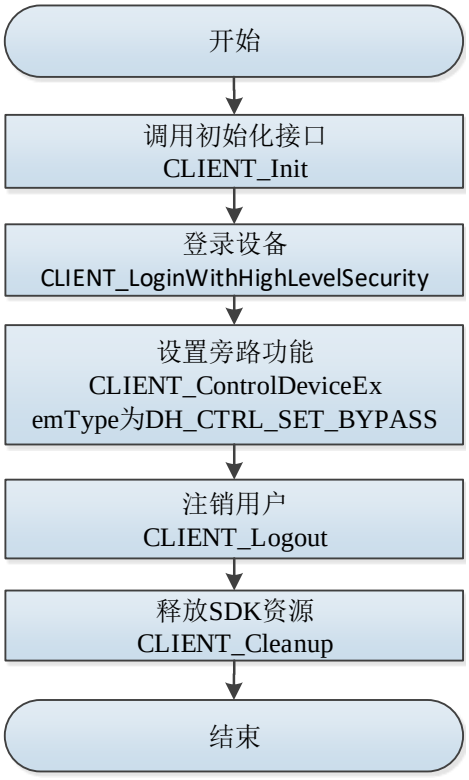
2.2.2.2 接口总览

表2-10 设置防区状态接口说明

接口	说明
CLIENT_ControlDeviceEx	设备控制扩展接口。

2.2.2.3 流程说明

图2-11 设置防区状态业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDeviceEx 函数来控制设备设置防区状态，函数参数 **emType** 值为 **DH_CTRL_SET_BYPASS**。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.2.2.4 示例代码

```
NET_CTRL_SET_BYPASS stuParam = {sizeof(stuParam)};
stuParam.dwSize = sizeof(stuParam);
stuParam.emMode = NET_BYPASS_MODE_BYPASS; //  设置防区状态为旁路状态
stuParam.szDevPwd = "admin";
stuParam.nExtendedCount = 1;
int nExtendChn[1] = {1};
stuParam.pnExtended = nExtendChn;
stuParam.nLocalCount = 2;
int nLocalChn[2] = {2,3};
stuParam.pnLocal = nLocalChn;
CLIENT_ControlDeviceEx(g_lLoginHandle, DH_CTRL_SET_BYPASS, &stuParam, NULL, 3000);
```

2.2.3 防区状态查询

2.2.3.1 简介

查询防区状态，包括报警输入状态、报警输出状态、警号状态等。

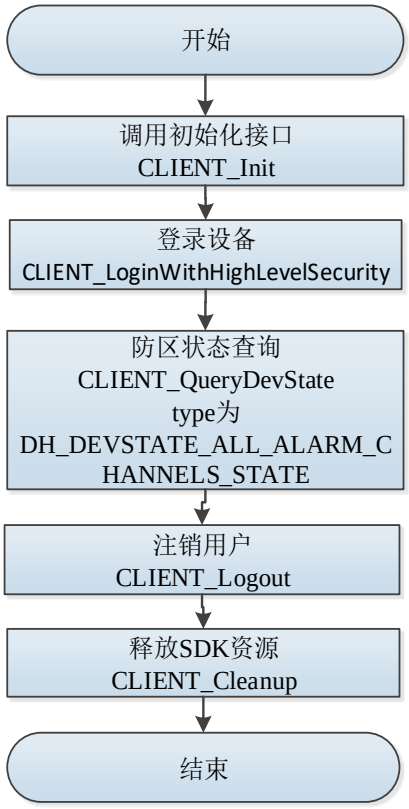
2.2.3.2 接口总览

表2-11 防区状态查询接口说明

接口	说明
CLIENT_QueryDevState	状态查询接口。

2.2.3.3 流程说明

图2-12 防区状态查询业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　业务实现，调用 CLIENT_QueryDevState 来实现防区状态查询，此时参数 **type** 为 **DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE**。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.2.3.4 示例代码

```
NET_CLIENT_ALARM_CHANNELS_STATE stuAlarmChannelState = {sizeof(stuAlarmChannelState)};
stuAlarmChannelState.emType = NET_ALARM_CHANNEL_TYPE_ALL; // 查询所有通道状态；
int nNum = 2;

// 警号通道相关字段初始化
stuAlarmChannelState.nAlarmBellCount = nNum;
stuAlarmChannelState.pbAlarmBellState = new BOOL[stuAlarmChannelState.nAlarmBellCount];
memset(stuAlarmChannelState.pbAlarmBellState, 0, stuAlarmChannelState.nAlarmBellCount *
sizeof(BOOL));
```

```
// 报警输入通道相关字段初始化
stuAlarmChannelState.nAlarmInCount = nNum;
stuAlarmChannelState.pbAlarmInState = new BOOL[stuAlarmChannelState.nAlarmInCount];
memset(stuAlarmChannelState.pbAlarmInState, 0, stuAlarmChannelState.nAlarmInCount * sizeof(BOOL));

// 报警输出通道相关字段初始化
stuAlarmChannelState.nAlarmOutCount = nNum;
stuAlarmChannelState.pbAlarmOutState = new BOOL[stuAlarmChannelState.nAlarmOutCount];
memset(stuAlarmChannelState.pbAlarmOutState, 0, stuAlarmChannelState.nAlarmOutCount *
sizeof(BOOL));

// 扩展模块报警输入通道相关字段初始化
stuAlarmChannelState.nExAlarmInCount = nNum;
stuAlarmChannelState.pbExAlarmInState = new BOOL[stuAlarmChannelState.nExAlarmInCount];
memset(stuAlarmChannelState.pbExAlarmInState, 0, stuAlarmChannelState.nExAlarmInCount *
sizeof(BOOL));
stuAlarmChannelState.pnExAlarmInDestionation = new int[1024];

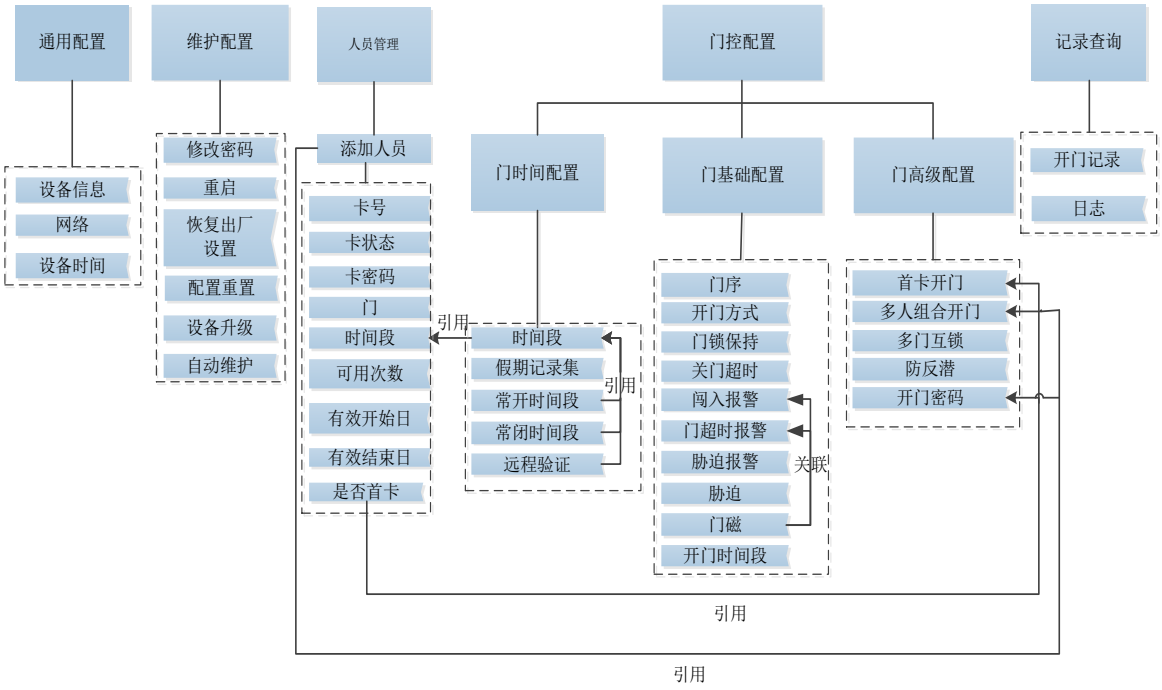
// 扩展模块报警输出通道相关字段初始化
stuAlarmChannelState.nExAlarmOutCount = nNum;
stuAlarmChannelState.pbExAlarmOutState = new BOOL[stuAlarmChannelState.nExAlarmOutCount];
memset(stuAlarmChannelState.pbExAlarmOutState, 0, stuAlarmChannelState.nExAlarmOutCount *
sizeof(BOOL));
stuAlarmChannelState.pnExAlarmOutDestionation = new int[1024];

int nRetLen = 0;
CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE,
(char*)&stuAlarmChannelState, sizeof(NET_CLIENT_ALARM_CHANNELS_STATE), &nRetLen, 3000);
```

2.3 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

图2-13 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

2.3.1.1 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。

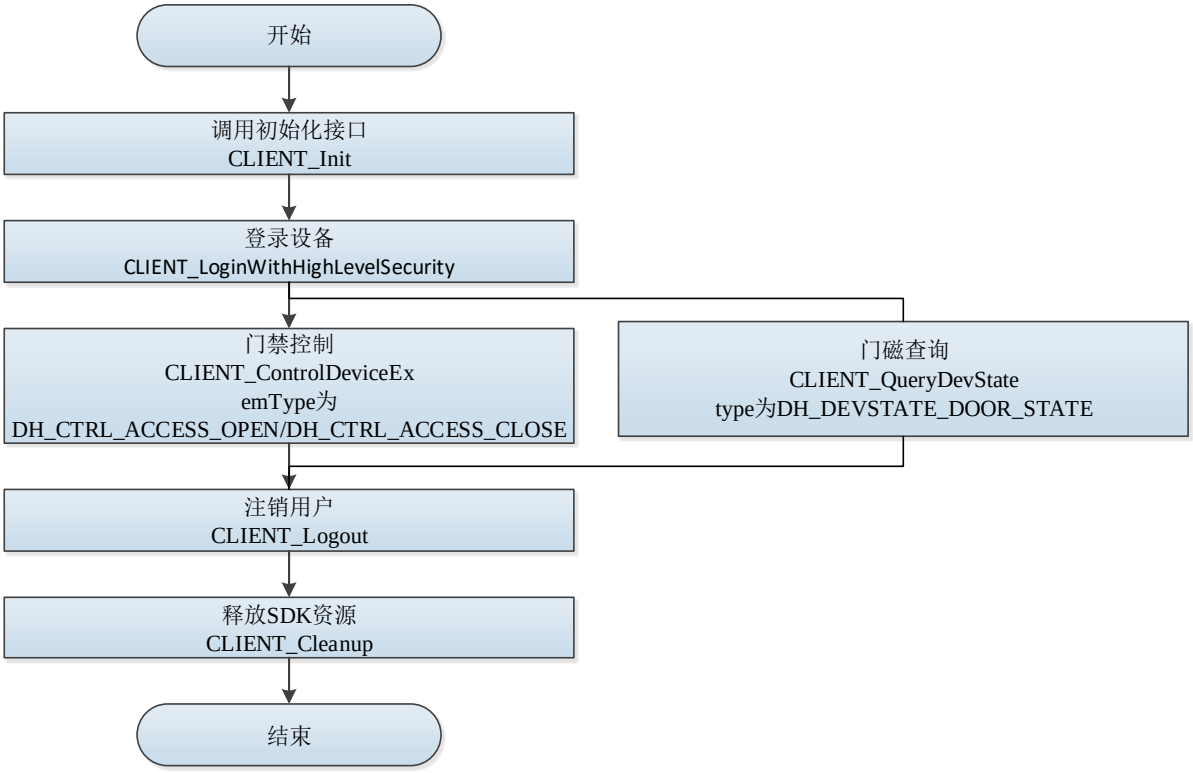
2.3.1.2 接口总览

表2-12 门禁控制接口说明

接口	说明
CLIENT_ControlDeviceEx	设备控制扩展接口。
CLIENT_QueryDevState	状态查询接口。

2.3.1.3 流程说明

图2-14 门禁控制业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDeviceEx 函数来控制门禁。

- 打开门禁：**emType** 值为 **DH_CTRL_ACCESS_OPEN**。
- 关闭门禁：**emType** 值为 **DH_CTRL_ACCESS_CLOSE**。

步骤4　业务实现，调用 CLIENT_QueryDevState 来实现门磁查询。

Type：DH_DEVSTATE_DOOR_STATE

pBuf：NET_DOOR_STATUS_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.1.4 示例代码

```
// 打开门禁
NET_CTRL_ACCESS_OPEN stOpen = {sizeof(stOpen)};
stOpen.nChannelID = 0;
strncpy(stOpen.szUserID, “admin”, sizeof(stOpen.szUserID) - 1);
CLIENT_ControlDeviceEx((LLONG)g_lLoginHandle, DH_CTRL_ACCESS_OPEN, &stOpen, NULL, 3000);


//关闭门禁
```

```
NET_CTRL_ACCESS_CLOSE stClose = {sizeof(stClose)};
CLIENT_ControlDeviceEx((LLONG)g_lLoginHandle, DH_CTRL_ACCESS_CLOSE, &stClose, NULL, 3000);

// 查询门磁状态信息
int nRet = 0;
    NET_DOOR_STATUS_INFO stuInfo = {sizeof(stuInfo)};
    stuInfo.nChannel = 0;
    BOOL bReturn = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DOOR_STATE, (char
*)&stuInfo, sizeof(stuInfo), &nRet, 5000);
    if (bReturn)
    {
        printf("门磁状态：%d\n",stuInfo.emStateType);
    }
    else{
        printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.2 报警事件

2.3.2.1 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。

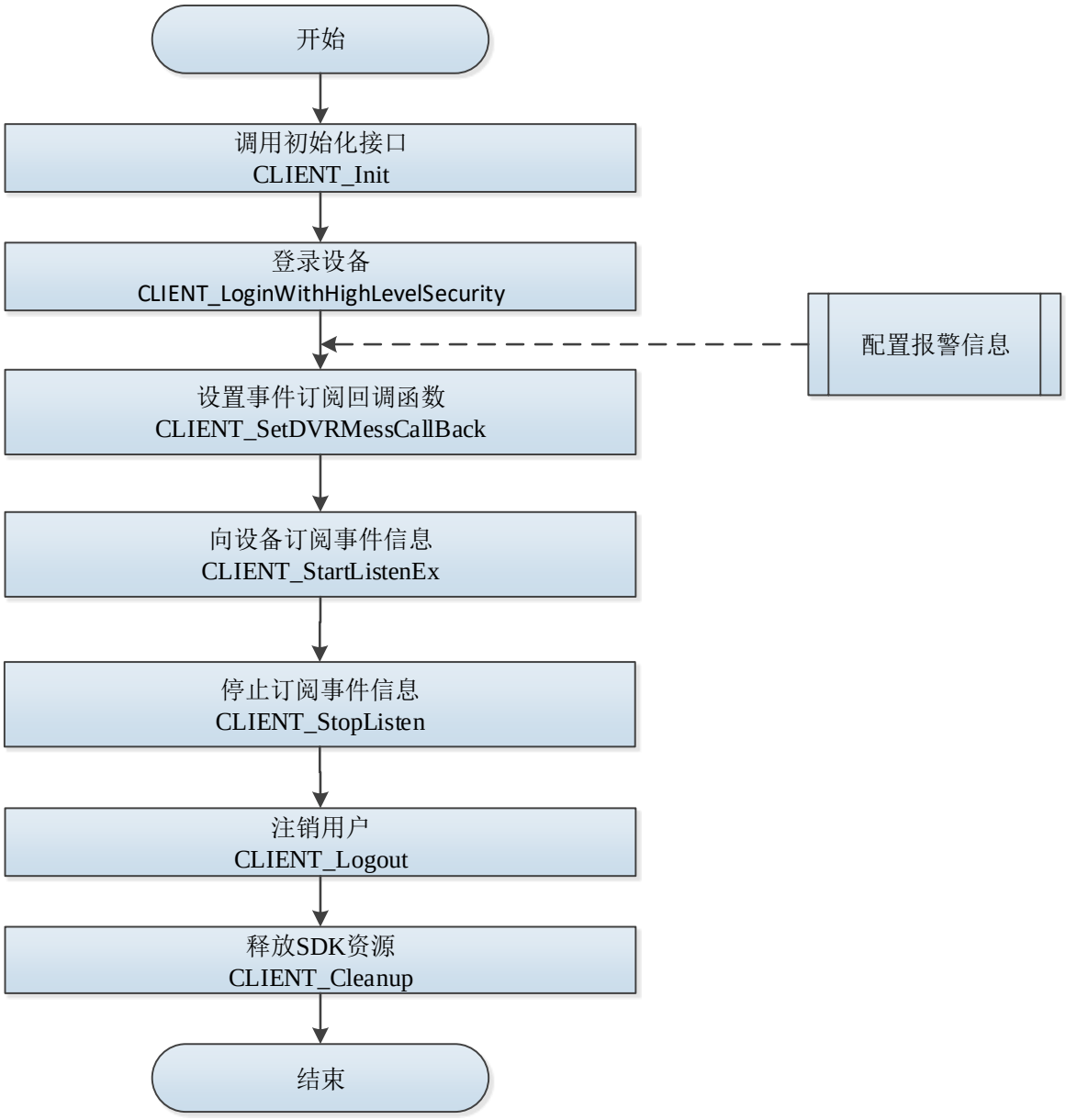
2.3.2.2 接口总览

表2-13 报警事件接口说明

接口	说明
CLIENT_StartListenEx	向设备订阅报警。
CLIENT_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 CLIENT_StartListen 或 CLIENT_StartListenEx 才有效。
CLIENT_StopListen	停止订阅。

2.3.2.3 流程说明

图2-15 报警事件业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_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 CLIENT_SetDVRMessCallBack。
步骤5　调用 CLIENT_StartListenEx 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 CLIENT_StopListen。
步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.2.4 示例代码

```
BOOL CALLBACK MessCallBack(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char
*pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    //设备/读卡器防拆
    if (DH_ALARM_CHASSISINTRUDED == lCommand)
    {
        ALARM_CHASSISINTRUDED_INFO* pstAlarm = (ALARM_CHASSISINTRUDED_INFO*)pBuf;
        printf("Chassis intrusion\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //外部报警事件
    else if (DH_ALARM_ALARM_EX2 == lCommand)
    {
        ALARM_ALARM_INFO_EX2* pstAlarm = (ALARM_ALARM_INFO_EX2*)pBuf;
        printf("LocalAlarm\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("ChannelID:%d,SenseType:%d\n", pstAlarm->nChannelID, pstAlarm->emSenseType);
        printf("DefenceAreaType:%d\n", pstAlarm->emDefenceAreaType);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //门超时事件
    else if (DH_ALARM_ACCESS_CTL_NOT_CLOSE == lCommand)
    {
        ALARM_ACCESS_CTL_NOT_CLOSE_INFO* pstAlarm =
(ALARM_ACCESS_CTL_NOT_CLOSE_INFO*)pBuf;
        printf("DoorNotClosed\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("DoorNO.:%d,EventID:%d\n", pstAlarm->nDoor, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
```

```
    //闯入事件
    else if (DH_ALARM_ACCESS_CTL_BREAK_IN == lCommand)
    {
        ALARM_ACCESS_CTL_BREAK_IN_INFO* pstAlarm =
(ALARM_ACCESS_CTL_BREAK_IN_INFO*)pBuf;
        printf("BreakIn\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("BreakMethod:%d,EventID:%d\n", pstAlarm->emMethod, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //胁迫事件
    else if (DH_ALARM_ACCESS_CTL_DURESS == lCommand)
    {
        ALARM_ACCESS_CTL_DURESS_INFO* pstAlarm = (ALARM_ACCESS_CTL_DURESS_INFO*)pBuf;
        printf("Duress\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("CardNo:%d,EventID:%d\n", pstAlarm->szCardNo, pstAlarm->nEventID);
        printf("DoorName:%s,SN:%s,UserID:%s\n", pstAlarm->szDoorName, pstAlarm->szSN,
pstAlarm->szUserID);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //反潜事件
    else if (DH_ALARM_ACCESS_CTL_REPEAT_ENTER == lCommand)
    {
        ALARM_ACCESS_CTL_REPEAT_ENTER_INFO* pstAlarm =
(ALARM_ACCESS_CTL_REPEAT_ENTER_INFO*)pBuf;
        printf("Duress\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("CardNo:%d,EventID:%d\n", pstAlarm->szCardNo, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    return TRUE;
}
```

```
void StartListen(LLONG g_lLoginHandle)
{
    CLIENT_SetDVRMessCallBack(MessCallBack, NULL);
    BOOL bRet = CLIENT_StartListenEx(g_lLoginHandle);

    if (bRet)
    {
        printf("CLIENT_StartListenEx success!\n");
    }
    else
    {
        printf("CLIENT_StartListenEx failed! LastError = %x\n" , CLIENT_GetLastError());
    }
}
```

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。

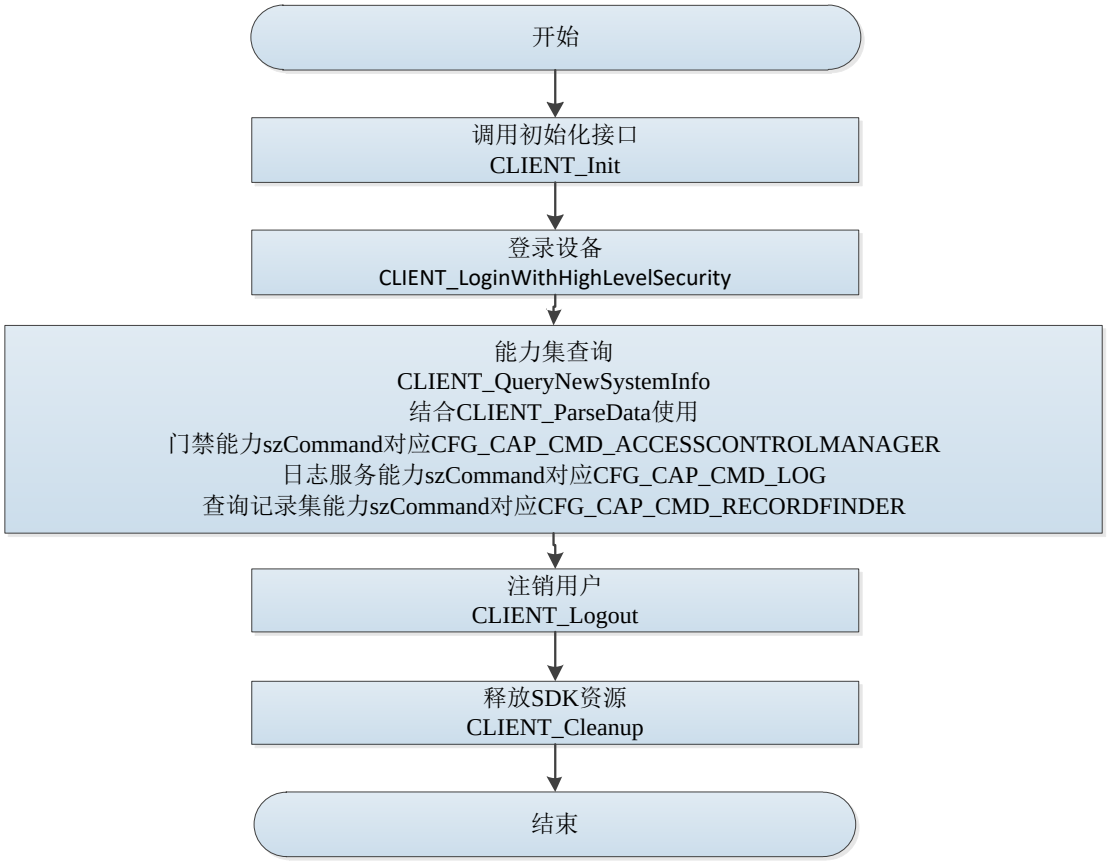
2.3.3.1.2 接口总览

表2-14 能力集查询接口说明

接口	说明
CLIENT_QueryNewSystemInfo	查询系统能力信息（日志、记录集、门控能力等）。
CLIENT_ParseData	解析查询到的配置信息。

2.3.3.1.3 流程说明

图2-16 设备信息查看流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_QueryNewSystemInfo 函数，结合 CLIENT_ParseData，来查询门禁能力集。

表2-15 szCommand 取值对应含义和结构体

szCommand	对应含义	szOutBuffer
CFG_CAP_CMD_ACCESSCONTROLMANAGER	门禁能力	CFG_CAP_ACCESSCONTROL
CFG_CAP_CMD_LOG	获取日志服务能力	CFG_CAP_LOG
CFG_CAP_CMD_RECORDFINDER	获取查询记录集能力	CFG_CAP_RECORDFINDER_INFO

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.3.1.4 示例代码

```
// 能力集查询
char szBuf[1024] = {0};
int nError = 0;
BOOL bRet = CLIENT_QueryNewSystemInfo(m_lLoginID, CFG_CAP_CMD_ACCESSCONTROLMANAGER, -1,
```

```
szBuf, sizeof(szBuf), &nError, 3000);
if (bRet)
{
        CFG_CAP_ACCESSCONTROL stuCap = {0};
        DWORD dwRet = 0;
        bRet = CLIENT_ParseData(CFG_CAP_CMD_ACCESSCONTROLMANAGER, szBuf, &stuCap,
sizeof(stuCap), &dwRet);
        if (bRet && dwRet == sizeof(CFG_CAP_ACCESSCONTROL))
        {
            int nCount = stuCap.nAccessControlGroups;
        }
        else
        {
            return FALSE;
        }
}
```

2.3.3.2 设备版本、MAC 查看

2.3.3.2.1 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。

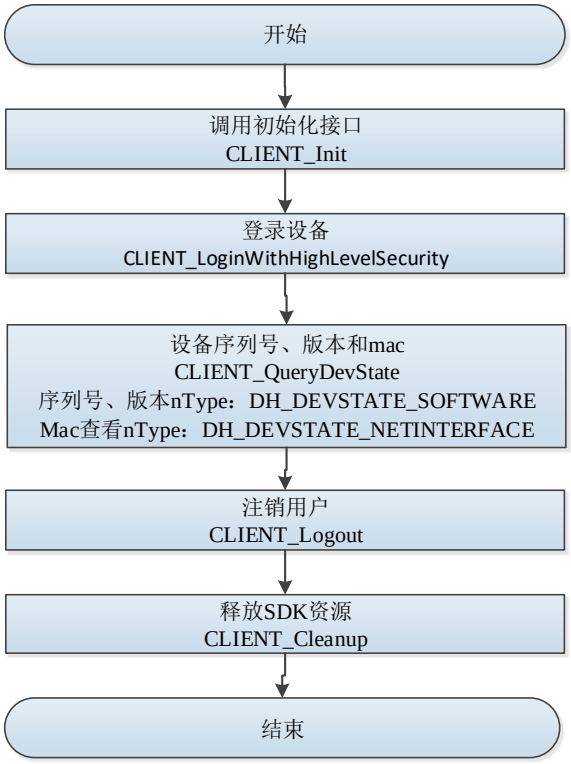
2.3.3.2.2 接口总览

表2-16 设备版本和 MAC 查看接口说明

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

2.3.3.2.3 流程说明

图2-17 设备信息查看业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_QueryDevState 函数，来查询门禁设备的序列号、版本和 mac 信息。

表2-17 nType 取值对应含义和结构体

nType	对应含义	pBuf
DH_DEVSTATE_SOFTWARE	序列号、版本	DHDEV_VERSION_INFO
DH_DEVSTATE_NETINTERFACE	Mac 地址	DHDEV_NETINTERFACE_INFO

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.3.2.4 示例代码

```
//查询设备序列号
int nRet = 0;
DHDEV_VERSION_INFO stuVersion = {sizeof(stuVersion)};
BOOL bRet = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_SOFTWARE, (char *)&stuVersion,
sizeof(stuVersion), &nRet, 5000);
//Mac 查看
int nRet = 0;
DHDEV_NETINTERFACE_INFO stuNet = {sizeof(stuNet)};
BOOL bRet0 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_NETINTERFACE, (char *)&stuNet,
```

```
sizeof(stuNet), &nRet, 5000);
```

2.3.4 网络配置

2.3.4.1 IP 配置

2.3.4.1.1 简介

IP 配置，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。

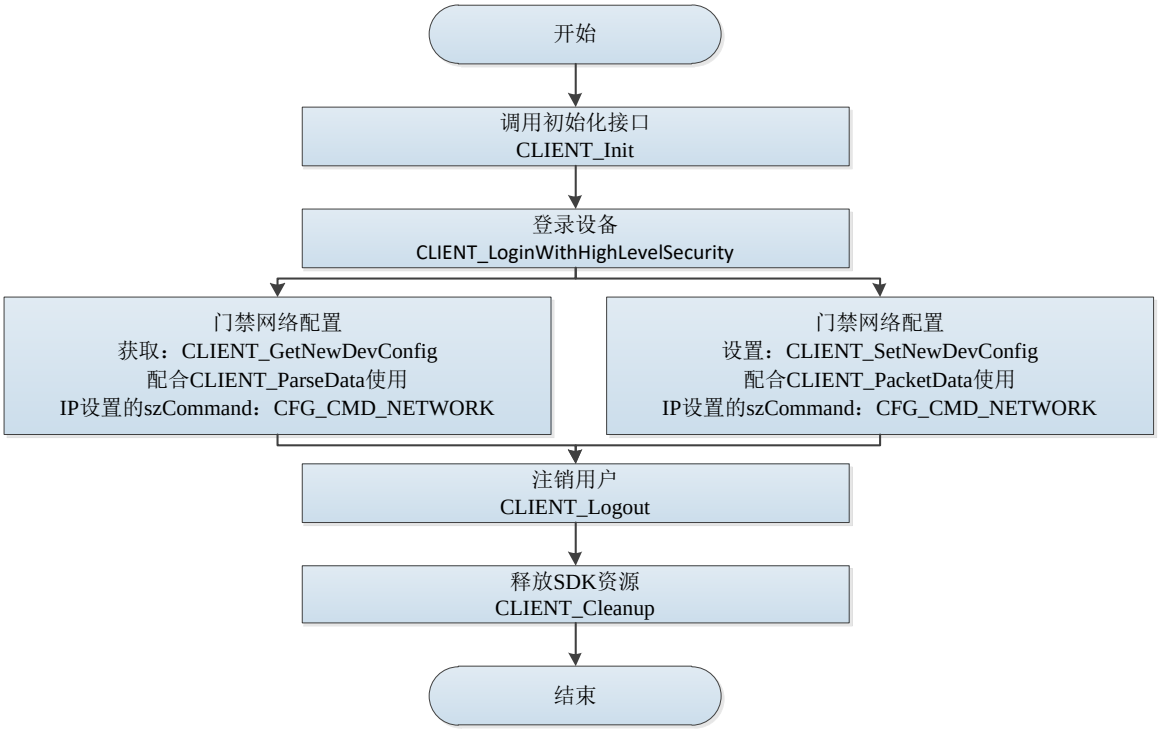
2.3.4.1.2 接口总览

表2-18 IP 配置接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息
CLIENT_ParseData	解析查询到的配置信息
CLIENT_SetNewDevConfig	设置配置信息
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式

2.3.4.1.3 流程说明

图2-18 IP 配置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁 IP 网络配置。

- szCommand：CFG_CMD_NETWORK。
- pBuf：CFG_NETWORK_INFO。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁 IP 网络配置。

- szCommand：CFG_CMD_NETWORK。
- pBuf：CFG_NETWORK_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.4.1.4 示例代码

```
// 获取 IP 网络配置信息
char * szOut1 = new char[1024*32];
CFG_NETWORK_INFO stOut2 = {sizeof(stOut2)};
int nError = 0;
BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_NETWORK, 0, szOut1, 1024*32,
&nError, 3000);
if(bRet){
    BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NETWORK, szOut1, &stOut2, sizeof(CFG_NTP_INFO),
NULL);
}
else{
        printf("parse failed!!!");
}
//设置 IP 网络配置信息
char * szOut = new char[1024*32];
stOut2.nInterfaceNum = 1;
memcpy(stOut2.stuInterfaces[0].szIP, "192.168.1.108", sizeof(stOut2.stuInterfaces[0].szIP)-1);
memcpy(stOut2.stuInterfaces[0].szDefGateway, "192.168.1.1", sizeof(stOut2.stuInterfaces[0].
szDefGateway)-1);
memcpy(stOut2.stuInterfaces[0].szSubnetMask, "255.255.255.0", sizeof(stOut2.stuInterfaces[0].
szSubnetMask)-1);
BOOL bRet0 = CLIENT_PacketData(CFG_CMD_NETWORK, (char *)&stOut2, sizeof(CFG_NETWORK_INFO),
szOut, 1024*32);
if(bRet){
    BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_NETWORK, 0, szOut, 1024*32,
NULL, NULL, 3000);
}
```

2.3.4.2 主动注册配置

2.3.4.2.1 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。

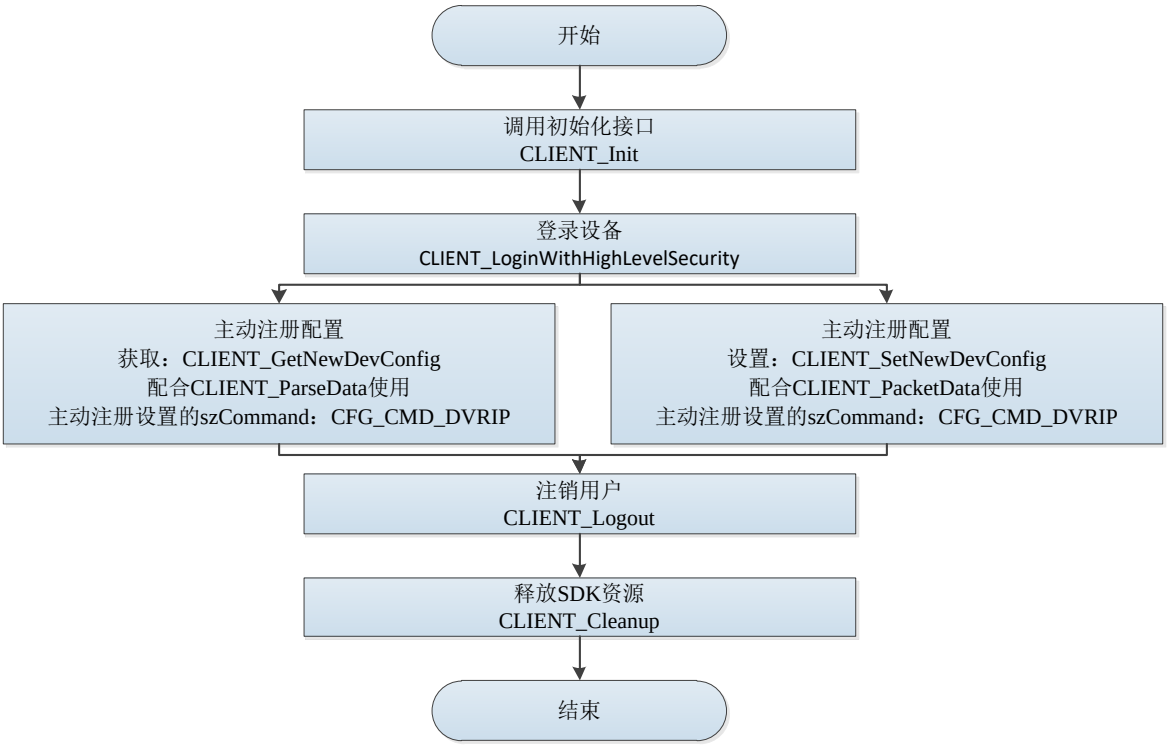
2.3.4.2.2 接口总览

表2-19 主动注册配置接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.4.2.3 流程说明

图2-19 主动注册配置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　主动注册配置。

- 调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询主动注册配置。
  - ◇ szCommand：CFG_CMD_DVRIP。
  - ◇ pBuf：CFG_DVRIP_INFO。

- 调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置主动注册配置。
  - ◇ szCommand：CFG_CMD_DVRIP。
  - ◇ pBuf：CFG_DVRIP_INFO。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.4.2.4 示例代码

```
// 获取主动注册网络配置信息
char * szOut1 = new char[1024*32];
CFG_DVRIP_INFO stOut2 = {sizeof(stOut2)};
int nError = 0;
BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_DVRIP, 0, szOut1, 1024*32, &nError, 3000);
if(bRet){
    BOOL bRet1 = CLIENT_ParseData(CFG_CMD_DVRIP, szOut1, &stOut2, sizeof(CFG_NTP_INFO), NULL);
}
else{
    printf("parse failed!!!");
}
//设置主动注册网络配置信息
char * szOut = new char[1024*32];
stOut2.nTcpPort = 46650;
BOOL bRet0 = CLIENT_PacketData(CFG_CMD_DVRIP, (char *)&stOut2, sizeof(CFG_DVRIP_INFO), szOut, 1024*32);
if(bRet)
{
    BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_DVRIP, 0, szOut, 1024*32, NULL, NULL, 3000);
}
```

2.3.5 设备时间设置

2.3.5.1 设备时间设置

2.3.5.1.1 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。

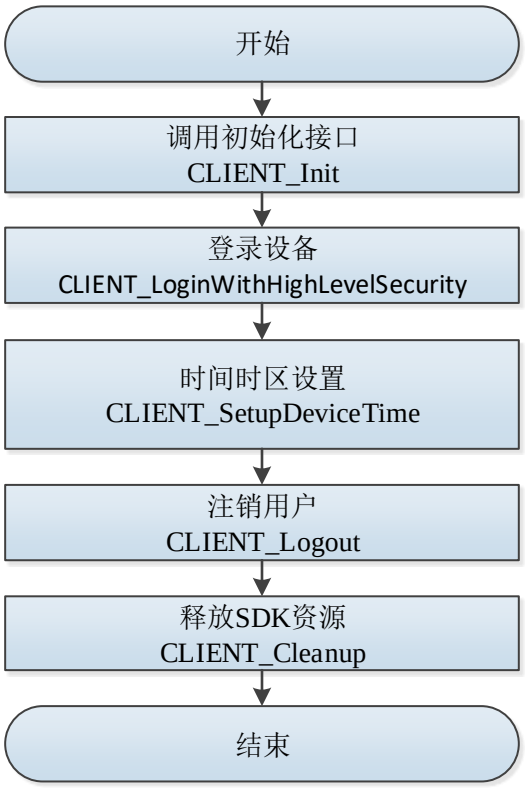
2.3.5.1.2 接口总览

表2-20 时间设置接口说明

接口	说明
CLIENT_SetupDeviceTime	设置设备当前时间。

2.3.5.1.3 流程说明

图2-20 时间设置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　调用 CLIENT_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 CLIENT_Logout 函数登出设备。
步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.5.1.4 示例代码

```
//时间时区设置
NET_TIME stuInfo = {sizeof(stuInfo)};
stuInfo.dwDay = 15;
stuInfo.dwYear = 2019;
stuInfo.dwMonth = 12;
stuInfo.dwHour = 17;
stuInfo.dwMinute = 45;
stuInfo.dwSecond = 25;
BOOL bRet = CLIENT_SetupDeviceTime(g_lLoginHandle, &stuInfo);
```

2.3.5.2 NTP 服务器和时区配置

2.3.5.2.1 简介

NTP 服务器和时区配置，即用户通过调用 SDK 接口，对 NTP 服务器和时区进行获取和配置。

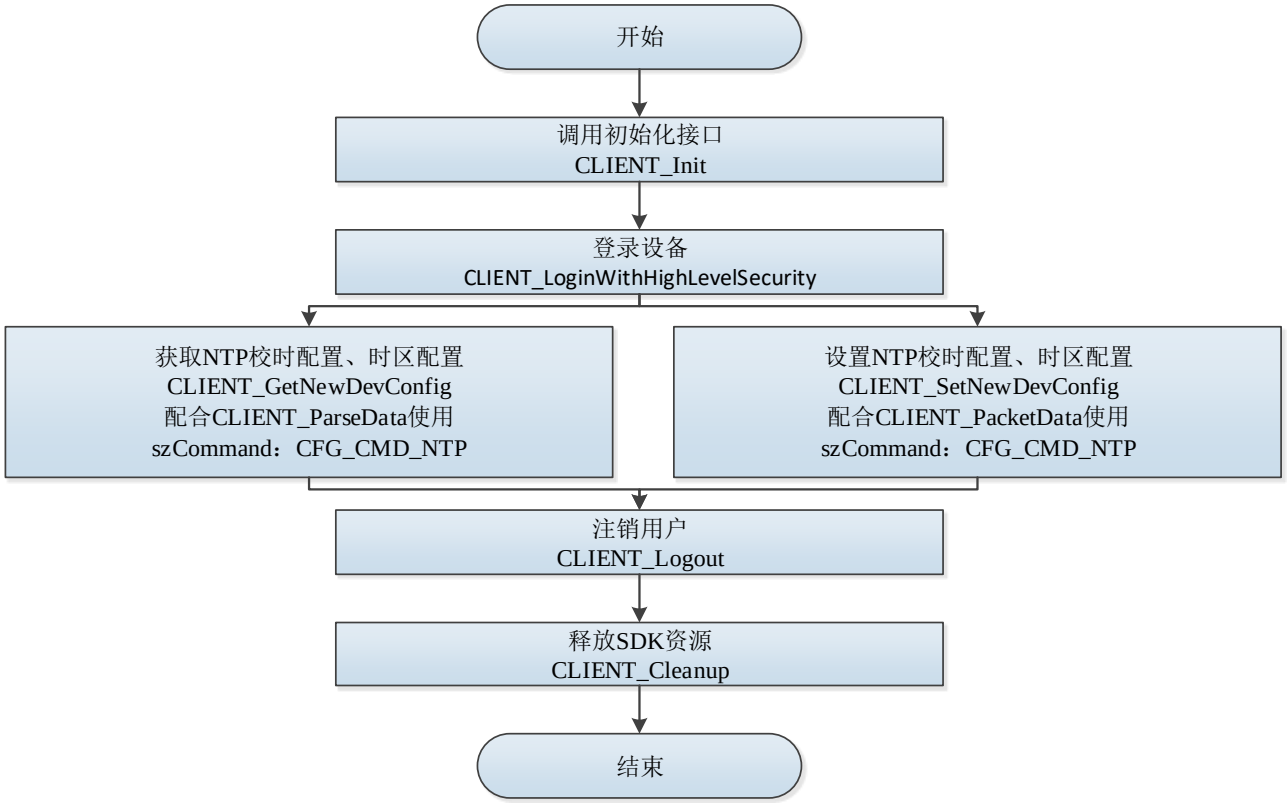
2.3.5.2.2 接口总览

表2-21 NTP 服务器和时区接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.5.2.3 流程说明

图2-21 NTP 校时业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁 NTP 校时配置、时区配置。

- szCommand：CFG_CMD_NTP。
- pBuf：CFG_NTP_INFO。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁 NTP 校时配

置、时区配置。

- szCommand：CFG_CMD_NTP。
- pBuf：CFG_NTP_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.5.2.4 示例代码

```
//设置 NTP 校时配置、时区配置信息
char * szOut1 = new char[1024*32];
    CFG_NTP_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_NTP, 0, szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NTP, szOut1, &stOut2, sizeof(CFG_NTP_INFO), NULL);
    }
    else{
        printf("parse failed!!!");
    }
//设置 NTP 校时配置信息、时区配置信息
    char * szOut = new char[1024*32];
    stOut2.bEnable = TRUE;
    BOOL bRet0 = CLIENT_PacketData(CFG_CMD_NTP, (char *)&stOut2, sizeof(CFG_NTP_INFO), szOut, 1024*32);
    if(bRet)
    {
        BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_NTP, 0, szOut, 1024*32, NULL, NULL, 3000);
    }
```

2.3.5.3 夏令时配置

2.3.5.3.1 简介

夏令时配置，即用户通过调用 SDK 接口，对夏令时进行获取和配置的操作。

2.3.5.3.2 接口总览

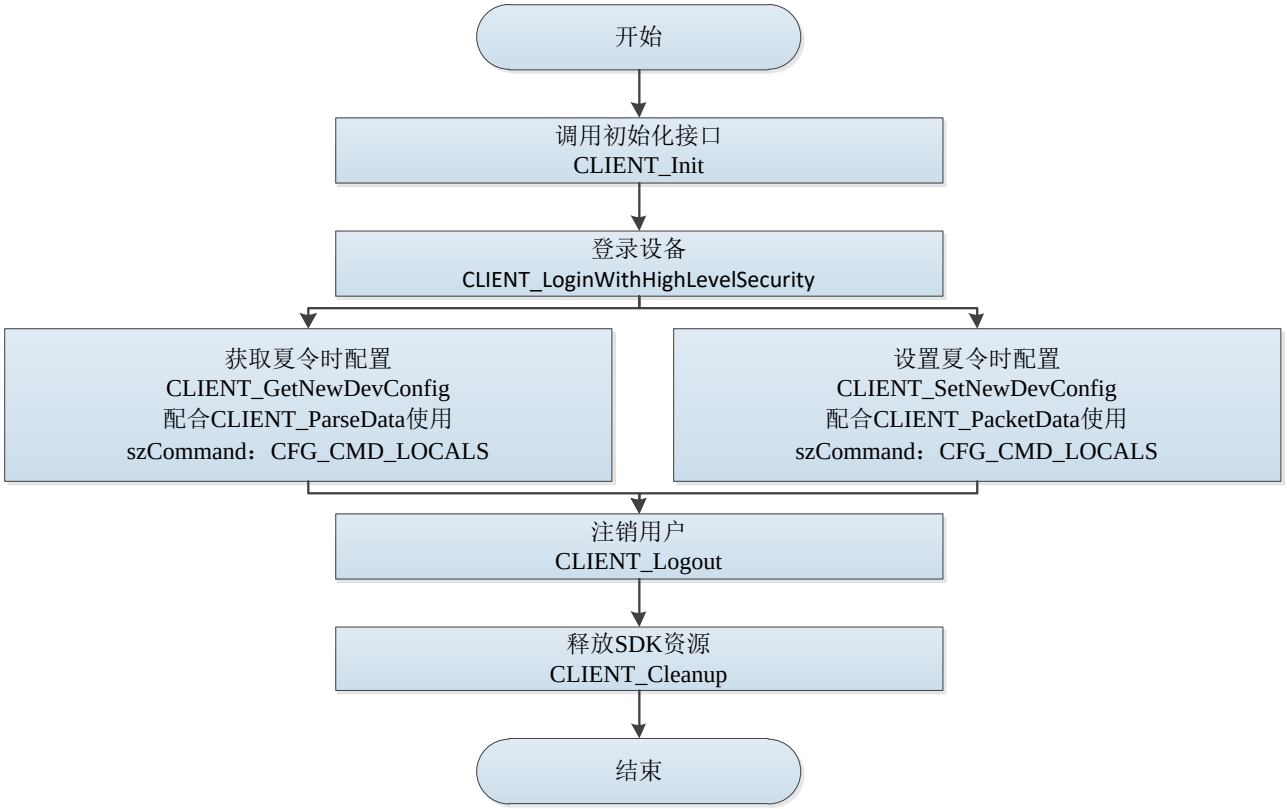
表2-22 夏令时配置接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。

接口	说明
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.5.3.3 流程说明

图2-22 夏令时配置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁夏令时信息。

- szCommand：CFG_CMD_LOCALS。
- pBuf：AV_CFG_Locales。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁夏令时信息。

- szCommand：CFG_CMD_ LOCALS。
- pBuf：AV_CFG_Locales。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.5.3.4 示例代码

```
//设置夏令时配置信息
char * szOut1 = new char[1024*32];
    AV_CFG_Locales stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_LOCALS, 0, szOut1, 1024*32,
```

```
&nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NTP, szOut1, &stOut2, sizeof(AV_CFG_Locales),
NULL);
    }
    else{
        printf("parse failed!!!");
    }
//设置夏令时配置信息
    char * szOut = new char[1024*32];
    stOut2.bEnable = TRUE;
    BOOL bRet0 = CLIENT_PacketData(CFG_CMD_LOCALS, (char *)&stOut2, sizeof(AV_CFG_Locales),
szOut, 1024*32);
    if(bRet)
    {
        BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_LOCALS, 0, szOut,
1024*32, NULL, NULL, 3000);
    }
```

2.3.6 维护配置

2.3.6.1 修改登录密码

2.3.6.1.1 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。

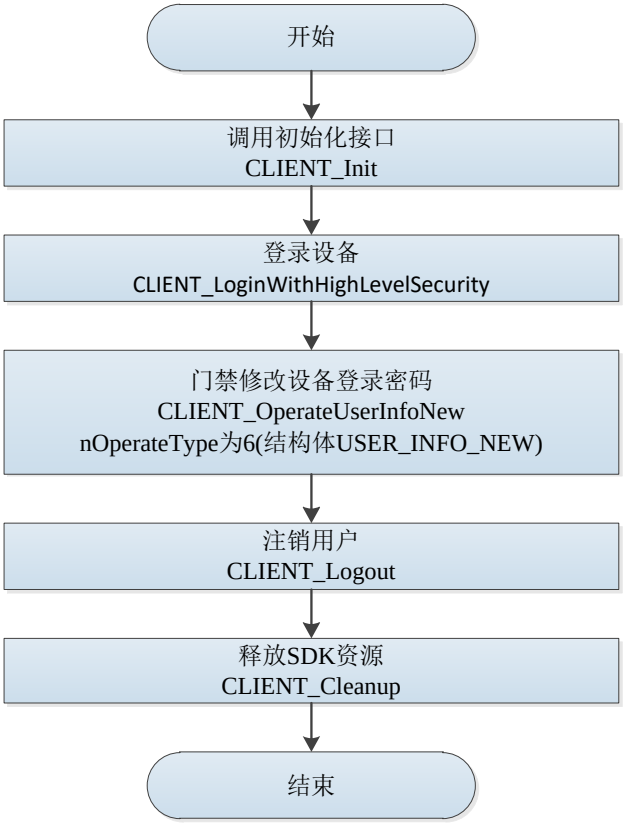
2.3.6.1.2 接口总览

表2-23 修改登录密码接口说明

接口	说明
CLIENT_OperateUserInfoNew	操作设备用户。

2.3.6.1.3 流程说明

图2-23 修改登录密码业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_OperateUserInfoNew 函数来操作设备用户信息，从而修改设备登录密码。

nOperateType：6。

opParam 和 subParam：USER_INFO_NEW。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.6.1.4 示例代码

```
// 修改设备登录密码
USER_INFO_NEW stuNewInfo = {sizeof(stuNewInfo)};
memcpy(stuNewInfo.passWord, "admin", sizeof(stuNewInfo.passWord)-1);

USER_INFO_NEW stuOldInfo = {sizeof(stuOldInfo)};
memcpy(stuOldInfo.passWord, "admin123", sizeof(stuOldInfo.passWord)-1);

BOOL bRet = CLIENT_OperateUserInfoNew(g_lLoginHandle, 7, &stuNewInfo, &stuOldInfo, NULL, 3000);
```

2.3.6.2 重启设备

2.3.6.2.1 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。

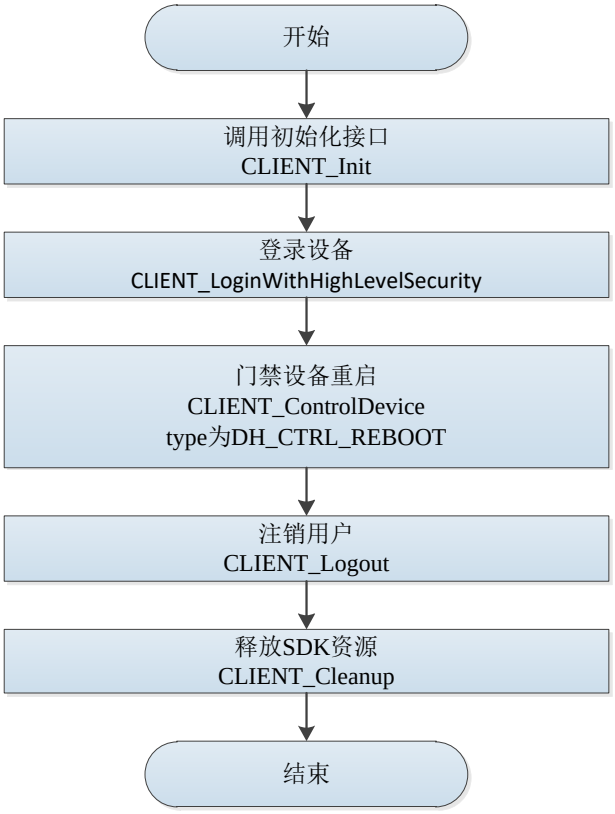
2.3.6.2.2 接口总览

表2-24 重启设备接口说明

接口	说明
CLIENT_ControlDevice	设备控制。

2.3.6.2.3 流程说明

图2-24 重启设备业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　调用 CLIENT_ControlDevice 函数来控制设备重启。
type：DH_CTRL_REBOOT。
步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。
步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.6.2.4 示例代码

```
// 重启设备
```

```
CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_REBOOT, NULL, 3000);
```

2.3.6.3 恢复出厂设置

2.3.6.3.1 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，点击生效后会清空设备端所有配置及人员信息。

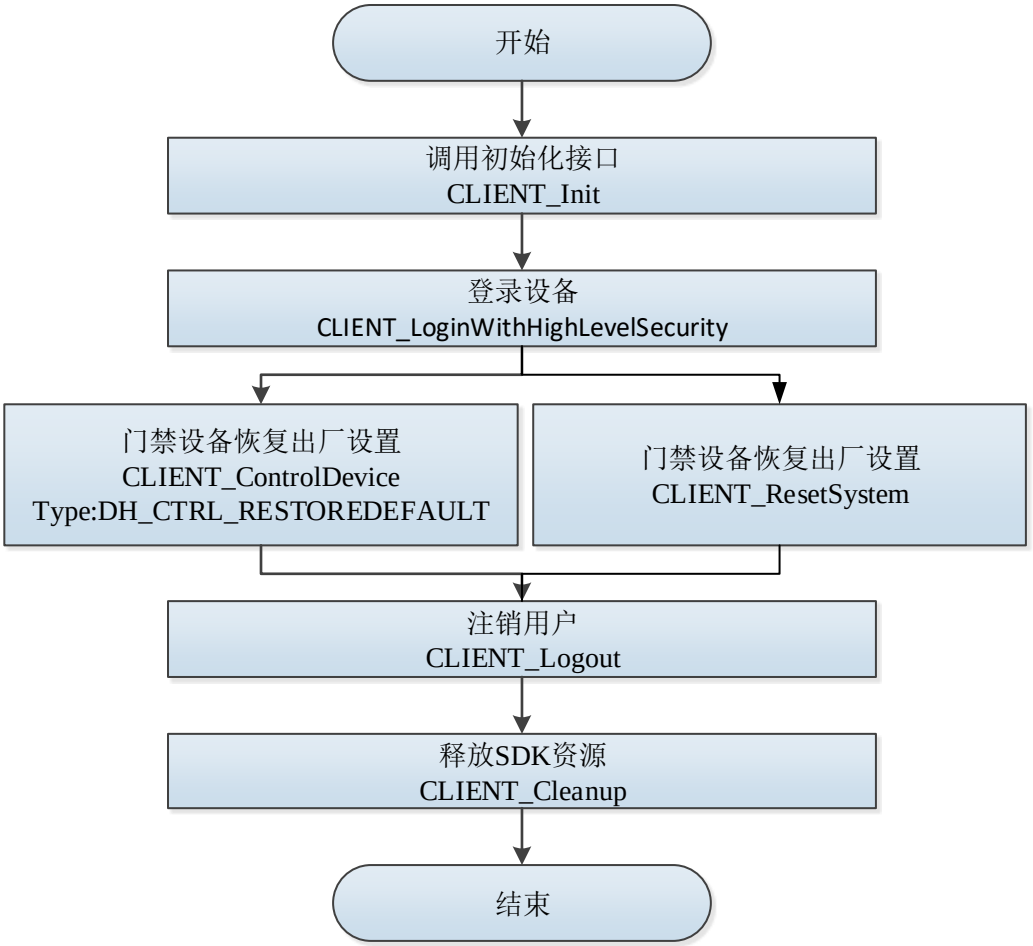
2.3.6.3.2 接口总览

表2-25 恢复出厂设置接口说明

接口	说明
CLIENT_ControlDevice	控制设备（恢复出厂设置），支持一体机、控制器
CLIENT_ResetSystem	控制设备（恢复出厂设置），支持一体机（推荐）

2.3.6.3.3 流程说明

图2-25 恢复出厂设置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ResetSystem 函数来控制设备（指纹一体机）恢复出厂设置。

步骤4　调用 CLIENT_ControlDevice 函数来控制设备（控制器、指纹一体机）恢复出厂设置。

Type：DH_CTRL_RESTOREDEFAULT。

Param：DH_RESTORE_COMMON。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.6.3.4 示例代码

```
// 恢复出厂设置
NET_IN_RESET_SYSTEM stind = {sizeof(stind)};
    NET_OUT_RESET_SYSTEM stoutd = {sizeof(stoutd)};
    BOOL bRet = CLIENT_ResetSystem(m_lLoginID,&stind, &stoutd ,5000);//可以重置一体机
    if (!bRet)
    {
        DWORD nparam = DH_RESTORE_ALL;
        BOOL bRet = CLIENT_ControlDevice(m_lLoginID, DH_CTRL_RESTOREDEFAULT, (void*)&nparam,
3000);//可以重置一体机和控制器
        if (!bRet)
        {
            return FALSE;
        }
    }
```

2.3.6.4 设备升级

2.3.6.4.1 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。

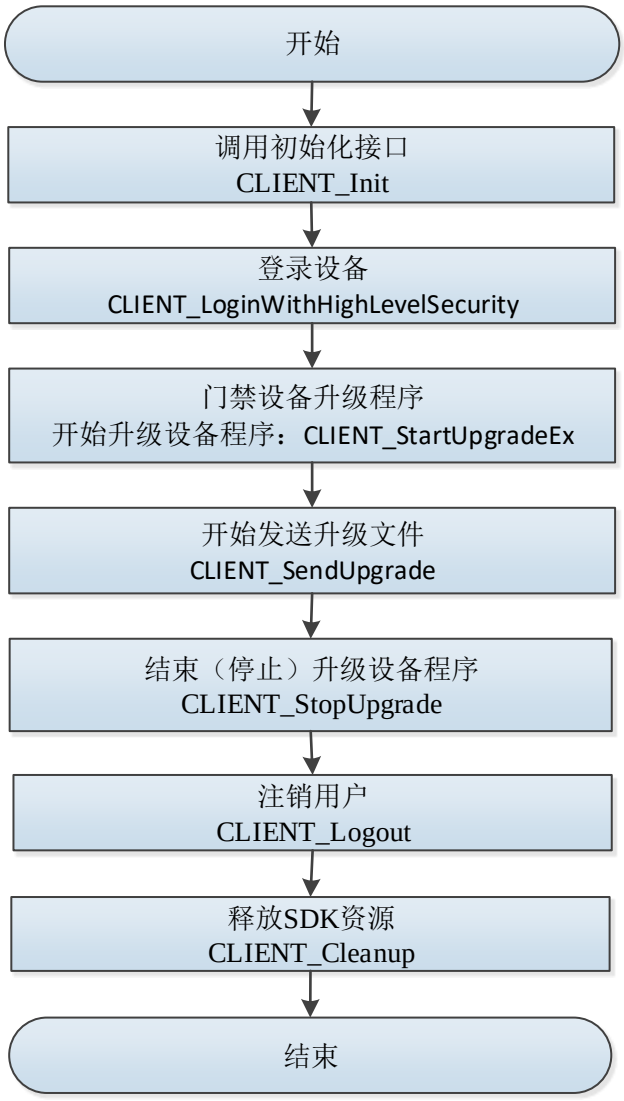
2.3.6.4.2 接口总览

表2-26 设备升级接口说明

接口	说明
CLIENT_StartUpgradeEx	开始升级设备程序--扩展。
CLIENT_SendUpgrade	开始发送升级文件。
CLIENT_StopUpgrade	停止升级。

2.3.6.4.3 流程说明

图2-26 设备升级业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　调用 CLIENT_StartUpgradeEx 函数来开始升级设备程序。
步骤4　调用 CLIENT_SendUpgrade 函数来发送设备升级文件。
步骤5　调用 CLIENT_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 CLIENT_Logout 函数登出设备。
步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.6.4.4 示例代码

```
BOOL m_isNeedStop = FALSE;
void CALLBACK UpgradeCallBack(LLONG lLoginID, LLONG lUpgradechannel, int nTotalSize, int nSendSize,
LDWORD dwUser)
{
    if (0 == lLoginID    || 0 == lUpgradechannel)
```

```
    {
        cout << "lLoginID or lUpgradechannel is zero" << endl;
        m_isNeedStop = TRUE;
        return;
    }
    if (0 == nTotalSize && -1 == nSendSize) // 表示升级结束
    {
        m_isNeedStop = TRUE;
        cout << "升级完成！" << endl;
    }
    else if (0 == nTotalSize && -2 == nSendSize) // 表示升级出错
    {
        m_isNeedStop = TRUE;
        cout << "升级出错" << endl;
    }
    else if (nTotalSize > 0 && nSendSize >= 0) // 表示发送进度
    {
        float fPross = (float)(nSendSize/nTotalSize);
        printf("升级文件发送进度（升级文件总大小:%d, 已发送大小:%d, 发送进度:%.2f%%）\n",
nTotalSize, nSendSize, fPross*100);
        if (nTotalSize == nSendSize)
        {
            cout << "升级文件发送完成！设备开始升级....." << endl;
        }
    }
    else if (nTotalSize == -1   && nSendSize   >= 0)
    {
        cout << "....................升级进度:" << nSendSize << "....................." << endl;
    }
}

void Test()
{
    char szFileName[256] = {0};
    cout << "输入升级程序文件名(包括完整路径):" << endl;
    cin >> szFileName;
    //开始升级设备程序
    LLONG lUpHandle =   CLIENT_StartUpgradeEx(g_lLoginHandle, DH_UPGRADE_BOOT_YPE,
szFileName, UpgradeCallBack, 0);
    if (0 == lUpHandle)
```

```
    {
        printf("CLIENT_StartUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
        return;
    }
    //发送升级文件
    BOOL bRet = CLIENT_SendUpgrade(lUpHandle);
    if (!bRet)
    {
        printf("CLIENT_SendUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
        //停止升级程序
        CLIENT_StopUpgrade(lUpHandle);
        return;
    }
    while (true)
    {
        if (m_isNeedStop)
        {
            //停止升级程序
            bRet = CLIENT_StopUpgrade(lUpHandle);
            if (!bRet)
            {
                printf("CLIENT_SendUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
                return;
            }
            cout << "Success to stop upgrade!!" << endl;
            break;
        }
    }
}
```

2.3.6.5 自动维护

2.3.6.5.1 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。

2.3.6.5.2 接口总览

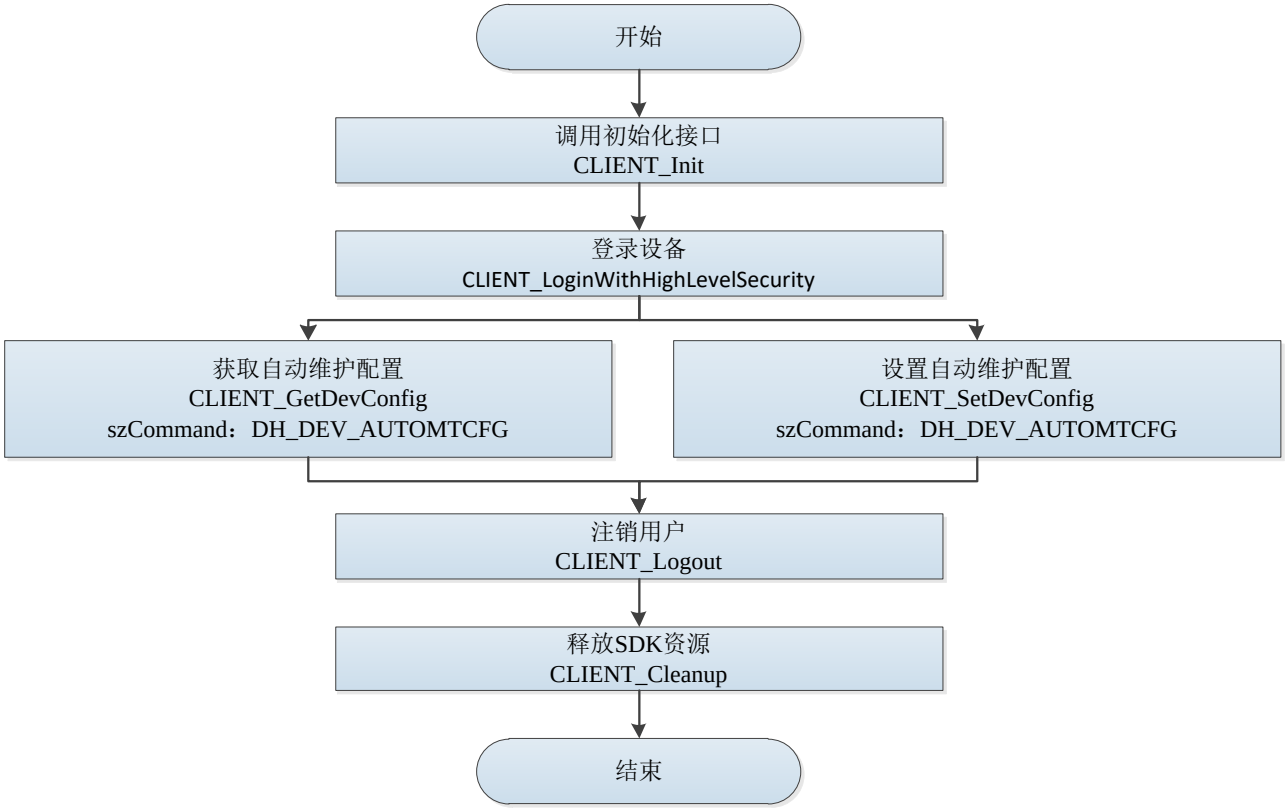
表2-27 自动维护接口说明

接口	说明
CLIENT_GetDevConfig	查询配置信息。

接口	说明
CLIENT_SetDevConfig	设置配置信息。

2.3.6.5.3 流程说明

图2-27 自动维护业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetDevConfig 函数来查询门禁自动维护信息。

- szCommand：DH_DEV_AUTOMTCFG。
- pBuf：DHDEV_AUTOMT_CFG。

步骤4　调用 CLIENT_SetDevConfig 函数来设置门禁自动维护信息。

- szCommand：DH_DEV_AUTOMTCFG。
- pBuf：DHDEV_AUTOMT_CFG。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.6.5.4 示例代码

```
// 获取自动维护配置信息
DHDEV_AUTOMT_CFG stInfo = {sizeof(stInfo)};
DWORD lpBytesReturned = 0;
BOOL bRet12 = CLIENT_GetDevConfig(g_lLoginHandle, DH_DEV_AUTOMTCFG, 0, &stInfo, sizeof(stInfo), &lpBytesReturned, 5000);
```

```
//设置自动维护配置信息
stInfo.byAutoRebootDay = 1;
BOOL bRet11 = CLIENT_SetDevConfig(g_lLoginHandle, DH_DEV_AUTOMTCFG, 0, &stInfo, sizeof(stInfo), 5000);
```

2.3.7 人员管理

2.3.7.1.1 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。

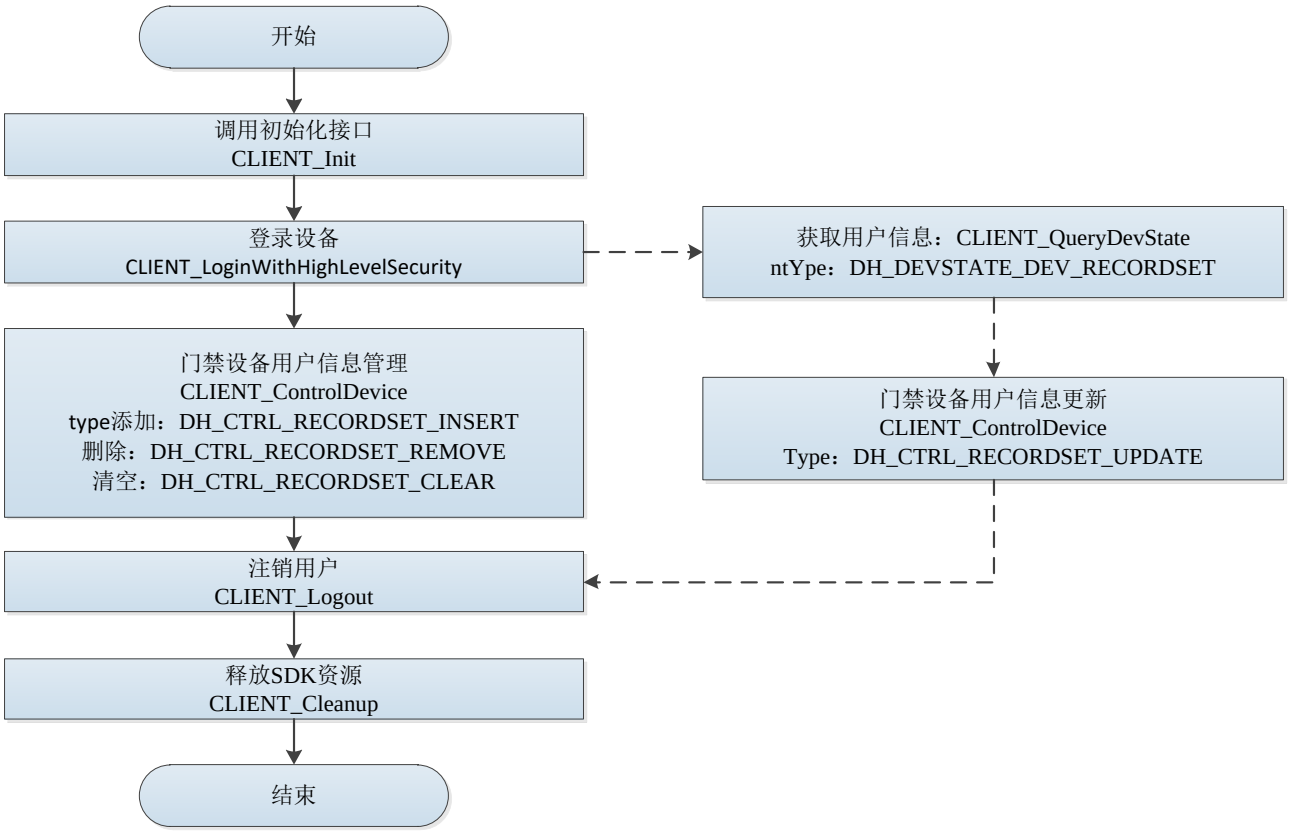
2.3.7.1.2 接口总览

表2-28 人员信息接口说明

接口	说明
CLIENT_ControlDevice	控制设备。
CLIENT_QueryDevState	查询设备状态。

2.3.7.1.3 流程说明

图2-28 用户信息管理业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDevice 函数来对用户信息进行操作。

表2-29 type 对应含义和结构体

Type	对应含义	emType	Param
● DH_CTRL_RECORDSET_INSERT ● DH_CTRL_RECORDSET_INSERTEX	添加用户信息	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_INSERT_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
DH_CTRL_RECORDSET_REMOVE	删除用户信息	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
DH_CTRL_RECORDSET_CLEAR	清空用户信息	NET_RECORD_ACCESSCTLCARD	NET_CTRL_RECORDSET_PARAM

步骤4　先调用 CLIENT_QueryDevState 接口来获取用户信息。

表2-30 type 对应含义和结构体

Type	对应含义	emType	Param
DH_DEVSTATE_DEV_RECORDSET	获取用户信息	NET_RECORD_ACCESSCTLCARD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_CARD

步骤5　再调用 CLIENT_ControlDevice 函数更新用户信息。

表2-31 type 对应含义和结构体

Type	对应含义	emType	Param
● DH_CTRL_RECORDSET_UPDATE ● DH_CTRL_RECORDSET_UPDATEEX	更新用户信息	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD

步骤6　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.7.1.4 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“2.3.9.1 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 2.3.10.5 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“2.3.10.1 分时段、首卡开门”。

2.3.7.1.5 示例代码

```
NET_RECORDSET_ACCESS_CTL_CARD stuInfo = {sizeof(stuInfo)};
    stuInfo.emSex = NET_ACCESSCTLCARD_SEX_MALE;
```

```
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;
    memcpy(stuInfo.szUserID, "ddjdj", sizeof(stuInfo.szUserID));
    memcpy(stuInfo.szPsw, "543543", sizeof(stuInfo.szPsw));



    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);

    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam,
5000);
//删除
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam1 = {sizeof(stuParam1)};
    stuParam1.emType = NET_RECORD_ACCESSCTLCARD;
    stuParam1.pBuf = (void*)&stuInfo.nRecNo;
    stuParam1.nBufLen = sizeof(stuInfo.nRecNo);

   BOOL bRet1 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam1,
5000);
//清空
    NET_CTRL_RECORDSET_PARAM stuParam2 = {sizeof(stuParam2)};
    stuParam2.emType = NET_RECORD_ACCESSCTLCARD;
    BOOL bRet2 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam2,
5000);
//获取
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
    stuParam3.emType = NET_RECORD_ACCESSCTLCARD;

    NET_RECORDSET_HOLIDAY stuHoliday = {sizeof(stuHoliday)};
    stuHoliday.nRecNo = stuInfo.nRecNo;
    stuParam3.pBuf = &stuHoliday;

    int nRet = 0;
```

```
    BOOL bRet3 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam3,
        sizeof(stuParam3), &nRet, 5000);
//更新
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam4 = {sizeof(stuParam4)};
    stuParam4.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam4.pBuf = (void*)&stuInfo;
    stuParam4.nBufLen = sizeof(stuInfo);

    int nRet4 = 0;
    BOOL bRet4 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam4,sizeof(stuParam4), &nRet4, 5000);
    if (bRet4)
    {
        stuInfo.emSex = NET_ACCESSCTLCARD_SEX_MALE;
        stuInfo.nDoorNum = 2;
        stuInfo.sznDoors[0] = 1223;
        memcpy(stuInfo.szUserID, "2222", sizeof(stuInfo.szUserID));
        memcpy(stuInfo.szPsw, "fdsfds", sizeof(stuInfo.szPsw));

        stuParam4.emType = NET_RECORD_ACCESSCTLHOLIDAY;
        stuParam4.pBuf = (void*)&stuInfo;
        stuParam4.nBufLen = sizeof(stuInfo);

        // update info
        BOOL bRet4 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE,
&stuParam, 5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!");
    }
```

2.3.8 门配置

2.3.8.1 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。

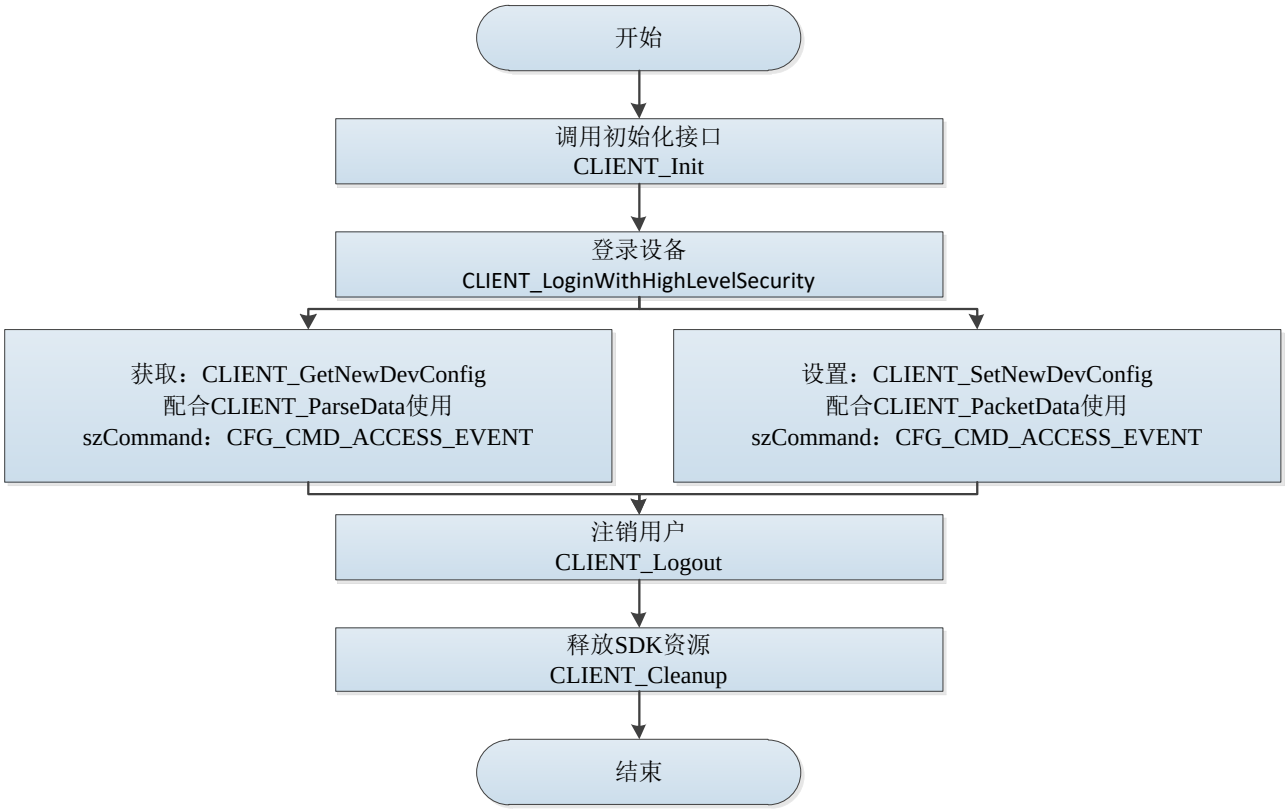
2.3.8.2 接口总览

表2-32 门配置信息接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.8.3 流程说明

图2-29 门配置信息业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁门配置信息。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-33 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

CFG_ACCESS_EVENT_INFO 参数	对应含义
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 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁门配置信息。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-34 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5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.8.4 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"2.3.9.1 时段配置"。

2.3.8.5 示例代码

```
// 获取门配置信息
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("门状态：%d\n",stOut2.emState);
            printf("开门时长：%d\n",stOut2.nUnlockHoldInterval);
```

```
            printf("关门超时时长：%d\n",stOut2.nCloseTimeout);
            printf("开门模式：%d\n",stOut2.emDoorOpenMethod);
            printf("胁迫：%d\n",stOut2.bDuressAlarmEnable);
        }
    }
    else{
        printf("parse failed!!!");
    }
//设置门配置信息
    char * szOut = new char[1024*32];
    stOut2.emState = ACCESS_STATE_NORMAL;//门状态
    stOut2.nUnlockHoldInterval = 10;//开门时长
    stOut2.nCloseTimeout = 10;//关门超时时长
    stOut2.emDoorOpenMethod = CFG_DOOR_OPEN_METHOD_PWD_ONLY;//开门模式
    stOut2.bDuressAlarmEnable = FALSE;//胁迫
    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.9 门时间配置

2.3.9.1 时段配置

2.3.9.1.1 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。

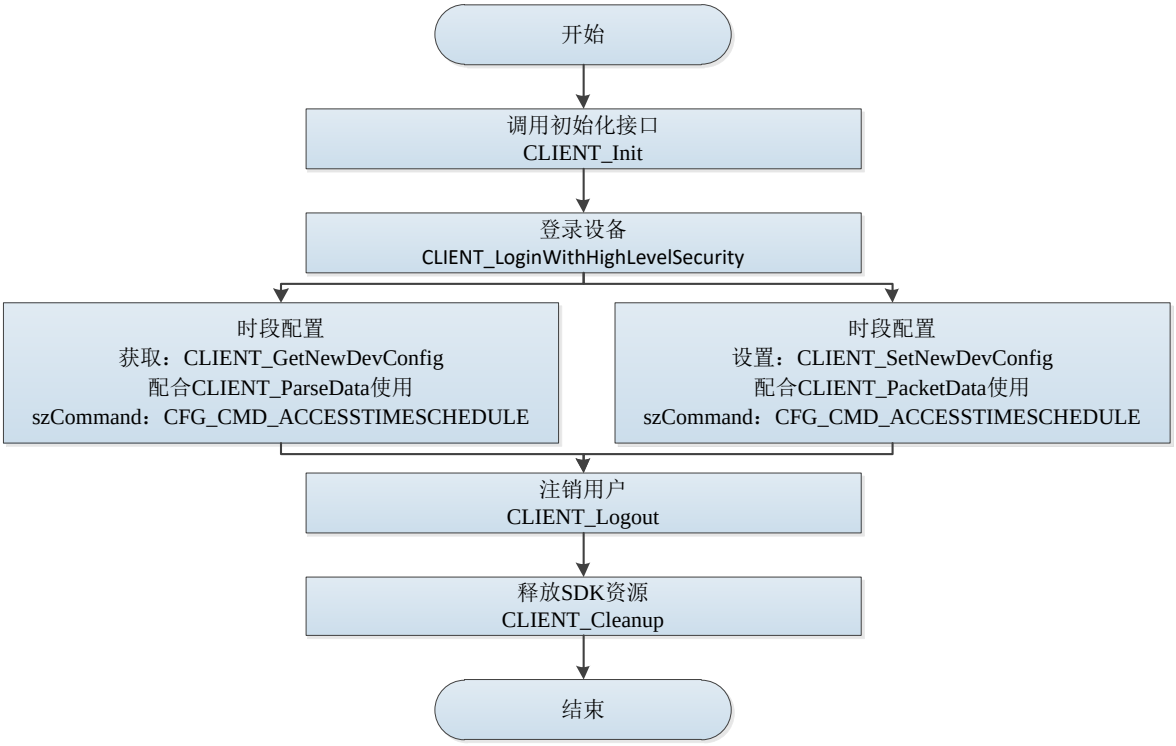
2.3.9.1.2 接口总览

表2-35 时段接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.9.1.3 流程说明

图2-30 时段业务配置流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁时段信息。

- szCommand：CFG_CMD_ACCESSTIMESCHEDULE。
- pBuf：CFG_ACCESS_TIMESCHEDULE_INFO。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁时段信息。

- szCommand：CFG_CMD_ACCESSTIMESCHEDULE。
- pBuf：CFG_ACCESS_TIMESCHEDULE_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.9.1.4 示例代码

```
// 获取时段配置信息
```

```
char * szOut1 = new char[1024*32];
    CFG_ACCESS_TIMESCHEDULE_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESSTIMESCHEDULE, 0,
szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESSTIMESCHEDULE, szOut1, &stOut2,
sizeof(CFG_ACCESS_TIMESCHEDULE_INFO), NULL);
        if (bRet1)
        {
            printf("使能：%d\n",stOut2.bEnable);
            printf("自定义名称：%s\n",stOut2.szName);
        }
    }
    else{
        printf("parse failed!!!");
    }
//设置时段配置 信息
    char * szOut = new char[1024*32];
    stOut2.bEnable = TRUE;
    memcpy(stOut2.szName, "ghgj", sizeof(stOut2.szName));

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESSTIMESCHEDULE, (char *)&stOut2,
sizeof(CFG_ACCESS_TIMESCHEDULE_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESSTIMESCHEDULE, 0,
szOut, 1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.9.2 常开常闭时间段配置

2.3.9.2.1 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

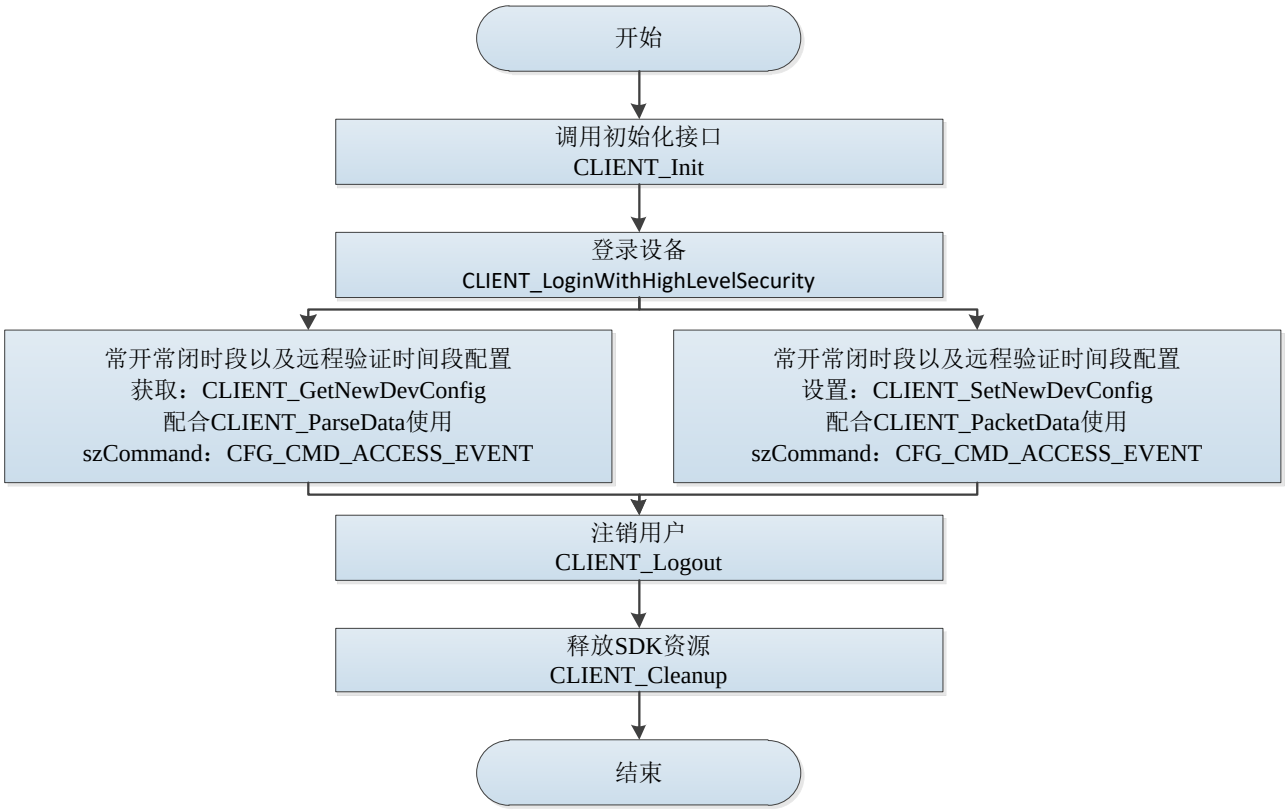
2.3.9.2.2 接口总览

表2-36 常开常闭时间段配置接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.9.2.3 流程说明

图2-31 常开常闭时间段配置流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁常开常闭时段配置信息以及远程验证时间段。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-37 CFG_ACCESS_EVENT_INFO 参数说明

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 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁常开常闭时段配置信息以及远程验证时间段。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-38 CFG_ACCESS_EVENT_INFO 参数说明

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 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.9.2.4 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"2.3.9.1 时段配置"。

2.3.9.2.5 示例代码

```
// 获取常开常闭、远程验证时段配置信息
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2, sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("常开时间段配置：%d\n",stOut2.nOpenAlwaysTimeIndex);
            printf("常闭时间段配置：%s\n",stOut2.nCloseAlwaysTimeIndex);
            printf("远程验证时间段使能：%d\n", stOut2.stuAutoRemoteCheck.bEnable);
        }
    }
    else{
        printf("parse failed!!!");
    }
    char * szOut = new char[1024*32];
```

```
    stOut2.nOpenAlwaysTimeIndex = 02;
    stOut2.nCloseAlwaysTimeIndex = 03;
    stOut2.stuAutoRemoteCheck.bEnable = TRUE;
// 获取常开常闭、远程验证时段配置信息
    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.9.3 假期配置

2.3.9.3.1 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。

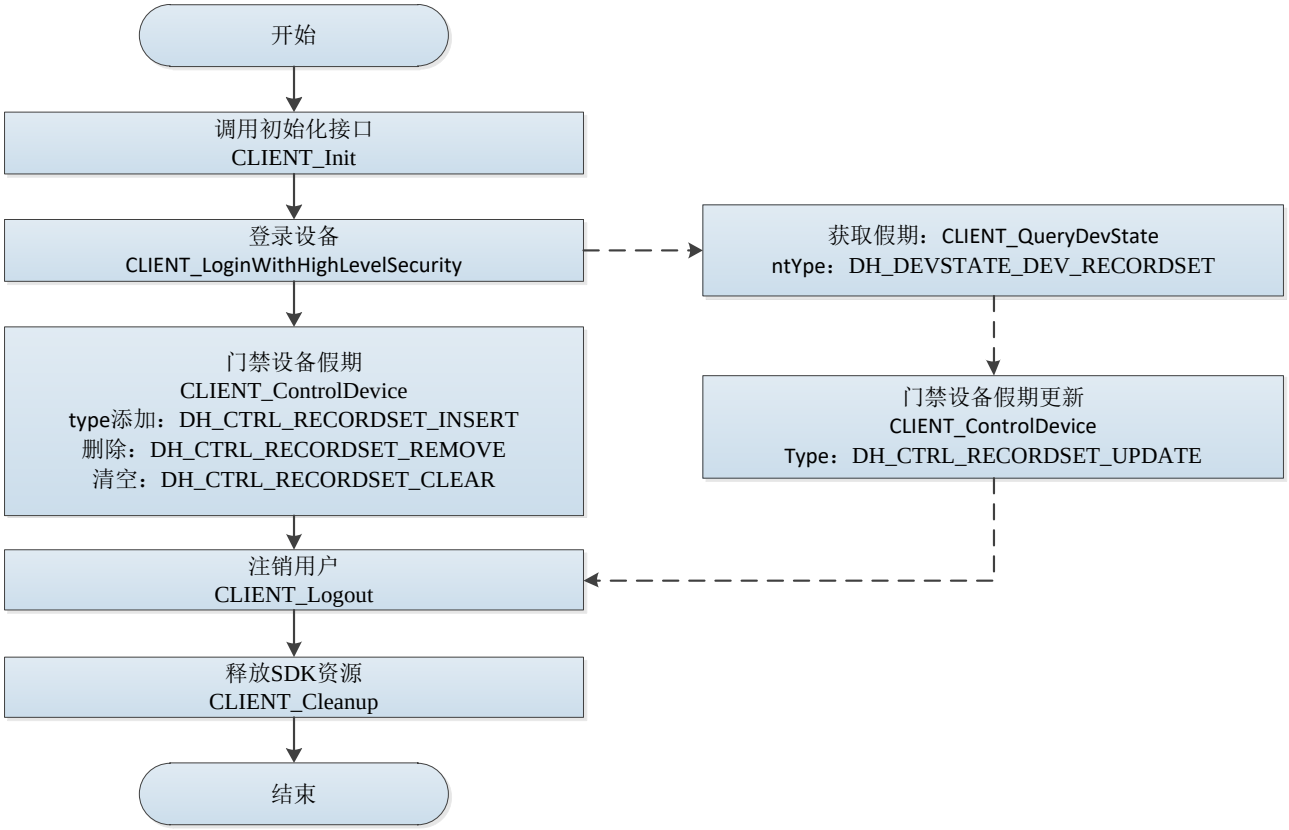
2.3.9.3.2 接口总览

表2-39 假期配置接口说明

接口	说明
CLIENT_ControlDevice	控制设备。
CLIENT_QueryDevState	查询设备状态。

2.3.9.3.3 流程说明

图2-32 假期配置流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDevice 函数来对假期信息进行操作。

表2-40 type 对应操作类型和结构体

type	对应含义	emType	Param
DH_CTRL_RECORDSET_INSERT	添加假期	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_INSERT_PARAM NET_RECORDSET_HOLIDAY
DH_CTRL_RECORDSET_REMOVE	删除假期	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY
DH_CTRL_RECORDSET_CLEAR	清空假期	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM

步骤4　先调用 CLIENT_QueryDevState 接口来获取假期信息。

表2-41 type 对应操作类型和结构体

type	对应含义	emType	Param
DH_DEVSTATE_DEV_RECORDSET	获取假期	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY

步骤5　再调用 CLIENT_ControlDevice 函数更新假期信息。

表2-42 type 对应操作类型和结构体

type	对应含义	emType	Param
DH_CTRL_RECORDSET_UPDATE	更新假期	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY

步骤6　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.9.3.4 示例代码

```
//添加假期
NET_RECORDSET_HOLIDAY stuInfo = {sizeof(stuInfo)};
    stuInfo.bEnable = TRUE;
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;
    stuInfo.stuEndTime.dwYear = 2019;
    stuInfo.stuEndTime.dwMonth = 12;
    stuInfo.stuEndTime.dwDay = 4;
    stuInfo.stuEndTime.dwHour = 12;
    stuInfo.stuEndTime.dwMinute = 22;
    stuInfo.stuEndTime.dwSecond = 12;

    stuInfo.stuStartTime.dwYear = 2019;
    stuInfo.stuStartTime.dwMonth = 12;
    stuInfo.stuStartTime.dwDay = 6;
    stuInfo.stuStartTime.dwHour = 12;
    stuInfo.stuStartTime.dwMinute = 22;
    stuInfo.stuStartTime.dwSecond = 12;

    memcpy(stuInfo.szHolidayName, "五一", sizeof(stuInfo.szHolidayName));
    memcpy(stuInfo.szHolidayNo, "12345", sizeof(stuInfo.szHolidayNo));


    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);
```

```
    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam,
5000);
//删除假期
stuInfo.nRecNo = 123456789;
NET_CTRL_RECORDSET_PARAM stuParam1 = {sizeof(stuParam1)};
    stuParam1.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam1.pBuf = (void*)&stuInfo.nRecNo;
    stuParam1.nBufLen = sizeof(stuInfo.nRecNo);

   BOOL bRet1 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam1,
5000);
//清空假期
    NET_CTRL_RECORDSET_PARAM stuParam = {sizeof(stuParam)};
    stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam,
5000);
//获取假期
stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
    stuParam3.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    NET_RECORDSET_HOLIDAY stuHoliday = {sizeof(stuHoliday)};
    stuHoliday.nRecNo = stuInfo.nRecNo;
    stuParam3.pBuf = &stuHoliday;
    int nRet = 0;
    BOOL bRet3 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam3,sizeof(stuParam3), &nRet, 5000);
//更新假期
stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam = {sizeof(stuParam)};
    stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam.pBuf = (void*)&stuInfo;
    stuParam.nBufLen = sizeof(stuInfo);

    int nRet = 0;
    BOOL bRet = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam,sizeof(stuParam), &nRet, 5000);
    if (bRet)
    {
        stuInfo.bEnable = TRUE;
        stuInfo.nDoorNum = 2;
```

```
        stuInfo.sznDoors[0] = 1223;
        stuInfo.stuEndTime.dwYear = 2019;
        stuInfo.stuEndTime.dwMonth = 10;
        stuInfo.stuEndTime.dwDay = 4;
        stuInfo.stuEndTime.dwHour = 12;
        stuInfo.stuEndTime.dwMinute = 22;
        stuInfo.stuEndTime.dwSecond = 12;

        stuInfo.stuStartTime.dwYear = 2019;
        stuInfo.stuStartTime.dwMonth = 12;
        stuInfo.stuStartTime.dwDay = 6;
        stuInfo.stuStartTime.dwHour = 12;
        stuInfo.stuStartTime.dwMinute = 22;
        stuInfo.stuStartTime.dwSecond = 12;

        memcpy(stuInfo.szHolidayName, "六一", sizeof(stuInfo.szHolidayName));
        memcpy(stuInfo.szHolidayNo, "12345", sizeof(stuInfo.szHolidayNo));

        stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
        stuParam.pBuf = (void*)&stuInfo;
        stuParam.nBufLen = sizeof(stuInfo);

        BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE,
&stuParam, 5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!");
    }
```

2.3.10 门高级配置

2.3.10.1 分时段、首卡开门

2.3.10.1.1 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。

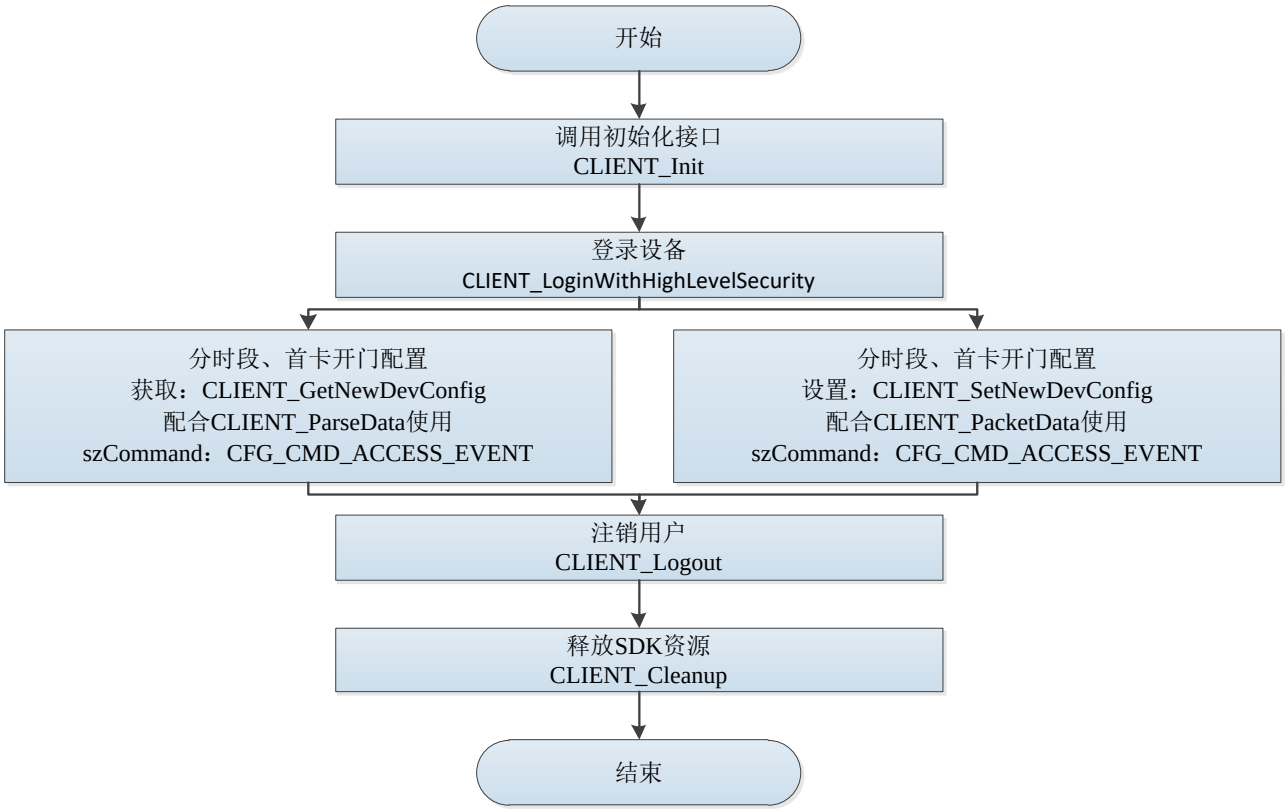
2.3.10.1.2 接口总览

表2-43 分时段和首卡开门接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.10.1.3 流程说明

图2-33 分时段和首卡开门业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁分时段、首卡开门配置信息。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-44 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 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁分时段、首卡开门配置信息。

- szCommand：CFG_CMD_ACCESS_EVENT。
- pBuf：CFG_ACCESS_EVENT_INFO。

表2-45 _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5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.10.1.4 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。

2.3.10.1.5 示例代码

```
// 获取分时段开门、首卡/用户开门配置信息
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("是否首卡/用户开门：%d\n",stOut2.stuFirstEnterInfo.bEnable);
            printf("首卡权限验证通过后的门禁状态：%d\n",stOut2.stuFirstEnterInfo.emStatus);
            printf("需要首卡验证的时间段：%d\n", stOut2.stuFirstEnterInfo.nTimeIndex);
        }
    }
    else{
        printf("parse failed!!!");
    }
    char * szOut = new char[1024*32];
    //首用户/卡开门配置
    stOut2.stuFirstEnterInfo.bEnable = TRUE;
    stOut2.stuFirstEnterInfo.emStatus = ACCESS_FIRSTENTER_STATUS_KEEPOPEN;
    stOut2.stuFirstEnterInfo.nTimeIndex = 0;
    //分时段开门配置
    stOut2.stuDoorTimeSection[0][0].emDoorOpenMethod = CFG_DOOR_OPEN_METHOD_PWD_ONLY;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwHour = 9;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwMinute = 11;
```

```
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwSecond = 45;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwHour = 19;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwMinute = 11;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwSecond = 45;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.2 多人组合开门

2.3.10.2.1 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。

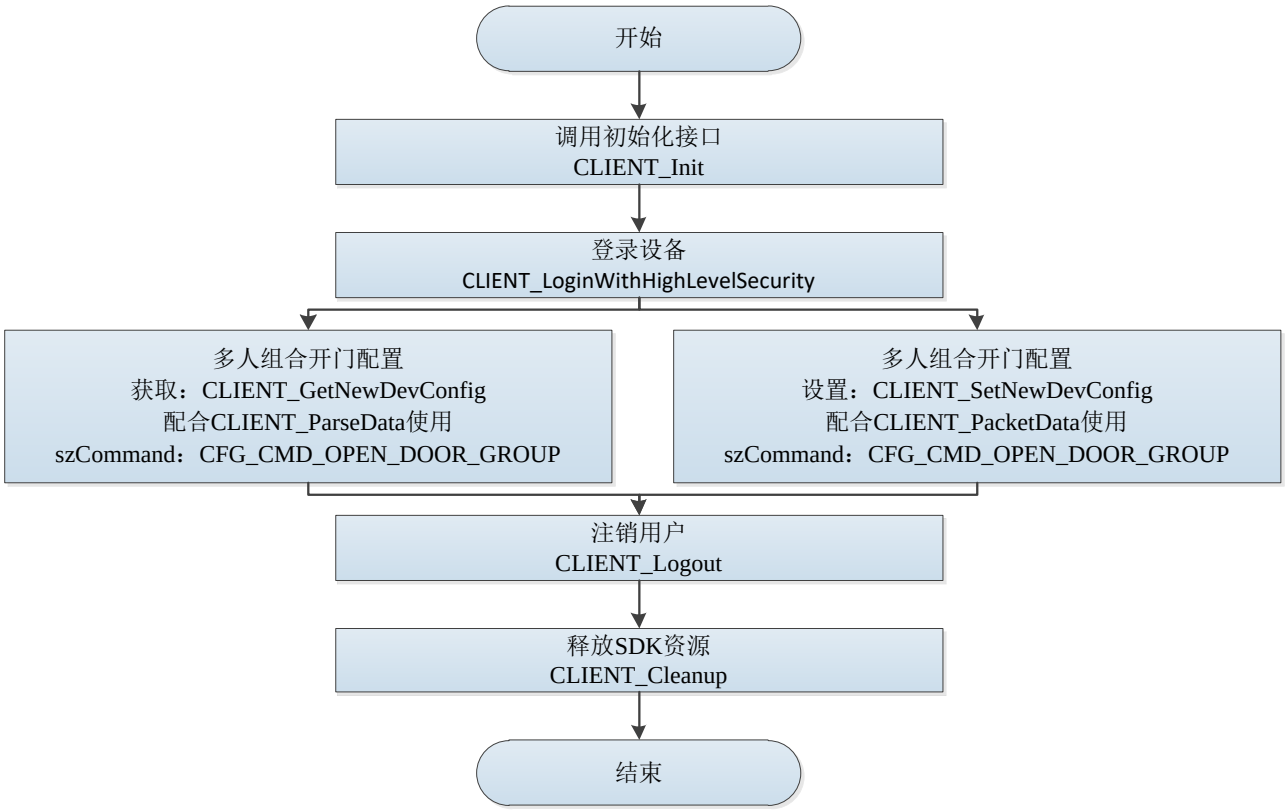
2.3.10.2.2 接口总览

表2-46 多人组合开门接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.10.2.3 流程说明

图2-34 多人组合开门业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁多人组合开门配置信息。

- szCommand：CFG_CMD_OPEN_DOOR_GROUP。
- pBuf：CFG_OPEN_DOOR_GROUP_INFO。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁多人组合开门配置信息。

- szCommand：CFG_CMD_OPEN_DOOR_GROUP。
- pBuf：CFG_OPEN_DOOR_GROUP_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.10.2.4 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。

2.3.10.2.5 示例代码

```
char * szOut1 = new char[1024*32];
```

```
    CFG_OPEN_DOOR_GROUP_INFO stOut2 = {sizeof(stOut2)};
    int nCount;
    CFG_OPEN_DOOR_GROUP_DETAIL* pstGroupDetail = new
CFG_OPEN_DOOR_GROUP_DETAIL[nCount];
    if (NULL == pstGroupDetail)
    {
        return;
    }
    memset(pstGroupDetail, 0, sizeof(CFG_OPEN_DOOR_GROUP_DETAIL)*nCount);

    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_GROUP, 0,
szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_OPEN_DOOR_GROUP, szOut1, &stOut2,
sizeof(CFG_OPEN_DOOR_GROUP_INFO), NULL);
        if (bRet1)
        {
            printf("有效组合数：%d\n",stOut2.nGroup);

            for (int i = 0; i < stOut2.nGroup; i++)
            {
                printf("[%d]分类组使能：%d\n", i, stOut2.stuGroupInfo[i].bGroupDetailEx);
                printf("[%d]用户数目：%d\n", i, stOut2.stuGroupInfo[i].nUserCount);
                printf("[%d]多人组合开门组的详细信息最大个数：%d\n", i,
stOut2.stuGroupInfo[i].nMaxGroupDetailNum);
                if (stOut2.stuGroupInfo[i].nMaxGroupDetailNum >
CFG_MAX_OPEN_DOOR_GROUP_DETAIL_NUM)
                {
                    for (int m = 0; m < stOut2.stuGroupInfo[i].nMaxGroupDetailNum; m++)
                    {
                        printf("[%d]-[%d]Method:%d\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].emMethod);
                        printf("[%d]-[%d]MethodExNum:%d\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].nMethodExNum);
                        for (int n = 0; n <
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].nMethodExNum; n++)
                        {
                            printf("[%d]-[%d]-[%d]MethodEx:%d\n", i, m, n,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].emMethodEx);
                        }
```

```
                                printf("[%d]-[%d]UserID:%s\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].szUserID);
                            }
                        }
                        printf("[%d]GroupNum：%d\n", i, stOut2.stuGroupInfo[i].nGroupNum);
                        for (int j = 0; j < stOut2.stuGroupInfo[i].nGroupNum; j++)
                        {
                            printf("[%d],[%d]用户 ID：%s\n", i, j,
stOut2.stuGroupInfo[i].stuGroupDetail[j].szUserID);
                        }

                    }
                }
            }
            else{
                printf("parse failed!!!");
            }
            char * szOut = new char[1024*32];

            stOut2.nGroup = 1;
            stOut2.stuGroupInfo[0].bGroupDetailEx = FALSE;
            stOut2.stuGroupInfo[0].nGroupNum = 1;
            stOut2.stuGroupInfo[0].stuGroupDetail[0].emMethod =
EM_CFG_OPEN_DOOR_GROUP_METHOD_ANY;

            BOOL bRet2 = CLIENT_PacketData(CFG_CMD_OPEN_DOOR_GROUP, (char *)&stOut2,
sizeof(CFG_OPEN_DOOR_GROUP_INFO), szOut, 1024*32);
            if(bRet2)
            {
                BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_GROUP, 0,
szOut, 1024*32, NULL, NULL, 3000);
                if (bRet3)
                {
                    printf("CLIENT_SetNewDevConfig Success!\n");
                }
                else{
                    printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
                }
            }
            else{
```

```
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.3 多门互锁

2.3.10.3.1 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。

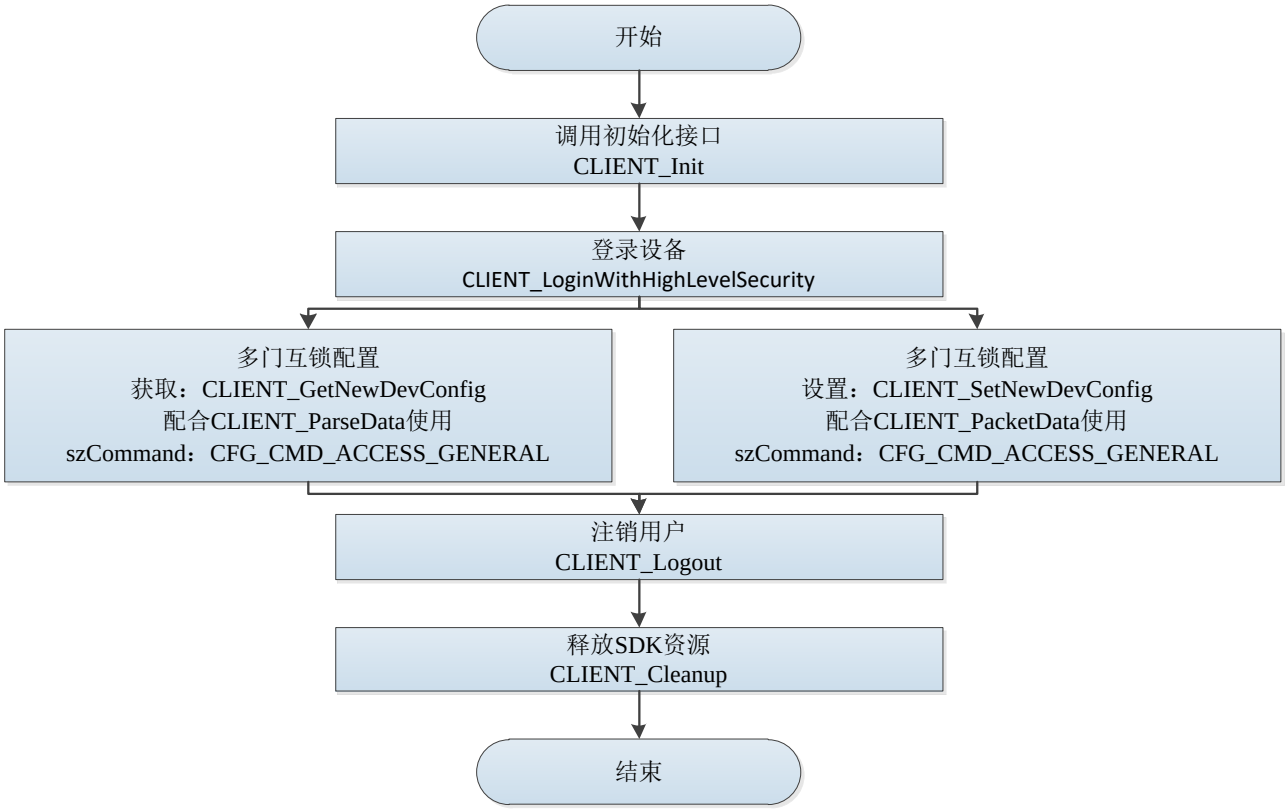
2.3.10.3.2 接口总览

表2-47 多门互锁接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.10.3.3 流程说明

图2-35 多门互锁配置业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁多门互锁配置信息。

- szCommand：CFG_CMD_ACCESS_GENERAL。
- pBuf：CFG_ACCESS_GENERAL_INFO。

步骤4 调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁多门互锁配置信息。

- szCommand：CFG_CMD_ACCESS_GENERAL。
- pBuf：CFG_ACCESS_GENERAL_INFO。

步骤5 业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6 SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.10.3.4 注意事项

1 个设备只能支持 1 种多门互锁方案。

2.3.10.3.5 示例代码

```
//获取多门互锁配置信息
char * szOut1 = new char[1024*32];
    CFG_ACCESS_GENERAL_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_GENERAL, 0, szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_GENERAL, szOut1, &stOut2, sizeof(CFG_ACCESS_GENERAL_INFO), NULL);
        if (bRet1)
        {
            printf("使能：%d\n",stOut2.stuABLockInfo.bEnable);
            printf("有效互锁组数：%d\n",stOut2.stuABLockInfo.nDoors);

            for (int i = 0; i < stOut2.stuABLockInfo.nDoors; i++)
            {
                printf("[%d]有效互锁门的个数：%d\n", i, stOut2.stuABLockInfo.stuDoors[i].nDoor);
                for (int j = 0; j < stOut2.stuABLockInfo.stuDoors[i].nDoor; j++)
                {
                    printf("[%d],[%d]互锁的门的通道号：%d\n", i, j, stOut2.stuABLockInfo.stuDoors[i].anDoor[j]);
                }
            }
        }
    }
    else{
        printf("parse failed!!!");
    }
```

```
//设置多门互锁配置信息
    char * szOut = new char[1024*32];

    stOut2.stuABLockInfo.bEnable = TRUE;
    stOut2.stuABLockInfo.nDoors = 1;
    stOut2.stuABLockInfo.stuDoors[0].nDoor = 2;
    stOut2.stuABLockInfo.stuDoors[0].anDoor[0] = 0;
    stOut2.stuABLockInfo.stuDoors[0].anDoor[0] = 1;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_GENERAL, (char *)&stOut2,
sizeof(CFG_ACCESS_GENERAL_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_GENERAL, 0,
szOut, 1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.4 防反潜

2.3.10.4.1 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。

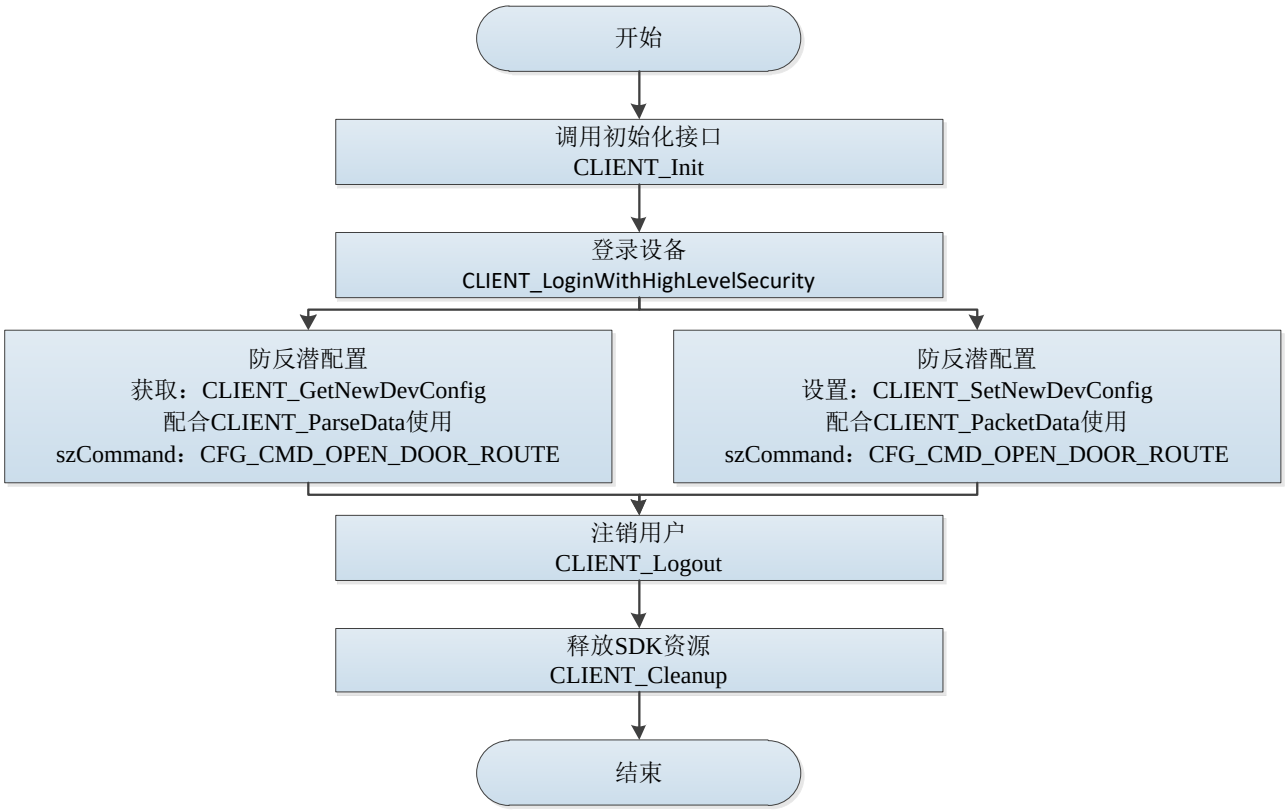
2.3.10.4.2 接口总览

表2-48 防反潜接口说明

接口	说明
CLIENT_GetNewDevConfig	查询配置信息。
CLIENT_ParseData	解析查询到的配置信息。
CLIENT_SetNewDevConfig	设置配置信息。
CLIENT_PacketData	将需要设置的配置信息，打包成字符串格式。

2.3.10.4.3 流程说明

图2-36 防反潜配置流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetNewDevConfig 函数，结合 CLIENT_ParseData 来查询门禁防反潜配置信息。

- szCommand：CFG_CMD_OPEN_DOOR_ROUTE。
- pBuf：CFG_OPEN_DOOR_ROUTE_INFO。

步骤4　调用 CLIENT_SetNewDevConfig 函数，结合 CLIENT_PacketData 来设置门禁防反潜配置信息。

- szCommand：CFG_CMD_OPEN_DOOR_ROUTE。
- pBuf：CFG_OPEN_DOOR_ROUTE_INFO。

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.10.4.4 注意事项

1 个设备只能支持 1 种防反潜方案。

2.3.10.4.5 示例代码

```
//获取防反潜配置信息
char * szOut1 = new char[1024*32];
    CFG_OPEN_DOOR_ROUTE_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
```

```
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_ROUTE, 0,
szOut1, 1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_OPEN_DOOR_ROUTE, szOut1, &stOut2,
sizeof(CFG_OPEN_DOOR_ROUTE_INFO), NULL);
        if (bRet1)
        {
            printf("反潜重置时间：%d\n",stOut2.nResetTime);
            printf("门列表数：%d\n",stOut2.nDoorList);
            printf("反潜路径对应时间段：%d\n",stOut2.nTimeSection);

            for (int i = 0; i < stOut2.nDoorList; i++)
            {
                printf("[%d]反潜重置时间：%d\n", i, stOut2.stuDoorList[i].nResetTime);
                printf("[%d]开门路线有效节点数：%d\n", i, stOut2.stuDoorList[i].nDoors);
                for (int j = 0; j < stOut2.stuDoorList[i].nDoors; j++)
                {
                    printf("[%d],[%d]读卡器 ID：%s\n", i, j,
stOut2.stuDoorList[i].stuDoors[j].szReaderID);
                }
            }
        }
    }
    else{
        printf("parse failed!!!");
    }
//配置防反潜配置信息
    char * szOut = new char[1024*32];

    stOut2.nDoorList = 1;
    stOut2.nResetTime = 1;
    stOut2.nTimeSection = 2;
    stOut2.stuDoorList[0].nResetTime = 0;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_OPEN_DOOR_ROUTE, (char *)&stOut2,
sizeof(CFG_OPEN_DOOR_ROUTE_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_ROUTE, 0,
szOut, 1024*32, NULL, NULL, 3000);
```

```
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.5 开门密码

2.3.10.5.1 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。

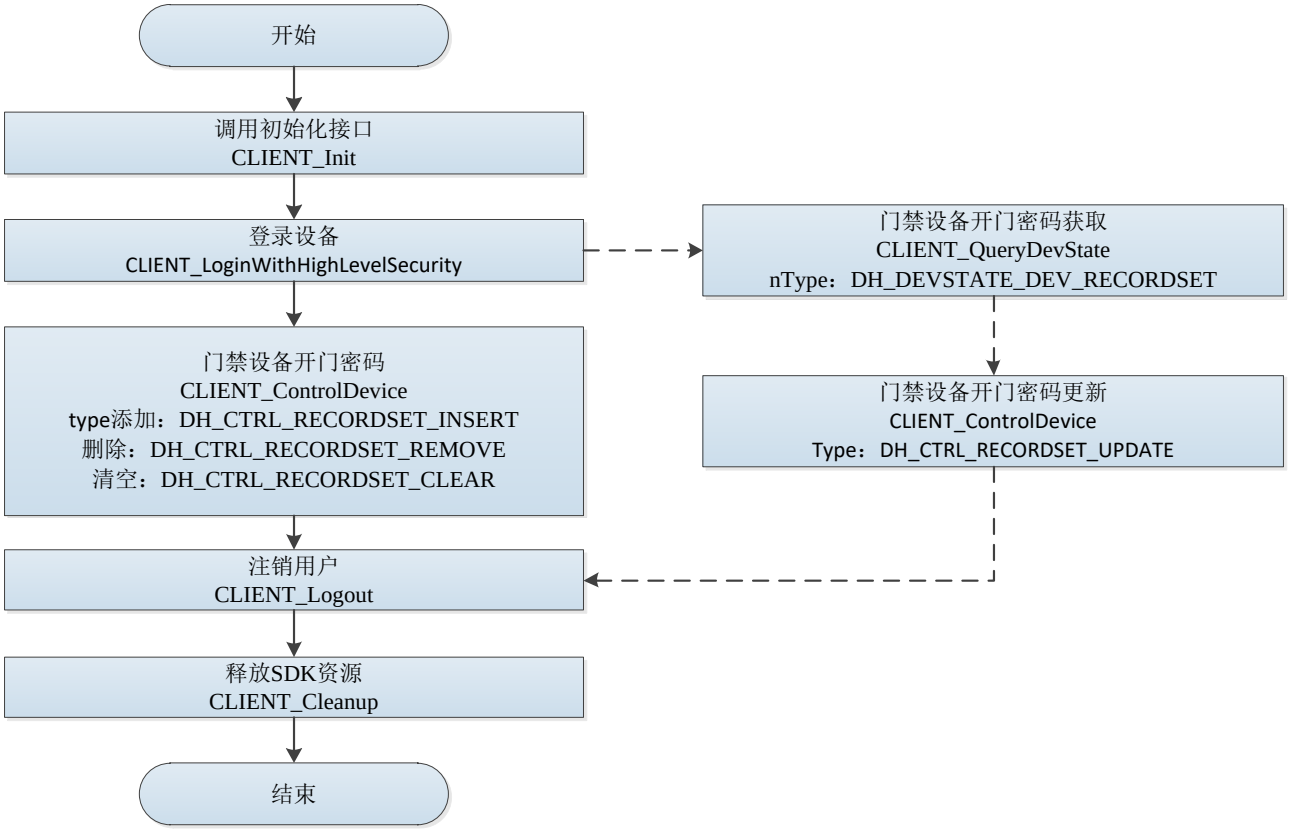
2.3.10.5.2 接口总览

表2-49 开门密码接口说明

接口	说明
CLIENT_ControlDevice	设备控制。

2.3.10.5.3 流程说明

图2-37 开门密码配置流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_ControlDevice 函数来对开门密码信息进行操作。

表2-50 Type 对应操作和结构体

type	含义	emType	Param
DH_CTRL_RECORDSET_INSERT	添加开门密码	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_INSERT_PARA NET_RECORDSET_ACCESS_CTL_PWD
DH_CTRL_RECORDSET_REMOVE	删除开门密码	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD
DH_CTRL_RECORDSET_CLEAR	清空开门密码	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM

步骤4　先调用 CLIENT_QueryDevState 接口来获取开门密码信息。

表2-51 Type 对应操作和结构体

type	含义	emType	Param
DH_DEVSTATE_DEV_RECORDSET	获取开门密码	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD

步骤5　再调用 CLIENT_ControlDevice 函数更新开门密码信息。

表2-52 Type 对应操作和结构体

type	含义	emType	Param
DH_CTRL_RECORDSET_UPDATE	获取开门密码	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD

步骤6　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤7　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.10.5.4 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。

2.3.10.5.5 示例代码

```
NET_RECORDSET_ACCESS_CTL_PWD stuInfo = {sizeof(stuInfo)};

    //新增
    stuInfo.bNewDoor = TRUE;
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;

    memcpy(stuInfo.szUserID, "11234", sizeof(stuInfo.szUserID));
    memcpy(stuInfo.szDoorOpenPwd, "12345", sizeof(stuInfo.szDoorOpenPwd));

    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);

    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam,
5000);
    //先获取再更新
    stuInfo.nRecNo = 123456;
    NET_CTRL_RECORDSET_PARAM stuParam2 = {sizeof(stuParam2)};
    stuParam2.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam2.pBuf = (void*)&stuInfo;
    stuParam2.nBufLen = sizeof(stuInfo);
```

```
    int nRet = 0;
    BOOL bRet1 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam2,
        sizeof(stuParam2), &nRet, 5000);
    if (bRet)
    {

        stuParam2.emType = NET_RECORD_ACCESSCTLPWD;
        stuParam2.pBuf = (void*)&stuInfo;
        stuParam2.nBufLen = sizeof(stuInfo);

        // update info
        BOOL bRet2 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE,
&stuParam2, 5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!\n");
    }
    //删除
    stuInfo.nRecNo = 123456;
    NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
    stuParam3.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam3.pBuf = (void*)&stuInfo.nRecNo;
    stuParam3.nBufLen = sizeof(stuInfo.nRecNo);

   BOOL bRet3 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam3,
5000);
    //清空
    NET_CTRL_RECORDSET_PARAM stuParam4 = {sizeof(stuParam4)};
    stuParam4.emType = NET_RECORD_ACCESSCTLPWD;
    BOOL bRet4 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam4,
5000);
```

2.3.11 记录查询

2.3.11.1 开门记录

2.3.11.1.1 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。

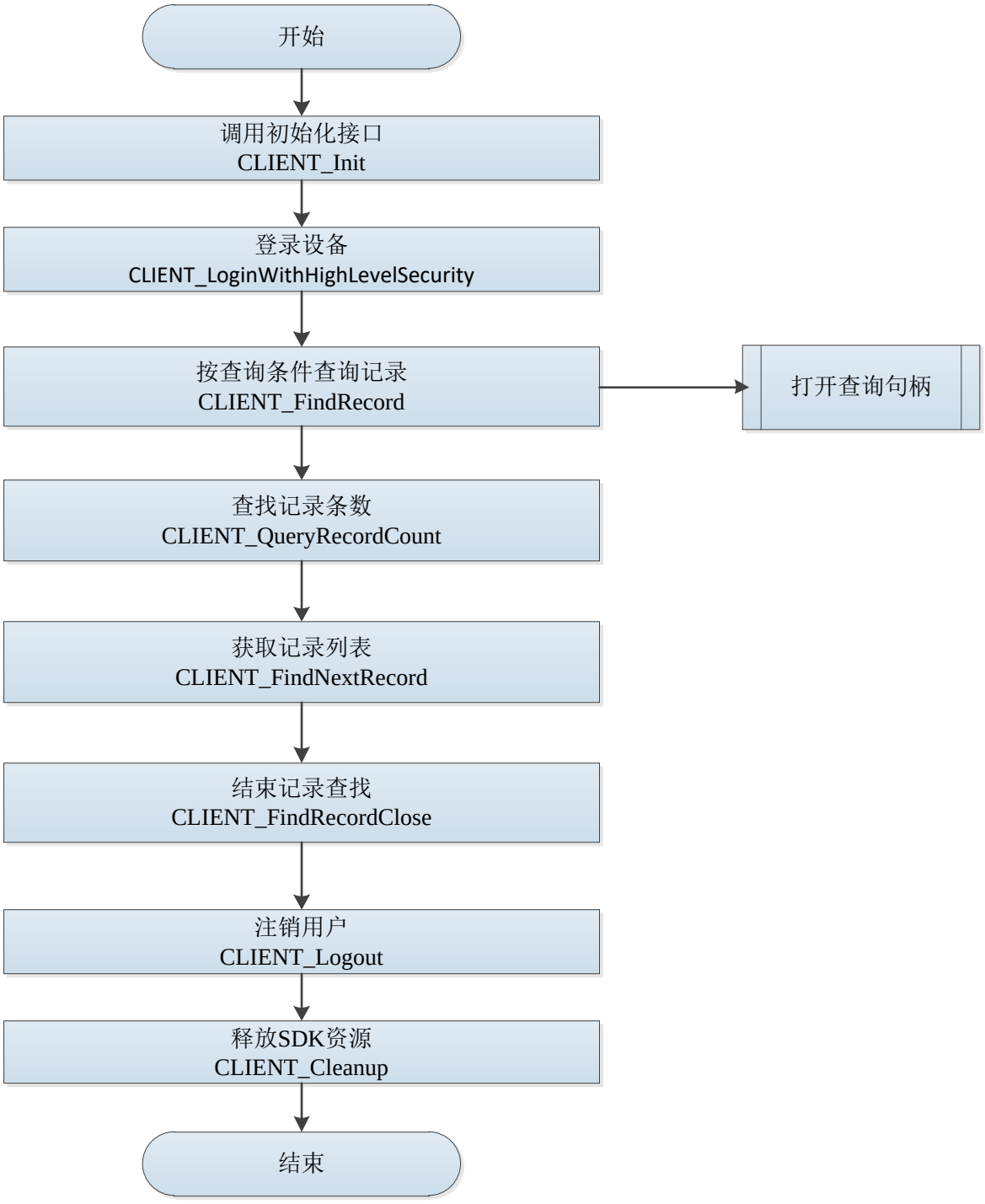
2.3.11.1.2 接口总览

表2-53 记录查询接口说明

接口	说明
CLIENT_QueryRecordCount	查找记录条数
CLIENT_FindRecord	按查询条件查询记录
CLIENT_FindNextRecord	查找记录:nFilecount:需要查询的条数，返回值为媒体文件条数 返回值小于 nFilecount 则相应时间段内的文件查询完毕
CLIENT_FindRecordClose	结束记录查询

2.3.11.1.3 流程说明

图2-38 记录查询业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　先调用 CLIENT_FindRecord 函数获得查询句柄。

emType 开门记录：NET_RECORD_ACCESSCTLCARDREC。

步骤4　再调用 CLIENT_QueryRecordCount 函数查找记录条数。

步骤5　再调用 CLIENT_FindNextRecord 函数获取记录列表。

步骤6　查询完毕可以调用 CLIENT_FindRecordClose 关闭查询句柄。

步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.3.11.1.4 示例代码

```
NET_IN_FIND_RECORD_PARAM stuIn = {sizeof(stuIn)};
    NET_OUT_FIND_RECORD_PARAM stuOut = {sizeof(stuOut)};

    stuIn.emType = NET_RECORD_ACCESS_ALARMRECORD;

    if (CLIENT_FindRecord(g_lLoginHandle, &stuIn, &stuOut, 5000))
    {
        printf("CLIENT_FindRecord success!\n");
    }
    else{
        printf("CLIENT_FindRecord failed!\n");
    }

    NET_IN_QUEYT_RECORD_COUNT_PARAM stuInCount = {sizeof(stuInCount)};
    stuInCount.lFindeHandle = stuOut.lFindeHandle;
    NET_OUT_QUEYT_RECORD_COUNT_PARAM stuOutCount = {sizeof(stuOutCount)};
    if (CLIENT_QueryRecordCount(&stuInCount, &stuOutCount, 5000))
    {
        printf("CLIENT_QueryRecordCount success!\n");
    }
    else{
        printf("CLIENT_QueryRecordCount failed!\n");
    }

    int i = 0, j = 0;
    int nMaxNum = 10;
    NET_IN_FIND_NEXT_RECORD_PARAM stuIn1 = {sizeof(stuIn1)};
    stuIn1.lFindeHandle = stuOut.lFindeHandle;
    stuIn1.nFileCount = nMaxNum;

    NET_OUT_FIND_NEXT_RECORD_PARAM stuOut2 = {sizeof(stuOut2)};
    stuOut2.nMaxRecordNum = nMaxNum;

    NET_RECORDSET_ACCESS_CTL_CARD* pstuCard = new
NET_RECORDSET_ACCESS_CTL_CARD[nMaxNum];
    if (NULL == pstuCard)
```

```
    {
        return;
    }
    memset(pstuCard, 0, sizeof(NET_RECORDSET_ACCESS_CTL_CARD) * nMaxNum);

    for (i = 0; i < nMaxNum; i++)
    {
        pstuCard[i].dwSize = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);
    }
    stuOut2.pRecordList = (void*)pstuCard;

    if (CLIENT_FindNextRecord(&stuIn1, &stuOut2, 5000) >= 0)
    {
        printf("CLIENT_FindNextRecord success!\n");
    }
    else
    {
        printf("CLIENT_FindNextRecord failed!\n");
    }
CLIENT_FindRecordClose(stuOut.lFindeHandle);
```

2.3.11.2 设备日志

2.3.11.2.1 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。

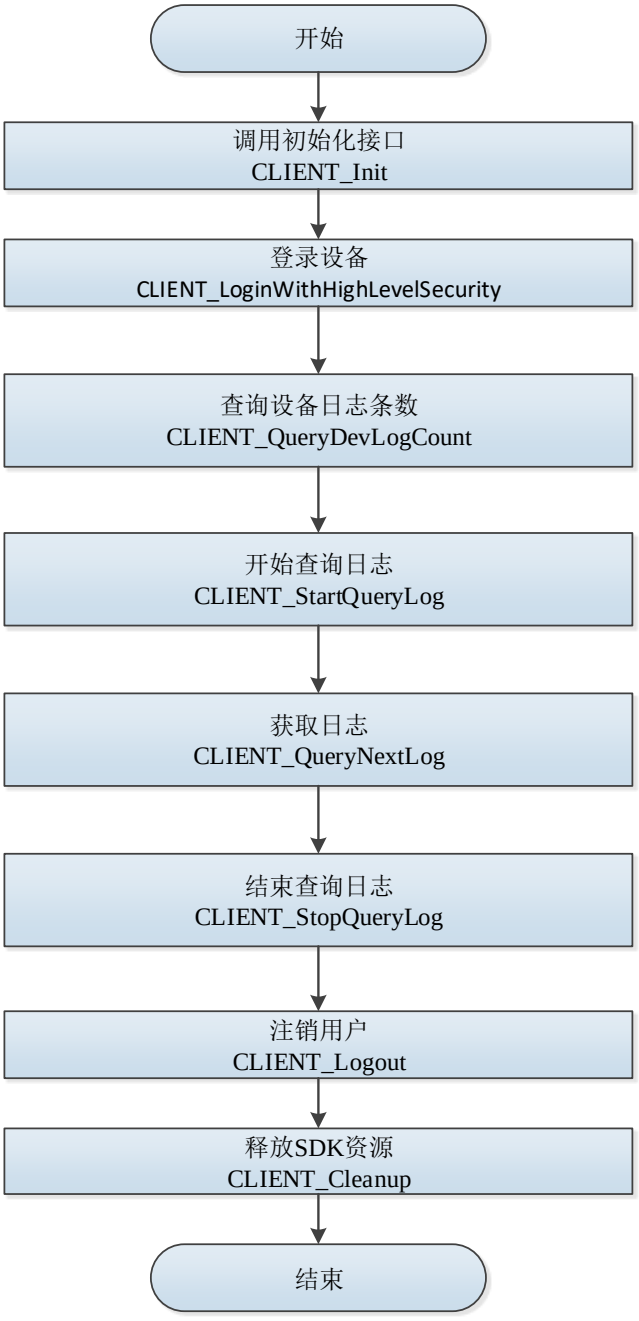
2.3.11.2.2 接口总览

表2-54 设备日志接口说明

接口	说明
CLIENT_QueryDevLogCount	查询设备日志条数。
CLIENT_StartQueryLog	开始查询日志。
CLIENT_QueryNextLog	获取日志。
CLIENT_StopQueryLog	结束查询日志。

2.3.11.2.3 流程说明

图2-39 设备日志业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_QueryDevLogCount 函数设置查询日志条数。

步骤4　调用 CLIENT_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 CLIENT_QueryNextLog 函数获取日志信息。

pInParam：NET_IN_QUERYNEXTLOG。

pOutParam：NET_OUT_QUERYNEXTLOG。

步骤6　调用 CLIENT_StopQueryLog 函数停止日志查询。

步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

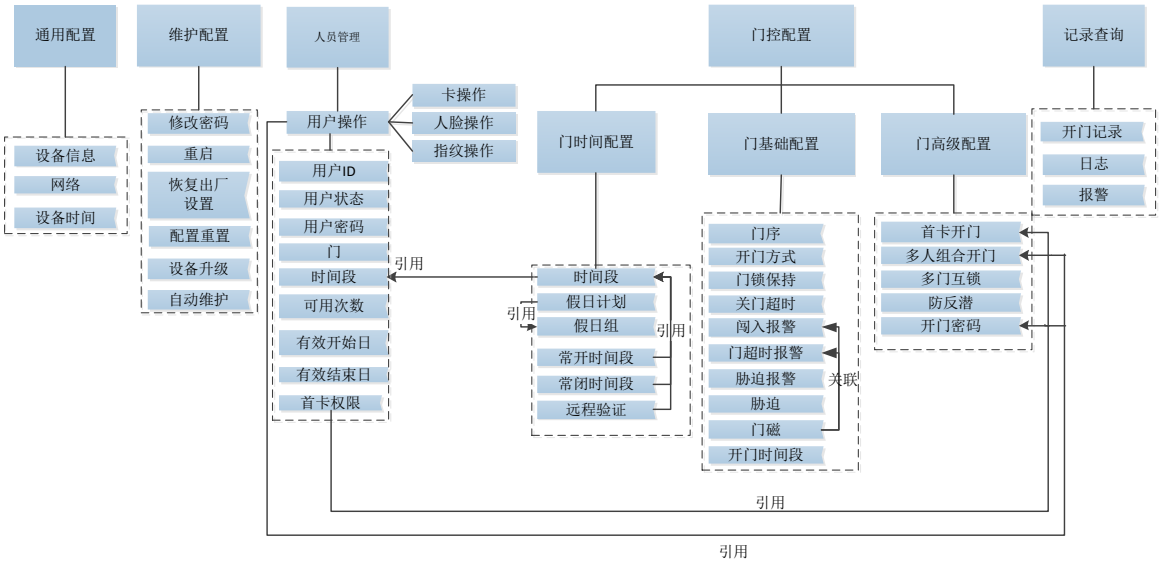
2.3.11.2.4 示例代码

```
// 开始查询日志信息
NET_IN_START_QUERYLOG stuIn = {sizeof(stuIn)};
NET_OUT_START_QUERYLOG stuOut = {sizeof(stuOut)};
LLONG lLogID = CLIENT_StartQueryLog(m_lLoginId, &stuIn, &stuOut, 5000);
//获取日志信息
NET_IN_QUERYNEXTLOG stuIn = {sizeof(stuIn)};
stuIn.nGetCount = m_nMaxPageSize;
NET_OUT_QUERYNEXTLOG stuOut = {sizeof(stuOut)};
stuOut.nMaxCount = 60;
stuOut.pstuLogInfo = new NET_LOG_INFO[60];
if (NULL == stuOut.pstuLogInfo)
{
	return -1;
}
memset(stuOut.pstuLogInfo, 0, sizeof(NET_LOG_INFO) * m_nMaxPageSize);
for (int i = 0; i < m_nMaxPageSize; i++)
{
	stuOut.pstuLogInfo[i].dwSize = sizeof(NET_LOG_INFO);
	stuOut.pstuLogInfo[i].stuLogMsg.dwSize = sizeof(NET_LOG_MESSAGE);
}
BOOL bRet = CLIENT_QueryNextLog(m_lLogID, &stuIn, &stuOut, 5000);
//停止查询日志信息
BOOL bRet0 = CLIENT_StopQueryLog(m_lLogID);
```

2.4 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

图2-40 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。

2.4.1 门禁控制

请参见“2.3.1 门禁控制”。

2.4.2 报警事件

请参见“2.3.2 报警事件”

2.4.3 设备信息查看

2.4.3.1 能力集查询

2.4.3.1.1 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。

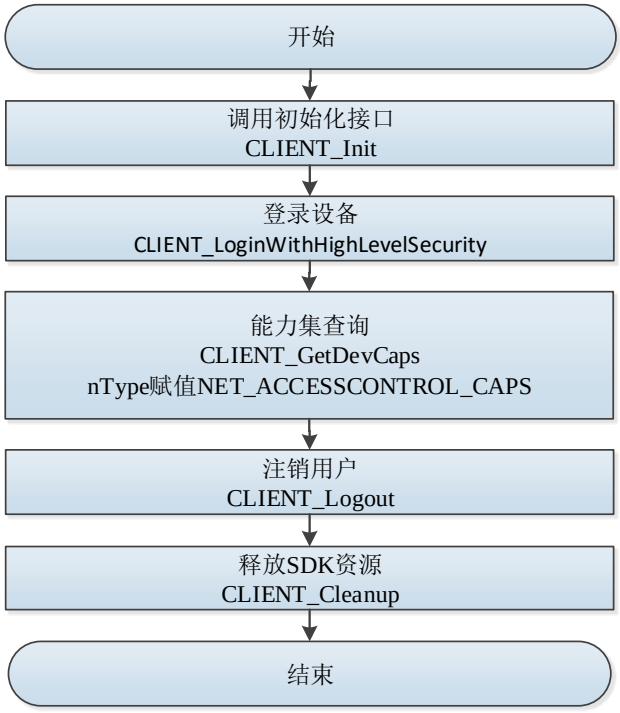
2.4.3.1.2 接口总览

表2-55 能力集查询接口说明

接口	说明
CLIENT_GetDevCaps	获取门禁能力（包括门禁、用户、卡、人脸、指纹能力）。

2.4.3.1.3 流程说明

图2-41 设备信息查看流程图



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_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 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.3.1.4 示例代码

```
NET_IN_AC_CAPS stuIn = {sizeof(stuIn)};
    NET_OUT_AC_CAPS stuOut = {sizeof(stuOut)};
    BOOL bRet = CLIENT_GetDevCaps(m_lLoginID, NET_ACCESSCONTROL_CAPS,&stuIn, &stuOut,5000);
    if (bRet)
    {
        NET_ACCESS_USER_CAPS     stuUserCaps = stuOut.stuUserCaps;
    }
    else
    {
         return FALSE;
    }
```

2.4.3.2 设备版本、MAC 查看

请参见“2.3.3.2 设备版本、MAC 查看”。

2.4.4 网络配置

请参见“2.3.4 网络配置”。

2.4.5 设备时间设置

请参见“2.3.5 设备时间设置”。

2.4.6 维护配置

请参见“2.3.6 维护配置”。

2.4.7 人员管理

2.4.7.1 用户管理

2.4.7.1.1 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。

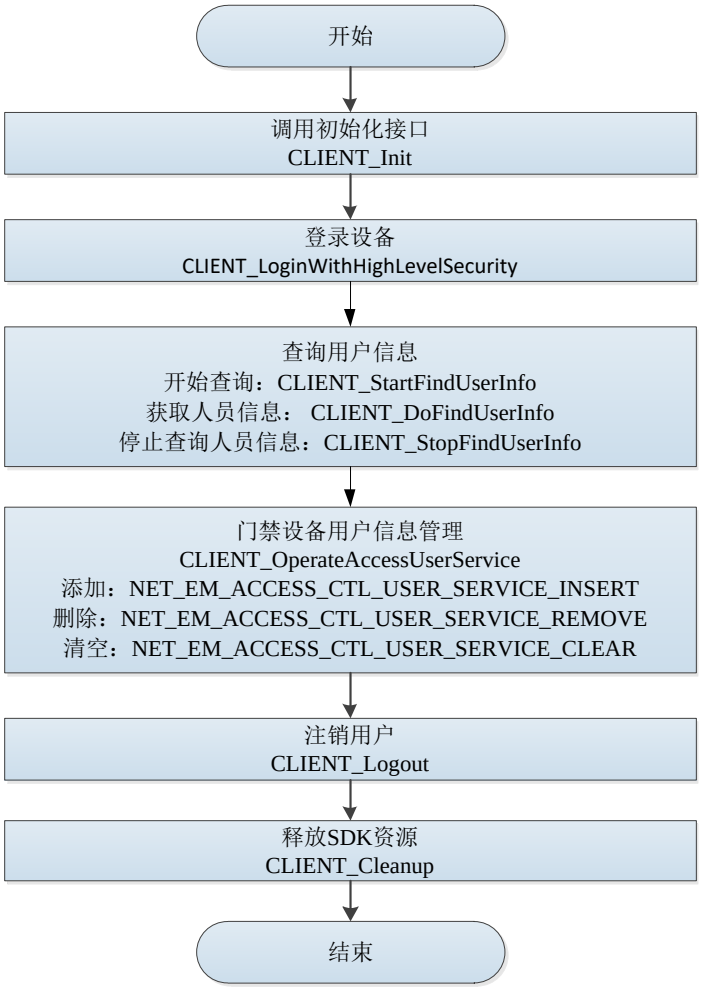
2.4.7.1.2 接口总览

表2-56 用户信息接口说明

接口	说明
CLIENT_OperateAccessUserService	门禁用户信息管理接口
CLIENT_StartFindUserInfo	开始查询用户信息
CLIENT_DoFindUserInfo	获取用户信息
CLIENT_StopFindUserInfo	停止查询用户信息

2.4.7.1.3 流程说明

图2-42 用户信息管理业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　调用 CLIENT_StartFindUserInfo 开始查询用户信息。
步骤4　调用 CLIENT_DoFindUserInfo 获取用户信息。
步骤5　调用 CLIENT_StopFindUserInfo 停止查询用户信息。
步骤6　调用 CLIENT_OperateAccessUserService 函数来对用户信息进行添加、删除、清空操作。
步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.7.1.4 示例代码

```
//获取
NET_IN_USERINFO_START_FIND stuStartIn = {sizeof(stuStartIn)};
    NET_OUT_USERINFO_START_FIND stuStartOut = {sizeof(stuStartOut)};
    stuStartOut.nTotalCount = 0;
```

```
    stuStartOut.nCapNum = 10;
    LLONG UserFindId = CLIENT_StartFindUserInfo(m_lLoginID, &stuStartIn,
&stuStartOut,SDK_API_WAIT);
    if (UserFindId != NULL)
    {
        m_UserInfoVector.clear();
        //
        int nStartNo = 0;
        m_bIsDoFindNext = TRUE;
        while (m_bIsDoFindNext)
        {
            NET_ACCESS_USER_INFO* pUserInfo = new NET_ACCESS_USER_INFO[10];
            if (pUserInfo)
            {
                int nRecordNum = 0;
                NET_IN_USERINFO_DO_FIND stuFindIn = {sizeof(stuFindIn)};
                stuFindIn.nStartNo = nStartNo;
                stuFindIn.nCount = 10;
                NET_OUT_USERINFO_DO_FIND stuFindOut = {sizeof(stuFindOut)};
                stuFindOut.nMaxNum = 10;
                stuFindOut.pstuInfo = pstuAlarm;
                if (CLIENT_DoFindUserInfo(m_UserFindId, &stuFindIn, &stuFindOut, SDK_API_WAIT))
                {
                    if (stuFindOut.nRetNum > 0)
                    {
                        nRecordNum = stuFindOut.nRetNum;
                        m_bIsDoFindNext = TURE;
                    }
                }
                m_bIsDoFindNext = FALSE;
                for (int i=0;i<nRecordNum;i++)
                {
                    NET_ACCESS_USER_INFO stuUserInfo;
                    memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
                    memcpy(&stuUserInfo,&pUserInfo[i],sizeof(NET_ACCESS_USER_INFO));
                    m_UserInfoVector.push_back(stuUserInfo);
                }
                nStartNo = nRecordNum;
                delete []pUserInfo;
                pUserInfo = NULL;
```

```
            }
            else
            {
                m_bIsDoFindNext = FALSE;
            }
        }
        CLIENT_StopFindUserInfo(m_UserFindId);
        return TRUE;
    }
    else
    {
        return FALSE;
    }

//新增
NET_ACCESS_USER_INFO stuUserInfo;
memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
memcpy(&stuUserInfo,pstuUserInfo,sizeof(NET_ACCESS_USER_INFO));
NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
NET_IN_ACCESS_USER_SERVICE_INSERT stuUserInsertIn = {sizeof(stuUserInsertIn)};
    stuUserInsertIn.nInfoNum = nNum;
    stuUserInsertIn.pUserInfo = &stuUserInfo;
    NET_OUT_ACCESS_USER_SERVICE_INSERT stuUserINsertOut = {sizeof(stuUserINsertOut)};
    stuUserINsertOut.nMaxRetNum = nNum;
    stuUserINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID,
NET_EM_ACCESS_CTL_USER_SERVICE_INSERT, &stuUserInsertIn, &stuUserINsertOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//删除
NET_IN_ACCESS_USER_SERVICE_REMOVE stuUserRemoveIn = {sizeof(stuUserRemoveIn)};
    stuUserRemoveIn.nUserNum = 1;
    NET_ACCESS_USER_INFO stuUserInfo;
    memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
    memcpy(&stuUserInfo,&m_UserInfoVector[m_nUserInfoIndex],sizeof(NET_ACCESS_USER_INFO));
    strncpy(stuUserRemoveIn.szUserID[0],stuUserInfo.szUserID,DH_MAX_USERID_LEN-1);
```

```
    NET_OUT_ACCESS_USER_SERVICE_REMOVE stuUserRemoveOut = {sizeof(stuUserRemoveOut)};
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuUserRemoveOut.nMaxRetNum = 1;
    stuUserRemoveOut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID,
NET_EM_ACCESS_CTL_USER_SERVICE_REMOVE, stuUserRemoveIn, &stuUserRemoveOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
//清空
NET_IN_ACCESS_USER_SERVICE_CLEAR stuUserClearIn = {sizeof(stuUserClearIn)};
    NET_OUT_ACCESS_USER_SERVICE_CLEAR stuUserClearOut = {sizeof(stuUserClearIn)};
    BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID,
NET_EM_ACCESS_CTL_USER_SERVICE_CLEAR, &stuUserClearIn, &stuUserClearOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
```

2.4.7.2 卡片管理

2.4.7.2.1 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。

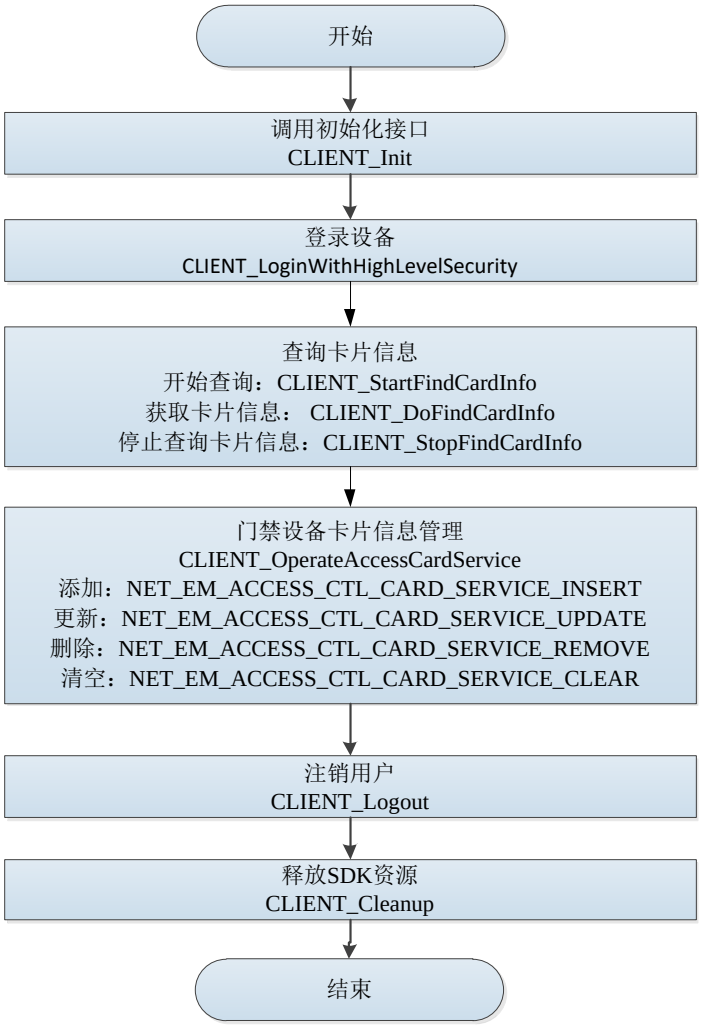
2.4.7.2.2 接口总览

表2-57 卡片信息接口说明

接口	说明
CLIENT_OperateAccessCardService	门禁卡片信息管理接口
CLIENT_StartFindCardInfo	开始查询卡片信息
CLIENT_DoFindCardInfo	获取卡片信息
CLIENT_StopFindCardInfo	停止查询卡片信息

2.4.7.2.3 流程说明

图2-43 卡片信息管理业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　调用 CLIENT_StartFindCardInfo 开始查询卡片信息。
步骤4　调用 CLIENT_DoFindCardInfo 获取卡片信息。
步骤5　调用 CLIENT_StopFindCardInfo 停止查询卡片信息。
步骤6　调用 CLIENT_OperateAccessCardService 函数来对卡片信息进行添加、更新、删除、清空操作。
步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.7.2.4 示例代码

```
//获取
NET_IN_CARDINFO _START_FIND stuStartIn = {sizeof(stuStartIn)};
    NET_OUT_CARDINFO _START_FIND stuStartOut = {sizeof(stuStartOut)};
```

```
    stuStartOut.nTotalCount = 0;
    stuStartOut.nCapNum = 10;
    LLONG CardFindId = CLIENT_StartFindCardInfo(m_lLoginID, &stuStartIn,
&stuStartOut,SDK_API_WAIT);
    if (CardFindId != NULL)
    {
        m_CardInfoVector.clear();
        //
        int nStartNo = 0;
        m_bIsDoFindNext = TRUE;
        while (m_bIsDoFindNext)
        {
            NET_ACCESS_CARD_INFO* pCardInfo = new NET_ACCESS_CARD_INFO[10];
            if (pCardInfo)
            {
                int nRecordNum = 0;
                NET_IN_CARDINFO_DO_FIND stuFindIn = {sizeof(stuFindIn)};
                stuFindIn.nStartNo = nStartNo;
                stuFindIn.nCount = 10;
                NET_OUT_CARDINFO_DO_FIND stuFindOut = {sizeof(stuFindOut)};
                stuFindOut.nMaxNum = 10;
                stuFindOut.pstuInfo = pstuAlarm;
                if (CLIENT_DoFindCardInfo(m_CardFindId, &stuFindIn, &stuFindOut, SDK_API_WAIT))
                {
                    if (stuFindOut.nRetNum > 0)
                    {
                        nRecordNum = stuFindOut.nRetNum;
                        m_bIsDoFindNext = TURE;
                    }
                }
                m_bIsDoFindNext = FALSE;
                for (int i=0;i<nRecordNum;i++)
                {
                    NET_ACCESS_CARD_INFO stuCardInfo;
                        memset(&stuCardInfo,0,sizeof(NET_ACCESS_CARD_INFO));
                        memcpy(&stuCardInfo,&pCardInfo[i],sizeof(NET_ACCESS_CARD_INFO));
                        m_CardInfoVector.push_back(stuCardInfo);
                }
                nStartNo = nRecordNum;
                delete []pCardInfo;
```

```
                pCardInfo = NULL;
            }
            else
            {
                m_bIsDoFindNext = FALSE;
            }
        }
        CLIENT_StopFindCardInfo (m_CardFindId);
        return TRUE;
    }
    else
    {
        return FALSE;
    }

//新增
NET_ACCESS_CARD_INFO stuCardInfo;
    memset(&stuCardInfo,0,sizeof(stuCardInfo));
    memcpy(&stuCardInfo, pstuCardInfo, sizeof(stuCardInfo));
    memcpy(stuCardInfo.szUserID,m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    NET_IN_ACCESS_CARD_SERVICE_INSERT stuCardInsertIn = {sizeof(stuCardInsertIn)};
    stuCardInsertIn.nInfoNum = nNum;
    stuCardInsertIn.pCardInfo = &stuCardInfo;
    NET_OUT_ACCESS_CARD_SERVICE_INSERT stuCardINsertOut = {sizeof(stuCardINsertOut)};
    stuCardINsertOut.nMaxRetNum = nNum;
    stuCardINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_INSERT, &stuCardInsertIn, &stuCardINsertOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//更新
NET_ACCESS_CARD_INFO stuCardInfo;
    memset(&stuCardInfo,0,sizeof(stuCardInfo));
    memcpy(&stuCardInfo, pstuCardInfo, sizeof(stuCardInfo));
    memcpy(stuCardInfo.szUserID,m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
```

```
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
NET_IN_ACCESS_CARD_SERVICE_UPDATE stuCardUpdateIn = {sizeof(stuCardUpdateIn)};
    stuCardUpdateIn.nInfoNum = nNum;
    stuCardUpdateIn.pCardInfo = &stuCardInfo;
    NET_OUT_ACCESS_CARD_SERVICE_UPDATE stuCardUpdateOut = {sizeof(stuCardUpdateOut)};
    stuCardUpdateOut.nMaxRetNum = nNum;
    stuCardUpdateOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_UPDATE, &stuCardUpdateIn, &stuCardUpdateOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//删除
NET_IN_ACCESS_CARD_SERVICE_REMOVE stuCardRemoveIn = {sizeof(stuCardRemoveIn)};
    stuCardRemoveIn.nCardNum = 1;
    NET_ACCESS_CARD_INFO stuCardInfo = m_CardInfoVector[m_nCardIndex];
    memcpy(&stuCardRemoveIn.szCardNo[0], stuCardInfo.szCardNo, sizeof(stuCardInfo.szCardNo));
    NET_OUT_ACCESS_CARD_SERVICE_REMOVE stuCardRemoveOut = {sizeof(stuCardRemoveOut)};
    stuCardRemoveOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    stuCardRemoveOut.pFailCode = &stuFailCode;
BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_REMOVE, &stuCardRemoveIn, &stuCardRemoveOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//清空
NET_IN_ACCESS_CARD_SERVICE_CLEAR stuCardClearIn = {sizeof(stuCardClearIn)};
    NET_OUT_ACCESS_CARD_SERVICE_CLEAR stuCardClearOut = {sizeof(stuCardClearOut)};
    BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_CLEAR, &stuCardClearIn, &stuCardClearOut, SDK_API_WAIT);
    if (bRet)
```

```
{
    return TRUE;
}
return FALSE;
```

2.4.7.3 人脸管理

2.4.7.3.1 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。

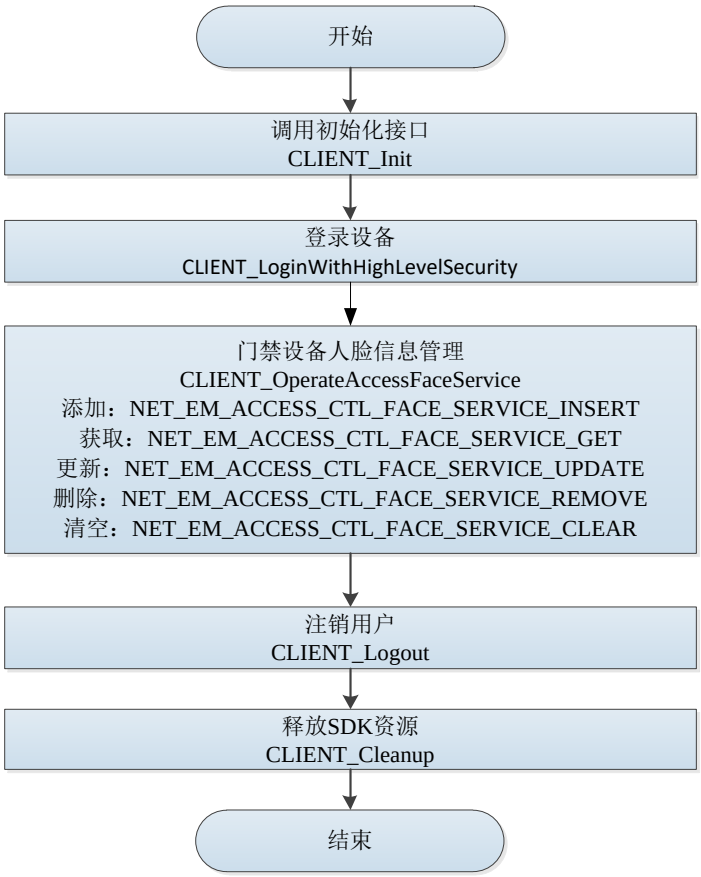
2.4.7.3.2 接口总览

表2-58 人脸信息接口说明

接口	说明
CLIENT_OperateAccessFaceService	门禁人脸信息管理接口

2.4.7.3.3 流程说明

图2-44 人脸信息管理业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_OperateAccessFaceService 函数来对人脸信息进行添加、获取、更新、删除操作。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.7.3.4 示例代码

```
//新增
NET_IN_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
    stuFaceInsertIn.nFaceInfoNum = 1;
    NET_ACCESS_FACE_INFO stuFaceInfo;
    memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
    memcpy(&stuFaceInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    stuFaceInfo.nFacePhoto = 1;

    FILE *fPic = fopen(m_szFilePath, "rb");
```

```
    if (fPic == NULL)
    {
        FaceUIState(TRUE);
        MessageBox(ConvertString("Open picture fail"), ConvertString("Prompt"));
        return;
    }

    fseek(fPic, 0, SEEK_END);
    int nLength = ftell(fPic);
    if (nLength <= 0)
    {
        goto FREE_RETURN;
    }
    rewind(fPic);

    stuFaceInfo.nInFacePhotoLen[0]= nLength;
    stuFaceInfo.nOutFacePhotoLen[0] = nLength;
    stuFaceInfo.pFacePhoto[0] = new char[nLength];
    memset(stuFaceInfo.pFacePhoto[0], 0, nLength);
    int nReadLen = fread(stuFaceInfo.pFacePhoto[0], 1, nLength, fPic);
    fclose(fPic);
    fPic = NULL;
    if (nReadLen <= 0)
    {
        goto FREE_RETURN;
    }
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;

    NET_OUT_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertOut = {sizeof(stuFaceInsertOut)};
    stuFaceInsertOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceInsertOut.pFailCode = &stuFailCodeR;
NET_IN_ACCESS_FACE_SERVICE_INSERT stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
    stuFaceInsertIn.nFaceInfoNum = nNum;
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;
    NET_OUT_ACCESS_FACE_SERVICE_INSERT stuFaceINsertOut = {sizeof(stuFaceINsertOut)};
    stuFaceINsertOut.nMaxRetNum = nNum;
    stuFaceINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_INSERT, &stuFaceInsertIn, &stuFaceINsertOut, SDK_API_WAIT);
```

```
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//获取
NET_IN_ACCESS_FACE_SERVICE_GET stuFaceGetIn = {sizeof(stuFaceGetIn)};
    stuFaceGetIn.nUserNum = 1;
    memcpy(&stuFaceGetIn.szUserID[0], &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));

    NET_OUT_ACCESS_FACE_SERVICE_GET stuFaceGetOut = {sizeof(stuFaceGetOut)};
    stuFaceGetOut.nMaxRetNum = 1;
    NET_ACCESS_FACE_INFO stuFaceInfo;
    memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
    for (int i=0;i<5;i++)
    {
        stuFaceInfo.nInFacePhotoLen[i] = 100*1024;
        stuFaceInfo.pFacePhoto[i] = new char[100*1024];
        memset(stuFaceInfo.pFacePhoto[i],0,100*1024);
    }
    stuFaceGetOut.pFaceInfo = &stuFaceInfo;

    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceGetOut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID, NET_EM_ACCESS_CTL_FACE_SERVICE_GET,
&stuFaceGetIn, &stuFaceGetOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//更新
NET_IN_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
    stuFaceInsertIn.nFaceInfoNum = 1;
    NET_ACCESS_FACE_INFO stuFaceInfo;
    memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
    memcpy(&stuFaceInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    stuFaceInfo.nFacePhoto = 1;
```

```
    FILE *fPic = fopen(m_szFilePath, "rb");
    if (fPic == NULL)
    {
        FaceUIState(TRUE);
        MessageBox(ConvertString("Open picture fail"), ConvertString("Prompt"));
        return;
    }

    fseek(fPic, 0, SEEK_END);
    int nLength = ftell(fPic);
    if (nLength <= 0)
    {
        goto FREE_RETURN;
    }
    rewind(fPic);

    stuFaceInfo.nInFacePhotoLen[0]= nLength;
    stuFaceInfo.nOutFacePhotoLen[0] = nLength;
    stuFaceInfo.pFacePhoto[0] = new char[nLength];
    memset(stuFaceInfo.pFacePhoto[0], 0, nLength);
    int nReadLen = fread(stuFaceInfo.pFacePhoto[0], 1, nLength, fPic);
    fclose(fPic);
    fPic = NULL;
    if (nReadLen <= 0)
    {
        goto FREE_RETURN;
    }
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;

    NET_OUT_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertOut = {sizeof(stuFaceInsertOut)};
    stuFaceInsertOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceInsertOut.pFailCode = &stuFailCodeR;
NET_IN_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateIn = {sizeof(stuFaceUpdateIn)};
    stuFaceUpdateIn.nFaceInfoNum = nNum;
    stuFaceUpdateIn.pFaceInfo = &stuFaceInfo;
    NET_OUT_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateOut = {sizeof(stuFaceUpdateOut)};
    stuFaceUpdateOut.nMaxRetNum = nNum;
    stuFaceUpdateOut.pFailCode = &stuFailCode;
```

```
    BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_UPDATE, &stuFaceUpdateIn, &stuFaceUpdateOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//删除
NET_IN_ACCESS_FACE_SERVICE_REMOVE stuFaceRIn = {sizeof(stuFaceRIn)};
    stuFaceRIn.nUserNum = 1;
    memcpy(&stuFaceRIn.szUserID[0], &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    NET_OUT_ACCESS_FACE_SERVICE_REMOVE stuFaceROut = {sizeof(stuFaceROut)};
    stuFaceROut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceROut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_REMOVE, &stuFaceRIn, &stuFaceROut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//清空
NET_IN_ACCESS_FACE_SERVICE_CLEAR stuFaceClearIn = {sizeof(stuFaceClearIn)};
    NET_OUT_ACCESS_FACE_SERVICE_CLEAR stuFaceClearOut = {sizeof(stuFaceClearOut)};
    BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_CLEAR, &stuFaceClearIn, &stuFaceClearOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
```

2.4.7.4 指纹管理

2.4.7.4.1 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。

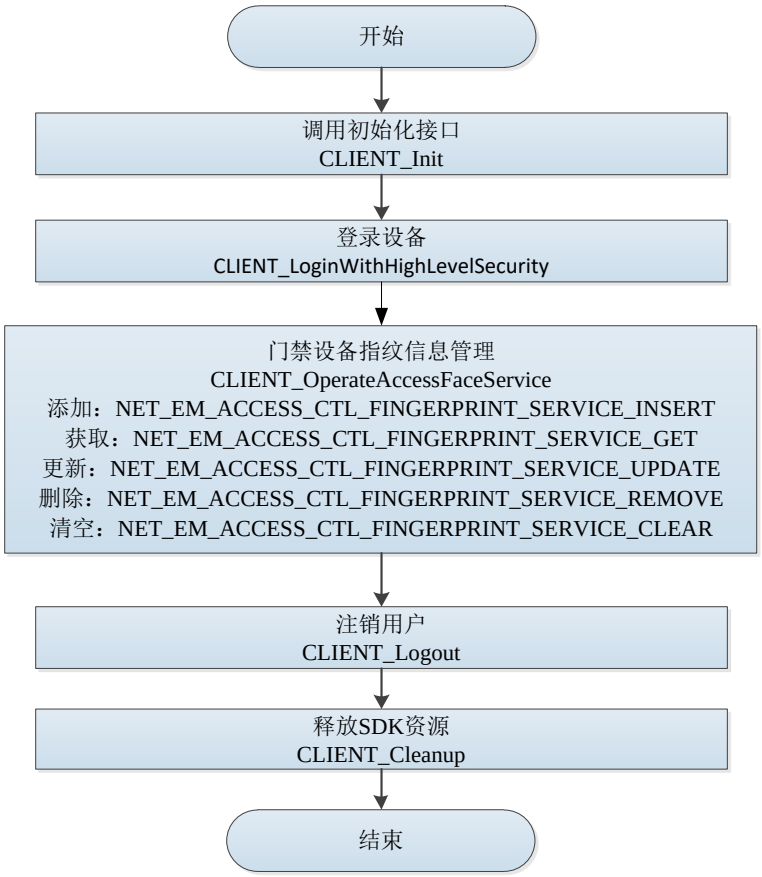
2.4.7.4.2 接口总览

表2-59 指纹信息接口说明

接口	说明
CLIENT_OperateAccessFingerprintService	指纹信息管理接口

2.4.7.4.3 流程说明

图2-45 指纹信息管理业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_OperateAccessFingerprintService 函数来对指纹信息进行添加、获取、更新、删除、清空操作。

步骤4　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤5　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.7.4.4 示例代码

```
//新增
NET_ACCESS_FINGERPRINT_INFO stuFingerPrintInfo;
    memset(&stuFingerPrintInfo,0,sizeof(stuFingerPrintInfo));
    memcpy(&stuFingerPrintInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
```

```
    stuFingerPrintInfo.nPacketLen = m_nFingerprintLen;
    stuFingerPrintInfo.nPacketNum = 1;
    stuFingerPrintInfo.szFingerPrintInfo = new char[m_nFingerprintLen];
    memset(stuFingerPrintInfo.szFingerPrintInfo, 0, m_nFingerprintLen);
    memcpy(stuFingerPrintInfo.szFingerPrintInfo, m_byFingerprintData, m_nFingerprintLen);
    if (bDuress)
    {
        stuFingerPrintInfo.nDuressIndex = 1;
    }
    //stuFingerPrintInfo.nDuressIndex
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT stuFingerPrintInsertIn =
{sizeof(stuFingerPrintInsertIn)};
    stuFingerPrintInsertIn.nFpNum = nNum;
    stuFingerPrintInsertIn.pFingerPrintInfo = &stuFingerPrintInfo;
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT stuFingerPrintINsertOut =
{sizeof(stuFingerPrintINsertOut)};
    stuFingerPrintINsertOut.nMaxRetNum = nNum;
    stuFingerPrintINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_INSERT, &stuFingerPrintInsertIn,
&stuFingerPrintINsertOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
    if (stuFingerPrintInfo.szFingerPrintInfo != NULL)
    {
        delete[] stuFingerPrintInfo.szFingerPrintInfo;
        stuFingerPrintInfo.szFingerPrintInfo = NULL;
    }

//获取
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetIn = {sizeof(stuFingerPrintGetIn)};
    memcpy(&stuFingerPrintGetIn.szUserID[0], &m_stuUserInfo.szUserID,
sizeof(m_stuUserInfo.szUserID));
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_GET, &stuFingerprintGetIn, &stuFingerprintGetOut,
SDK_API_WAIT);
    if (bRet)
```

```
    {
        return TRUE;
    }
    return FALSE;

//更新
NET_ACCESS_FINGERPRINT_INFO stuFingerPrintInfo;
    memset(&stuFingerPrintInfo,0,sizeof(stuFingerPrintInfo));
    memcpy(&stuFingerPrintInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    if (m_nFingerprintLen != m_stuFingerprint.nSinglePacketLength)
    {
        MessageBox(ConvertString("FingerprintLen error"), ConvertString("Prompt"));
        FingerPrintUIState(TRUE);
        return;
    }
    stuFingerPrintInfo.nPacketLen = m_nFingerprintLen;
    stuFingerPrintInfo.nPacketNum = m_stuFingerprint.nRetFingerPrintCount;
    stuFingerPrintInfo.szFingerPrintInfo = new char[m_nFingerprintLen *
m_stuFingerprint.nRetFingerPrintCount];
    memset(stuFingerPrintInfo.szFingerPrintInfo, 0, m_nFingerprintLen *
m_stuFingerprint.nRetFingerPrintCount);
    memcpy(stuFingerPrintInfo.szFingerPrintInfo, m_stuFingerprint.pbyFingerData,
m_stuFingerprint.nRetFingerDataLength);
    memcpy(stuFingerPrintInfo.szFingerPrintInfo + m_nFingerprintIndex * m_nFingerprintLen,
m_byFingerprintData, m_nFingerprintLen);
    if (bDuress)
    {
        stuFingerPrintInfo.nDuressIndex = m_nFingerprintIndex + 1;
    }
    //stuFingerPrintInfo.nDuressIndex
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE stuFingerprintUpdateIn =
{sizeof(stuFingerprintUpdateIn)};
    stuFingerprintUpdateIn.nFpNum = nNum;
    stuFingerprintUpdateIn.pFingerPrintInfo = &stuFingerPrintInfo;
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE stuFingerprintUpdateOut =
{sizeof(stuFingerprintUpdateOut)};
    stuFingerprintUpdateOut.nMaxRetNum = nNum;
    stuFingerprintUpdateOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_UPDATE, &stuFingerprintUpdateIn,
```

```
&stuFingerprintUpdateOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
    if (stuFingerPrintInfo.szFingerPrintInfo != NULL)
    {
        delete[] stuFingerPrintInfo.szFingerPrintInfo;
        stuFingerPrintInfo.szFingerPrintInfo = NULL;
    }

//删除
NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE stuFingerPrintRemoveIn =
{sizeof(stuFingerPrintRemoveIn)};
    stuFingerPrintRemoveIn.nUserNum = 1;
    memcpy(&stuFingerPrintRemoveIn.szUserID[0], &m_stuUserInfo.szUserID,
sizeof(m_stuUserInfo.szUserID));
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE stuFingerPrintRemoveOut =
{sizeof(stuFingerPrintRemoveOut)};
    stuFingerPrintRemoveOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    stuFingerPrintRemoveOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_REMOVE, &stuFingerprintRemoveIn,
&stuFingerprintRemoveOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//清空
NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR stuFingerprintClearIn = {sizeof(stuFingerprintClearIn)};
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR stuFingerprintClearOut =
{sizeof(stuFingerprintClearOut)};
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_CLEAR, &stuFingerprintClearIn, &stuFingerprintClearOut,
SDK_API_WAIT);
    if (bRet)
    {
```

```
        return TRUE;
    }
    return FALSE;
```

2.4.8 门配置

请参见“2.3.8 门配置”。

2.4.9 门时间管理

2.4.9.1 时段配置

请参见“2.3.9.1 时段配置”。

2.4.9.2 常开常闭时间段配置

请参见“2.3.9.2 常开常闭时间段配置”。

2.4.9.3 假日组

2.4.9.3.1 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。

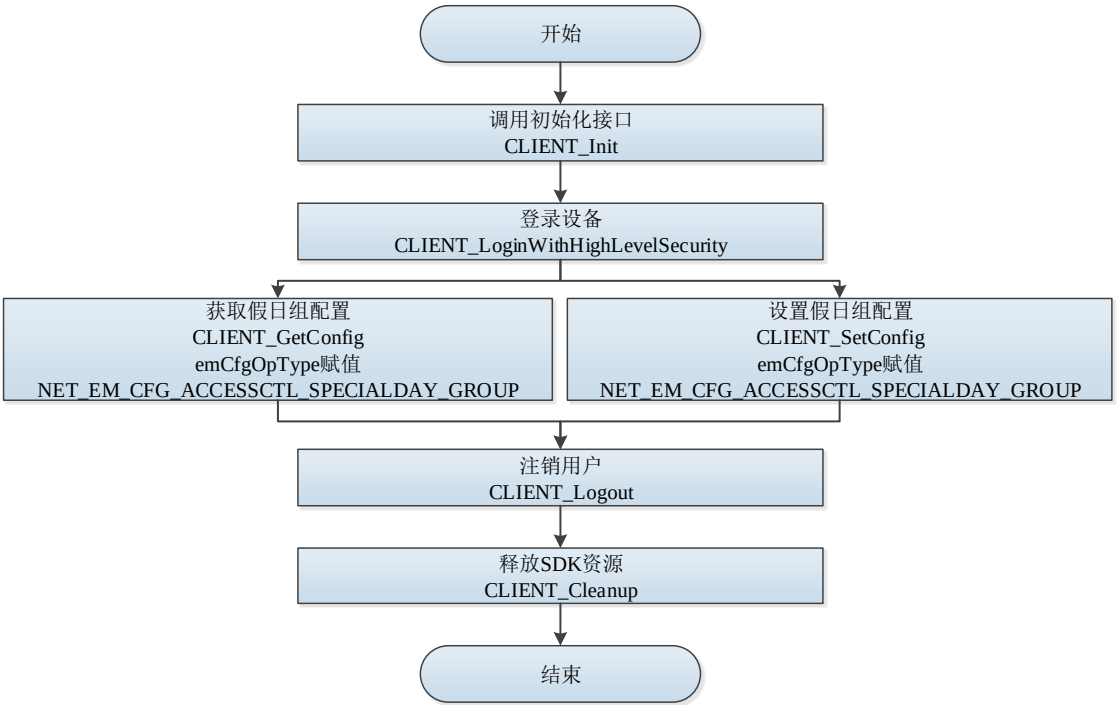
2.4.9.3.2 接口总览

表2-60 假日组接口说明

接口	说明
CLIENT_GetConfig	查询配置信息。
CLIENT_SetConfig	设置配置信息。

2.4.9.3.3 流程说明

图2-46 假日组业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetConfig 函数来查询门禁假日组配置信息。

表2-61 获取假日组信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	获取假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO 结构体尺寸

步骤4　调用 CLIENT_SetConfig 函数来设置门禁假日组配置信息。

表2-62 设置假日组信息 mCfgOpType 参数赋值说明

emCfgOpType	描述	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	设置假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO 结构体尺寸

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.9.3.4 示例代码

```
// 获取
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuIn = {sizeof(stuIn)};
    BOOL bret = CLIENT_GetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP,
```

```
nId,&stuSpecialdayGroup, sizeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }

//设置
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuSpecialdayGroup;
stuSpecialdayGroup.bGroupEnable = m_chkGroupEnable.GetCheck() ? TRUE : FALSE;
BOOL bret = CLIENT_SetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP, nId,
&stuSpecialdayGroup, sizeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));//通道是假日组序号
0-127
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
}
```

2.4.9.4 假日计划

2.4.9.4.1 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。

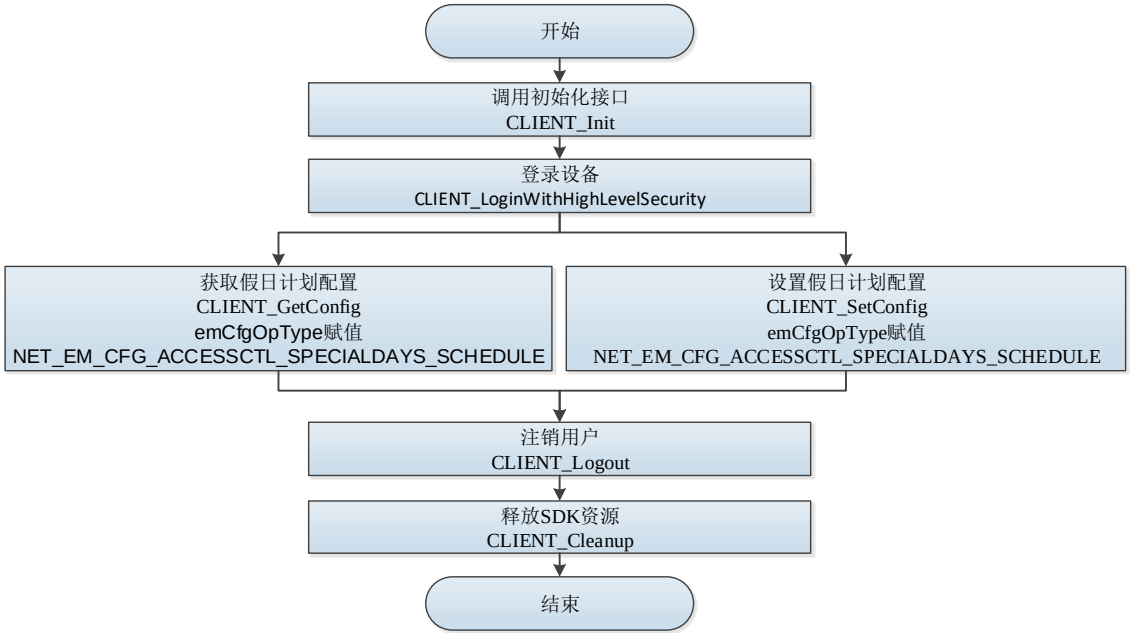
2.4.9.4.2 接口总览

表2-63 假日计划接口说明

接口	说明
CLIENT_GetConfig	查询配置信息。
CLIENT_SetConfig	设置配置信息。

2.4.9.4.3 流程说明

图2-47 假日计划业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。

步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。

步骤3　调用 CLIENT_GetConfig 函数来查询门禁假日计划配置信息。

表2-64 获取假日计划信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	获取假日计划信息	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO 结构体尺寸	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO 结构体尺寸

步骤4　调用 CLIENT_SetConfig 函数来设置门禁假日计划配置信息。

表2-65 设置假日计划信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	设置假日计划信息	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO 结构体尺寸	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO 结构体尺寸

步骤5　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。

步骤6　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.9.4.4 示例代码

```
// 获取
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuSpecialdaySchedule =
{sizeof(stuSpecialdaySchedule)};
    BOOL bret = CLIENT_GetConfig((LLONG)m_lLoginID,
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE, nId,& stuSpecialdaySchedule,
sizeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
```

```
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }

//设置
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuSpecialdaySchedule;
stuSpecialdaySchedule.bSchdule = m_chkSpeciadayEnable.GetCheck() ? TRUE : FALSE;
stuSpecialdaySchedule.nGroupNo = 1;
BOOL bret = CLIENT_SetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE,
nId, & stuSpecialdaySchedule, sizeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
}
```

2.4.10 门高级配置

请参见“2.3.10 门高级配置”。

2.4.11 记录查询

2.4.11.1 开门记录

请参见“2.3.11.1 开门记录”

2.4.11.2 设备日志

请参见“2.3.11.2 设备日志”。

2.4.11.3 报警记录

2.4.11.3.1 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。

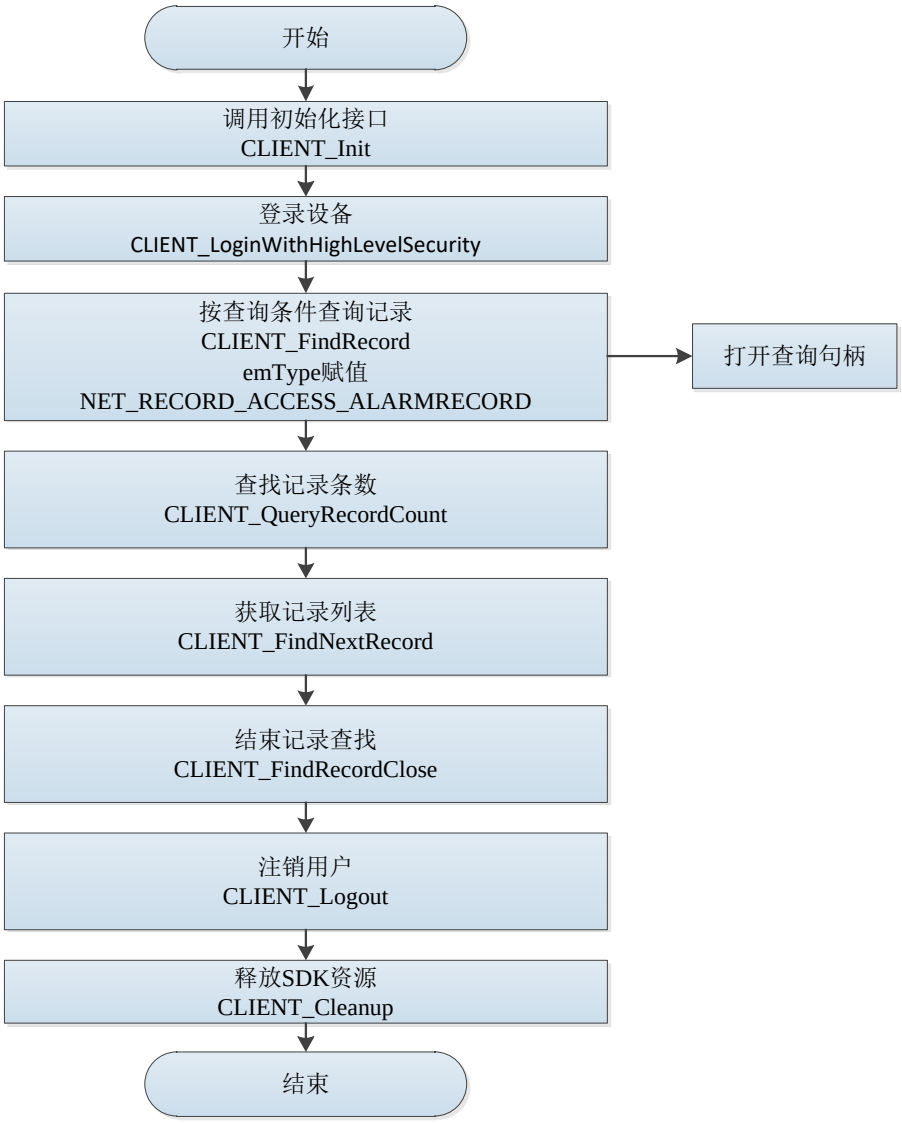
2.4.11.3.2 接口总览

表2-66 记录查询接口说明

接口	说明
CLIENT_QueryRecordCount	查找记录条数
CLIENT_FindRecord	按查询条件查询记录
CLIENT_FindNextRecord	查找记录:nFilecount:需要查询的条数，返回值为媒体文件条数。返回值小于 nFilecount 则相应时间段内的文件查询完毕
CLIENT_FindRecordClose	结束记录查询

2.4.11.3.3 流程说明

图2-48 记录查询业务流程



流程说明

步骤1　调用 CLIENT_Init 函数，完成 SDK 初始化流程。
步骤2　调用 CLIENT_LoginWithHighLevelSecurity 函数登录设备。
步骤3　先调用 CLIENT_FindRecord 函数获得查询句柄。
pInParam 中 emType 赋值 NET_RECORD_ACCESS_ALARMRECORD。
步骤4　再调用 CLIENT_QueryRecordCount 函数查找记录条数。
步骤5　再调用 CLIENT_FindNextRecord 函数获取记录列表。
步骤6　查询完毕可以调用 CLIENT_FindRecordClose 关闭查询句柄。
步骤7　业务执行完成之后，调用 CLIENT_Logout 函数登出设备。
步骤8　SDK 功能使用完后，调用 CLIENT_Cleanup 函数释放 SDK 资源。

2.4.11.3.4 示例代码

```
//开始查询
NET_IN_FIND_RECORD_PARAM stuIn = {sizeof(stuIn)};
    NET_OUT_FIND_RECORD_PARAM stuOut = {sizeof(stuOut)};
    stuIn.emType = NET_RECORD_ACCESS_ALARMRECORD;
    if (CLIENT_FindRecord(m_lLoginID, &stuIn, &stuOut, SDK_API_WAIT))
    {
        m_AlarmFindId = stuOut.lFindeHandle;
        return TRUE;
    }
    else
    {
        return FALSE;
}

//查询记录数
NET_IN_QUEYT_RECORD_COUNT_PARAM stuIn = {sizeof(stuIn)};
    stuIn.lFindeHandle = m_AlarmFindId;
    NET_OUT_QUEYT_RECORD_COUNT_PARAM stuOut = {sizeof(stuOut)};
    if (CLIENT_QueryRecordCount(&stuIn, &stuOut, SDK_API_WAIT))
    {
        nCount = stuOut.nRecordCount;
        return TRUE;
    }
    else
    {
        return FALSE;
    }
//查找记录
```

```
    NET_RECORD_ACCESS_ALARMRECORD_INFO pstuAlarm;
NET_IN_FIND_NEXT_RECORD_PARAM stuIn = {sizeof(stuIn)};
    stuIn.lFindeHandle = m_AlarmFindId;
    stuIn.nFileCount = nMaxNum;

    NET_OUT_FIND_NEXT_RECORD_PARAM stuOut = {sizeof(stuOut)};
    stuOut.nMaxRecordNum = nMaxNum;
    stuOut.pRecordList = (void*)pstuAlarm;

    if (CLIENT_FindNextRecord(&stuIn, &stuOut, SDK_API_WAIT) >= 0)
    {
        if (stuOut.nRetRecordNum > 0)
        {
            nRecordNum = stuOut.nRetRecordNum;
            return TRUE;
        }
    }
    return FALSE;
//结束记录查找
BOOL bret = CLIENT_FindRecordClose(m_AlarmFindId);
    if (bret)
    {
        m_AlarmFindId = NULL;
        return TRUE;
    }
    else
    {
        return FALSE;
    }
```

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 CLIENT_Init

表3-1 SDK 初始化说明书

选项	说明	
描述	对整个 SDK 进行初始化	
函数	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);	
参数	[in]cbDisConnect	断线回调函数
	[in]dwUser	断线回调函数的用户参数
返回值	成功返回 TRUE，败返回 FALSE	
说明	● 调用网络 SDK 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	

3.1.1.2 SDK 清理 CLIENT_Cleanup

表3-2 SDK 清理说明

选项	说明
描述	清理 SDK
函数	void CLIENT_Cleanup()
参数	无
返回值	无
说明	SDK 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 CLIENT_SetAutoReconnect

表3-3 设置断线重连回调函数说明

选项	说明	
描述	设置自动重连回调函数	
函数	void CLIENT_SetAutoReconnect(fHaveReConnect cbAutoConnect, LDWORD 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

选项	说明
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 CLIENT_SetNetworkParam

表3-4 设备网络参数说明

选项	说明	
描述	设置网络环境相关参数	
函数	void CLIENT_SetNetworkParam(NET_PARAM *pNetParam);	
参数	[in]pNetParam	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 CLIENT_StartSearchDevicesEx

表3-5 搜索设备 CLIENT_StartSearchDevicesEx

选项	说明	
描述	搜索设备信息	
函数	LLONG CLIENT_StartSearchDevicesEx (NET_IN_STARTSERACH_DEVICE* pInBuf, NET_OUT_STARTSERACH_DEVICE* pOutBuf);	
参数	[in] pInBuf	异步搜索设备入参，具体参考 NET_IN_STARTSERACH_DEVICE 结构体定义
	[out] pOutBuf	异步搜索设备出参，具体参考 NET_OUT_STARTSERACH_DEVICE 结构体定义
返回值	搜索句柄	
说明	不支持多线程调用	

3.1.2.2 设备初始化 CLIENT_InitDevAccount

表3-6 设备初始化说明

选项	说明
描述	初始化设备
函数	BOOL CLIENT_InitDevAccount(const NET_IN_INIT_DEVICE_ACCOUNT *pInitAccountIn, NET_OUT_INIT_DEVICE_ACCOUNT *pInitAccountOut, DWORD dwWaitTime, char *szLocalIp);

选项	说明	
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	● 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	成功返回 TRUE，败返回 FALSE	
说明	无	

3.1.2.3 获取密码重置信息 CLIENT_GetDescriptionForResetPwd

表3-7 获取密码重置信息说明

选项	说明	
描述	获取密码重置信息	
函数	BOOL CLIENT_GetDescriptionForResetPwd(const NET_IN_DESCRIPTION_FOR_RESET_PWD *pDescriptionIn, NET_OUT_DESCRIPTION_FOR_RESET_PWD *pDescriptionOut, DWORD dwWaitTime, char *szLocalIp);	
参数	[in]pDescriptionIn	输入参数，对应 NET_IN_DESCRIPTION_FOR_RESET_PWD 结构体
	[out]pDescriptionOut	输出参数，对应 NET_OUT_DESCRIPTION_FOR_RESET_PWD 结构体
	[in]dwWaitTime	超时时间
	[in]szLocalIp	● 在单网卡的情况下，最后一个参数可不填 ● 在多网卡的情况下，最后一个参数填主机 IP
返回值	成功返回 TRUE，败返回 FALSE	
说明	无	

3.1.2.4 检验安全码是否有效 CLIENT_CheckAuthCode

表3-8 检验安全码是否有效说明

选项	说明	
描述	检验安全码否有效	
函数	BOOL CLIENT_CheckAuthCode(const NET_IN_CHECK_AUTHCODE *pCheckAuthCodeIn, NET_OUT_CHECK_AUTHCODE *pCheckAuthCodeOut, DWORD dwWaitTime, char *szLocalIp);	
参数	[in]pCheckAuthCodeIn	输入参数，对应 NET_IN_CHECK_AUTHCODE 结构体

选项	说明	
	[out]pCheckAuthCodeOut	输出参数，对应 NET_OUT_CHECK_AUTHCODE 结构体
	[in]dwWaitTime	超时时间
	[in]szLocalIp	● 在单网卡的情况下，最后一个参数可不填 ● 在多网卡的情况下，最后一个参数填主机 IP
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.2.5 重置密码 CLIENT_ResetPwd

表3-9 重置密码说明

选项	说明	
描述	重置密码	
函数	BOOL CLIENT_ResetPwd(const NET_IN_RESET_PWD *pResetPwdIn, NET_OUT_RESET_PWD *pResetPwdOut, DWORD dwWaitTime, char *szLocalIp);	
参数	[in]pResetPwdIn	输入参数，对应 NET_IN_RESET_PWD 结构体
	[out]pResetPwdOut	输出参数，对应 NET_OUT_RESET_PWD 结构体
	[in]dwWaitTime	超时时间
	[in]szLocalIp	● 在单网卡的情况下，最后一个参数可不填 ● 在多网卡的情况下，最后一个参数填主机 IP
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.2.6 获取密码规则 CLIENT_GetPwdSpecification

表3-10 获取密码规则说明

选项	说明	
描述	获取密码规则	
函数	BOOL CLIENT_GetPwdSpecification(const NET_IN_PWD_SPECI *pPwdSpeciIn, NET_OUT_PWD_SPECI *pPwdSpeciOut, DWORD dwWaitTime, char *szLocalIp);	
参数	[in]pPwdSpeciIn	输入参数，对应 NET_IN_PWD_SPECI 结构体
	[out]pPwdSpeciOut	输出参数，对应 NET_OUT_PWD_SPECI 结构体
	[in]dwWaitTime	超时时间
	[in]szLocalIp	● 在单网卡的情况下，最后一个参数可以不填 ● 在多网卡的情况下，最后一个参数填主机 IP
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.2.7 停止搜索设备 CLIENT_StopSearchDevices

表3-11 停止搜索设备说明

选项	说明	
描述	停止搜索设备信息	
函数	BOOL CLIENT_StopSearchDevices (LLONG lSearchHandle);	
参数	[in] lSearchHandle	输入参数，搜索句柄
返回值	成功返回 TRUE，失败返回 FALSE	
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 CLIENT_LoginWithHighLevelSecurity

表3-12 用户登录设备说明

选项	说明	
描述	用户登录设备	
函数	LLONG CLIENT_LoginWithHighLevelSecurity (NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam);	
参数	[in] pstInParam	登录参数包括 IP、端口、用户名、密码、登录模式等
	[out] pstOutParam	设备登录输出参数包括设备信息、错误码
返回值	● 成功返回非 0 ● 失败返回 0	
说明	高安全级别登录接口。 **说明** CLINET_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 CLIENT_LoginWithHighLevelSecurity 登录设备。	

参数 pstOutParam 中 error 的错误码及含义说明，请参见表 3-13。

表3-13 参数 error 的错误码及含义

error 的错误码	对应的含义
1	密码不正确
2	用户名不存在
3	登录超时
4	账号已登录
5	账号已被锁定
6	账号被列为黑名单
7	资源不足，设备系统忙
8	子连接失败
9	主连接失败

error 的错误码	对应的含义
10	超过最大用户连接数
11	缺少 avnetsdk 或 avnetsdk 的依赖库
12	设备未插入 U 盘或 U 盘信息错误
13	客户端 IP 地址没有登录权限

3.1.3.2 用户登出设备 CLIENT_Logout

表3-14 用户登出设备说明

选项	说明	
描述	用户登出设备	
函数	BOOL CLIENT_Logout(LLONG lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.4 实时预览

3.1.4.1 打开预览 CLIENT_RealPlayEx

表3-15 打开预览说明

选项	说明	
描述	打开实时预览	
函数	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);	
参数	[in]lLoginID	CLIENT_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	● 成功返回非 0 ● 失败返回 0	
说明	在 Windows 环境下： ● hWnd 为有效值时，在对应窗口显示画面 ● hWnd 为 NULL 时，取流方式是通过设置回调函数来获取视频数据，再交由用户处理	

预览类型及含义，请参见表 3-16。

表3-16 预览类型说明

预览类型	含义
DH_RType_Realplay	实时预览

预览类型	含义
DH_RType_Multiplay	多画面预览
DH_RType_Realplay_0	实时预览-主码流，等同于 DH_RType_Realplay
DH_RType_Realplay_1	实时预览-从码流 1
DH_RType_Realplay_2	实时预览-从码流 2
DH_RType_Realplay_3	实时预览-从码流 3
DH_RType_Multiplay_1	多画面预览－1 画面
DH_RType_Multiplay_4	多画面预览－4 画面
DH_RType_Multiplay_8	多画面预览－8 画面
DH_RType_Multiplay_9	多画面预览－9 画面
DH_RType_Multiplay_16	多画面预览－16 画面
DH_RType_Multiplay_6	多画面预览－6 画面
DH_RType_Multiplay_12	多画面预览－12 画面
DH_RType_Multiplay_25	多画面预览－25 画面
DH_RType_Multiplay_36	多画面预览－36 画面

3.1.4.2 关闭预览 CLIENT_StopRealPlayEx

表3-17 关闭预览说明

选项	说明	
描述	关闭实时预览	
函数	BOOL CLIENT_StopRealPlayEx(LLONG lRealHandle);	
参数	[in]lRealHandle	CLIENT_RealPlayEx 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.4.3 保存预览数据 CLIENT_SaveRealData

表3-18 保存预览数据说明

选项	说明	
描述	保存实时预览数据为文件	
函数	BOOL CLIENT_SaveRealData(LLONG lRealHandle, const char *pchFileName);	
参数	[in]lRealHandle	CLIENT_RealPlayEx 的返回值
	[in]pchFileName	需要保存的文件路径
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.4.4 停止保存预览数据 CLIENT_StopSaveRealData

表3-19 停止保存预览数据说明

选项	说明	
描述	停止保存实时预览数据为文件	
函数	BOOL CLIENT_StopSaveRealData(LLONG lRealHandle);	
参数	[in]lRealHandle	CLIENT_RealPlayEx 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.4.5 设置预览数据回调 CLIENT_SetRealDataCallBackEx2

表3-20 设置预览数据回调说明

选项	说明	
描述	设置实时预览数据回调	
函数	BOOL CLIENT_SetRealDataCallBackEx2(LLONG lRealHandle, fRealDataCallBackEx2 cbRealData, LDWORD dwUser, DWORD dwFlag);	
参数	[in]lRealHandle	CLIENT_RealPlayEx 的返回值
	[in]cbRealData	预览数据流回调函数
	[in]dwUser	预览数据流回调函数的参数
	[in]dwFlag	回调中预览数据的类型，EM_REALDATA_FLAG 类型，支持或运算
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-21 dwFlag 类型及含义

dwFlag	含义
REALDATA_FLAG_RAW_DATA	原始数据标志
REALDATA_FLAG_DATA_WITH_FRAME_INFO	带有帧信息的数据标志
REALDATA_FLAG_YUV_DATA	YUV 数据标志
REALDATA_FLAG_PCM_AUDIO_DATA	PCM 音频数据标志

3.1.5 设备控制

3.1.5.1 设备控制 CLIENT_ControlDeviceEx

表3-22 设备控制说明

选项	说明
描述	设备控制

选项	说明	
函数	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, Void *pInBuf, Void *pOutBuf = NULL, int nWaitTime = 1000);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]emType	控制类型
	[in]pInBuf	输入参数，因 emType 不同而不同
	[out]pOutBuf	输出参数，默认为 NULL，对于有些 emType 有相应输出结构体
	[in]waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

emType、pInBuf 和 pOutBuf 的对照关系，请参见表 3-23。

表3-23 emType、pInBuf 和 pOutBuf 对照关系

emType	描述	pInBuf	pOutBuf
DH_CTRL_ARMED_EX	布撤防	CTRL_ARM_DISARM_PARAM	NULL
DH_CTRL_SET_BYPASS	设置旁路功能	NET_CTRL_SET_BYPASS	NULL
DH_CTRL_ACCESS_OPEN	门禁控制-开门	NET_CTRL_ACCESS_OPEN	NULL
DH_CTRL_ACCESS_CLOSE	门禁控制-关门	NET_CTRL_ACCESS_CLOSE	NULL

3.1.6 报警侦听

3.1.6.1 设置报警回调函数 CLIENT_SetDVRMessCallBack

表3-24 设置报警回调函数说明

选项	说明	
描述	设置报警回调函数	
函数	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);	
参数	[in]cbMessage	消息回调函数 ● 可回调设备的状态，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	● 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 fMessCallBack 必须先调用报警消息订阅接口 CLIENT_StartListenEx 才生效	

3.1.6.2 订阅报警 CLIENT_StartListenEx

表3-25 订阅报警说明

选项	说明	
描述	订阅报警	
函数	BOOL CLIENT_StartListenEx(LLONG lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	订阅设备消息，得到的消息从 CLIENT_SetDVRMessCallBack 的设置值回调出来	

3.1.6.3 停止订阅报警 CLIENT_StopListen

表3-26 停止订阅报警说明

选项	说明	
描述	停止订阅报警	
函数	BOOL CLIENT_StopListen(LLONG lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.7 智能事件订阅

3.1.7.1 智能事件订阅 CLIENT_RealLoadPictureEx

表3-27 智能事件订阅 CLIENT_RealLoadPictureEx

选项	说明	
描述	订阅智能事件接口	
函数	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
参数	[in] lLoginID	登录句柄
	[in] nChannelID	通道号
	[in] dwAlarmType	报警类型

选项	说明	
	[in] bNeedPicFile	是否订阅图片文件，1-订阅图片文件，在回调函数中会返回智能图片信息，0-不订阅图片文件，在回调函数中不会返回智能图片信息（在不需要图片信息时，可减少网络流量）
	[in] cbAnalyzerData	智能数据分析回调函数
	[in] dwUser	用户参数
	[in] Reserved	保留参数
返回值	成功返回 LLONG 类型的订阅句柄，失败返回 0	
说明	接口返回失败，请用 CLIENT_GetLastError 获取错误码	

本手册使用到的智能事件请参见下表。

表3-28 智能事件类型说明

预览类型	含义
EVENT_IVS_CITIZEN_PICTURE_COMPARE	人证比对事件
EVENT_IVS_ACCESS_CTL	门禁事件

3.1.7.2 智能事件的取消订阅 CLIENT_StopLoadPic

表3-29 智能事件的取消订阅 CLIENT_StopLoadPic

选项	说明	
描述	停止智能事件订阅	
函数	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);	
参数	[in] lAnalyzerHandle	事件订阅句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.8 获取设备状态

3.1.8.1 获取设备状态 CLIENT_QueryDevState

表3-30 获取设备状态说明

选项	说明	
描述	直接获取远程设备连接状态	
函数	BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 返回值
	[in]nType	查询信息类型

选项	说明	
	[out]pBuf	用于接收查询返回的数据缓存。根据查询类型的不同，返回数据的数据结构也不同，详细请参见表 3-31
	[in]nBufLen	缓存长度，单位：字节
	[out]pRetLen	实际返回的数据长度，单位：字节
	[in]waittime	查询状态等待时间，默认 1000ms，可根据需要设置
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-31 查询信息类型和结构体对应关系

查询项	nType	pBuf
查询报警通道状态	DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE	NET_CLIENT_ALARM_CHANNELS_STATE
查询电源与电池信息	DH_DEVSTATE_POWER_STATE	DH_POWER_STATUS

3.1.9 语音对讲

3.1.9.1 获取设备支持对讲类型 CLIENT_GetDevProtocolType

表3-32 获取设备支持对讲类型说明

选项	说明	
描述	获取设备支持对讲类型	
函数	BOOL CLIENT_GetDevProtocolType(LLONG lLoginID, EM_DEV_PROTOCOL_TYPE *pemProtocolType);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[out]pemProtocolType	设备支持的协议类型，对应 EM_DEV_PROTOCOL_TYPE 结构体
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.9.2 设置设备语音对讲工作模式 CLIENT_SetDeviceMode

表3-33 设置设备语音对讲工作模式说明

选项	说明	
描述	设置设备语音对讲工作模式	
函数	BOOL CLIENT_SetDeviceMode(LLONG lLoginID, EM_USEDEV_MODE emType, void *pValue);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in]emType	枚举值
	[in]pValue	与枚举值对应的结构体数据指针，请参见表 3-34

选项	说明
返回值	成功返回 TRUE，失败返回 FALSE
说明	无

emType 和 pValue 对照关系，请参见表 3-34。

表3-34 emType 和 pValue 对照关系

emType	描述	pValue
DH_TALK_ENCODE_TYPE	指定某种格式进行对讲	DHDEV_TALKDECODE_INFO
DH_TALK_CLIENT_MODE	设置语音对讲客户端方式	无
DH_TALK_SPEAK_PARAM	设置语音对讲喊话参数	NET_SPEAK_PARAM
DH_TALK_MODE3	设置三代设备的语音对讲参数	NET_TALK_EX

3.1.9.3 开启对讲 CLIENT_StartTalkEx

表3-35 开启对讲说明

选项	说明	
描述	打开语音对讲	
函数	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);	
参数	[in]lLoginID	CLIENT_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	● 成功返回非 0 ● 失败返回 0	
说明	无	

3.1.9.4 关闭对讲 CLIENT_StopTalkEx

表3-36 关闭对讲说明

选项	说明	
描述	关闭语音对讲	
函数	BOOL CLIENT_StopTalkEx(LLONG lTalkHandle);	
参数	[in]lTalkHandle	CLIENT_StartTalkEx 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.1.9.5 开启录音 CLIENT_RecordStartEx

表3-37 开启录音说明

选项	说明	
描述	开启本地录音	
函数	BOOL CLIENT_RecordStartEx(LLONG lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	此接口只在 Windows 下有效	

3.1.9.6 关闭录音 CLIENT_RecordStopEx

表3-38 关闭录音说明

选项	说明	
描述	关闭本地录音	
函数	BOOL CLIENT_RecordStopEx(LLONG lLoginID);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	此接口只在 Windows 下有效	

3.1.9.7 发送语音 CLIENT_TalkSendData

表3-39 发送语音说明

选项	说明	
描述	发送音频数据给设备	
函数	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);	
参数	[in]lTalkHandle	CLIENT_StartTalkEx 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	

3.1.9.8 解码语音 CLIENT_AudioDecEx

表3-40 解码语音说明

选项	说明
描述	解码音频数据

选项	说明	
函数	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);	
参数	[in]lTalkHandle	CLIENT_StartTalkEx 的返回值
	[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	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.2 报警主机

3.3 门禁控制器/指纹一体机（一代）

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 CLIENT_ControlDeviceEx”。

关于门磁状态接口的详细介绍，请参见“3.3.3.4 查询设备状态 CLIENT_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 查询系统能力信息 CLIENT_QueryNewSystemInfo

表3-41 查询系统能力信息说明

选项	说明	
描述	查询系统能力信息，按字符串格式	
函数	BOOL CLIENT_QueryNewSystemInfo (LLONG lLoginID, char *szCommand, int nChannelID, char *szOutBuffer, DWORD dwOutBufferSize, int *error, int nWaitTime = 1000);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值

选项	说明	
	[in] szCommand	命令参数，详细介绍请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”
	[in] nChannelID	通道号
	[out] szOutBuffer	接收的协议缓冲区
	[in] dwOutBufferSize	接收的总字节数(单位字节)
	[out] error	错误号
	[in]waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	成功返回 TRUE，失败返回 FALSE	
说明	获取到的信息按照字符串格式，各个字符串包含的信息由 CLIENT_ParseData 解析	

表3-42 参数 error 的错误码及含义

error 的错误码	对应的含义
0	成功
1	失败
2	数据不合法
3	暂时无法设置
4	没有权限

3.3.3.2 解析查询到的配置信息 CLIENT_ParseData

表3-43 解析查询到的配置信息说明

选项	说明	
描述	解析查询到的配置信息	
函数	BOOL CLIENT_ParseData (char *szCommand, char *szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, int *pReserved);	
参数	[in] szCommand	命令参数，详细介绍请参见表 3-44
	[in] szInBuffer	输入缓冲，字符配置缓冲
	[out] lpOutBuffer	输出缓冲，结构体类型请参见表 3-44
	[in] dwOutBufferSize	输出缓冲的大小
	[in] pReserved	保留参数
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-44 szCommand、查询类型和对应结构体的对照关系

szCommand	查询类型	对应结构体
CFG_CAP_CMD_ACCESSCONTROLMANAGER	门禁能力	CFG_CAP_ACCESSCONTROL
CFG_CMD_NETWORK	IP 配置	CFG_NETWORK_INFO
CFG_CMD_DVRIP	主动注册配置	CFG_DVRIP_INFO
CFG_CMD_NTP	NTP 校时	CFG_NTP_INFO

szCommand	查询类型	对应结构体
CFG_CMD_ACCESS_EVENT	门禁事件配置（门配置信息、常开常闭时段配置、分手段、首卡开门配置）	CFG_ACCESS_EVENT_INFO
CFG_CMD_ACCESSTIMESCHEDULE	门禁刷卡时段（时段配置）	CFG_ACCESS_TIMESCHEDULE_INFO
CFG_CMD_OPEN_DOOR_GROUP	多人组合开门配置	CFG_OPEN_DOOR_GROUP_INFO
CFG_CMD_ACCESS_GENERAL	门禁基本配置（多门互锁）	CFG_ACCESS_GENERAL_INFO
CFG_CMD_OPEN_DOOR_ROUTE	开门路线集合，或称防反潜路线配置	CFG_OPEN_DOOR_ROUTE_INFO

3.3.3.3 获取设备能力 CLIENT_GetDevCaps

表3-45 获取设备能力说明

选项	说明	
描述	获取设备能力	
函数	BOOL CLIENT_GetDevCaps (LLONG lLoginID, int nType, void* pInBuf, void* pOutBuf, int nWaitTime);	
参数	[in] lLoginID	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	成功返回 TRUE，失败返回 FALSE	
说明	无	

nType、pInBuf 和 pOutBuf 的对照关系，请参见表 3-46。

表3-46 nType、pInBuf 和 pOutBuf 对照关系

nType	描述	pInBuf	pOutBuf
NET_FACEINFO_CAPS	获得人脸门禁控制器能力集	NET_IN_GET_FACEINFO_CAPS	NET_OUT_GET_FACEINFO_CAPS

3.3.3.4 查询设备状态 CLIENT_QueryDevState

表3-47 查询设备状态说明

选项	说明
描述	获取前端设备的当前工作状态

选项	说明	
函数	BOOL CLIENT_QueryDevState (LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);	
参数	[in] lLoginID	登录句柄
	[in] nType	设备类型 控制参数,根据 type 不同而不同
	[out] pBuf	输出参数，用于接收查询返回的数据的缓存。根据查询类型的不同，返回数据的数据结构也不同
	[in] nBufLen	缓存长度，单位字节
	[in] waittime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

nType、查询类型和结构体的对应关系，请参见表 3-48。

表3-48 nType、查询类型和结构体的对应关系

nType	描述	pBuf
DH_DEVSTATE_SOFTWARE	查询设备软件版本信息	DHDEV_VERSION_INFO
DH_DEVSTATE_NETINTERFACE	查询网络接口信息	DHDEV_NETINTERFACE_INFO
DH_DEVSTATE_DEV_RECORDSET	查询设备记录集信息	NET_CTRL_RECORDSET_PARAM
DH_DEVSTATE_DOOR_STATE	查询门禁状态（门磁）	NET_DOOR_STATUS_INFO

3.3.4 网络配置

3.3.4.1 IP 配置

3.3.4.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig

表3-49 查询配置信息说明

选项	说明
描述	获取配置，按照字符串格式

选项	说明	
函数	BOOL CLIENT_GetNewDevConfig (LLONG lLoginID, char *szCommand, int nChannelID, char *szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime =500);	
参数	[in] lLoginID	登录句柄
	[in] szCommand	命令参数，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”
	[in] nChannelID	通道号
	[out]szOutBuffer	输出缓冲
	[in] dwOutBufferSize	输出缓冲大小
	[out] error	错误码
	[in] waittime	等待超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	获取配置，按照字符串格式，各个字符串包含的信息由 CLIENT_ParseData 解析	

表3-50 参数 error 的错误码及含义说明

error 的错误码	对应的含义
0	成功
1	失败
2	数据不合法
3	暂时无法设置
4	没有权限

3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig

表3-51 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	BOOL CLIENT_SetNewDevConfig (LLONG lLoginID, char *szCommand, int nChannelID, char *szInBuffer, DWORD dwInBufferSize, int *error, int * restart int waittime =500);	
参数	[in] lLoginID	登录句柄
	[in] szCommand	命令参数信息，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”

选项	说明	
	[in] nChannelID	通道号
	[in] szInBuffer	输出缓冲
	[in] dwInBufferSize	输出缓冲大小
	[out] error	错误码
	[out] restart	配置设置后是否需要重启设备，1 表示需要重启，0 表示不需要重启
	[in] waittime	等待超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	设置配置，按照字符串格式，各个字符串包含的信息由 CLIENT_PacketData 组包	

表3-52 参数 error 的错误码及含义说明

error 的错误码	对应的含义
0	成功
1	失败
2	数据不合法
3	暂时无法设置
4	没有权限

3.3.4.1.4 打包字符串格式 CLIENT_PacketData

表3-53 打包字符串格式说明

选项	说明	
描述	将需要设置的配置信息，打包成字符串格式	
函数	BOOL CLIENT_PacketData (char *szCommand, LPVOID lpInBuffer, DWORD dwInBufferSize, char *szOutBuffer, DWORD dwOutBufferSize);	
参数	[out] szCommand	命令参数，具体请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”
	[in] lpInBuffer	输入缓冲，结构体类型请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”
	[in] dwInBufferSize	输出缓冲的大小
	[out] szOutBuffer	输出缓冲
	[in] dwOutBufferSize	输出缓冲大小
返回值	成功返回 TRUE，失败返回 FALSE	
说明	此接口配合 CLIENT_SetNewDevConfig 使用，使用 CLIENT_PacketData 后，将打包的信息通过 CLIENT_SetNewDevConfig 设置到设备上	

3.3.4.2 主动注册配置

3.3.4.2.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.4.2.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.4.2.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.4.2.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.5 时间设置

3.3.5.1 时间设置

表3-54 时间设置说明

选项	说明	
描述	设置设备当前时间	
函数	BOOL CLIENT_SetupDeviceTime (LLONG lLoginID, LPNET_TIME pDeviceTime,);	
参数	[in] lLoginID	登录句柄
	[in] pDeviceTime	设置的设备时间指针
返回值	成功返回 TRUE，失败返回 FALSE	
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	

3.3.5.2 NTP 校时、时区配置

3.3.5.2.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.5.2.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.5.2.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.5.2.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.5.3 夏令时配置

3.3.5.3.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.5.3.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.5.3.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.5.3.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.6 维护配置

3.3.6.1 修改登录密码

3.3.6.1.1 操作设备用户 CLIENT_OperateUserInfoNew

表3-55 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	BOOL CLIENT_OperateUserInfoNew (LLONG lLoginID, int nOperateType, void *opParam, void *subParam, void* pReserved, int nWaitTime = 1000);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值

选项	说明	
	[in] nOperateType	操作类型，具体请参见表 3-56
	[in] opParam	设置用户信息的输入缓冲，具体请参见表 3-56
	[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 3-56
	[in] pReserved	保留
	[in]waittime	超时时间，默认 1000ms，可根据需要自行设置
返回值	成功返回 TRUE，失败返回 FALSE	
说明	为实现要求的功能，设置更改设备的用户信息	

表3-56 nOperateType、opParam 和 subParam 之间的对应关系

nOperateType	opParam	subParam
6	USER_INFO_NEW	USER_INFO_NEW

3.3.6.2 重启设备

3.3.6.2.1 设备控制 CLIENT_ControlDevice

表3-57 设备控制说明

选项	说明	
描述	设备控制	
函数	BOOL CLIENT_ControlDevice(LLONG lLoginID, CtrlType type, void *param, int nWaitTime = 1000);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[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	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-58 Type 和 param 的对照关系

Type	描述	Param
DH_CTRL_REBOOT	重启设备	无
DH_CTRL_RECORDSET_INSERT	添加记录，获得记录集编号	NET_CTRL_RECORDSET_INSERT_PARAM
DH_CTRL_RECORDSET_INSERTEX	添加指纹记录，获得记录集编号	NET_CTRL_RECORDSET_INSERT_PARAM
DH_CTRL_RECORDSET_REMOVE	根据记录集编号删除某记录	NET_CTRL_RECORDSET_PARAM
DH_CTRL_RECORDSET_CLEAR	清除所有记录集信息	NET_CTRL_RECORDSET_PARAM

Type	描述	Param
DH_CTRL_RECORDSET_UPDATE	更新某记录集编号的记录	NET_CTRL_RECORDSET_PARAM
DH_CTRL_RECORDSET_UPDATEEX	更新指纹记录集编号的记录	NET_CTRL_RECORDSET_PARAM
DH_CTRL_ACCESS_OPEN	门禁控制-开门	CTRL_ARM_DISARM_PARAM
DH_CTRL_RESTOREDEFAULT	恢复设备的默认设置	DH_RESTORE_COMMON

3.3.6.3 恢复出厂设置

3.3.6.3.1 恢复出厂设置 CLIENT_ControlDevice、CLIENT_ResetSystem

- 关于CLIENT_ControlDevice的详细介绍，请参见“3.3.6.2.1 设备控制CLIENT_ControlDevice”。
- 关于 CLIENT_ResetSystem 的详细介绍，请参见表 3-59。

表3-59 恢复出厂设置说明

选项	说明	
描述	恢复出厂设置	
函数	BOOL CLIENT_ResetSystem (LLONG lLoginID, const NET_IN_RESET_SYSTEM* pstInParam, NET_OUT_RESET_SYSTEM* pstOutParam, int nWaitTime);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值
	[in] pstInParam	恢复出厂设置入参
	[in] pstOutParam	恢复出厂设置出厂
	[in] nWaitTime	超时时间
返回值	成功返回 Ture，失败返回 False	

3.3.6.4 设备升级

3.3.6.4.1 开始升级设备程序 CLIENT_StartUpgradeEx

表3-60 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	LLONG CLIENT_StartUpgradeEx (LLONG lLoginID, EM_UPGRADE_TYPE emType char *pchFileName, fUpgradeCallBack cbUpgrade, LDWORD dwUser);	
参数	[in]lLoginID	CLIENT_LoginWithHighLevelSecurity 的返回值

选项	说明	
	[in] emType	枚举值，详细介绍请参见表 3-61
	[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	成功返回升级句柄 ID，失败返回 0	
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 CLIENT_SendUpgrade 接口，数据才能发送。	

表3-61 枚举值

emType	意义
DH_UPGRADE_BIOS_TYPE	BIOS 升级
DH_UPGRADE_WEB_TYPE	WEB 升级
DH_UPGRADE_BOOT_YPE	BOOT 升级
DH_UPGRADE_CHARACTER_TYPE	汉字库
DH_UPGRADE_LOGO_TYPE	LOGO
DH_UPGRADE_EXE_TYPE	EXE,例如播放器等
DH_UPGRADE_DEVCONSTINFO_TYPE	设备固有信息设置(如：硬件 ID、MAC、序列号)
DH_UPGRADE_PERIPHERAL_TYPE	外设接入从片（如车载芯片）
DH_UPGRADE_GEOINFO_TYPE	地理信息定位芯片
DH_UPGRADE_MENU	菜单（设备操作界面的图片）
DH_UPGRADE_ROUTE	线路文件（如公交线路）
DH_UPGRADE_ROUTE_STATE_AUTO	报站音频（与线路配套的报站音频）
DH_UPGRADE_SCREEN	调度屏（如公交操作屏）

3.3.6.4.2 开始发送升级文件 CLIENT_SendUpgrade

表3-62 开始发送升级文件说明

选项	说明	
描述	开始发送升级文件	
函数	BOOL CLIENT_SendUpgrade (LLONG lUpgradeID);	
参数	[in] lUpgradeID	升级句柄 ID
返回值	成功返回 TRUE，失败返回 FALSE	
说明	发送升级程序数据	

3.3.6.4.3 停止升级 CLIENT_StopUpgrade

表3-63 停止升级说明

选项	说明	
描述	开始发送升级文件	
函数	BOOL CLIENT_StopUpgrade (LLONG lUpgradeID);	
参数	[in] lUpgradeID	升级句柄 ID
返回值	成功返回 TRUE，失败返回 FALSE	

选项	说明
说明	不要在回调函数里调用此接口

3.3.6.5 自动维护

3.3.6.5.1 查询配置信息 CLIENT_GetDevConfig

表3-64 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 CLIENT_GetDevConfig (LLONG lLoginID, DWORD dwCommand, LONG lChannel, LPVOID lpOutBuffer, DWORD dwOutBufferSize, LPDWORD lpBytesReturned, int waittime =500);	
参数	[in] lLoginID	设备登录句柄
	[in] dwCommand	设备配置命令，具体请参见表 3-65。不同 dwCommand，lpOutBuffer 对应的结构体将会不同，详细介绍请参见表 3-65
	[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
	[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] lpBytesReturned	实际收到数据的长度
	[in] waittime	等待超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-65 dwCommand、lpOutBuffer 对应关系

dwCommand	查询类型	对应结构体 lpOutBuffer
DH_DEV_DST_CFG	夏令时配置	CFG_NTP_INFO
DH_DEV_AUTOMTCFG	自动维护配置	DHDEV_AUTOMT_CFG

3.3.6.5.2 设置配置信息 CLIENT_SetDevConfig

表3-66 设置配置信息说明

选项	说明
描述	设置设备的配置信息

选项	说明	
函数	BOOL CLIENT_SetDevConfig (LLONG lLoginID, DWORD dwCommand, LONG lChannel, LPVOID lpInBuffer, DWORD dwInBufferSize, int waittime =500);	
参数	[in] lLoginID	设备登录句柄
	[in] dwCommand	设备配置命令，具体请参见表 3-65。不同 dwCommand，lpInBuffer 对应的结构体将会不同，具体请参见表 3-65
	[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	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.3.7 人员管理

3.3.7.1 人员信息字段集合

请参见“3.3.6.2.1 设备控制 CLIENT_ControlDevice”和“3.3.3.4 查询设备状态 CLIENT_QueryDevState”。

3.3.8 门配置

3.3.8.1 门配置信息

3.3.8.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.8.1.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.8.1.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.8.1.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.9 门时间配置

3.3.9.1 时段配置

3.3.9.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.9.1.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.9.1.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.9.1.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.9.2 常开常闭时段配置

3.3.9.2.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.9.2.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.9.2.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.9.2.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见 3.3.4.1.4 打包字符串格式 CLIENT_PacketData

3.3.9.3 假日配置

请参见“3.3.6.2.1 设备控制 CLIENT_ControlDevice”和“3.3.3.4 查询设备状态 CLIENT_QueryDevState”。

3.3.10 门高级配置

3.3.10.1 分时段、首卡开门

3.3.10.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.10.1.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.10.1.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.10.1.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.10.2 多人组合开门

3.3.10.2.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.10.2.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.10.2.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.10.2.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.10.3 多门互锁

3.3.10.3.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.3.10.3.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.10.3.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.10.3.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.10.4 防反潜

3.3.10.4.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”

3.3.10.4.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.3.10.4.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.3.10.4.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.3.10.5 开门密码

关于 CLIENT_ControlDevice 的详细介绍，请参见“3.3.6.2.1 设备控制 CLIENT_ControlDevice”。

3.3.10.6 设备日志

3.3.10.6.1 查询设备日志条数 CLIENT_QueryDevLogCount

表3-67 查询设备日志条数说明

选项	说明	
描述	查询设备日志条数	
函数	int CLIENT_QueryDevLogCount (LLONG lLoginID, NET_IN_GETCOUNT_LOG_PARAM* pInParam, NET_OUT_GETCOUNT_LOG_PARAM* pOutParam, int waittime);	
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	

3.3.10.6.2 开始查询日志 CLIENT_StartQueryLog

表3-68 开始查询日志说明

选项	说明	
描述	开始查询设备日志	
函数	LLONG CLIENT_StartQueryLog (LLONG lLoginID, const NET_IN_START_QUERYLOG* pInParam, 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	成功返回查询句柄，失败返回 0	
说明	无	

3.3.10.6.3 获取日志 CLIENT_QueryNextLog

表3-69 获取日志说明

选项	说明
描述	获取日志

选项	说明	
函数	BOOL CLIENT_QueryNextLog (LLONG lLogID, NET_IN_QUERYNEXTLOG* pInParam, NET_OUT_QUERYNEXTLOG* pOutParam, int nWaitTime);	
参数	[in] lLogID	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	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.3.10.6.4 结束查询日志 CLIENT_StopQueryLog

表3-70 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 CLIENT_StopQueryLog (LLONG lLogID,);	
参数	[in] lLogID	查询日志句柄
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.3.11 记录查询

3.3.11.1 开门记录

3.3.11.1.1 查找记录条数 CLIENT_QueryRecordCount

表3-71 查询记录条数说明

选项	说明	
描述	查找记录条数	
函数	BOOL CLIENT_QueryRecordCount (NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, int waittime);	
参数	[in] pInParam	查询记录条数入参，pInParam ->lFindeHandle 为 CLIENT_FindRecord 的 pOutParam-> lFindeHandle

选项	说明	
	[out] pOutParam	查询记录条数出参，pOutParam ->nRecordCount 为记录个数
	[in] waittime	查询超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	调用本接口之前应先调用 CLIENT_FindRecord 以打开查询句柄	

3.3.11.1.2 按查询条件查询记录 CLIENT_FindRecord

表3-72 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	BOOL CLIENT_FindRecord (LLONG lLoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam , int waittime=3000);	
参数	[in]lLoginID	设备登录句柄
	[in] pInParam	查询日志的参数，参数赋值说明请参见表 3-73
	[out] pOutParam	查询日志的出参，返回查询句柄
	[in] waittime	等待超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	可以先调用本接口获得查询句柄，再调用 CLIENT_FindNextRecord 函数获取记录列表，查询完毕可以调用 CLIENT_FindRecordClose 关闭查询句柄	

表3-73 查询开门记录入参数说明

pInParam 结构体	赋值	说明
emType	NET_RECORD_ACCESSCTLCARDREC_EX	用于开门记录查询

3.3.11.1.3 查找记录 CLIENT_FindNextRecord

表3-74 查找记录说明

选项	说明	
描述	查找记录:nFilecount:需要查询的条数,返回值为媒体文件条数返回值小于 nFilecount 则相应时间段内的文件查询完毕	
函数	int CLIENT_FindNextRecord (NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, int waittime);	
参数	[in] pInParam	查询记录入参，pInParam ->lFindeHandle 为 CLIENT_FindRecord 的 pOutParam-> lFindeHandle
	[out] pOutParam	查询记录出参返回记录信息
	[in] waittime	等待超时时间
返回值	1：成功取回一条记录，0：记录已取完，-1：参数出错	

选项	说明
说明	无

表3-75 查询开门记录出参说明

pOutParam 结构体	赋值	说明
pRecordList	NET_RECORDSET_ACCESS_CTL_CARDREC	用于开门记录查询

3.3.11.1.4 结束记录查询 CLIENT_FindRecordClose

表3-76 结束记录查询说明

选项	说明	
描述	结束记录查询	
函数	BOOL CLIENT_FindRecordClose (LLONG lFindHandle,);	
参数	[in] lFindHandle	CLIENT_FindRecord 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	调用 CLIENT_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	

3.4 门禁控制器/人脸一体机（二代）

3.4.1 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 CLIENT_ControlDeviceEx”。

关于门磁状态接口的详细介绍，请参见“3.3.3.4 查询设备状态 CLIENT_QueryDevState”。

3.4.2 报警事件

请参见“3.1.6 报警侦听”。

3.4.3 设备信息查看

3.4.3.1 获取设备能力 CLIENT_GetDevCaps

表3-77 获取设备能力说明

选项	说明
描述	获取设备能力

选项	说明	
函数	BOOL CLIENT_GetDevCaps (LLONG lLoginID, int nType, void* pInBuf, void* pOutBuf, int nWaitTime);	
参数	[in] lLoginID	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	成功返回 TRUE，失败返回 FALSE	
说明	无	

nType、pInBuf 和 pOutBuf 的对照关系，请参见表 3-46。

表3-78 nType、pInBuf 和 pOutBuf 对照关系

nType	描述	pInBuf	pOutBuf
NET_ACCESSCONTROL_CAPS	获取门禁能力	NET_IN_AC_CAPS	NET_OUT_AC_CAPS

3.4.3.2 查询设备状态 CLIENT_QueryDevState

关于 CLIENT_QueryDevState 详细介绍，请参见“3.3.3.4 查询设备状态 CLIENT_QueryDevState”。

3.4.4 网络配置

请参见“3.3.4 网络配置”。

3.4.5 时间设置

请参见“3.3.5 时间设置”。

3.4.6 维护配置

请参见“3.3.6 维护配置”。

3.4.7 人员管理

3.4.7.1 用户管理

3.4.7.1.1 门禁用户信息管理接口 CLIENT_OperateAccessUserService

表3-79 门禁用户信息管理接口说明

选项	说明	
描述	门禁人员信息管理接口	
函数	BOOL CLIENT_OperateAccessUserService (LLONG lLoginID, NET_EM_ACCESS_CTL_USER_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
参数	[in] lLoginID	登录句柄
	[in] emtype	用户信息操作类型
	[in] pInBuf	用户信息管理（入参）
	[out] pOutBuf	用户信息管理（出参）
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

emtype、pInBuf 和 pOutBuf 的对照关系，请参见表 3-80。

表3-80 nType、pInBuf 和 pOutBuf 对照关系

emtype	描述	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_USER_SERVICE_INSERT	添加用户信息	NET_IN_ACCESS_USER_SERVICE_INSERT	NET_OUT_ACCESS_USER_SERVICE_INSERT
NET_EM_ACCESS_CTL_USER_SERVICE_REMOVE	删除用户信息	NET_IN_ACCESS_USER_SERVICE_REMOVE	NET_OUT_ACCESS_USER_SERVICE_REMOVE
NET_EM_ACCESS_CTL_USER_SERVICE_CLEAR	清空所有用户信息	NET_IN_ACCESS_USER_SERVICE_CLEAR	NET_OUT_ACCESS_USER_SERVICE_CLEAR

3.4.7.1.2 开始查询人员信息接口 CLIENT_StartFindUserInfo

表3-81 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	LLONG CLIENT_StartFindUserInfo (LLONG lLoginID, NET_IN_USERINFO_START_FIND* pstIn, NET_OUT_USERINFO_START_FIND* pstOut, int nWaitTime);	
参数	[in] lLoginID	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	成功返回查询句柄，失败返回 0	
说明	无	

3.4.7.1.3 获取人员信息接口 CLIENT_DoFindUserInfo

表3-82 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 CLIENT_DoFindUserInfo (LLONG lFindHandle, NET_IN_USERINFO_DO_FIND* pstIn, NET_OUT_USERINFO_DO_FIND* pstOut, int nWaitTime);	
参数	[in] lFindHandle	CLIENT_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	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.4.7.1.4 停止查询人员信息接口 CLIENT_StopFindUserInfo

表3-83 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 CLIENT_StopFindUserInfo (LLONG lFindHandle);	
参数	[in] lFindHandle	CLIENT_StartFindUserInfo 返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.4.7.2 卡片管理

3.4.7.2.1 门禁卡片信息管理接口 CLIENT_OperateAccessCardService

表3-84 门禁卡片信息管理接口说明

选项	说明	
描述	门禁卡片信息管理接口	
函数	BOOL CLIENT_OperateAccessCardService (LLONG lLoginID, NET_EM_ACCESS_CTL_CARD_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
参数	[in] lLoginID	登录句柄
	[in] emtype	卡片信息操作类型
	[in] pInBuf	卡片信息管理（入参）
	[out] pOutBuf	卡片信息管理（出参）
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

emtype、pInBuf 和 pOutBuf 的对照关系，请参见表 3-46。

表3-85 nType、pInBuf 和 pOutBuf 对照关系

emtype	描述	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_CARD_SERVICE_INSERT	添加卡片信息	NET_IN_ACCESS_CARD_SERVICE_INSERT	NET_OUT_ACCESS_CARD_SERVICE_INSERT
NET_EM_ACCESS_CTL_CARD_SERVICE_REMOVE	删除卡片信息	NET_IN_ACCESS_CARD_SERVICE_REMOVE	NET_OUT_ACCESS_CARD_SERVICE_REMOVE
NET_EM_ACCESS_CTL_CARD_SERVICE_CLEAR	清空所有卡片信息	NET_IN_ACCESS_CARD_SERVICE_CLEAR	NET_OUT_ACCESS_CARD_SERVICE_CLEAR

3.4.7.2.2 开始查询卡片信息接口 CLIENT_StartFindCardInfo

表3-86 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	LLONG CLIENT_StartFindCardInfo (LLONG lLoginID, NET_IN_CARDINFO_START_FIND* pstIn, NET_OUT_CARDINFO_START_FIND* pstOut, int nWaitTime);	
参数	[in] lLoginID	登录句柄

选项	说明	
	[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	
说明	无	

3.4.7.2.3 获取卡片信息接口 CLIENT_DoFindCardInfo

表3-87 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 CLIENT_DoFindCardInfo (LLONG lFindHandle, NET_IN_CARDINFO_DO_FIND* pstIn, NET_OUT_CARDINFO_DO_FIND* pstOut, int nWaitTime);	
参数	[in] lFindHandle	CLIENT_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	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.4.7.2.4 停止查询卡片信息接口 CLIENT_StopFindUserInfo

表3-88 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 CLIENT_StopFindCardInfo (LLONG lFindHandle);	
参数	[in] lFindHandle	CLIENT_StartFindCardInf 返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

3.4.7.3 人脸管理

3.4.7.3.1 门禁人脸信息管理接口 CLIENT_OperateAccessFaceService

表3-89 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

选项	说明	
函数	BOOL CLIENT_OperateAccessFaceService (LLONG lLoginID, NET_EM_ACCESS_CTL_FACE_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
参数	[in] lLoginID	登录句柄
	[in] emtype	人脸信息操作类型
	[in] pInBuf	人脸信息管理（入参）
	[out] pOutBuf	人脸信息管理（出参）
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

emtype、pInBuf 和 pOutBuf 的对照关系，请参见表 3-90。

表3-90 nType、pInBuf 和 pOutBuf 对照关系

emtype	描述	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_FACE_SERVICE_INSERT	添加人脸信息	NET_IN_ACCESS_FACE_SERVICE_INSERT	NET_OUT_ACCESS_FACE_SERVICE_INSERT
NET_EM_ACCESS_CTL_FACE_SERVICE_GET	获取人脸信息	NET_IN_ACCESS_FACE_SERVICE_GET	NET_OUT_ACCESS_FACE_SERVICE_GET
NET_EM_ACCESS_CTL_FACE_SERVICE_UPDATE	更新人脸信息	NET_IN_ACCESS_FACE_SERVICE_UPDATE	NET_OUT_ACCESS_FACE_SERVICE_UPDATE
NET_EM_ACCESS_CTL_FACE_SERVICE_REMOVE	删除人脸信息	NET_IN_ACCESS_FACE_SERVICE_REMOVE	NET_OUT_ACCESS_FACE_SERVICE_REMOVE
NET_EM_ACCESS_CTL_FACE_SERVICE_CLEAR	清空人脸信息	NET_IN_ACCESS_FACE_SERVICE_CLEAR	NET_OUT_ACCESS_FACE_SERVICE_CLEAR

3.4.7.4 指纹管理

3.4.7.4.1 门禁指纹信息管理接口 CLIENT_OperateAccessFingerprintService

表3-91 门禁指纹信息管理接口说明

选项	说明
描述	门禁指纹信息管理接口

选项	说明	
函数	BOOL CLIENT_OperateAccessFingerprintService (LLONG lLoginID, NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
参数	[in] lLoginID	登录句柄
	[in] emtype	指纹信息操作类型
	[in] pInBuf	指纹信息管理（入参）
	[out] pOutBuf	指纹信息管理（出参）
	[in] nWaitTime	超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

emtype、pInBuf 和 pOutBuf 的对照关系，请参见表 3-92。

表3-92 nType、pInBuf 和 pOutBuf 对照关系

emtype	描述	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_INSERT	添加指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT	NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_GET	获取指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_GET	NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_UPDATE	更新指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE	NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_REMOVE	删除指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE	NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_CLEAR	清空指纹信息	NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR	NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR

3.4.8 门配置

3.4.8.1 门配置信息

3.4.8.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.4.8.1.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息

CLIENT_GetNewDevConfig”。

3.4.8.1.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.4.8.1.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.4.9 门时间配置

3.4.9.1 时段配置

3.4.9.1.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”

3.4.9.1.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.4.9.1.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”。

3.4.9.1.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.4.9.2 常开常闭时段配置

3.4.9.2.1 解析查询到的配置信息 CLIENT_ParseData

关于 CLIENT_ParseData 的详细介绍，请参见“3.3.3.2 解析查询到的配置信息 CLIENT_ParseData”。

3.4.9.2.2 查询配置信息 CLIENT_GetNewDevConfig

关于 CLIENT_GetNewDevConfig 的详细介绍，请参见“3.3.4.1.2 查询配置信息 CLIENT_GetNewDevConfig”。

3.4.9.2.3 设置配置信息 CLIENT_SetNewDevConfig

关于 CLIENT_SetNewDevConfig 的详细介绍，请参见“3.3.4.1.3 设置配置信息 CLIENT_SetNewDevConfig”

3.4.9.2.4 打包字符串格式 CLIENT_PacketData

关于 CLIENT_PacketData 的详细介绍，请参见“3.3.4.1.4 打包字符串格式 CLIENT_PacketData”。

3.4.9.3 假日组

3.4.9.3.1 获取假日组接口 CLIENT_GetConfig

表3-93 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 CLIENT_GetConfig (LLONG lLoginID NET_EM_CFG_OPERATE_TYPE emCfgOpType int nChannelID void* szOutBuffer DWORD dwOutBufferSize int waittime=3000 void * reserve=NULL);	
参数	[in] lLoginID	登录句柄
	[in] emCfgOpType	设置配置信息的类型 假日组配置： NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP
	[in] nChannelID	通道号
	[out] szOutBuffer	获取配置信息的缓存地址
	[in] dwOutBufferSize	缓存地址大小
	[in] waittime	超时时间
	[in] reserve	保留参数
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-94 获取假日组信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	获取假日组信息	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO 结构体尺寸

3.4.9.3.2 设置假日组接口 CLIENT_SetConfig

表3-95 设置假日组接口说明

选项	说明
描述	设置假日组接口

选项	说明	
函数	BOOL CLIENT_SetConfig (LLONG lLoginID NET_EM_CFG_OPERATE_TYPE emCfgOpType int nChannelID void* szInBuffer DWORD dwInBufferSize int waittime=3000 int * restart=NULL void * reserve=NULL);	
参数	[in] lLoginID	登录句柄
	[in] emCfgOpType	设置配置的类型 假日组配置： NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP
	[in] nChannelID	通道号
	[in] szInBuffer	配置的缓存地址
	[in] dwInBufferSize	缓存地址大小
	[in] waittime	超时时间
	[in] restart	是否需要重启
	[in] reserve	保留参数
返回值	成功返回 TRUE，失败返回 FALSE	
说明	无	

表3-96 设置假日组信息 mCfgOpType 参数赋值说明

emCfgOpType	描述	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESS CTL_SPECIALDAY_GR OUP	设置假日组信息	NET_CFG_ACCESSCT L_SPECIALDAY_GRO UP_INFO	NET_CFG_ACCESSCTL_ SPECIALDAY_GROUP_I NFO 结构体尺寸

3.4.9.4 假日计划

关于接口的详细介绍，请参见"3.4.9.3 假日组"。

表3-97 获取假日计划信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESS CTL_SPECIALDAYS_SC HEDULE	获取计划信息	NET_CFG_ACCESSCT L_SPECIALDAYS_SCH EDULE_INFO	NET_CFG_ACCESSCTL_ SPECIALDAYS_SCHEDU LE_INFO 结构体尺寸

表3-98 设置假日计划信息 emCfgOpType 参数赋值说明

emCfgOpType	描述	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESS CTL_SPECIALDAYS_SC HEDULE	设置假日计划信息	NET_CFG_ACCESSCT L_SPECIALDAYS_SCH EDULE_INFO	NET_CFG_ACCESSCTL_ SPECIALDAYS_SCHEDU LE_INFO 结构体尺寸

3.4.10 门高级配置

请参见“3.3.10 门高级配置”。

3.4.11 记录查询

3.4.11.1 开门记录

请参见“3.3.11.1 开门记录”。

3.4.11.2 报警记录

3.4.11.2.1 查找记录条数 CLIENT_QueryRecordCount

表3-99 查询记录条数说明

选项	说明	
描述	查找记录条数	
函数	BOOL CLIENT_QueryRecordCount (NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, int waittime);	
参数	[in] pInParam	查询记录条数入参，pInParam ->lFindeHandle 为 CLIENT_FindRecord 的 pOutParam-> lFindeHandle
	[out] pOutParam	查询记录条数出参，pOutParam ->nRecordCount 为记录个数
	[in] waittime	查询超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	调用本接口之前应先调用 CLIENT_FindRecord 以打开查询句柄	

3.4.11.2.2 按查询条件查询记录 CLIENT_FindRecord

表3-100 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	BOOL CLIENT_FindRecord (LLONG lLoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam , int waittime=3000);	
参数	[in]lLoginID	设备登录句柄
	[in] pInParam	查询日志的参数，参数赋值说明请参见表 3-73
	[out] pOutParam	查询日志的出参，返回查询句柄

选项	说明	
	[in] waittime	等待超时时间
返回值	成功返回 TRUE，失败返回 FALSE	
说明	可以先调用本接口获得查询句柄，再调用 CLIENT_FindNextRecord 函数获取记录列表，查询完毕可以调用 CLIENT_FindRecordClose 关闭查询句柄	

表3-101 查询开门记录入参数说明

pInParam 结构体	赋值	说明
emType	NET_RECORD_ACCESS_ALARMRECORD	用于报警记录查询

3.4.11.2.3 查找记录 CLIENT_FindNextRecord

表3-102 查找记录说明

选项	说明	
描述	查找记录:nFilecount:需要查询的条数，返回值为媒体文件条数返回值小于 nFilecount 则相应时间段内的文件查询完毕	
函数	int CLIENT_FindNextRecord (NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, int waittime);	
参数	[in] pInParam	查询记录入参，pInParam ->lFindeHandle 为 CLIENT_FindRecord 的 pOutParam-> lFindeHandle
	[out] pOutParam	查询记录出参返回记录信息
	[in] waittime	等待超时时间
返回值	1：成功取回一条记录，0：记录已取完，-1：参数出错	
说明	无	

表3-103 查询开门记录出参说明

pOutParam 结构体	赋值	说明
pRecordList	NET_RECORD_ACCESS_ALARMRECORD_INFO	用于报警记录查询

3.4.11.2.4 结束记录查询 CLIENT_FindRecordClose

表3-104 结束记录查询说明

选项	说明	
描述	结束记录查询	
函数	BOOL CLIENT_FindRecordClose (LLONG lFindHandle,);	
参数	[in] lFindHandle	CLIENT_FindRecord 的返回值
返回值	成功返回 TRUE，失败返回 FALSE	
说明	调用 CLIENT_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	

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	typedef void(CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX * pDevNetInfo, void* 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

选项	说明	
描述	搜索设备回调函数	
函数	typedef void(CALLBACK * fSearchDevicesCBEx)(LLONG lSearchHandle, DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);	
参数	[out]lSearchHandle	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 fDisConnect

表4-3 断线回调函数说明

选项	说明
描述	断线回调函数

选项	说明	
函数	typedef void (CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
参数	[out]lLoginID	CLIENT_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 fHaveReConnect

表4-4 断线重连回调函数说明

选项	说明	
描述	断线重连回调函数	
函数	typedef void (CALLBACK *fHaveReConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
参数	[out]lLoginID	CLIENT_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 实时预览数据回调函数 fRealDataCallBackEx2

表4-5 实时预览数据回调函数说明

选项	说明
描述	实时预览数据回调函数

选项	说明	
函数	typedef void (CALLBACK *fRealDataCallBackEx2)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);	
参数	[out]lRealHandle	CLIENT_RealPlayEx 的返回值
	[out]dwDataType	数据类型 ● 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	预览数据块地址
	[out]dwBufSize	预览数据块的长度，单位：字节
	[out]param	回调数据参数结构体，dwDataType 值不同类型不同。 ● dwDataType 为 0 时，param 为空指针 ● dwDataType 为 1 时，param 为 tagVideoFrameParam 结构体指针 ● dwDataType 为 2 时，param 为 tagCBYUVDataParam 结构体指针 ● dwDataType 为 3 时，param 为 tagCBPCMDataParam 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 音频数据回调函数说明

选项	说明	
描述	语音对讲的音频数据回调函数	
函数	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser);	
参数	[out]lTalkHandle	CLIENT_StartTalkEx 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

选项	说明	
	[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
	[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 fMessCallBack

表4-7 报警回调函数说明

选项	说明	
描述	报警回调函数	
函数	BOOL (CALLBACK *fMessCallBack)(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
参数	[out]lCommand	报警类型，具体请参见表 4-8
	[out]lLoginID	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 lCommand 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]dwUser	用户自定义数据
返回值	成功返回 TRUE，失败返回 FALSE	
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	lCommand	pBuf
报警主机	本地报警事件	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	电源故障事件	DH_ALARM_POWERFAULT	ALARM_POWERFAULT_INFO
	防拆事件	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	扩展报警输入通道事件	DH_ALARM_ALARMEXTENDED	ALARM_ALARMEXTENDED_INFO

报警所属业务	报警类型名称	lCommand	pBuf
	紧急事件	DH_URGENCY_ALARM_EX	数据为 16 个字节数组，每个字节表示一个通道状态 ● 1 为有报警 ● 0 为无报警
	蓄电池低电压事件	DH_ALARM_BATTERYLOWPOWER	ALARM_BATTERYLOWPOWER_INFO
	设备邀请平台对讲事件	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	设备布防模式变化事件	DH_ALARM_ARMMODE_CHANGE_EVENT	ALARM_ARMMODE_CHANGE_INFO
	防区旁路状态变化事件	DH_ALARM_BYPASSMODE_CHANGE_EVENT	ALARM_BYPASSMODE_CHANGE_INFO
	报警输入源信号事件	DH_ALARM_INPUT_SOURCE_SIGNAL	ALARM_INPUT_SOURCE_SIGNAL_INFO
	消警事件	DH_ALARM_ALARMCLEAR	ALARM_ALARMCLEAR_INFO
	子系统状态改变事件	DH_ALARM_SUBSYSTEM_STATE_CHANGE	ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	扩展模块掉线事件	DH_ALARM_MODULE_LOST	ALARM_MODULE_LOST_INFO
	PSTN 掉线事件	DH_ALARM_PSTN_BREAK_LINE	ALARM_PSTN_BREAK_LINE_INFO
	模拟量报警事件	DH_ALARM_ANALOG_PULSE	ALARM_ANALOGPULSE_INFO
	报警传输事件	DH_ALARM_PROFILE_ALARM_TRANSMIT	ALARM_PROFILE_ALARM_TRANSMIT_INFO
	无线设备低电量报警事件	DH_ALARM_WIRELESSDEV_LOWPOWER	ALARM_WIRELESSDEV_LOWPOWER_INFO
	防区布撤防状态改变事件	DH_ALARM_DEFENCE_ARMMODE_CHANGE	ALARM_DEFENCE_ARMMODECHANGE_INFO
	子系统布撤防状态改变事件	DH_ALARM_SUBSYSTEM_ARMMODE_CHANGE	ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	探测器异常报警	DH_ALARM_SENSOR_ABNORMAL	ALARM_SENSOR_ABNORMAL_INFO
	病人活动状态报警事件	DH_ALARM_PATIENTDETECTION	ALARM_PATIENTDETECTION_INFO
门禁	门禁事件	DH_ALARM_ACCESS_CTL_EVENT	ALARM_ACCESS_CTL_EVENT_INFO
	门禁未关事件详细信息	DH_ALARM_ACCESS_CTL_NOT_CLOSE	ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	闯入事件详细信息	DH_ALARM_ACCESS_CTL_BREAK_IN	ALARM_ACCESS_CTL_BREAK_IN_INFO
	反复进入事件详细信息	DH_ALARM_ACCESS_CTL_REPEAT_ENTER	ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	恶意开门事件	DH_ALARM_ACCESS_CTL_MALICIOUS	ALARM_ACCESS_CTL_MALICIOUS
	胁迫卡刷卡事件详细信息	DH_ALARM_ACCESS_CTL_DURESS	ALARM_ACCESS_CTL_DURESS_INFO

报警所属业务	报警类型名称	lCommand	pBuf
	多人组合开门事件	DH_ALARM_OPENDOORGROUP	ALARM_OPEN_DOOR_GROUP_INFO
	防拆事件	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	本地报警事件	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	门禁状态事件	DH_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	锁舌报警	DH_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	获取指纹事件	DH_ALARM_FINGER_PRINT	ALARM_CAPTURE_FINGER_PRINT_INFO
可视对讲	直连情况下，呼叫无答应事件	DH_ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO
	手机号码上报事件	DH_ALARM_TELEPHONE_CHECK	ALARM_TELEPHONE_CHECK_INFO
	VTS 状态上报	DH_ALARM_VTSTATE_UPDATE	ALARM_VTSTATE_UPDATE_INFO
	VTO 人脸检测	DH_ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	设备请求对方发起对讲事件	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	设备取消对讲请求事件	DH_ALARM_TALKING_IGNORE_INVITE	ALARM_TALKING_IGNORE_INVITE_INFO
	设备主动挂断对讲事件	DH_ALARM_TALKING_HANGUP	ALARM_TALKING_HANGUP_INFO
	雷达监测超速报警事件	DH_ALARM_RADAR_HIGH_SPEED	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	升级进度回调函数	
函数	void (CALLBACK *fUpgradeCallBackEx)(LLONG lLoginID, LLONG lUpgradechannel, INT64 nTotalSize, INT64 nSendSize, LDWORD dwUser);	
参数	[out]lLoginID	登录接口返回值
	[out] lUpgradechannel	CLIENT_StartUpgradeEx2 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	

选项	说明
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

- HDMI™ HIGH-DEFINITION MULTIMEDIA INTERFACE：本声明适用所有产品。如本产品使用 HDMI 技术，词语 HDMI、HDMI High-Definition Multimedia Interface（高清晰度多媒体接口）、HDMI 商业外观和 HDMI 徽标均为 HDMI Licensing Administrator, Inc.的商标或注册商标。本产品已经获得 HDMI Licensing Administrator, Inc.授权使用 HDMI 技术。
- 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

- 本文档供多个型号产品使用，产品外观和功能请以实物为准。
- 如果不按照本文档中的指导进行操作而造成的任何损失由使用方自己承担。
- 本文档会实时根据相关地区的法律法规更新内容，具体请参见产品的纸质、电子光盘、二维码或官网，如果纸质与电子档内容不一致，请以电子档为准。
- 本公司保留随时修改本文档中任何信息的权利，修改的内容将会在本文档的新版本中加入，恕不另行通知。
- 本文档可能包含技术上不准确的地方、或与产品功能及操作不相符的地方、或印刷错误，以公司最终解释为准。
- 如果获取到的 PDF 文档无法打开，请使用最新版本或最主流的阅读工具。

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。
- 不包含账户名称或账户名称的倒序。
- 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（如 NVR、DVR 和 IP 摄像机等）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. 开启账户锁定

出厂默认开启账户锁定功能，建议您保持开启状态，以保护账户安全。在攻击者多次密码尝试失败后，其对应账户及源 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. MAC 地址绑定

建议您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。

8. 合理分配账户及权限

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。

9. 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。

10. 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。

11. 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。

12. 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。

13. 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。
- 开启设备 IP/MAC 地址过滤功能，限制允许访问设备的主机范围。